多波束测深布线问题研究

摘要

本文主要研究了多波束测深技术中测线规划的问题。根据多波束测设技术的原理，以降维、建系的思想对测深问题进行建模，并通过二分逼近、贪心算法等方法进行了不同海面即测试条件下测量参数的分析及最佳航线的规划。

针对问题一，在横切面的合适位置建立直角坐标系，将海底坡面、极端位置的波束等信息抽象成直线，并通过在极端位置求解坐标和角度的方法得到测量指标。

针对问题二，选择测量船发出波束的横切面进行研究，将三维问题进行降维。利用立体几何知识，可以用坡面夹角以及侧线方向夹角表示出横切面上的坡面角即$\tan\alpha' = \frac{OC}{OA} = \tan\alpha \cdot \cos(\beta - 90)$至此第二问可以转化为并看作第一问中测线方向平行性于坡平面且海底中心深度已知的问题。

针对问题三，首先使用由问题二获得的推论：当测量船沿等深线测量时所覆盖的宽度最大，可知当所有规划的航线都是正南北走向的时候测绘效率最高。在明确这一点之后，再采取贪心算法，使测量船波束最西侧刚好测到最西侧的坡面，再贪心地使得东测的测线覆盖率尽量逼近 10%。这个过程通过二分逼近实现，求解出 34 条最优测线的坐标。

最后针对问题四，延续第三问中沿等深线进行航线规划的原则，通过连接海底平面在每个网格边上的等高点形成等折线近似线性描述等深线；通过插值选取等深线周围垂直等深线坡度并取平均值的方法获得近似坡度。在选取可能航线的时候，采用将连续转化为离散的思想，选取深度从 21m~196m，间隔为 1m 的 175 条等深线作为待选航线，通过与第三问相似的贪心算法在尽可能减少航线总路程的前提下尽量增加覆盖率并降低漏测率，以设计出最合理的航线。

关键词：多波束测深，航线规划，贪心算法，二分逼近，线性化

一、 问题重述

1.1 问题背景

在水体深度测绘领域，传统的单波束测深技术将声波垂直向海底发射，利用声波从发射到传回接收器的时间间隔测得水深。此方法获得的地形数据沿航迹分布十分密集，但是在远离测绘船航迹的测线之间分布稀疏。

而基于单波束测深原理的多波束测深系统会在与航迹垂直的平面内发射多个不同方向的波束打到海底，从而获得沿航迹有一定宽度的水深条带的测量数据。

在理想的平坦海底海域内，多波束探测系统能够高效地测量出全覆盖海底地形，然而由于真实海域起伏不定，多条测线间的间隔较难选择。如果测线间隔过大，测量条带重叠率过低，在水浅处会出现漏测的情况；如果测线间隔过小，测量条带重叠率过高，在深水地区会出现冗余数据，降低测量效率。

1.2 问题阐述

问题 1：在一片中心点处深度为 70m，坡度为 1.5°的斜面海底的海域内，测绘船以 120°的多波束换能器开角沿垂直于海底地形负梯度方向的测线进行测绘。如何描述测量条带覆盖宽度、相邻两测量条带的重合率与海水深度及海底地形坡度的关系？

问题 2：在一片中心深度为 120m，坡度为 1.5°的斜面海底的海域内，测绘船以 120°的多波束换能器开角沿与海底地形梯度方向不垂直的测线进行测绘如何描述测线与该负梯度方向的夹角与测量条带的覆盖宽度的关系？

问题 3：在南北长 2 海里、东西宽 4 海里、由西向东坡度为 +1.5°、中心点处深度为 110 m 的矩形海域内，测绘船以 120°的多波束换能器开角沿测线进行测绘。怎样的测线设计可使得条带在相邻两者之间的重叠率 10%~20%的情况下完全覆盖整个待测海域？

问题 4：已知若干年前某南北长 5 海里、东西宽 4 海里的海域单波束测量的测深数据。如何（1）以总测线长度尽可能短、测量条带覆盖海域面积尽可能大为目标设计测线；（2）在定量计算 a)侧线总长度；b)漏测海区面积占百分比；c)相邻条带之间重叠率超 20%部分占比的基础上评估这一设计？

二、问题分析

2.1 基本概念的解释

条带宽度：

本小组发现关于条带宽度 W 在斜面上的定义存在模糊之处。第一种定义方法将 W 定义为左右两条波束与海平面交点之间的距离，第二种定义方法将 W 定义为左右两条波束与海平面交点之间的水平距离。考虑到本题的物理意义为规划测线，而测线规划实际上是将三维的物理空间投射到二维的地图上，因此将 W 定义为波束在端点的水平距离更加具有物理意义。本文选用第一种定义模式。

2.2 问题一的分析

在问题一中，覆盖宽度与坡面角、能量转换器开角、中心海域深度之间存在平面几何关系。可通过建立笛卡尔坐标系用解析几何方式求出声波与海底接触点的坐标来确定覆盖宽度。根据η的物理意义来推导出η在此情况下的决定式进而求得η。

2.3 问题二的分析

该问题可以看作问题一的扩展形式。问题一是问题二在β=90°时的一种特例。如果能建立起α与β的数学关系，便可将问题二简化为问题一进行求解。而只要做出波束线所在平面与海底坡面的切面，就能通过三角函数关系算出此时的等价坡面倾角，从而解决第二题。

2.4 问题三的分析

由问题二可知当船只的前进方向在水平面上的投影与坡面法向在水平面上的投影之间的夹角为 90°时，条带的覆盖宽度最大，因此最优航向的角度是确定的。只需要通过调整测线间距来使得条带能完整覆盖整个海域的前提下重叠率尽可能小，就能规划出最优航线。

2.5 问题四的分析

我们可以通过对数据的预处理大概判断此海底的地势走向。根据问题二的推论，我们知道为了使测量效率最高，航迹应该与等深线大致重合。通过对深度数据的分析我们可以绘制出等深线，即测线的最优走向。只需对测线下对应的 α 进行估计，并以η的有效范围为约束条件确定相邻测线之间的适当间距，即可得出布线策略。

