多波束测深合理探测方案的设计及效果分析

摘要

多波束测深系统比单波束测深系统测量范围大、精度高，适合大面积海底地形的勘探。在实际探测中，海底地形起伏大，不能简单地根据海区平均水深设计测线间隔。本文基于对覆盖宽度、重叠率等的分析，建立了适当的数学模型，设计了合理的探测方案，对提高探测效率和准确性有一定作用。

对于问题一，本文建立了**二维平面上计算多波束测深的覆盖宽度及条带重叠率的数学模型**。根据问题的变化，我们首先给出了坡面情况下重叠率的计算方法。利用三角形中存在的几何关系推导出了海水深度、覆盖宽度、重叠率的计算公式，并计算相关数值将其填入表 1，发现最浅两条测线出现了漏测现象。

对于问题二，本文在第一问的基础上，在不同测线方向的情况下以深度与坡度角度完善了在三维情况下多波束测深覆盖宽度的数学模型。我们首先提取出了三个主要参数：当前坡度γ、当前深度x_D、当前开角θ，并分析了它们之间存在的关系。推导出了各参数在三维情况下的求解公式，以此公式计算出了题中所列位置多波束测深的覆宽，并将它们填入表 2。

对于问题三，本文制定了**基于贪心思想**的测线设计方案。我们首先分析出需要使得w_i在各点最宽以得到最短路径和最佳航向，以重叠率为控制条件进行迭代计算。经过优化后，我们得到了**平行的测线方向，共 34 条**。同时对不同重叠率下的路线数量和平行方向上的航线密度做了**横向分析**。

对于问题四，本文首先**采用微分思想，利用插值拟合和随机森林模型**对数据进行预处理以及数据特点分析，深入探讨了不同曲面以及开角情况下的测量状态，**基于第三问的研究改进了飞蛾火焰算法**，并给出具体的路径搜索流程，在对**区域进行分块**之后，**以重叠率为10%**，在空间内**迭代搜索**了各区域最佳路径，给出各个分区的数据，经过整合之后，得到了最优化的路线且覆盖率**达99.96%**，测线**总长度为434662m**。

综上所述，本文基于微分思想和贪心思想建立层层递进的计算与智能搜索模型。我们首先建立二维平面上的多波束测深的覆盖宽度及条带重叠率的数学模型，然后扩展至三维空间中，从简单地形测线的设计推进到复杂真实海底测量策略的优化，解决了多波束测深系统最优路线方案的设计问题，并且给出具体的图像和精确数据。对使用多波束测深系统进行实际海底地形测绘有一定参考价值。

关键词：空间几何 贪心思想 微分 随机森林 飞蛾火焰算法 多波束测深

一、 问题重述

1.1 问题背景

一份详细的海底地形图可以保障水下船体航行，同时在预测海洋灾害、探测海底资源等方面具有重要作用。测绘海底地形图有多种方法，其中多波束测深系统是一种用于测量海底地形的、承载多传感器的系统。它比单波束测深系统测量范围大、速度快、精度高，特别适合大面积海底地形的勘探。

进行探测时，测线产生的相邻条带需要有 10%~20%的重叠率。重叠率过高会导致数据冗余、处理测量结果过程繁琐；重叠率过低会导致漏测。由于在实际探测中，海底地形较为复杂，不能简单地根据海区平均水深设计测线间隔。为了避免重叠率过高或过低的情况、提高探测效率和准确性，需要建立适当的数学模型，来设计合理的测线间隔。

1.2 需要解决的问题

1．当测线方向垂直的平面与海底坡面的交线面面角为α时：

(1) 建立计算多波束测深的覆盖宽度及相邻条带之间重叠率的数学模型。

(2) 依据该模型计算表 1 中的各指标值。

2．当待测海域为矩形、测线方向与海底坡面的法向在水平面上投影的夹角为β时：

(1) 建立矩形待测海域的计算多波束测深覆盖宽度的模型。

(2) 依据该计算模型探究表 2 中所列位置多波束测深的覆盖宽度。

3．当矩形海域南北长 2 海里，东西宽 4 海里，深度为 $110m$，海底西深东浅时：

(1) 依据 1、2 中的数据模型，综合考虑探测效率和准确性，设计一组测线，使得测量长度最短、$\eta\epsilon[10\%, 20\%]$，且要求测线完全覆盖待测的矩形海域。

4．现有南北长 5 海里、东西宽 4 海里的某海域单波束测量的测深数据：

(1) 设计多波束测量船的测量布线方案，使得测线形成的条带尽可能覆盖待测海域；η尽可能不超过 20%；测线的总长度最短。

(2) 计算设计出的测线的总长度、漏测海区占总待测海域面积的百分比，以及$\eta > 20\%$部分的总长度。

二、 问题分析

2.1 问题一的分析

题目中给出的重叠率的定义是基于水平状态下的，当海底并不平坦时，坡度会给重叠率的计算带来一定的问题。我们需要将重叠率的计算做出适应非水平海底的修正，以适应海底的复杂地形情况。本题要求我们在海底平面坡度为α时分析，

我们可以根据修正后的定义画出该情景下的几何关系简图。利用这些几何关系，在二维平面上给出给定测线距中心点处的距离时，可以做出海水深度、覆盖宽度、与前一条测线重复率的计算。

2.2 问题二的分析

由第一问所建立的模型可知，当前深度由距初始航线的距离来决定。那么对于问题二的解决，我们需要沿用第一问的模型并对此做出相应扩展。参考问题一的模型，可以得到参数间存在的关系。由于当海底平面为均一平面，测量船的航向确定时，坡度也确定为一定值，因此我们需要从深度与坡度着手来解决关于该矩形待测海域的相关计算，完善问题一中的多波束测深覆盖宽度的数学模型。

2.3 问题三的分析

本题要求在南北长 2 海里、东西宽 4 海里的矩形海域内求解一组长度最短，且覆盖全海域的测线方案。这里可以提取出两个信息：测线方案的设计中覆盖面积为全海域、且长度最短是必然要求。基于贪心算法的思想，为使测线的长度最短，需要使得在各处尽可能宽，由此可得到最短测线距离。在取得最优的覆宽以及测线长度的条件下，以重叠率为优化与控制结果的关键指标进行迭代计算。

2.4 问题四的分析

由于单波束测深方法是一种单点连续的测量方法，它沿航迹测得的数据密集，却在测线间没有数据。为了解决测得的数据是离散的问题，需要采用学习模型进行拟合，将离散转为连续，从而做到对任意一点坡面高度、梯度、深度的拟合。为了确定出测量方案，需要从航向的确定和重叠率来考虑，以局部最优解组成总体的最优解。得出测量路径后，还需要统计与计算该方案的测线总长、漏测海域占比与重叠超过 20%路线总长。

三、 模型假设

1．多波束测深系统进行海底探测得到的数据真实有效。
2．不考虑船体颠簸，只考虑航行路线的规划。
3．海平面水平，不考虑潮汐、洋流、风向对此造成的影响。
4．海底无鱼群或其他物体的遮挡。

四、 符号说明

符号	说明
$L_{重}$	相邻条带重叠部分的长度
x_{Di}	当前深度，i为标号
d'	投影后的条带间距
γ	当前坡度
Δx	测量船距海域中心点处的距离（行进距离）
r_i	测线路径，i为标号
w_i	某点的覆盖宽度
S	船体行进后完成的总覆盖面积
$\nabla\vec{g}(x_i, y_i)$	点(x_i, y_i)处的梯度向量

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 相关概念的说明

当处于测线相互平行且海底地形平坦的水平状态下，由题目所给的情景可知，相邻条带之间的重叠率η被定义为：

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

d为测线间距，W为测线的覆盖宽度，D为深度，$L_{重}$为相邻条带重叠部分的长度。各字母量的几何含义如下图所示：

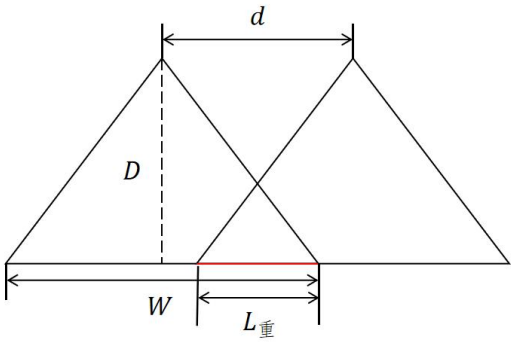

图 1 水平状态下重叠率的定义

当海底地形并非平坦时，在后续问题中可能会出现的更为复杂的情景，例如有复杂坡度（问题二、三）或海底存在略微弧度（第四问）的情况：

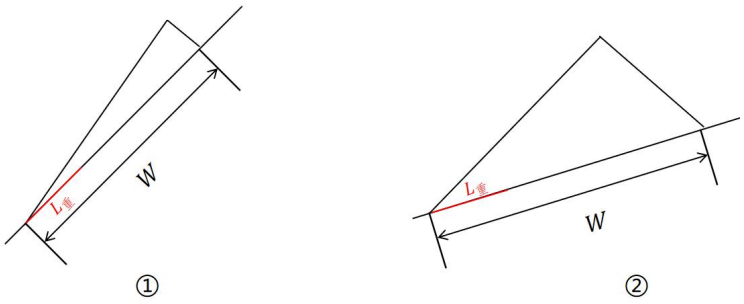

图 2 不同坡度的情况

在这里，我们假设d与$L_{重}$都为定值，此时若再沿用 $1-\frac{d}{W}$的定义，则图 2 中①的重叠率与②的重叠率相比，显然有：

$$\eta_{①} < \eta_{②} \tag{2}$$

而(2)中的结果是反直觉的，显然可以观察出①中的重叠比例是大的，在这样的情况下，计算出的重叠率将失去意义。这种η公式的失效可能会在后续规划中造成错误。为了适应不同的情景，我们将η的定义做出扩展性的解读，并参考相关文献[1]与题目中的场景，给出对重叠率改进后的定义：**即重叠率是重叠部分长度与总覆盖长度的比值。**

$$\eta = 1 - \frac{W - L_{重}}{W} \tag{3}$$

对于改进后的合理性也可以从这个角度来解释：我们作一条与海底斜坡平行的l_2，将测线间距d以最右侧两测线为平行线投影到l_2上，并平移至l_1上。由于此时$d' = W - L_{重}$，因此与水平状态下计算重叠率的(1)式相比，修正重叠率计算公式做出的改动只是将d替换为d'。而由于l_1与l_2平行，这一步修正可以消除坡度对重叠率计算的影响。

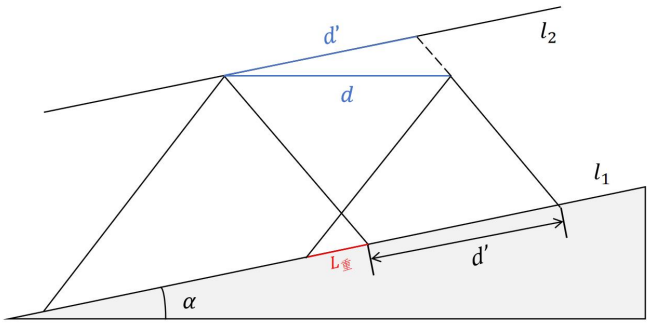

图 3 投影测线间距

利用式(3)所示的重叠率计算方法可以有效避免在不同场景下造成的η失效，**在后续计算中我们以该定义为准。**

5.1.2 模型构建

依据题意，此时需要我们构建当与测线方向垂直的平面和海底坡面的交线构成一条与水平面夹角为 α 的斜线时的计算模型，并依据该模型，在二维平面上给出给定测线距中心点处的距离时，海水深度、覆盖宽度、与前一条测线重复率的计算。

如图，以左侧部分为更深区域，我们给出相关位置简图：

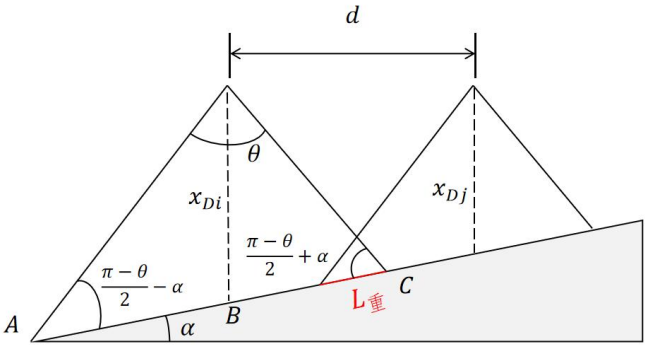

图 4 相关位置简图

利用图 3 中的几何关系，我们可以写出表格中相关参数的计算方式。其中AB记为$(\frac{W_i}{2})_{左}$，BC记为$(\frac{W_i}{2})_{右}$。

