基于贪心算法的多波束测量测线布设优化

摘要

多波束测深系统是一种测量水体深度的技术，通过声波信号计算海水深度，为海洋的开发提供数据基础。为了使得多波束测量兼顾效率与质量，我们使用几何模型、贪心算法、深度优先搜索以求解多波束测量测线布设的最优方案，使得沿总长度尽可能短的测线扫描形成的条带尽可能地覆盖整个待测海域，同时控制相邻条带之间的重叠率在一定范围内。

针对问题一，我们首先根据海底坡度、测线距中心点的距离，通过几何关系计算不同测线对应的海水深度。再进一步计算扫描形成的条带在水平方向投影得到的覆盖宽度。最后根据题中给出的重叠率的定义计算每条测线与前一条测线的重叠率，并将相应结果填入了 result1.xlsx 文件。本题以**平面几何计算**为主。

针对问题二，与问题一相比，加入了测线方向与坡面法向在水平面上的投影方向之间的夹角，可以在问题一模型的基础上建模计算。通过几何关系，我们得到了不同测线方向夹角对应沿斜面覆盖宽度方向与水平方向的夹角以及不同位置的海水深度。进而可以将问题二的模型拆分为多个问题一模型的组合，用不同测线方向夹角对应沿斜面覆盖宽度方向与水平方向的夹角代替问题一中的海底坡角，计算投影到水平面上的覆盖宽度，并将相应结果填入了 result2.xlsx 文件. 本题以**立体几何计算**为主。

针对问题三，需要测线长度最短、能够完全覆盖整个海域且相邻条带之间的重叠率在 10%~20%的范围内。我们提出当**测线方向与等深线方向平行**时，有最优方案，并通过**圆锥曲线模型**与**问题二的模型**严格证明了在这种情况下，测线完全覆盖整个海域时总长度最短且相邻条带之间重叠率能够符合要求。接下来使用 **C++**编程求解，经过**二分法逐步优化**最终得到：重叠率取 **11.37%**时，一共有 **34** 条测线，总长度为 **125936** 米，并生成了大致示意图。

针对问题四，为了对于题目数据有更直观的认识，我们通过 **MATLAB** 编程将附件数据可视化，得到海底地形图及海水等深线图。根据这两幅图我们决定采取**“以点代面”**的思路，将模型简化，利用**贪心算法**走一步算三步，使用**深度优先搜索**探索在极大数据处理和整体面情况下探索局部最优解以期得到相对优秀的测线设计，并将设计方案以表格的方式呈现。最后计算题目要求的指标。

最后，我们分析了模型的优缺点，并针对模型的缺点提出相应的改进方式。

关键词：几何模型、贪心算法、深度优先搜索、测线优化

一、 问题重述

1.1 问题背景

海洋，占据着地球表面积的 70%以上，对人类来说是一个新鲜而陌生的领域。随着科技的不断发展，海洋资源逐渐成为人类探索和开发的重要领域。其中，海底矿产资源的开发与利用成为了人们关注的焦点之一。海底矿产资源的开发需要先进行海底地形地貌的测量。

目前海底探测有多种手段，尤其是在单波束测深基础上发展起来的多波束测深。在这一探测过程中，该系统能够在与航线垂直的平面内向各方向发出众多的波束，使用接受换能器将从海底返回的声波接收，形成以测量船测线为轴线的全覆盖水深条带。在测线布设时，测线间距过小能提高海底地形分辨率，但会增加工作量、产生冗余数据、降低工作效率；而测线间距过大会大幅度降低海底地形分辨率。因此需要进行多波束测量测线布设优化，同时考虑水深对测线条带宽度的影响，平衡测量的效率与质量。

1.2 问题提出

基于上述背景，要求建立数学模型解决如下问题：

题目中定义了测深条带的宽度、换能器的开角、水深以及相邻条带之间的重叠率，提出了标准的重叠率范围。

问题一：已知多波束换能器的开角、海底坡度、海域中心点处的海水深度，建立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并计算当测线距中心点处距离改变时，海水深度、覆盖宽度、与前一条测线的重叠率三组数据。

问题二：基于第一题的模型，当测线方向与坡面法向在水平面上的投影夹角改变时，建立多波束测深覆盖宽度的数学模型并计算测量船距海域中心点处距离不同时的覆盖宽度。

问题三：一个南北长 2 海里、东西宽 4 海里的矩形海域，已知海域中心点处的海水深度、海底坡度、多波束换能器的开角，设计一组可完全覆盖整个待测海域的测线，使得相邻条带间的重叠率在 10%~20%的范围内。

问题四：根据附件中的海水深度测量数据，为多波束测量船布设测线，要求扫描的条带尽可能覆盖整个待测海域、相邻条带之间的重叠率尽量控制在 20%一下、测线的总长度尽可能短。并计算所设计的测线的总长度、漏测海域面积占比、重叠率超标部分长度。

二、 问题分析

2.1 问题一的分析

在问题一中，已知多波束换能器的开角、海底坡度以及海域中心点处的海水深度，需要建立多波束测深的覆盖宽度与相邻条带之间重叠率的数学模型。根据已知内容，可以首先计算距中心点不同距离的海水深度；接下来用多波束换能器开角和海底坡度，算出不同测线的覆盖宽度；最后用题中给出的重叠率计算公式得出与前一条测线的重叠率。本题以平面几何计算为主。

2.2 问题二的分析

在问题二中，测线方向与坡面法向在水平面上的投影的夹角改变，是问题一的进阶。为了解决这一问题，可以将问题二拆分为在不同点的问题一模型，先根据海域中心点处的海水深度、测量船距中心点处的距离以及海底坡度计算不同位置的海水深度；再根据几何关系计算不同测线方向夹角对应覆盖宽度的方向与水平方向的夹角，将该点处转为问题一的模型，计算覆盖宽度。本题使用几何方法将模型转化为问题一的模型进行求解。

2.3 问题三的分析

在问题三中，给出了一个矩形海域，需要布设长度最短、可完全覆盖整个待测海域的测线。由于多波束换能器开角不变，当测量船在同一点但测线方向不同时，可获得波束旋转而成的圆锥，而海底坡面可视为穿过锥体的斜面，故在海底坡面上会截得一个椭圆。经几何证明当测线方向与等深线方向平行时，覆盖宽度为椭圆的长轴，且每条测线的覆盖宽度均为定值，可以控制整个范围内的重叠率较小，而待测海域的面积一定，故当测线与等深线方向平行时为最优解。在此基础上，我们通过 C++进行模拟并使用二分法逐步优化得到最优测线布设方式。

2.4 问题四的分析

在问题四中，给出了某海域水深数据，需要利用这组数据进行测量船的测量布线，使得扫描形成的条带尽可能地覆盖整个待测海域，相邻条带的重叠率尽量在 20%以下且测线的总长度尽可能短。首先使用 MATLAB 绘制海底地形图及海水等深线，进行直观分析。再利用“以点代面”的思路将题目简化，利用贪心算法走一步算三步，使用深度优先搜索探索在极大数据处理和整体面情况下探索局部最优解以期得到相对优秀的测线设计。最后通过测线走过的步数计算测线的总长度，根据未探测到的点的占比计算漏测海区的占比，根据探测次数过大的点计算重叠率超过 20%部分的总长度。

