微分思想求解多波束测线问题

摘　要

多波束测深系统在与航迹垂直的平面内一次能发射出数十个乃至上百个波束，再由接收换能器接收由海底返回的声波，能获取一定宽度的全覆盖水深条带，从而比较可靠地描绘出海底地形地貌的精细特征。

同时如何合理地布置多波束测线成为了具有挑战的问题，一个好的测线布置应该满足尽可能覆盖整个待测海域，条带之间的重叠率尽可能地低，以及测线总长度尽可能地短。

运用**“微分”**的思想将多波束测深过程分割成若干个线性的矩形斜坡海域的测深过程，近似地计算多波束测深的对应指标，同时利用**贪心**算法计算出满足一定条件下的最优的航线方案。

针对问题一，根据平面几何关系，计算各测线距中心点处的距离的海底深度，再根据正弦定理，算出波束测线的覆盖宽度和对应的覆盖率。

针对问题二，根据立体几何关系，计算不同的角度的航线下，波束投影与坡面间的偏角以及航线投影与坡面的偏角，将第二问转换为对应的第一问。

针对问题三，求解当测深一定时，覆盖宽度取极大值时对应的航向角度，采取贪心算法，确立出对应的测线布置方案，将求解问题转换为**一维线段覆盖问题**，再进行**数形结合，**将几何问题转换为代数**数列求解**问题。通过分析得到，测线的总长度为 **68 海里。**

针对问题四，根据离散的单波束测深数据，先进行线性近似，将离散的海底数据转换为连续的海底数据。同时运用**“微分”思想**建模**测线微元**，采取贪心策略用测线微元按指定重叠率全覆盖得填充完整个待测海域，将问题转换为对应的**面积覆盖**问题，求解出所有的测点坐标，得到航线的贪心最优化模型。根据测点坐标，用欧氏距离近似代替实际航线长度，从而计算出总的测线长度，漏测区域占整个待测海域的百分比，以及重叠率超过 20%的测线的总长度。

关键词 微分　贪心　数列求解　覆盖策略

1 问题重述

1.1. 问题背景

单波束测深是利用声波在水中的传播特性来测量水体深度的技术。多波束测深度系统是在单波束测深系统的基础上发展起来的，能够测量出以测量船测线为轴线且具有一定宽度的全覆盖水深条带。

真实的海底地形起伏变化大，需要设计不同间距的测线来满足尽可能地覆盖整片海域同时测量条带之间的重叠率尽可能地小。

1.2. 问题简述

（1）计算多波束测深的覆盖宽度和相邻条带之间的重叠率；

（2）计算矩形海域多波束测深的覆盖宽度；

（3）求解具体矩形海域满足较低重叠率的最短测线长度；

（4）基于具体海域的测线设计。

2 符号声明

序号	符号	意义	单位
1	α	矩形海域的坡度角	度（°）
2	D	测量船距海底的深度	米（m）
3	θ	多波束转换器的开角	度（°）
4	β	测量船的航向角	度（°）
5	W	多波束测深的覆盖宽度（投影长度）	米（m）
6	k	覆盖宽度与测深的比值	1
7	k_L	左半部分的覆盖宽度与测深的比值	1
8	k_R	右半部分的覆盖宽度与测深的比值	1
9	A	待测海域的面积（投影面积）	平方米（m^2）
10	S	测线路径	米（m）
11	p	测点，即测量船在水平面上的投影点	（m，m）

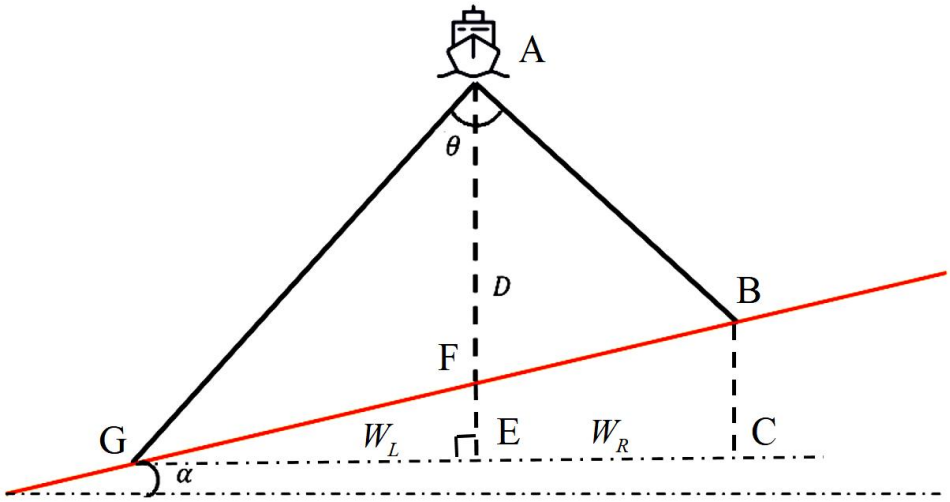
3 问题一建模与求解

3.1. 问题分析

转换为对应的几何问题，求出覆盖区域对应的长度即可。

3.2. 数学建模

3.2.1. 单条测线建模



（1）变量表示：

序号	变量	意义	对应线段
1	W_L	多波束测深左半部分的覆盖宽度	GE
2	W_R	多波束测深右半部分的覆盖宽度	EC
3	D	测量船距离海底的垂直距离	AF

（2）待求解量：

单条测线的多波束测深的覆盖宽度W

$$W = W_L + W_R \tag{3-1}$$

（3）求解结果：

利用正弦定理，即可求解出多波束测深的覆盖宽度W：

$$W_L = D[\frac{\sin(\frac{\theta}{2})\cos\alpha}{\cos(\frac{\theta}{2}+\alpha)}] \tag{3-2}$$

$$W_R = D[\frac{\sin(\frac{\theta}{2})\cos\alpha}{\cos(\frac{\theta}{2}-\alpha)}] \tag{3-3}$$

$$W = D[\frac{\sin(\frac{\theta}{2})\cos\alpha}{\cos(\frac{\theta}{2}+\alpha)} + \frac{\sin(\frac{\theta}{2})\cos\alpha}{\cos(\frac{\theta}{2}-\alpha)}] \tag{3-4}$$

（4）系数定义：

根据上述的计算求解结果，为了简化下面的公式表达，现定义相关的比例系数：

$$k_L = \frac{W_L}{D_L}, \quad k_L = [\frac{\sin(\frac{\theta}{2})\cos\alpha}{\cos(\frac{\theta}{2}+\alpha)}] \tag{3-5}$$

（左半部分覆盖宽度与测深的比例系数k_L）

$$k_R = \frac{W_R}{D_R}, \quad k_R = [\frac{\sin(\frac{\theta}{2})\cos\alpha}{\cos(\frac{\theta}{2}-\alpha)}] \tag{3-6}$$