海水深度：若x_{Dj}已知，参考三角相关的公式，有

$$x_{Dj} = x_{Di} + \Delta d \cdot tan\alpha \tag{4}$$

覆盖宽度：将三角形分割成左右并运用正弦定理，有

$$\frac{sin(\frac{\pi-\theta}{2}-\alpha)}{x_{Dj}} = \frac{sin\frac{\theta}{2}}{(\frac{W_i}{2})_{左}} \tag{5}$$

$$\frac{sin(\frac{\pi-\theta}{2}+\alpha)}{x_{Di}} = \frac{sin(\frac{\theta}{2})}{(\frac{W_i}{2})_{右}} \tag{6}$$

$$(\frac{W_i}{2})_{左} + (\frac{W_i}{2})_{右} = W_{i总} \tag{7}$$

解出

$$W_i = x_{Di} \cdot sin\frac{\theta}{2}[\frac{1}{sin(\frac{\pi-\theta}{2}+\alpha)} + \frac{1}{sin(\frac{\pi-\theta}{2}-\alpha)}] \tag{8}$$

同时，易得覆盖宽度在海平面的投影长度为$W_{i总} \cdot cos\alpha$。

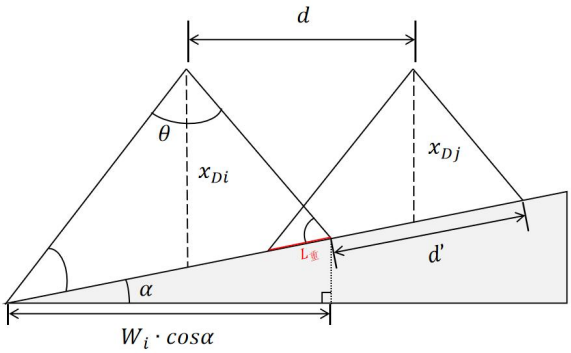

图 5 投影位置简图

对于此处重叠率η的计算，我们采取前文中在斜面以及复杂情况下修正后的定义，即将d以两条最右侧波束作为一组平行线投影到海底斜面上，那么有

$$\eta = 1 - \frac{d'}{W} \tag{9}$$

$$d' = d \cdot \frac{sin(\frac{\pi}{2} - \frac{\theta}{2})}{sin(\frac{\pi}{2} - \alpha + \frac{\theta}{2})} \tag{10}$$

5.1.3 模型求解

依据前文中所述的计算方法，此时我们可以得到计算所需的相关参数信息，并以此建立二维平面模型，模型计算结果如下表所示：

表 1 问题一的计算结果

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	90.949	85.712	80.474	75.237	70	64.763	59.526	54.288	49.051
覆盖宽度/m	315.813	297.628	279.442	261.256	243.070	224.884	206.699	188.513	170.327
与前一条测线的重叠率/%	--	0.357	0.315	0.267	0.213	0.149	0.074	-0.015	-0.124

可以发现最后两条测线出现了漏测，更详细的数值计算结果将在表格文件 result1.xlsx 中展示。

5.2 问题二模型的建立与求解

5.2.1 模型的准备

从第一问的求解中，我们可以提取出各种可以获取信息的参数。在对于后续问题的求解过程中，有 3 个重要参数：当前坡度γ，当前深度x_D，与当前多波束换能器的开角θ。由第一问所建立的模型可知，当前深度x_D由距初始航线的距离来决

定，所以对于问题二的解决，我们需要沿用第一问的模型并对此做出相应扩展。参考问题一的模型，可以得到参数间存在以下关系：

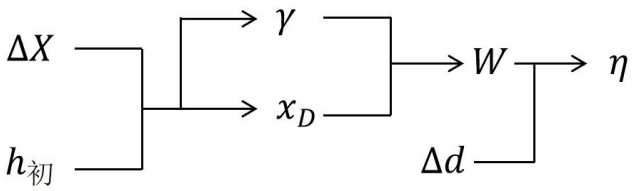

图 6 相关参数关系

在解决该问题情境下的矩形待测海域之前，我们首先需要证明 1 点：若海底平面为一均一平面，当测量船的航向确定（测线方向）时，其坡度也确定，且坡度为定值。这一点结论的证明过程如下：

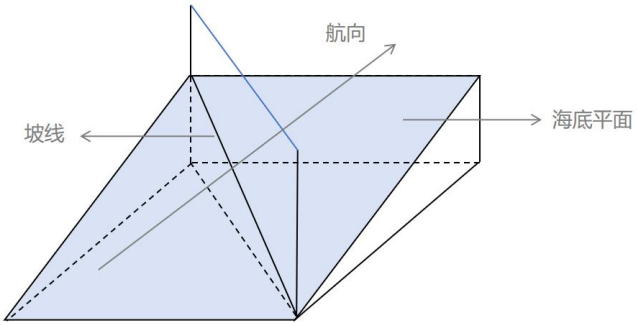

图 7 航向坡线关系图

假设航向法向为$\vec{n}_{航}=(a,b,c)$，则其代表了一个以$\vec{n}_{航}$为法向的平面。由于海平面水平，不难证明该平面与海平面垂直。若海底平面法向为$\vec{n}_{海}=(a',b',c')$，则坡线为两平面交线，且由克莱默法则可以计算出坡线方向为$(bc'-b'c,ca'-ac',ab'-ba')$。所以当测量船的航向确定（测线方向）时，坡度是唯一确定的，且为定值。

基于此，我们决定从深度与坡度着手来解决关于该矩形待测海域的相关计算，并完善多波束测深覆盖宽度的数学模型。考虑下图的情况（这里我们以$\beta=135°$为例）：

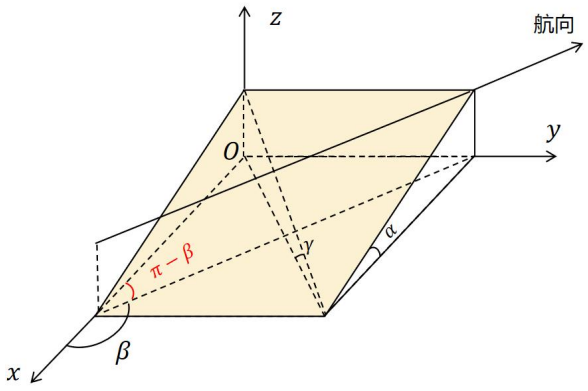

图 8 $\beta=135°$时几何关系图

在图 7 中，我们需要得知当前坡度γ的值。设某一高度为h，那么有

$$tan\gamma = \frac{h}{\sqrt{(\frac{h}{tan\alpha})^2 + (\frac{h}{tan\beta \cdot tan\alpha})^2}} = tan\alpha \cdot |sin\beta| \tag{11}$$

对于深度的计算，可以得到

$$x_{Di} = x_{D初} - \Delta x \cdot cos(\pi - \beta) \cdot tan\alpha \tag{12}$$

将变化后的参数引入第一问，可以求得覆盖宽度$W_{覆}$的数值

$$W_{覆} = x_D \cdot sin\frac{\theta}{2}(\frac{1}{sin\frac{\pi-\theta}{2} - \gamma} + \frac{1}{sin\frac{\pi-\theta}{2} + \gamma}) \tag{13}$$

至此，我们已经实现了将第一问的模型升维到第二问的工作。在改变Δx与β的情况下，可以实现对特定参数的数值计算。

5.2.2 模型的求解

上文中，我们建立了在不同Δx与β的数值下，计算多波束测深的覆盖宽度的数学模型，求解的结果以保留小数点后三位的形式放在如下表格中，由于篇幅问题，更详细的数值将在表格文件 result2.xlsx 中呈现。

表 2 问题二的计算结果

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	415.692	466.091	516.490	566.889	617.288	667.687	718.085	768.484
	45	416.192	451.872	487.552	523.232	558.912	594.592	630.273	665.953
	90	416.692	416.692	416.692	416.692	416.692	416.692	416.692	416.692
	135	416.192	380.511	344.831	309.151	273.471	237.791	202.110	116.430
	180	415.692	365.293	314.895	264.496	214.097	163.698	113.299	62.900
	225	416.192	380.511	344.831	309.151	273.471	237.791	202.110	116.430
	270	416.692	416.692	416.692	416.692	416.692	416.692	416.692	416.692
	315	416.192	451.872	487.552	523.232	558.912	594.592	630.273	665.953

通过观察求解结果，可以发现以下规律：(1)在同一点不同航向会使覆盖宽度产生较大变化。(2)在不同深浅的水域中覆盖宽度变化较大，直线行驶时产生的轨迹为类三角（锥）形，并不均匀。(3)经过检验，在参数一致的情况下该模型与第一问建立的模型得出了相同的结果，验证了模型的正确性。

5.3 **问题三模型的建立与求解：**

5.3.1 问题分析

这个问题中，需要我们在南北长 2 海里、东西宽 4 海里的矩形海域内求解一组长度最短，且覆盖全海域的测线方案。这里可以提取出两个重要信息：对于测线方案的设计来说，覆盖整个海域面积和长度最短是必然要求。假设在每处其行进微小的距离为Δx，在该处的覆盖宽度为w_i，那么有$w_i = f(\gamma, x_{Di})$，则船体行进完成后总的覆盖面积S为

$$S = \int w_i \, \Delta x \tag{14}$$

为了满足题干要求，该面积要覆盖整个海域面积。

在第二问的最后一部分结果分析中，我们提到了在不同深度、不同角度下，覆盖宽度w_i也不同。基于贪心算法的思想，为使测线的长度Δx最短，需要使得w_i在各处尽可能宽。由此可得到最短测线距离。

下面我们对不同深度下不同角度的w_i均做出计算，并且可视化出了较深处与较浅处的结果用作说明：

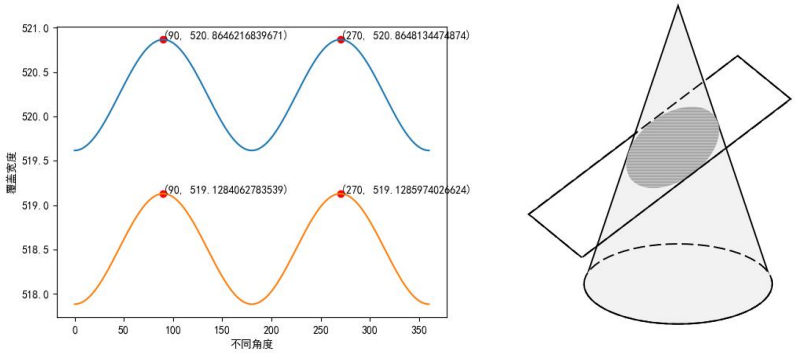

图 9 较浅较深情况（左）与单独抽取的圆锥结构（右）

由图可看出，不同坡度下，无论大小，总是在方向为$\frac{\pi}{2} + n\pi$的地方取得最高值，此时是与其等高线共线的情况。如果单独抽取出该结构，将不同的方向模拟为一个圆锥，而坡面模拟为一个斜切的面，此时由几何知识可知，该切面为椭圆。而当W的长度为其长轴时为最大，这也就是之前航向与等高线平行的原因。

由图同时可以观察到，无论是左图中的较浅情况还是右图中所示的较深情况，均于测线方向在 90°与 270°时取得最优的覆盖宽度。**而这个方向恰巧是与各点梯度向量垂直的方向**。也就是当沿图 10 所示方向的情况下，各处可以取得最优的覆宽以及测线长度：

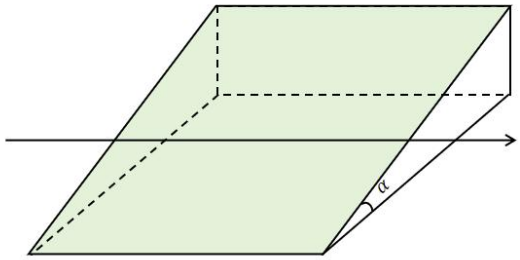

图 10 最优方向

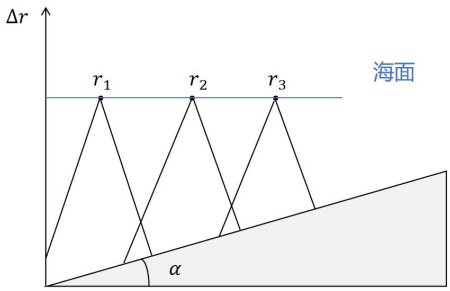

图 11 测线方案问题简化图

在确定了每条航线的方向后，该问题就演变成在图 11 所示平面中合理安排测线方案的问题。

5.3.2 模型建立

为了保证数据的准确性和处理的简单性，重叠率的范围有一定要求。在题目重叠率$\eta \in [10\%, 20\%]$的要求下，我们给出以下计算步骤。其中“重复”的步骤指的是以η为优化与控制结果的关键指标进行迭代计算。

