

# 基于二分法的板凳龙运动分析

## 摘 要

“板凳龙”，是浙闽地区的一种民间传统舞蹈活动。本文基于二分法，分析了板凳龙不同行进模式下的运动状态，求解各轨迹内所有把手的位置坐标及速度，通过分别优化螺距、调头旋转角以及龙头前把手速度，实现不同条件下舞龙队自如地行进。

**针对问题一：**首先推导螺线上弧长计算公式，计算每一秒内龙头沿弧运动距离，从而确定龙头前把手坐标。接着根据该坐标，采用**螺旋角二分法**，确定后续每一个把手在螺线上的位置坐标。针对每一点的速度，我们采用**微元法**，计算在很小一段时间微元内每个点在弧上走过的距离，从而得到速度。具体求解结果见正文表 1, 2 及附件。

**针对问题二：**首先计算各把手坐标，再通过将木板抽象成矩形的方法利用把手坐标求得矩形四端点坐标。利用欧氏几何中判断两边相交的**外积方法**作为碰撞判断条件，通过**时间二分法**，求得区间大小小于浮点数数据误差 $eps(10^{-12})$ 时区间的下界，并将其作为**终止时刻**，结果为**412.469482s**，我们将此时间重输入问题一中函数，求得此时整个舞龙队把手的位置坐标和运动速度。具体求解结果见正文表 3 及附件。

**针对问题三：**首先根据螺线与圆的联立方程，确定恰进入时龙头前把手的旋转角及位置坐标。根据问题一中**螺旋角二分法**求得所有把手位置坐标。再根据问题二的碰撞判断条件，通过**螺距二分法**，使得二分区间大小小于 $eps$ 。此时区间的下界即为所求**最小螺距**，结果为**42.021690cm**。

**针对问题四：**首先分析调头点与调头曲线对应关系以及调头曲线长度影响因素，在此基础上二分**旋转角**，遍历尚未碰撞的调头点。在每个调头点确定的 S 形曲线上，将龙头把手以**定步长遍历搜索**整条曲线，利用**二分数轴方法**，求出每种位置状态下所有后续把手坐标并记录，判断是否碰撞及是否存在龙头前进龙身后退的不合理情况。重复二分过程，直至区间大小小于 $eps$ ，此时旋转角即为所求**最小掉头旋转角**，结果为 **15.696400rad**。最后，根据所得到的螺线方程及 S 形调头曲线方程，结合问题一中坐标推导模型及速度计算模型，求解对应秒舞龙队把手的位置和速度。具体求解结果见正文表 4, 5 及附件。

**针对问题五：**首先对把手行进速度影响要素进行分析，得出位置坐标与速度关联。接着根据问题四**数轴坐标二分模型**，将把手行进速度用龙头行进速度表征。然后在问题四给定行进路径上，采用**速度二分法**，将龙头把手以定步长遍历搜索整条曲线，在每个位置下生成后续所有把手，并利用**微元法**求解所有把手的速度，判断是否超过  $2m/s$ 。重复二分，直至区间大小小于 $eps$ ，取区间的下界作为**龙头最大行进速度**，求得结果为 **109.515381cm/s**。

本文最后进行了坐标求解模型的**灵敏度分析**，对速度和螺距进行微小扰动，所得灵敏度参照值与标准值 1 偏差均小于 2%，可见模型具有良好的**鲁棒性**。

**关键词：**二分法 微元法 板凳龙 定步长遍历搜索 欧氏几何

## 一、问题重述

### 1.1 问题背景

“板凳龙”，是浙闽地区的一种民间传统舞蹈活动。人们将多条木制板凳连接在一起，通过一系列舞蹈动作，来模拟龙蜿蜒曲折的活动形态。盘龙，作为“板凳龙”表演中的经典动作，需要舞龙者在舞台上围绕一个中心进行旋转。龙头在前引路，龙身龙尾在后盘旋相随。盘龙动作队列变换复杂，需要所有舞龙者的高度配合，如何在保证舞龙正常盘入盘出的前提下，减少盘龙面积，提高行进速度，是舞龙队提高观赏性的关键所在。

### 1.2 问题提出

一种板凳龙是由 223 节板凳拼接构成，其中第一节为 341cm 板长的龙头，最后一节为 220cm 板长的龙尾，中间 221 节为 220cm 板长的龙身，所有板宽 30cm。板与板之间通过打孔立把手连接，孔径 5.5cm，孔中心离板边缘 27.5cm。

基于以上背景及相关数据，我们需要建立数学模型来解决下述舞龙运动问题：

**问题一：**固定盘入螺距 55cm，龙头运动速度 1m/s，根据舞龙队起点建立对应运动轨迹模型，计算从初始时刻到 300s 为止，每一秒整个舞龙队内所有把手的位置坐标及行进速度。

**问题二：**在问题一的基础上，进一步确定舞龙队内部发生碰撞，盘龙终止的时刻，计算此时舞龙队内所有把手的位置坐标及行进速度。

**问题三：**允许盘入螺距可变，固定掉头空间为一个以螺线中心为圆心，直径为 9m 的圆，计算能够使龙头前把手沿螺线进入掉头空间边界时的最小运动螺距。

**问题四：**固定盘入螺距 1.7m，要求在问题三的掉头空间内完成掉头，满足掉头路径为两段相切圆弧组成的 S 形曲线，并使得后一段半径为前一段的一半，圆弧与螺线相切，计算此时最短的掉头曲线。

**问题五：**舞龙队以问题四的运动螺距及掉头曲线行进，保证龙头运动速度不变，各把手速度不超过 2m/s，计算龙头最大运动速度。

## 二、问题分析

### 2.1 问题一的分析

问题一要求我们在给定的盘入螺距和龙头运动速度下计算出 0-300s 内整个舞龙队把手每秒的位置坐标和运动速度。首先我们将盘入螺线延长，龙头位于 A 点，龙身及龙尾在后续延长线上，使得盘入轨迹始终保持螺形。通过推导螺线弧长计算公式计算每一秒内龙头沿弧距离，确定龙头前把手坐标。接着根据该坐标，采用旋转角二分法，在螺线上确定后续每一把手位置。针对每一点的速度，我们取极短一段时间作为时间微元，计算在该时间微元内每一把手的位移。当微元足够小时，我们认为该段时间内的平均速度即为每一把手的速度。

### 2.2 问题二的分析

问题二要求我们在给定的盘入螺距和龙头运动速度下确定舞龙队内部碰撞时刻，以及此时整个舞龙队把手的位置坐标和运动速度。在问题一中，我们所封装的函数可

以求出任意时刻把手的位置。所以我们首先通过建系，将每个板凳抽象为空间中四点形成的矩形，板凳边与边的碰撞可视为不相邻矩形在空间中重合。接着调整代码，将欧式几何中判定两边相交的外积方法作为碰撞判断条件。然后，我们取定一个碰撞时间与一个不碰撞时间，通过不断二分时间，找到二分区间大小小于浮点数数据误差 $eps$  ( $10^{-12}$ ) 时区间的下界，并将其作为最后时间结果。最后我们将该时间重输入程序，得到此时整个舞龙队把手的位置坐标和运动速度。

### 2.3 问题三的分析

问题三要求我们计算龙头前把手恰能够沿着螺线进入掉头空间边界时的最小盘入螺距。我们可以将其理解为在此螺距下，舞龙队内部碰撞时，龙头前把手正好在以螺线中心为圆心，直径  $9m$  的圆形上。我们首先根据螺线与圆的联立方程，确定碰撞时龙头前把手的旋转角，从而确定龙头前把手坐标。接着根据问题一封装函数，由龙头前把手推导后续所有把手的位置坐标。然后根据问题二的碰撞判断条件，取一碰撞螺距作为上界，未碰撞螺距作为下界，不断二分螺距，使得二分区间大小小于 $eps$ 。此时区间的下界即为所求最小螺距。

### 2.4 问题四的分析

问题四要求我们在给定螺距大小和调头曲线形状下，调整调头曲线圆弧，使得最终调头曲线变短，并给出在此行进轨迹下每秒舞龙队所有把手的位置和速度。我们首先简化模型分析调头点与调头曲线的关系，以及调头曲线长度的影响因素。然后依据分析结果以及问题二的碰撞判断条件判断，先二分旋转角，选取已碰撞的调头点为下界，未碰撞的调头点为上界，遍历尚未碰撞的调头点。在每个调头点确定的 S 形曲线中，采用龙头把手定步长遍历搜索的方法，模拟舞龙队不同时间内在该 S 形曲线的位置状态。接着我们将弧长抽象为数轴坐标，利用二分数轴的方法，将每种位置状态下所有把手坐标全部求出并记录，判断是否碰撞，以及是否出现龙头前进龙身后退的非合理情况。若出现上述情况，则将此旋转角设为新的下界；若未出现，则将此旋转角设为新的上界。重复二分过程，直至上下界区间大小小于 $eps$ ，将此时的旋转角记为所求最小调头旋转角，此时所生成的调头曲线最短。最后，我们依据所得到的螺线方程以及 S 形调头曲线方程，根据问题一坐标推导模型以及速度计算模型求解每秒舞龙队把手的位置和速度。