三、 模型假设

为了构建更为精确的数学模型，本文根据实际情况做出了以下合理的假设或条件约束：

- 假设一：船速、海底地势等外部条件对测深结果不产生任何影响。
- 假设二：每一次测深结果均有效且完整。
- 假设三：海平面无波浪，多波束开角平面始终垂直于海平面。
- 假设四：在无数据的两点间深度按照线性关系发生变化。

四、 符号说明

符号	说明
D_0	中心点处海水深度
D	海水深度
D'	前一条测线的海水深度
W	覆盖宽度（沿水平方向）
η	重叠率
m	重叠部分宽度
d	相邻测线间距
α	海底坡度
x	测线距中心点处的距离
θ	多波束换能器开角
γ	沿斜面覆盖宽度方向与水平方向的夹角
y	测量船距海域中心点处的距离
β	测线方向夹角
H	海里与米的换算尺度（为常数 1852）
L	测线实际覆盖海域宽度
l	海域宽度
σ	测线实际覆盖海域宽度与海域宽度的差异率

注：未申明的变量以其在符号出现处的具体说明为准

五、 模型的建立与求解

5.1 问题一：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

5.1.1 多波束测深的覆盖宽度

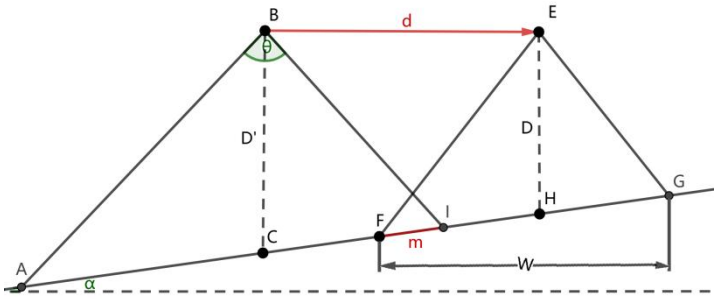

图 1

在问题一中，已知多波束换能器的开角 θ 、相邻测线间距 d、海底坡度 α，首先需要计算多波束测深的不同测线的覆盖宽度。本文先计算了每条测线的海水深度，如公式(1)所示：

$$D = D_0 - x * tan\,\alpha \tag{1}$$

其中 D 为海水深度，D_0为测线距中心点处的距离为 0 时的海水深度，x 为测线距中心点处的距离， α 为海底坡度。

接下来由几何关系，可以计算 FH、HG 的长度如下公式(2)(3)：

$$FH = \frac{D\,sin\frac{\theta}{2}}{cos\left(\frac{\theta}{2}+\alpha\right)} \tag{2}$$

$$HG = \frac{D\,sin\frac{\theta}{2}}{cos\left(\frac{\theta}{2}-\alpha\right)} \tag{3}$$

其中 θ为多波束换能器的开角。

最后由几何关系可以得到覆盖宽度 W，如公式(4)：

$$W = (FH + HG) * cos\,\alpha \tag{4}$$

其中 W 为测量船波束在坡底覆盖的宽度投影到水平线上的覆盖宽度。

5.1.2 相邻条带之间重叠率

根据题干中给出的海底平坦时计算相邻条带之间重叠率的公式：

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

可知，重叠率的定义为相邻条带重叠的部分宽度与覆盖宽度之比。故在本题中需要先计算沿坡面方向重叠部分的宽度 m，向水平方向投影后，与 5.1.1 中的覆盖宽度计算结果相除得到测线与前一条测线的重叠率。

由几何关系可得：

$$CI = \frac{D' \sin\frac{\theta}{2}}{\cos\left(\frac{\theta}{2} - \alpha\right)} \tag{2}$$

$$CH = \frac{d}{\cos\alpha} \tag{3}$$

$$m = CI + FH - CH \tag{4}$$

其中D'为前一条测线的海水深度，d 为相邻测线的间距，m 为沿坡面方向重叠部分的宽度。接下来将 m 投影至水平方向在除以水平方向的覆盖宽度 W，得到测线与前一条测线的重叠率：

$$\eta = \frac{m\cos\alpha}{W} * 100\% \tag{5}$$

代入数据计算后可得下表：

表 1　问题一计算结果

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	70.000	64.763	59.526	54.288	49.051
覆盖宽度/m	315.706	297.527	279.345	261.166	242.987	224.808	206.629	188.447	170.268
与前一条测线的重叠率/%	——	35.696	31.511	26.743	21.262	14.895	7.408	-1.525	-12.366

5.2 问题二：建立多波束测深覆盖宽度的数学模型

问题二中测线方向与坡面法向在水平面上的投影的夹角改变，而已知测线方向夹角 β、测量船距海域中心点的距离 y、多波束换能器的开角 θ、海底坡度 α 以及海域中心点的海水深度 D_0，故可以将问题二模型中每个点简化为问题一的模型，对于覆盖宽度 W 进行求解。

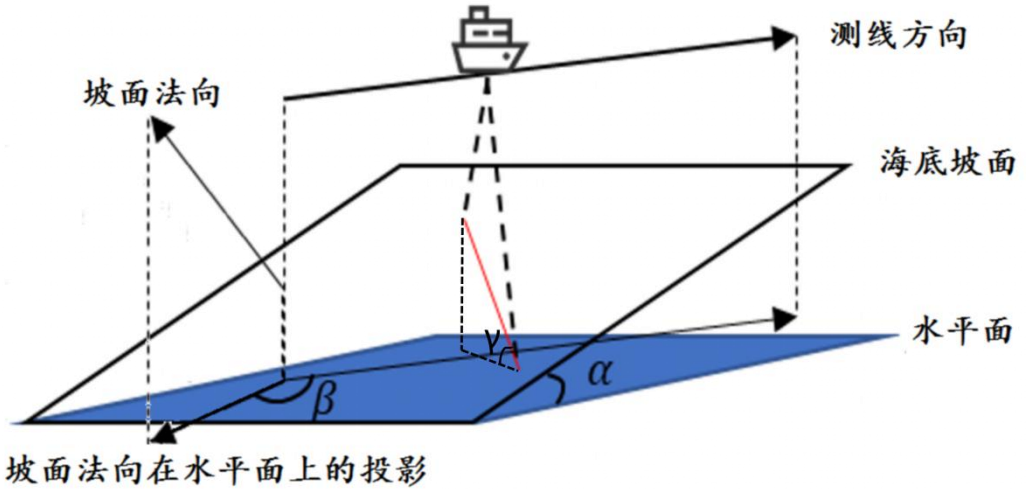

图 2

在图二中可以观察到，当测线方向与坡面法向在水平面上的投影夹角为 β 时，沿斜面覆盖宽度方向(图中红线)与水平方向的夹角 γ，可以视作测量船按照问题一模型在该点的“海底坡度”，进而使用问题一的公式计算。因此首先需要求得不同夹角对应的 γ。