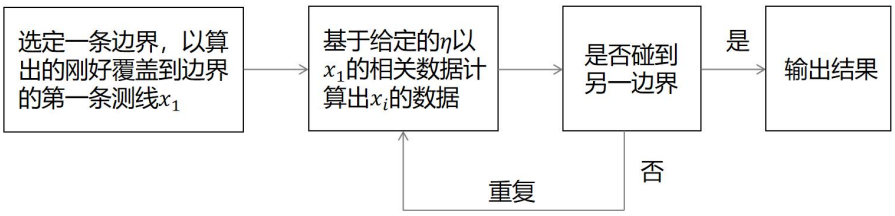

图 12 计算流程图

在已知边界深度的计算下，第一条测线的计算方法如下：

由$x_{D初}$、α得

$$(x_{D初} - \Delta r \cdot tan\alpha) \cdot sin\frac{\theta}{2} \cdot cos\alpha = x \cdot sin(\frac{\pi - \theta}{2} - \alpha) \tag{15}$$

$$\Delta r = \frac{cos\alpha \cdot x_{D初} \cdot sin\frac{\theta}{2}}{sin(\frac{\pi-\theta}{2} - \alpha) + sin\alpha \cdot sin\frac{\theta}{2}} \tag{16}$$

$$x_{D1} = x_{D初} - \Delta r \cdot tan\alpha \tag{17}$$

其中Δr为第一条测线距边界得距离，x_D为其测线深度。同时，在相邻测线之间还具有以下关系：

$$(\frac{w_i}{2})_{左} + (\frac{w_i}{2})_{右} - \Delta d = L_{重} \tag{18}$$

$$\frac{L_{重}}{w_i}=\eta \tag{19}$$

参考一、二问中建立的计算宽度的模型，并结合上几式易得

$$\frac{x_{Di-1}\cdot sin\frac{\theta}{2}}{A}+\frac{x_{Di}\cdot sin\frac{\theta}{2}}{B}-\frac{x_{Di-1}-x_{Di}}{tan\alpha}=\eta\cdot x_{Di}\cdot sin\frac{\theta}{2}(\frac{1}{A}+\frac{1}{B}) \tag{20}$$

将式(20)化简，那么有

$$x_{Di}=x_{Di-1}\frac{\frac{sin\frac{\theta}{2}}{A}-\frac{1}{tan\alpha}}{\eta\cdot sin\frac{\theta}{2}(\frac{1}{A}+\frac{1}{B})-\frac{sin\frac{\theta}{2}}{B}-\frac{1}{tan\alpha}} \tag{21}$$

式(21)即为以前一条测线深度推出后一条测线深度的递推公式。其中x_{Di}为待计算深度，x_{Di-1}为前一已知深度。其中，$A=sin(\frac{\pi-\theta}{2}+\alpha)$，$B=sin(\frac{\pi-\theta}{2}-\alpha)$。

由于在前文的流程中未确定从较浅处开始迭代还是较深处开始迭代，因此本组对两种情况都做出了计算。我们发现当选择从较深处开始迭代时，最后多测的面积（即超出待测海域的面积）要小于从最浅处开始迭代多测的面积。推测产生此现象的原因，可能是浅处测线覆盖宽度较小、密度较大，所以并不会重复测得很多面积。

在以不同重叠率为横坐标、测线数量为纵坐标的图 13（左）中，我们不难看出测线数量呈阶梯状比例，最小为 34 个，最多为 38 个。这样的结果符合重叠越少，测线越少的直观感受。

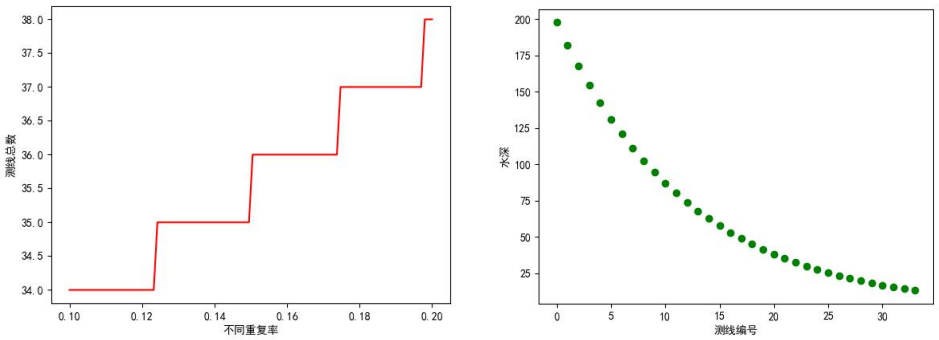

图 13 不同重叠率与测线数量关系（左）与测线位置（右）

同时，我们对当测线数量最少的情况下，$\eta\in[10\%,12.2\%]$测线的具体位置进行了可视化处理，在图 13（右）中，横坐标为距初始点距离，纵坐标为测线的垂直深度。可以观察到在垂直深度较深处分布稀疏，而在较浅处分布较为密集。

我们以$Max(W_i)$，$Min(number(r_i))$为优化目标，找出了理想的测线数量。在这里给出了平行的测线方位与位置（见附件）。我们还需要将这些平行测线以“以”字形方式连接起来，连接后便得到了一条完整路径，见图。

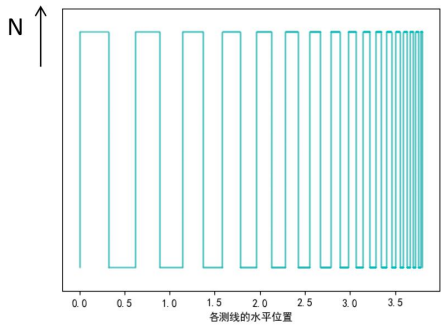

图 14 连接后的路径

5.4 问题四的求解

5.4.1 数据初步处理及可视化

在文件附件.xlsx 中展示了某海域单波束测量的测深深度数据，该海域范围较大，且数据单位不统一，因此坐标点显得较为稀疏。为了更直观地呈现附件中的数据，并对问题做出初步的分析，我们首先采用分段三次埃尔米特插值对数据进行差值扩充与可视化：

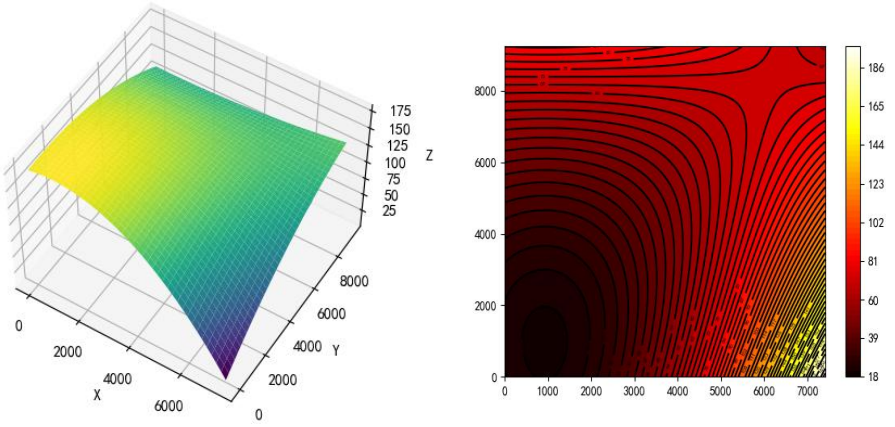

图 15 海底深度三维图（左）与等深线地形图（右）

在图 15 中可以观察到，海域大概是从(0,0)点向周围下降，根据图 15（右）所示的等深线地形图来看，在右下角的位置达到最深，在右上角线条稀疏，形成了一个较为平缓的鞍部。

由于给出的数据是有一定间隔的离散点。在路径优化的时候无法得到任一坐标的相关数据，要克服这一问题的一种方法是插入足够多的数值，用离散的点代替平滑图像。但这种方法在后续计算中有一定缺点：(1)精度较低。(2)由于某些点的缺失导致计算失效。(3)误差累积，致使结果误差很大。(4)数据量过于庞大，使

用复杂，计算困难，耗时较长。

针对上述问题，经过思考，我们决定采用大型学习模型进行拟合。也就是将已有点的数据作为训练集，建立一个用于辅助计算的机器学习模型，这样，我们给定坐标后，便可以给出精确的数据。使用这种方法将离散转为连续，能够大大提高计算的精确度和便利性。

为此，我们尝试了线性回归、支持向量机、随机森林等多种模型，并以拟合接近度R^2进行评估，**最终发现随机森林模型与原数据拟合度可以达到99.999%，几乎能完全拟合原数据。因此，我们选择训练该模型并作为后续数据的支撑。拟合效果如下：**

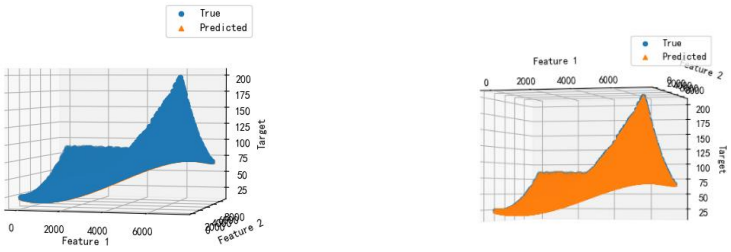

图 16 海底深度三维图（左）与模型拟合效果图（右）

基于该模型，同样可以做到对任意一点深度，梯度的拟合，进而计算出每个点的坡度，因此我们对前面部分的某些概念做出优化：

设有一点(x_i, y_i)，则该点

梯度单位向量（最大坡度向量）：$\vec{g}(x_i, y_i)$

在该梯度上的坡度：$\alpha(x_i, y_i)$

在该点深度：$x_D(x_i, y_i)$

在该点探测区域宽度：$w(x_i, y_i)$

至此，我们完成了相关数据的准备工作。

5.4.2 问题分析

基于前面几个问题的研究，我们可以明确以下信息：

$$目的优化\begin{cases} min(L_{总}) \\ min(S_{测} - S) \end{cases} \tag{22}$$

则有

$$\begin{cases} min(L_{总}) \\ min(S_{测} - S) \end{cases} \rightarrow max(w(x_i, y_i)) \rightarrow 最佳航线方向 \tag{23}$$

找到每一处的最大覆盖宽度是解决问题的关键，这同样也关系到确定航向的问题。为了贴合题目中的要求，需要分别从航向的确定和重叠率来考虑。

关于航向的确定，在第三问结尾的相关探索部分已经有所说明：沿等高方向

行驶会获得最大覆宽w。曲面情况下也需要考虑，因此我们做出了下图中的模拟：

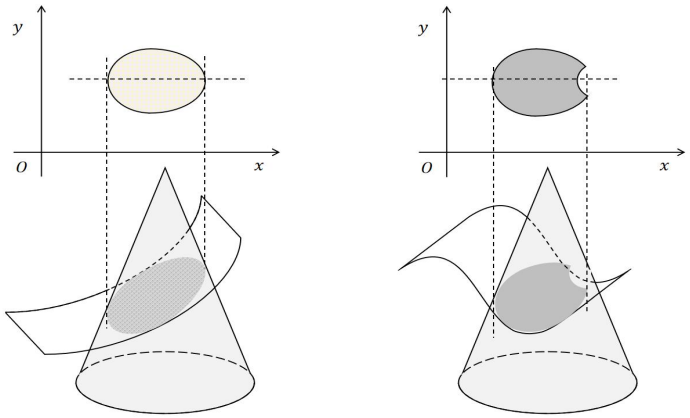

图 17 曲面截取圆锥的截面图

在上图中，我们用圆锥模拟不同方向，用两类斜面去截圆锥，从其在水平面的投影来寻找最优的w_i，可以看出它们的投影为类椭圆，总体仍然以长轴（即沿梯度为开角方向）取最大，但在极端情况下，可能会出现右侧中长轴变短的情况，不过该地形在本题地图中并未出现，且也可近似认为其类长轴为最宽处。

关于重叠率，本题并未给出前几问中那样精确的范围，只需要不超过 20%，参考题干中的“为保证测量的便利性和数据的完整性，相邻条带之间应有 10%~20% 的重叠率”，我们以此作为标准。