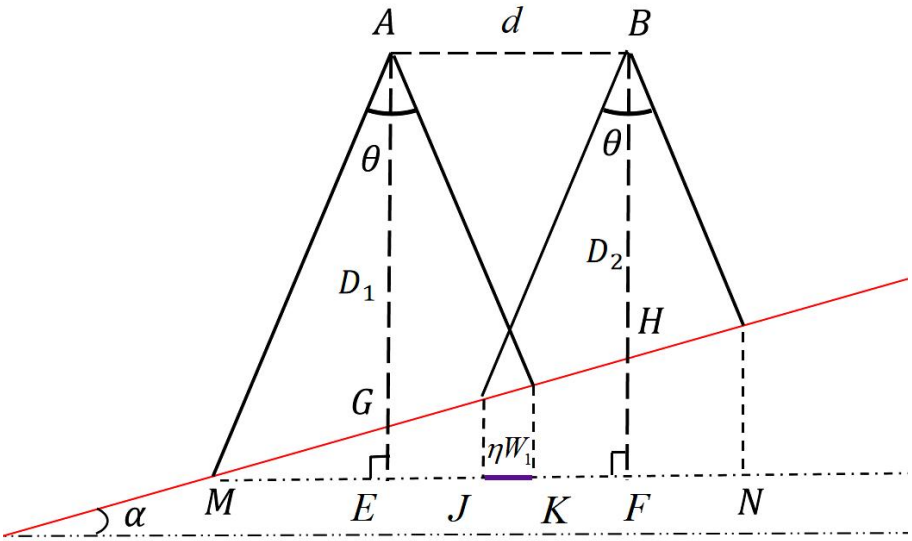
（右半部分覆盖宽度与测深的比例系数k_R）

$$k = \frac{W}{D}, \quad k = [\frac{\sin(\frac{\theta}{2})\cos\alpha}{\cos(\frac{\theta}{2}+\alpha)} + \frac{\sin(\frac{\theta}{2})\cos\alpha}{\cos(\frac{\theta}{2}-\alpha)}] \tag{3-7}$$

（覆盖宽度与测深的比例系数k）

由比例系数的定义可知，在给定多波束转换器的开角θ和矩形海域坡面的坡度角α时，比例系数为常数。

3.2.2. 多条测线建模



（1）变量表示：

序号	变量	意义	对应线段
1	W_1	上条测线的多波束覆盖宽度	MK
2	W_{1L}	上条测线的多波束左半部分覆盖宽度	ME
3	W_{1R}	上条测线的多波束右半部分覆盖宽度	EK
4	D_1	上条测线的测深	AG
5	W_2	当前测线的多波束覆盖宽度	JN
6	W_{2L}	当前测线的多波束左半部分覆盖宽度	JF
7	W_{2R}	当前测线的多波束右半部分覆盖宽度	FN
8	D_2	当前测线的测深	BH
9	d	两条测线的间距	AB

（2）待求解量：

与上一条测线的覆盖率η：

以两条测线的总的覆盖宽度为中间量，可建立如下方程：

$$D_1k + D_2k - \eta W_1 = D_1k_L + d + D_2k_R \tag{3-8}$$

（3）求解结果：

$$\eta = \frac{D_1 k + D_2 k - (D_1 k_L + d + D_2 k_R)}{W_1} \tag{3-9}$$

3.3. 结果数据

代入题目一中的具体数值，可求解出不同测深处多波束的覆盖宽度和与上一条测线的重叠率：

$$\begin{cases} \alpha = 1.5^{\circ} \\ \theta = 120^{\circ} \\ D_n = D_{n-1} + d\tan\alpha \\ D_5 = 70 \end{cases}$$

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	70.00	64.76	59.53	54.29	49.05
覆盖宽度/m	315.71	297.52	279.35	261.17	242.99	224.81	206.63	188.45	170.27
与前一条测线的重叠率/%	--	33.64	29.59	25.00	19.78	13.78	6.81	-1.39	-11.17

注：重叠率为**负值**表示条带之间有漏测。论文中数据保留到小数点后两位，更精确的结果见附件result1.xlsx 。

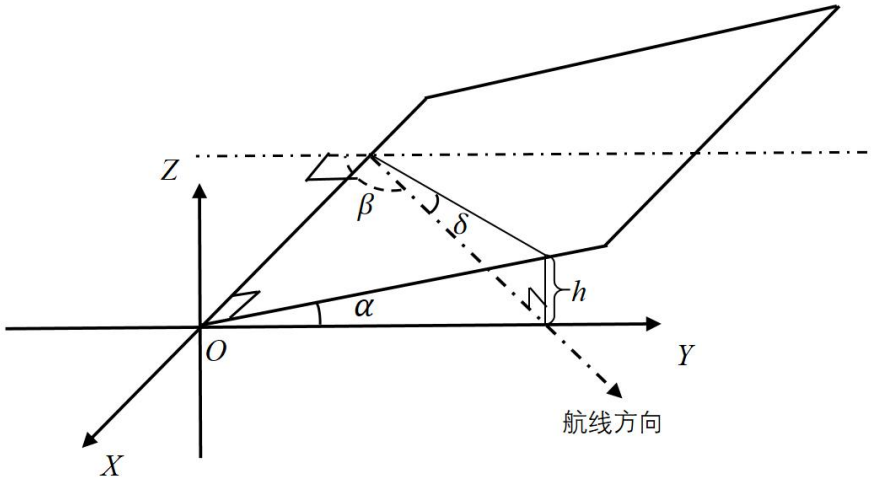
4 问题二建模与求解

4.1. 问题分析

思路继承自第一问，只需将空间几何问题转换为对应的平面几何问题。

4.2. 数学建模

4.2.1. 上升坡面建模



（1）待求解量：

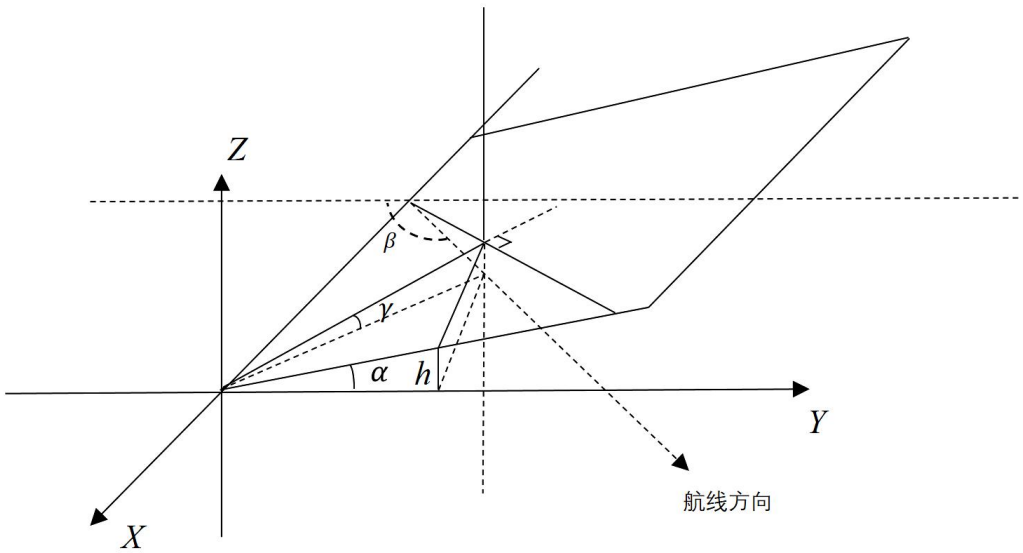
航行坡面对应的坡度角δ，满足如下方程：

$$\frac{h}{\tan\delta}=\frac{h}{\tan\alpha\sin(\beta-\frac{\pi}{2})} \tag{4-1}$$

（2）求解结果：

$$\delta=\arctan(-\cos\beta\tan\alpha) \tag{4-2}$$

4.2.2. 多波束扫描面建模



（1）待求解量：

多波束所在平面的坡度角γ，满足如下方程：

$$\frac{h}{\tan\alpha\cos(\beta-\frac{\pi}{2})}=\frac{h}{\tan\gamma} \tag{4-3}$$

（2）求解结果：

$$\gamma=\arctan(\sin\beta\tan\alpha) \tag{4-4}$$

结合第一问的求解公式，可得：

$$W=D[\frac{1}{\cos(\frac{\theta}{2}+\gamma)}+\frac{1}{\cos(\frac{\theta}{2}-\gamma)}]\sin(\frac{\theta}{2})\cos\gamma \tag{4-5}$$