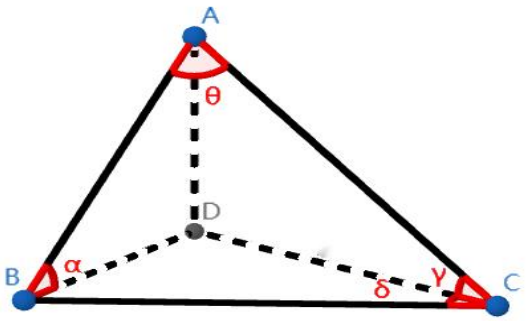

图 3

为计算 γ，将图 3 中包含 γ 的四面体从图 2 中提取出来，其中 AC 为图 2 中沿斜面的覆盖宽度，CD 为 AC 在水平方向的投影，BD 为坡面法向在水平面的投影，θ 为多波束换能器的开角，α 为海底坡度，δ 的值为 $\pi-\beta$。在图 3 中，由几何关系可得：

$$\tan\gamma = \sin(\pi - \beta)\tan\alpha \tag{1}$$

为了求得覆盖宽度 W，还需要在不同点的海水深度 D：

$$D = D_0 + H * y * \cos\beta * \tan\alpha \tag{2}$$

其中 D_0 为中心点处海水深度，H 为海里与米的换算尺度（常数 1852），y 为测量船距海域中心点处的距离。

根据以上计算结果，γ 对应问题一中的海底坡度 α，即可使用问题一的模型对问题二进行计算，公式如下：

$$W = \left(\frac{D * \sin\frac{\theta}{2}}{\cos\left(\frac{\theta}{2} + \gamma\right)} + \frac{D * \sin\frac{\theta}{2}}{\cos\left(\frac{\theta}{2} - \gamma\right)} \right) * \cos\gamma \tag{3}$$

代入数据后，可以计算得 γ、W 如下表：

表 2　γ 的取值

测线方向夹角/°	沿斜面覆盖宽度方向与水平方向的夹角/RAD
0	0
45	0.018514127
90	0.026179939
135	0.018514127
180	0
225	-0.018514127
270	-0.026179939
315	-0.018514127

表 3　覆盖宽度 W 的取值

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	667.686	718.085	768.484
	45	416.120	451.794	487.468	523.142	558.816	594.491	630.165	665.839
	90	416.549	416.549	416.549	416.549	416.549	416.549	416.549	416.549
	135	416.120	380.446	344.772	309.098	273.424	237.750	202.076	166.402
	180	415.692	365.293	314.894	264.496	214.097	163.698	113.299	62.900
	225	416.120	380.446	344.772	309.098	273.424	237.750	202.076	166.402
	270	416.549	416.549	416.549	416.549	416.549	416.549	416.549	416.549

	315	416.120	451.794	487.468	523.142	558.816	594.491	630.165	665.839

5.3 问题三：设计最优多波束测量测线布设方案

5.3.1 最优测线方向与等深线平行的证明

本题要求在给定的规则海域中找到测线布设的最优方案，要求测线长度最短、可覆盖整片海域且相邻条带重叠率在 10%~20%之间。经过文献查找，有说法为对于测线方向，通常为使扫测宽度尽可能保持不变，测线方向尽量与等深线方向保持一致。[1]

本文认为，当测深船测线的方向平行于等深线方向时，此时会存在最优解。因为其满足：

1. 测深船在平行于等深线方向时，覆盖宽度为最大值，总长度最短；
2. 在实现全海域覆盖时，重叠率会稳定在一个合适区间。

证明过程如下：

针对条件 1：当测深船舶在固定一个点位向下测深时，假设此时海域为平坦海域。当船舶的测深方向不发生变化时，在几何层面上，其多波光束及覆盖宽度在某一切面上呈现为一个固定大小、角度的三角形（如图 4）。当船舶的测线方向发生变化时，即等效于该切面三角形沿着 A 点进行旋转，最终会在空间中形成一个圆锥体（如图 2）。

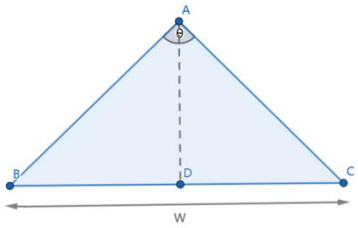

图 4

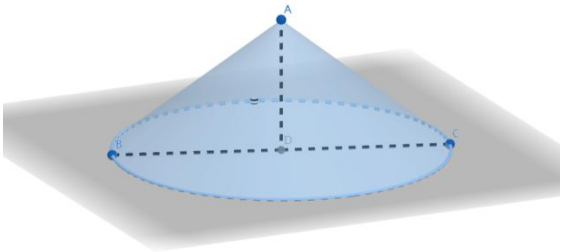

图 5

此时考虑该海域为坡度为 1.5° 的非平坦海域（如图 6 中平面 BEF），该斜

面海域与该圆锥体所截出来的几何图形为椭圆。因而对于任意一个船舶行驶的测线方向，其覆盖宽度一定等于该椭圆上过中心点的某一条弦的长度（图 7）。

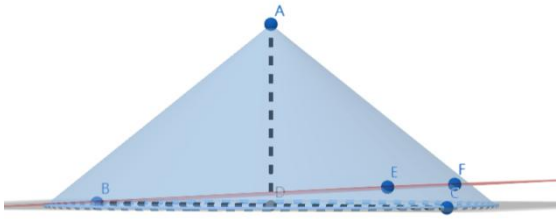

图 6

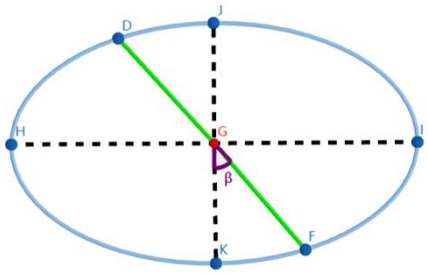

图 7

下面证明斜面海域与该圆锥体所截出来的几何图形为椭圆：

在图 8 中，斜面 ABP 为上述 1.5°海底坡面，下面证明几何图形曲线 ABP 为椭圆。

在圆锥中插入两个球体，满足与圆锥相切且与平面 ABP 相切两个条件。记与切面的切点分别为 F1,F2。则对于 ABP 平面上的任意一点 P，过该点做一条圆锥的母线分别交两个圆于点 M、点 Q；由切线长定理可知，PF1=PM、PF2=PQ。而易知 MQ=CD 且为常数，故该曲线上任意一点 P 到两点的距离之和为定值，符合椭圆定义。而其中 F1,F2 即为该椭圆的两个焦点。[2]