对于开角的确定，题目中并未给出数值上的参考。我们对开角θ在$\pm 30°$内进行了覆宽变化的探究。在深度为 70 的情况下，做出如下三维图。

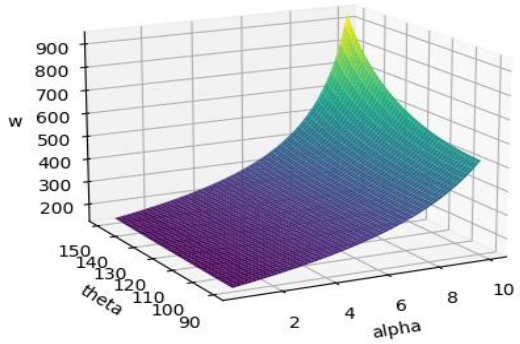

图 18 不同张角和坡度下最大覆宽变化图

可以看出，在开角变大，坡度变大的情况下，测宽显著变大，在地形较倾斜范围海域造成误差，因此开角的选择应根据地形进行合理的制定。开角变小会让测量宽度变小而使航线变密集，航线的分布也会产生微小变化。在本题仍以$\theta = 120°$为标准。

5.4.3 模型建立

本题实际上是一个特殊的优化问题，其总体的最优解总是由局部最优解组成。

基于前文航向对w大小的探究，其航向需要与其梯度方向垂直（或与等高线相切）.针对这个过程，本文查阅到一个很符合该题的群体智能算法“飞蛾火焰算法”[2][4]，**同时基于贪心思想和微分思想，并结合第三问，修改了该算法的螺旋函数与适应函数的确定方式，形成自适应的“飞蛾火焰”算法。**

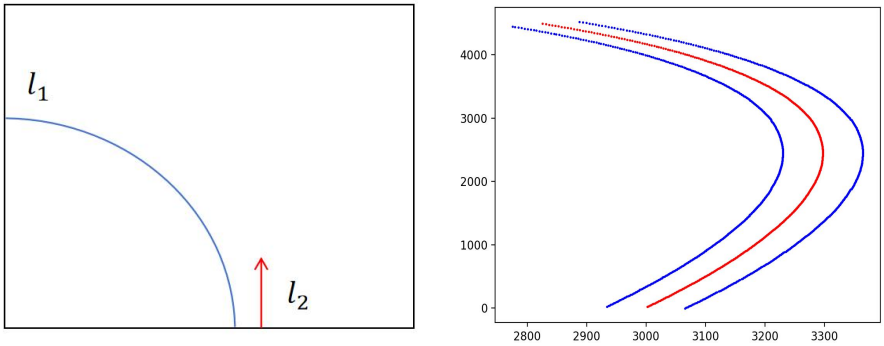

图 19 航线图示说明（左）航线运行方向及覆盖示意图（右）

假设有如图所示的航线，并有正在探索的航线l_2，本组做出以下调整：

(1) 自适应火源调整方案

该火源存在的本意是为了让搜索粒子与火源位置保持一定角度，但本题地形复杂，因此采用随机森林预测出的梯度$\nabla\vec{g}(x_i,y_i)$和一个与$\nabla\vec{g}(x_i,y_i)$正交的单位向量来作为测线方向的自调整的角度。

(2) 地图微分化：

由于地图范围非常大，而探索船每次探测的部分占比很小，因此我们采用微分思想，将每一点(x_i,y_i)处的区域认为是与题三中平坦的情形，并采用(x_i,y_i)处的w、α、g作为整个小区域的数据，对于相邻区域内的另一点(x_j,y_j)满足：

$$\alpha(x_i,y_i)=\alpha(x_j,y_j) \tag{24-1}$$

$$\nabla\vec{g}(x_i,y_i)=\nabla\vec{g}(x_j,y_j) \tag{24-2}$$

$$x_D(x_i,y_i)=x_D(x_j,y_j) \tag{24-3}$$

(3) 适应度函数：

适应度函数是我们需要满足与优化的函数，这里主要是相邻覆盖度，示意图中，在l_2路线向前探索时不仅要考虑与$\nabla\vec{g}(x_i,y_i)$垂直的方向来保证每处最大的$w_{max}(x_i,y_i)$，还需要考虑相邻（l_1对l_2）产生的重叠区域。因此本组在算法寻找最短路径时，设定一个量$\eta(x_i,y_i)$。

计算方式如下：图示情况中(x_i,y_i)为要搜索的点，(x_j,y_j)为相邻已确定的点，且l_2深度更深。则

$$L_{重}=\frac{x_D(x_i,y_i)\cdot sin\frac{\theta}{2}}{sin[\frac{\pi-\theta}{2}+\alpha(x_i,y_i)]}+\frac{x_D(x_i,y_i)\cdot sin\frac{\theta}{2}}{sin[\frac{\pi-\theta}{2}-\alpha(x_i,y_i)]}-\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}\cdot cos\alpha(x_i,y_i) \tag{24}$$

$$\frac{L_{重}}{w(x_i,y_i)}=\eta(x_i,y_i) \tag{25}$$

$$w(x_i,y_i)=x_D(x_i,y_i)\cdot sin\frac{\theta}{2}(\frac{1}{sin[\frac{\pi-\theta}{2}+\alpha(x_i,y_i)]}+\frac{1}{sin[\frac{\pi-\theta}{2}-\alpha(x_i,y_i)]}) \tag{26}$$

该式与第三问的计算结构类似，由于本题地形更加起伏，所以其中坡度、梯度等数据均随位置改变，并由训练出的随机森林数据模型提供。

位置更新：

在计算重叠率时，以沿梯度向上（下）的临近点作为参照，所以我们的算法保存住其沿梯度$\nabla\vec{g}(x_i,y_i)$上的最近点。

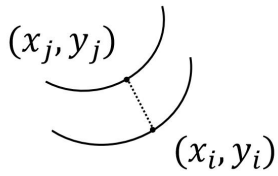

图 20 最近点图示

如图，由于梯度垂直其等高线，那么(x_j,y_j)一定是最近点。在搜索完一路后，我们以梯度作为方向，让粒子跃迁到下一路。并且有

$$x_{Di}=x_{Di-1}\frac{\frac{sin\frac{\theta}{2}}{A}-\frac{1}{tan\alpha(x_{i-1},y_{i-1})}}{\eta\cdot sin\frac{\theta}{2}(\frac{1}{A}+\frac{1}{B})-\frac{sin\frac{\theta}{2}}{B}-\frac{1}{tan\alpha(x_{i-1},y_{i-1})}} \tag{27}$$

$$x_i=x_{i-1}+\frac{\nabla\vec{g}(x_{i-1},y_{i-1})}{|\nabla\vec{g}(x_{i-1},y_{i-1})|}\cdot(1,0)\cdot\frac{x_{Di}-x_{Di-1}}{tan\alpha(x_{i-1},y_{i-1})} \tag{28}$$

$$y_i=y_{i-1}+\frac{\nabla\vec{g}(x_{i-1},y_{i-1})}{|\nabla\vec{g}(x_{i-1},y_{i-1})|}\cdot(0,1)\cdot\frac{x_{Di}-x_{Di-1}}{tan\alpha(x_{i-1},y_{i-1})} \tag{29}$$

其中，$A=sin[\frac{\pi-\theta}{2}+\alpha(x_{i-1},y_{i-1})]$，$B=sin[\frac{\pi-\theta}{2}-\alpha(x_{i-1},y_{i-1})]$。上式是其跃迁的迭代公式。

5.4.4 模型的应用及求解

基于上述已建模型，我们对航线的计算流程如下：

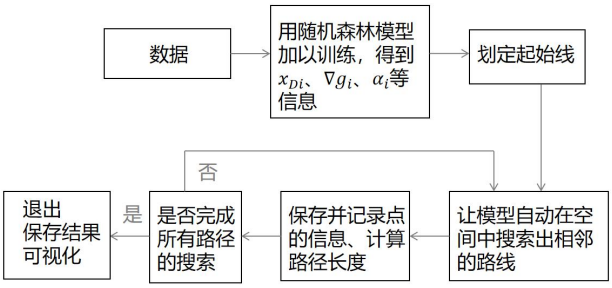

图 21 航线计算流程图示

经过初步尝试，发现不同起始线最后以固定η探索出来的路径不同，且在某些区域存在漏测和鲁棒性较差等问题，可用性较差：

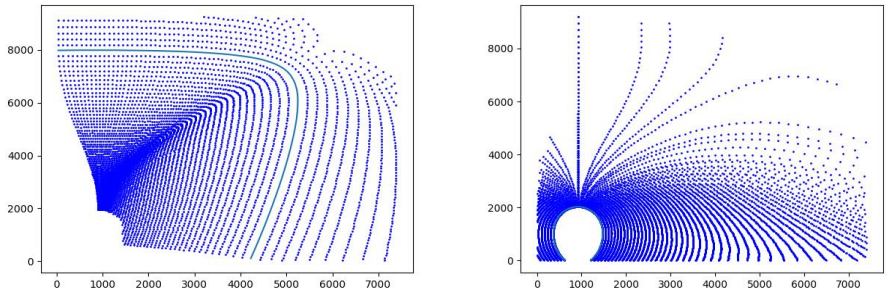

图 22 初步探索路径

这是由于区域参数不同造成的，针对该问题，我们首先考虑到的是分区进行规划，参考等高线，将地图分成了下面几个区域，并适当改进探索步长，参数计算等算法：

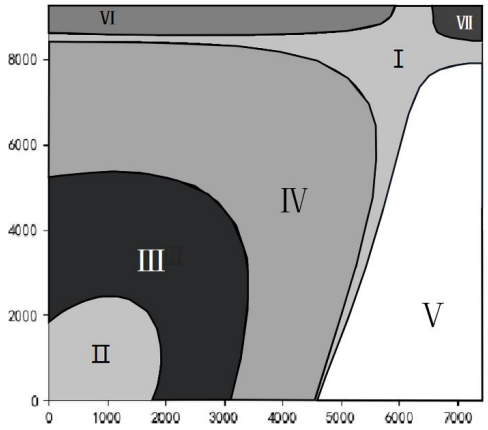

图 23 地区划分

在每个区域内部，走势较为统一，但某些区内，由于区域内浅且坡度平缓（Ⅰ，Ⅱ区域）存在随机搜索复杂度较高的情况，于是我们将这类区域单独拿出来进行拟合，得到最终结果。同时，对于不同的初始线，本组也做了相关统计，并最终选择总路线最短的一组作为出实现来作为最佳初始线分布。最终我们综合算法自探索的路线和我们处理的路线得到了所有测线，船只仍可按“己”字形遍历所有

测线（如下图）。

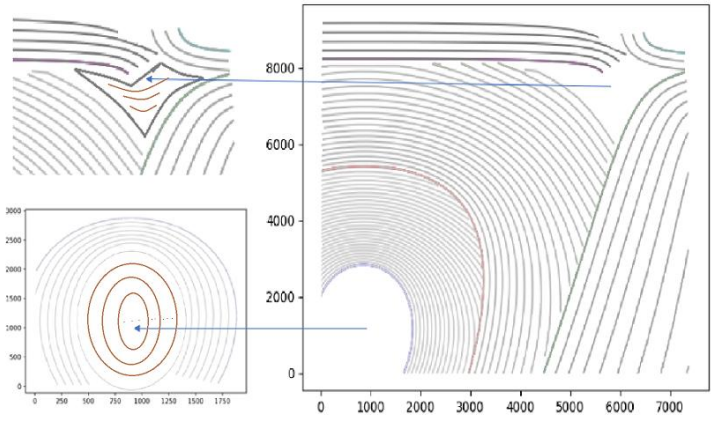

图 24 路径规划示意图

(注：左下角圆形区域的曲线仅供位置上参考，具体条数见下表)

由于图线过于密集，数量较多，图像提供大概位置示意，具体的位置，已在附件里以散点数据的形式呈现

基于积分的思想，我们用下列式子得到总长和覆盖面积，并对其他值进行计算：

$$S' = \sum_{i=1}^{n} w(x_i, y_i) \cdot \Delta L \tag{30}$$

$$L_{总} = \sum_{i=1}^{n} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{31}$$

并给出了我们每个分区的路线相关数据与统计分析数据（如下表）。**由于测线之间有 10%重叠率，所以有效覆盖面积为 68567951，占总海洋面积 68598080 的 99.956%。**