4.3. 结果数据

代入题目中的具体数值，即可求解出对应航行角度和测深下的多波束覆盖宽度。

$$\begin{cases}\alpha=1.5^{\circ},\quad \theta=120^{\circ}\\ \delta=\arctan(-\cos\beta\tan\alpha)\\ D_n=D_{n-1}+d\tan\delta\\ D_1=120\\ \gamma=\arctan(\sin\beta\tan\alpha)\\ W=D[\frac{1}{\cos(\frac{\theta}{2}+\gamma)}+\frac{1}{\cos(\frac{\theta}{2}-\gamma)}]\sin(\frac{\theta}{2})\cos\gamma\end{cases}$$

覆盖宽度/m		测量船距海域中心处的距离/海里							
		0	0.3	0.6	0.9	1.2	1.5	1.8	2.1
测线方向夹角/°	0	415.69	466.09	516.49	566.89	617.29	667.69	718.09	768.48
	45	416.12	451.79	487.47	523.14	558.82	594.49	630.16	665.84
	90	416.55	416.55	416.55	416.55	416.55	416.55	416.55	416.55
	135	416.12	380.45	344.77	309.10	273.42	237.75	202.08	166.40
	180	415.69	365.29	314.89	264.50	214.10	163.70	113.30	62.90
	225	416.12	380.45	344.77	309.10	273.42	237.75	202.08	166.40
	270	416.55	416.55	416.55	416.55	416.55	416.55	416.55	416.55
	315	416.12	451.79	487.47	523.14	558.81	594.49	630.16	665.84

注：论文中数据保留到小数点后两位，更精确的结果见附件result2.xlsx 。

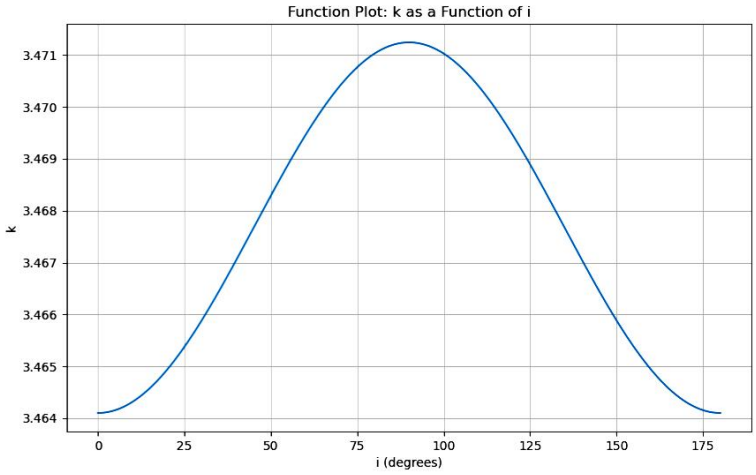
5 问题三建模与求解

5.1. 问题分析

在第二问的求解结果基础上，定义覆盖宽度与测深的比值$k=\frac{W}{D}$，即：

$$\begin{cases} k=[\dfrac{1}{\cos(\frac{\theta}{2}+\gamma)}+\dfrac{1}{\cos(\frac{\theta}{2}-\gamma)}]\sin(\frac{\theta}{2})\cos\gamma \\ \gamma=\arctan(\sin\beta\tan\alpha) \end{cases} \tag{5-1}$$

由k随航线角β的变化函数图像可知，当$\beta=90^{\circ}$时，取值范围为$[0^{\circ},180^{\circ})$，k取到极大值。



同时，在无漏测的情况下，任何一种测线方案均满足如下的方程：

$$A=\int_{S}Wds-\int_{S}\eta Wds \tag{5-2}$$

其中各个变量的含义如下：

序号	变量	含义
1	A	待测海域的面积
2	S	测线路径
3	ds	路径积分微元
4	W	该路径点对应的覆盖宽度
5	η	测量条带与下一条测量条带的重叠率

方程的含义：待测海域面积＝测量总的覆盖面积－重叠面积 。

同时，$W = Dk$，其中D测深只与测线路径S（具体地，是当前测点坐标(S_x, S_y)）有关，在α，θ角度给定的情况下，k只与航向角β有关（5-1），即：

$$W = f(S_x, S_y, \beta) \tag{5-3}$$

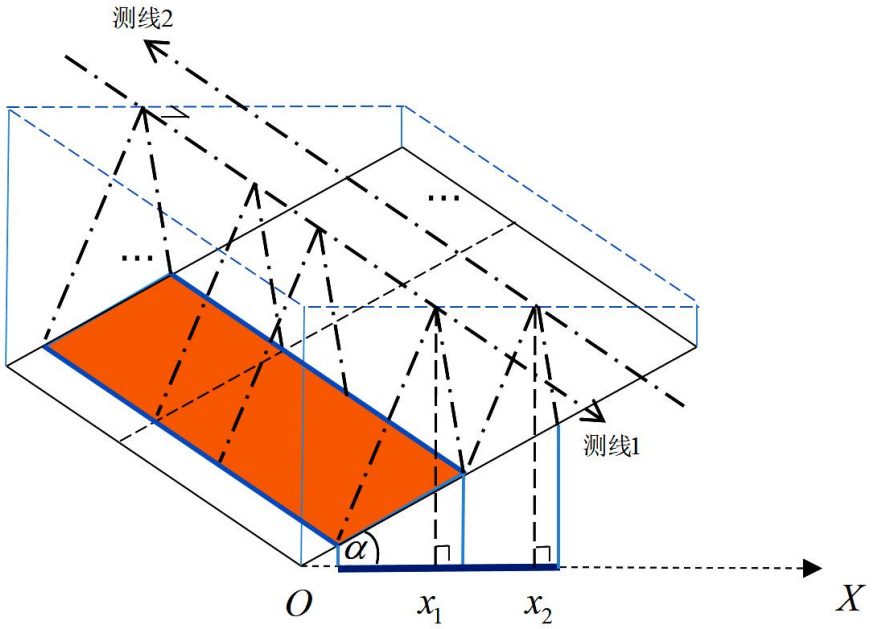
$$A = \int_S (1-\eta) f(S_x, S_y, \beta) ds \tag{5-4}$$