3、模型假设

（1）船只为空间中一点，假设多波束换能器在开角范围内所发射的波束能够全部被接受传感器吸收。不考虑声波在水中的能量损失[1]。

（2）船在不同测线间的衔接航程可以被忽略。

（3）船是可以按照任意给定方向在任意给定位置随时变相的物体。

（4）能量转换器开角不能任意变化，恒为 120°。

（5）五年前的测量数据准确，无异常值，且五年内该海域海底地形没有发生显著变化。

（6）该海域不存在极为陡峭的海底悬崖、海沟等极端地形。

（7）假设海水静止，无波浪。

4、 符号说明

符号	说明	量纲

d	两条侧线间的距离	m
D	测量船所在位置的海水深度	m
s	当前测线距中心测线的距离	m
W	条带的覆盖宽度	m
x_L	最左侧波束与坡面交点横坐标	m
x_R	最右侧波束与坡面交点横坐标	m
α	海底坡面的倾斜角度	degree
η	相邻条带的重叠率	-
θ	多波束换能器开角	degree

5、模型的建立与求解

5.1 问题一模型建立与求解

5.1.1 解析几何建模

在与测线垂直的切面内，以测线正下方的海平面为原点建立直角坐标系。假设船只所在点为 B 点，多波束换能器发射的最左及最右波束分别为 l1、l2，坡面(所代表的)与坐标系平面交于直线（为）L。l1、l2 与 L 的（焦点）交点分别为 P1，P2。

（对于海水深度，由基本三角函数知识可知，）（当前）第 k 条测线正下方海水深度可表述为：

$$D_k = D_0 - k \cdot d \cdot \tan(\alpha).$$

其中，D_0 为海域中心处深度，d 为两条测线间距，α 为坡面角。

设 L 为海底坡面，最左侧声波为 l1,最右侧声波为 l2，θ 为换能器开角。对于覆盖宽度，在直角坐标系中，

$$L : y = x \cdot \tan(\alpha)$$

$$l_1 : y = x \cdot \tan(90 - \frac{\theta}{2}) + h$$

$$l_2 : y = -x \cdot \tan(90 - \frac{\theta}{2}) + h$$

联立直线

$$\begin{cases} x_{kL} \cdot \tan(\alpha) = x_{kL} \cdot \cot(\frac{\theta}{2}) + D_k \\ x_{kR} \cdot \tan(\alpha) = -x_{kR} \cdot \cot(\frac{\theta}{2}) + D_k \end{cases}$$

得到

$$\begin{cases} x_{kR} = \frac{D_k}{\tan(\alpha) - \cot(\frac{\theta}{2})} \\ x_{kL} = \frac{D_k}{\tan(\alpha) + \cot(\frac{\theta}{2})} \end{cases}$$

故

$$W = x_{kR} - x_{kL} = \frac{2D_k \cot\frac{\theta}{2}}{\cot^2\frac{\theta}{2} - \tan^2\alpha}.$$

对于重叠，若想要得到当条测线与前一条侧线的重叠率，就需要知道前一条测线最右侧波束与坡面的交点　的横坐标　　.由于两次建立的坐标线之

间距离为 d,可以理解为前一条测线计算所建立的直角坐标系可由当前直角坐标系向左平移 d 得到.因此

$$x_{(k-1)R} = \frac{h'}{\tan(\alpha) - \tan(90 - \frac{\theta}{2})} - d$$

可知重叠率

$$\eta = \frac{x_{(k-1)R} - x_L}{W}.$$

5.1.2 算法求解

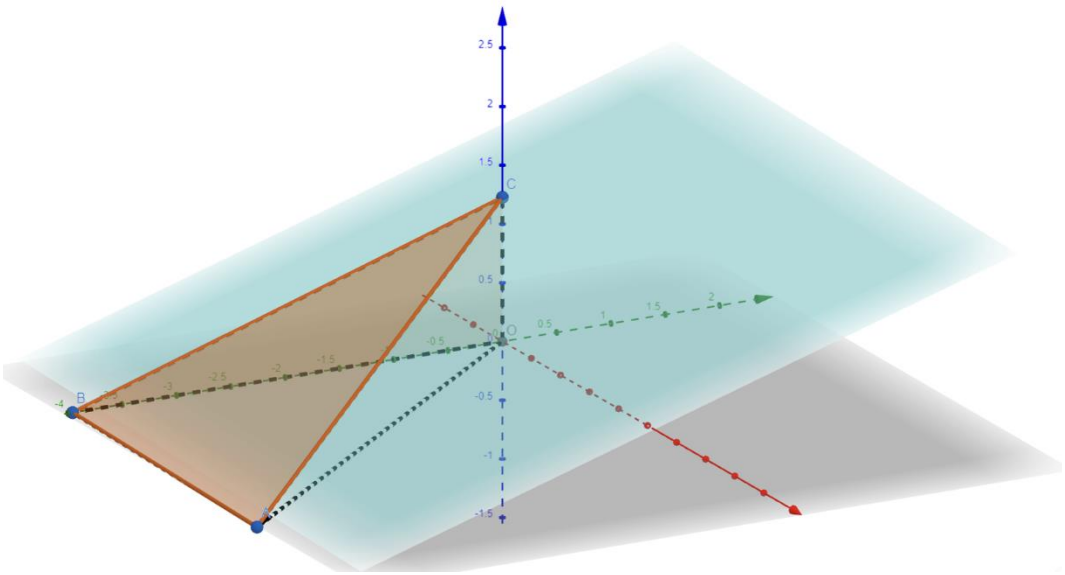
使用 JAVA 语言代入数据进行计算，可知结果如下表所示：

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	90.94	85.71	80.47	75.23	70	64.76	59.52	54.28	49.05
覆盖宽度/m	315.70	297.52	279.34	261.16	242.98	224.80	206.62	188.44	170.26
与前一条测线的重叠率/%	39.40%	35.70%	31.51%	26.74%	21.26%	14.89%	7.41%	-1.52%	-12.36%

5.2 问题二模型的建立与求解

5.2.1 立体几何建模

假设直线 AC 为波束平面与海底坡面的交线， A 点在水平面上。做 C 点在水平面上的投影 CO 交水平面于点 O。做 OB⊥AB 于 B 点。以 O 为原点建系如图。由几何知识可知 AB⊥平面 OBC， OA 垂直于侧线方向,因此∠CBO =α 。∠BAO = 180 - β 。



$$\triangle OBC \text{ 中}, OB = \frac{OC}{\tan\alpha};$$

$$\triangle OBA \text{ 中，} OA = \frac{OB}{\sin(180-\beta)} = \frac{OC}{sin(180-\beta)\cdot\tan\alpha}$$

设∠CAO = α’，则 $\triangle OCA \text{ 中，} \tan\alpha' = \frac{OC}{OA} = \tan\alpha\cdot\cos(\beta-90)$.