### 2.5 问题五的分析

问题五要求我们在问题四的行进路径中保持龙头速度不变以及各把手速度不超过 $2m/s$ 的条件下，确定龙头的最大行进速度。我们首先分析出单位时间内，龙头的位置坐标可由龙头行进速度表征，再根据问题四数轴坐标二分模型，所有把手坐标均可用龙头行进速度表征。我们取很短的一段时间微元，在这段时间里所有把手的平均速度可看为把手的行进速度，根据问题一速度求解，这同样可用龙头行进速度表征。所以我们在给定行进路径中，让龙头前把手以  $5cm$  搜索遍历，取较大的一个速度 $v$ 作为上界，满足在搜索遍历过程中存在有把手速度超过 $2m/s$ ，取较小的一个速度 $v$ 作为下界，满足在搜索遍历过程中所有把手速度均不超过 $2m/s$ ，重复二分速度，直到区间大小小于 $eps$ ，取区间的下界作为龙头最大行进速度。

### 三、模型假设

1. 假设所有表演者舞龙技术精湛，不存在表演失误。
2. 忽略不同计算设备浮点数数据误差 $eps$ 的大小差异。

### 四、符号说明

| 符号       | 说明                      | 单位   |
|----------|-------------------------|------|
| $S$      | 螺距                      | cm   |
| $t$      | 行进时间                    | s    |
| $l_i$    | 第 $i$ 个把手               | /    |
| $\theta$ | 旋转角                     | rad  |
| $v$      | 把手运动速度                  | cm/s |
| $eps$    | 浮点数数据误差（定值 $10^{-12}$ ） | /    |

### 五、模型的建立与求解

五个问题整体求解流程图如图所示：

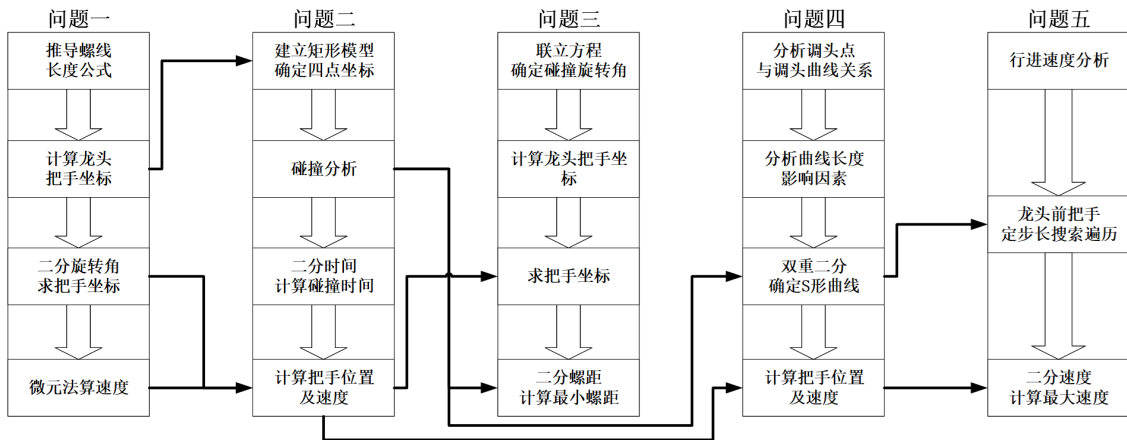


图 1：求解流程图

#### 5.1 问题一模型的建立与求解

##### 5.1.1 初始舞龙队确定

由于题目仅指出，初始时龙头位于螺线 A 点处，并未说明初始时龙身龙尾位置。为便于模型求解，我们可将盘入轨迹延长，将龙身龙尾放于后续延长线上，如图 2：

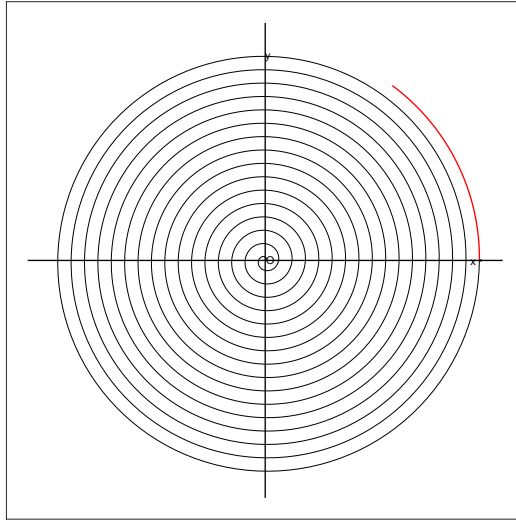
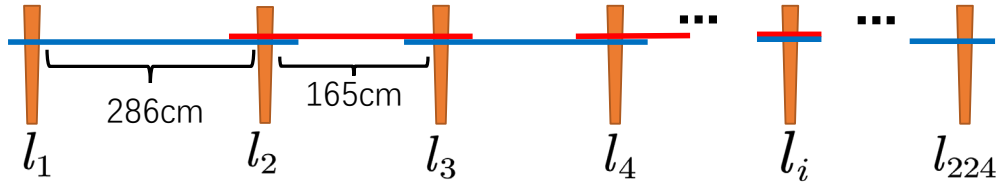


图 2：舞龙队初始图

其中 O 点为螺线中心，A 点为初始时龙头把手位置，OA 距离 880cm，黑色为盘入螺线轨迹，红色为舞龙队初始示意线。

### 5.1.2 龙头前把手坐标推导

在此问中，孔径大小对位置坐标求解没有影响，我们将所有把手都视为一点，如图 3：



(龙头前把手)

图 3：板凳示意图

其中  $l_i$  代表第  $i$  号把手。

在螺线上的把手点坐标满足：

$$\begin{cases} x = \frac{S}{2\pi} \theta \cos \theta \\ y = \frac{S}{2\pi} \theta \sin \theta \end{cases} \quad (1)$$

其中  $S$  为螺距， $\theta$  为把手的旋转角。我们可以利用积分的方式，算出把手旋转角从  $\theta_1$  到  $\theta_2$  时，所走过的螺线长度

$$L = \int_{\theta_1}^{\theta_2} \sqrt{\left(\frac{dx}{d\theta}\right)^2 + \left(\frac{dy}{d\theta}\right)^2} d\theta = \frac{S}{2\pi} \int_{\theta_1}^{\theta_2} \sqrt{1 + \theta^2} d\theta \quad (2)$$

$$L = \frac{S}{4\pi} \left[ \theta \sqrt{1 + \theta^2} + \ln(\theta + \sqrt{1 + \theta^2}) \right] \Big|_{\theta_1}^{\theta_2}$$

据此我们可以计算螺线的总长度为 44259cm。

由于龙头前把手的运动速度恒为 1m/s，在一定时间里走过的弧线长度为定值，我们可以据此反解方程（3），求得龙头前把手一定时间内在螺线上的旋转角 $\theta_h$

$$L = \frac{S}{2\pi} \int_{32\pi}^{\theta_h} \sqrt{1 + \theta^2} d\theta = 1 \times t \quad (3)$$

将所得到的 $\theta_h$ 带入到（1）的参数方程中，可得到龙头前把手坐标：

$$\left( \frac{s}{2\pi} \theta_h \cos \theta_h, \frac{s}{2\pi} \theta_h \sin \theta_h \right) \quad (4)$$

### 5.1.3 二分求解位置坐标

将龙头前把手坐标视为第一个定点，并将其与螺线上任意两点 $a_1$ 和 $a_2$ 连接，做螺线上的割线 $L_1$ 、 $L_2$ ，保证其中一条割线 $L_1$ 长度大于 286cm，另一条割线 $L_2$ 长度小于 286cm，如图 4：

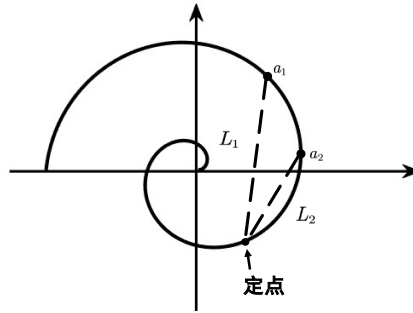


图 4：二分初始图

接着在螺线上继续取一点 $a_3$ ，满足该点旋转角为点 $a_1$ 与 $a_2$ 旋转角之和的一半，即 $\theta_3 = \frac{1}{2}(\theta_1 + \theta_2)$ ，如图 5：

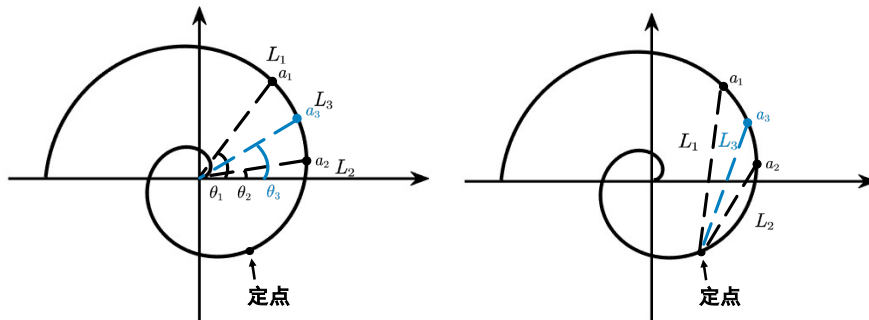


图 5：二分作点图

若 $L_3 > 286\text{cm}$ ，则在 $a_2$ 与 $a_3$ 间重复上述步骤二分取点，反之则在 $a_1$ 与 $a_3$ 间二分取点，直至找到一点 $L_x = 286\text{cm}$ 。此时点 $x$ 即为第一节龙身把手。接着以该把手为第二个定点，继续使用上述二分法，将所要比对的长度修改为165cm，可求出第二节龙身