由（5-1）对应的函数图像可知，对于任意给定的测点坐标(S_x, S_y)：

$$f_{\max}(S_x, S_y, \beta) = f(S_x, S_y, 90^\circ) \tag{5-5}$$

故要使测线路径最短，测线需要与待测海域坡面的法线在水平面上的投影垂直，同时控制重叠率为最低。

5.2. 数学建模



5.2.1. 模型设计

（1）测线方案：

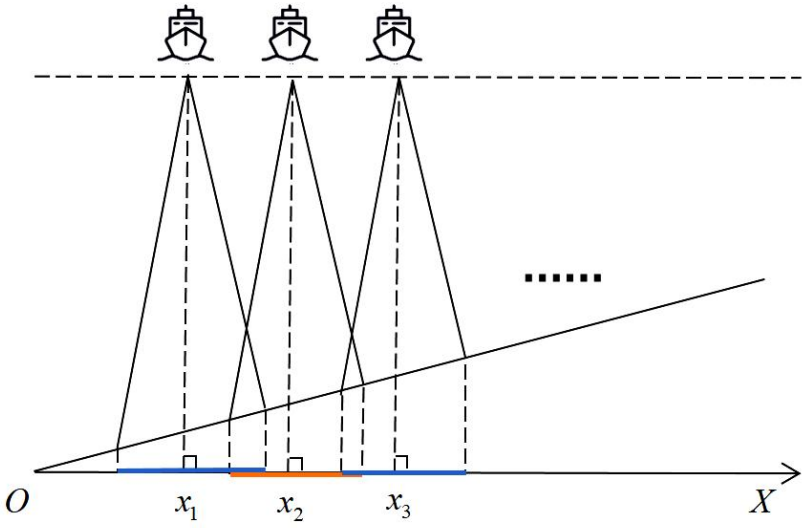
每条测线均垂直于待测海域法线在水平面上的投影；

控制每一条测线与上一条测线的覆盖宽度的重叠率最小。

（2）降维处理：

由于每一条测线的方向均与待测海域法线在水平面上的投影垂直，故测线上每一点

的测深 D 相同，即同一条测线上的每一点的覆盖宽度都相同，形成的条带为矩形，则可以将二维的面积覆盖问题转换为一维的线段覆盖问题。



（3）问题转换：

以海域的西边界与南边界的交线为 X 轴，上述最优测线方案的求解即可转换为寻找一个合适的数列，其中数列中的每个值表示测线的 X 轴坐标点。

5.2.2. 数学求解

（1）变量定义：

序号	变量	含义
1	x_i	第 i 条测线的 X 轴坐标点
2	r	测深随 x 变化的斜率
3	h	坐标原点对应的测深
4	L	待测海域的东西长度（垂直距离）

其中 D, x_i, r, h 四个变量（ D 为测深）满足如下关系：

$$D = h - rx_i \tag{5-6}$$

（2）待求解量：

满足既能全覆盖整个线段，同时每个覆盖宽度的重叠率尽可能地小的合适的测线 X 轴坐标数列 x 。

（3）求解过程：

第一条测线要满足覆盖线段的左边界刚好和原点重合，根据公式（3-2）和（5-6），得：

$$(h-rx_1)k_L=x_1 \tag{5-7}$$

之后的每一条测线与上一条测线的覆盖宽度重叠率要等于设定的值η，根据公式（3-8）和（5-6），得：

$$(h-rx_{n-1})k_L+(h-rx_n)k_R+x_n-x_{n-1}=(h-rx_{n-1})k+(h-rx_n)k-\eta(h-rx_{n-1})k \tag{5-8}$$

最后一条测线要满足覆盖线段的右边界要超过待测海域的东边界，根据公式（3-2）和（5-6），得：

$$(h-rx_n)k_R>L-x_n \tag{5-9}$$

求解可得：

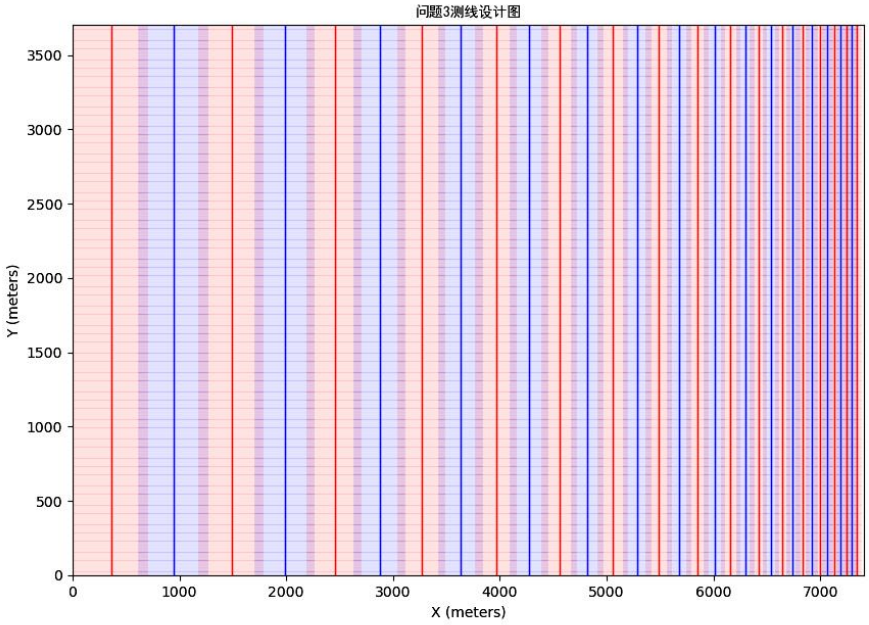
$$\begin{cases} x_1=\dfrac{hk_L}{1+rk_L}, \\ x_n=\dfrac{rk-1-rk_L-\eta rk}{rk_R-1-rk}x_{n-1}+\dfrac{h(k_L+k_R)+(\eta-2)hk}{rk_R-1-rk}, \\ x_n\geq\dfrac{L-hk_R}{1-rk_R} \end{cases}$$

5.3. 结果数据

代入题目中具体的数值：

$$\begin{cases} r=\tan 1.5^\circ, \quad L=7408, \\ h=110+\dfrac{rL}{2}, \\ \eta=10\% \end{cases}$$