观察切面 OAC,则第二问中测量船在每一个位置的情况又可视为第一问中测线与坡面平行,坡面倾角为α’的情形。带入α’，θ,D,d,则可求出测量船在每个位置，每个倾角时的条带覆盖宽度。

5.2.2 算法求解

使用 JAVA 语言代入数据进行计算，可知结果如下表所示：

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	415.69	415.69	415.69	415.69	415.69	415.69	415.69
	45	416.12	380.45	344.77	309.19	273.42	237.75	202.08	166.49
	90	416.55	366.05	315.54	265.04	214.54	164.04	113.53	63.03
	135	416.12	380.45	344.77	309.1	273.42	237.75	202.08	166.4
	180	415.69	415.69	415.69	415.69	415.69	415.69	415.69	415.69
	225	416.12	451.79	487.47	523.14	558.82	594.49	630.16	665.84
	270	416.55	467.05	517.55	568.06	618.56	669.06	719.57	770.07
	315	416.12	451.79	487.47	523.14	558.82	594.49	630.16	665.84

5.2.3 结果分析

可得出结论，当 β=90° 时，覆盖宽度最大，测量效率最高。即当测线与斜坡梯度垂直，与等高线平行时，测量效率最高。此推论十分重要，在后续问题中会用到。

5.3 问题三模型的建立与求解

5.3.1 模型建立

根据问题二可知，当航线垂直于坡面梯度时，覆盖宽度最大。因此，测线应为一组南北走向的平行线。以下从最西处测线为起始点，使得最偏西的航线恰好能发出与坡面最西端相交的波束。之后采用贪心算法，根据西侧测线的位

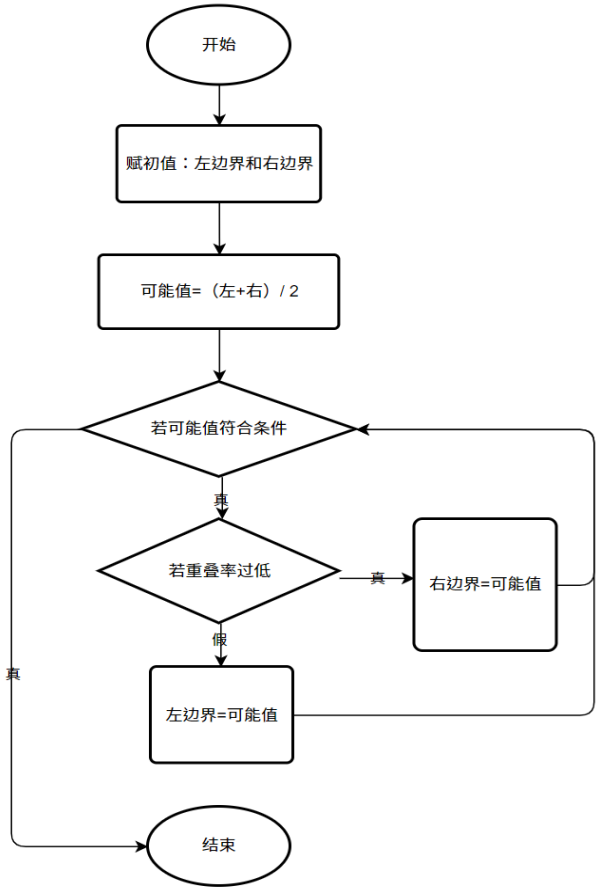
置确定其东侧的测线恰好能使得重叠率为 10%的位置，知道测线所发出的光束能够覆盖整个海域。

由于重叠率与测量效率负相关，为使测量长度最短，我们采用重叠率为 10%时的极限情况安排测线。

为尽量覆盖的全部海域，使第一条测线的覆盖范围刚好达到海域最西端。考虑海域在南北方向的横截面，以海域最西端为原点，x 轴方向向东，y 轴方向向上建立坐标系。第一条测线的坐标为（220tan30°,220）。当 η 取 10%时，最后一条测线的结束条件为：$x \geq= 4 - \tan 60^{\circ} \cdot （220 - 4tan1.5^{\circ}）$

5.3.2 算法求解

在第一条和最后一条测线之间使用二分法，逼近第二条的所在位置，其流程图如图所示。



其中 η 的限制条件为[10%,10.001%]。

当 x= 3883.26m 时， η = -8.22 ，不符合条件，继续二分。直到 x= 953.01m 时， η = 0.100009，符合条件。故第二条测线位置为 x= 953.01m。再以第二条测线与最后一条测线之间使用二分法，逼近第三条测线所在位置。以此类推，直到最后倒数第二条测线的位置被计算出。

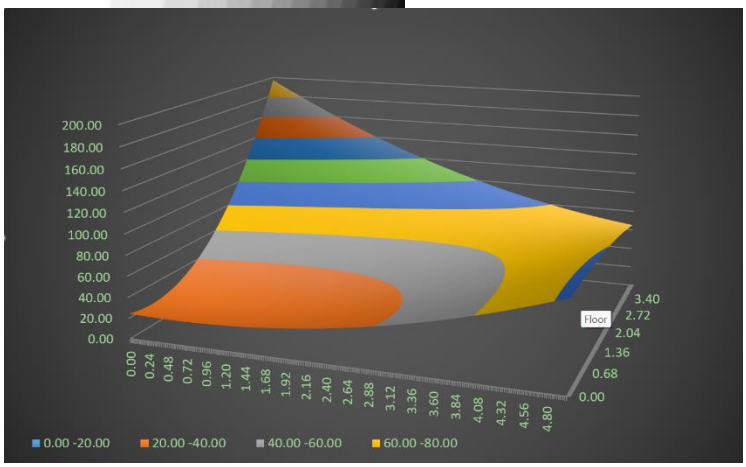
经过程序得到一组完整的测线，即其方向为南北走向，长度为 2 海里的 34 组平行测线，其东西方向坐标分别为{358.52, 953.01, 1500.68, 2005.19, 2469.96, 2898.11, 3292.54,

3655.89, 3990.62, 4298.97, 4583.04, 4844.72, 5085.79, 5307.87, 5512.45, 5700.92, 5874.54, 6034.48, 6181.82, 6317.55, 6442.59, 6557.78, 6663.89, 6761.65, 6851.70, 6934.66, 7011.08, 7081.48, 7146.34, 7206.09, 7261.12, 7311.83, 7358.54, 7401.57}。