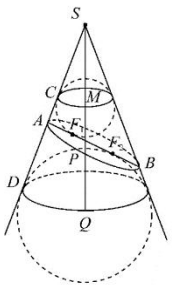

图 8

当船舶的测线方向平行于等深线方向时，其覆盖宽度线恰好为过两焦点的一条弦。即覆盖宽度为椭圆的长轴，为所有角度中最大的。当测定的海域面积一定

时，最大的覆盖宽度对应的测线总长度最短，实现最优解。

针对条件 2：已知船舶测线方向沿着等深线方向时，会有最大的覆盖宽度。并且当船舶沿着与等深线平行的方向行驶测深时，所测量的深度范围面不会随着海面地势的变化而呈现出梯形状（如图 9），相反会呈现出均匀的矩形测深面（如图 10）。因而在取一个适合的测线间距后，不会出现漏测、重叠率过高的情况，因而重叠率会满足在一个合适的区间范围之中。

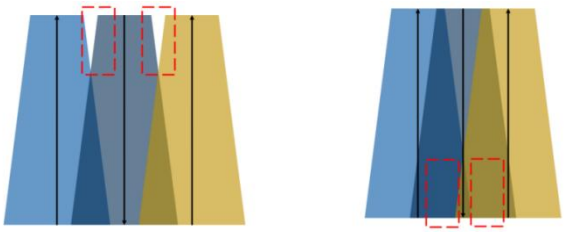

图 9

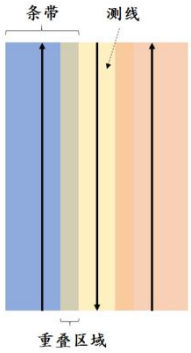

图 10

除严格的数学证明之外，沿着等深线方向进行测深在生活实际方面有如下优势：

1. 减少重叠区域：在生活中，为了保证数据的连续性和准确性，船舶的路径会有一定的重叠。如果船沿着等深线航行，这能使得每次测量覆盖的区域与上一次有最小的重叠，以最大化每次测量的有效范围。

2. 减少测量时间：沿着等深线航行意味着船在一个相对稳定的深度范围内移动。使得测量更加稳定，减少了因深浅突变导致的需要重新校准或调整设备的情况，从而减少测量时间。

3. 简化数据处理：当船沿等深线航行时，接收到的数据更为一致，那么在后期的数据处理和地图生成中，更易于处理。从长远来看，这不仅节省了测量时间，提高了数据的精度，还节省了数据处理的时间。

4. 提高航行效率：沿等深线航行可以使船的航行路径更直观，更易于规划和跟踪。这减少了导航和规划上的困难。

因此，沿着等深线航行不仅可以满足题述测线总长度最小的要求，减少实际的航行距离，从实际角度出发还可以提高测量效率，减少数据处理时间，并简化

导航和路径规划。

5.3.2 确定最优多波束测量测线布设方案

5.3.1 中证明了最优测线方向与等深线方向平行的合理性，接下来将建立模型，求得最优方案的具体信息。

在问题三的处理中，本文首先以 10%的重叠率进行初始模型建设，这样的好处是：一定会得到最短的测线长度组，并且所划定的测线条数一定为 n 或 n+1，即仅会在最后一条测线布设时出现重叠率的大程度改变，因此本文使用二分法进行对算法的逐步优化。以很小的量为基础累加至 10%的重叠率之上，逐渐增加重叠率。直到实现所覆盖的面积完全覆盖整个海域且测线总覆盖海域的面积与实际面积的差异率 σ 尽可能小于 0.001%，进而输出最终的结果。

其中差异率的定义为：

$$\sigma = \frac{L - l}{l} * 100\% \tag{1}$$

以下是算法流程图：

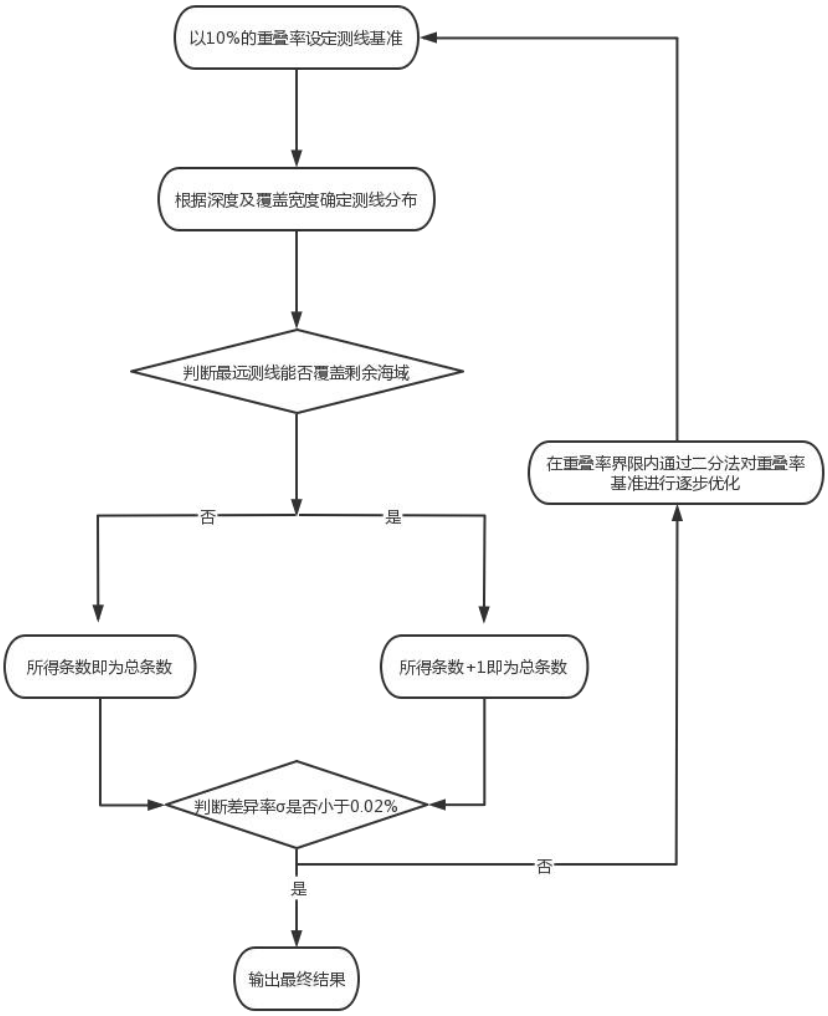

图 11

在此处将重叠率 η 进行了化简计算，最终的化解结果如下：

$$\eta = 1 - \frac{d}{D} * \frac{cos\left(\frac{\theta}{2} - \alpha\right) sin\alpha + \frac{cos\left(\frac{\theta}{2} - \alpha\right) cos\left(\frac{\theta}{2} + \alpha\right)}{sin\frac{\theta}{2}}}{2\,cos\frac{\theta}{2}(cos\alpha)^2} \tag{2}$$