求解结果：



（1）共有 34 条平行于待测海域南北边界的测线；

（2）测线的总长度为 68 海里，即 125936m；

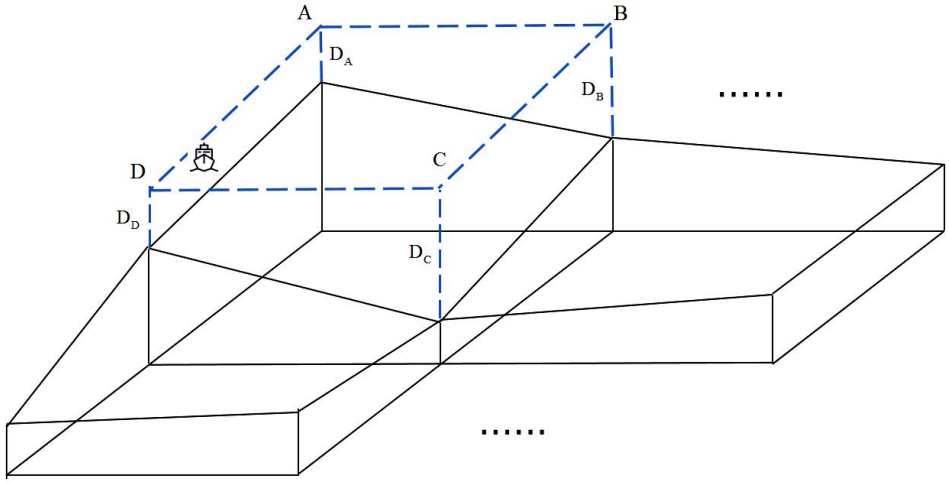
（3）具体的测线坐标数据见附件 result3.xlsx 。

6 问题四建模与求解

6.1. 问题分析

依然可以沿用解决前面问题的思路，根据提供的单波束测量的测深数据，可以将整个海域划分为若干个离散化的网络，其中每个网格点的测深已知。

将对应海域的真实海底进行线性近似建模，即将海底看成由每个网格的四个端点所对应的测深构成的六面体。如下图所示：

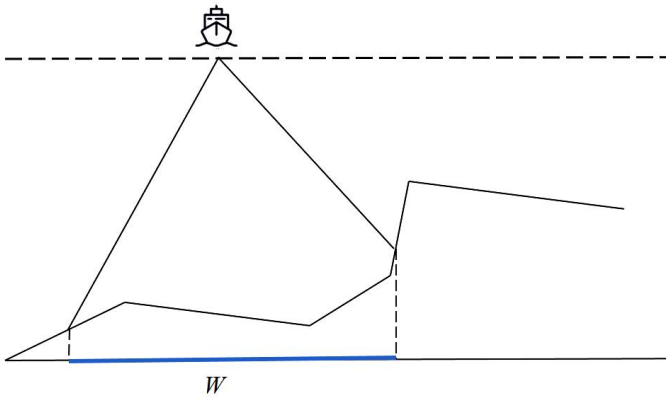


其中，A B C D 为网格点，D_A D_B D_C D_D 为对应网点处单波束的测深。

这样将待测海域建模成由若干个六面体（底面是正方形）组成的几何图形。在此基础上进行求解即可。

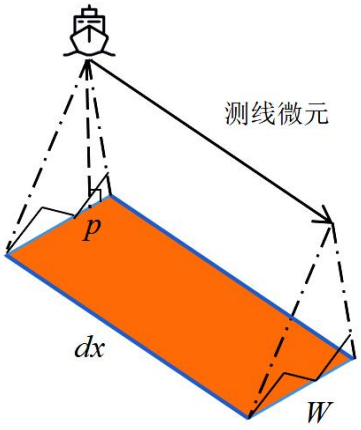
6.2. 数学建模

6.2.1. 模型设计



对于测量船只所在的测量点，根据上述建立的由离散变为连续的海底模型，通过**模拟**可以近似计算出给定多波束转换器开角θ下该处测量点的覆盖宽度W。模拟的方法是，通过程序实现声波向海底传播的过程，直到声波与海底相交，得到交点的坐标，于是就可以计算覆盖宽度。

利用**“微分”**的思想，我们取一个较小的位移微元dx，假设测量船在该位移内的覆盖宽度均为位移起始点处的覆盖宽度W，则可建立测线微元：

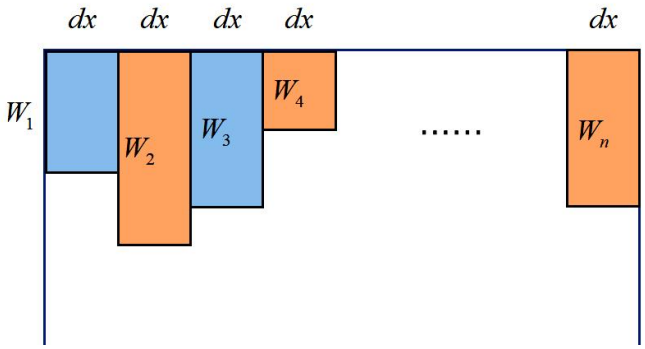


其中 p 点为测量船垂直于水平面的投影点，简称“测点”。这样，测线方案设计问题即可转换为**面积覆盖问题**。

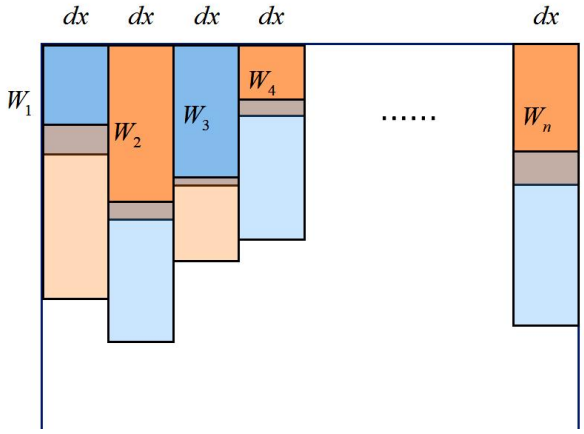
6.2.2. 算法设计

采用**贪心算法**，在满足尽可能覆盖整个待测海域，各个条带之间的重叠率尽可能小于 20%的条件下，尽可能地使测线总长度最短。

设定重叠率 η，计算出第一层的覆盖微元，使得刚好沿上边界覆盖：



然后根据重叠率 η 和上一层的覆盖微元坐标计算下一层的覆盖微元：



其中上下两层对应的覆盖微元的重叠率刚好等于设定的η。如此递推下去，即可求出所有能够覆盖整个待测海域且条带之间的覆盖率满足指定η的尽可能短的测线方案。

6.2.3. 数学求解

用 python 进行编程实现，求解出测线微元的总个数n，同时进行如下假设：

（1）用相邻两个测点p之间的欧氏距离近似代替实际的航行距离；

（2）不考虑测量船转向产生的多余的椭圆测量条带。

则相应指标对应结果如下：

变量	含义	公式
S	测线的总长度	$S=\sum_{i=2}^{n}\sqrt{(p_{i_x}-p_{(i-1)_x})^2+(p_{i_y}-p_{(i-1)_y})^2}$
--	漏测区域占整个海域的百分比	0%
--	重叠率超过 20%的测线长度	0 m

