基于几何方法的多波束测深系统测线布设模型

摘　要

在应用多波束测深系统进行海底探测任务时，为避免因排布过疏造成的数据缺失以及排布过密造成的数据冗余、资源浪费，需要对测线布设进行合理的分析、排布。本文通过应用几何方法，构建了精确的计算测量覆盖宽度、重叠率的**公式模型**，并在此基础上设计了**测线布设模型**，且对排布策略的优越性进行了**严格数学论证**。

针对问题一，我们建立了**基本几何关联多波束测深模型**。模型建立分为三个部分，使用**平面几何方法**进行构建、求解。第一部分，我们利用已知的斜率得出第 N 条测线位置的海水深度：$D_N = D_0 - Nd \cdot \tan(\alpha)$；第二部分，我们使用正弦定理并利用第一部分的模型得出覆盖宽度与测线位置间的关系：$W = \frac{2D_N \cos^2(\alpha)\sin(\theta)}{\cos(\theta)+\cos(2\alpha)}$ ；第三部分，通过使用相似三角形，我们将重叠率转化为相似比，得到相邻测线的覆盖率与测线位置之间的关系 $\eta_N = 1 - \frac{d}{2} \cdot \frac{\tan(\alpha)-\cot(\theta/2)}{D_N - Nd\tan(\alpha)}$ 。最后根据以上公式计算出结果并填入表格3。

针对问题二，我们建立了**变角度几何关联多波束测深模型**。在基本几何关联多波束测深模型的基础上，我们绘制了测量船测深的立体几何示意图7，将问题转化成求解探测面 (与侧线垂直且经过测量船的平面) 跟海底坡面的交线和水平面之间的夹角的余弦值 $\cos(\gamma)$，并通过**立体几何方法**计算得出结果。将该角度结果带入问题一中的模型，求得探测宽度与测量船位置、测线方向间的关系：$W = \frac{2D_N \sin(\theta)}{\cos(\theta)+(\cos(\theta)-1)\tan^2(\alpha)\sin^2(\beta)+1}$ 。最后利用上述公式计算出结果并填入表格4。

针对问题三，我们建立了**平行布设模型**与**扇形布设模型**。对于平行布设模型，我们对测线总长度进行**下界估计**。通过设计**迭代算法**，对于海平面上任意一条垂直于测线方向的直线，在给定其与中心点的水平距离 x、测线方向 β 后，我们可计算出与该直线垂直相交且符合要求的**最少测线数** $f(\beta, x)$。使用**积分估计方法**，将其在海域范围内对 x 积分得到函数 $g(\beta)$，得出初步估计的下界为 67.8 海里。进一步，对于测线总和位于 67.8 至 68 海里的区间，考虑边界漏测并对模型进行修正，发现修正后的测线总和大于 68 海里。另一方面，我们以南北走向作为测线方向，构造出了总测线长为 **68 海里**的结果，因此**严格说明** 68 海里为平行布设策略的最小值。对于扇形布设策略，我们考虑所有测线汇聚于一点的情况。在经过对测线利用率的分析后，我们选择将汇聚点设立在海平面与海底坡面的交线上，并利用弧度重新定义了重叠率 $\eta = \frac{v}{u}$。当汇聚点取在海域对称轴的位置上时，我们给出了**同为 68 海里**的测线排布策略。考虑到扇形排布策略在矩形海域边界会发生漏测现象、实际最小测线和应该大于以南北走向进行测线布设可取得的 68 海里，并且平行布置策略在实际中的简易操作性，最终我们采取了**平行布置策略**并给出的 68 海里的方案。

针对问题四，我们分析了全部横向、全部竖向的特殊测线布设模式后，设计了相对较优的**分块测线排布策略**。首先，我们**建立微分计算模型**，对相关数据进行估计计算。对于任意一条给定位置的测线，在局部计算出其探测区域以及和相邻测线的重叠率，并通过积分估计方法，可以计算出重叠率大于 20% 的测线位置总长度。然后考虑一条垂直于测线方向的狭长区域，计算出该区域上的漏测长度，将区域宽度微分再沿测线方向进行积分，进而得到总漏测面积。接下来，结合前三问的研究结果，我们考虑采用大致沿等高线布置平行测线的方式，最终我们提出了一套测线设计方案，经过计算，其测线的总长度为 **291.52** 海里，漏测海区占总待测海域面积的百分比为 **0.995%**，重叠率超过 **20%** 部分的长度为 **0**。

关键词：多波束测深系统，测线布设，几何计算，最优化，微分方法

1 问题重述

1.1 背景信息

多波束测深系统是在单波束测深的基础上发展而来的、利用声波在水中的传播特性以测量水体深度的探测技术，被广泛应用于海底地形探测的工作中。

在使用单波探测系统时，我们利用声波在均匀介质中作匀速直线传播、在不同界面上产生反射的原理，可以通过记录垂直向海底发射的声波从信号发射到信号接收的时间间隔并结合声波在海水中的传播速度计算出该测量点海水的深度。然而，由于单波探测系统仅能在不同时刻向下发射声波信号，最终只能得到测线（即航行线路）沿途的数据，而无法获得测线之间的有效海底地形数据。

我们在单波探测系统上发展出多波束测深系统，通过令测量船在一定的角度限制范围内发射大量具有不同发射角度的波束，使得每发射一个声脉冲，不仅可以在获得船下方的垂直海洋深度，同时还能获得与测量船航迹相垂直的直线上的许多水深值数据，相比于传统的单波探测系统具有范围大、速度快、精度高等优势。[1][2] 在实际应用中，测线的选择是十分重要的，需要考虑测量海域的地形进行相应的调整。如果测线布得过密，会造成数据冗余，使得工作量增大、造成人力物力的浪费; 如果测线布设过疏，会使相邻测线的覆盖范围没有重叠，造成该地区的数据缺失，导致目标漏测。[3]

为了确保测量的完备性、有效性，我们设立公式引入了“重复率”的变量，其定义为两条测线测量范围的重复区域与单个测线覆盖范围的比值。若某两条测线间不存在重复区域（此时通过公式计算得到的“重复率”可能为负数）则说明出现数据缺失，若重复率过高则说明发生数据冗余。实际应用中，邻测线的测幅应有 10% 的重叠率以避免产生探测盲区、保持测图精度，且同时应小于 20% 的重叠率以减少资源的浪费。[3]

在执行探测任务中，我们需要通过计算测线覆盖范围从而得到不同测线间的覆盖率。并为了达到效率最高、成果最好的探测结果，通过将覆盖率维持在某一限定范围内、测线距离总和最短作为目标对测线排布进行规划。

1.2 问题提出

1.2.1 问题 1 的提出

考虑探测水域的海底为斜坡、测量船与斜坡平行的情况，假设海底坡面与跟测线方向垂直的平面的交线是一条与海平面夹角为 α 的斜线，并称这个角度为“坡度”。要求建立数学模型，在已知坡度 α、多波束换能器的开角 θ、海域中心点处的海水深度 D、相邻两条测线的间距 d 的情况下，计算特定位置测线的覆盖宽度及相邻条带之间重叠率。

特别地，在给定了 $\alpha = 1.5°$、$\theta = 120°$、$D = 70\,\mathrm{m}$ 的前提条件下，要求结合测线距中心点处的距离计算对应的海水深度、覆盖宽度、某条测线与前一条测线测量区域的重叠率，并填入以表1为模板的表格中。

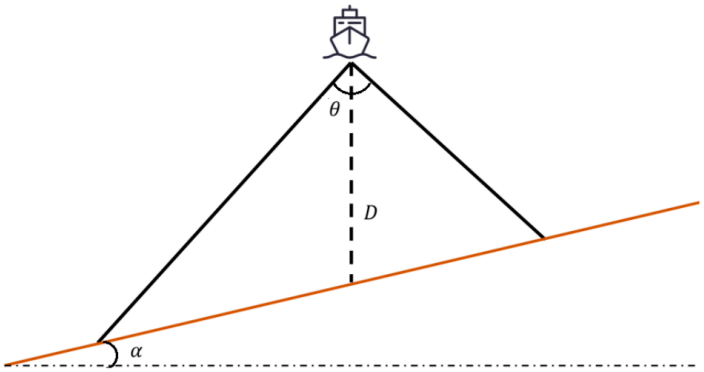


图 1: 问题 1 示意图

表 1: 问题 1 的表格模板

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					70				
覆盖宽度/m									
与前一条测线的重叠率/%	——								

1.2.2 问题 2 的提出

考虑探测水域的海底为斜坡、测量船与斜坡不一定平行的情况，考察一个矩形待测海域。假设测线方向与海底坡面的法向在水平面上投影的夹角为 β，同时已知坡度 α、多波束换能器的开角 θ、海域中心点处的海水深度 D、相邻两条测线的间距 d。要求建立数学模型，在此条件下计算各位置的多波束测深系统覆盖宽度。

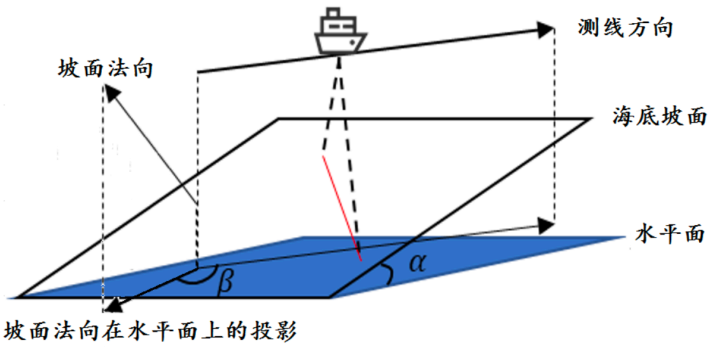


图 2: 问题 2 示意图

特别地，在给定了 $\alpha = 1.5°$、$\theta = 120°$、$D = 120\,\text{m}$ 的前提条件下，要求结合测线方向夹角及测量船距海域中心点处的距离计算对应的覆盖宽度，并填入以表2为模板的表格中。

表 2: 问题 2 的表格模板

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								
	45								
	90								
	135								
	180								
	225								
	270								
	315								

1.2.3 **问题 3 的提出**

考虑相对较一般的情况，对于探测水域的海底为斜坡、测量船与斜坡不一定平行的情况，考察一个南北为 2 海里、东西为 4 海里的矩形待测海域。在给定了坡度 $\alpha = 1.5°$、多波束换能器的开角 $\theta = 120°$、海域中间处的海水深度 $D = 110\,\mathrm{m}$ 的前提条件下，设计一个满足相邻条带的重叠率处于 10% 至 20% 之间的测线排布策略，使得测线可完全覆盖整个海域并且测量总长度最短。

1.2.4 **问题 4 的提出**

考虑相最一般的情况，根据附件中给出的若干年前某海域的南北长 5 海里、东西宽 4 海里的海水深度数据，根据海水数据设计测线排布策略，使得测线覆盖范围尽可能包括整个海域、相邻条带之间的重复率尽可能在 20% 以下、测线的总长度尽可能短。同时，在得出策略后计算测线的总长度、漏测海区占总待测海域面积的百分比、重叠率超过 20% 部分的长度。

2 问题分析

2.1 问题 1 的分析

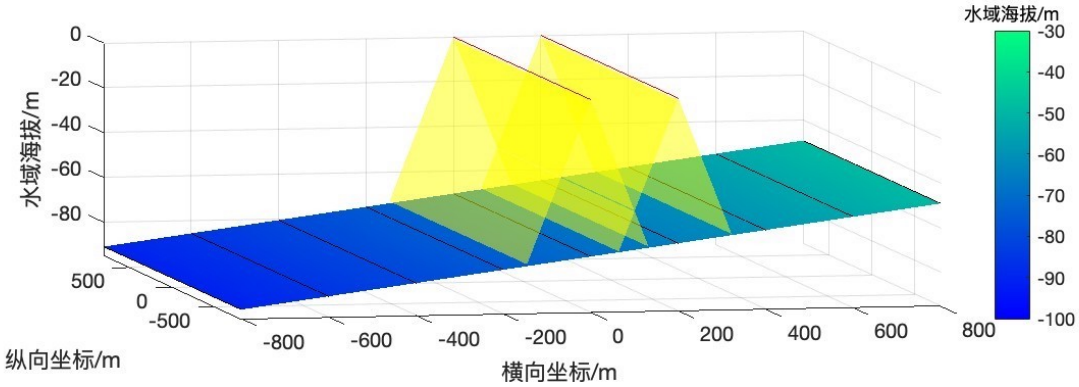
首先，为了建立模型进行求解，我们可以考虑与测线方向垂直的平面作为截面截出的平面图形，从而进行平面几何模型的建立并进行相应的求解。问题一的模型可以分成三部分建立。

第一部分，计算各个位置的海水深度：在题目假设下，我们发现海水深度与侧线距中心点处的距离呈线性关系，通过计算斜率可进而计算各个位置的海水深度。

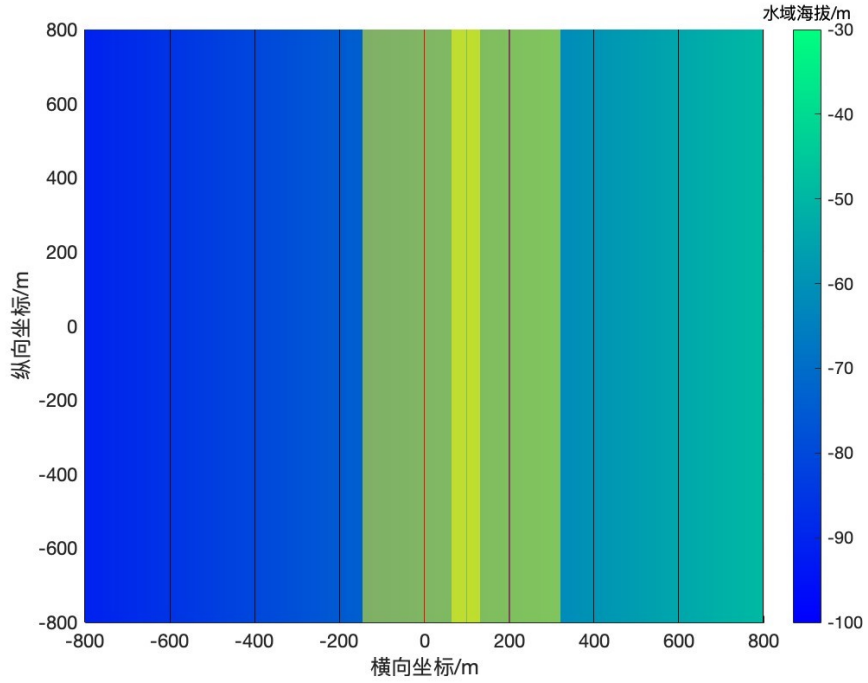
第二部分，计算各个位置的覆盖宽度：可以运用平面几何模型中的解三角形方法，利用第一部分求出的海水的深度并结合正弦定理从而求得探测宽度。

第三部分，计算与前一条侧线的重叠率：可以利用平面几何模型中的相似三角形方法，将重叠区域宽度与探测宽度的比值转化成相似三角形的相似比，进而简化运算过程得出结果。

在得到以上公式模型后，我们可以将题中数据带入，得到问题一的最终结果。



(a) 问题 1 三维海域示意图



(b) 问题 1 平面海域示意图

图 3: 问题 1 海域示意图

2.2 问题 2 的分析

相较于问题一，问题二中的测量船航行方向与海底坡面不平行，因此无法简单转化成截面下的平面几何模型，需要通过立体几何模型进行相应求解。我们尝试在问题一的结论的基础上进一步扩展，进而建立数学模型对该问题进行求解。在问题一中，我们建立了关于计算探测宽度的公式和模型，该模型的使用需要海水深度、多波束换能器的开角、在与测量船行驶方向垂直的平面跟海底坡面的交线和水平面之间的夹角三个参数。对于前两个参数，我们可以通过简单计算、题目已知信息获得；对于最后一个参数，我们可以利用立体几何模型并结合勾股定理等平面几何方法求得。在完成该模型的建立后，带入题中所给数据即可得到本题结果。

2.3 问题 3 的分析

在问题三中，我们需要设计测线布设方式，对已知坡度的海底坡面进行探测。为此，我们提出了“平行布设法”与“扇形布设法”两种方案，即分别假设测线是互相平行或延长线汇聚于某一点。

对于平行布设模型，我们首先考虑在忽略改变角度造成的边界漏测时估计出给定测线角度下所需要的测线总长的最小值。我们考虑一条垂直于所有测线的直线 L，其与海域边界产生两个交点，我们在这两个交点之间的线段上进行计算，准确计算与其相交的测线的最小数目，使得线段上所有点均满足探测要求。假定所有测线的方向与正西方向的偏角是 β，那么 L 的方向也可以确定。当已知 L 距离中心点的距离 x 时，我们可用迭代算法计算与 L 相交的测线的最小数目，并用函数 $f(\beta, x)$ 表示。然后，对该函数关于 x 进行积分，得到一个关于 β 的函数 $g(\beta)$，使用该函数可以计算出 β 方向对应的最小测线总长度。求出每个方向 β 对应的最小值后，我们可以把边界漏测带来的影响考虑在内，进行进一步的估计、最终找到测线总长的最小值。

对于扇形布设模型，注意到若两条间隔测线的重叠率会随着测量船在测线上的位置变化而改变、无法维持在同一数值，则会造成覆盖率偏大、数据冗余的情况，无法成为最优策略。因此，为找到最优策略，我们考虑覆盖率随着测量船位置改变保持不变的情形，应将扇形汇聚点选于海平面与坡面的交线上。然后，我们可以通过编程对扇形分布时的最优情况进行数值计算。

最终，通过比较平行布设模型、扇形布设模型能够取得的最小总测线长度并结合实际情况得出最终结果。

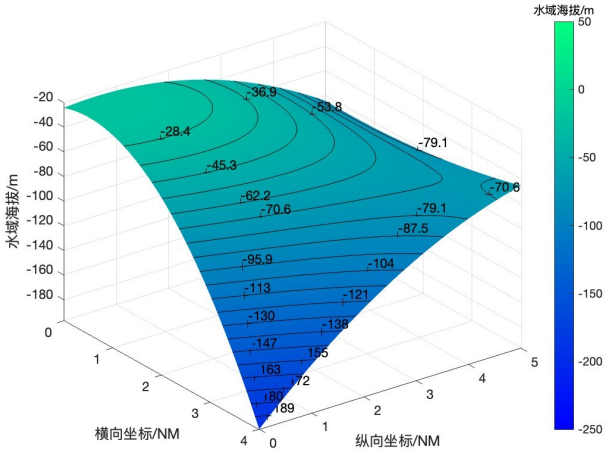
2.4 问题 4 的分析

我们首先将问题 4 给的数据进行可视化，得到了海底地形的等高线图，如图4(a)以及图4(b)所示，可以观察得出最大坡角出现在东南海域，我们使用 MATLAB 拟合得出的最大坡角为 3.37°。由于海底地形起伏相对较小、较为平整，可以沿用问题三中的模型辅助求解。根据问题三建立的模型以及论证过程，我们得知，沿等高线的方式能取得