5.4 问题四模型的建立与求解

5.4.1 数据预处理

利用 excel 和 c++程序，以二十米为一组进行分层，生成海底地形示意图。由示意图可以判断，此海底地形的几何性质较为平滑，其等深线具备拟合的可行性。



5.4.2 等深线拟合

5.4.2.1 模型建立

我们以 1 米为间距，拟合一组等深线。方法如下：

以所有已知深度的点为各点作网格线，假设每一最小单位段的网格线的高度变化是线性的，则可以计算出网格线与等深线的交点位置（若存在),再假设等深线在每一最小网格中是线性变化的，我们即可根据两个交点求出网格内的等深线。将所有等深线段相连并拟合，我们即可拟合出完整的等深线。

5.4.2.2 算法求解

step.0 将深度数据储存在二维数组中

step.1 找到所有锚点

step.1.1 在东西方向上使用双层循环遍历每一个点，判断这个点与下一个点是否跨越等高线。

step.1.2 若此点与下一个点跨越等深线，则通过插值计算出等高线与网格边界的相交点坐标，并将其添加到锚点集合 1 中。

step.1.3 在南北方向上使用双层循环遍历每一个点，判断这个点与下一个点是否跨越等高线。

step.1.4 若此点与下一个点跨越等深线，则通过插值计算出等高线与网格边界的相交点坐标，并将其添加到锚点集合 2 中。

step.2 将同一深度锚点排序

step.2.1 将集合 1,2 混合生成包含所有锚点的集合

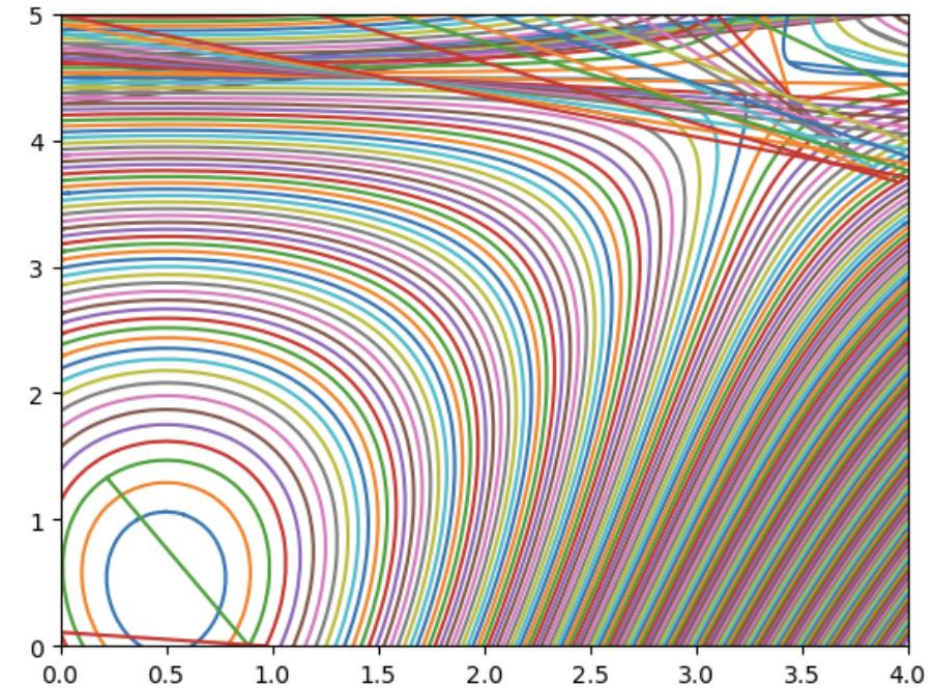
step.2.2 标记出所有在边缘上的锚点.

step2.3 按照曼哈顿距离能超过 2 的原则从边缘锚点开始给所有锚点排序.

step.3 将锚点坐标以 csv 输出.

step.4 利用 python 语言将所有锚点坐标在二维坐标系内按顺序描出,得到等深线地形图。

以下是程序画出的等深线图：



5.4.2.3 模型改进

在得到的等深线地形图中,我们可以看到有些直线与其他等深线相交.根据地理学知识,等深线不可相交,先考虑改进模型以消除这些相交的等深线.

在给锚点排序时,当曼哈顿距离大于 2 时,我们判定此处等高线不连续.因此,程序会在此高度下从另一个边界点开始继续排序,并将第二段排序得到的坐标数据与第一段的坐标数据在同一个 csv 文件中输出,当我们用 python 来按顺序描出这些离散坐标时,断点出的两个坐标会被以一条直线连接起来,因此形成了与其他等高线相交的"干扰线"。

我们可以通过在曼哈顿距离大于 2 时，则不会制这二点的连线，来消除这些干扰线。

5.4.2.4 等深线模型结果分析

将我们得到的等深线地形图与用画图软件直接绘制的分层深度图对比,可以观察出两图大体吻合,等深线拟合基本准确.

5.4.3 布线策略

5.4.3.1 模型建立

根据第二问的推论，我们知道测线的方向应当与等深线平行。因此问题转化为提取某些等深线作为测线以满足 1）尽量覆盖整个海域 2)重叠率尽量小 。要解决这个问题，我们需要建立 w 以及 eta 的数学模型。

考虑一条已知测线，我们在此测线左右两侧各 δ 米划定一条以测线为轴线的条带。提供计算此条带内所包含的等深线数量，即可估算出此条带的平均坡度。此坡度可近似视为测线对应的坡度（即 α 角）。再代入问题一中的模型，即可得出 W 与 η 的数学模型。

5.4.3.2 算法求解

等深线可看作为拥有大量拐点的折线。在每个拐点处，以与拐点相邻的线段垂直的方向做一条射线，并在上面截取与拐点距离为 δ 的点。

以与测线相邻最近的一条等深线为起点，使用贪心算法按照深度递增（减）的顺序遍历每一条等深线，直到最终覆盖率达到题设要求。

六、模型的优缺点与改进

6.1 模型的优点

（1）不依赖过于复杂的拟合方法，参数具有明确的几何意义，可解释性好；

（2）实现的算法不确定性低；

（3）适用于形状简单的海底地形快速规划测线；

6.2 模型的缺点

（1）等深线的拟合不一定适用于更复杂的海底地形，因为我们的模型在应用参数方面，根据数据特征进行取巧。此点实际上既是缺点也是优点。

（2）依赖少量海底地形数据的某些统计特征；这些特征可能没法代表所有海域的海底地形；模型可能难以迁移至其它场景。

6.3 模型的改进

可以引入较为复杂的模型拟合等深线或直接拟合海底地形，如多元多项式回归（由于计算量过大，可解释性不高，我们并未采用此方法），提高模型的可迁移性，使其成为为海底地形勘测规划测线的通用模型。

参考文献

[1]成芳,and 胡迺成."多波束测量测线布设优化方法研究." 海洋技术学报 35.02(2016):87-91.