把手。

重复上述操作，可求得在该时间内所有把手的位置。我们将此过程封装为 C++ 语言函数，可以实现给出任意时间下所有把手点的位置坐标。但由于我们初始时将龙身与龙尾进行延伸，所以所得数据中需要删除与原点距离大于  $880\text{cm}$  的点。

二分旋转角流程图如图 6 所示：

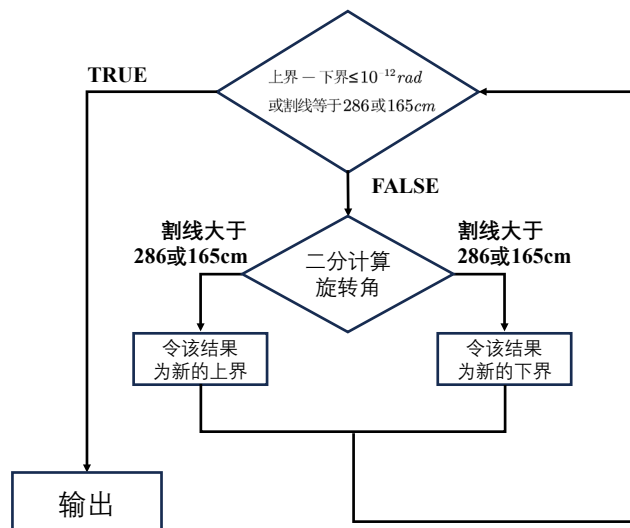


图 6：二分旋转角流程图

#### 5.1.4 点行进速度求解

根据上述函数，我们可以求得任意时间所有点的位置坐标。因此，针对每一点的速度，我们可以取极小的一段时间（此处我们记为  $10^{-4}\text{s}$ ）作为时间微元，通过计算该段时间前后把手在弧线上走过的弧长  $\Delta l$ ，如图 7：

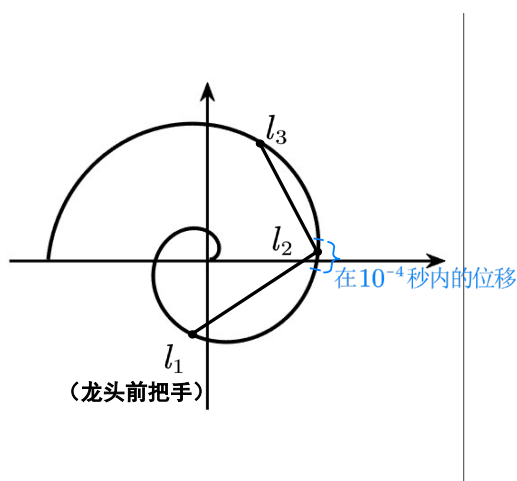


图 7：速度求解图

根据公式 (5)，我们求得该段时间内的平均速度，当时间微元足够小时，可以用此平均速度代替该时间点的瞬时速度，从而求出所有点的速度。

$$v = \frac{\Delta l}{\Delta t} \quad (5)$$

5.1.5 问题一求解结果

表 1：问题一位置结果

|                 | 0 s      | 60 s      | 120 s     | 180 s     | 240 s     | 300 s     |
|-----------------|----------|-----------|-----------|-----------|-----------|-----------|
| 龙头 x (m)        | 8.800000 | 5.799209  | -4.084887 | -2.963609 | 2.594494  | 4.420274  |
| 龙头 y (m)        | 0.000000 | -5.771092 | -6.304479 | 6.094780  | -5.356743 | 2.320429  |
| 第 1 节龙身 x (m)   | ---      | 7.456758  | -1.445473 | -5.237118 | 4.821221  | 2.459489  |
| 第 1 节龙身 y (m)   | ---      | -3.440399 | -7.405883 | 4.359627  | -3.561949 | 4.402476  |
| 第 51 节龙身 x (m)  | ---      | ---       | -5.543150 | 2.890455  | 5.980011  | -6.301346 |
| 第 51 节龙身 y (m)  | ---      | ---       | 6.377946  | 7.249289  | -3.827758 | 0.465829  |
| 第 101 节龙身 x (m) | ---      | ---       | ---       | 1.898794  | -4.917371 | -6.237722 |
| 第 101 节龙身 y (m) | ---      | ---       | ---       | -8.471614 | -6.379874 | 3.936008  |
| 第 151 节龙身 x (m) | ---      | ---       | ---       | ---       | ---       | 7.040740  |
| 第 151 节龙身 y (m) | ---      | ---       | ---       | ---       | ---       | 4.393013  |
| 第 201 节龙身 x (m) | ---      | ---       | ---       | ---       | ---       | ---       |
| 第 201 节龙身 y (m) | ---      | ---       | ---       | ---       | ---       | ---       |
| 龙尾（后）x (m)      | ---      | ---       | ---       | ---       | ---       | ---       |
| 龙尾（后）y (m)      | ---      | ---       | ---       | ---       | ---       | ---       |

表 2：问题一速度结果

|                 | 0 s      | 60 s     | 120 s    | 180 s    | 240 s    | 300 s    |
|-----------------|----------|----------|----------|----------|----------|----------|
| 龙头 (m/s)        | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| 第 1 节龙身 (m/s)   | ---      | 0.999961 | 0.999945 | 0.999917 | 0.999859 | 0.999709 |
| 第 51 节龙身 (m/s)  | ---      | ---      | 0.999538 | 0.999330 | 0.998941 | 0.998065 |
| 第 101 节龙身 (m/s) | ---      | ---      | ---      | 0.998970 | 0.998436 | 0.997302 |
| 第 151 节龙身 (m/s) | ---      | ---      | ---      | ---      | ---      | 0.996861 |
| 第 201 节龙身 (m/s) | ---      | ---      | ---      | ---      | ---      | ---      |
| 龙尾（后）(m/s)      | ---      | ---      | ---      | ---      | ---      | ---      |

5.2 问题二模型的建立与求解

5.2.1 板凳矩形化建系

由于在问题一中所封装的函数可以求出任意时刻所有把手的坐标，我们可以将板凳抽象为一个矩形，并由把手坐标求得矩形四端点坐标，如图 8：

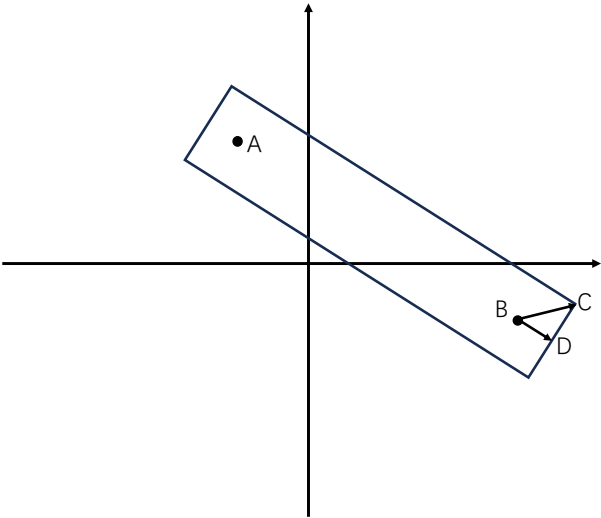


图 8：板凳矩形化示意图

其中 A,B 为两个把手，作 BD 垂直于 DC，具体求坐标过程如下：



**Step1:** 记  $B - A = (a, b)$ , 则  $\overrightarrow{BD} = \frac{27.5}{165}(a, b)$  (或  $\frac{27.5}{286}(a, b)$ )

**Step2:** 由垂直关系可得到  $\overrightarrow{DC} = \frac{15}{165}(b, -a)$  (或  $\frac{15}{286}(b, -a)$ )

**Step3:** 根据  $\overrightarrow{BC} = \overrightarrow{BD} + \overrightarrow{DC}$ , 我们可求出 C 点坐标。同理, 四个角坐标均能求出。

### 5.2.2 碰撞分析

此时, 我们可表示出所有矩形板凳四角坐标。在此基础上, 板凳边与边的碰撞即可视为不相邻矩形在空间中的重合, 即不相邻矩形在空间中存在边的相交, 如图 9, 板凳碰撞即代表  $ge$  边与  $hf$  边相交。

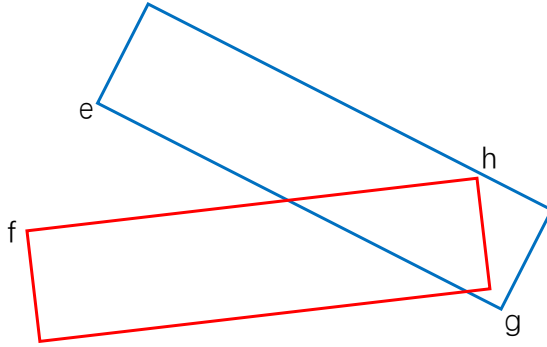


图 9: 板凳碰撞示意图

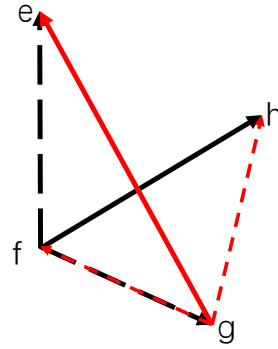


图 10: 两边相交示意图

基于欧几里得空间性质, 我们可以通过外积的方法来判断两边的相交, 如图 10。两边相交当且仅当:

$$\begin{cases} \overrightarrow{fe} \times \overrightarrow{fh} \text{ 与 } \overrightarrow{fg} \times \overrightarrow{fh} \text{ 反向} \\ \overrightarrow{gf} \times \overrightarrow{ge} \text{ 与 } \overrightarrow{gh} \times \overrightarrow{ge} \text{ 反向} \end{cases} \quad (6)$$