6.3. 结果数据

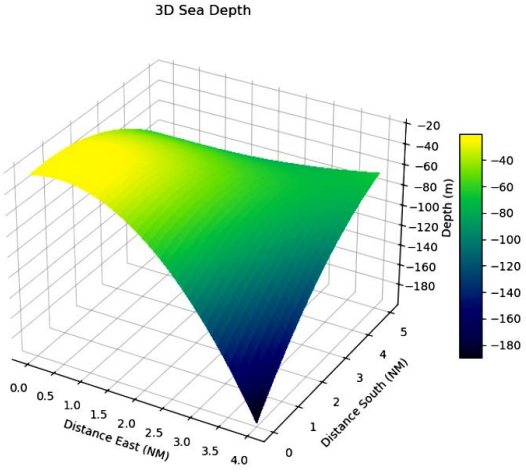


图 待测海域的近似模型图

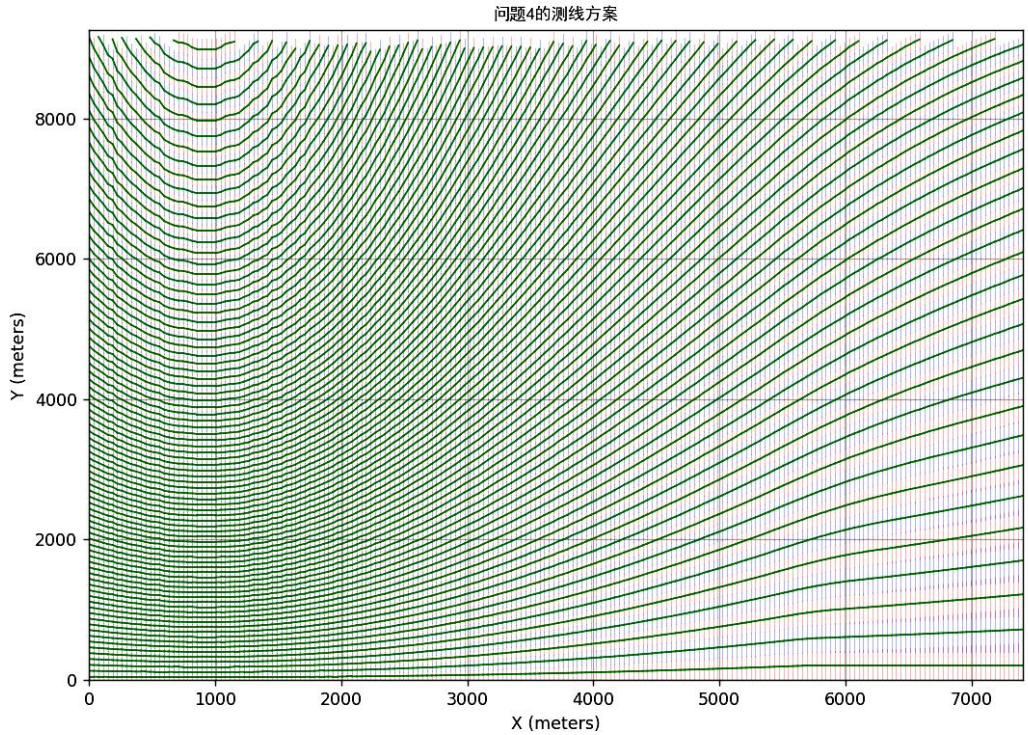


图 问题 4 的测线方案示意图

变量	含义	求解结果
S	测线的总长度	614554 m（≈332 NM）
--	漏测区域占整个海域的百分比	0%
--	重叠率超过 20%的测线长度	0 m

注：具体的测点坐标见附件 result4.xlsx 。

7 模型评价与推广

7.1. 模型优点

（1） 微分法方便我们找到海域中的极限条件，从而方便我们进行最优化分析。结合梯度的数据以及海域的建模数据，并依据贪心算法的策略，可以找出理论下的优解。

（2） 微分法与线性近似拟合相结合进行组合分析，可以探究航线中的各变量与海域深度之间的关系。更利于对航线进行拟合，从而得到更优化的航行线与航行角度。

7.2. 模型缺点

（1） 基于贪心策略与梯度分析下的航线模型下，虽然贪心策略下可以得到近似最优解，但并非严谨的数学推导，所以无法从数学层面完整证明出模型为最优解。含有一定的主观性。

（2） 在进行海域深度建模时，方式为线性填充，忽略了海底的实际情况。从而导致结果偏离现实。

（3） 在使用微分法和线性拟合对海域进行分析，并规划航线时，由于用梯度的向量性来决定航线的方向，所以航线可能存在一定的偏差与曲线型式，非完全的直线。

7.3. 模型推广

（1） 本文问题三中主要运用了贪心策略来规划航线最优模型。贪心策略的加入，有助于从局部最优解，来导向整体最优解。这样便可以规划出一条相对优良的航线。

（2） 本文问题四中主要应用了微分和线性近似来构建最优化模型。通过该模型我们构建了海底深度与波束测线之间的关联性，便于我们进行优化时，更深入的分析各变量对测绘所构成的影响。同时，该模型还可以在 tsp、无人机侦察等路径规划领域发挥更大的作用，

8 参考文献

[1]周志华.机器学习[M].清华大学出版社,2016.
[2]秋叶拓哉,岩田阳一,北川宜稔.挑战程序设计竞赛.人民邮电出版社,2013.
[3]胡大伟,朱志强,胡勇.车辆路径问题的模拟退火算法[J].中国公路学报,2006.
[4]刘保华,丁继胜,裴彦良等.海洋地球物理探测技术及其在近海工程中的应用[J].海洋科学进展,2005.
[5]冯琦,周德云.基于微分进化算法的时间最优路径规划[J].计算机工程与应用,2005.

附录

1. 模型计算代码

（1）第一、二、三问：

```
# encoding: utf-8

import math



class TestLine:
    def __init__(self, distance_to_center):
        super()
        # 距离中心的距离
        self.distance_to_center = distance_to_center
        # 海水深度
        self.height = 0
        # 覆盖宽度系数
        self.k = 0
        # 覆盖宽度
        self.w = 0



"""
------------------------------math tools------------------------------
"""



def tan(angle_degrees):
    # 将角度转换为弧度
    angle_radians = math.radians(angle_degrees)

    # 计算正切值
```

```
    tan_value = math.tan(angle_radians)

    return tan_value


def cos(angle_degrees):
    # 将角度转换为弧度
    angle_radians = math.radians(angle_degrees)

    # 计算正切值
    tan_value = math.cos(angle_radians)

    return tan_value


def K(theta_degrees, alpha_degrees):
    # 将角度转换为弧度
    theta = math.radians(theta_degrees)
    alpha = math.radians(alpha_degrees)

    # 计算公式中的各部分
    part1 = 1 / math.cos(theta / 2 + alpha)
    part2 = 1 / math.cos(theta / 2 - alpha)
    part3 = math.sin(theta / 2)

    # 计算 k 的值
    res = (part1 + part2) * part3 * cos(alpha_degrees)

    return res
```

```
def get_gamma(beta_degrees, alpha_degrees):
    """
    计算波束所在的平面对应的坡角
    :param beta_degrees: 航向角
    :param alpha_degrees: 矩形海域的坡度角
    :return: 计算出的坡角
    """
    # 将角度转换为弧度
    beta = math.radians(beta_degrees)
    alpha = math.radians(alpha_degrees)

    # 计算公式的值
    gamma = math.atan(-math.sin(beta) * math.tan(alpha))

    # 将弧度转换为度数
    gamma_degrees = math.degrees(gamma)

    return gamma_degrees


def get_delta(beta_degrees, alpha_degrees):
    """
    计算测量船前进的坡度角
    :param beta_degrees: 航向角
    :param alpha_degrees: 矩形海域的坡度角
    :return: 计算出的对应的角度
    """
    # 将角度转换为弧度
    beta = math.radians(beta_degrees)
```

```
    alpha = math.radians(alpha_degrees)

    # 计算公式的值
    delta = math.atan(-math.cos(beta) * math.tan(alpha))

    # 将弧度转换为度数
    delta_degrees = math.degrees(delta)

    return delta_degrees


def higher_k(theta_degrees, alpha_degrees):
    # 将角度转换为弧度
    theta = math.radians(theta_degrees)
    alpha = math.radians(alpha_degrees)

    # 计算公式的值
    result = math.sin(theta / 2) / math.cos(theta / 2 + alpha) * cos(alpha_degrees)

    return result


def lower_k(theta_degrees, alpha_degrees):
    # 将角度转换为弧度
    theta = math.radians(theta_degrees)
    alpha = math.radians(alpha_degrees)

    # 计算公式的值
    result = math.sin(theta / 2) / math.cos(theta / 2 - alpha) * cos(alpha_degrees)
```

```
    return result


"""
------------------------------------------------------------------------
"""


def problem1_solution():
    test_line_list = [TestLine(i) for i in range(-800, 801, 200)]

    # 初始赋值
    test_line_list[4].height = 70

    # 计算海水深度
    for i in range(-4, 5):
        test_line_list[4 + i].height = test_line_list[4].height + (
                test_line_list[4].distance_to_center - test_line_list[4 +
i].distance_to_center) * tan(1.5)

    # 计算覆盖宽度
    for i in range(len(test_line_list)):
        test_line_list[i].k = K(120, 1.5)
        test_line_list[i].w = test_line_list[i].k * test_line_list[i].height

    overlap_ratio_list = [-1]

    # 计算与前一条覆盖宽度的重叠率
    for i in range(1, len(test_line_list)):
        t1 = test_line_list[i - 1]
```

```
        t2 = test_line_list[i]

        overlap_ratio = t1.height * t1.k + t2.height * t2.k - t1.height * higher_k(120,
1.5) - t2.height * lower_k(120,

1.5) - (
                                    t2.distance_to_center - t1.distance_to_center)

        overlap_ratio_list.append(overlap_ratio / t1.w)

    print('距离中心的距离：')
    for item in test_line_list:
        print(item.distance_to_center)

    print('海水深度：')
    for item in test_line_list:
        print(item.height)

    print('覆盖宽度：')
    for item in test_line_list:
        print(item.w)

    print('与上一条测线的重叠率：')
    for item in overlap_ratio_list:
        print(item)


def problem2_solution():
    test_line_list = [TestLine(i / 10 * 1852) for i in range(0, 22, 3)]
    # for item in test_line_list:
```

```
    #     print(item.distance_to_center)

    test_line_list[0].height = 120

    for beta_degrees in range(0, 320, 45):
        # 获取波束新的坡度角
        gamma_degrees = get_gamma(beta_degrees, 1.5)

        # 获取前进坡面的坡度角
        delta_degrees = get_delta(beta_degrees, 1.5)

        # 计算海水深度
        for i in range(1, 8):
            test_line_list[i].height = test_line_list[0].height + (
                    test_line_list[0].distance_to_center -
test_line_list[i].distance_to_center) * tan(
                delta_degrees)

        # 计算覆盖宽度
        for i in range(len(test_line_list)):
            test_line_list[i].k = K(120, gamma_degrees)
            test_line_list[i].w = test_line_list[i].k * test_line_list[i].height

        print(beta_degrees, ":")

        print("海水深度：")
        for item in test_line_list:
            print(item.height)

        for item in test_line_list:
```

```
            print(item.w, end='   ')
        print()


def problem3_solution():
    theta_degrees = 120
    alpha_degrees = 1.5
    L = 4 * 1852
    r = tan(alpha_degrees)
    h = 110 + L / 2 * r
    k = K(theta_degrees, alpha_degrees)
    k_left = higher_k(theta_degrees, alpha_degrees)
    k_right = lower_k(theta_degrees, alpha_degrees)
    eta = 0.1

    x_list = list()

    # 首项
    x = h * k_left / (1 + r * k_left)
    x_list.append(x)

    p1 = r * k_right - 1 - r * k
    p2 = r * k - 1 - r * k_left - eta * r * k
    p3 = h * (k_left + k_right) + (eta - 2) * h * k

    while True:
        x = x_list[-1]
        y = p2 / p1 * x + p3 / p1
        x_list.append(y)
        bound = (L - h * k_right) / (1 - r * k_right)
```

```
        if y >= bound:
            break

    for i in x_list:
        print(i)
        print(i - (h - r * i) * k_left, i + (h - r * i) * k_right)

    print(len(x_list))
```