将该公式以及相关常量代入求解，下面是 C++代码：

```
#include <iostream>
#include <vector>

double calculate_depth(double distance, double depth) {
    return depth - distance * 0.02618592;
}

double calculate_distance(double depth) {
    return depth * 0.8863 * 0.99965732 * 0.99965732 /
(0.52249856*0.0261769+0.522498567*0.47715876*2/1.732050807568877);
}

int main() {
    double initial_depth = (110+2*1852*0.02618592)/(2.09573853 *
1.732050807568877 / 2 * 0.02617695 + 1);
    double depth = initial_depth;
    double total_distance
=(110+2*1852*0.02618592-initial_depth)/0.02618592;
    int flag=0;
    std::vector<double> depths, distances;

    while (total_distance / 1852 < 4) {
        double dist = calculate_distance(depth);
        total_distance += dist;
        depth = calculate_depth(dist, depth);
        flag++;
        depths.push_back(depth);
```

```
            distances.push_back(dist);
        }

        for (int i = 0; i < depths.size(); i++) {
            std::cout << "Depth: " << depths[i] << ", Distance: " <<
distances[i] << std::endl;
        }
        std::cout<<total_distance<<std::endl;
        std::cout<<flag;
        return 0;
}
```

最终得到的结果为重叠率取 11.37%时，一共有 34 条测线，总长度为 34*2*1852=125936 米，测线实际覆盖海域宽度 L=7408.08，计算出差异率 σ =0.00108%，满足约束条件。下图为测线大致分布情况：

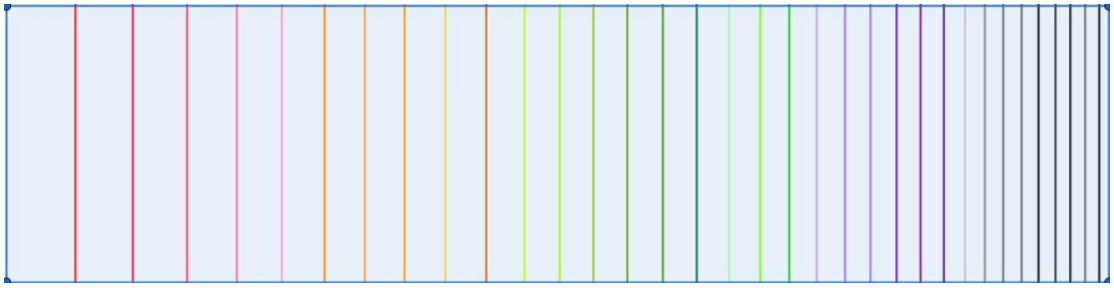

图 12

5.4 问题四：多波束测量船测量布线优化

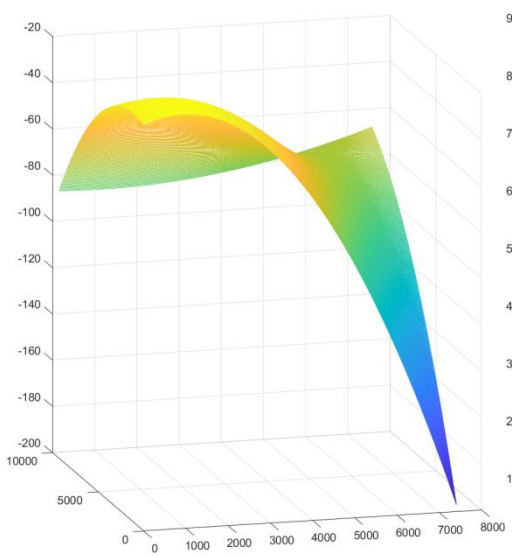

图 13

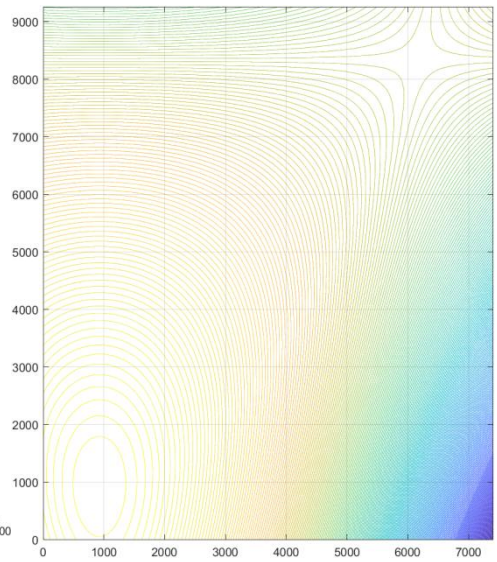

图 14

5.4.1 模型准备

对于问题四，我们首先根据海底深度数据画出了海底深度的图样（如图 13）并测绘出了等深线图样（如图 14），通过问题三可知，如果沿着平行于等深线方向进行布设测线，可以获得最优的一组测线解。通过观察等深线走向可知，若按平行于等深线方向严格布设测线，则会在某些具有复杂海势的区域（如图 15）由于等深线方向不定等原因难以布设合适角度的测线方向且具有较高重叠率的风险。因此我们提出了“以点代面”的算法思想，通过数据库中所给出的相应点坐标及其深度，用历经该点代替历经该海域面这一行为。在此基础上，我们将测线布设覆盖整个海域测深这一过程简化为多个以点为出发基础的搜索式覆盖过程，其简化示意图如下：

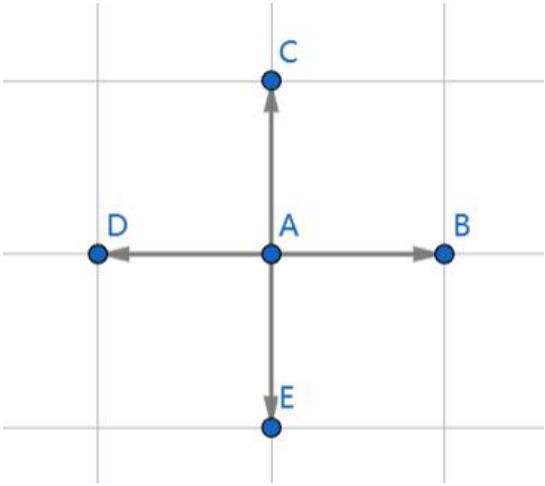

图 15

当船舶历经 A 点时，根据模型假定接下来会有 4 个方向的测线布置选择。而在 A 点处，多波束测深存在可能会扫到相应的其他点位。例如，假若处于 A 点的船舶即将像正北方向行驶，则需要判断在测量船 A 点处沿着正北方向上的测深波束是否会历经东西两侧的相关点位，这里我们利用几何建模模型提出了下列的判断依据。