附录

一、问题一 java 代码

```
// all angle should be radius, all length in meter UNLESS STATED

import java.lang.Math;
import java.util.Scanner;
// import java.util.zip.CRC32;

public class Question1{
    // public double alpha = 1.5 * Math.PI / 180;
    public double alpha = toRad(1.5);
    public double D = 70;
    public double theta = toRad(120);
    public double d = 200;

    public static Question1 obj = new Question1();

    public double toRad(double degree){
        return degree * Math.PI / 180;
    }

    public double nauticalToMeter(double nau){
        return nau * 1852;
    }

    public double returnh(double n){
        double s = n * d;
        return D - s * Math.tan(alpha);
    }

    public double returnhs(double s){
        return D - s * Math.tan(alpha);
    }

    public double calx1(double h){
        return h / (Math.tan(alpha) - Math.tan(Math.PI/2 - theta/2));
    }

    public double calx2(double h){
        return h / (Math.tan(alpha) + Math.tan(Math.PI/2 - theta/2));
    }

    public double returnW(double h){
        return calx2(h) - calx1(h);
    }

    public double returneta(double n, double h, double W)
    {
        double hprime = returnh(n - 1);
        double x1 = calx1(h);
        double x2prime = calx2(hprime) - d;
        double eta = (x2prime - x1) / W;
```

```
        // System.out.println(hprime);
        // System.out.println(x2prime);

        return eta;
    }

    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        double n = input.nextDouble();
        input.close();

        // // new object
        // Question1 obj = new Question1();

        // calculate h
        double h = obj.returnh(n);
        System.out.println("h = " + h);

        // calculate W
        double W = obj.returnW(h);
        System.out.println("W = " + W);

        // calculate eta
        double eta = obj.returneta(n, h, W);
        System.out.println("eta = " + eta*100 + "%");
    }
}
```

二、 问题二 java 代码

```
import java.lang.Math;
import java.util.Scanner;

public class Question2 {
    public static Question1 q1 = new Question1();
    public static Question2 obj = new Question2();

    public double calNewAlpha(double oldAlpha, double beta){
        return Math.atan(Math.tan(oldAlpha) * Math.cos(beta - q1.toRad(90)));
    }

    public double nauticalToMeter(double nau){
        return nau * 1852;
    }

    public static void main(String[] args){
        // initialize data
        double ans[][] = new double[8][8];
        q1.theta = q1.toRad(120);
        q1.D = 120;
        q1.d = obj.nauticalToMeter(0.3);
        double oldAlpha = q1.toRad(1.5);
```

```
        // loop answer
        for(int betaIndex = 0; betaIndex <= 7; betaIndex ++)
        {
            for(int n = 0; n <= 7; n ++)
            {
                // initialize
                double beta = q1.toRad(betaIndex * 45);
                q1.alpha = obj.calNewAlpha(oldAlpha, beta);

                double h = q1.returnh(n);
                ans[betaIndex][n] = q1.returnW(h);
            }
        }

        // print answer
        for (int i = 0; i < ans.length; i++) {
            for (int j = 0; j < ans[i].length; j++) {
                String formattedNumber = String.format("%.2f", ans[i][j]);
                System.out.print(formattedNumber + " ");
            }
            System.out.println();
        }
    }
}
```

三、问题三 java 代码

```
import java.lang.Math;

public class Question3 {
    public static Question1 q1 = new Question1();
    public static Question3 obj = new Question3();

    public void printCurLine(){

System.out.println(Thread.currentThread().getStackTrace()[2].getLineNumber());
    }

    public double calEtaQ3(double curPos, double lastPos){
        double hCur = q1.returnhs(curPos);
        double x1 = q1.calx1(hCur);
        double hLast = q1.returnhs(lastPos);
        double x2prime = q1.calx2(hLast) - (curPos - lastPos);
        double W = q1.returnW(hCur);

        return (x2prime - x1) / W;
    }

    public double biSearchEta(double left, double right, double lastPos){
        double delta = 0.00001;
        double temp = 0;
        // System.out.println("temp left right" + temp+ " " +left+" " + right);
```

```
    // System.out.println("here " + temp);

    while(true){
        temp = (left + right) / 2;
        // System.out.println("temp left right" + temp+ " " +left+" " + right);
        double curEta = calEtaQ3(temp, lastPos);
        // System.out.println(curEta);
        // obj.printCurLine();
        if(0.1 <= curEta && curEta <= 0.1 + delta)
        {
            // System.out.println("Current Pos: " + temp + " eta " + curEta);
            // System.out.println(temp +" "+curEta);
            return temp;
        }
        else if(curEta < 0.1)   // too far
        {
            // System.out.println("curEta: " + curEta);
            // obj.printCurLine();
            right = temp;
            // System.out.println("right " + right);
        }
        else{
            // obj.printCurLine();
            left = temp;
        }
    }
}

public static void main(String[] args){
    // initialize
    double nslen = q1.nauticalToMeter(2);
    double welen = q1.nauticalToMeter(4);
    // System.out.println(welen);
    q1.alpha = q1.toRad(1.5);
    // See most west is central
    q1.D = 110 + welen / 2 * Math.tan(q1.alpha);
    q1.theta = q1.toRad(120);

    int lineIdx = 0;
    double linePos[] = new double[1000];

    // cal first position
    linePos[lineIdx] = q1.D * Math.tan(q1.theta / 2);
    // System.out.println(linePos[lineIdx]);
    lineIdx ++;

    // cal lastSectionLeft
    double h0 =  110 - welen / 2 * Math.tan(q1.alpha);
    double lastSectionLeft = welen - h0 * Math.tan(q1.theta / 2);
    // System.out.println("LSL" + lastSectionLeft);

    // obj.printCurLine();
    // while pos not in range, bisearch find next
```

```
        do{
            linePos[lineIdx] = obj.biSearchEta(linePos[lineIdx - 1], welen,
linePos[lineIdx -1]);
            lineIdx ++;
            // System.out.println("index Pos" + lineIdx + " " + linePos[lineIdx -
1]);
        }while(linePos[lineIdx - 1] <= lastSectionLeft);

        // obj.printCurLine();

        // cal total len
        System.out.println(lineIdx + " lines in total;");
        for(int i = 0; i < lineIdx; i ++)
        {
            System.out.printf("%.2f, ", linePos[i]);
        }
        System.out.println();

        System.out.println("Total length = " + lineIdx * nslen + " meters");
    }
}
```