表 3 分区计算结果

分区	测线长度	覆盖面积	测线条数
Ⅰ	2922	810000	3
Ⅱ	49074	3740000	22
Ⅲ	128930	13552572	28
Ⅳ	164724	30913439	24
Ⅴ	54820	18319129	9
Ⅵ	29974	7809427	5
Ⅶ	4218	1042046	3
总和	434662	76186613	94

由于测线之间有 10%重叠率，所以有效覆盖面积为 68567951，占总海洋面积 68598080 的 99.956%。

相关指标的计算如下表：

表 4 相关指标计算结果

测线的总长度	漏测海区所占百分比	重叠率超过 20% 部分的总长度
434662	0.04%	21998

对于该路线，是在$\eta = 10\%$的情况下给出的最短距离，我们尝试了计算之后，当$\eta = 0\%$其路线总长最短，大概为$\eta = 10\%$情形下的总长的 90%，这也符合其数学原理。

如果测量船需走连续的路线或回到最开始的原位置，还可以参考相关汉密尔顿图和欧拉路径的布置方式对路线进行规划。

六、 模型的评价

6.1 模型的优点

1．对于覆盖宽度等等信息给出了确切的公式，计算精准。

2．采用学习类模型来处理数据，方便了后续的计算的进行，同时增加了精度

3．融合并改进了相关算法，并分类解决，更加贴近问题，并能给出较好的解

6.2 模型的缺点

1．在最后路线的运算中我们只是抽一些点来进行，但运算量已经较大，要想获得更高精度的结果可能需要的时间较长。

2．划分区域解决问题后，在不同区域交汇处需要二次处理。

6.3 模型的推广

该模型可以优化高度为目标更换高度，开角也可切换为更多情况，可用于建筑扫描，无人机地形扫描等多种情况，具有较多的可用空间。

七、 参考文献

[1]. GB/T 12763.10-2007 海洋调查规范[S].第 10 部分:海底地形地貌调查

[2]. 徐炜翔，朱志宇.基于飞蛾火焰算法的 AUV 三维全局路径规划.上海理工大学学报,2021,43(2): 148-155.

[3]. 陆丹，李海森，魏玉阔，邓平.多波束测深系统中的海底地形可视化技术研究[A].仪器仪表学报,2012.2

[4]. 王智慧，代永强，刘欢.基于自适应飞蛾扑火优化算法的三维路径规划[A].计算机应用研究.2023.1

附录

附录材料目录：

1. 处理过的部分数据

2. 代码部分

第二问.py
第三问(1).py
第三问(2).py
第三问(3).py
第三问(4).py
第三问(5)张角与坡度.py
第四问(1)拟合函数.py
第四问(2)梯度.py
第四问(3)插值.py
第四问(4).py
第四问(5).py
第四问(6).py
第四问(7).py
第四问(8).py
第四问(9).py
第一问(1).py

代码部分：

第一问：

```
import pandas as pd
import numpy as np
# 初始化参数
D_0 = 70
theta = 120
alpha = 1.5
d = 200
d = d*np.sin(np.radians(90-theta/2))/np.sin(np.radians(90-alpha+theta/2))

distances = np.array([-800, -600, -400, -200, 0, 200, 400, 600, 800])
D= D_0 - distances * np.tan(np.radians(alpha))

print(D)

W=D*np.sin(np.radians(theta/2))*( 1/np.sin(np.radians((180-theta)/2+alpha)) +1/np.sin(np.radians((180-theta)/2-alpha)))
print(W)
n=1-d/W
print(n)
# 创建 DataFrame 用于保存结果
df = pd.DataFrame({'测线距中心点处的距离/m': distances})
```

```
df['海水深度/m'] = D
df['覆盖宽度/m'] = W
df['与前一条测线的重叠率/%'] = n
# # 将 DataFrame 保存为 Excel 文件
# path=r'C:\Users\PC\Desktop\res1.xlsx'
# df.to_excel(path, index=False)
```

第二问：

```
import pandas as pd
import numpy as np
def get_width(B):
    # 初始化参数
    D_0 = 120  # 海底深度（单位：m）
    alpha = 1.5  # 坡度（单位：度）
    D = D_0 - distances * np.tan(np.radians(alpha)) * np.cos(np.radians(180 - B))
    theta = 120  # 换能器的开角（单位：度）
    alpha= np.arctan(abs(np.sin(np.radians(B)))*np.tan(np.radians(alpha)))*180/np.pi
    print(D)
    W = D * np.sin(np.radians(theta / 2)) * (
                1 / np.sin(np.radians((180 - theta) / 2 + alpha)) + 1 / np.sin(np.radians((180 - theta) / 2 - alpha)))
    print(W)
    return W
distances = np.array([0, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1])
distances = distances * 1852
print(distances)

angle=[0,45,90,135,180,225,270,315]
W=[]
for i in angle:
    W.append(get_width(i))

# 将 DataFrame 保存为 Excel 文件
path=r'C:\Users\PC\Desktop\res2.xlsx'
pd.DataFrame(W).to_excel(path, index=False)
```

第三问：

(1).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] #显示中文
def get_width(B,D_0):
    # 初始化参数
    alpha = 1.5  # 坡度（单位：度）

    D = D_0
    theta = 120  # 换能器的开角（单位：度）
    alpha= np.arctan(abs(np.sin(np.radians(B)))*np.tan(np.radians(alpha)))*180/np.pi
```

```
    print(D)

    W = D * np.sin(np.radians(theta / 2)) * (
                1 / np.sin(np.radians((180 - theta) / 2 + alpha)) + 1 / np.sin(np.radians((180 - theta) / 2 - alpha)))

    print(W)
    return W

angle=np.linspace(0,360,360)
W=[]
for i in angle:
    W.append(get_width(i,150))

print(W)
plt.plot(angle,W)

plt.scatter(90,W[89],color='r')
plt.scatter(270,W[269],color='r')
plt.text(90,W[89], '({}, {})'.format(90,W[89]))
plt.text(270,W[269], '({}, {})'.format(270,W[269]))

angle=np.linspace(0,360,360)
W=[]
for i in angle:
    W.append(get_width(i,149.5))

print(W)
plt.plot(angle,W)

plt.scatter(90,W[89],color='r')
plt.scatter(270,W[269],color='r')
plt.text(90,W[89], '({}, {})'.format(90,W[89]))
plt.text(270,W[269], '({}, {})'.format(270,W[269]))

plt.xlabel("不同角度")
plt.ylabel("覆盖宽度")
plt.show()
```

(2).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] #显示中文

def sin(a):
    return np.sin(np.radians(a))
def cos(a):
    return np.cos(np.radians(a))
def tan(a):
    return np.tan(np.radians(a))
```

```
12.
13. angle=np.linspace(0,360,360)
14. low=110-2*1852*np.tan(np.radians(1.5))
15. high=110+2*1852*np.tan(np.radians(1.5))
16.
17. alpha = 1.5  # 坡度（单位：度）
18. theta = 120  # 换能器的开角（单位：度）
19.
20. n=np.linspace(0.1,0.2,100)
21. cnt=[]
22. for i in n:
23.     x = sin(theta / 2) * cos(alpha) * high / (sin(90 - theta / 2 - alpha)
 + sin(alpha) * sin(theta / 2))
24.     x = high - x * tan(alpha)
25.     print(x)
26.     ans = []
27.     ans.append(x)
28.     A = sin(90 - theta / 2 + alpha)
29.     B = sin(90 - theta / 2 - alpha)
30.     C = sin(theta / 2) / A - 1 / tan(alpha)
31.     D = i * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B - 1 / t
an(alpha)
32.
33.     while True:
34.         x = x * C / D
35.         if x < low:
36.             break
37.         ans.append(x)
38.
39.     # print(len(ans))
40.     cnt.append(len(ans))
41.     # print(ans[-1])
42.
43. n=np.array(n)
44. print(n)
45. print(cnt)
46. plt.plot(n,cnt,color='r')
47. plt.xlabel("不同重复率")
48. plt.ylabel("测线总数")
49.
50. plt.show()
```

(3).

```
1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt
4. plt.rcParams['font.sans-serif']=['SimHei'] #显示中文
5.
6. def sin(a):
7.     return np.sin(np.radians(a))
8. def cos(a):
9.     return np.cos(np.radians(a))
```

```
def tan(a):
    return np.tan(np.radians(a))
def get_Wleft(D):
    return D*sin(theta/2)/sin(90-theta/2-alpha)


angle=np.linspace(0,360,360)
low=110-2*1852*np.tan(np.radians(1.5))
high=110+2*1852*np.tan(np.radians(1.5))

alpha = 1.5  # 坡度（单位：度）
theta = 120  # 换能器的开角（单位：度）

n=0.1
cnt=[]
x = sin(theta / 2) * cos(alpha) * high / (sin(90 - theta / 2 - alpha) + sin(alpha) * sin(theta / 2))
x = high - x * tan(alpha)
print(x)
ans = []
ans.append(x)
A = sin(90 - theta / 2 + alpha)
B = sin(90 - theta / 2 - alpha)
C = sin(theta / 2) / A - 1 / tan(alpha)
D = n * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B - 1 / tan(alpha)

while True:
    x = x * C / D
    if x < low:
        break
    ans.append(x)

# print(len(ans))
# print(ans[-1])
index=np.arange(len(ans))
ans=np.array(ans)
dis=[]
for i in range(len(ans)-1):
    dis.append((ans[i]-ans[i+1])/tan(alpha))
for i in range(len(dis)-1):
    dis[i+1]+=dis[i]


# plt.scatter(index,ans,color='g')
# plt.xlabel("测线编号")
# plt.ylabel("水深")
# plt.show()
print(dis)
dis.insert(0,0)
dis=np.array(dis)/1852
```

```
y=np.zeros(len(dis))
# plt.scatter(dis,y,marker='x',s=10)
plt.xlabel("各测线的水平位置")
plt.ylim(-1.2,1.2)
plt.yticks(alpha=0)
plt.tick_params(axis='y', width=0)
y=np.linspace(-1,1,10000)
for i in range(len(dis)-1):
    x=np.full((1,10000),dis[i])
    plt.scatter(x,y,s=0.0001,color='c')
    tx=np.linspace(dis[i],dis[i+1],1000)
    if i %2==0:
        ty = 1
    else:
        ty=-1
    ty = np.full((1, 1000), ty)
    plt.scatter(tx, ty, s=0.0001, color='c')
x=np.full((1,10000),dis[-1])
plt.scatter(x,y,s=0.0001,color='c')
plt.show()
path=r'C:\Users\PC\Desktop\距离.xlsx'


# pd.DataFrame(dis).to_excel(path)
# # path=r'C:\Users\PC\Desktop\水深.xlsx'
# # pd.DataFrame(ans).to_excel(path)
```

(4).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] #显示中文

def sin(a):
    return np.sin(np.radians(a))
def cos(a):
    return np.cos(np.radians(a))
def tan(a):
    return np.tan(np.radians(a))
def get_Wleft(D):
    return D*sin(theta/2)/sin(90-theta/2-alpha)
def get_WRight(D):
    return D*sin(theta/2)/sin(90-theta/2+alpha)

angle=np.linspace(0,360,360)
low=110-2*1852*np.tan(np.radians(1.5))
high=110+2*1852*np.tan(np.radians(1.5))

alpha = 1.5  # 坡度（单位：度）
theta = 120  # 换能器的开角（单位：度）

n=0.1
```

```
cnt=[]
x = sin(theta / 2) * cos(alpha) * high / (sin(90 - theta / 2 - alpha) + sin(alpha) * sin(theta / 2))
x = high - x * tan(alpha)
print(x)
ans = []
ans.append(x)

while True:
    high=x-(get_WRight(x)-(get_WRight(x)+get_Wleft(x))*n)*sin(alpha)
    x = sin(theta / 2) * cos(alpha) * high / (sin(90 - theta / 2 - alpha) + sin(alpha) * sin(theta / 2))
    x = high - x * tan(alpha)
    if x<low:
        break
    ans.append(x)

print(ans)
print(len(ans))
print(ans[-1])
index=np.arange(len(ans))
plt.scatter(index,ans,color='g')
plt.xlabel("测线编号")
plt.ylabel("水深")

plt.show()
# path=r'C:\Users\PC\Desktop\水深.xlsx'
# pd.DataFrame(ans).to_excel(path)
# # path=r'C:\Users\PC\Desktop\水深.xlsx'
# # pd.DataFrame(ans).to_excel(path)
```

(5).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
def sin(a):
    return np.sin(np.radians(a))
def cos(a):
    return np.cos(np.radians(a))
def tan(a):
    return np.tan(np.radians(a))
def get_Wleft(D):
    return D*sin(theta/2)/sin(90-theta/2-alpha)


angle= 90
alpha = np.linspace(0.5,10,100)
theta = np.linspace(90,150,100)
x_d=70
w=[]
for i in alpha:
    for j in theta:
```

```
            w.append(x_d*sin(j/2)*(1/sin(90-j/2-i)+1/sin(90-j/2+i)))
print(w)
w=np.array(w).reshape(100,100)
alpha, theta = np.meshgrid(alpha, theta)

# 创建三维图形对象和坐标轴
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# 绘制三维图形
ax.plot_surface(alpha, theta, w, cmap='viridis')

# 设置坐标轴标签
ax.set_xlabel('alpha')
ax.set_ylabel('theta')
ax.set_zlabel('w')

# 显示图形
plt.show()
```