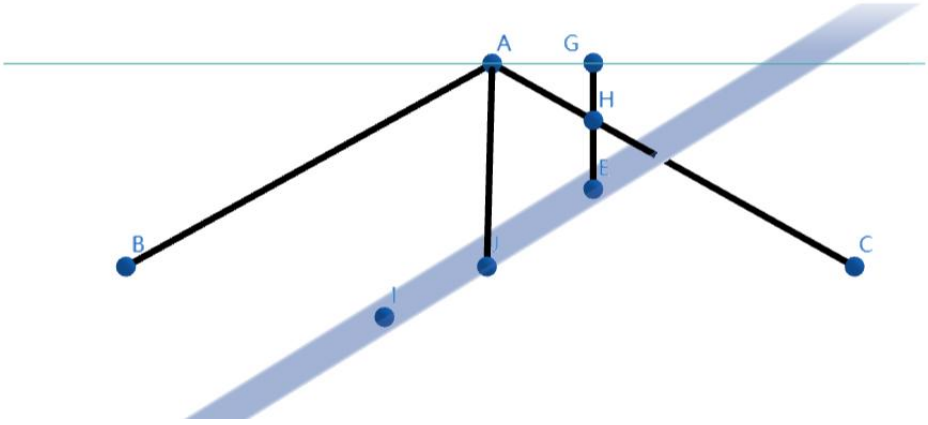

图 16

如图中所示，对于 AC 直线我们可以得出其判断深度解析式：

$$z = z(A) - \frac{tan\theta}{2} * 0.02H * i \tag{1}$$

其中 i 为被判断点 G 与 A 点的横坐标差，z(x)为相关点为所对应的深度。

当满足：

$$z \geq z(G) \tag{2}$$

时，认为船舶行驶至 A 点处时，G 点可以被检测到。

其中当船舶在某一点处发生决策变化时会面临着两种选择：

1. 方向不发生变化。
2. 方向发生变化。

当选择 1 发生时，我们对于此中心点位所覆盖的范围仅计算一次；当选择 2 发生时，我们会将该点在南北、东西两个方向上所覆盖的范围都纳入到结果计算之中。

5.4.2 编程计算

基于上述模型准备，我们建立了基于贪心算法的深度优先搜索对该问题进行解决，其核心代码思路如下：

```
double simulate(std::vector<std::vector<Point> >& points, int x, int y, int steps,int
pre)
//我们以递归的模型思想来替船舶作出决定，在走一步同时判断三步以内的点位覆盖情况
{
    if (steps == 0) {
        return computeCoverage(points);
    }
    double bestCoverage = -1.0;
    for (int dir = 0; dir < directions.size(); dir++) {
        int newX = x + directions[dir].first;
        int newY = y + directions[dir].second;
        if(newX < 0 || newY < 0 || newX >= points.size() || newY >=
points[0].size())
            continue;
```

```
            std::vector<std::pair<int, int> > changedPoints;
            cover(points, newX, newY, dir, pre, changedPoints);
//继续搜索
            double coverage = simulate(points, newX, newY, steps - 1,dir);
            if (coverage > bestCoverage)
            {
                bestCoverage = coverage;
            }
            // Undo the simulation for next iteration
            for(size_t i = 0; i < changedPoints.size(); i++) {
                int px = changedPoints[i].first;
                int py = changedPoints[i].second;
                points[px][py].v--;
            }
        }
        return bestCoverage;
}
int greedy_search(std::vector<std::vector<Point> >& points, int x, int y,int
previousdirection,std::vector<std::vector<int> >& visited) //贪心搜索完成
{
        int bestDirection = -1;
        double bestCoverage = -1.0;
        const double unvisitedBonus = 1; //  未访问区域的额外奖励
        for (int dir = 0; dir < directions.size(); dir++) {
            int newX = x + directions[dir].first;
            int newY = y + directions[dir].second;
            if(newX < 0 || newY < 0 || newX >= points.size() || newY >=
points[0].size())
                continue;
            if(previousdirection-dir==2||dir-previousdirection==2)
                continue;
            if(visited[newX][newY]>20) {
                continue; //  如果这个位置已经被访问过多次，就跳过
            }
            std::vector<std::pair<int, int> > changedPoints;
```

```
            visited[newX][newY]++;
            cover(points, newX, newY, dir, previousdirection, changedPoints);
            double coverage = simulate(points, newX, newY,2,previousdirection);
            // 如果该点未被访问或访问次数较少，为其覆盖值增加一个权重。
            if(visited[newX][newY] <= 1) {
                coverage += unvisitedBonus;
            }
            if (coverage > bestCoverage) {
                bestCoverage = coverage;
                bestDirection = dir;
            }
            // Undo the simulation for next iteration 回溯时复原影响
            for(size_t i = 0; i < changedPoints.size(); i++) {
                int px = changedPoints[i].first;
                int py = changedPoints[i].second;
                points[px][py].v--;
            }
        }
        return bestDirection;
}
```

输出结果为：Minimum distance: 10755，Coverage : 96.5848%，Coverage with v>2: 22.9639%。其中测线总长度需换算单位，实际值为 10755*37.04=398365.2 米，重叠率超过 20%的总长度为 398365.2*22.9639%=91480.2 米。

我们最终所得到的结果为：测线总长度 398365.2 米，漏测海区占总待测海域面积的百分比为 96.5848%，重叠率超过 20%部分的总长度 91480.2 米。

测线规划方案见支撑材料中的 output.txt，表示方法为从(200，250)出发，数字代表行进方向，一个数字行进一格，0，1，2，3 分别代表向上，向左，向下，向右。

六、 模型的评价

6.1 模型优点

6.1.1 通过圆锥曲线模型简化计算

在问题三的模型中，我们发现多波束换能器张角不变时，其多波光束及覆盖宽度在某一切面上呈现为一个固定大小、角度的三角形，当测线方向改变时，该三角形会旋转得到一个圆锥，而海底坡面相当于斜面截取圆锥，会在坡面上留下一个椭圆截面，而测线与等深线方向平行时，坡面上的覆盖宽度会经过椭圆的两个焦点，是其长轴。从总长度最短和控制重叠率两个要求来看，均能证明该模型的测线方向是最优方案，避免了大量的编程计算。

6.1.2 “以点代面”的思路简化模型

在问题四的模型中，我们在处理附件数据时采取“以点代面”的思路，更加贴合题目条件且简化了计算过程，能够得到细致的测线布设方案。

6.1.3 基于贪心算法的深度优先搜索

在以点代面的思想指导下，我们独立的考虑每一个点，因此采取搜索算法解题。考虑到数据量较多，如果采取广度优先搜索会导致计算时间过长，因为时间复杂度高，计算量呈指数级增长。而如果采取直接深度优先算法，会导致栈溢出。因此本题引入贪心算法，考虑局部最优解，以局部最优的结果推算整体较优的结果，以实现效率与完成度的平衡。其中在贪心算法中，采用迭代的方法，走一步看三步，得到相对更优的局部解。

6.2 模型缺点

以贪心算法探索局部最优解会带来一定程度上的盲目，而且有可能陷入死循环之中需要可以规避。这种算法得出来的结果也不会是整体最优的最短测线，只是一种可能性大解。

6.3 模型改进

问题四的模型受限于计算机的算力，只能采取深度优先搜索，以贪心算法探索局部最优解，而在现实的测线布设时，可以使用算力更加强大的计算机，利用广度优先算法找到全局最优解，大幅提高覆盖范围并降低重叠率过高的部分占比。

七、 参考文献

[1]成芳,胡迺成.多波束测量测线布设优化方法研究[J].海洋技术学报,2016,35(02):87-91.
[2]昌明.Dandelin 双球之问[J].数学通报,2018,57(02):21-24.