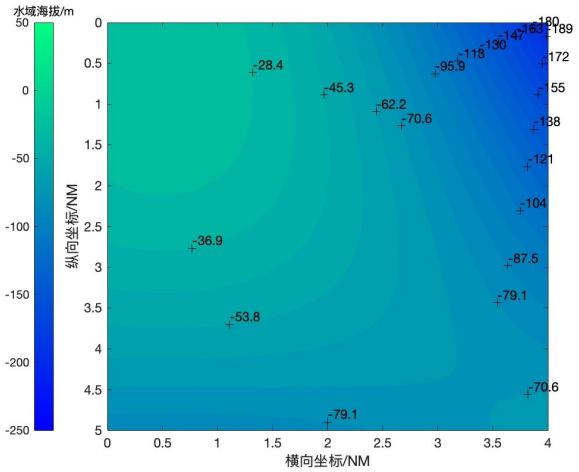
较好的测深效果，但考虑到等高线并非直线，我们尝试将整片海域分块，每一区域内的等高线接近平行的直线。我们在此基础上布设测线，并用 Python 设计程序来分别计算测线的总长度、漏测海区占总待测海域面积的百分比、重叠区域中重叠率超过 20% 部分的长度。

考虑到前三问给出的结果，我们第四问的测线分布理应尽可能沿着海底等高线以期获得最大的测绘效率。因此，通过观察地形图，我们考虑将区域划分成斜线布设、竖直布设、水平布设三部分，分别以最高效率对给定区域内进行探深，最终得到对整个海域的探测结果。

同时，为建立能够计算测线长度、重复率超过 20% 部分的总长度、漏测面积，我们考虑使用微积分的方法。考虑垂直于测线方向厚度为 dx 的狭长海域，计算其对应漏测长度、重叠率超标长度、漏测面积，并对其进行积分从而得到结果。



(a) 问题 4 三维海域示意图



(b) 问题 4 平面海域示意图

图 4: 问题 4 海域示意图

3 模型假设

- 测量船在航行过程中可以被看作质点，忽略船身长度宽度及形状
  - 船身长度远小于测线间距离、海洋水深，将船视作点不会影响计算结果，但能大大简化模型。
- 测量船按照直线路径匀速航行
  - 有关实验表明，在应用多波束系统采集数据的过程中，若测量船出现转弯和速度增减，则船只会发生横摇和纵摇误差。为消除误差带来的影响，测量船应并保持匀速直线航行，因此我们在规划的时候假设测量船以匀速直线的模式航行。[3]
- 不考虑海域情况、天气情况、测量船运作情况等外界因素对测量船航行路线的影响，即测量船可以严格按照设计的测线进行运作
  - 此类外部状况属于难以预测的突发事件，若将其考虑在内则模型会过于复杂。
- 测深系统执行期间，不考虑海洋侵蚀等因素对海底地形、测量船设备造成的微小变化
  - 测量时间相对较短，在此期间内海洋造成的侵蚀作用可以忽略不计。
- 假设海底声波沿直线传播，且不考虑声波来回时间
  - 测量时间相对较短，声波传播速度可以忽略不记。测量深度较低，测量时海水密度变化可以忽略不计。

4 符号说明

符号	说明	单位
N	测线之间的编号	
D	船只测量位置到海底的直线距离	m
d	测线与最近的测线之间的距离	m
W	多波束测深覆盖范围的水平距离	m
α	海底坡面的坡度	°
β	在与测量船行驶方向垂直的平面跟海底坡面的交线和水平面之间的夹角	°
γ	在与测量船行驶方向垂直的平面与海底坡面的交线与水平面之间的夹角	°
θ	测量船多波束换能器的开角	°
u, v	扇形布设模型下的覆盖角弧度	°

5 模型的建立与求解

5.1 问题 1：基本几何关联多波束测深模型的建立及求解

我们考虑分析与测线方向垂直的平面作为截面（如图5所表示），并建立基于平面几何模型的基本几何关联多波束测深模型。图中，线段 AB 代表海平面，线段 CD 代表海底坡面，点 G 与点 H 表示相邻的两条测线所在的位置（测线可视为垂直于平面的直线），线段 GI 和线段 GJ 分别是测量船在点 G 位置发出的多波束的左右两个边界，边界与海底坡面的交点分别为点 I 和点 J。线段 GE 是过点 G 的竖直线，其与海底坡面的交点记为点 E。同样地，对处于点 H 位置的测量船，我们也可以定义左边界与海底坡面的交点 K、竖直线与海底坡面的交点 L、右边界与海底坡面的交点 F。接下来我们运用平面几何方法对问题进行分析与求解。

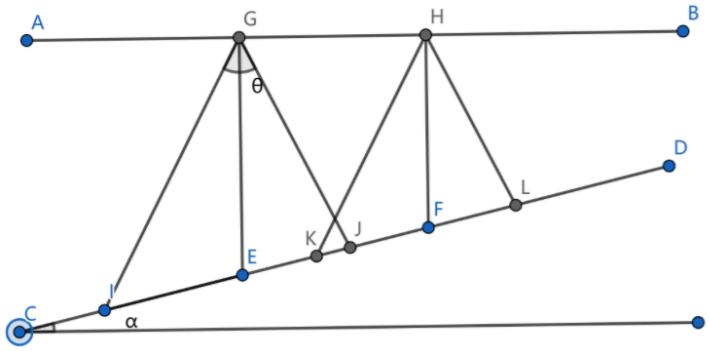


图 5: 基本几何关联多波束测深模型示意图 1

5.1.1 海水深度的计算

首先, 计算给定位置点的海水深度。我们将中心测线视作 “第 0 条测线”，记水平位置为正向 200 米的测线为第 1 条测线，−200 米处的测线为第 −1 条测线，以此类推。假设 G 和 H 分别为第 $N-1$ 条测线和第 N 条测线，则可通过平面几何方法计算得到 H 处的海水深度为:

$$D_N = D_0 - Nd \cdot \tan(\alpha)$$

5.1.2 覆盖宽度的计算

然后，我们考虑对 H 处的探测宽度进行计算，即线段 KL 的水平距离。在三角形 $\triangle HKF$ 中。由正弦定理可得：$KF/HF = \sin(\frac{\theta}{2})/\sin\angle HKF$，注意到 $\angle HKF = \frac{\pi}{2} - \frac{\theta}{2} - \alpha$，因此可以得到 $KF = D_N \sin(\frac{\theta}{2})/\cos(\frac{\theta}{2}+\alpha)$。同理，我们可以求得 $FL = D_N \sin(\frac{\theta}{2})/\cos(\frac{\theta}{2}-\alpha)$ 。将两者相加，并运用和差化积公式化简后可得

$$KL = \frac{2D_N \cos(\alpha)\sin(\theta)}{\cos(\theta)+\cos(2\alpha)}$$

注意到 $W_N = KL\cos(\alpha)$，我们可以进一步得到覆盖宽度：

$$W = \frac{2D_N \cos^2(\alpha)\sin(\theta)}{\cos(\theta)+\cos(2\alpha)}$$

5.1.3 重叠率的计算

最后，我们计算第 N 条测线与第 $N-1$ 条测线的重叠率。如图6所示，根据重叠率公式 $\eta = 1 - \frac{d}{w}$ ，KJ/KL 的数值即可表示 H 处的重叠率。设线段 KH 与线段 GJ 相交于点 N，过点 N 作竖直线分别交线段 AB、CD 于点 O、P。注意到 $NJ \parallel HL$，因此 $\triangle NKJ$ 相似于 $\triangle HKL$，故得 $KJ/KL = \triangle NKJ$ 与 $\triangle HKL$ 的相似比 $= NP/HF$。HF 等于 H 处的海水深度 D_N。$NP = OP - ON$。注意到 OP 是梯形 $GEFH$ 的中位线，则 $OP = (D_N + D_{N-1})/2$；而 $ON = OH\cot(\frac{\theta}{2}) = \frac{d}{2}\cot(\frac{\theta}{2})$。进而，结合前一小节计算海水深度的模型，我们可以求得重叠率

$$\eta_N = \frac{NP}{HF} = \frac{OP-ON}{HF} = \frac{\frac{D_N+D_{N-1}}{2} - \frac{d}{2}\cot(\frac{\theta}{2})}{D_N} = 1 - \frac{d}{2}\cdot\frac{\tan(\alpha)-\cot(\frac{\theta}{2})}{D - Nd\tan(\alpha)}$$

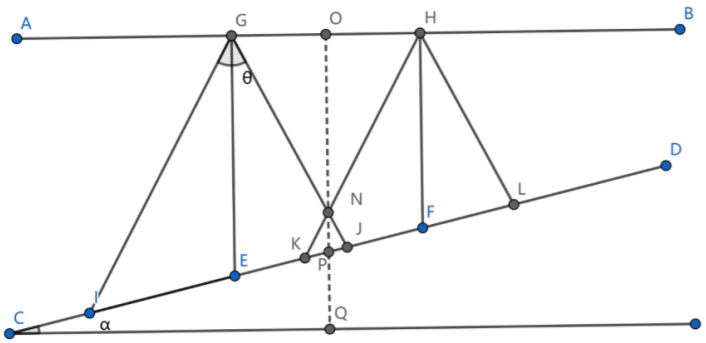


图 6: 基本几何关联多波束测深模型示意图 2

5.1.4 问题 1 的结果

根据前几个小节得到的计算公式，带入表1中的数据，我们最终得到如表3所示的结果，并将结果存入文件附录 7 的文件 result1.xlsx。

5.2 问题 2：变角度几何关联多波束测深模型的建立及求解

5.2.1 变角度几何关联多波束测深模型的建立

我们考虑在空间内建立基于立体几何模型的变角度几何关联多波束测深模型，如图7所示建立以射线 AB、AC、AD 为坐标轴方向的空间直角坐标系，平面 CAB 是平行

表 3: 问题 1 计算结果

测线距中心点处的距离/m	-800	-600	-400	-200	0	200	400	600	800
海水深度/m	90.95	85.71	80.47	75.24	70	64.76	59.53	54.29	49.05
覆盖宽度/m	315.71	297.53	279.35	261.17	242.99	224.81	206.63	188.45	170.27
与前一条测线的重叠率/%	——	35.7	31.5	26.7	21.3	14.9	7.4	-1.5	-13.4

于水平面的海底平面，平面 JAC 是海底坡面，其与海底平面呈 α 度，直线 AE 是测线 PQ 在海底平面的投影，根据定义有 $\angle EAB = \beta$，直线 AR 是测线 AQ 在海底坡面上的投影，点 N 表示测量船的位置，线段 NM、NO 是由 N 发射的多波束的两个边界，线段 NG 是竖直线、与海底坡面交于点 G，点 M、G、O 均位于海底坡面 JAC 上。根据基本几何关联多波束测深模型给出的关于覆盖宽度的公式

$$W = \frac{2D_N \cos^2(\gamma)\sin(\theta)}{\cos(\theta) + \cos(2\gamma)}$$

其中 γ 为与测量船行驶方向垂直的平面与海底坡面的交线与水平面之间的夹角。我们只需要计算 γ 的余弦值，即可利用第一问模型得到适用于该情况的较为一般的立体模型。

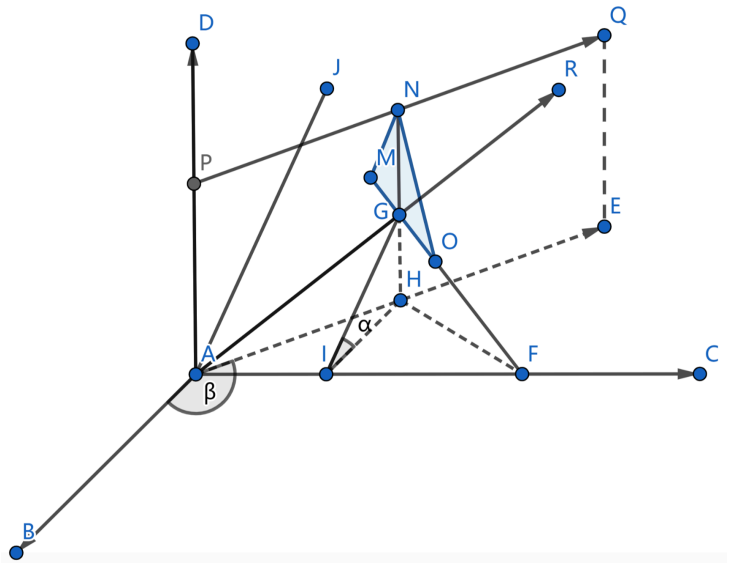


图 7: 变角度几何关联多波束测深模型示意图 1

如图7所示，延长线段 MO 交线段 AC 于点 F，延长线段 NG 交线段 AE 于点 H，$\angle GFH$ 的余弦值即为我们需要计算的目标。由于点 H、F 均位于平面 NMO 上，结合 $AE \perp NMO$，可得 $FH \perp AH$。过点 G 作线段 GI 与 AC 垂直，交 AC 于点 I，连接 HI，则 $\angle GIH = \alpha$。通过角度计算可得 $\angle HFA = \pi - \beta$，$GH = IH\tan(\alpha)$, $HF = IH/\sin(\pi - \beta)$, $\tan\angle GFH = \tan(\alpha)\sin(\beta)$。因此，我们可以得到

$$\cos\angle GFH = \left(\tan^2(\alpha)\sin^2(\beta) + 1\right)^{-\frac{1}{2}}$$

表 4: 问题 2 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	415.69	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	309.1	273.42	237.75	202.08	166.4
	180	415.69	365.29	314.89	264.5	214.1	163.7	113.3	62.9
	225	416.12	380.45	344.77	309.1	273.42	237.75	202.08	166.4
	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	558.82	594.49	630.16	665.84

在覆盖宽度计算公式中，用 $\cos\angle GFH$ 替代 $\cos\alpha$ 后并计算化简得

$$W = \frac{2D\sin(\theta)}{\cos(\theta) + (\cos(\theta) - 1)\tan^2(\alpha)\sin^2(\beta) + 1}$$

其中，D 代表的海水深度可用问题一中建立的模型进行计算。

5.2.2 问题 2 的结果

根据上一小节得到的计算公式，带入表2中的数据，我们得到如表4所示的结果，并将结果存入文件附录 8 的文件 result2.xlsx。

5.3 问题 3：平行布设模型与扇形布设模型的建立与求解

在该问题中，我们需要在已知海底地形为斜面且坡度、中心海底深度等数据已知的情况下设计一个合理的测线排布方式。为了减少测量误差，测量船应以匀速直线的模式航行。在同时考虑实际应用的可行性及资源的利用率后，我们提出两个效用较高的测线布设方式：平行布设法、扇形布设法，相应地建立了模型并进行了求解。

5.3.1 平行布设模型

仍然使用几何办法对该问题进行建模，考虑所有的测线平行分布的情况。使用俯视视图，如图8所示。其中，矩形 $ABCD$ 表示海域，点 E 是海域中心，射线 EM 方向是正西方向，所有测线均平行且与直线 EM 的夹角呈 β 角，直线 L 垂直于所有测线且与矩形边界交于点 S、T。在图示中，点 T 处海水较浅，点 S 处海水较深。

为了计算 ST 上至少需要多少测线经过，我们考虑对测量海域沿测线方向微分，考虑沿着 ST 的狭长海域，如图9。为了使测线数最少，我们注意到在测线向深海方向移动的时候探测宽度会变大，因此我们不妨假设测量船测深时产生的探测面恰好探测到 T

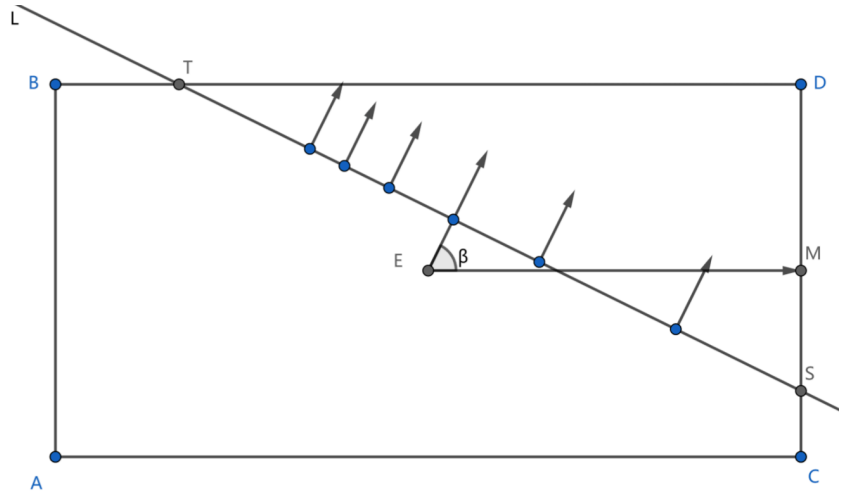


图 8: 平行布设法示意图

点，并且相邻的测线重叠率为 10%，以此来估算不同测线排布方向所需的最小测线数的下界。

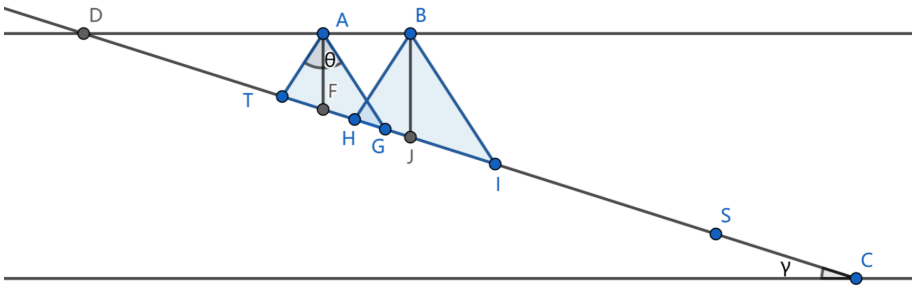


图 9: 计算最少测线数示意图

考虑检测平面的侧视图，如图9中，A，B 是相邻两条测线，由 A 产生的探测面 ATG 刚好经过最浅处 T 点，由 B 产生的探测面 BHI 满足重叠区域 HG 是 TG 的 10%，T 的海水深度记为 $D_{1,\text{left}} = D_{\min}$，利用正弦定理，因此 G 的海水深度 $D_{1,\text{right}}$ 可使用下式计算：

$$\frac{D_{1,\text{right}}}{D_{1,\text{left}}} = \frac{DG}{DT} = \frac{DG}{DA} \cdot \frac{DA}{DT} = \frac{sin(\frac{\pi}{2}+\frac{\theta}{2}) \cdot \sin(\frac{\pi}{2}+\frac{\theta}{2}-\alpha)}{\sin(\frac{\pi}{2}-\frac{\theta}{2}) \cdot \sin(\frac{\pi}{2}-\frac{\theta}{2}-\alpha)} = \frac{\cos(\alpha-\frac{\theta}{2})}{\cos(\alpha+\frac{\theta}{2})}$$