### 5.2.3 二分计算碰撞时间

我们利用 C++ 语言编写上述模型, 将不相邻板凳的边第一次相交作为我们的约束条件, 任意取一已碰撞时间作为上界与一未碰撞时间作为下界, 将其区间进行重复二分, 我们认为上下界之差小于  $10^{-12}s$  时的下界即为所要求的碰撞时间。二分时间流程图如图 11 所示:

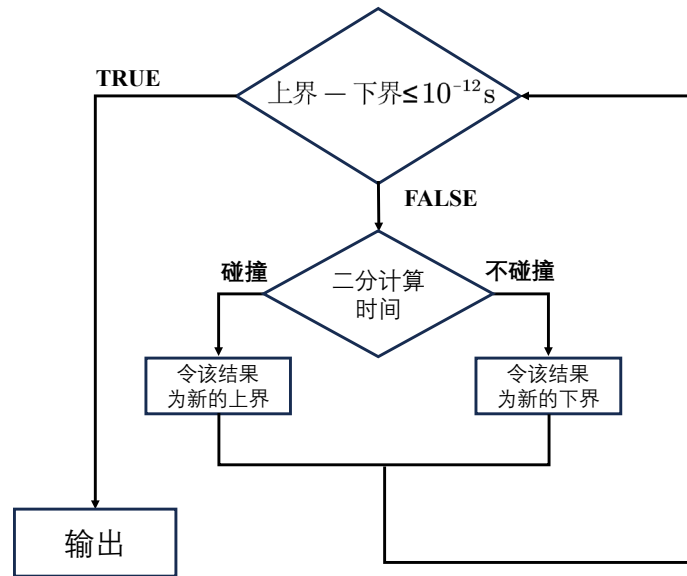


图 11：二分时间流程图

#### 5.2.4 把手位置速度计算

将上述所得时间重输入问题一所封装的函数，即可求得此时舞龙队内所有把手的位置和速度。

#### 5.2.5 问题二求解结果。

我们所求舞龙队盘入的终止时刻为 412.469482 秒。

表 3：问题二位置结果与速度结果

|         | 横坐标x (m)  | 纵坐标y (m)  | 速度 (m/s) |
|---------|-----------|-----------|----------|
| 龙头      | 1.206322  | 1.945223  | 1.000000 |
| 第1节龙身   | -1.647045 | 1.750559  | 0.991554 |
| 第51节龙身  | 1.277145  | 4.327873  | 0.976863 |
| 第101节龙身 | -0.532025 | -5.880585 | 0.974556 |
| 第151节龙身 | 0.973047  | -6.956945 | 0.973614 |
| 第201节龙身 | -7.892553 | -1.234958 | 0.973102 |

### 5.3 问题三模型的建立与求解

#### 5.3.1 碰撞时龙头前把手旋转角确定

问题三要求我们计算龙头前把手能够沿着螺线进入掉头空间边界时的最小盘入螺距。也就是说，在此螺距下，舞龙队内部刚好在进入掉头空间边界时发生内部碰撞，如图 12：

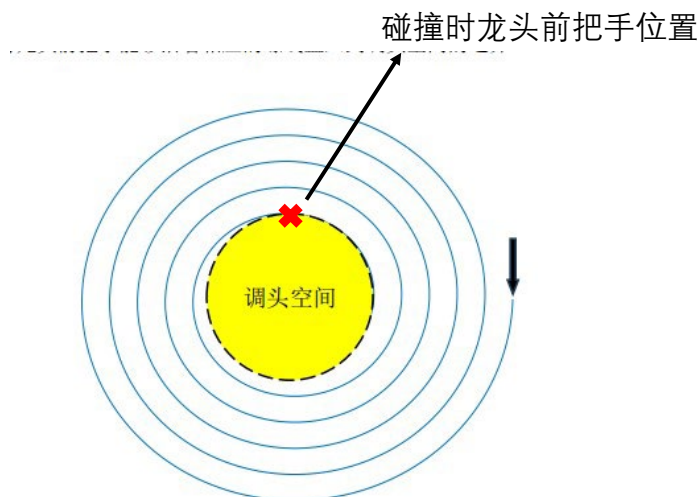


图 12: 碰撞点示意图

通过联立螺旋方程和圆方程，可以得到旋转角与螺距之间满足关系：

$$x^2 + y^2 = \left( \frac{S}{2\pi} \theta_0 \right)^2 = r_0^2 = 450^2 \quad (7)$$

其中  $\theta_0$  为旋转角， $S$  为螺距，求解出此时龙头前把手螺旋角  $\theta_0 = \frac{900\pi}{S}$ 。

### 5.3.2 二分计算最短螺距

龙头前把手螺旋角确定，即代表龙头前把手坐标确定。根据问题一二分法确定后续坐标方法，我们可以遍历求得后续所有龙身、龙尾的坐标，再根据问题二判断碰撞方法，我们可以遍历所有矩形观察是否存在重合情况。并将上述模型用 C++ 编写封装成函数。

我们确定一个较大的  $S$  作为上界，满足龙头前把手到达该旋转角时不发生碰撞，再确定一个较小的  $S$  作为下界，满足龙头前把手到达该旋转角时已发生碰撞。在此区间内重复二分  $S$  的大小，当上下界之差小于  $10^{-12}\text{cm}$  时的下界即为所要求的最小螺距。二分螺距流程图如图 13 所示：

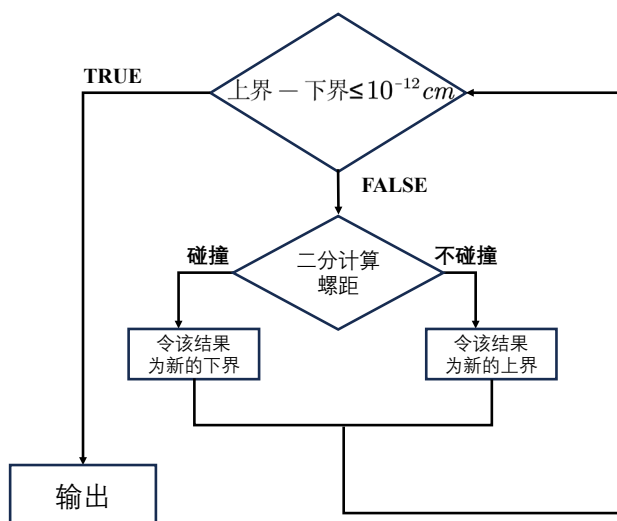


图 13: 二分螺距流程图

5.3.3 问题三求解结果

我们所求最短螺距为 42.021690cm。

5.4 问题四模型的建立与求解

5.4.1 调头点确定整个 S 形曲线轨迹证明

我们抽象龙头在掉头空间中的运动轨迹，如图 14。其中点  $B$  是盘入螺线的终点，点  $F$  是盘出螺线的始点， $BC, DC$  是以点  $C$  为圆心的圆弧半径， $DE, ED$  是以点  $E$  为圆心的圆弧半径。根据题意， $B, F$  点关于原点对称， $\widehat{BD} = 2\widehat{DF}$ 。

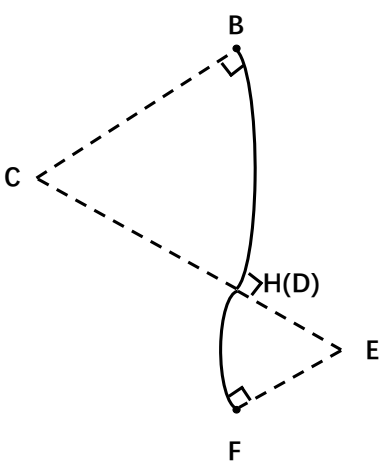


图 14: 掉头空间内运动轨迹

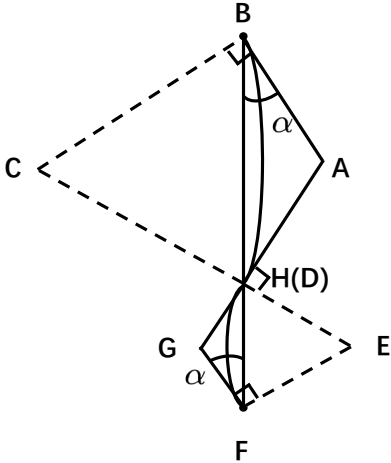


图 15: 证明示意图

取线段  $AG$ ，与  $AF$  交于点  $H$ ，使得满足  $FH:HB = 1:2$ 。

由题意，在图弧  $C_1$ 、 $C_2$  交点  $D$  处  $C_1$ 、 $C_2$  相切

$\Rightarrow$  半径  $ED$  和  $EC$  在同一直线上

要证明共线，即证明  $AG$  与  $EC$  垂直。

由  $B, F$  关于原点对称

$\Rightarrow FG \parallel AB$

我们以  $FB$  为  $y$  轴， $F$  为原点建立新的坐标系，不失一般的，设  $B$  点坐标为  $(0,3)$ ， $AB$ 、 $FG$  方向均为  $(-1,1)$ 。