附　录

一、 计算结果表格

表 1 问题一计算结果

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	70.000	64.763	59.526	54.288	49.051
覆盖宽度/m	315.706	297.527	279.345	261.166	242.987	224.808	206.629	188.447	170.268
与前一条测线的重叠率/%	——	35.696	31.511	26.743	21.262	14.895	7.408	-1.525	-12.366

表 2 问题二计算结果

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	667.686	718.085	768.484
	45	416.120	451.794	487.468	523.142	558.816	594.491	630.165	665.839
	90	416.549	416.549	416.549	416.549	416.549	416.549	416.549	416.549
	135	416.120	380.446	344.772	309.098	273.424	237.750	202.076	166.402
	180	415.692	365.293	314.894	264.496	214.097	163.698	113.299	62.900
	225	416.120	380.446	344.772	309.098	273.424	237.750	202.076	166.402
	270	416.549	416.549	416.549	416.549	416.549	416.549	416.549	416.549
	315	416.120	451.794	487.468	523.142	558.816	594.491	630.165	665.839

二、 代码

问题三：C++规划代码

```
#include <iostream>
#include <vector>
double calculate_depth(double distance, double depth) {
    return depth - distance * 0.02618592;
}

double calculate_distance(double depth) {
    return depth * 0.8863 * 0.99965732 * 0.99965732 /
(0.52249856*0.0261769+0.522498567*0.47715876*2/1.732050807568877);
}

int main() {
    double initial_depth = (110+2*1852*0.02618592)/(2.09573853 *
1.732050807568877 / 2 * 0.02617695 + 1);
    double depth = initial_depth;
    double total_distance
=(110+2*1852*0.02618592-initial_depth)/0.02618592;
    int flag=0;
    std::vector<double> depths, distances;

    while (total_distance / 1852 < 4) {
        double dist = calculate_distance(depth);
        total_distance += dist;
        depth = calculate_depth(dist, depth);
        flag++;
        depths.push_back(depth);
        distances.push_back(dist);
    }

    for (int i = 0; i < depths.size(); i++) {
```

```
        std::cout << "Depth: " << depths[i] << ", Distance: " <<
distances[i] << std::endl;
    }
    std::cout<<total_distance<<std::endl;
    std::cout<<flag;
    return 0;
}
```

问题四：MATLAB 可视化代码

```
clear;
clc;
[num]=xlsread("D:\大学科研实践\数模国赛\附件(1).xlsx");
z=[num]
x=0:37.04:7408;
y0=0:37.04:9260;
y=y0';
figure(2)
subplot(1,2,1);
mesh(x,y,z)
subplot(1,2,2);
contour3(x,y,z,250);
c.LineWidth = 5;
```

问题四：C++深度优先搜索代码

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include<cmath>
#include<math.h>
```

```
#include <queue>
#include<fstream>
using namespace std;
struct State
{
    int x, y;
    int distance;
    int direction; // 添加当前方向
    std::vector<int> path;
};

long mindistance=0;
std::vector<std::pair<int, int> > directions;
void initializeDirections()
{
    directions.push_back(std::make_pair(0, -1)); // 上
    directions.push_back(std::make_pair(-1, 0)); // 左
    directions.push_back(std::make_pair(0, 1));  // 下
    directions.push_back(std::make_pair(1, 0));  // 右
}
class Point
{
public:
    double x, y, z, v;
    Point(double x_ = 0, double y_ = 0, double z_ = 0) : x(x_), y(y_), z(z_),
v(0) {}
};
double percentage_covered(const std::vector<std::vector<Point> >&
points) {
    int totalPoints = 0;
    int coveredPoints = 0;

    for (size_t i = 0; i < points.size(); ++i) {
        for (size_t j = 0; j < points[i].size(); ++j) {
            totalPoints++;
```

```
            if (points[i][j].v >= 1) {
                coveredPoints++;
            }
        }
    }
    return static_cast<double>(coveredPoints) / totalPoints;
}

bool is_whole(const std::vector<std::vector<Point> >& points) {
    return percentage_covered(points) > 0.9; // 大于90%被覆盖
}

double find_depth(int i,int z)
{
    return i*37.04/sqrt(3);
}
void cover(std::vector<std::vector<Point> >& points, int x, int y, int
direction, int previousDirection, std::vector<std::pair<int, int> >&
changedPoints)
{
    if(previousDirection==-1)
    {int sideDirections[2];
    if(direction == 0 || direction == 2) { // 左或右
        sideDirections[0] = 1; // 上
        sideDirections[1] = 3; // 下
    } else { // 上或下
        sideDirections[0] = 0; // 左
        sideDirections[1] = 2; // 右
    }
    for(int dirIdx = 0; dirIdx < 2; dirIdx++) {
        for(int i = 1; i < 150; i++) {
            int perpendicularDirection = sideDirections[dirIdx];
            int newx = x + directions[perpendicularDirection].first * i;
            int newy = y + directions[perpendicularDirection].second * i;
```

```
            if (newx < 0 || newy < 0 || newx >= points.size() || newy >=
points[0].size())
                continue;

            if(points[newx][newy].z < find_depth(i, points[x][y].z)) {
                break;
            } else {
                points[newx][newy].v++;
                //cout<<"d"<<newx<<','<<newy<<'v'<<points[newx][newy].v<
<endl;
                changedPoints.push_back(std::make_pair(newx, newy));
            }
        }
    }}
    else
    {
        int sideDirections[2];
        if(direction == 0 || direction == 2)
        { // 左或右
            sideDirections[0] = 1; // 上
            sideDirections[1] = 3; // 下
        }
        else
        { // 上或下
            sideDirections[0] = 0; // 左
            sideDirections[1] = 2; // 右
        }
        for(int dirIdx = 0; dirIdx < 2; dirIdx++)
        {
            for(int i = 1; i < 150; i++)
            {
                int perpendicularDirection = sideDirections[dirIdx];
                int newx = x + directions[perpendicularDirection].first *
i;
                int newy = y + directions[perpendicularDirection].second *
```

```
i;
            if (newx < 0 || newy < 0 || newx >= points.size() || newy >=
points[0].size())
               continue;
            if(points[newx][newy].z < find_depth(i, points[x][y].z))
            {
               break;
            }
            else
            {
            points[newx][newy].v++;
            //cout<<"d"<<newx<<','<<newy<<'v'<<points[newx][newy].v<
<endl;
            changedPoints.push_back(std::make_pair(newx, newy));
            }
         }
      }
      if(previousDirection == 0 || previousDirection == 2) { // 左或
右
         sideDirections[0] = 1; // 上
         sideDirections[1] = 3; // 下
      }
      else
      { // 上或下
         sideDirections[0] = 0; // 左
         sideDirections[1] = 2; // 右
      }
      for(int dirIdx = 0; dirIdx < 2; dirIdx++)
      {
         for(int i = 1; i < 150; i++)
         {
         int perpendicularDirection = sideDirections[dirIdx];
         int newx = x + directions[perpendicularDirection].first * i;
         int newy = y + directions[perpendicularDirection].second * i;
         if (newx < 0 || newy < 0 || newx >= points.size() || newy >=
```