因此可以得到迭代公式：

$$D_{1,\text{right}} = D_{1,\text{left}} \cdot \frac{\cos(\alpha-\frac{\theta}{2})}{\cos(\alpha+\frac{\theta}{2})}$$

再由重叠率 10% 的条件得到下一探测区域最浅方向的极限坐标：

$$D_{2,\text{left}} = 0.9D_{2,\text{right}} + 0.1D_{1,\text{left}}$$

设 $D_{\max}$ 表示 S 处的海水深度。我们对 H 重复这一计算过程，从而构建出以下迭代算法可以知道所有方案中经过 L 的测线数的下界。

Algorithm 1: 计算与指定直线相垂直的排布方向满足条件的最小测线数目

Result: 与指定直线相垂直的排布方向满足条件的最小测线数目

$D_{1,\text{left}} = D_{\min}$;
$n = 1$;
while True do
    $D_{n,\text{right}} = D_{n,\text{left}} \cdot \frac{\cos(\alpha-\frac{\theta}{2})}{\cos(\alpha+\frac{\theta}{2})}$;
    if $D_{n,\text{right}} > D_{\max}$ then
        Result = n;
        break;
    else
        $D_{n+1,\text{left}} = 0.9 \cdot D_{n,\text{right}} + 0.1 \cdot D_{n+1,\text{left}}$;
        $n = n + 1$;
    end
end

定义函数 $f(\beta, x)$ 为与 L 相交的最小测线数，β 表示测线方向，x 表示距离中心点距离。我们给出的算法表明 $f(\beta, x)$ 可以被计算。我们对函数 $f(\beta, x)$ 关于 x 在区域内积分得到关于 β 的函数 $g(\beta)$，从而得到 β 方向上测线长度的最短估计。计算数据并作出 $g(\beta)$ 的图像，如图10所示。

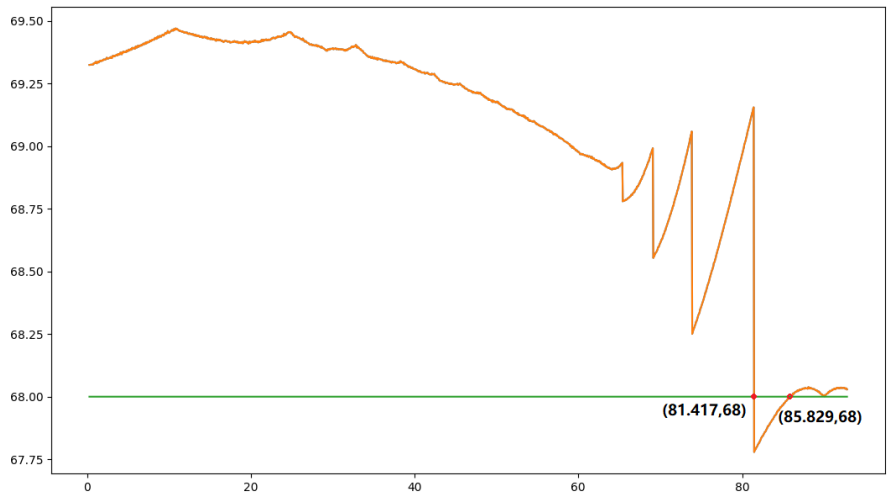


图 10: 测线总长度关于角度 β 的下界估计

从图10中可以看出，当 $\beta = 90°$ 时取到极小值 68；当 $81.42° < \beta < 85.83°$ 时，总侧线长度小于 68 海里，但是当 β 不为 0° 或者 90° 时，以 β 方向设计测线会发生漏测现象，因此我们对函数 $g(\beta)$ 做一定的修正。

如图11所示，m 是矩形海洋的南边界，整个图是从天空往海洋看的俯视图，CD、EH 是相邻两测线，方向为与正西方向呈 β 角，蓝色区域是被探测到的区域，深蓝色区

域是重叠区域，红色区域是漏测区域，因此，每条测线还应当延长以保证无漏测现象。我们计算 CD 测线应当延长的距离 DD_1 并且找到 QF 与 DD_1 的关系：由于海底坡度较小，不妨假设 $DK = DL$, 则

$$DM = \frac{0.5-\eta}{1-\eta} LM$$
$$DD_1 = DM\cot(\beta)$$
$$LM = QF\sin(\beta)$$

联立可得

$$DD_1 = \frac{0.5-\eta}{1-\eta} QF\cos(\beta) > 0.375QF\cos(\beta)$$

对上式两边求和，式子左边求出的和是矩形海洋南侧应当补充的测线长度之和，对于式子右边，由于 $0.1 < \eta < 0.2$, 故 $\frac{0.5-\eta}{1-\eta} > 0.375$, 因此可得

$$DD_1 > 0.375QF\cos(\beta)$$

所有 QF 求和之后就是矩形海洋的南边界长度，因此，再算上北边应当补充的测线长度，可以得到至少应当补充的测线长度 (由于东西边界应当补充的测线长度没有算入)

$$\Delta S > 4 \times 1852 \times 0.375 \times 2 \cdot \cos(\beta) = 5556\cos(\beta)$$

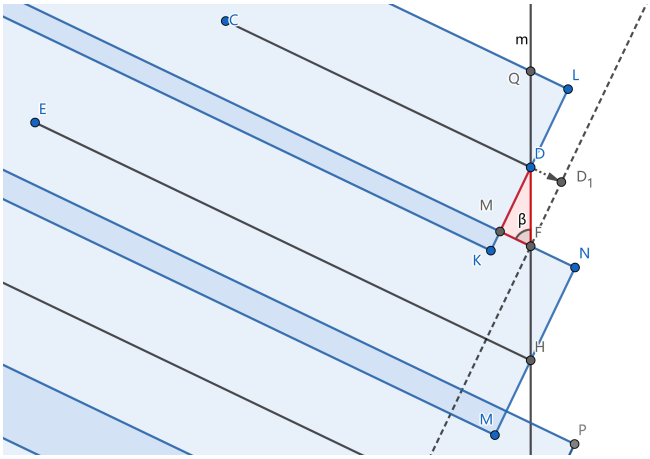


图 11: 漏测现象以及补充测线长度示意图

根据这个公式我们在原先函数小于 68 海里时，即 $81.42° < \beta < 85.83°$ 时重新计算了该方向测线总长度的下界函数，如图12所示。其中，蓝线代表未补充修正路径前的测线总长度下界估计，黄线代表补上这一段漏测的测线距离后总测线长度的下界估计，而绿线指出了 68 海里的位置。可以看到，在补上这部分漏测的测线长度后，在非 90 度的情况下我们计算得到的测线总长度下界大于 68 海里，因此，南北朝向的测线排布方式正是最优。其中，一组符合要求的测线位置如表5所示。

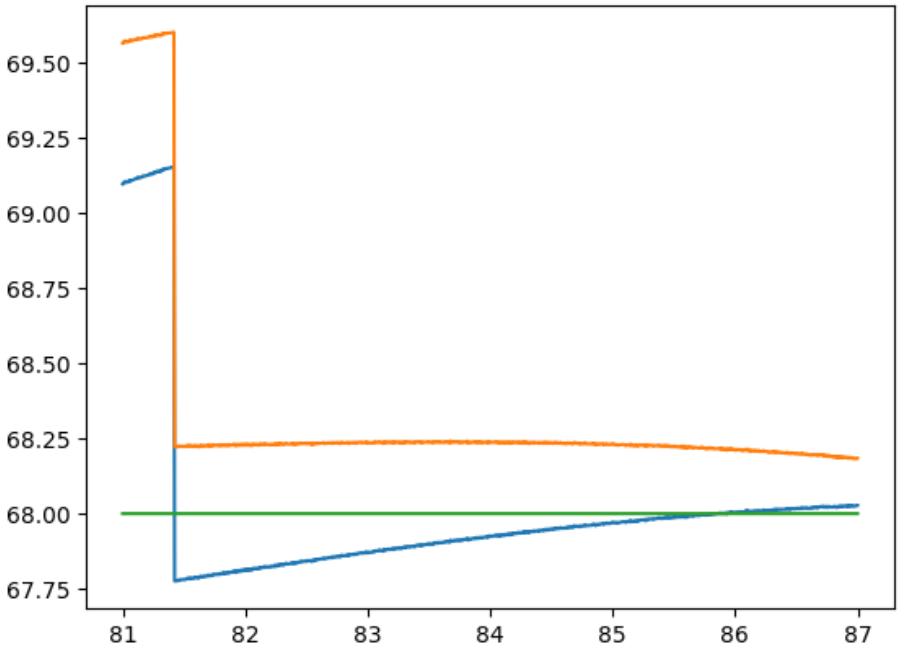


图 12: 补上漏测部分后测线总长度关于角度 β 的下界估计

表 5: 南北走向测线实例

航道编号	1	2	3	4	5	6	7
测线横坐标/m	-3345.84	-2756.47	-2213.13	-1712.23	-1250.44	-824.72	-432.24
测线处深度/m	197.61	182.18	167.95	154.84	142.74	131.6	121.32
航道编号	8	9	10	11	12	13	14
测线横坐标/m	-70.42	263.15	570.66	854.16	1115.52	1356.47	1578.6
测线处深度/m	111.84	103.11	95.06	87.63	80.79	74.48	68.66
航道编号	15	16	17	18	19	20	21
测线横坐标/m	1783.38	1972.17	2146.21	2306.67	2454.59	2590.96	2716.68
测线处深度/m	63.3	58.36	53.8	49.6	45.72	42.15	38.86
航道编号	22	23	24	25	26	27	28
测线横坐标/m	2832.58	2939.43	3037.93	3128.74	3212.46	3289.64	3360.8
测线处深度/m	35.83	33.03	30.45	28.07	25.88	23.86	21.99
航道编号	29	30	31	32	33	34	
测线横坐标/m	3426.39	3486.87	3542.62	3594.02	3641.4	3685.08	
测线处深度/m	20.28	18.69	17.23	15.89	14.65	13.5	

5.3.2 扇形布设法

考虑测线不平行但延长线汇聚于同一点的情况，此时可以将测线看作以该点作为顶点的射线。

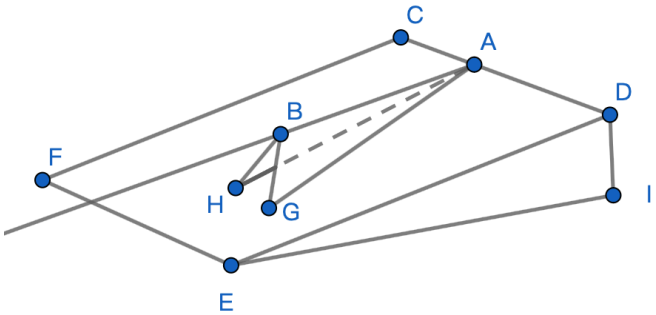
汇聚点的选取与重复率的定义

使用类似于平行布设法的思路，我们应当考虑一种测线排布形式，使得所有测线上的重叠率均为 10% 的情况。否则，若存在位置重叠率大于 10%，则说明该部分存在资源浪费、不应当为最优解；若存在位置重叠率小于 10%，则说明该位置不满足题设要求。因此，我们考虑寻找汇聚点位置，使得以该点出发的所有测线中，相邻测线间重叠率严格等于 10%。

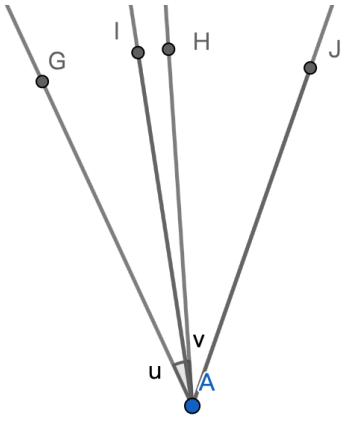
为保证这一点，我们只能将汇聚点取在海平面与海底坡面的交线上。因为无论汇聚点取在交线内侧还是外侧，都会出现重复率随着测量船在测线上的位置变化而变化的情况。下面，我们将说明：把汇聚点取在交线处时，可以使得测量船在测线上移动时重复率保持不变。

事实上，如图13(a)所示，考虑点 A 为汇聚点、平面 $CDEF$ 为坡面、射线 AB 为一条测线、点 B 为某一时刻测量船在海平面上的位置、平面 ABH 与平面 ABG 分别是多波束测量的两个边界，其与坡面分别交得直线 AH 以及直线 AG。换言之，该测线的探测范围为两条从汇聚点 A 发出的射线围成的范围。

接下来，我们可以由角度的弧度给出关于重叠率的定义。如图13(b)所示，考虑某两条测线的覆盖区域分别为射线 AG 与射线 AH 围成的区域、射线 AI 与射线 AJ 围成的区域, u 与 v 分别对应重叠区域 GAH、覆盖区域 HAI 的弧度，因此我们可以定义 $\eta = \frac{v}{u}$。当测量船在测线上运行时，重复区域始终仅占覆盖区域的固定比值。特别地，我们可以通过选择、调整，使得相邻的测线间覆盖率均等于 10%。因此，我们可以认为，将汇聚点取在该交线上是合适的选择。



(a) 扇形布设法汇聚点选取示意图



(b) 扇形布设法重叠率示意图

图 13: 扇形布设法的汇聚点选取及重叠率定义示意图

扇形布设模型中测线的排布及分析

为了计算机实现这一点，对于给定的一条测线，我们需要计算它产生的覆盖角 u 以及与相邻测线的重叠率 η。如图14, AB 是海底坡面与海平面的交线，BC 是测线方向，且与正西方向呈 β 角，CDE 是 C 产生的探测面，我们所计算的覆盖角 u 是从俯视海平面的角度看的，为了精确我们需要计算 u 的两个分角 u_1、u_2，其中 $\tan(u_1) = DF$ 的水平距 $/BC$, $\tan(u_2) = EF$ 的水平距 $/BC$。

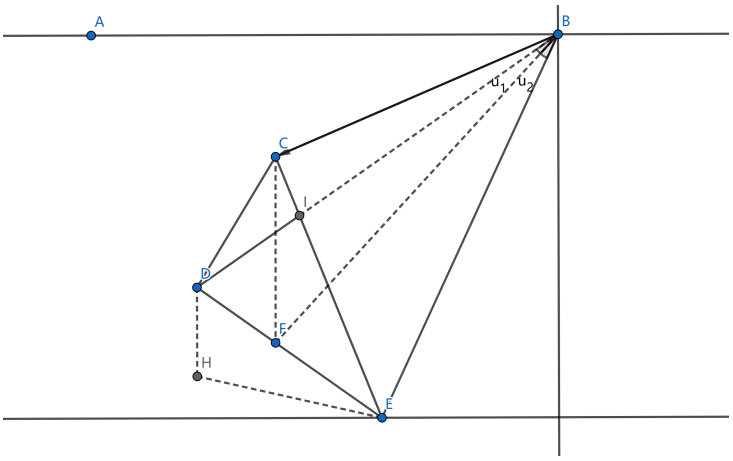


图 14: 计算覆盖角示意图 1

作出探测面与海底平面的交线，H 是 D 在该交线上的垂足，考虑图15, 在 $\triangle CDF$ 中使用正弦定理，得到

$$DF = \frac{CF\sin(\frac{\theta}{2})}{\cos(\gamma - \frac{\theta}{2})}$$

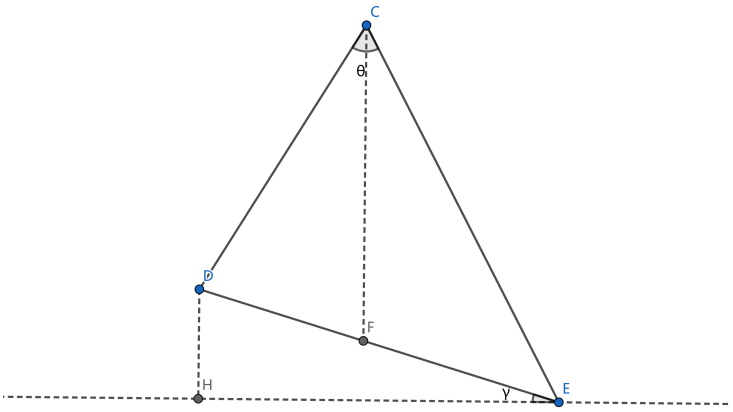


图 15: 计算覆盖角示意图 2

从而

$$DF\text{的水平距} = DF\cos(\gamma) = \frac{CF\sin(\frac{\theta}{2})\cos(\gamma)}{\cos(\gamma - \frac{\theta}{2})}$$

带入 $\tan(u_1) = DF$ 的水平距 $/BC$ 并结合 $CF/BC = \tan(\alpha)\cos(\beta)$ 得到:

$$\tan(u_1) = \frac{\tan(\alpha)\cos(\beta)\sin(\frac{\theta}{2})\cos(\gamma)}{\cos(\gamma - \frac{\theta}{2})}$$

类似可得：

$$\tan(u_2) = \frac{\tan(\alpha)\cos(\beta)\sin(\frac{\theta}{2})\cos(\gamma)}{\cos(\gamma+\frac{\theta}{2})}$$

再由合角公式可得：

$$\tan(u) = \tan(u_1+u_2) = \frac{\tan(\alpha)\cos(\beta)\sin(\theta)\cos^2(\gamma)}{\cos(\gamma-\frac{\theta}{2})\cos(\gamma+\frac{\theta}{2}) - \tan^2(\alpha)\cos^2(\beta)\sin^2(\frac{\theta}{2})\cos^2(\gamma)}$$

故对于给定方向的测线，我们都可以算出其覆盖角，而计算相邻两测线重叠角 v 可利用其中一条测线的覆盖角的右侧分角加上另一条测线的覆盖角的左侧分角再减去两测线的夹角得到。具体公式放在代码里。由此我们可以计算两相邻测线的重叠率。

接下来，我们考虑将汇聚点取在海域对称轴位置，并构造相应的测线排布方案。如图16所示矩形 $ABCD$ 表示题中给定的海域，海域中心为坐标原点，E 为汇聚点，OE 方向为正方向（正东方向），E 发出的射线与矩形相交得到的线段表示相应的测线。