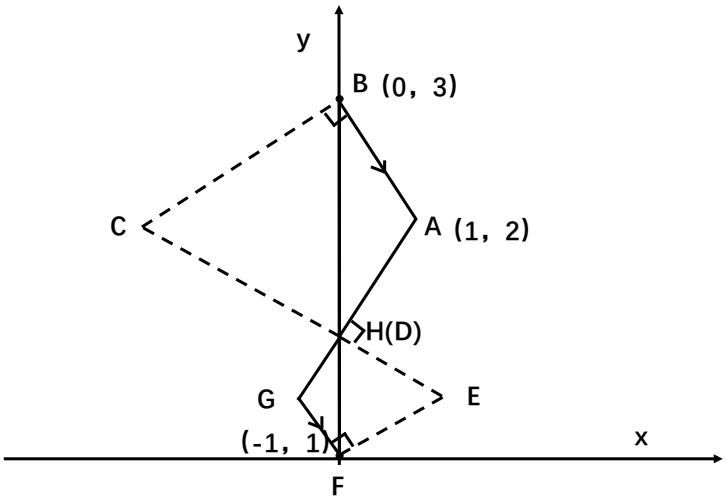


图 16: 建系分析图

通过内积我们得出  $\overrightarrow{AG} \cdot \overrightarrow{EC} = 0$ ，从而证明了  $AG \perp CE$ ，从而证明 S 形曲线的连接点 D 与线段 BF 共线。

这样一来，我们便可以通过由盘入螺线的终点就唯一确定 S 型调头曲线。具体方法如下：

1. 由对称性，通过盘入螺线的终点  $(e, f)$  可确定盘出螺线的始点  $(-e, -f)$
2. 通过这两点可以算出靠近盘出始点一端的  $1/3$  分点  $-\frac{1}{3}(e, f)$
3. 由盘入螺线的终点的切线

$$y = \frac{dy}{dx}|_{(e,f)}x + f - e \frac{dy}{dx}|_{(e,f)}$$

和盘出螺线始点的切线

$$y = \frac{dy}{dx}|_{(-e,-f)}x + e \frac{dy}{dx}|_{(-e,-f)} - f$$

记  $P_1 = \frac{-1}{\frac{dy}{dx}|_{(e,f)}}$ ， $P_2 = \frac{-1}{\frac{dy}{dx}|_{(-e,-f)}}$ ，则通过两切线的垂线方程分别为

$$y = P_1x + f - P_1e$$

$$y = P_2x - f + P_2e$$

可确定两圆弧圆心所在直线，再经过  $1/3$  分点做满足  $\angle GFH = \angle FHG = \angle AHB = \angle HBA = \alpha$  的直线段 CE：

$$y = \frac{f/e + \tan(\alpha)}{1 - (f/e)\tan(\alpha)}(x + \frac{1}{3}e) - \frac{1}{3}f$$

且此时两圆弧的圆心角均为  $2\alpha$ ，CE 与两切线垂线的交点即为圆心，不妨记圆心就是 C、E，则大小圆半径分别等于 CB、FE。

由待定圆方程的系数知道，有了圆心，圆周角和半径和圆周的起点和终点 S 型曲线的坐标就唯一确定。

#### 5.4.2 调头曲线长影响因素分析

如图 13，当 BF 长度恒定时，圆弧切线与 BF 所成角  $\alpha$  越小，弧越贴近直线 BF，即弧长越短；反之当  $\alpha$  恒定时，BF 长度越短，弧长也越短。接下来我们分析  $\alpha$  角与  $\theta$  的函数关系，以便确定  $\theta$  与 S 型调头曲线的弧长关系。

不失一般的，我们简化盘入螺线方程为

$$\begin{cases} x = \theta \cos \theta \\ y = \theta \sin \theta \end{cases} \quad (8)$$

那么，盘入螺线的导数为

$$\frac{dy}{dx} = \frac{\sin \theta + \theta \cos \theta}{\cos \theta - \theta \sin \theta} \quad (9)$$

设掉头点 B 坐标  $(\theta \cos \theta, \theta \sin \theta)$ ，其切线方向  $(\cos \theta - \theta \sin \theta, \sin \theta + \theta \cos \theta)$

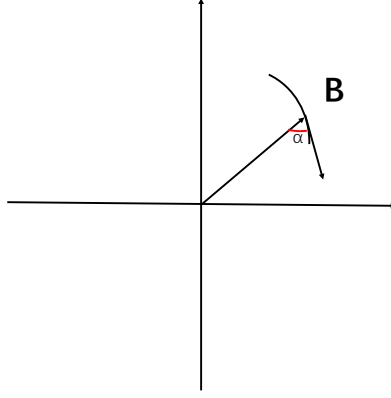


图 17: 掉头曲线长度分析示意图

由以下外积计算公式:

$$\frac{1}{\sqrt{(1+\theta^2)\theta^2}} \begin{vmatrix} c\theta \cos \theta & \cos \theta - \theta \sin \theta \\ \theta \sin \theta & \sin \theta + \theta \cos \theta \end{vmatrix} = \sin \alpha \quad (10)$$

可求得

$$\alpha = \arcsin \frac{\theta}{\sqrt{1+\theta^2}}, \alpha \in \left(0, \frac{\pi}{2}\right) \quad (11)$$

根据题意, 此时螺距为 1.7 米, 根据公式 (7) 可解得此时旋转角  $\theta < 17\text{rad}$ , 利用 matlab 画出  $f(\theta) = \frac{\theta}{\sqrt{1+\theta^2}}$  的图像, 如图 18。

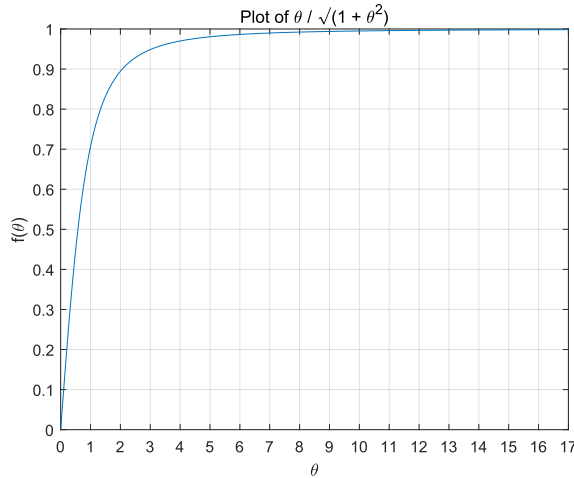


图 18: 函数图像

根据图像可知, 此函数关于  $\theta$  递增, 又由于  $\arcsin \alpha$  在  $\alpha \in (0, \pi/2)$  也单调递增, 可知此时  $\alpha$  关于  $\theta$  单调递增, 由此, 要让  $\alpha$  小只需让旋转角  $\theta$  变小即可, 也就是说在不发生碰撞的前提下, 龙头沿螺线前进越多, 调头开始的位置越靠里,  $\alpha$  角就小, BF 长度越短, 此时掉头曲线就越短。

#### 5.4.3 双重二分确定 S 形曲线

基于上述证明与分析, 我们只需要找到龙头沿螺线前进最远的掉头点, 即可确定整个 S 形曲线。

题目给定螺距为 1.7m, 根据问题三, 我们可以知道此时龙头把手可顺利进入掉头区域。对于此情形下掉头点的确定, 我们仍采用旋转角二分法, 任取一个较小的旋转

角 $\theta_1$ 作为下界,满足该条件下舞龙队内部会发生碰撞,再任取一个较大的旋转角 $\theta_2$ 作为上界,满足龙头前把手进入调头区但不发生碰撞,在该区间内重复进行二分操作。

每二分出一个旋转角 $\theta$ ,都可以唯一确定一个调头点,从而唯一确定一个 S 形曲线。但舞龙队沿 S 形曲线行进过程中,新的路径 F (即盘入螺线、S 型调头曲线和盘出曲线组成的曲线)上各把手的位置无法通过旋转角确定。为此,我们提出采用数轴坐标二分法。

我们将开始调头的位置视为原点,将整条路径 F 视为一维数轴、数轴的正方向为龙把手前进方向, F 上一点到调头点的弧线长度作为数轴上的坐标的绝对值,则 F 上的坐标与该数轴坐标有一一对应关系,我们用该坐标进行二分即可。例如求龙身第一把手,我们取龙头后把手 $l_1$ 后一定距离的两点 $a, b$ ,满足在数轴上 $|l_1 - b| > |l_1 - a|$ 时,有割线 $l_1 b > 286\text{cm}$ ,  $l_1 a < 286\text{cm}$ , 在 $a, b$ 两点内不断二分坐标找到点  $c$ , 满足 $l_1 c = 286\text{cm}$  (或上下界之差 $\leq 10^{-12}\text{cm}$ ), 此时 $c$ 点恰为龙身第一把手 $l_2$ 。龙身及龙尾剩余把手同理。

但我们需要强调,沿 S 形曲线行进过程中,舞龙队不仅要考虑是否碰撞,还需要考虑是否合理的问题。是否合理包括以下两种情况:

**情况 1:** 在龙头前把手 ( $l_1$ ) 位置确定后,第一节龙身前把手 ( $l_2$ ) 可能与 $l_1$ 不在同一圈上,如图中 V2,

**情况 2:**  $l_1$  确定后,通过坐标二分确定 $l_2$ ,  $l_2$ 跟随 $l_1$ 向正向运动,在一定时间后, $l_2$ 新位置在数轴上会发生后退,如图中 V3。

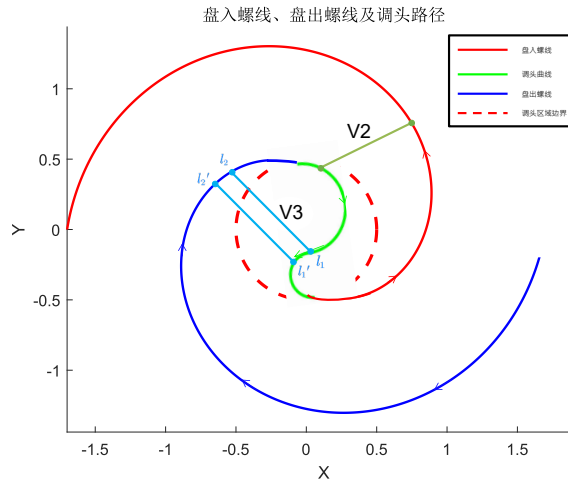


图 19: 合理性情况分析

**情况 1 的处理:** 例如龙身第一把手 $l_2$ 的确定,我们取上界是从 $l_1$ 往图示箭头反方向走弧长 $286\text{cm}$ (即 $l_1$ 与 $l_2$ 的距离)的点 $b_1$ 确定的坐标,此时割线 $l_1 b_1$ 的长度小于 $286\text{cm}$ ;从 $b_1$ 开始往后依次走弧长 $10\text{cm}$ ,直至出现第一个点 $b_2$ 使得割线长度大于 $286\text{cm}$ ,取 $b_2$ 的坐标作为下界。

**情况 2 的处理:** 在每一个旋转角 $\theta$ 确定的 S 形曲线上,龙头前把手以步长为 $5\text{cm}$ 进行搜索式生成,记录每一次每一个后续把手的坐标,当出现某一把手坐标随龙头前把手前进而减少时,我们认为发生了情况 2,将此旋转角 $\theta$ 拒绝,将其作为新的下界,重新二分生成 $\theta$ 。

我们每生成一次后续把手都需要根据问题二进行一次碰撞分析,当发生碰撞,也拒绝此时的旋转角 $\theta$ ,将其作为新的下界,重新二分生成 $\theta$ 。直到上下界之差 $\leq 10^{-12}\text{rad}$ ,

我们将此时所得 $\theta$ 作为确定的调头点旋转角，基于此，我们可得出最短的调头曲线。  
双重二分流程图如图 20 所示：

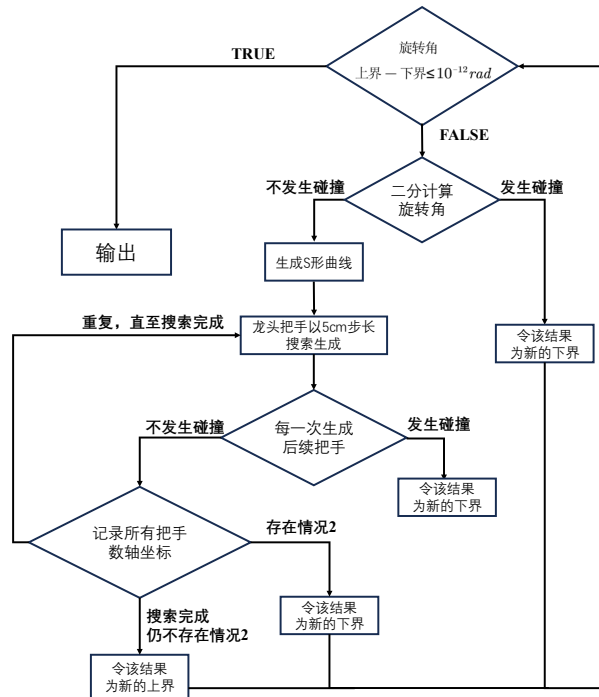


图 20：双重二分流程图

其中生成后续把手所采用的二分数轴坐标流程图如图 21 所示：

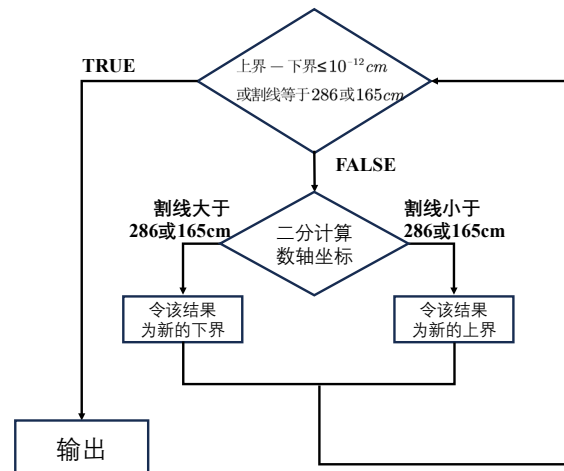


图 21：二分数轴坐标流程图

#### 5.4.4 把手位置速度计算

基于上述所求的调头曲线方程以及给定螺距的螺线方程，我们可以求出任意龙头前把手位置坐标。根据上述图 21 中的数轴坐标二分法，可以得出后续所有把手的位置坐标。在极短的一段时间微元内（这里我们取 $10^{-3}s$ ），我们可以利用问题一封装的瞬时速度计算函数，求得对应把手的速度。

#### 5.4.5 问题四求解结果

我们所求最小调头旋转角： $15.696400rad$ ，此时仍保持各部分相切且调头曲线最短。



表 4：问题四位置结果

|                 | -100 s     | -50 s      | 0 s        | 50 s       | 100 s      |
|-----------------|------------|------------|------------|------------|------------|
| 龙头 x (m)        | 8.486402   | 6.365258   | -4.246587  | 4.169300   | 0.083732   |
| 龙头 y (m)        | -0.280868  | -2.131256  | 0.049107   | 4.554176   | 8.071759   |
| 第 1 节龙身 x (m)   | 8.190193   | 6.792016   | -3.500783  | 5.664380   | 2.873042   |
| 第 1 节龙身 y (m)   | 2.563751   | 0.696725   | -2.711939  | 2.116075   | 7.439817   |
| 第 51 节龙身 x (m)  | -8.753862  | -6.923957  | -1.573765  | -4.885600  | 4.085884   |
| 第 51 节龙身 y (m)  | 6.457674   | -6.585645  | -7.865677  | -3.694413  | 1.474161   |
| 第 101 节龙身 x (m) | -12.741574 | 7.409438   | 6.544853   | -4.561124  | -5.074410  |
| 第 101 节龙身 y (m) | -0.807224  | -9.004766  | 8.135982   | 7.836943   | 5.425168   |
| 第 151 节龙身 x (m) | -14.256722 | 11.175506  | -3.283741  | -7.894499  | 7.222574   |
| 第 151 节龙身 y (m) | 2.099241   | -7.467161  | 11.954739  | -8.035849  | -6.906905  |
| 第 201 节龙身 x (m) | -8.833450  | 7.199343   | -3.051098  | -3.665848  | 10.809703  |
| 第 201 节龙身 y (m) | 13.201858  | -13.171176 | 13.749624  | -12.574806 | 5.262387   |
| 龙尾（后）x (m)      | -5.012653  | 4.216902   | -5.828207  | 9.199458   | -12.458551 |
| 龙尾（后）y (m)      | -15.711170 | 15.072408  | -13.566265 | 10.324021  | -2.996601  |

表 5：问题四速度结果

|                 | -100 s   | -50 s    | 0 s      | 50 s     | 100 s    |
|-----------------|----------|----------|----------|----------|----------|
| 龙头 (m/s)        | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| 第 1 节龙身 (m/s)   | 0.999898 | 0.999738 | 0.998338 | 1.000399 | 1.000131 |
| 第 51 节龙身 (m/s)  | 0.999312 | 0.998535 | 0.994244 | 1.090082 | 1.004465 |
| 第 101 节龙身 (m/s) | 0.999048 | 0.998125 | 0.993520 | 1.088269 | 0.982709 |
| 第 151 节龙身 (m/s) | 0.998898 | 0.997919 | 0.993218 | 1.087737 | 0.981797 |
| 第 201 节龙身 (m/s) | 0.998801 | 0.997795 | 0.993053 | 1.087483 | 0.981451 |
| 龙尾（后）(m/s)      | 0.998768 | 0.997754 | 0.993001 | 1.087409 | 0.981358 |

## 5.5 问题五模型的建立与求解

### 5.5.1 行进速度分析

设龙头前把手速度为 $w$  cm/s，则在 $t$ 时间内龙头走过的路程 $x$ 满足：

$$x = w \times t \quad (12)$$

由于本题中舞龙队沿问题四所设定的路径前进，那么在问题四所建立的数轴上，我们可以根据龙头走过的路程，确定龙头前把手坐标，并依据上述图 21 中的数轴坐标二分法，得出后续所有把手位置坐标。通过取极小一段时间微元（这里我们取 $10^{-3}s$ ），利用该时间间隔内的平均速度来替代瞬时速度，即可求出任意时间每点的速度。

### 5.5.2 二分速度计算

根据问题四行进轨迹，我们设定龙头前把手以 5cm 为步长搜索遍历整条运动曲线，并生成后续所有把手。我们取一个较大的前把手速度 $v_1$ 作为上界，满足在上述搜索过程中存在把手的速度超过 $2m/s$ ，再取一个较小的前把手速度 $v_2$ 作为下界，满足在上述搜索过程中所有把手的速度不超过 $2m/s$ ，在此区间内不断二分，若新速度下存在把手的速度超过 $2m/s$ ，则将其设为上界，新速度下不存在把手的速度超过 $2m/s$ ，则将其设为下界。直至区间上界减下界的值小于等于 $10^{-12}cm/s$ ，取下界作为最后最大龙头速度。