第四问：

(1).拟合函数

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei'] #显示中文

path=r'C:\Users\PC\Desktop\附件.xlsx'

df=pd.read_excel(path)
x=np.array(df.iloc[0][2:],dtype="float64")

y=[]
for i in range(1,df.shape[0]):
    y.append(df.iloc[i][1])

y=np.array(y,dtype="float64")

x=x*1852
y=y*1852

path=r'C:\Users\PC\Desktop\高度.xlsx'
df=pd.read_excel(path)
Z=np.array(df.iloc[0:][0:],dtype="float64")

print(x)
print(x.shape)
print(y.shape)
print(Z.shape)
data=[]
for j in range(len(y)):
```

```
        for i in range(len(x)):
            t=[x[i],y[j],Z[j][i]]
            data.append(t)
data=np.array(data)
print(data.shape)

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from joblib import dump, load
def get_model(data):
    X = data[:, 0:2]
    y = data[:, 2]

    # 数据分割
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
2, random_state=42)

    # 创建随机森林模型
    rf = RandomForestRegressor(n_estimators=100, random_state=42)

    # 训练模型
    rf.fit(X_train, y_train.ravel())

    # 预测
    y_pred = rf.predict(X_test)

    # 评估模型
    score = rf.score(X_test, y_test)
    print(f"R^2 Score: {score}")

    # 绘图展示预测结果和实际结果
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(X_test[:, 0], X_test[:, 1], y_test, label='True')
    ax.scatter(X_test[:, 0], X_test[:, 1], y_pred, label='Predicted', mar
ker='^')
    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
    ax.set_zlabel('Target')
    ax.legend()
    plt.show()
    return rf

rf_height=get_model(data)
print(rf_height.predict([[30,30]]))
# 保存模型
dump(rf_height, 'height_random_forest_model.pkl')
```

(2).梯度

```
import numpy as np
from scipy.interpolate import interp2d
import pandas as pd

path=r'C:\Users\PC\Desktop\高度.xlsx'
df=pd.read_excel(path)
Z=np.array(df.iloc[0:][0:],dtype="float64")

import numpy as np
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from joblib import dump, load

def compute_normalized_gradient(array):
    m, n = array.shape

    gradient_x = np.zeros((m, n))
    gradient_y = np.zeros((m, n))

    normalized_gradient_x = np.zeros((m, n))
    normalized_gradient_y = np.zeros((m, n))

    for i in range(m):
        for j in range(n):
            # 计算 x 方向的梯度
            if j == 0:
                gradient_x[i, j] = array[i, j + 1] - array[i, j]
            elif j == n - 1:
                gradient_x[i, j] = array[i, j] - array[i, j - 1]
            else:
                gradient_x[i, j] = (array[i, j + 1] - array[i, j - 1]) /
2.0

            # 计算 y 方向的梯度
            if i == 0:
                gradient_y[i, j] = array[i + 1, j] - array[i, j]
            elif i == m - 1:
                gradient_y[i, j] = array[i, j] - array[i - 1, j]
            else:
                gradient_y[i, j] = (array[i + 1, j] - array[i - 1, j]) /
2.0

            # 计算梯度的模
            magnitude = np.sqrt(gradient_x[i, j] ** 2 + gradient_y[i, j]
** 2)

            # 进行单位化，考虑到除数可能为 0 的情况
            if magnitude == 0:
                normalized_gradient_x[i, j] = 0
```

```
                normalized_gradient_y[i, j] = 0
            else:
                normalized_gradient_x[i, j] = gradient_x[i, j] / magnitud
e
                normalized_gradient_y[i, j] = gradient_y[i, j] / magnitud
e

    return normalized_gradient_x, normalized_gradient_y

def compute_gradient_angle(normalized_gradient_x, normalized_gradient_y):

    m, n = normalized_gradient_x.shape
    gradient_angle = np.zeros((m, n))

    for i in range(m):
        for j in range(n):
            gradient_angle[i, j] = np.arctan2(normalized_gradient_y[i, j],
 normalized_gradient_x[i, j])

    return gradient_angle*180/np.pi


# 测试函数
array = Z
gradient_x, gradient_y = compute_normalized_gradient(array)
print("Gradient in x direction:\n", gradient_x.shape)
print("Gradient in y direction:\n", gradient_y)


x=np.linspace(0,4*1852,201)
y=np.linspace(0,5*1852,251)
print(x)
print(x.shape)
print(y.shape)
print(Z.shape)
gx=[]
for j in range(len(y)):
    for i in range(len(x)):
        t=[x[i],y[j],gradient_x[j][i]]
        gx.append(t)
gx=np.array(gx)
gy=[]
for j in range(len(y)):
    for i in range(len(x)):
        t=[x[i],y[j],gradient_y[j][i]]
        gy.append(t)
gy=np.array(gy)
def get_model(data):
    X = data[:, 0:2]
    y = data[:, 2]

    # 数据分割
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # 创建随机森林模型
    rf = RandomForestRegressor(n_estimators=100, random_state=42)

    # 训练模型
    rf.fit(X_train, y_train.ravel())

    # 预测
    y_pred = rf.predict(X_test)

    # 评估模型
    score = rf.score(X_test, y_test)
    print(f"R^2 Score: {score}")

    # 绘图展示预测结果和实际结果
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(X_test[:, 0], X_test[:, 1], y_test, label='True')
    ax.scatter(X_test[:, 0], X_test[:, 1], y_pred, label='Predicted', marker='^')
    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
    ax.set_zlabel('Target')
    ax.legend()
    plt.show()
    return rf

# rf_gx=get_model(gx)
# print(rf_gx.predict([[30,30]]))
# # 保存模型
# dump(rf_gx, 'gx_random_forest_model.pkl')
rf_gy=get_model(gy)
print(rf_gy.predict([[30,30]]))
# 保存模型
dump(rf_gy, 'gy_random_forest_model.pkl')


# # 利用之前已经计算的单位化梯度来计算角度
# gradient_angle = compute_gradient_angle(gradient_x, gradient_y)
# print("Gradient Angle:\n", gradient_angle)
# gradient_angle=gradient_angle+90
# path=r'C:\Users\PC\Desktop\角度.xlsx'
# pd.DataFrame(gradient_angle).to_excel(path)
```

(3).插值

```
import numpy as np
from scipy.interpolate import interp2d
import pandas as pd

path=r'C:\Users\PC\Desktop\1.xlsx'
```

```
df=pd.read_excel(path)
Z=np.array(df.iloc[0:][0:],dtype="float64")

# 原始二维数组
original_array = Z

# 原始数组的行列数
original_rows, original_cols = original_array.shape

# 扩充后的行列数
expanded_rows, expanded_cols = 2510, 2010

# 创建行列索引
x = np.arange(original_cols)
y = np.arange(original_rows)

# 创建插值函数
interp_func = interp2d(x, y, original_array, kind='linear')

# 创建扩充后的行列索引
expanded_x = np.linspace(0, original_cols - 1, expanded_cols)
expanded_y = np.linspace(0, original_rows - 1, expanded_rows)

# 进行插值
expanded_array = interp_func(expanded_x, expanded_y)

print(expanded_array)

path=r'C:\Users\PC\Desktop\插值.xlsx'
pd.DataFrame(expanded_array).to_excel(path)
```

(4).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from joblib import dump, load

# 加载模型
height_rf = load('height_random_forest_model.pkl')
gx_rf = load('gx_random_forest_model.pkl')
gy_rf = load('gy_random_forest_model.pkl')
# print(height_rf.predict([[30,30]]))
# print(gy_rf.predict([[30,30]]))
# print(gx_rf.predict([[30,30]]))

def get_height(x,y):
    return float(height_rf.predict([[x,y]]))
```

```
def get_gx(x,y):
    return float(gx_rf.predict([[x,y]]))
def get_gy(x,y):
    return float(gy_rf.predict([[x,y]]))
def get_alpha(x,y):
    step=0.01
    tx1 = x + step*get_gx(x,y)
    ty1 = y + step*get_gy(x,y)
    h1=get_height(tx1,ty1)
    tx2 = x - step * get_gx(x, y)
    ty2 = y - step * get_gy(x, y)
    h2 = get_height(tx2, ty2)
    return float(np.arctan((abs(h1-h2))/(2*step))*180/np.pi)
def sin(a):
    return np.sin(np.radians(a))
def cos(a):
    return np.cos(np.radians(a))
def tan(a):
    return np.tan(np.radians(a))
def get_Wleft(x,y):
    D=get_height(x,y)
    alpha=get_alpha(x,y)
    return (D*sin(theta/2)/sin(90-theta/2+alpha))*cos(alpha)
def get_WRight(x,y):
    D = get_height(x, y)
    alpha = get_alpha(x, y)
    return (D*sin(theta/2)/sin(90-theta/2-alpha))*cos(alpha)
def forward_dirction(gx,gy):
    return (-gy,gx)


print(get_height(7000,1000))
print(get_alpha(7000,1000))
line=[]
loc_x=4200
loc_y=10
step=100
theta=120
for _ in range(1000):
    gx=get_gx(loc_x,loc_y)
    gy=get_gy(loc_x,loc_y)
    wl=get_Wleft(loc_x,loc_y)
    wr=get_WRight(loc_x,loc_y)
    lx=loc_x-wl*gx
    ly=loc_y-wr*gy
    rx = loc_x + wl * gx
    ry = loc_y + wr * gy
    dx,dy=forward_dirction(gx,gy)
    loc_x+=step*dx
    loc_y+=step*dy
    # plt.scatter(loc_x,loc_y,color='r',s=1)
```

```
    # plt.scatter(lx, ly, color='b', s=1)
    # plt.scatter(rx, ry, color='b', s=1)
    # plt.pause(0.1)
    if(loc_x>4*1852 or loc_y>5*1852 or loc_y<0 or loc_x<0):
        break
    line.append([loc_x,loc_y])
line=np.array(line)

# plt.show()
# plt.clf()
plt.plot(line[:,0],line[:,1])
n=0.1
for i in line:
    x=i[0]
    y=i[1]
    while True:
        alpha=get_alpha(x,y)
        h=get_height(x,y)
        if alpha == 0:
            d=2*h*tan(theta/2)*(1-n)
            tx = d * get_gx(x, y)
            ty = d * get_gy(x, y)
            x = x+tx
            y = y+ty
        else:
            A = sin(90 - theta / 2 + alpha)
            B = sin(90 - theta / 2 - alpha)
            C = sin(theta / 2) / A - 1 / tan(alpha)
            D = n * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B
 - 1 / tan(alpha)
            next_h = h * C / D
            tx = (h - next_h) * get_gx(x, y)
            ty = (h - next_h) * get_gy(x, y)
            x = x + tx
            y = y + ty
        if (x > 4 * 1852 or y > 5 * 1852 or y < 0 or x < 0 or get_height(
x,y)<23):
            break
        plt.scatter(x, y, color='b', s=1)
    plt.pause(0.001)
for i in line:
    x=i[0]
    y=i[1]
    while True:
        alpha=get_alpha(x,y)
        h=get_height(x,y)
        if alpha == 0:
            d=2*h*tan(theta/2)*(1-n)
            tx = d * get_gx(x, y)
            ty = d * get_gy(x, y)
            x = x-tx
            y = y-ty
```

```
        else:
            A = sin(90 - theta / 2 + alpha)
            B = sin(90 - theta / 2 - alpha)
            C = sin(theta / 2) / A - 1 / tan(alpha)
            D = n * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B
 - 1 / tan(alpha)
            next_h = h * C / D
            tx = (h - next_h) * get_gx(x, y)
            ty = (h - next_h) * get_gy(x, y)
            x = x - tx
            y = y - ty
        if (x > 4 * 1852 or y > 5 * 1852 or y < 0 or x < 0 or get_height(
x,y)<21):
            break
        plt.scatter(x, y, color='b', s=1)
    plt.pause(0.001)


plt.show()
```

(5).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei'] #显示中文

path=r'C:\Users\PC\Desktop\附件.xlsx'

df=pd.read_excel(path)
print(df.head())
x=np.array(df.iloc[0][2:],dtype="float64")

y=[]
for i in range(1,df.shape[0]):
    y.append(df.iloc[i][1])

y=np.array(y,dtype="float64")

x=x*1852
y=y*1852

x, y = np.meshgrid(x, y)
path=r'C:\Users\PC\Desktop\高度.xlsx'
df=pd.read_excel(path)

Z=np.array(df.iloc[0:][0:],dtype="float64")
print(Z)
Z=200-Z
# 创建三维图形对象和坐标轴
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
32.
33. # 绘制三维图形
34. ax.plot_surface(x, y, Z, cmap='viridis')
35.
36. # 设置坐标轴标签
37. ax.set_xlabel('X')
38. ax.set_ylabel('Y')
39. ax.set_zlabel('Z')
40.
41. # 显示图形
42. plt.show()
```

(6).