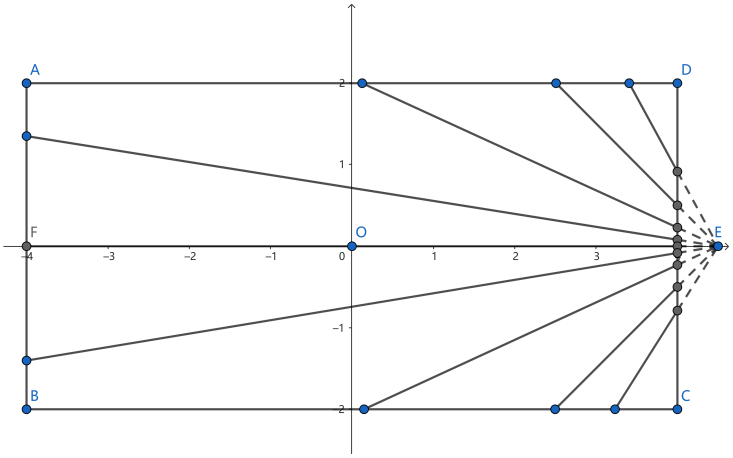


图 16: 扇形布设法测线排布示意图

注意到，欲使测线总长度取得最小值，应使得任意两条测线之间的重复率最小。在模型中，我们应当选取测线排布，使得相邻测线间重复率严格等于 10%。因此，我们通过 Python 进行计算求解并将结果置于支撑材料中，如附录 9 所示，表中给出了所需的 51 条测线对应的方向（以该方向的单位方向向量进行表示）以及计算所得的测线长度总和。

根据计算得到的结果，测线排布情况下的测线总长度最小值为 68.0145 海里，接近 68 海里。但同样地，类似于图11, 以此计算方法会造成边界区域的一定漏测，需要补充一部分测线长度来弥补侧漏。因此，满足条件的测线总和应严格大于 68 海里。

5.3.3 问题 3 的结果

通过构建平行布设模型与扇形布设模型，我们发现，若要满足题设条件，测线的总长度至少为 68 海里。68 海里的布设方法可以通过排布沿南北方向的测线得到，并在表5中给出了具体排布策略。我们运用了严格的数学方法证明了平行排布情况下，沿其

余方向排布的测线总海里数会大于 68 海里；并且应用计算数值结果说明了，扇形排布的策略下，满足条件的测线的最小总海里数也大于 68 海里。至此，我们得出结论：通过排布沿南北方向的测线可取得最优排布方案，至少需要 68 海里的测线总长以满足题设要求。

5.4 问题 4: 布设模型在海洋测深中的实际应用

对于问题 4 中给出的实际海域数据，我们首先将问题 4 给的数据进行可视化，得到了海底地形的等高线图，如图4(a)以及图4(b)所示，可以观察得出最大坡角出现在东南海域，我们使用 MATLAB 拟合得出的最大坡角为 3.37° 并且得到了各个位置的海底坡度。由于海底地形起伏相对较小、较为平整，可以沿用问题三中的模型辅助求解。将局部海底地形近似成平整的海底坡面，因此我们可以使用变角度几何多波束测深模型对测线上的某一点的探测宽度做出精度在两位小数以内（单位：m）的近似估计。

5.4.1 微分计算模型的建立

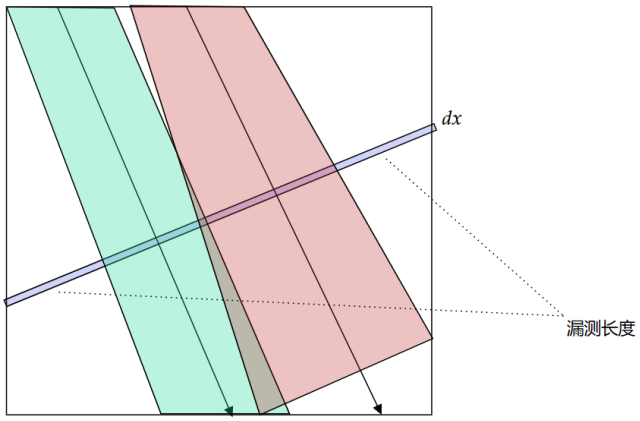


图 17: 微分计算模型示意图

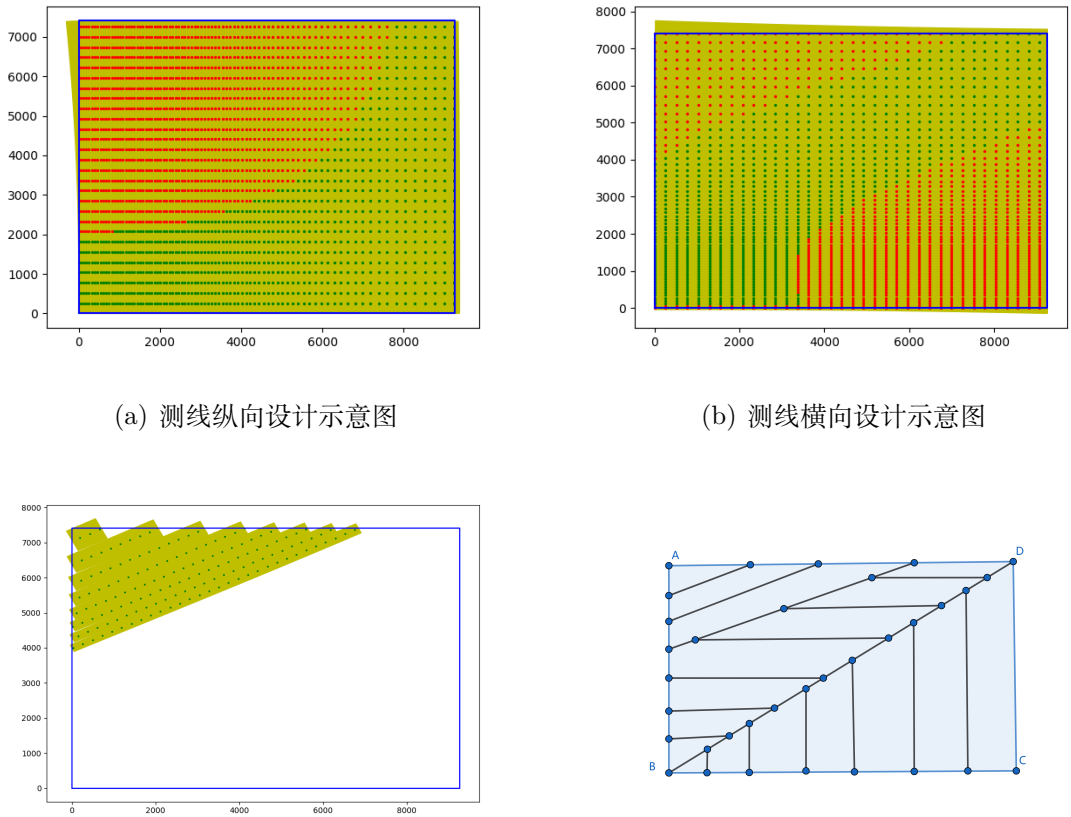
如图17所示，对于任意给定一组平行测线，我们考虑垂直于测线方向厚度为 dx 的狭长海域 L，使用最基本的几何运算可以计算得到该狭长海域上的测线总长度 $N(x)dx$，使用变角度几何关联多波束测深模型可以计算出一条侧线上给定位置的探测宽度，再使用基本几何关联多波束测深模型给出的重叠率公式，可以得出该位置与相邻测线的重叠率，接着就可以计算出该狭长海域内与相邻测线重叠率大于 20% 的测线总长度 $EN(x)dx$。接着利用 L 上每一条测线的坐标与测线相关联的探测宽度，计算出该狭长海域内未被探测区间覆盖的面积 $LUM(x)dx$. 最后将该海域内每一条这样的狭长海域中的测线长度、重测率大于 20% 的总长度、漏测面积累加在一起，得到如下积分式

$$\text{测线的总长度} = \int N(x)dx$$

$$\text{重叠率超过 20\% 部分的总长度} = \int EN(x)dx$$

$$\text{未被测量处的面积} = \int LUM(x)dx$$

由此即可建立关于测线总长度、重叠率超过 20% 的测线总长度、漏测面积的数值估算模型。



(a) 测线纵向设计示意图

(b) 测线横向设计示意图

(c) 斜向测量示意图

(d) 最终测线设计示意图

图 18: 测线设计示意图

5.4.2 方案设计

由海底纵深图4(b)，我们发现该海域各处深度变化较大，因此我们考虑分块设计

如图18(b)和图18(a)所示，我们利用 Python 绘出了漏测率为 0 的航道设计。在纵向设计中，测线均为东西方向，横向设计中，测线均为南北方向，红色点代表测线此处重叠率 > 20%，原点位于整片海域的西南方向。在纵向设计图中，测线总长度为 340 海里，重叠率 > 20% 的测线长度为 192 海里占比 56.5%。在横向设计图中，测线总长度为 300 海里，重叠率 > 20% 的测线长度为 165 海里，占比 55.0%，为了降低重叠率，我们考虑一部分区域采用横向设计，一部分区域采用纵向设计，一部分区域采用斜向设

计。由 Python（即附件中 model3.py）给出的斜向设计图如??所示。综合考虑三图，最终设计方案如图??所示，由 Python 计算出测线总长度为 291.52 海里，重叠率 > 20% 的测线长度为 0。漏测面积为 0.199 平方海里，占比为 0.995%。所有侧线的具体位置放在附录 10 中。

6 模型的分析与检验

6.1 基本、变角度几何关联多波束测深模型的分析与检验

6.1.1 基本、变角度几何关联多波束测深模型的灵敏度分析

基本几何关联多波束测深模型和变角度几何关联多波束测深模的分析的灵敏性在于，其对海底平面坡度测量角度、声波发射装置最大张角、实际测线之间间隔设置及测线运行方向的精度的依赖性。事实上，两个模型均给出了具体计算公式，通过海底平面坡度测量角度 α、声波发射装置最大张角 θ、实际测线之间间隔设置 d 及测线运行方向 β 等数据，基本几何关联多波束测深模型能够计算平行于海底坡面时测线位置海底深度、测量覆盖宽度以及相邻测线间的重叠率。根据求得的公式，当角度 α、θ 测量出现误差时会出现计算结果的偏差，特别是在角度较小时，小范围的角度测量误差也会引起计算结果的较大变化；对于角度 β、距离 d，小范围的误差对公式的计算结果的影响较小。

6.1.2 基本、变角度几何关联多波束测深模型的数值检验

对于问题一和问题二计算得到的基本几何关联多波束测深模型、变角度几何关联多波束测深模型进行数值检验，带入一些 α、θ、β、d 的特殊值进行检查，结果基本符合预测。相关代码在附录 1 中给出。

6.2 平行布设模型的分析与检验

6.2.1 平行布设模型的稳定性分析

平行排布模型的稳定性依赖于变角度几何关联多波束测深模型的计算精确性以及海底坡度变化、海域选择的影响。事实上，在本问中，我们仅考虑了 $\alpha = 1.5°$、题中给定水域的情形，在此条件下得到测线沿南北方向排列最优的结论。事实上，若 α 的弧度、水域选取位置发生改变，计算求得的 $f(\beta, x)$ 及 $g(\beta)$ 会随之发生改变，以该方法估计的测线总长下界不一定会大于测线沿南北方向排布时的情形，因此可能无法得出相同的结论、需要进一步确定最优排布策略以及相应的最优测线总长度。

6.2.2 平行布设模型计算结果的检验

我们需要验证模型对于题中问题求解的准确性。计算表5中各测线的测量范围，并进行可视化，结果如图19，其中探测区域由黄色矩形表示。由图可知，此排布模式的测

量范围能够覆盖目标水域；同时，由于设计测线排布方案时，我们考虑将相邻测线间的重复率调整为 10%，因此该设计方案满足题目对于重复率的要求。因此，平行布设模型计算求得的测线排布方式基本正确。

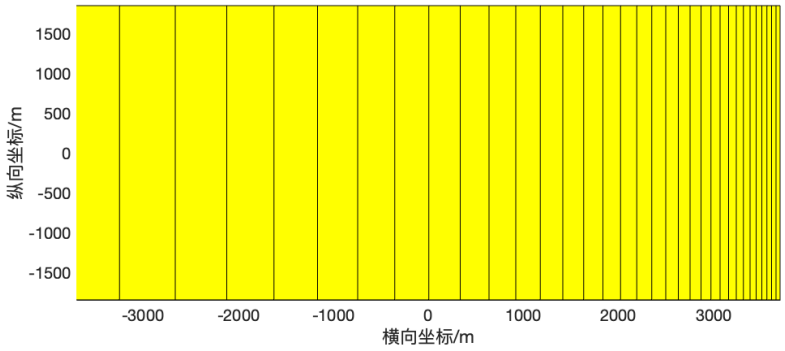


图 19: 平行布设模型结果检验

6.3 扇形布设模型的分析与检验

6.3.1 扇形布设模型的稳定性分析

扇形布设模型依赖于汇聚点位置的选取，其会受到海底坡度、海域范围等因素的影响。事实上，若汇聚点位置发生变化，其得到的结果也会受到影响。在本问中，选取的汇聚点位斜坡与海平面的交线上，且位于水域中心轴上。当海底地形、海域范围发生改变时，可能会影响汇聚点位置的选择，从而对模型的计算产生影响。

6.4 微分计算模型的分析与检验

6.4.1 微分计算模型的稳定性分析

微分计算模型具有较强的稳定性。事实上，由于海底平面的连续性且变化梯度较小，短距离变化对计算求得的数值结果影响不大。因此，我们可以认为，模型具有较好的稳定性、不容易受诸如航线运行偏差等外界因素带来的影响。

7 模型的评价、改进与推广

7.1 模型的优点

7.1.1 基本几何关联多波束测深模型的优点

在给定了测量船位置、海底坡度、海域中心深度的情况下，该模型能精确快速地求出探测点处的海水深度、覆盖宽度以及重叠率，具有算量小、精度高的特点。

7.1.2 变角度几何关联多波束测深模型优点

该模型在基本几何关联多波束测深模型的基础上改进，在给定了测线方向以及测量船位置的情况下，可以快速得出海底坡面上的探测宽度，同样具有算量小、精度高的特点。

7.1.3 平行布设模型的优点

- 在实际工作中，实施平行路径的测线较为容易，测量得到的数据准确性高。
- 平行布设方法具有简化得到的计算方法，可以使用相对较小的算量得到测线的分布、数量及总长度。
- 在海底坡度平缓且稳定的海域，平行布设模型能够得到沿等高线排布的结果，可以最大程度节省测线长度。
- 模型在应用求解的过程中，使用了严格的数学方法进行论证，模型的严谨性较强。

7.1.4 扇形布设法的优点

- 在海底坡面坡度相对稳定的情况下，布设者可以通过沿着等高线下降的方向适当布置测线，确保探测重叠率相对稳定。
- 对于海岛附近的海域，采取扇形布设法不仅可以保持重叠率的稳定、减少漏测的可能性，还能够节省测线总长度。

7.1.5 微分计算模型的优点

- 可以在海底不规则的情况下对于超过 20% 重复率的测线总长度、未测面积进行计算。
- 并不依赖于航线朝向与海底梯度之间的关系

7.2 模型的缺点

7.2.1 基本、变角度几何关联多波束测深模型的缺点

- 模型较为理想，将坡底地形视为均匀坡面，若海底地形复杂，则会产生计算误差。
- 实际使用中，对于海底坡度、声波发射装置张角的估计要求较高，若估计出现误差，可能会影响计算结果。

7.2.2 平行布设法的缺点

- 对于并非平行于海域边界的测线排布方案，在海域边界处容易存在漏测现象，需要进行进一步调整。
- 在海底地形陡峭的情况下，计算得到的结果不一定为所有方案中的最优结果。
- 在浅海地区需要大量铺设测线，而在深海区域容易造成重叠率高。

7.2.3 扇形布设法的缺点

- 在海域边界上存在漏测现象，需要进行部分调整。
- 对于侧线的设计需要花费较大算力。
- 非常依赖于测线与海底坡面之间的关系

7.2.4 微分计算模型的优点

- 需要考虑细分的区域，若区域选择过于疏则会导致精度下降；若区域选择过于密集则会导致计算量过大。

7.3 模型的改进

- 在给出了海底地形数据后，可以采用绘制截面图、使用解析几何的方法，解出探测束边界与海底的交点，再精确计算覆盖宽度，将基本几何关联多波束测深模型改进至可以测量曲面海底的覆盖范围。
- 可以将边界漏测的情况考虑在测线布设模型中，而不仅仅是使得相邻测线间的重叠率较低。
- 可以将扇形布设法中没有汇聚点的情况、汇聚点在在海平面各个位置的情况纳入考虑范围。
- 可以考虑将布设模型进行改进，使其能够应用于不同坡度、不同海域的情形。

7.4 模型的推广

- 海洋深度、探测宽度及重叠率的计算模型可应用于海洋勘测活动，能在海底管道布设、资源探测等工程中进行实际应用。
- 可以将模型应用于其他水域，例如湖泊、江河的勘探工作。
- 可以将模型用于其他探测方法，例如使用电磁波等媒介进行探测的工作中。

- 可以将模型应用至其他探测工作，例如使用飞机对地面地形或云层情况进行的探测，亦或太空中其他星球地形的探测。

参考文献

[1] 付盈盈. 多波束测深系统在河床变化分析中的应用. 城市道桥与防洪, 07:266–269+29, 2023.

[2] 张伟. 多波束测深系统在水下地形测量中的应用研究, 2009.

[3] 张旭, 叶小心, and 洪德玫. 多波束系统在长江航道测量中的测线布设方法研究. 中国水运. 航道科技, 01:52–55, 2017.