二分速度流程图如图所示：

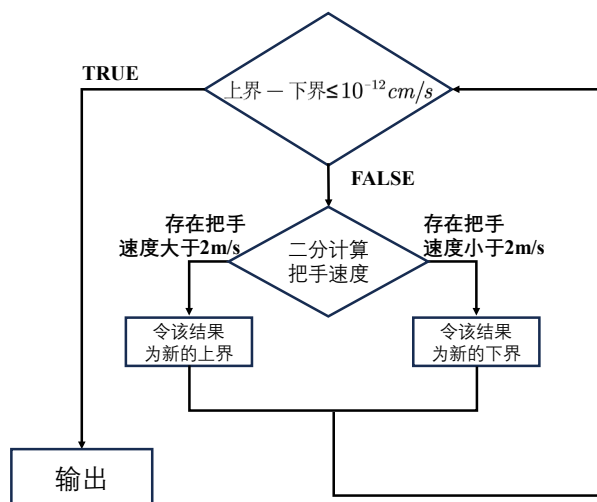


图 22：二分速度流程图

### 5.5.3 问题五求解结果

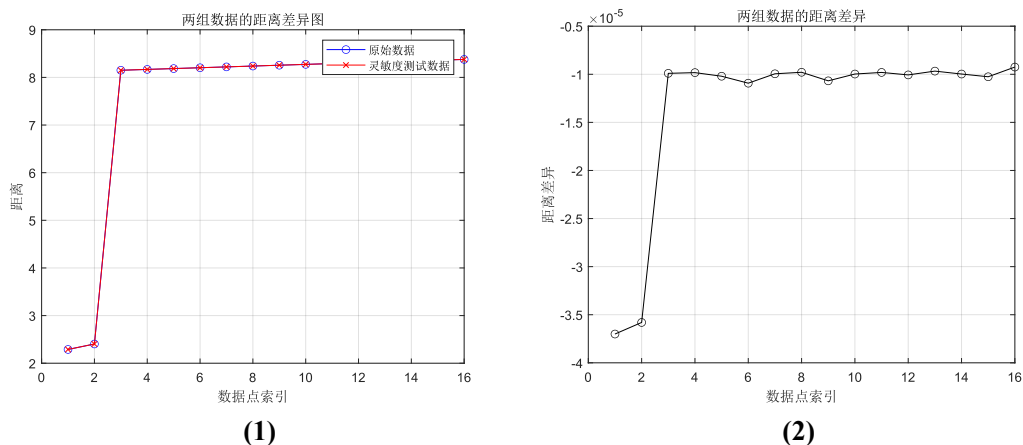
我们所求最大龙头行进速度为  $109.515381\text{cm/s}$ 。

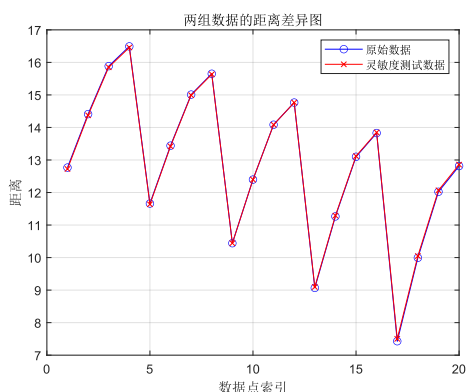
## 六、模型的灵敏度分析

根据二分法求得的点坐标作为模型重要结果，后者的搜索建立在前者的生成之上，在理论上存在灵敏度的累计式增长，所以点坐标数据具有极强的灵敏度分析价值。在本模型中，龙头前把手速度和螺线螺距是点坐标生成的重要影响因子。因此我们分别选取不同问题中的速度和螺距，来对点坐标生成模型进行灵敏度分析检测。

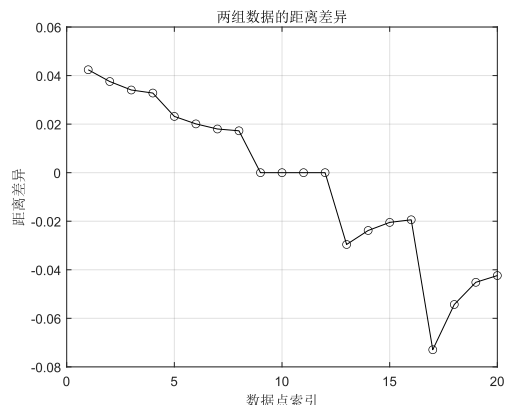
### 6.1 龙头前把手速度

问题二和问题四给定龙头前把手速度为  $1\text{m/s}$ ，但在实际中很难维持恒定速度运动。因此我们仅修改程序中的速度参数，保留螺距等其他因素不变，取速度为  $0.98\text{m/s}$ 。将两次所得坐标结果导入 matlab，随机抽取对应点计算离原点距离，并作差异分析，结果如图：





(3)



(4)

图 23: 龙头前把手速度灵敏度分析图

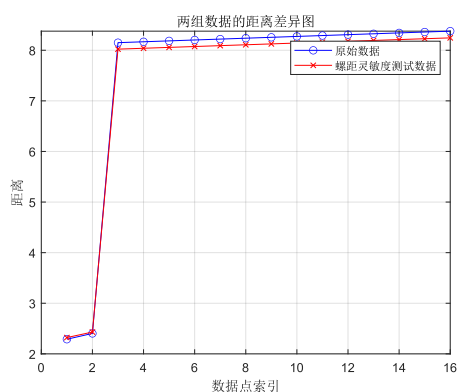
其中图(1)(2)为问题二两组数据距离差异示意图与数值图，图(3)(4)为问题四两组数据距离差异示意图与数值图。我们可以得出问题二中原始数据的平均距离为 7.5245m，测试数据的平均距离为 7.5245m，结果保持一致。问题四中原始数据的平均距离为 12.826m，测试数据的平均距离：12.8303m，将其按照下述公式计算灵敏度参照值：

$$\text{参照值} = \frac{\text{测试数据平均距离}/(\text{原始数据平均距离}+\text{测试数据平均距离})}{\text{新速度}/(\text{原速度}+\text{新速度})} \quad (13)$$

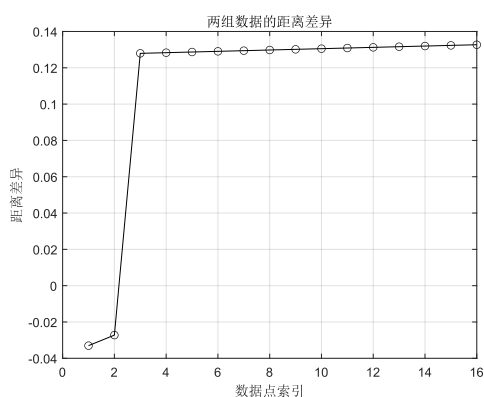
带入数据，求得参照值结果为 1.0104，与 1 偏差 1.04%，说明该模型对速度灵敏度低，对数据噪声低敏，稳定性、鲁棒性强。

## 6.2 螺距

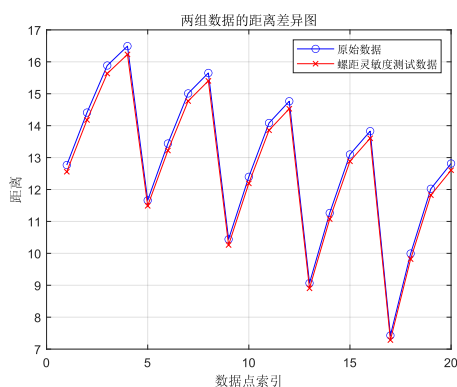
问题二和问题四给出了固定的螺距，而实际情况下，板凳龙也很难沿着等距螺线行进，我们对原 55cm 的螺距取新螺距为 53cm；对原 170cm 的螺距取新螺距为 165cm，将所得位置坐标导入 matlab，随机抽取对应点计算离原点距离，并作差异分析，结果如图：



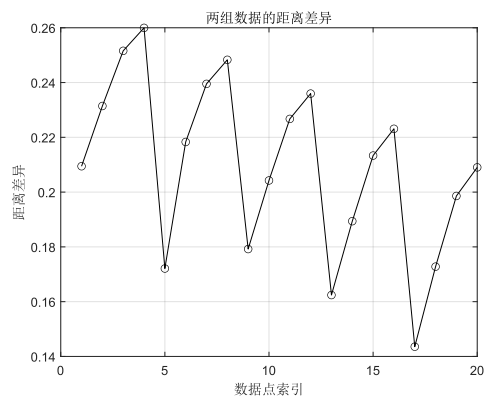
(1)



(2)



(3)



(4)

图 24: 螺距灵敏度分析图

其中图(1)(2)为问题二两组数据距离差异示意图与数值图，图(3)(4)为问题四两组数据距离差异示意图与数值图。我们可以得出问题二中原始数据的平均距离为 7.5245m，测试数据的平均距离为 7.4142m；问题四中原始数据的平均距离为 12.826m，测试数据的平均距离：12.6167m。我们根据公式（13）计算二者的灵敏度参照值。

对于原 55cm 的情况，我们计算得参照值为 1.011，与 1 偏差 1.1%；对于原 170cm 的情况，我们计算得参照值为 1.0068，与 1 偏差 0.68%。由此我们看出该模型对螺距灵敏度不高，稳定性高，适用性好。