```
points[0].size())
            continue;
         if(points[newx][newy].z < find_depth(i, points[x][y].z)) {
            break;
         } else {
            points[newx][newy].v++;
            //cout<<'p'<<newx<<','<<newy<<'v'<<points[newx][newy].v<
<endl;
            changedPoints.push_back(std::make_pair(newx, newy));
         }
         }
      }
   }
}
double computeCoverage(const std::vector<std::vector<Point> >& points)
{
   int totalPoints = 0;
   int covered = 0;
   for (size_t i = 0; i < points.size(); ++i)
   {
      for (size_t j = 0; j < points[i].size(); ++j)
      {
         totalPoints++;
         if (points[i][j].v>=1)
         {
            covered++;
         }
      }
   }
   return static_cast<double>(covered)*2 / totalPoints;
}
double computeCoverage_2(const std::vector<std::vector<Point> >&
points)
{
   int totalPoints = 0;
```

```
    int coveredMoreThanTwice = 0;
    for (size_t i = 0; i < points.size(); ++i)
    {
        for (size_t j = 0; j < points[i].size(); ++j)
        {
            totalPoints++;
            if (points[i][j].v>1)
            {
                coveredMoreThanTwice++;
            }
        }
    }
    return static_cast<double>(coveredMoreThanTwice)/2/ totalPoints;
}
double simulate(std::vector<std::vector<Point> >& points, int x, int y,
int steps,int pre)
{
    if (steps == 0) {
        return computeCoverage(points);
    }
    double bestCoverage = -1.0;
    for (int dir = 0; dir < directions.size(); dir++) {
        int newX = x + directions[dir].first;
        int newY = y + directions[dir].second;
        if(newX < 0 || newY < 0 || newX >= points.size() || newY >=
points[0].size())
            continue;
        std::vector<std::pair<int, int> > changedPoints;
        cover(points, newX, newY, dir, pre, changedPoints);
        double coverage = simulate(points, newX, newY, steps - 1,dir);
        if (coverage > bestCoverage)
        {
            bestCoverage = coverage;
        }
        // Undo the simulation for next iteration
```

```
        for(size_t i = 0; i < changedPoints.size(); i++) {
            int px = changedPoints[i].first;
            int py = changedPoints[i].second;
            points[px][py].v--;
        }
    }
    return bestCoverage;
}
int greedy_search(std::vector<std::vector<Point> >& points, int x, int
y,int previousdirection,std::vector<std::vector<int> >& visited)
{
    int bestDirection = -1;
    double bestCoverage = -1.0;
    for (int dir = 0; dir < directions.size(); dir++) {
        int newX = x + directions[dir].first;
        int newY = y + directions[dir].second;
        if(newX < 0 || newY < 0 || newX >= points.size() || newY >=
points[0].size())
            continue;
        if(previousdirection-dir==2||dir-previousdirection==2)
        continue;
        if(visited[newX][newY]>20) {
    continue; // 如果这个位置已经被访问过，就跳过
}
        std::vector<std::pair<int, int> > changedPoints;
        visited[newX][newY]++;
        cover(points, newX, newY, dir, previousdirection, changedPoints);
        double coverage = simulate(points, newX,
newY,2,previousdirection); // We already moved once, so 9 steps left
        if (coverage > bestCoverage) {
            bestCoverage = coverage;
            bestDirection = dir;
        }
        // Undo the simulation for next iteration
        for(size_t i = 0; i < changedPoints.size(); i++) {
```

```
            int px = changedPoints[i].first;
            int py = changedPoints[i].second;
            points[px][py].v--;
        }
    }
    return bestDirection;
}
int main()
{

    std::ofstream outfile("output.txt"); // 打开输出文件
    std::streambuf* orig_cout = std::cout.rdbuf(); // 保存原始 cout 缓冲区
    std::cout.rdbuf(outfile.rdbuf()); // 重定向 cout 到文件

    initializeDirections();
    std::ifstream file("data.csv");
    std::string line;

    // 读取 X 坐标
    std::getline(file, line);
    std::istringstream iss(line);
    std::string value;
    std::vector<double> x_coords;

    std::getline(iss, value, ','); // 忽略第一个值
    while (std::getline(iss, value, ',')) {
        x_coords.push_back(atof(value.c_str()));
    }

    // 初始化二维 vector
    std::vector<std::vector<Point> > points;
    for (size_t i = 0; i < x_coords.size(); ++i)
    {
        std::vector<Point> column;
```

```
        points.push_back(column);
    }
    while (std::getline(file, line))
    {
        std::istringstream iss_yz(line);
        double y;

        std::getline(iss_yz, value, ',');
        y = atof(value.c_str());
        for (size_t idx = 0; idx < x_coords.size(); ++idx)
        {
            std::getline(iss_yz, value, ',');
            double z = atof(value.c_str());
            Point p(x_coords[idx], y, z);
            points[idx].push_back(p);
        }
    }
    std::vector<std::vector<int> > visited(points.size(),
std::vector<int>(points[0].size(), 0));
    std::vector<int> currentPath;
    int x =200, y =250; // 初始位置
    int direction=-1;
    std::vector<std::pair<int, int> > changedPoints;
    while(!is_whole(points))
    {
        int previousdirection=direction;
        direction = greedy_search(points, x,
y,previousdirection,visited);
        if(direction==-1)
        break;
        cout<<direction<<endl;
        mindistance++;
        //cout<<mindistance<<endl;
        int newX = x + directions[direction].first;
        int newY = y + directions[direction].second;
```

```
        visited[x][y]++;
        cover(points, newX,
newY,direction,previousdirection,changedPoints);
        x=newX;y=newY;
    }
    cout<<endl;
    double coverage = computeCoverage(points);
    cout << "Minimum distance: " << mindistance << endl;
    cout << "Coverage : " << coverage * 100 << "%" << endl;
    double coverage_2 = computeCoverage_2(points);
    cout << "Coverage with v>2: " << coverage_2 * 100 << "%" << endl;
    std::cout.rdbuf(orig_cout); // 恢复原始 cout 缓冲区
    outfile.close(); // 关闭输出文件

    // 示例访问方式
    //cout << "x: " << points[1][1].x << ", y: " << points[1][1].y << ",
z: " << points[1][1].z << ", v: " << points[0][0].v << endl;
    return 0;
}
```

三、 支撑材料内容组成

文件名	主要功能
result1.xlsx	问题一计算结果
result2.xlsx	问题二计算结果
visualization.m	问题四可视化代码
3.cpp	问题三 C++规划代码
4.cpp	问题四 C++深度优先搜索代码
output.txt	问题四测线规划方案