四、问题四建立模型，计算等深线，航线路程，近似坡度 C++代码

```
#include <bits/stdc++.h>

using namespace std;

#define __DEBUG__ false

#define coord_2d std::pair<double,double>

#define xc(coord) coord.first
#define yc(coord) coord.second
#define unit(coord)
make_pair(coord.first/sqrt(coord.first*coord.first+coord.second*coord.sec
ond),
coord.second/sqrt(coord.first*coord.first+coord.second*coord.second))
#define vecsum(coordA, coordB) make_pair(coordA.first+coordB.first,
coordA.second+coordB.second)
#define vecscl(coord, c) make_pair(coord.first*c, coord.second*c)
#define GRID 0.02

#define manh_dist(coord_a, coord_b) ( abs(coord_a.first-coord_b.first) +
abs(coord_a.second-coord_b.second) )

const double CONTOUR_DEPTH_BEGIN=21.005;
const double CONTOUR_DEPTH_END=196.005;
const double CONTOUR_DEPTH_STEP=1.000;

double field[251][201];
```

```
void _read(std::string file_name){
    FILE* raw_data=fopen(file_name.c_str(), "r+");
    FILE* raw_data_clone=fopen("raw_data_clone.txt", "w+");
    for(int i=0;i<251;++i){
        for(int j=0;j<201;++j){
            fscanf(raw_data, "%lf", &field[i][j]);
            fprintf(raw_data_clone, "%2.2lf\t",field[i][j]);
        }   fprintf(raw_data_clone, "\n");
    }
    fclose(raw_data);
    fclose(raw_data_clone);
}

/**
 * @brief Find anchors of the contour along east-west grid borders.
 *
 * @param height
 * @return std::vector<coord_2d>
 */
std::vector<coord_2d> findAnchorsAlongEWGrids(double height){
    std::vector<coord_2d> anchors;  // Anchors are points, or coordinates
at which the contour intersects with the field.
    for(int i=0;i<251;++i){ // For every line of vertices...
        double global_y=i*GRID;
        for(int j=0;j<200;++j){     // For each vertex in the line, see
if it's on different sides of the contour than its next vertex.
            double global_x=j*GRID;
            if( (field[i][j]<=height && field[i][j+1]>height) ||
(field[i][j]>=height && field[i][j+1]<height) ){
                bool inc_flag=field[i][j]<field[i][j+1]; // Whether the
field's height is increasing along the grid-border.
                double _lvert=inc_flag?field[i][j]:field[i][j+1];
// Height of higher vertex.
                double _hvert=inc_flag?field[i][j+1]:field[i][j];
// Height of lower vertex.

                double local_x=GRID*(height-_lvert)/(_hvert-_lvert);
// Interpolate for the x-distance between the anchor and the lower
vertex.

                double real_x=global_x+(inc_flag?local_x:(GRID-local_x));
// Obtain x coordinate of the anchor.

                anchors.push_back(make_pair(real_x, global_y));
            }
        }
    }
    return anchors;
}

/**
 * @brief Find anchors of the contour along north-south grid borders.
 *
```

```
 * @param height
 * @return std::vector<coord_2d>
 */
std::vector<coord_2d> findAnchorsAlongNSGrids(double height){
    std::vector<coord_2d> anchors;  // Anchors are points, or coordinates
at which the contour intersects with the field.
    for(int i=0;i<201;++i){ // For every column of vertices...
        double global_x=i*GRID;
        for(int j=0;j<250;++j){     // For each vertex in the line, see
if it's on different sides of the contour than its next vertex.
            double global_y=j*GRID;
            if( (field[j][i]<=height && field[j+1][i]>height) ||
(field[j][i]>=height && field[j+1][i]<height) ){
                bool inc_flag=field[j][i]<field[j+1][i]; // Whether the
field's height is increasing along the grid-border.
                double _lvert=inc_flag?field[j][i]:field[j+1][i];
// Height of higher vertex.
                double _hvert=inc_flag?field[j+1][i]:field[j][i];
// Height of lower vertex.

                double local_y=GRID*(height-_lvert)/(_hvert-_lvert);
// Interpolate for the y-distance between the anchor and the lower
vertex.

                double real_y=global_y+((inc_flag)?local_y:(GRID-
local_y));     // Obtain y coordinate of the anchor.

                anchors.push_back(make_pair(global_x, real_y));
            }
        }
    }
    return anchors;
}

/**
 * @brief Print a list of coordinates at which the contour intersects
with east-west grid borders on the console.
 *
 */
void test_ew_grid_anchors(double depth){
    std::vector<coord_2d> ew_grid_anchors=findAnchorsAlongEWGrids(depth);
    for(int i=0;i<ew_grid_anchors.size();++i){
        printf("<%1.5lf,%1.5lf>\n", ew_grid_anchors.at(i).first,
ew_grid_anchors.at(i).second);
    }
}

/**
 * @brief Print a list of coordinates at which the contour intersects
with east-west grid borders on the console.
 *
 */
void test_ns_grid_anchors(double depth){
```

```
    std::vector<coord_2d> ns_grid_anchors=findAnchorsAlongNSGrids(depth);
    for(int i=0;i<ns_grid_anchors.size();++i){
        printf("<%1.5lf,%1.5lf>\n", ns_grid_anchors.at(i).first,
ns_grid_anchors.at(i).second);
    }
}

std::vector<coord_2d> get_contour(double depth, bool
discontinue_contour){
    std::vector<coord_2d> anchors=findAnchorsAlongEWGrids(depth);   //
Find all anchors along ew-grid-borders.

    std::vector<coord_2d> _ns_anchors_=findAnchorsAlongNSGrids(depth);
    anchors.insert(anchors.end(), _ns_anchors_.begin(),
_ns_anchors_.end());    // Include all anchors along ns_grid_borders.

    int ancsiz=anchors.size();  // Number of anchors in total.

    bool *ancflags=new bool[ancsiz];    // Marks if each of the anchor is
visited.
    for(int i=0;i<ancsiz;++i){
        ancflags[i]=false;
    }

    std::vector<int> _extreme_inds;     // Marks all the anchors on the