附录

支撑材料的文件列表
1、用 Python 编写的求解第一问、第二问及直线布设方法的计算模型 modol1.py
2、用 Python 编写的求解扇形布设方法的计算模型 modol2.py
3、用 Python 编写的求解微分计算模型及第四问测线规划的计算模型 modol3.py
4、用 MATLAB 编写的绘制第一问示意图的程序 Map1.m
5、用 MATLAB 编写的绘制直线布设模型结果检验示意图的程序 Map2.m
6、用 MATLAB 编写的绘制绘制第四问海域地形示意图 Map3.m
7、问题 1 计算结果 result1.xlsx
8、问题 2 计算结果 result2.xlsx
9、扇形布设策略计算结果 result3-2.xlsx
10、最终方案所有测线具体位置.txt

附录 1：第一问、第二问及直线布设方法的计算模型（model1.py）

```
import math
import sympy
import numpy as np
import matplotlib.pyplot as plt


#################### 问题一
# 深度
def simple_deep(D_0, x, d, alpha):
    return D_0 - x * d * math.tan(alpha)


# 探测宽度
def simple_W(D, x, d, theta, alpha):
    return 2 * simple_deep(D, x, d, alpha) * (math.cos(alpha) ** 2) * math.sin(theta) / (
            math.cos(theta) + math.cos(2 * alpha))


#
def simple_mu(D, x, d, theta, alpha):
    return (D - (d / 2) * (1 / math.tan(theta / 2)) - d * (x - 0.5) * math.tan(alpha)) / (D - x * d
        * math.tan(alpha))


D = 70
d = 200
t = 120 / 180 * math.pi
a = 1.5 / 180 * math.pi
# a = math.atan((84.80 - 24.17)/((5.00 - 1.64)*1852))
hl2m = 1852
```

```
print('位置   \t深度     \t覆盖宽度       \t重叠率')
print('800', '\t', simple_deep(D, -4, d, a), '\t', simple_W(D, -4, d, t, a), '\t', '-')

for i in range(-3, 5):
    print('{}00'.format(i * 2), '\t', simple_deep(D, i, d, a), '\t', simple_W(D, i, d, t, a), '\t',
          simple_mu(D, i, d, t, a))

print('带上beta')


# 问题2
def Coverage_W(D, length, theta, alpha, beta):
    gamma = math.acos(1 / (math.sqrt(1 + (math.tan(alpha) * math.sin(beta)) ** 2)))
    D_1 = D + length * math.cos(beta) * math.tan(alpha)
    return simple_W(D_1, 0, 0, theta, gamma)


b = math.pi / 2

for j in range(0, 8):
    s = '' + str( round(j * 45, 2)) + '\t'
    for i in range(0, 8):
        s += '\t'
        s += str( round(Coverage_W(120, i * 0.3 * hl2m, t, a, math.pi / 4 * j), 2))
    print(s)

print(Coverage_W(110, 0, t, a, 0))
print(Coverage_W(110, 0, t, a, math.pi / 2))

print('西岸深度', 110 + 2 * hl2m * math.tan(a))
print('东岸深度', 110 - 2 * hl2m * math.tan(a))

print(22.51 * math.tan(a))
print(Coverage_W(158, 0, t, a, 0))
print('最深处的探测宽度', Coverage_W(110 + 2 * hl2m * math.tan(a), 0, t, a, 0))


def defInt_trapezoid(f, a, b, N):
    # trapezoidal rule
    result = 0;
    FXa, FXb = [], [];
    Xn = []
    dx = abs(b - a) / N
    while a < b:
        result += (f(a) + f(a + dx)) * dx / 2
        FXa += [f(a)];
        FXb += [f(a + dx)]
        Xn += [a]
        a += dx
    return result
```

```
def func(x):
    return  int((2 * hl2m - Coverage_W(110 + x * math.tan(a), 0, t, a, 0)) / (
            Coverage_W(110 + x * math.tan(a), 0, t, a, 0) * 0.9) - 0.0000001) + 1 + 1


print('东西方向测线下届', defInt_trapezoid(func, - 2 * hl2m, 2 * hl2m, 100000) / hl2m)

# x = [i for i in range(-2 * hl2m, 2*hl2m)]
# y = [func(i) for i in x]
# line = plt.plot(x,y)
# plt.setp(line, color='r')
# plt.show()


print('算算每条测线的深度')


def left2rightD(D, theta, alpha):
    return D * (math.cos(alpha - theta / 2) / math.cos(alpha + theta / 2))


def left2midD(lD, theta, alpha):
    return lD * math.cos(alpha - theta / 2) / (math.cos(alpha) * math.cos(theta / 2))


v = 110 - 2 * hl2m * math.tan(a)-0.01
c = 0
while True:

    c += 1
    h = left2rightD(v, t, a)
    # print(c, v, left2midD(v, t, a), h)
    print(c, '\t', (110 - left2midD(v, t, a))/math.tan(a)/hl2m, '\t', left2midD(v, t, a))
    if (h > 110 + 2 * hl2m * math.tan(a)):

        break
    v = v  + (h-v)*0.9
print('总共需要航线数量:', c)

def right2left(D, theta, alpha):
    return D / ((math.cos(alpha - theta / 2) / math.cos(alpha + theta / 2)))

v = 110 + 2 * hl2m * math.tan(a)+0.01
c = 0
while True:

    c += 1
    h = right2left(v, t, a)
```

```
    # print(c, v, left2midD(v, t, a), h)
    print(c, '\t', round((110 - left2midD(h, t, a))/math.tan(a), 2), '\t',
        round(left2midD(h, t, a), 2))
    if (h < 110 - 2 * hl2m * math.tan(a)):

        break
    v = v  + (h-v)*0.9
print('总共需要航线数量:', c)


# 将计算最小航线数的公式模块化
def minCountOfLines(Dmin, Dmax, theta, alpha):
    if alpha == 0: print('warning')
    v = Dmin
    c = 0
    while True:

        c += 1
        h = left2rightD(v, theta, alpha)
        if (h > Dmax):
            break
        v = v + (h - v) * 0.9
    return c


w = hl2m * 4
h = hl2m * 2

print('最少航线数', minCountOfLines(110 - 2 * hl2m * math.tan(a), 110 + 2 * hl2m * math.tan(a), t, a
    ))


def totalLength(beta):  # 假设取值范围是[0,pi]
    if beta == 0 or beta == math.pi: return defInt_trapezoid(func, - 2 * hl2m, 2 * hl2m, 100000) /
        hl2m
    if beta == math.pi / 2: return minCountOfLines(110 - 2 * hl2m * math.tan(a), 110 + 2 * hl2m *
        math.tan(a), t, a) * 2
    gamma = math.acos(1 / math.sqrt(1 + (math.tan(a) * math.sin(beta)) ** 2))
    m_beta = beta
    if m_beta > math.pi / 2: m_beta = math.pi - beta

    # 需要求得与边界交点深度
    def findDminDmax(x):
        # 考虑往浅的方向走一定会比往深的方向走更浅

        x1 = hl2m * math.tan(m_beta) + x / math.cos(m_beta)
        x2 = -hl2m * math.tan(m_beta) + x / math.cos(m_beta)
        if x1 > 2 * hl2m: x1 = 2 * hl2m
        if x2 < -2 * hl2m: x2 = -2 * hl2m
        return (110 - x1 * math.tan(a), 110 - x2 * math.tan(a))
```

```
    def func1(x):

        D = findDminDmax(x)
        if D[0] >= D[1]: return 0
        return minCountOfLines(D[0], D[1], t, gamma)

    print(beta, func1(0))
    # x = list(np.arange(-3*hl2m, 3*hl2m, 1))
    # y = [func1(i) for i in x]
    # plt.plot(x,y)
    # plt.show()

    return defInt_trapezoid(func1, -3 * hl2m, 3 * hl2m, 10000) / hl2m


print(totalLength(math.pi/2 + 0.000001))


# 画一下总航线距离的下届与beta角的关系
x = list(np.arange(81/180 * math.pi, 87/180 * math.pi, 0.0001))
y = [totalLength(i) for i in x]
# y0 = [i[0] for i in y]
# y1 = [i[1] for i in y]
const = [68 for i in y]
d = [i/math.pi * 180 for i in x]
y_plus = [y[i] + 3*math.cos(x[i]) for i in range( len(x))]
inter_x = []
# for i in range(len(x)-1):
#     if y[i] <= 68 <= y[i+1] or y[i+1] <= 68 <= y[i]:
#         inter_x.append(d[i])
# print(inter_x)

# for x_ in inter_x:
#     plt.scatter(x_, 68, c='r', s = 3)
plt.plot(d, y)
plt.plot(d, y_plus)
plt.plot(d, const)

plt.show()

##########################################################################


def CountLinesAt(beta):  # 假设取值范围是[0,pi]
    if beta == 0 or beta == math.pi: return defInt_trapezoid(func, - 2 * hl2m, 2 * hl2m, 100000) /
        hl2m
    if beta == math.pi / 2: return minCountOfLines(110 - 2 * hl2m * math.tan(a), 110 + 2 * hl2m *
        math.tan(a), t, a) * 2
    gamma = math.acos(1 / math.sqrt(1 + (math.tan(a) * math.sin(beta)) ** 2))
```

```
    m_beta = beta
    if m_beta > math.pi / 2: m_beta = math.pi - beta

    # 需要求得与边界交点深度
    def findDminDmax(x):
        # 考虑往浅的方向走一定会比往深的方向走更浅

        x1 = hl2m * math.tan(m_beta) + x / math.cos(m_beta)
        x2 = -hl2m * math.tan(m_beta) + x / math.cos(m_beta)
        if x1 > 2 * hl2m: x1 = 2 * hl2m
        if x2 < -2 * hl2m: x2 = -2 * hl2m
        return (110 - x1 * math.tan(a), 110 - x2 * math.tan(a))

    def func1(x):

        D = findDminDmax(x)
        if D[0] >= D[1]: return 0
        return minCountOfLines(D[0], D[1], t, gamma)

    print(func1(0))
    x = list(np.arange(-2 * hl2m, 2 * hl2m, 1))
    y = [func1(i) for i in x]
    plt.plot(x, y)
    plt.show()


# print(totalLength(1.42103))
# CountLinesAt(1.42103)
#
# print(2*hl2m*math.tan(math.pi/2 - 1.42103))

## 想办法找到
# lbeta = 0.45*math.pi
# rbeta = 0.46*math.pi
# while rbeta > lbeta + 0.0001:
#     if CountLinesAtZero(0.5*(lbeta+rbeta)) == 34: rbeta = 0.5*(lbeta+rbeta)
#     if CountLinesAtZero(0.5*(lbeta+rbeta)) == 35: lbeta = 0.5*(lbeta+rbeta)
#
# print(lbeta, rbeta)


# 给出一个方向和个测线坐标，然后列出测线的长度、画出测线测量范围
w2e = 4 * hl2m
n2s = 2 * hl2m


def GetStartEndPointOfLine(y, beta):  # 假设beta取值[0,pi/2]，默认xy成一右手系
    if beta == 0: return [(-2 * hl2m, y), (2 * hl2m, y)]
    ans = [(-hl2m / math.tan(beta) - y * math.sin(beta), -hl2m),
           (+hl2m / math.tan(beta) - y * math.sin(beta), +hl2m)]
```

```
    if ans[0][0] < -2 * hl2m: ans[0] = (-2 * hl2m, -2 * hl2m * math.tan(beta) + y / math.cos(beta))
    if ans[1][0] > 2 * hl2m: ans[1] = (2 * hl2m, 2 * hl2m * math.tan(beta) + y / math.cos(beta))
    return ans
```

附录 2：扇形布设方法的计算模型

```
import math


#################### 问题一
# 深度
def simple_deep(D_0, x, d, alpha):
    return D_0 - x * d * math.tan(alpha)


# 探测宽度
def simple_W(D, x, d, theta, alpha):
    return 2 * simple_deep(D, x, d, alpha) * (math.cos(alpha) ** 2) * math.sin(theta) / (
            math.cos(theta) + math.cos(2 * alpha))


#
def simple_mu(D, N, d, theta, alpha):
    return (D - (d / 2) * (1 / math.tan(theta / 2)) - d * (N - 0.5) * math.tan(alpha)) / (D - N * d
        * math.tan(alpha))


d = 200
t = 120 / 180 * math.pi
a = 1.5 / 180 * math.pi
hl2m = 1852


# 问题2
def Coverage_W(D, length, theta, alpha, beta):
    gamma = math.acos(1 / (math.sqrt(1 + (math.tan(alpha) * math.sin(beta)) ** 2)))
    D_1 = D + length * math.cos(beta) * math.tan(alpha)
    return simple_W(D_1, 0, 0, theta, gamma)


def LowerBoundOfExtra(D, theta, alpha, beta):
    gamma = math.acos(1 / (math.sqrt(1 + (math.tan(alpha) * math.sin(beta)) ** 2)))
    return (D * math.sin(theta) / math.sin(math.pi / 2 - theta / 2 + gamma) - 0.2 * simple_W(D, 0,
        0, theta,
                                                                                            gamma))
                                                                                                /
                                                                                            math
                                                                                            .
                                                                                            tan
                                                                                            (
```

```
                                                                    beta
                                                                    )


#TODO 考虑开发一个函数，输入海底边界，海底高度函数，与测线位置方向，返回测量端点


################ 辐射状测线
# 考虑

def detectAngle(beta, alpha, theta):
    gamma = math.acos(1 / (math.sqrt(1 + (math.tan(alpha) * math.sin(beta)) ** 2)))
    ans = (math.tan(alpha) * math.cos(beta) * math.sin(theta / 2) * math.cos(gamma)) / math.cos(
        gamma + theta / 2) \
          + (math.tan(alpha) * math.cos(beta) * math.sin(theta / 2) * math.cos(gamma)) / math.cos(
            gamma - theta / 2)
    return ans


# x = list(np.arange(0, math.pi, 0.01))
# y = [detectAngle(i, a, t) for i in x]
# plt.plot(x, y)
# plt.show()
def drawPicture(start, end, step, func):
    x = list(np.arange(start, end, step))
    print( len(x))
    y = [func(i) for i in x]
    plt.plot(x, y)
    plt.show()


################## 算一下放射性线条
w, h = 2 * hl2m, hl2m
AlmostZero = 0.000001


def dist(p, q):
    return math.sqrt((p[0] - q[0]) ** 2 + (p[1] - q[1]) ** 2)


def getAngle(dir1, dir2):
    if type(dir1) != list: dir1 = dir1.ToList()
    if type(dir2) != list: dir2 = dir2.ToList()
    l1 = math.sqrt(dir1[0] ** 2 + dir1[1] ** 2)
    l2 = math.sqrt(dir2[0] ** 2 + dir2[1] ** 2)
    # print(l1,l2)
    return math.acos((dir1[0] * dir2[0] + dir1[1] * dir2[1]) / (l1 * l2))


# 简单的数值积分
```

```
def defInt_trapezoid(f, a, b, N):
    # trapezoidal rule
    result = 0;
    FXa, FXb = [], [];
    Xn = []
    dx = abs(b - a) / N
    while a < b:
        result += (f(a) + f(a + dx)) * dx / 2
        FXa += [f(a)];
        FXb += [f(a + dx)]
        Xn += [a]
        a += dx
    return result


# 将习惯性的矢量方向换算成beta角
def direct2beta(dx, dy):
    if dx == 0 and dy == 0:
        print('零向量没有方向')
    if dx == 0:
        return math.pi / 2
    ans = math.acos(-dx / math.sqrt(dx ** 2 + dy ** 2))
    if dy > 0:
        ans = - ans
    return ans


class Point:
    def __init__(self, x=0., y=0.):
        self.x = x
        self.y = y

    def __str__(self):
        return "Point({}, {})".format(self.x, self.y)

    def distance(self, point):
        return dist((self.x, self.y), (point.x, point.y))


class Circle:
    def __init__(self, x=0., y=0., R=1.):
        self.x = x
        self.y = y
        self.R = R

    def isExist(self, point):
        if abs(dist((point.x, point.y), (self.x, self.y)) - self.R) < AlmostZero:
            return True
        return False
```

```
    def findBeta(self, point):
        if self.isExist(point):
            return direct2beta(point.x - self.x, point.y - self.y)


class Rec:
    def __init__(self, x1, y1, x2, y2):
        self.x1 = min(x1, x2)
        self.y1 = min(y1, y2)
        self.x2 = max(x1, x2)
        self.y2 = max(y1, y2)

    def isExist(self, point):
        if abs(point.x - self.x1) < AlmostZero and self.y1 <= point.y <= self.y2:
            return True
        if abs(point.x - self.x2) < AlmostZero and self.y1 <= point.y <= self.y2:
            return True
        if abs(point.y - self.y1) < AlmostZero and self.x1 <= point.x <= self.x2:
            return True
        if abs(point.y - self.y2) < AlmostZero and self.x1 <= point.x <= self.x2:
            return True
        return False


def intersectionOfCirAndRec(circle, rec):
    ans = []
    if abs(circle.x - rec.x1) == circle.R and rec.y1 <= circle.y <= rec.y2:
        ans.append(Point(rec.x1, circle.y))
    if abs(circle.x - rec.x2) == circle.R and rec.y1 <= circle.y <= rec.y2:
        ans.append(Point(rec.x2, circle.y))
    if abs(circle.y - rec.y1) == circle.R and rec.x1 <= circle.x <= rec.x2:
        ans.append(Point(circle.x, rec.y1))
    if abs(circle.y - rec.y2) == circle.R and rec.x1 <= circle.x <= rec.x2:
        ans.append(Point(circle.x, rec.y2))

    if abs(circle.x - rec.x1) < circle.R:
        p1 = Point(rec.x1, circle.y + math.sqrt(circle.R ** 2 - (circle.x - rec.x1) ** 2))
        p2 = Point(rec.x1, circle.y - math.sqrt(circle.R ** 2 - (circle.x - rec.x1) ** 2))
        if rec.isExist(p1):
            ans.append(p1)
        if rec.isExist(p2):
            ans.append(p2)
    if abs(circle.x - rec.x2) < circle.R:
        p1 = Point(rec.x2, circle.y + math.sqrt(circle.R ** 2 - (circle.x - rec.x2) ** 2))
        p2 = Point(rec.x2, circle.y - math.sqrt(circle.R ** 2 - (circle.x - rec.x2) ** 2))
        if rec.isExist(p1):
            ans.append(p1)
        if rec.isExist(p2):
            ans.append(p2)
    if abs(circle.y - rec.y1) < circle.R:
```

```
        p1 = Point(circle.x + math.sqrt(circle.R ** 2 - (circle.y - rec.y1) ** 2), rec.y1)
        p2 = Point(circle.x - math.sqrt(circle.R ** 2 - (circle.y - rec.y1) ** 2), rec.y1)
        if rec.isExist(p1):
            ans.append(p1)
        if rec.isExist(p2):
            ans.append(p2)
    if abs(circle.y - rec.y2) < circle.R:
        p1 = Point(circle.x + math.sqrt(circle.R ** 2 - (circle.y - rec.y2) ** 2), rec.y2)
        p2 = Point(circle.x - math.sqrt(circle.R ** 2 - (circle.y - rec.y2) ** 2), rec.y2)
        if rec.isExist(p1):
            ans.append(p1)
        if rec.isExist(p2):
            ans.append(p2)

    return ans


print("单位正方形与单位圆的交点：", [
    str(p) for p in intersectionOfCirAndRec(Circle(0, 0, 1), Rec(0, 0, 1, 1))])
```

```
print(direct2beta(1, -1) / math.pi)


def minLinesOfArc(beta1, beta2):  # 假设 beta1 < beta2 <= pi
    minDetectAngle = detectAngle(beta2, a, t)
    return math.ceil((beta2 - beta1) / minDetectAngle)


centerLine_x = 110 / math.tan(a)
sea_rec = Rec(-2 * hl2m, -hl2m, 2 * hl2m, hl2m)


def minLinesOfR(center, R):
    inter_ps = intersectionOfCirAndRec(Circle(center.x, center.y, R), sea_rec)
    inter_ps.sort(key=lambda p: p.y)
    if len(inter_ps) % 2 != 0:
        print('交点个数为奇数个，要小心', [str(p) for p in inter_ps])
    ans = 0
    for i in range(len(inter_ps) // 2):
        p1 = inter_ps[i]
        p2 = inter_ps[i + 1]
        if p1.y <= center.y <= p2.y:
            ans += math.ceil(
                getAngle((p1.x - center.x, p1.y - center.y), (p2.x - center.x, p2.y - center.y)) / (
                    detectAngle(
                    0, a, t) * 0.9))
        else:
            ans += math.ceil(
                getAngle((p1.x - center.x, p1.y - center.y), (p2.x - center.x, p2.y - center.y)) / (
                    detectAngle(
```

```
                    min(direct2beta(p1.x - center.x, p1.y - center.y), direct2beta(p2.x - center.x,
                        p2.y - center.y)),
                    a, t) * 0.9))
    return ans


# minLinesOfR(Point(0, 0), 1.5 * hl2m)
# drawPicture(0, 5 * hl2m, 1, lambda x: minLinesOfR(Point(centerLine_x, 0), x))


def minTotalLenthOfCenter(center_y):
    center = Point(centerLine_x, center_y)
    ans = defInt_trapezoid(lambda x: minLinesOfR(center, x), 0, 5 * hl2m, 10000)
    print(center_y / hl2m, ans / hl2m)  # 给出中间结果以确定程序正常运行
    return ans


# drawPicture(0, 2 * hl2m, 100, minTotalLenthOfCenter)


class Direction:
    def __init__(self, dx, dy):
        self.dx = dx
        self.dy = dy

    def normalized(self):
        l = math.sqrt(self.dx ** 2 + self.dy ** 2)
        return Direction(self.dx / l, self.dy / l)

    def dot(self, dir):
        return self.dx * dir.dx + self.dy * dir.dy

    def __str__(self):
        return '({}, {})'.format(self.dx, self.dy)

    def ToList(self):
        return [self.dx, self.dy]

class Line:
    def __init__(self, start_p, direction):
        self.start = start_p
        self.direction = direction


def intersectionOfLineAndRec(line, rec):
    ans = []
    if line.direction.dy != 0:
        p = Point((rec.y1 - line.start.y) * (line.direction.dx / line.direction.dy) + line.start.x,
            rec.y1)
        if rec.isExist(p): ans.append(p)
```

```
    if line.direction.dy != 0:
        p = Point((rec.y2 - line.start.y) * (line.direction.dx / line.direction.dy) + line.start.x,
            rec.y2)
        if rec.isExist(p): ans.append(p)
    if line.direction.dx != 0:
        p = Point(rec.x1, (rec.x1 - line.start.x) * (line.direction.dy / line.direction.dx) + line.
            start.y)
        if rec.isExist(p): ans.append(p)
    if line.direction.dx != 0:
        p = Point(rec.x2, (rec.x2 - line.start.x) * (line.direction.dy / line.direction.dx) + line.
            start.y)
        if rec.isExist(p): ans.append(p)
    return ans


class RadioactiveMeasure:
    def __init__(self, center, *directions):
        self.center = center
        self.directions = directions
        self.sort_direction()

    def TotalLength(self):
        ans = 0
        for dir_ in self.directions:
            inter_ps = intersectionOfLineAndRec(Line(self.center, dir_), sea_rec)
            if len(inter_ps) <= 1: continue
            ans += inter_ps[0].distance(inter_ps[1])
        return ans

    def sort_direction(self):
        self.directions = [dir_.normalized() for dir_ in self.directions]
        self.directions.sort(key=lambda i: i.dy)

    def showTable(self):
        alpha = a

        def phi1phi2(dir_):
            beta = direct2beta(dir_.dx, dir_.dy)
            gamma = math.acos(1 / (math.sqrt(1 + (math.tan(alpha) * math.sin(beta)) ** 2)))
            phi1 = (math.tan(a) * math.cos(beta) * math.sin(t / 2) * math.cos(gamma)) / math.cos(
                gamma + t / 2)
            phi2 = (math.tan(a) * math.cos(beta) * math.sin(t / 2) * math.cos(gamma)) / math.cos(
                gamma - t / 2)
            return phi1, phi2

        print('方向\t有效长度\t与上一条测线的重叠率\t与下一条测线的重叠率')
        for i in range( len(self.directions)):
            s = ''
            dir_ = self.directions[i]
            s += str(dir_) + '\t'
```

```
            inter_ps = intersectionOfLineAndRec(Line(self.center, dir_), sea_rec)
            if len(inter_ps) <= 1:
                s += '----'+ '\t'
            else:
                s+= str( round(inter_ps[0].distance(inter_ps[1])/hl2m, 2)) + '\t'

            phi = phi1phi2(dir_)
            if dir_.dy >= 0:
                delta_l = phi[1]
                delta_n = phi[0]
            else:
                delta_l = phi[0]
                delta_n = phi[1]

            # 与上一条的重叠率
            if i == 0:
                s+='----'+ '\t'
            else:
                dir_last = self.directions[i - 1]
                phi_last = phi1phi2(dir_last)
                if dir_.dy >= 0:
                    delta_last_n = phi_last[0]
                else:
                    delta_last_n = phi_last[1]

                eta_l = (delta_l + delta_last_n - getAngle(dir_last, dir_)) / (delta_l + delta_n)
                s+=str(eta_l)+ '\t'

            # 与下一条的重叠率
            if i == len(self.directions) - 1:
                s+='----'+ '\t'
            else:
                dir_next = self.directions[i + 1]
                phi_next = phi1phi2(dir_next)
                if dir_.dy >= 0:
                    delta_next_l = phi_next[1]
                else:
                    delta_next_l = phi_next[0]

                eta_n = (delta_n + delta_next_l - getAngle(dir_next, dir_)) / (delta_l + delta_n)
                s+= str(eta_n) + '\t'


            print(s)
        print('总航线长度', self.TotalLength()/hl2m)


rm = RadioactiveMeasure(Point(centerLine_x, 0 ),
                        Direction(-1,-3.76964),
```

```
Direction(-1,-3.463525),
Direction(-1,-3.180515),
Direction(-1,-2.91872),
Direction(-1,-2.676396),
Direction(-1,-2.45192),
Direction(-1,-2.243799),
Direction(-1,-2.050645),
Direction(-1,-1.87117),
Direction(-1,-1.704175),
Direction(-1,-1.548546),
Direction(-1,-1.403246),
Direction(-1,-1.267305),
Direction(-1,-1.139813),
Direction(-1,-1.019923),
Direction(-1,-0.906832),
Direction(-1,-0.799784),
Direction(-1,-0.698066),
Direction(-1,-0.600998),
Direction(-1,-0.50793),
Direction(-1,-0.41824),
Direction(-1,-0.331328),
Direction(-1,-0.24661),
Direction(-1,-0.16352),
Direction(-1,-0.0815),
Direction(-1,0),
Direction(-1,0.0815),
Direction(-1,0.16352),
Direction(-1,0.24661),
Direction(-1,0.331328),
Direction(-1,0.41824),
Direction(-1,0.50793),
Direction(-1,0.600998),
Direction(-1,0.698066),
Direction(-1,0.799784),
Direction(-1,0.906832),
Direction(-1,1.019923),
Direction(-1,1.139813),
Direction(-1,1.267305),
Direction(-1,1.403246),
Direction(-1,1.548546),
Direction(-1,1.704175),
Direction(-1,1.87117),
Direction(-1,2.050645),
Direction(-1,2.243799),
Direction(-1,2.45192),
Direction(-1,2.676396),
Direction(-1,2.91872),
Direction(-1,3.180515),
Direction(-1,3.463525),
Direction(-1,3.76964)
```

```
                                )
rm.showTable()
```