（2）第三问绘制图像代码：

```
%matplotlib notebook

import pandas as pd

# 定义读取的起始和结束列

start_col = 'B'

end_col = 'e'



# 定义读取的起始和结束行

start_row = 2

end_row = 34



# 使用`usecols`参数来定义读取的列范围

cols = f"{start_col}:{end_col}"
```

```
# 读取 Excel 文件中的指定范围数据

df = pd.read_excel("Q3.xlsx", engine='openpyxl', usecols=cols, header=None,
skiprows=range(start_row - 1))


# 为了只读取到 end_row 行，需要从读取的数据中截取相应的行


cycle = end_row - start_row + 1

df = df.iloc[:cycle]


# 显示数据

print(df)

from matplotlib import pyplot as plt

import numpy as np


plt.figure(figsize=(10, 7))

# 设置标题和轴标签

plt.title('问题 3 测线设计图', fontproperties="SimHei")
```

```
plt.xlabel("X (meters)")

plt.ylabel("Y (meters)")



x_lim = 4 * 1852

y_lim = 2 * 1852

# 设置轴长度

plt.xlim(0, x_lim)

plt.ylim(0, y_lim)



val = df.values

print(val[0, 1])

for i in range(cycle):

    plt.plot([val[i, 0] , val[i, 0]], [0, y_lim]

             , color='red' if i % 2 == 0 else 'blue', linewidth=1, alpha=1)

    for j in np.linspace(0, y_lim, 200):

        plt.plot([val[i, 2], val[i, 3]], [[j, j], [j, j]],

                 color='red' if i % 2 == 0 else 'blue', linewidth=2, alpha=0.05)

    # plt.scatter(x_coords_left, y_coords_left, color='red', s=0.1)  # 散点图
```

```
    # plt.scatter(x_coords_right, y_coords_right, color='red', s=0.1)  # 散点图



# 显示图

plt.grid(False)

plt.show()
```