```
1.  import pandas as pd
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.  import numpy as np
5.  import matplotlib.pyplot as plt
6.  from mpl_toolkits.mplot3d import Axes3D
7.  from sklearn.ensemble import RandomForestRegressor
8.  from sklearn.model_selection import train_test_split
9.  from joblib import dump, load
10. import copy
11.
12. # 加载模型
13. height_rf = load('height_random_forest_model.pkl')
14. gx_rf = load('gx_random_forest_model.pkl')
15. gy_rf = load('gy_random_forest_model.pkl')
16. # print(height_rf.predict([[30,30]]))
17. # print(gy_rf.predict([[30,30]]))
18. # print(gx_rf.predict([[30,30]]))
19.
20. def get_height(x,y):
21.     return float(height_rf.predict([[x,y]]))
22. def get_gx(x,y):
23.     return float(gx_rf.predict([[x,y]]))
24. def get_gy(x,y):
25.     return float(gy_rf.predict([[x,y]]))
26. def get_alpha(x,y):
27.     step=0.0001
28.     tx1 = x + step*get_gx(x,y)
29.     ty1 = y + step*get_gy(x,y)
30.     h1=get_height(tx1,ty1)
31.     tx2 = x - step * get_gx(x, y)
32.     ty2 = y - step * get_gy(x, y)
33.     h2 = get_height(tx2, ty2)
34.     return float(np.arctan((abs(h1-h2))/(2*step))*180/np.pi)
35. def sin(a):
36.     return np.sin(np.radians(a))
37. def cos(a):
38.     return np.cos(np.radians(a))
39. def tan(a):
```

```
    return np.tan(np.radians(a))
def get_Wleft(x,y):
    D=get_height(x,y)
    alpha=get_alpha(x,y)
    return (D*sin(theta/2)/sin(90-theta/2+alpha))*cos(alpha)
def get_WRight(x,y):
    D = get_height(x, y)
    alpha = get_alpha(x, y)
    return (D*sin(theta/2)/sin(90-theta/2-alpha))*cos(alpha)
def forward_dirction(gx,gy):
    return (-gy,gx)
def figure_lenth(line):
    sum=0
    for i in range(len(line)-1):
        # print(line[i][0],line[i][1])
        # print(i,np.sqrt((line[i][0]-line[i+1][0])**2+(line[i][1]-line[i
+1][1])**2))
        sum+=np.sqrt((line[i][0]-line[i+1][0])**2+(line[i][1]-line[i+1][1]
)**2)
    return sum
def figure_width(line):
    sum = 0
    for i in range(len(line) - 1):
        sum += np.sqrt((line[i][0] - line[i + 1][0]) ** 2 + (line[i][1] -
 line[i + 1][1]) ** 2)*(get_WRight(line[i][0],line[i][1])+get_Wleft(line[i][0],li
ne[i][1]))
    return sum
dot=np.array([[0,0]])
n=0.1
length=[[],[],[],[],[]]
width=[[],[],[],[],[]]

line=[]
loc_x=1700
loc_y=10
step=50
theta=120
while True:
    gx=get_gx(loc_x,loc_y)
    gy=get_gy(loc_x,loc_y)
    wl=get_Wleft(loc_x,loc_y)
    wr=get_WRight(loc_x,loc_y)
    lx=loc_x-wl*gx
    ly=loc_y-wr*gy
    rx = loc_x + wl * gx
    ry = loc_y + wr * gy
    dx,dy=forward_dirction(gx,gy)
    loc_x+=step*dx
    loc_y+=step*dy
    if(loc_x>4*1852 or loc_y>5*1852 or loc_y<0 or loc_x<0):
        break
    line.append([loc_x,loc_y])
```

```
line=np.array(line)
plt.plot(line[:-1,0],line[:-1,1],color='b')
while True:
    flag=0
    tl=[]
    for index,i in enumerate(line):
        x = i[0]
        y = i[1]
        alpha=get_alpha(x,y)
        h=get_height(x,y)
        if alpha <= 0.005:
            d=2*h*tan(theta)*(1-n)
            tx = d * get_gx(x, y)
            ty = d * get_gy(x, y)
            x = x+tx
            y = y+ty
        else:
            A = sin(90 - theta / 2 + alpha)
            B = sin(90 - theta / 2 - alpha)
            C = sin(theta / 2) / A - 1 / tan(alpha)
            D = n * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B
- 1 / tan(alpha)
            next_h = h * C / D
            tx = (h - next_h) * get_gx(x, y)
            ty = (h - next_h) * get_gy(x, y)
            x = x+tx
            y = y+ty
        if (y < 0 or x < 0 or get_height(x,y)<21 or get_height(x,y)>39):

            flag=1
        else:
            tl.append([x, y])
    if len(line)>5:
        plt.plot(line[:-1, 0], line[:-1, 1], color='lightgray')
        length[0].append(figure_lenth(line))
        width[0].append(figure_width(line))
    dot = np.concatenate((dot, line),axis=0)
    plt.pause(0.1)
    if len(tl)==0:
        break
    line=copy.deepcopy(tl)
    loc_x=line[-1][0]
    loc_y=line[-1][1]
    step=25
    while True:
        gx = get_gx(loc_x, loc_y)
        gy = get_gy(loc_x, loc_y)
        wl = get_Wleft(loc_x, loc_y)
        wr = get_WRight(loc_x, loc_y)
        lx = loc_x - wl * gx
        ly = loc_y - wr * gy
        rx = loc_x + wl * gx
```

```
        ry = loc_y + wr * gy
        dx, dy = forward_dirction(gx, gy)
        loc_x += step * dx
        loc_y += step * dy
        if (loc_x > 4 * 1852 or loc_y > 5 * 1852 or loc_y < 0 or loc_x < 
0):
            break
        line.append([loc_x, loc_y])
    line=np.array(line)



loc_x=3000
loc_y=10
line=[]
step=50
theta=120
while True:
    gx=get_gx(loc_x,loc_y)
    gy=get_gy(loc_x,loc_y)
    wl=get_Wleft(loc_x,loc_y)
    wr=get_WRight(loc_x,loc_y)
    lx=loc_x-wl*gx
    ly=loc_y-wr*gy
    rx = loc_x + wl * gx
    ry = loc_y + wr * gy
    dx,dy=forward_dirction(gx,gy)
    loc_x+=step*dx
    loc_y+=step*dy
    if(loc_x>4*1852 or loc_y>5*1852 or loc_y<0 or loc_x<0):
        break
    line.append([loc_x,loc_y])
line=np.array(line)
len1=figure_lenth(line)
plt.plot(line[:-1,0],line[:-1,1],color='r')
while True:
    flag=0
    tl=[]
    for index,i in enumerate(line):
        x = i[0]
        y = i[1]
        alpha=get_alpha(x,y)
        h=get_height(x,y)
        if alpha <= 0.005:
            d=2*h*tan(theta)*(1-n)
            tx = d * get_gx(x, y)
            ty = d * get_gy(x, y)
            x = x-tx
            y = y-ty
        else:
            A = sin(90 - theta / 2 + alpha)
```

```
            B = sin(90 - theta / 2 - alpha)
            C = sin(theta / 2) / A - 1 / tan(alpha)
            D = n * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B
 - 1 / tan(alpha)
            next_h = h * C / D
            tx = (h - next_h) * get_gx(x, y)
            ty = (h - next_h) * get_gy(x, y)
            x = x-tx
            y = y-ty
        if (x > 4 * 1852 or y > 8100 or y < 0 or x < 0 or get_height(x,y)
>75 or 2500*y-9260*(x-400)<-37040000):
            flag=1
        else:
            tl.append([x, y])
    if len(line)>10:
        plt.plot(line[:-1, 0], line[:-1, 1], color='silver')
        length[1].append(figure_lenth(line))
        width[1].append(figure_width(line))
    dot = np.concatenate((dot, line),axis=0)
    plt.pause(0.1)
    if len(tl)==0:
        break
    line=copy.deepcopy(tl)
    step=50
    loc_x=line[-1][0]
    loc_y=line[-1][1]
    while True:
        gx = get_gx(loc_x, loc_y)
        gy = get_gy(loc_x, loc_y)
        wl = get_Wleft(loc_x, loc_y)
        wr = get_WRight(loc_x, loc_y)
        lx = loc_x - wl * gx
        ly = loc_y - wr * gy
        rx = loc_x + wl * gx
        ry = loc_y + wr * gy
        dx, dy = forward_dirction(gx, gy)
        loc_x += step * dx
        loc_y += step * dy
        if (loc_x > 4 * 1852 or loc_y >8000 or loc_y < 0 or loc_x < 0):
            break
        line.append([loc_x, loc_y])
    line=np.array(line)



line=[]
loc_x=4500
loc_y=10
step=50
theta=120
while True:
```

```
    gx=get_gx(loc_x,loc_y)
    gy=get_gy(loc_x,loc_y)
    wl=get_Wleft(loc_x,loc_y)
    wr=get_WRight(loc_x,loc_y)
    lx=loc_x-wl*gx
    ly=loc_y-wr*gy
    rx = loc_x + wl * gx
    ry = loc_y + wr * gy
    dx,dy=forward_dirction(gx,gy)
    loc_x+=step*dx
    loc_y+=step*dy
    if(loc_x>4*1852 or loc_y>5*1852 or loc_y<0 or loc_x<0):
        break
    line.append([loc_x,loc_y])
line=np.array(line)
len2=figure_lenth(line)
plt.plot(line[:-1,0],line[:-1,1],color='g')
while True:
    flag=0
    tl=[]
    for index,i in enumerate(line):
        x = i[0]
        y = i[1]
        alpha=get_alpha(x,y)
        h=get_height(x,y)
        if alpha <= 0.005:
            d=2*h*tan(theta)*(1-n)
            tx = d * get_gx(x, y)
            ty = d * get_gy(x, y)
            x = x-tx
            y = y-ty
        else:
            A = sin(90 - theta / 2 + alpha)
            B = sin(90 - theta / 2 - alpha)
            C = sin(theta / 2) / A - 1 / tan(alpha)
            D = n * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B
 - 1 / tan(alpha)
            next_h = h * C / D
            tx = (h - next_h) * get_gx(x, y)
            ty = (h - next_h) * get_gy(x, y)
            x = x-tx
            y = y-ty
        if (x > 4 * 1852 or y > 5 * 1852 or y < 0 or x < 0 or 2500*y-9260
*x>-37040000):
            flag=1
        else:
            tl.append([x, y])

    plt.plot(line[:-1,0],line[:-1,1],color='darkgray')
    dot = np.concatenate((dot, line),axis=0)
    length[2].append(figure_lenth(line))
    width[2].append(figure_width(line))
```