附录 3：微分计算模型及第四问测线规划的计算模型

```
import math
import sympy
import numpy as np
import matplotlib.pyplot as plt

t = 120 / 180 * math.pi
hl2m = 1852
# w, h = 2 * hl2m, hl2m
AlmostZero = 0.000001

with open("附件.txt", "r", encoding='utf-8') as f:  # 打开文本
    data = f.read()  # 读取文本
    # print(data)

data_matrix = [t.split('\t') for t in data.split('\n')]
data_matrix = data_matrix[:-1]
dw = len(data_matrix)
dh = len(data_matrix[0])

# print(len(data_matrix), len(data_matrix[0]))

with open("dx.txt", "r", encoding='utf-8') as f:  # 打开文本
    dxdata = f.read()  # 读取文本
    # print(data)

dx_matrix = [t.split('\t') for t in dxdata.split('\n')]
dx_matrix = dx_matrix[:-1]

with open("dy.txt", "r", encoding='utf-8') as f:  # 打开文本
    dydata = f.read()  # 读取文本
    # print(data)

dy_matrix = [t.split('\t') for t in dydata.split('\n')]
dy_matrix = dy_matrix[:-1]


def DeepOfSea(x, y):
    if 0 <= x <= 5 * hl2m and 0 <= y <= 4 * hl2m:
        return float(data_matrix[ int(x / (5 * hl2m) * dw)][ int(y / (4 * hl2m) * dh)])
```

```
    return 0


def GradientOfSea(x, y):
    if 0 <= x <= 5 * hl2m and 0 <= y <= 4 * hl2m:
        return float(dx_matrix[int(x / (5 * hl2m) * dw)][int(y / (4 * hl2m) * dh)]), float(
            dy_matrix[int(x / (5 * hl2m) * dw)][int(y / (4 * hl2m) * dh)])
    return 0, 0


def alphaOfSeaAmongDir(x, y, dir_):
    dx, dy = GradientOfSea(x, y)
    dir___ = dir_.normalized()
    alpha = math.atan(dx * dir___.dx + dy * dir___.dy)
    return abs(alpha)


# def pltSea():
#     for i in range(dw//10):
#         for j in range(dh//10):
#             plt.scatter(i,j,float(data_matrix[i][j]))


def simple_W(D, theta, alpha):
    return 2 * D * (math.cos(alpha) ** 2) * math.sin(theta) / (
            math.cos(theta) + math.cos(2 * alpha))


def dist(p, q):
    return math.sqrt((p[0] - q[0]) ** 2 + (p[1] - q[1]) ** 2)


def getAngle(dir1, dir2):
    if type(dir1) != list: dir1 = dir1.ToList()
    if type(dir2) != list: dir2 = dir2.ToList()
    l1 = math.sqrt(dir1[0] ** 2 + dir1[1] ** 2)
    l2 = math.sqrt(dir2[0] ** 2 + dir2[1] ** 2)
    # print(l1,l2)
    return math.acos((dir1[0] * dir2[0] + dir1[1] * dir2[1]) / (l1 * l2))


# 简单的数值积分
def defInt_trapezoid(f, a, b, N):
    # trapezoidal rule
    result = 0
    dx = abs(b - a) / N
    while a < b:
        result += (f(a) + f(a + dx)) * dx / 2

        a += dx
```

```
    return result


# 将习惯性的矢量方向换算成beta角
def direct2beta(dx, dy):
    if dx == 0 and dy == 0:
        print('零向量没有方向')
    if dx == 0:
        return math.pi / 2
    ans = math.acos(-dx / math.sqrt(dx ** 2 + dy ** 2))
    if dy > 0:
        ans = - ans
    return ans


class Point:
    def __init__(self, x=0., y=0.):
        self.x = x
        self.y = y

    def __str__(self):
        return "Point({}, {})".format(self.x, self.y)

    def distance(self, point):
        return dist((self.x, self.y), (point.x, point.y))


class Circle:
    def __init__(self, x=0., y=0., R=1.):
        self.x = x
        self.y = y
        self.R = R

    def isExist(self, point):
        if abs(dist((point.x, point.y), (self.x, self.y)) - self.R) < AlmostZero:
            return True
        return False

    def findBeta(self, point):
        if self.isExist(point):
            return direct2beta(point.x - self.x, point.y - self.y)


class Rec:
    def __init__(self, x1, y1, x2, y2):
        self.x1 = min(x1, x2)
        self.y1 = min(y1, y2)
        self.x2 = max(x1, x2)
        self.y2 = max(y1, y2)
```

```
    def isExist(self, point):
        if abs(point.x - self.x1) < AlmostZero and self.y1 <= point.y <= self.y2:
            return True
        if abs(point.x - self.x2) < AlmostZero and self.y1 <= point.y <= self.y2:
            return True
        if abs(point.y - self.y1) < AlmostZero and self.x1 <= point.x <= self.x2:
            return True
        if abs(point.y - self.y2) < AlmostZero and self.x1 <= point.x <= self.x2:
            return True
        return False

    def isInside(self, point):
        return (self.x1 <= point.x <= self.x2) and (self.y1 <= point.y <= self.y2)


class Tri:
    def __init__(self, p1, p2, p3):
        self.p1 = p1
        self.p2 = p2
        self.p3 = p3


class Direction:
    def __init__(self, dx, dy):
        self.dx = dx
        self.dy = dy

    def normalized(self):
        l = math.sqrt(self.dx ** 2 + self.dy ** 2)
        return Direction(self.dx / l, self.dy / l)

    def dot(self, dir):
        return self.dx * dir.dx + self.dy * dir.dy

    def __str__(self):
        return '({}, {})'.format(self.dx, self.dy)

    def ToList(self):
        return [self.dx, self.dy]


class Line:
    def __init__(self, start_p, direction):
        self.start = start_p
        self.direction = direction


class Segment:
    def __init__(self, start_p, end_p):
        self.start = start_p
```

```
        self.end = end_p
        self.length = start_p.distance(end_p)

    def getDirection(self):
        return Direction(self.end.x - self.start.x, self.end.y - self.start.y)

    def isExist(self, point):
        return  abs(self.start.distance(point) + self.end.distance(point) - self.length) < AlmostZero

    def __str__(self):
        return 'Line: ({}, {})----({}, {})'.
            format(self.start.x, self.start.y, self.end.x, self.end.y)


def intersectionOfCirAndRec(circle, rec):
    ans = []
    if abs(circle.x - rec.x1) == circle.R and rec.y1 <= circle.y <= rec.y2:
        ans.append(Point(rec.x1, circle.y))
    if abs(circle.x - rec.x2) == circle.R and rec.y1 <= circle.y <= rec.y2:
        ans.append(Point(rec.x2, circle.y))
    if abs(circle.y - rec.y1) == circle.R and rec.x1 <= circle.x <= rec.x2:
        ans.append(Point(circle.x, rec.y1))
    if abs(circle.y - rec.y2) == circle.R and rec.x1 <= circle.x <= rec.x2:
        ans.append(Point(circle.x, rec.y2))

    if abs(circle.x - rec.x1) < circle.R:
        p1 = Point(rec.x1, circle.y + math.sqrt(circle.R ** 2 - (circle.x - rec.x1) ** 2))
        p2 = Point(rec.x1, circle.y - math.sqrt(circle.R ** 2 - (circle.x - rec.x1) ** 2))
        if rec.isExist(p1):
            ans.append(p1)
        if rec.isExist(p2):
            ans.append(p2)
    if abs(circle.x - rec.x2) < circle.R:
        p1 = Point(rec.x2, circle.y + math.sqrt(circle.R ** 2 - (circle.x - rec.x2) ** 2))
        p2 = Point(rec.x2, circle.y - math.sqrt(circle.R ** 2 - (circle.x - rec.x2) ** 2))
        if rec.isExist(p1):
            ans.append(p1)
        if rec.isExist(p2):
            ans.append(p2)
    if abs(circle.y - rec.y1) < circle.R:
        p1 = Point(circle.x + math.sqrt(circle.R ** 2 - (circle.y - rec.y1) ** 2), rec.y1)
        p2 = Point(circle.x - math.sqrt(circle.R ** 2 - (circle.y - rec.y1) ** 2), rec.y1)
        if rec.isExist(p1):
            ans.append(p1)
        if rec.isExist(p2):
            ans.append(p2)
    if abs(circle.y - rec.y2) < circle.R:
        p1 = Point(circle.x + math.sqrt(circle.R ** 2 - (circle.y - rec.y2) ** 2), rec.y2)
        p2 = Point(circle.x - math.sqrt(circle.R ** 2 - (circle.y - rec.y2) ** 2), rec.y2)
        if rec.isExist(p1):
```

```
            ans.append(p1)
        if rec.isExist(p2):
            ans.append(p2)

    return ans


def intersectionOfLineAndRec(line, rec):
    ans = []
    if line.direction.dy != 0:
        p = Point((rec.y1 - line.start.y) * (line.direction.dx / line.direction.dy) + line.start.x,
            rec.y1)
        if rec.isExist(p): ans.append(p)
    if line.direction.dy != 0:
        p = Point((rec.y2 - line.start.y) * (line.direction.dx / line.direction.dy) + line.start.x,
            rec.y2)
        if rec.isExist(p): ans.append(p)
    if line.direction.dx != 0:
        p = Point(rec.x1, (rec.x1 - line.start.x) * (line.direction.dy / line.direction.dx) + line.
            start.y)
        if rec.isExist(p): ans.append(p)
    if line.direction.dx != 0:
        p = Point(rec.x2, (rec.x2 - line.start.x) * (line.direction.dy / line.direction.dx) + line.
            start.y)
        if rec.isExist(p): ans.append(p)
    return ans


def intersectionOfSegmentAndRec(segment, rec):
    ans = []
    for p in intersectionOfLineAndRec(Line(segment.start, segment.getDirection()), rec):
        if segment.isExist(p):
            ans.append(p)
    return ans


class Measure:
    def __init__(self, dir_, yList):
        self.x_direction = dir_.normalized()
        self.y_direction = Direction(-dir_.dy, dir_.dx).normalized()
        self.yList = yList
        self.yList.sort()
        self.n = len(self.yList)

    def setyList(self, yList):
        self.yList = yList
        self.yList.sort()
        self.n = len(self.yList)

    def getDetecty(self, x):
```

```
        detect_y = []
        for y in self.yList:
            x_, y_ = self.getRealCoordinate(x, y)
            W = simple_W(DeepOfSea(x_, y_), t, alphaOfSeaAmongDir(x_, y_, self.y_direction))
            detect_y.append((y - 0.5 * W,
                             y + 0.5 * W))
        return detect_y

    def getRealPoint(self, x, y):
        return Point(x * self.x_direction.dx + y * self.y_direction.dx,
                     x * self.x_direction.dy + y * self.y_direction.dy)

    def getRealCoordinate(self, x, y):
        return x * self.x_direction.dx + y * self.y_direction.dx, x * self.x_direction.dy + y * self
            .y_direction.dy

    def real2Local_y(self, x, y):
        return x * self.y_direction.dx + y * self.y_direction.dy

    def real2Local_x(self, x, y):
        return x * self.x_direction.dx + y * self.x_direction.dy

    def real2Local(self, x, y):
        return self.real2Local_x(x, y), self.real2Local_y(x, y)

    def ifCoincideIllegal(self, x, minRate=0.1, maxRate=0.2):

        detect_y = self.getDetecty(x)
        ans = [True for y in self.yList]
        for i in range(0, len(detect_y) - 1):
            if (not (detect_y[i][1] - detect_y[i][0]) == 0):
                ans[i] &= ((detect_y[i][1] - detect_y[i + 1][0]) / (
                        detect_y[i][1] - detect_y[i][0]) <= maxRate)
        for i in range(1, len(detect_y)):
            if (not (detect_y[i][1] - detect_y[i][0]) == 0):
                ans[i] &= ((detect_y[i - 1][1] - detect_y[i][0]) / (
                        detect_y[i][1] - detect_y[i][0]) <= maxRate)
        return ans

    def CoincideMin(self, xmin, xmax, Nx):
        x = xmin
        dx = (xmax - xmin) / Nx
        ans = [[1, 1] for y in self.yList]
        while x < xmax:
            detect_y = self.getDetecty(x)
            inOcean = self.ifInOcean(x)

            for i in range(0, len(detect_y) - 1):
                if inOcean[i]:
                    ans[i][1] = min(((detect_y[i][1] - detect_y[i + 1][0]) / (
```

```
                        detect_y[i][1] - detect_y[i][0]), ans[i][1]))
            for i in range(1, len(detect_y)):
                if inOcean[i]:
                    ans[i][0] = min(((detect_y[i - 1][1] - detect_y[i][0]) / (
                            detect_y[i][1] - detect_y[i][0]), ans[i][0]))

            x += dx
        return ans

    def ifInOcean(self, x):
        segment_ps = intersectionOfLineAndRec(Line(self.getRealPoint(x, 0), self.y_direction),
            sea_rec)
        if len(segment_ps) <= 1: return [False for i in range( len(self.yList))]
        segment = Segment(segment_ps[0], segment_ps[1])
        return [segment.isExist(self.getRealPoint(x, y)) for y in self.yList]

    def pltLines(self, xmin, xmax, N):
        x = xmin
        dx = (xmax - xmin) / N
        while x < xmax:
            inOcean = self.ifInOcean(x)
            for y, inOcean_ in zip(self.yList, inOcean):
                if not inOcean_: continue
                x_, y_ = self.getRealCoordinate(x, y)
                plt.scatter(x_, y_, s=2, c='grey')
            x += dx

    def pointBeDetected(self, x, y):
        x_, y_ = self.real2Local(x, y)
        ans = False
        for detect_y in self.getDetecty(x_):
            if detect_y[0] <= y_ <= detect_y[1]:
                break
        return ans

    def pltAreaBeDetect(self, xmin, xmax, Nx, ymin, ymax, Ny):
        x, y = xmin, ymin
        dx, dy = (xmax - xmin) / Nx, (ymax - ymin) / Ny
        while x < xmax:
            detect_y = self.getDetecty(x)

            y = ymin
            while y < ymax:

                if detect_y[0][0] <= y <= detect_y[-1][1] and sea_rec.isInside(self.getRealPoint(x,
                    y)):
                    flag = False
                    for d_y in detect_y:
                        if d_y[0] <= y <= d_y[1]:
                            flag = True
```

```
                    break

                x_, y_ = self.getRealCoordinate(x, y)
                if flag:
                    plt.scatter(x_, y_, c='g', s=2)
                else:
                    plt.scatter(x_, y_, c='r', s=2)

            y += dy
        x += dx

def calculate(self, xmin, xmax, N):
    x = xmin
    dx = (xmax - xmin) / N
    AreaForNoMeasure = 0
    TotalLength = 0
    TotalLengthOfIllegal = 0
    while x < xmax:
        segment_ps = intersectionOfLineAndRec(Line(self.getRealPoint(x, 0), self.y_direction),
            sea_rec)
        if len(segment_ps) <= 1:
            x += dx
            continue
        segment = Segment(segment_ps[0], segment_ps[1])
        bundary_y = [self.real2Local_y(segment_ps[i].x, segment_ps[i].y) for i in [0, 1]]
        bundary_y.sort()
        inOcean = [segment.isExist(self.getRealPoint(x, y)) for y in self.yList]
        detect_y = self.getDetecty(x)
        coll = self.ifCoincideIllegal(x, 0.1, 0.2)

        # Effective = [coll[i] and inOcean[i] for i in range(len(self.yList))]
        count = 0

        for i in range(self.n):
            if inOcean[i]:
                count += 1
        TotalLength += count * dx

        count = 0
        for i in range(self.n):
            if inOcean[i] and not coll[i]:
                count += 1
        TotalLengthOfIllegal += count * dx

        countUnmeasureLength = 0
        d_y_min = bundary_y[0]
        for i in range(self.n):
            if inOcean[i]:
                if d_y_min < detect_y[i][0]:
                    countUnmeasureLength += detect_y[i][0] - d_y_min
```

```
                d_y_min = detect_y[i][1]
            if d_y_min < bundary_y[1]:
                countUnmeasureLength += bundary_y[1] - d_y_min
            AreaForNoMeasure += countUnmeasureLength * dx

            x += dx

        return AreaForNoMeasure / (hl2m * hl2m), TotalLength / hl2m, TotalLengthOfIllegal / hl2m

    def plotMeasureArea(self, xmin, xmax, N):
        x = xmin
        dx = (xmax - xmin) / N
        detect_line = [[[], [], [], []] for i in range(self.n)]
        while x < xmax:
            segment_ps = intersectionOfLineAndRec(Line(self.getRealPoint(x, 0), self.y_direction),
                sea_rec)
            if len(segment_ps) <= 1:
                x += dx
                continue
            segment = Segment(segment_ps[0], segment_ps[1])
            bundary_y = [self.real2Local_y(segment_ps[i].x, segment_ps[i].y) for i in [0, 1]]
            bundary_y.sort()
            # inOcean = [segment.isExist(self.getRealPoint(x, y)) for y in self.yList]
            inOcean = [sea_rec.isInside(self.getRealPoint(x, y)) for y in self.yList]
            detect_y = self.getDetecty(x)
            # coll = self.ifCoincideIllegal(x, 0.1, 0.2)

            for i in range(self.n):
                if inOcean[i]:
                    p0 = self.getRealPoint(x, detect_y[i][0])
                    detect_line[i][0].append(p0.x)
                    detect_line[i][1].append(p0.y)
                    p1 = self.getRealPoint(x, detect_y[i][1])
                    detect_line[i][2].append(p1.x)
                    detect_line[i][3].append(p1.y)

            x += dx
            # print(x)

        for i in range(self.n):
            detect_line[i][2].reverse()
            detect_line[i][3].reverse()
            plt.fill(detect_line[i][0] + detect_line[i][2], detect_line[i][1] + detect_line[i][3], c
                ='y')
            # plt.plot(detect_line[i][2], detect_line[i][3], c='y')

    def plotMeasureLine(self, xmin, xmax, N):

        x = xmin
        dx = (xmax - xmin) / N
```

```
        while x < xmax:
            segment_ps = intersectionOfLineAndRec(Line(self.getRealPoint(x, 0), self.y_direction),
                sea_rec)
            if len(segment_ps) <= 1:
                x += dx
                continue
            segment = Segment(segment_ps[0], segment_ps[1])
            bundary_y = [self.real2Local_y(segment_ps[i].x, segment_ps[i].y) for i in [0, 1]]
            bundary_y.sort()
            # inOcean = [segment.isExist(self.getRealPoint(x, y)) for y in self.yList]
            inOcean = [sea_rec.isInside(self.getRealPoint(x, y)) for y in self.yList]
            detect_y = self.getDetecty(x)
            coll = self.ifCoincideIllegal(x, 0.1, 0.2)

            for i in range(self.n):
                if inOcean[i]:
                    p = self.getRealPoint(x, self.yList[i])
                    if coll[i]:
                        plt.scatter(p.x, p.y, s=2, c='g')
                    else:
                        plt.scatter(p.x, p.y, s=2, c='r')

            x += dx
            # print(x)


sea_rec = Rec(0, 0, 5 * hl2m, 4 * hl2m)
plt.plot(
    [sea_rec.x1, sea_rec.x1, sea_rec.x2, sea_rec.x2, sea_rec.x1],
    [sea_rec.y1, sea_rec.y2, sea_rec.y2, sea_rec.y1, sea_rec.y1], c='blue'
)

##
m = Measure(Direction(2, 1),
            [3560.390625, 3815.6484375, 4105.546875, 4420.0859375, 4780.0546875, 5200.96875,
                5700.921875, 6300]
            )
print(m.calculate(-6.5 * hl2m, 6.5 * hl2m, 10000))
print(m.CoincideMin(-7 * hl2m, 7 * hl2m, 200))
##m.pltLines(0, 7 * hl2m, 100)
## print('航线绘制完成')
##m.pltAreaBeDetect(0, 7 * hl2m, 100, -2*hl2m, 2*hl2m, 100)
m.plotMeasureArea(-7 * hl2m, 7 * hl2m, 1000)
print('探测区域绘画完毕')
m.plotMeasureLine(-7 * hl2m, 7 * hl2m, 100)
print('探测线绘画完毕')
p = m.getRealPoint(0,3560.4)
p1,p2 = [p for p in intersectionOfLineAndRec(Line(m.getRealPoint(0, 3560.4 -150), m.x_direction),
    sea_rec)]
print(p1,p2)
```

```
plt.plot([p1.x,p2.x], [p1.y,p2.y],c='grey')
print()
plt.show()

# plt.plot([0,7000], [4000,4*hl2m], c='grey')
#m = Measure(Direction(0,-1), [42.0, 113.6541264139558, 184.92707566184043, 255.92374735460805,
    326.67737470433207, 397.118945204696, 467.2840225291539, 537.2237953511599, 606.9728507816682,
    676.5657759316329, 746.0374155289325, 815.423333247021, 884.7571396343526, 954.0736794188058,
    1023.3895002576834, 1092.740681058288, 1162.1603710404224, 1231.6843914944652,
    1301.3473295313704, 1371.183568550365, 1441.244297224903, 1511.4789380167117,
    1581.9035772544426, 1652.6399317323385, 1723.7231457913376, 1795.1695094718186,
    1867.0146403524327, 1939.3099847235583, 2012.0899259844223, 2085.372964917328,
    2159.2100864840017, 2233.635674083671, 2308.6850876780627, 2384.392710666404,
    2460.7926688309976, 2537.920837367419, 2615.8103654954716, 2694.7048786063065,
    2774.6386823758294, 2855.4558082390886, 2937.2084753372355, 3019.9306057408457,
    3103.9000045285707, 3189.1679863622703, 3275.5103311123876, 3363.255625107574,
    3452.455901955265, 3543.180683126676, 3635.4639088703943, 3729.3753582745603,
    3824.9668073296853, 3922.290714615833, 4021.761879756591, 4123.086357439459, 4226.33372304085,
    4331.953932434351, 4440.0165791743875, 4550.607139432306, 4663.813761450033, 4779.704343273916,
    4898.366314200302, 5020.354415634391, 5145.771312627088, 5274.219268445958, 5406.304854391905,
    5542.685644276738, 5683.500489869059, 5828.299818900612, 5977.827771996397, 6132.811396803189,
    6292.782919883488, 6458.556752366158, 6630.306668471003, 6808.222660374995, 6993.1873907716035,
    7185.427080551634, 7385.929208027241, 7594.935319476168, 7812.740006976168, 8040.451921038668,
    8279.266374163668, 8524.022233538668, 8763.965592913668, 8999.810319476168, 9231.593522601168]
# )
# print(m.calculate(-6.5*hl2m, 6.5*hl2m, 1000))
## print(m.CoincideMin(-7*hl2m, 7*hl2m, 200))
###m.pltLines(0, 7 * hl2m, 100)
### print('航线绘制完成')
###m.pltAreaBeDetect(0, 7 * hl2m, 100, -2*hl2m, 2*hl2m, 100)
#m.plotMeasureArea(-7 * hl2m, 7 * hl2m, 1000)
# print('探测区域绘画完毕')
#m.plotMeasureLine(-7 * hl2m, 7 * hl2m, 100)
# print('探测线绘画完毕')
# plt.show()

#
#m = Measure(Direction(1,0), [42.0, 120.56575064218059, 197.62466534686118, 273.29783786404175,
    347.70636194372236, 420.98890946090296, 493.26657416558356, 564.6956060577642,
    635.4322551374447, 705.5976151546253, 775.3293814218058, 844.8365383139864, 914.2216248936669,
    983.5373755358474, 1052.922462115528, 1122.5155565077087, 1192.4553305873892,
    1262.8804562295697, 1333.9462068717503, 1405.7912543889308, 1478.5542706561114,
    1552.373927548292, 1627.3888969404725, 1703.754452270153, 1781.5575075998336,
    1860.9367348045141, 1942.0132276341947, 2024.9432360888752, 2110.4025414185558,
    2198.5464170607365, 2289.5301364529173, 2383.510926157598, 2480.747575237279,
    2582.0369586919596, 2686.8771233966404, 2795.511233413821, 2908.751788743502,
    3027.7081643856827, 3152.5190322153635, 3283.4959156700443, 3420.777486624725,
    3564.519995079406, 3715.7791050965866, 3875.6817698012674, 4044.227012630948, 4221.501747648129,
     4408.268670165309, 4604.31879580749, 4810.20778863717, 5026.12510177935, 5251.0834305465305,
    5483.870860876211, 5723.500088080891, 5967.7884949730715, 6214.328854990252, 6460.003004069932,
```

```
    6702.126371899612, 6940.286849104293, 7174.331115371473, 7404.433975388653]
# )
# print(m.calculate(-6.5*hl2m, 6.5*hl2m, 1000))
## print(m.CoincideMin(-7*hl2m, 7*hl2m, 200))
###m.pltLines(0, 7 * hl2m, 100)
### print('航线绘制完成')
###m.pltAreaBeDetect(0, 7 * hl2m, 100, -2*hl2m, 2*hl2m, 100)
#m.plotMeasureArea(-7 * hl2m, 7 * hl2m, 1000)
# print('探测区域绘画完毕')
#m.plotMeasureLine(-7 * hl2m, 7 * hl2m, 100)
# print('探测线绘画完毕')
# plt.show()


##################################自动化调参
# cases = [42.]
#m = Measure(Direction(1,0), cases)
# for i in range(100):
#     l = hl2m
#     ly = cases[-1]# + math.tan(t / 2) * DeepOfSea(m.y_direction.dx * cases[-1] + m.x_direction.dx*
    hl2m, m.y_direction.dy * cases[-1]+ m.x_direction.dy*hl2m)
#     ry = ly + 1
#     if len(intersectionOfCirAndRec(Circle(0, 0, abs(ly)), sea_rec)) <= 1:
#         break
#
#     while True:
#         m.setyList(cases + [ry])
#         flag = m.CoincideMin(-6.5 * hl2m, 6.5 * hl2m, 100)[-1][0]
#         if flag < 0:
#             break
#         ry += 1
#         ly += 1
#
#     m.setyList(cases + [ly])
#     flag = m.CoincideMin(-6.5 * hl2m, 6.5 * hl2m, 100)[-1][0]
#     if flag < 0:
#         break
#
#     while ly + 0.01 < ry:
#         midy = 0.5 * (ry + ly)
#         m.setyList(cases + [midy])
#         flag = m.CoincideMin(-6.5 * hl2m, 6.5 * hl2m, 100)[-1][0]
#         if flag < 0:
#             ry = midy
#         else:
#             ly = midy
#     cases.append(ly)
#     print(cases)
# print(cases)
```

```
# cases = [6300]
# m = Measure(Direction(2, 1), cases)
# for i in range(100):
#     ry = cases[-1] - 1.5*math.tan(t / 2) * DeepOfSea(m.y_direction.dx * cases[-1], m.y_direction.
    dx * cases[-1])
#     ly = ry - 1
#     # if len(intersectionOfCirAndRec(Circle(0, 0, abs(ly)), sea_rec)) <= 1:
#     #     break
#
#     while True:
#         m.setyList(cases + [ly])
#         flag = m.CoincideMin(-7 * hl2m, 7 * hl2m, 100)[0][1]
#         if flag < 0:
#             break
#         ry -= 1
#         ly -= 1
#
#     while ly + 0.001 < ry:
#         midy = 0.5 * (ry + ly)
#         m.setyList(cases + [midy])
#         flag = m.CoincideMin(-6.5 * hl2m, 6.5 * hl2m, 100)[0][1]
#         if flag < 0:
#             ry = midy
#         else:
#             ly = midy
#     cases.append(ly)
#     print(cases)
# print(cases)
```