（3）第四问：

```
from utils import _sin, print_list, _cos, _tan, miles, three_cos_theory, center_elements
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# 定义读取的起始和结束列
start_col = 'C'
end_col = 'GU'

# 定义读取的起始和结束行
start_row = 3
end_row = 253

# 使用`usecols`参数来定义读取的列范围
cols = f"{start_col}:{end_col}"

# 读取 Excel 文件中的指定范围数据
df = pd.read_excel("附件.xlsx", engine='openpyxl', usecols=cols, header=None,
skiprows=range(start_row - 1))
```

```
# 为了只读取到 end_row 行，需要从读取的数据中截取相应的行
df = df.iloc[:end_row - start_row + 1]

# 显示数据
print(df)

# 将深度变为负值
depth_values = -df.values
print(center_elements(depth_values))



def bilinear_interpolation(x, y, values):
    """双线性插值函数。
    values: 列表，包含四个点的深度值，按左上、右上、左下、右下的顺序。
    """
    q11, q21, q12, q22 = values
    return q11 * (1 - x) * (1 - y) + q21 * x * (1 - y) + q12 * (1 - x) * y + q22 * x * y



def find_intersections(depth_matrix, center_x_meters, center_y_meters, center_z_meters,
step_size=1, cell_size=1.0):
    matrix_x = center_x_meters / cell_size
    matrix_y = center_y_meters / cell_size
    matrix_z = center_z_meters / cell_size
    max_range = max(depth_matrix.shape) * 2 * cell_size
    intersections = []
    matrix_x_size = depth_matrix.shape[1]
    matrix_y_size = depth_matrix.shape[0]
    for angle in np.linspace(30, 150, 3):
```

```
        dy = step_size * np.cos(np.radians(angle))
        dz = step_size * np.sin(np.radians(angle))

        y_offset, z = 0, matrix_z

        for _ in range(int(max_range / step_size)):
            y_offset += dy
            z -= dz
            # 在矩阵的坐标
            x_index = matrix_x
            y_index = matrix_y + y_offset / cell_size

            ix, iy = int(x_index), int(y_index)

            if ix < matrix_x_size - 1 and iy < matrix_y_size - 1:
                # 用双线性插值计算深度
                interpolated_depth = bilinear_interpolation(x_index - ix, y_index - iy,
                                                            [depth_matrix[iy, ix],
depth_matrix[iy, ix + 1],
                                                             depth_matrix[iy + 1, ix],
depth_matrix[iy + 1, ix + 1]])
                if z <= interpolated_depth:
                    intersections.append((x_index * cell_size, y_index * cell_size, z))
                    break
            elif ix < matrix_x_size - 1:
                # print('iy: ', (matrix_y_size - 1)  * cell_size)
                intersections.append((x_index * cell_size, (matrix_y_size - 1)  *
cell_size, z))
                break
            elif iy < matrix_y_size- 1:
```

```
                # print('ix: ', (matrix_x_size - 1)  * cell_size)
                intersections.append(((matrix_y_size - 1)  * cell_size, y_index *
cell_size, z))
                break
            else:
                # print('ix, iy: ', ((matrix_x_size- 1)  *cell_size, (matrix_y_size - 1)
* cell_size))
                intersections.append(((matrix_x_size - 1)  *cell_size, (matrix_y_size -
1) * cell_size, z))
                break
    return intersections


def picture_draw(intersections, center_x_meters, center_y_meters):
    # 创建 x, y 坐标网格
    x = np.linspace(0, df.shape[1] - 1, df.shape[1]) * 0.02
    y = np.linspace(0, df.shape[0] - 1, df.shape[0]) * 0.02
    x, y = np.meshgrid(x, y)

    # 创建 3D 图形
    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, projection='3d')

    # 添加交点和线段
    x_intersections = [pt[0] / 1852 for pt in intersections]
    y_intersections = [pt[1] / 1852 for pt in intersections]
    z_intersections = [pt[2] for pt in intersections]
    # 标注交点坐标
    # for xi, yi, zi in zip(x_intersections, y_intersections, z_intersections):
    #     label = '({:.2f}NM, {:.2f}NM, {:.2f}m)'.format(xi, yi, zi)
```

```
    #     ax.text(xi, yi, zi, label, color='black', zorder=6)  # Adjusting z-position
slightly higher for visibility

    print(x_intersections)
    print(y_intersections)
    print(z_intersections)
    # 先画散点图
    ax.scatter(x_intersections, y_intersections, z_intersections, color='red', s=300,
label='Intersections')
    if len(x_intersections) > 1:
        ax.plot(x_intersections, y_intersections, z_intersections, color='brown',
label='Sonar Line')
     # 绘制海底深度图
    surf = ax.plot_surface(x, y, depth_values, cmap='viridis', linewidth=0,
antialiased=False)

    # 添加颜色条
    fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10)

    # 设置标题和轴标签
    ax.set_title('3D Sea Depth')
    ax.set_xlabel('Distance East (NM)')
    ax.set_ylabel('Distance South (NM)')
    ax.set_zlabel('Depth (m)')

    # 显示图形
plt.show()

# 调用部分
cell_size = 0.02 * 1852
```

```
length_meters = 4 * miles
width_meters = 5 * miles
center_x_meters = (depth_values.shape[1] // 2) * cell_size
center_y_meters = (depth_values.shape[0] // 2) * cell_size
all_intersections = []

"""超参数"""
ita = 0.0   # 重叠率η
delta_x = 0.02  # 步长（海里）
eps = 1e-6
length_step = int(4 / delta_x)
"""""""
result_line = []
result_matrix = []

min_y = 0.0
cycle = 0
while min_y < width_meters:
# while cycle < 40:
    print(f'cycle:{cycle}')
    x_idx = 0
    intended_y_list = []
    max_y_list = []
    for x_meter in np.linspace(0, length_meters - 1, length_step):
        # print('x_idx: ', x_idx)
        old_intersects = result_matrix[-1][x_idx] if cycle > 0 else [(0, 0, 0), (0, 0, 0),
(0, 0, 0)]
        y_upper = old_intersects[0][1]   # 靠近上面的y
        y_lower = old_intersects[2][1]   # 靠近下面的y
        W_last = y_upper - y_lower # 上一行的覆盖宽度
```

```
            max_y_list.append(y_upper)
            intended_y = W_last * (1 - ita) + y_lower  # 这一条带的靠近下面的 y 所在的位置
            intended_y_list.append(intended_y)
            l, r = 0, -np.min(depth_values)
            new_intersects = None
             # 二分，找到覆盖的点
            while r - l > eps:
                mid = (l + r) / 2
                new_intersects = find_intersections(depth_matrix=depth_values,
                                                    center_x_meters=x_meter,
                                                    center_y_meters=intended_y + mid, # 靠近下面的 y 值+伸长的长度
                                                    center_z_meters=0,
                                                    cell_size=cell_size)
                new_y_lower = new_intersects[2][1]
                # print(intersects)
                if new_y_lower < intended_y:
                    l = mid
                else:
                    r = mid

            x_idx += 1
            # print(f'cycle:{cycle}, ', intersects)
            result_line.append(new_intersects)
        result_matrix.append(result_line)
        # print(result_matrix[-1][0], result_matrix[-1][-1])
        # 使用 min()函数和 lambda 函数找到最小的 y 值
        min_y = min(max_y_list)
        print("最小的 y 值:", min_y)
        result_line = []
```

```
cycle += 1


import math
def get_dist(x1, y1, x2, y2):
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
plt.figure(figsize=(10, 7))
# 设置标题和轴标签
plt.title("问题 4 的测线方案", fontproperties='SimHei')
plt.xlabel("X (meters)")
plt.ylabel("Y (meters)")

# 设置轴长度
plt.xlim(0, 4 * 1852)
plt.ylim(0, 5 * 1852)
total_length = 0
total_cover_area = 0
total_overlap_area = 0
for cyc_cnt in range(cycle):
    # 提取 x 和 y 坐标
    x_coords_left = [pt[0][0] for pt in result_matrix[cyc_cnt]]
    y_coords_left = [pt[0][1] for pt in result_matrix[cyc_cnt]]
    x_coords_mid = [pt[1][0] for pt in result_matrix[cyc_cnt]]
    y_coords_mid = [pt[1][1] for pt in result_matrix[cyc_cnt]]
    x_coords_right = [pt[2][0] for pt in result_matrix[cyc_cnt]]
    y_coords_right = [pt[2][1] for pt in result_matrix[cyc_cnt]]
    # 创建图并绘制
    print(f'cyc_cnt: {cyc_cnt}, ')
    for i in range(len(x_coords_right)):
        if y_coords_left[i] < width_meters:
            plt.plot([x_coords_left[i], x_coords_right[i]], [y_coords_left[i],
```

```
y_coords_right[i]]
                , color='red' if cyc_cnt % 2 == 0 else 'blue', linewidth=0.5, alpha=0.3)
            total_cover_area += delta_x * miles * (y_coords_left[i] - y_coords_right[i])

    for i in range(1, len(x_coords_mid)):
        if y_coords_left[i] < width_meters:
            plt.plot([x_coords_mid[i - 1], x_coords_mid[i]], [y_coords_mid[i - 1],
y_coords_mid[i]]
                     , color='green', linewidth=1, alpha=1)
            total_length += get_dist(x_coords_mid[i - 1], y_coords_mid[i - 1],
x_coords_mid[i], y_coords_mid[i])

    if cyc_cnt > 0:
        for i in range(len(y_coords_right)):
            y_upper = result_matrix[cyc_cnt - 1][i][0][1]
            y_lower = result_matrix[cyc_cnt - 1][i][2][1]
            if y_lower < width_meters and (y_coords_right[i] - y_lower) / (y_upper -
y_lower) > 0.2:
                total_overlap_area += delta_x * (y_upper - y_coords_right[i])

print('测线总长度: ', total_length)
print('测线覆盖面积: ', total_cover_area)
print('待测海域面积: ', width_meters * length_meters)
print('未测百分比: {:.3f}%'.format((1 - total_cover_area / (width_meters *
length_meters) ) * 100))
print('重叠部分超过20%的总面积: ', total_overlap_area)
# plt.scatter(x_coords_left, y_coords_left, color='red', s=0.1)  # 散点图
# plt.scatter(x_coords_right, y_coords_right, color='red', s=0.1)  # 散点图

# 显示图
```

```
plt.grid(True)
plt.show()
```

2. 支撑材料内容

文件类	文件名	主要功能/用途
源代码	Q_1_2_3.py	问题 1、2、3 的 python 程序代码
	Q_3_figure.ipynb	问题 3 图像绘制 jupyter notebook 代码
	Q_4.py	问题 4 的 python 程序代码
数据	result1.xlsx	问题 1 的结果数据
	result2.xlsx	问题 2 的结果数据
	result3.xlsx	问题 3 的测线坐标数据
	result4.xlsx	问题 4 的测线微元的坐标数据