```
    plt.pause(0.1)
    line=copy.deepcopy(tl)
    line=np.array(line)
    if len(tl)==0:
        break


line=[]
loc_x=6500
loc_y=9000
step=50
theta=120
while True:
    gx=get_gx(loc_x,loc_y)
    gy=get_gy(loc_x,loc_y)
    wl=get_Wleft(loc_x,loc_y)
    wr=get_WRight(loc_x,loc_y)
    lx=loc_x-wl*gx
    ly=loc_y-wr*gy
    rx = loc_x + wl * gx
    ry = loc_y + wr * gy
    dx,dy=forward_dirction(gx,gy)
    loc_x+=step*dx
    loc_y+=step*dy
    if(loc_x>4*1852 or loc_y>5*1852 or loc_y<0 or loc_x<0):
        break
    line.append([loc_x,loc_y])
line=np.array(line)
plt.plot(line[:-1,0],line[:-1,1],color='c')
while True:
    flag=0
    tl=[]
    for index,i in enumerate(line):
        x = i[0]
        y = i[1]
        alpha=get_alpha(x,y)
        h=get_height(x,y)
        if alpha <= 0.005:
            d=2*h*tan(theta)*(1-n)
            tx = d * get_gx(x, y)
            ty = d * get_gy(x, y)
            x = x-tx
            y = y-ty
        else:
            A = sin(90 - theta / 2 + alpha)
            B = sin(90 - theta / 2 - alpha)
            C = sin(theta / 2) / A - 1 / tan(alpha)
            D = n * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B
- 1 / tan(alpha)
            next_h = h * C / D
            tx = (h - next_h) * get_gx(x, y)
```

```
            ty = (h - next_h) * get_gy(x, y)
            x = x-tx
            y = y-ty
        if (x > 4 * 1852 or y > 5 * 1852 or y < 0 or x < 0 or 4408*y+2260
*x<47598080 or get_height(x,y)>75):
            flag=1
        else:
            tl.append([x, y])
    if len(line)>20:
        plt.plot(line[:-1, 0], line[:-1, 1], color='darkgrey')
        length[3].append(figure_lenth(line))
        width[3].append(figure_width(line))
    dot = np.concatenate((dot, line),axis=0)
    plt.pause(0.1)
    line=copy.deepcopy(tl)
    line=np.array(line)
    if len(tl)==0:
        break


line=[]
loc_x=5700
loc_y=7900
step=50
theta=120
while True:
    gx=get_gx(loc_x,loc_y)
    gy=get_gy(loc_x,loc_y)
    wl=get_Wleft(loc_x,loc_y)
    wr=get_WRight(loc_x,loc_y)
    lx=loc_x-wl*gx
    ly=loc_y-wr*gy
    rx = loc_x + wl * gx
    ry = loc_y + wr * gy
    dx,dy=forward_dirction(gx,gy)
    loc_x+=step*dx
    loc_y+=step*dy
    if(loc_x>4*1852 or loc_y>5*1852 or loc_y<0 or loc_x<0):
        break
    line.append([loc_x,loc_y])
line=np.array(line)
len3=figure_lenth(line)
plt.plot(line[:-1,0],line[:-1,1],color='m')
while True:
    flag=0
    tl=[]
    for index,i in enumerate(line):
        x = i[0]
        y = i[1]
        alpha=get_alpha(x,y)
        h=get_height(x,y)
```

```
        if alpha <= 0.005:
            d=2*h*tan(theta)*(1-n)
            tx = d * get_gx(x, y)
            ty = d * get_gy(x, y)
            x = x-tx
            y = y-ty
        else:
            A = sin(90 - theta / 2 + alpha)
            B = sin(90 - theta / 2 - alpha)
            C = sin(theta / 2) / A - 1 / tan(alpha)
            D = n * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B
- 1 / tan(alpha)
            next_h = h * C / D
            tx = (h - next_h) * get_gx(x, y)
            ty = (h - next_h) * get_gy(x, y)
            x = x-tx
            y = y-ty
        if (x > 4 * 1852  or y > 1852*5 or x < 0):
            flag=1
        else:
            tl.append([x, y])
    if len(line)>50:
        plt.plot(line[:-1, 0], line[:-1, 1], color='gray')
        length[4].append(figure_lenth(line))
        width[4].append(figure_width(line))
    dot = np.concatenate((dot, line), axis=0)
    plt.pause(0.1)
    line=copy.deepcopy(tl)
    line=np.array(line)
    if len(tl)==0:
        break

plt.show()

for i in range(5):
    print(sum(length[i]))
    print(sum(width[i]))
    print(len(length[i]))
    print(len(width[i]))

print(len1,len2,len3)

path=r'C:\Users\PC\Desktop\dot.xlsx'
pd.DataFrame(dot).to_excel(path, index=False)
```

(7).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from joblib import dump, load

# 加载模型
height_rf = load('height_random_forest_model.pkl')
gx_rf = load('gx_random_forest_model.pkl')
gy_rf = load('gy_random_forest_model.pkl')
# print(height_rf.predict([[30,30]]))
# print(gy_rf.predict([[30,30]]))
# print(gx_rf.predict([[30,30]]))

def get_height(x,y):
    return float(height_rf.predict([[x,y]]))
def get_gx(x,y):
    return float(gx_rf.predict([[x,y]]))
def get_gy(x,y):
    return float(gy_rf.predict([[x,y]]))
def get_alpha(x,y):
    step=0.01
    tx1 = x + step*get_gx(x,y)
    ty1 = y + step*get_gy(x,y)
    h1=get_height(tx1,ty1)
    tx2 = x - step * get_gx(x, y)
    ty2 = y - step * get_gy(x, y)
    h2 = get_height(tx2, ty2)
    return float(np.arctan((abs(h1-h2))/(2*step))*180/np.pi)
def sin(a):
    return np.sin(np.radians(a))
def cos(a):
    return np.cos(np.radians(a))
def tan(a):
    return np.tan(np.radians(a))
def get_Wleft(x,y):
    D=get_height(x,y)
    alpha=get_alpha(x,y)
    return (D*sin(theta/2)/sin(90-theta/2+alpha))*cos(alpha)
def get_WRight(x,y):
    D = get_height(x, y)
    alpha = get_alpha(x, y)
    return (D*sin(theta/2)/sin(90-theta/2-alpha))*cos(alpha)
def forward_dirction(gx,gy):
    return (-gy,gx)
def figure_lenth(line):
    sum=0
    for i in range(len(line)-1):
        # print(line[i][0],line[i][1])
        # print(i,np.sqrt((line[i][0]-line[i+1][0])**2+(line[i][1]-line[i
+1][1])**2))
        sum+=np.sqrt((line[i][0]-line[i+1][0])**2+(line[i][1]-line[i+1][1]
)**2)
    return sum
```

```
def figure_width(line):
    sum = 0
    for i in range(len(line) - 1):
        sum += np.sqrt((line[i][0] - line[i + 1][0]) ** 2 + (line[i][1] - line[i + 1][1]) ** 2)*(get_WRight(line[i][0],line[i][1])+get_Wleft(line[i][0],line[i][1]))
    return sum
line=[]
loc_x=4200
loc_y=10
step=100
theta=120
for _ in range(1000):
    gx=get_gx(loc_x,loc_y)
    gy=get_gy(loc_x,loc_y)
    wl=get_Wleft(loc_x,loc_y)
    wr=get_WRight(loc_x,loc_y)
    lx=loc_x-wl*gx
    ly=loc_y-wr*gy
    rx = loc_x + wl * gx
    ry = loc_y + wr * gy
    dx,dy=forward_dirction(gx,gy)
    loc_x+=step*dx
    loc_y+=step*dy
    # plt.scatter(loc_x,loc_y,color='r',s=1)
    # plt.scatter(lx, ly, color='b', s=1)
    # plt.scatter(rx, ry, color='b', s=1)
    # plt.pause(0.1)
    if(loc_x>4*1852 or loc_y>5*1852 or loc_y<0 or loc_x<0):
        break
    line.append([loc_x,loc_y])
line=np.array(line)

# plt.show()
# plt.clf()
plt.plot(line[:-1,0],line[:-1,1])
n=0.1
length=[]
width=[]
while True:
    flag=0
    for index,i in enumerate(line):
        x = i[0]
        y = i[1]
        alpha=get_alpha(x,y)
        h=get_height(x,y)
        if alpha <= 0.005:
            d=2*h*tan(theta)*(1-n)
            tx = d * get_gx(x, y)
            ty = d * get_gy(x, y)
            x = tx + x
            y = ty + y
```

```
        else:
            A = sin(90 - theta / 2 + alpha)
            B = sin(90 - theta / 2 - alpha)
            C = sin(theta / 2) / A - 1 / tan(alpha)
            D = n * sin(theta / 2) * (1 / A + 1 / B) - sin(theta / 2) / B
 - 1 / tan(alpha)
            next_h = h * C / D
            tx = (h - next_h) * get_gx(x, y)
            ty = (h - next_h) * get_gy(x, y)
            x = tx + x
            y = ty + y
        line[index][0]=x
        line[index][1]=y
        if (x > 4 * 1852 or y > 5 * 1852 or y < 0 or x < 0 or get_height(
x,y)<21):
            flag=1

    plt.scatter(line[:-1,0],line[:-1,1],color='b',s=1)
    length.append(figure_lenth(line))
    width.append(figure_width(line))
    plt.pause(0.1)
    if flag==1:
        break
plt.show()
print(length)
print(width)
```

(8).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei'] #显示中文
path=r'C:\Users\PC\Desktop\附件.xlsx'
df=pd.read_excel(path)
print(df.head())
x=np.array(df.iloc[0][2:],dtype="float64")
y=[]
for i in range(1,df.shape[0]):
    y.append(df.iloc[i][1])
y=np.array(y,dtype="float64")
x=x*1852
y=y*1852
x, y = np.meshgrid(x, y)
path=r'C:\Users\PC\Desktop\1.xlsx'
df=pd.read_excel(path)
Z=np.array(df.iloc[0:][0:],dtype="float64")
cset = plt.contourf(x,y,Z,60,cmap=plt.cm.hot)
contour = plt.contour(x,y,Z,60,colors='k')
plt.clabel(contour,fontsize=5,colors='k')
plt.colorbar(cset)
plt.show()
```

(9).

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from joblib import dump, load
# 加载模型
height_rf = load('height_random_forest_model.pkl')
gx_rf = load('gx_random_forest_model.pkl')
gy_rf = load('gy_random_forest_model.pkl')
# print(height_rf.predict([[30,30]]))
# print(gy_rf.predict([[30,30]]))
# print(gx_rf.predict([[30,30]]))
def get_height(x,y):
    return float(height_rf.predict([[x,y]]))
def get_gx(x,y):
    return float(gx_rf.predict([[x,y]]))
def get_gy(x,y):
    return float(gy_rf.predict([[x,y]]))
def get_alpha(x,y):
    step=0.01
    tx1 = x + step*get_gx(x,y)
    ty1 = y + step*get_gy(x,y)
    h1=get_height(tx1,ty1)
    tx2 = x - step * get_gx(x, y)
    ty2 = y - step * get_gy(x, y)
    h2 = get_height(tx2, ty2)
    return np.arctan((abs(h1-h2))/(2*step))*180/np.pi
def sin(a):
    return np.sin(np.radians(a))
def cos(a):
    return np.cos(np.radians(a))
def tan(a):
    return np.tan(np.radians(a))
def get_Wleft(x,y):
    D=get_height(x,y)
    alpha=get_alpha(x,y)
    return (D*sin(theta/2)/sin(90-theta/2+alpha))*cos(alpha)
def get_WRight(x,y):
    D = get_height(x, y)
    alpha = get_alpha(x, y)
    return (D*sin(theta/2)/sin(90-theta/2-alpha))*cos(alpha)
def forward_dirction(gx,gy):
    return (-gy,gx)

print(get_height(7000,1000))
print(get_alpha(7000,1000))
```

```
x=np.linspace(0,4*1852,201)
y=np.linspace(0,5*1852,251)
#
# alpha=[]
# for j in range(len(y)):
#     for i in range(len(x)):
#         print(get_alpha(x[i],y[j]))
#         alpha.append(get_alpha(x[i],y[j]))
# alpha=np.array(alpha).reshape([251,201])
# print(alpha)
loc_x=3000
loc_y=10
step=10
theta=120
for _ in range(1000):
    gx=get_gx(loc_x,loc_y)
    gy=get_gy(loc_x,loc_y)
    wl=get_Wleft(loc_x,loc_y)
    wr=get_WRight(loc_x,loc_y)
    lx=loc_x-wl*gx
    ly=loc_y-wr*gy
    rx = loc_x + wl * gx
    ry = loc_y + wr * gy
    dx,dy=forward_dirction(gx,gy)
    loc_x+=step*dx
    loc_y+=step*dy
    plt.scatter(loc_x,loc_y,color='r',s=1)
    plt.scatter(lx, ly, color='b', s=1)
    plt.scatter(rx, ry, color='b', s=1)
    plt.pause(0.1)
    if(loc_x>4*1852 or loc_y>5*1852 or loc_y<0 or loc_x<2000):
        break
plt.show()
```