附录 4：绘制第一问示意图（Map1.m）

```
clc;
clear;

t = -800:2:800;
x = t; % 对范围内取横坐标
y = x; % 画一个同样大小的y坐标
d = 70 - x.*sind(1.5); % 带入公式计算深度
depth = -repmat(d, length(y), 1); % 计算维度

% 创建横坐标和纵坐标网格
[x, y] = meshgrid(x, y);

%% 海域深度三维图

% 绘制深度图
figure(1);
paintMap(x, y, depth, t)
title('问题1三维海域示意图');
```

```
%% 海域深度平面图

% 绘制深度图的填充等高线图和颜色
figure(2);
paintMap(x, y, depth, t)
view(0, 90);
title('问题1平面海域示意图');

%% 用surf函数画图
function paintMap(x, y, depth, t)

    s = surf(x, y, depth);
    colormap('winter'); % 使用'parula'颜色映射
    xlabel('横向坐标/m');
    ylabel('纵向坐标/m');
    zlabel('水域海拔/m');
    shading flat % 消除栅格线
    s.FaceColor='interp';% 允许跨栅格绘制颜色，即边缘平滑一些
    c = colorbar;
    title(c,'水域海拔/m')

    caxis([-100, -30]);

    hold on

    x_lines = [-800, -600, -400, -200, 0, 200, 400, 600, 800];
    for i = 1:length(x_lines)
        x_line = x_lines(i);
        plot3(x_line*ones(1, length(t)), t, -70 + x_line*sind(1.5)*ones(1, length(t)), 'k')
    end

    hold off; % 结束图形上下文

    % 设置坐标轴的可见性

    as = gca;
    as.XLim = [-800, 800];
    as.YLim = [-800, 800];
    as.ZLim = [-95, 0];


    hold on
    plot3(zeros(1, length(t)), t, zeros(1, length(t)), 'r')
    plot3(200*ones(1, length(t)), t, zeros(1, length(t)), 'r')

    d1 = repmat(t.*sind(30), length(y), 1);
    d1 = d1.*(d1>depth);
    d1(d1 == 0) = NaN;

    d2 = -repmat(t.*sind(30), length(y), 1);
```

```
    d2 = d2.*(d2>depth);
    d2(d2 == 0) = NaN;

    d3 = repmat((t-200)*sind(30), length(y), 1);
    d3 = d3.*(d3>depth);
    d3(d3 == 0) = NaN;

    d4 = -repmat((t-200)*sind(30), length(y), 1);
    d4 = d4.*(d4>depth);
    d4(d4 == 0) = NaN;

    s1 = surf(x, y, d1,'EdgeColor','none','FaceColor','y'); % 设置颜色为黄色
    alpha(s1,0.5); % 设置透明度为50%

    s2 = surf(x, y, d2,'EdgeColor','none','FaceColor','y'); % 设置颜色为黄色
    alpha(s2,0.5); % 设置透明度为50%

    s3 = surf(x, y, d3,'EdgeColor','none','FaceColor','y'); % 设置颜色为黄色
    alpha(s3,0.5); % 设置透明度为50%

    s4 = surf(x, y, d4,'EdgeColor','none','FaceColor','y'); % 设置颜色为黄色
    alpha(s4,0.5); % 设置透明度为50%

    hold off
end
```