edge of the map.
    for(int i=0;i<ancsiz;++i){
        if(anchors.at(i).first==0.0 || anchors.at(i).second==0.0 ||
anchors.at(i).first==4.0 || anchors.at(i).second==5.0){
            _extreme_inds.push_back(i);
        }
    }
    printf("Extreme Points acquired:\n");
    for(int i=0;i<_extreme_inds.size();++i){
        printf("%d\t<%.5lf,%.5lf>\t\n", _extreme_inds.at(i),
anchors.at(_extreme_inds.at(i)).first,
anchors.at(_extreme_inds.at(i)).second);
    }   printf("\n");

    int cur_anc=_extreme_inds.at(0);  // Start visiting anchors from the
first extreme point.
    printf("Iterating from extreme point <%.5lf,%.5lf>\n",
anchors.at(cur_anc).first, anchors.at(cur_anc).second);
    std::vector<coord_2d> contour;

    while(1){
        ancflags[cur_anc]=true;     // Mark the current anchor to be
visited.
        contour.push_back(anchors.at(cur_anc));       // Add the current
anchor to the route.

        bool forward_flag=false;    // Marks whether the forward anchor
is found.
```

```
        double cur_min_manh_dist=2.0;   // Stores the minimum manhattan
distance to the forward anchor found so far.

        // Iterate through all anchors and search for the forward anchor
that the contour progresses to.
        for(int i=0;i<ancsiz;++i){  // For each of the anchors in the
list...
            if(manh_dist(anchors.at(i), anchors.at(cur_anc))<(2.0*GRID)
&& !ancflags[i]){  // If a first anchor within manhattan distance of 2 is
found unvisited...
                if(manh_dist(anchors.at(i), anchors.at(cur_anc))<
cur_min_manh_dist){  // If the minimum manhattan distance is updated...
                    cur_anc=i;  // Visit this anchor.
                    cur_min_manh_dist=manh_dist(anchors.at(i),
anchors.at(cur_anc));
                    forward_flag=true;  // Mark that a forward anchor is
found.
                }
            }
        }


        if(!forward_flag){  // If no forward anchor is found, check if
there's any anchor at extremes yet to be visited.
            if(!discontinue_contour){break;}     // If the graph doesn't
allow discontinuity, end the loop
            bool fill_flag=true;    // Check if all anchors are visited.
            int unfill_index=-1;    // Marks index of the first unvisited
anchor.
            for(int i=0;i<_extreme_inds.size();++i){
                if(!ancflags[_extreme_inds.at(i)]){
                    fill_flag=false;
                    unfill_index=_extreme_inds.at(i);
                    break;
                }
            }

            if(!fill_flag){     // If any anchor is left unvisited...
                cur_anc=unfill_index;   // Jump to the first unfilled
index and resume the trip.
                printf("Iterating from extreme point <%.5lf,%.5lf>\n",
anchors.at(cur_anc).first, anchors.at(cur_anc).second);
                continue;
            }else{
                break;  // Break the loop if no anchor is left unvisited.
            }
        }
    }

    delete [] ancflags;
    return contour;
}
```

```
void test_contour(double depth, bool discontinue_contour=false){
    std::vector<coord_2d> _contour_=get_contour(depth, true);

    for(int i=0;i<_contour_.size();++i){
        printf("<%1.5lf,%1.5lf>\n", _contour_.at(i).first,
_contour_.at(i).second);
    }
}

void write_contour_to_file(std::vector<coord_2d> contour, std::string
file_name){
    FILE* _file_=fopen(file_name.c_str(), "w+");
    fprintf(_file_,"XCOORD,YCOORD\n");
    for(int i=0;i<contour.size();++i){
        fprintf(_file_, "%.5lf,%.5lf\n", contour.at(i).first,
contour.at(i).second);
    }
    fclose(_file_);
}

/** Analysis of contours */

/**
 * @brief Evaluate the span of an interval of the contour over x and y.
Result represented as vector-2d.
 *
 * @param contour
 * @param cur_ind Vertex at which the interval vector begins.
 * @param forward_flag Whether the calculation should use the next
interval from the vertex at cur_ind.
 *                  Set to false to use the previous interval.
 * @return coord_2d
 */
coord_2d interval_vector(std::vector<coord_2d> contour, int cur_ind, bool
forward_flag){
    coord_2d cur_pos=contour.at(cur_ind);
    coord_2d next_pos=contour.at(cur_ind+ (forward_flag?1:-1) );

    coord_2d local_pos=make_pair(next_pos.first-cur_pos.first,
next_pos.second-cur_pos.second);
    return local_pos;
}

/**
 * @brief Evaluate Length of an interval
 *
 * @param contour
 * @param cur_ind Vertex at which the interval vector begins.
 * @param forward_flag Whether the calculation should use the next
interval from the vertex at cur_ind.
 *                  Set to false to use the previous interval.
 * @return double
 */
```

```
double interval_length(std::vector<coord_2d> contour, int cur_ind, bool
forward_flag){
    coord_2d local_pos=interval_vector(contour, cur_ind, forward_flag);

    if(manh_dist(contour.at(cur_ind), contour.at(cur_ind+
(forward_flag?1:-1) )) <=2){
        return 0.0;
    } else{
        return sqrt(local_pos.first*local_pos.first +
local_pos.second*local_pos.second);
    }
}



/**
 *
 * @param contour
 * @return double Length of the designated contour.
 */
double lengthOf(std::vector<coord_2d> contour){
    double sum = 0;

    for(int i = 0; i < contour.size()-1; i ++){
        sum+=interval_length(contour, i, true);
    }

    return sum;
}

/**
 * @brief Evaluate the horizontal projection of normal of the contour on
an interval. Result represented in 2D-unit-vector.
 *
 * @param contour
 * @param cur_ind