附录 5：绘制直线布设模型结果检验示意图（Map3.m）

```
clc;
clear;

t = -800:2:800;
x = -3704:4:3704; % 画一个同样大小的y坐标
y = -1852:2:1852; % 对范围内取横坐标
d = 110 - x.*sind(1.5); % 带入公式计算深度
depth = -repmat(d, length(y), 1); % 计算维度

% 创建横坐标和纵坐标网格
[x, y] = meshgrid(x, y);

%% 导入数据
D = [1 13.007346507731626 13.59729921694709 14.243309453975241
     2 14.11971315935088 14.760117643597757 15.461373602253115
     3 15.327207557962891 16.022378369177396 16.783604571740213
     4 16.63796487036248 17.392585533790463 18.21891053583297
     5 18.06081596928592 18.879970537466964 19.77696148010888
     6 19.605346929026584 20.494554234222406 21.46825435123792
     7 21.28196360901679 22.247214444850325 23.304183777320503
     8 23.10196176049013 24.149759243296636 25.297118836111252
     9 25.07760312854914 26.21500651036293 27.46048638833361
```

```
    10 27.22219806235516 28.456870290710143 29.808861537521032
    11 29.550195190004445 30.89045453496973 32.35806582583339
    12 32.0772787622505 33.532154858524315 35.12527382741641
    13 34.82047432089982 36.39976900253007 38.12912885744794
    14 37.79826340379313 39.51261674138221 41.38986857642964
    15 41.03070805916599 42.89167004446756 44.92946133594882
    16 44.53958600827054 46.559694369133815 48.771754184503216
    17 48.348537366879945 50.54140203679871 52.94263353053782
    18 52.48322391417203 54.86361872552907 57.470199545117666
    19 56.9715019820231 59.555464200788506 62.384955479227195
    20 61.843610129506786 64.64854850197813 67.72001317117041
    21 67.13237286700405 70.1771849065242 73.51131612862223
    22 72.87342180246041 76.1786211063006 79.79788168828611
    23 79.10543569970355 82.69329015387449 86.6220638846431
    24 85.87040106614914 89.76508286925795 94.02983879879913
    25 93.21389502553413 97.44164354243205 102.07111430989175
    26 101.185392381456 105.77469092385947 110.80006633592194
    27 109.83859894047535 114.82036666557187 120.27550382934308
    28 119.23181334045631 124.63961356036 130.56126498646606
    29 129.42831982186507 135.29858612735083 141.72664734003394
    30 140.49681458821703 146.86909631018034 153.84687463259874
    31 152.51186862816058 159.4290972905334 167.0036036161334
    32 165.55443011733612 173.06320867661336 181.285474192304
    33 179.7123697848072 187.86328660485643 196.78870659982363
    34 195.08107291832198 203.92904259579726 213.61774967227606];

X1 = (110-D(:,2))./tand(1.5);
X2 = (110-D(:,3))./tand(1.5);
X3 = (110-D(:,4))./tand(1.5);

hold on

for i = 1:length(X1)
    plot(ones(1,length(y))*X2(i),y,'r')
    x0 = X3(i);              % 左上角 x 坐标
    y0 = -1852;              % 左上角 y 坐标
    width = X1(i)-X3(i);     % 矩形的宽度
    height = 3704;     % 矩形的高度
    r = rectangle('Position', [x0, y0, width, height], 'EdgeColor', 'k', 'FaceColor', 'y');

end

as = gca;
as.XLim = [-3704, 3704];
as.YLim = [-1852, 1852];

xlabel('横向坐标/m');
ylabel('纵向坐标/m');
title('问题1结果覆盖区域示例图');
```

附录 6：绘制第四问海域地形示意图（Map4.m）

```
clc;clear

filename = 'data.xlsx';

% 使用xlsread函数读取数据
data = xlsread(filename);

% 提取横坐标、纵坐标和深度数据
x = data(1, 2:end); % 第一行是横坐标
y = data(2:end, 1); % 第一列是纵坐标
depth = -data(2:end, 2:end); % 深度数据在第二行和第二列之后的区域

% 创建横坐标和纵坐标网格
[x, y] = meshgrid(x, y);

%% 海域深度三维图

% 绘制深度图
figure(1);
surf(x, y, depth);
c = colorbar;
title(c,'水域海拔/m');
colormap('winter');
xlabel('横向坐标/NM');
ylabel('纵向坐标/NM');
zlabel('水域海拔/m');
title('海域深度三维图');
caxis([-250, 50]);

% 添加等高线

hold on; % 保持图形上下文
contour3(x, y, depth, 20, 'LineColor', 'k'); % 添加等高线，'k'表示黑色线条，20表示等高线数量

% 获取等高线矩阵
h = contourc(x(1,:), y(:,1), depth, 20);

% 添加等高线上的数值标签
clabel(h, 'LabelSpacing', 200); % 调整LabelSpacing以控制标签间隔

hold off; % 结束图形上下文

% 设置坐标轴的可见性
shading flat % 消除栅格线
s1.FaceColor='interp';% 允许跨栅格绘制颜色，即边缘平滑一些

%% 海域深度平面图
```

```
% 绘制深度图的填充等高线图和颜色
figure(2);
contourf(x, y, depth, 20, 'LineColor', 'none'); % 使用contourf函数绘制填充等高线图和颜色
colormap('winter');
xlabel('横向坐标/NM');
ylabel('纵向坐标/NM');
title('海域深度平面图');
c1 = colorbar('Location', 'westoutside');
title(c1,'水域海拔/m')
% 获取等高线矩阵
h = contourc(x(1,:), y(:,1), depth, 20);

% 添加等高线上的数值标签
clabel(h, 'LabelSpacing', 200); % 调整LabelSpacing以控制标签间隔

set(gca, 'YDir', 'reverse'); % 使用'YDir'属性将y轴坐标反转

caxis([-250, 50]);
```

附录 7：问题 1 计算结果（result1.xlsx）

测线距中心点处的距离/m	-800	-600	-400	-200	0	200	400	600	800
海水深度/m	90.95	85.71	80.47	75.24	70	64.76	59.53	54.29	49.05
覆盖宽度/m	315.71	297.53	279.35	261.17	242.99	224.81	206.63	188.45	170.27
与前一条测线的重叠率/%	——	35.7	31.5	26.7	21.3	14.9	7.4	-1.5	-13.4

附录 8: 问题 2 计算结果（result2.xlsx）

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	415.69	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	309.1	273.42	237.75	202.08	166.4
	180	415.69	365.29	314.89	264.5	214.1	163.7	113.3	62.9
	225	416.12	380.45	344.77	309.1	273.42	237.75	202.08	166.4
	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	558.82	594.49	630.16	665.84

附录 9: 扇形布设策略计算结果（result3-2.xlsx）

测线编号	1	2	3	4	5
方向向量	(-0.256, -0.967)	(-0.277, -0.961)	(-0.3, -0.954)	(-0.324, -0.946)	(-0.35, -0.937)
测线编号	6	7	8	9	10
方向向量	(-0.378, -0.926)	(-0.407, -0.913)	(-0.438, -0.899)	(-0.471, -0.882)	(-0.506, -0.862)
测线编号	11	12	13	14	15
方向向量	(-0.542, -0.84)	(-0.58, -0.814)	(-0.619, -0.785)	(-0.659, -0.752)	(-0.7, -0.714)
测线编号	16	17	18	19	20
方向向量	(-0.741, -0.672)	(-0.781, -0.625)	(-0.82, -0.572)	(-0.857, -0.515)	(-0.892, -0.453)
测线编号	21	22	23	24	25
方向向量	(-0.923, -0.386)	(-0.949, -0.315)	(-0.971, -0.239)	(-0.987, -0.161)	(-0.997, -0.081)
测线编号	26	27	28	29	30
方向向量	(-1, 0)	(-0.997, 0.081)	(-0.987, 0.161)	(-0.971, 0.239)	(-0.949, 0.315)
测线编号	31	32	33	34	35
方向向量	(-0.923, 0.386)	(-0.892, 0.453)	(-0.857, 0.515)	(-0.82, 0.572)	(-0.781, 0.625)
测线编号	36	37	38	39	40
方向向量	(-0.741, 0.672)	(-0.7, 0.714)	(-0.659, 0.752)	(-0.619, 0.785)	(-0.58, 0.814)
测线编号	41	42	43	44	45
方向向量	(-0.542, 0.84)	(-0.506, 0.862)	(-0.471, 0.882)	(-0.438, 0.899)	(-0.407, 0.913)
测线编号	46	47	48	49	50
方向向量	(-0.378, 0.926)	(-0.35, 0.937)	(-0.324, 0.946)	(-0.3, 0.954)	(-0.277, 0.961)
测线编号	51	测线总长度总和（NM）			
方向向量	(-0.256, 0.967)	68.0145			

附录 10: 最终设计方案侧线具体坐标（最终设计方案侧线具体坐标.txt）

斜向

Line(Direction(0.8944271909999159, 0.4472135954999579), StartPoint(-1592.2550927905925, 3184.510185581185))
Line(Direction(0.8944271909999159, 0.4472135954999579), StartPoint(-1706.4098568981715, 3412.819713796343))
Line(Direction(0.8944271909999159, 0.4472135954999579), StartPoint(-1836.0563794623663, 3672.1127589247326))
Line(Direction(0.8944271909999159, 0.4472135954999579), StartPoint(-1976.7225245281772, 3953.4450490563545))
Line(Direction(0.8944271909999159, 0.4472135954999579), StartPoint(-2137.7054434833026, 4275.410886966605))
Line(Direction(0.8944271909999159, 0.4472135954999579), StartPoint(-2325.943934770422, 4651.887869540844))
Line(Direction(0.8944271909999159, 0.4472135954999579), StartPoint(-2549.5297693831117, 5099.059538766223))
Line(Direction(0.8944271909999159, 0.4472135954999579), StartPoint(-2817.445651649735, 5634.89130329947))

纵向

Line(Direction(0.0, -1.0), StartPoint(42.0, 0.0))
Line(Direction(0.0, -1.0), StartPoint(113.6541264139558, 0.0))
Line(Direction(0.0, -1.0), StartPoint(184.92707566184043, 0.0))
Line(Direction(0.0, -1.0), StartPoint(255.92374735460805, 0.0))
Line(Direction(0.0, -1.0), StartPoint(326.67737470433207, 0.0))
Line(Direction(0.0, -1.0), StartPoint(397.118945204696, 0.0))
Line(Direction(0.0, -1.0), StartPoint(467.2840225291539, 0.0))
Line(Direction(0.0, -1.0), StartPoint(537.2237953511599, 0.0))
Line(Direction(0.0, -1.0), StartPoint(606.9728507816682, 0.0))
Line(Direction(0.0, -1.0), StartPoint(676.5657759316329, 0.0))
Line(Direction(0.0, -1.0), StartPoint(746.0374155289325, 0.0))
Line(Direction(0.0, -1.0), StartPoint(815.423333247021, 0.0))
Line(Direction(0.0, -1.0), StartPoint(884.7571396343526, 0.0))
Line(Direction(0.0, -1.0), StartPoint(954.0736794188058, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1023.3895002576834, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1092.740681058288, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1162.1603710404224, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1231.6843914944652, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1301.3473295313704, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1371.183568550365, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1441.244297224903, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1511.4789380167117, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1581.9035772544426, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1652.6399317323385, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1723.7231457913376, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1795.1695094718186, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1867.0146403524327, 0.0))
Line(Direction(0.0, -1.0), StartPoint(1939.3099847235583, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2012.0899259844223, 0.0))

Line(Direction(0.0, -1.0), StartPoint(2085.372964917328, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2159.2100864840017, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2233.635674083671, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2308.6850876780627, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2384.392710666404, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2460.7926688309976, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2537.920837367419, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2615.8103654954716, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2694.7048786063065, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2774.6386823758294, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2855.4558082390886, 0.0))
Line(Direction(0.0, -1.0), StartPoint(2937.2084753372355, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3019.9306057408457, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3103.9000045285707, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3189.1679863622703, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3275.5103311123876, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3363.255625107574, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3452.455901955265, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3543.180683126676, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3635.4639088703943, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3729.3753582745603, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3824.9668073296853, 0.0))
Line(Direction(0.0, -1.0), StartPoint(3922.290714615833, 0.0))
Line(Direction(0.0, -1.0), StartPoint(4021.761879756591, 0.0))
Line(Direction(0.0, -1.0), StartPoint(4123.086357439459, 0.0))
Line(Direction(0.0, -1.0), StartPoint(4226.33372304085, 0.0))
Line(Direction(0.0, -1.0), StartPoint(4331.953932434351, 0.0))
Line(Direction(0.0, -1.0), StartPoint(4440.0165791743875, 0.0))
Line(Direction(0.0, -1.0), StartPoint(4550.607139432306, 0.0))
Line(Direction(0.0, -1.0), StartPoint(4663.813761450033, 0.0))
Line(Direction(0.0, -1.0), StartPoint(4779.704343273916, 0.0))
Line(Direction(0.0, -1.0), StartPoint(4898.366314200302, 0.0))
Line(Direction(0.0, -1.0), StartPoint(5020.354415634391, 0.0))
Line(Direction(0.0, -1.0), StartPoint(5145.771312627088, 0.0))
Line(Direction(0.0, -1.0), StartPoint(5274.219268445958, 0.0))
Line(Direction(0.0, -1.0), StartPoint(5406.304854391905, 0.0))
Line(Direction(0.0, -1.0), StartPoint(5542.685644276738, 0.0))
Line(Direction(0.0, -1.0), StartPoint(5683.500489869059, 0.0))
Line(Direction(0.0, -1.0), StartPoint(5828.299818900612, 0.0))
Line(Direction(0.0, -1.0), StartPoint(5977.827771996397, 0.0))
Line(Direction(0.0, -1.0), StartPoint(6132.811396803189, 0.0))
Line(Direction(0.0, -1.0), StartPoint(6292.782919883488, 0.0))
Line(Direction(0.0, -1.0), StartPoint(6458.556752366158, 0.0))
Line(Direction(0.0, -1.0), StartPoint(6630.306668471003, 0.0))
Line(Direction(0.0, -1.0), StartPoint(6808.222660374995, 0.0))
Line(Direction(0.0, -1.0), StartPoint(6993.1873907716035, 0.0))
Line(Direction(0.0, -1.0), StartPoint(7185.427080551634, 0.0))
Line(Direction(0.0, -1.0), StartPoint(7385.929208027241, 0.0))
Line(Direction(0.0, -1.0), StartPoint(7594.935319476168, 0.0))
Line(Direction(0.0, -1.0), StartPoint(7812.740006976168, 0.0))

Line(Direction(0.0, -1.0), StartPoint(8040.451921038668, 0.0))
Line(Direction(0.0, -1.0), StartPoint(8279.266374163668, 0.0))
Line(Direction(0.0, -1.0), StartPoint(8524.022233538668, 0.0))
Line(Direction(0.0, -1.0), StartPoint(8763.965592913668, 0.0))
Line(Direction(0.0, -1.0), StartPoint(8999.810319476168, 0.0))
Line(Direction(0.0, -1.0), StartPoint(9231.593522601168, 0.0))

横向
Line(Direction(1.0, 0.0), StartPoint(0.0, 42.0))
Line(Direction(1.0, 0.0), StartPoint(0.0, 120.56575064218059))
Line(Direction(1.0, 0.0), StartPoint(0.0, 197.62466534686118))
Line(Direction(1.0, 0.0), StartPoint(0.0, 273.29783786404175))
Line(Direction(1.0, 0.0), StartPoint(0.0, 347.70636194372236))
Line(Direction(1.0, 0.0), StartPoint(0.0, 420.98890946090296))
Line(Direction(1.0, 0.0), StartPoint(0.0, 493.26657416558356))
Line(Direction(1.0, 0.0), StartPoint(0.0, 564.6956060577642))
Line(Direction(1.0, 0.0), StartPoint(0.0, 635.4322551374447))
Line(Direction(1.0, 0.0), StartPoint(0.0, 705.5976151546253))
Line(Direction(1.0, 0.0), StartPoint(0.0, 775.3293814218058))
Line(Direction(1.0, 0.0), StartPoint(0.0, 844.8365383139864))
Line(Direction(1.0, 0.0), StartPoint(0.0, 914.2216248936669))
Line(Direction(1.0, 0.0), StartPoint(0.0, 983.5373755358474))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1052.922462115528))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1122.5155565077087))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1192.4553305873892))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1262.8804562295697))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1333.9462068717503))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1405.7912543889308))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1478.5542706561114))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1552.373927548292))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1627.3888969404725))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1703.754452270153))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1781.5575075998336))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1860.9367348045141))
Line(Direction(1.0, 0.0), StartPoint(0.0, 1942.0132276341947))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2024.9432360888752))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2110.4025414185558))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2198.5464170607365))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2289.5301364529173))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2383.510926157598))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2480.747575237279))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2582.0369586919596))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2686.8771233966404))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2795.511233413821))
Line(Direction(1.0, 0.0), StartPoint(0.0, 2908.751788743502))
Line(Direction(1.0, 0.0), StartPoint(0.0, 3027.7081643856827))
Line(Direction(1.0, 0.0), StartPoint(0.0, 3152.5190322153635))
Line(Direction(1.0, 0.0), StartPoint(0.0, 3283.4959156700443))
Line(Direction(1.0, 0.0), StartPoint(0.0, 3420.777486624725))
Line(Direction(1.0, 0.0), StartPoint(0.0, 3564.519995079406))

```
Line(Direction(1.0, 0.0), StartPoint(0.0, 3715.7791050965866))
Line(Direction(1.0, 0.0), StartPoint(0.0, 3875.6817698012674))
Line(Direction(1.0, 0.0), StartPoint(0.0, 4044.227012630948))
Line(Direction(1.0, 0.0), StartPoint(0.0, 4221.501747648129))
Line(Direction(1.0, 0.0), StartPoint(0.0, 4408.268670165309))
Line(Direction(1.0, 0.0), StartPoint(0.0, 4604.31879580749))
Line(Direction(1.0, 0.0), StartPoint(0.0, 4810.20778863717))
Line(Direction(1.0, 0.0), StartPoint(0.0, 5026.12510177935))
Line(Direction(1.0, 0.0), StartPoint(0.0, 5251.0834305465305))
Line(Direction(1.0, 0.0), StartPoint(0.0, 5483.870860876211))
Line(Direction(1.0, 0.0), StartPoint(0.0, 5723.500088080891))
Line(Direction(1.0, 0.0), StartPoint(0.0, 5967.7884949730715))
Line(Direction(1.0, 0.0), StartPoint(0.0, 6214.328854990252))
Line(Direction(1.0, 0.0), StartPoint(0.0, 6460.003004069932))
Line(Direction(1.0, 0.0), StartPoint(0.0, 6702.126371899612))
Line(Direction(1.0, 0.0), StartPoint(0.0, 6940.286849104293))
Line(Direction(1.0, 0.0), StartPoint(0.0, 7174.331115371473))
Line(Direction(1.0, 0.0), StartPoint(0.0, 7404.433975388653))
```