 * @param forward_flag
 * @return coord_2d
 */
coord_2d interval_normal(std::vector<coord_2d> contour, int cur_ind, bool
forward_flag){
    coord_2d intvl_vec=interval_vector(contour, cur_ind, forward_flag);

    coord_2d normal_vec=make_pair(0.0-intvl_vec.second, intvl_vec.first);

    coord_2d unit_normal_vec=unit(normal_vec);

    return unit_normal_vec;
}

std::pair<std::vector<coord_2d>, std::vector<coord_2d>>
getScanBounds(std::vector<coord_2d> contour, double scan_width){
    std::vector<coord_2d> lbound;
```

```
    std::vector<coord_2d> rbound;

    coord_2d cur_unit_normal;
    for(int i=0;i<contour.size()-1;++i){
        cur_unit_normal=interval_normal(contour, i, true);
        lbound.push_back(vecsum(contour.at(i), vecscl(cur_unit_normal,
scan_width)));
        lbound.push_back(vecsum(contour.at(i+1), vecscl(cur_unit_normal,
scan_width)));
        rbound.push_back(vecsum(contour.at(i), vecscl(cur_unit_normal,
0.0-scan_width)));
        rbound.push_back(vecsum(contour.at(i+1), vecscl(cur_unit_normal,
0.0-scan_width)));
    }

    return make_pair(lbound, rbound);
}

/**
 * @brief Returns interpolated depth at given water-surface coordinate.s
 *
 * @param acoord
 * @return double
 */
double anchor_depth(coord_2d acoord){
    double sw_corner_depth, se_corner_depth, nw_corner_depth,
ne_corner_depth;  // Acquire local grid corner depth.

sw_corner_depth=field[(int)floor(xc(acoord)/GRID)][(int)floor(yc(acoord)/
GRID)];
    se_corner_depth=field[(int)ceil(xc(acoord))][(int)floor(yc(acoord))];
    nw_corner_depth=field[(int)floor(xc(acoord))][(int)ceil(yc(acoord))];
    ne_corner_depth=field[(int)ceil(xc(acoord))][(int)ceil(yc(acoord))];

    coord_2d global_coord=make_pair(floor(xc(acoord)/GRID)*GRID,
floor(yc(acoord)/GRID)*GRID);

    double n_intersect_depth=nw_corner_depth + ( (xc(acoord)-
xc(global_coord))/GRID ) * (ne_corner_depth-nw_corner_depth);
    double s_intersect_depth=sw_corner_depth + ( (xc(acoord)-
xc(global_coord))/GRID ) * (se_corner_depth-sw_corner_depth);

    double local_depth=s_intersect_depth + ( (yc(acoord)-
yc(global_coord))/GRID ) * (n_intersect_depth-s_intersect_depth);

    return local_depth;
}

double evaluateAngleByNeighbourContourDensity(std::vector<coord_2d>
contour, double scan_width, double centre_depth, double depth_step){
    std::pair<std::vector<coord_2d>, std::vector<coord_2d>>
scan_bounds=getScanBounds(contour, scan_width);
```

```
    std::vector<double> lbound_depths;
    std::vector<double> rbound_depths;
    for(int i=0;i<scan_bounds.first.size();++i){
        lbound_depths.push_back(anchor_depth(scan_bounds.first.at(i)));
    }
    for(int i=0;i<scan_bounds.second.size();++i){
        rbound_depths.push_back(anchor_depth(scan_bounds.second.at(i)));
    }

    double depth_neib_hb=centre_depth;
    double depth_neib_lb=centre_depth;

    while(1){
        bool aflag=false;
        for(int i=0;i<lbound_depths.size();++i){
            if(lbound_depths.at(i)>depth_neib_hb){
                aflag=true;
                break;
            }
        }
        if(!aflag){break;}
        else{
            depth_neib_hb+=depth_step;
        }
    }

    while(1){
        bool aflag=false;
        for(int i=0;i<rbound_depths.size();++i){
            if(rbound_depths.at(i)<depth_neib_lb){
                aflag=true;
                break;
            }
        }
        if(!aflag){break;}
        else{
            depth_neib_hb-=depth_step;
        }
    }


    return atan( (depth_neib_hb-depth_neib_lb+depth_step) /
(2*scan_width));


}

void writeContourAnalysis(std::string analysis_file_name){
    FILE* _file=fopen(analysis_file_name.c_str(), "w+");

    int csv_serial=0;
```

```
    for(double
cur_depth=CONTOUR_DEPTH_BEGIN;cur_depth<CONTOUR_DEPTH_END;cur_depth+=CONT
OUR_DEPTH_STEP){
        write_contour_to_file(get_contour(cur_depth, true),
"contour_map_"+to_string(csv_serial)+".csv");
        csv_serial++;
    }

    for(int i=0;i<csv_serial;++i){
        double cur_depth=CONTOUR_DEPTH_BEGIN+i*CONTOUR_DEPTH_END;
        fprintf(_file, "[CONTOUR-DEPTH]%.5lf\t[ESTIMATED-SLOPE]%.5lf\n",
                cur_depth,

evaluateAngleByNeighbourContourDensity(get_contour(cur_depth,true),
100.0, cur_depth, 2.0)
                );

    }
}

void debug(){
    _read("raw_data.txt");
    test_contour(70.005, true);
}

int main(){
    if(__DEBUG__){
        debug();
        exit(0);
    }
    _read("raw_data.txt");

    writeContourAnalysis("analysis.txt");




    exit(0);
}
```

五、问题四 c++渲染绘制等高线代码

```
#include <bits/stdc++.h>
using namespace std;

double dat[251][201];

void _read(std::string file_name){
    FILE* raw_data=fopen(file_name.c_str(), "r+");
    FILE* raw_data_clone=fopen("raw_data_clone.txt", "w+");
    for(int i=0;i<251;++i){
        for(int j=0;j<201;++j){
            fscanf(raw_data, "%lf", &dat[i][j]);
            fprintf(raw_data_clone, "%2.2lf\t",dat[i][j]);
        }   fprintf(raw_data_clone, "\n");
```

```
    }
    fclose(raw_data);
    fclose(raw_data_clone);
}

void _write(std::string file_name){
    FILE* pix_data=fopen(file_name.c_str(), "w+");
    fprintf(pix_data, "%d %d\n", 200, 250);
    for(int i=0;i<250;++i){
        for(int j=0;j<200;++j){
            int cur_pix=16-((int)( (dat[i][j]/200.0)*16.0 ));
            fprintf(pix_data, "%d ", cur_pix);
        }
        fprintf(pix_data, "\n");
    }
    fclose(pix_data);
}

void make(){
    _read("raw_data.txt");
    _write("gs_bitmap.txt");
}

// int main(){make();}
```