## 七、模型的评价、改进与推广

### 7.1 模型的优点

1. 五问模型均采用二分法求解，过程易于解释与理解，模型原创，创新性强。
2. 代码结构清晰，相关函数封装性好，便于功能调用与功能扩展。
3. 二分模型所求解均为计算机算力支持下的最精确解，结果准确性高。
4. 经灵敏度检测，本模型对数据噪声低敏，鲁棒性、稳定性强

### 7.2 模型的缺点

二分算法时间复杂度高，需要较高的算力资源。

### 7.3 模型的改进

可以进一步优化二分查找策略与处理模式，如优化取值空间，与其他取值算法结合，或采用并行处理或位处理模式，来进一步提高求解的精度和效率。

### 7.4 模型的推广

本模型广泛适用于求解一维最优解问题；模型在处理板凳龙螺形运动轨迹时所采用的二分方法亦可用于蛇形机器人、多关节机械臂、粒子运动等其他类链条运动的动力学模型动态建模。

## 八、参考文献



无

## 附录



### 附录 1

介绍：支撑材料的文件列表




支撑材料文件夹中有以下 4 个子文件夹。

|  |                |     |
|--|----------------|-----|
|  程序   | 2024/9/8 17:51 | 文件夹 |
|  答案   | 2024/9/8 16:25 | 文件夹 |
|  图片   | 2024/9/8 16:37 | 文件夹 |
|  运行数据 | 2024/9/8 16:29 | 文件夹 |







在\支撑材料\程序 中有一个可以直接运行出所有答案的封装好的程序 **Main.cpp**，另外是 1 至 4 题分开写的代码。

|  |                |         |       |
|--|----------------|---------|-------|
|  1至4题的分程序 | 2024/9/6 17:47 | 文件夹     |       |
|  Main.cpp | 2024/9/8 17:50 | C++ 源文件 | 19 KB |










在\支撑材料\答案 中有题目要求的 3 个答案表格。

|   |                |                      |          |
|---|----------------|----------------------|----------|
|  result1.xlsx  | 2024/9/8 16:24 | Microsoft Excel 工... | 1,270 KB |
|  result2.xlsx  | 2024/9/7 11:37 | Microsoft Excel 工... | 23 KB    |
|  result4 .xlsx | 2024/9/7 11:39 | Microsoft Excel 工... | 1,474 KB |

在\支撑材料\图片 中有本题用到的部分图片，和绘制所用的.pptx 及.vsdx 文件。

|  |                |                        |        |
|--|----------------|------------------------|--------|
|  16.svg             | 2024/9/8 15:35 | Microsoft Edge HT...   | 144 KB |
|  untitled.svg       | 2024/9/8 14:44 | Microsoft Edge HT...   | 97 KB  |
|  绘图1.svg            | 2024/9/8 15:27 | Microsoft Edge HT...   | 36 KB  |
|  绘图1.vsdx           | 2024/9/8 16:34 | Microsoft Visio Dra... | 38 KB  |
|  数学建模9.8 16.34.pptx | 2024/9/8 16:34 | Microsoft PowerPoi...  | 841 KB |
|  图片1.svg            | 2024/9/8 15:16 | Microsoft Edge HT...   | 137 KB |

在\支撑材料\运行数据 中有程序运行得到的数据文件。

|   |                |    |          |
|---|----------------|----|----------|
|  dot_place_q1  | 2024/9/6 19:10 | 文件 | 988 KB   |
|  dot_place_q2  | 2024/9/6 19:10 | 文件 | 6 KB     |
|  dot_place_q4  | 2024/9/6 19:11 | 文件 | 1,069 KB |
|  dot_place2_q1 | 2024/9/6 19:10 | 文件 | 1,054 KB |
|  dot_place2_q2 | 2024/9/6 19:10 | 文件 | 6 KB     |
|  dot_place2_q4 | 2024/9/6 19:11 | 文件 | 1,113 KB |
|  velocity_q1   | 2024/9/6 19:10 | 文件 | 489 KB   |
|  velocity_q2   | 2024/9/6 19:10 | 文件 | 3 KB     |
|  velocity_q4   | 2024/9/6 19:12 | 文件 | 494 KB   |

### 附录 2

介绍：该代码由 C++编写，程序内封装了本体内所有解题所用函数，并可以调试得到所有问题的解。

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

```

2. #define IOS std::ios::sync_with_stdio(false);std::cin.tie(nullptr);std::cout.tie(nullptr);
3. #define all(x) (x).begin(),(x).end()
4. #define Yes(x,y) cout<<((x)?"Yes":"No")<<y
5. #define YES(x,y) cout<<((x)?"YES":"NO")<<y
6. #define ls ((u)<<1)
7. #define rs ((u)<<1|1)
8. #define mid (((l)+(r))>>1)
9. #define lowbit(x) ((x)&(-(x)))
10.#define itn int
11.#define asn ans
12.#define reisz resize
13.#define pdd pair<double,double>
14.#define pll pair<LL,LL>
15.#define pii pair<int,int>
16.#define tll tuple<LL,LL,LL>
17.#define tii tuple<int,int,int>
18.#define pll1 pair<LLL,LLL>
19.#define ULL unsigned long long
20.#define LL long long
21.#define LLL __int128
22.#define ld long double
23.#define ui64 uint64_t
24.#define ui32 uint32_t
25.using namespace std;
26 ofstream now_file;
27.streambuf* console_buf;
28.string now_bind;
29.void write_to_file(const string& file_name){
30.    if(file_name==now_bind)return;
31.    if(!now_bind.empty()){
32.        now_file.close();
33.    }else{
34.        console_buf=cout.rdbuf();
35.    }
36.    now_bind=file_name;
37.    now_file=ofstream(file_name);
38.    cout.rdbuf(now_file.rdbuf());
39.}
40.void write_to_console(){
41.    if(now_bind.empty())return;
42.    now_bind.clear();
43.    now_file.close();
44.    cout.rdbuf(console_buf);

```

```

45.}
46.template<typename T>
47.T fang(const T& a){
48.    return a*a;
49.}
50.template<typename T,typename Q>
51.bool chmax(T& u1,T& u2,const Q& v){
52.    if(v>u1) { u2 = u1, u1 = v;return true;}
53.    if(v>u2){u2=v;return true;}
54.    return false;
55.}
56.template<typename T,typename Q>
57.bool chmin(T& u1,T& u2,const Q& v){
58.    if(v<u1) { u2 = u1, u1 = v;return true;}
59.    if(v<u2){u2=v;return true;}
60.    return false;
61.}
62.template<typename T,typename Q>
63.bool chmin(T& a,const Q& b){
64.    return a > b && (a = b, true);
65.}
66.template<typename T,typename Q>
67.bool chmax(T& a,const Q& b){
68.    return a<b&&(a=b,true);
69.}
70.template<typename t1,typename t2>
71.istream& operator>>(istream& in,pair<t1,t2>& pa){
72.    in>>pa.first>>pa.second;
73.    return in;
74.}
75.template<typename t1,typename t2>
76.ostream& operator<<(ostream& out,const pair<t1,t2>& pa){
77.    out<<pa.first<< ' '<<pa.second;
78.    return out;
79.}
80.template<typename T>
81.istream& operator>>(istream& in,vector<T>& arr){
82.    for(auto& v:arr)in>>v;
83.    return in;
84.}
85.template<typename T>
86.ostream& operator<<(ostream& out,const vector<T>& arr){
87.    for(auto& v:arr)out<<v<< ' ';
88.    return out;

```

```

89.}
90.//以上为常见函数重载,用于加快编程速度
91.const int N=6e5+5;
92.const int SIZ=1e7;
93.const LL inf=1e17;
94.const ld PI= acos(-1);//圆周率
95.const ld eps=1e-12;
96.//ld arcsinh(ld x){return log(x+sqrt(x*x+1));}
97.
98.int sig(ld x){      //判断浮点数的符号(减少精度误差)
99.    if(abs(x)<eps)return 0;
100.    return x<0?-1:1;
101.}
102. ld K=55.0/(2*PI);
103. //角度->路程:1/2(sqrt(x^2+1)x+sinh^-1(x))
104. ld per_dis(ld x){
105.    return 0.5*(sqrt(x*x+1)*x+ asinh(x));
106.}
107. //上一个函数的反函数:路程->角度
108. ld inv_per_dis(ld d){
109.    ld l=0,r=64*PI;
110.    while(r-l>eps){
111.        ld m=(l+r)/2;
112.        if(per_dis(m)>d)r=m;
113.        else l=m;
114.    }
115.    return l;
116.}
117. //16圈总长:44259cm
118. ld velocity=100;
119. ld rA=880;
120. vector<ld> dis_handle;
121. struct Point{ //点(向量)类
122.    ld x,y;
123.    Point(){x=0,y=0;};
124.    Point(ld x,ld y):x(x),y(y){};
125.    Point operator*(ld effc) const{ //向量的数乘
126.        Point res;
127.        res.x=x*effc;
128.        res.y=y*effc;
129.        return res;
130.    }
131.    friend ostream& operator<<(ostream& out,const Point& dot
    ){ //输出点的坐标

```



```

132.         out<<dot.x<<' '<<dot.y;
133.         return out;
134.     };
135. };
136. ld Euler_distance(const Point& a,const Point& b){ //计算两点
    之间欧拉距离
137.     return sqrt(fang(a.x-b.x)+fang(a.y-b.y));
138. }
139. Point operator+(const Point& a,const Point& b){ //向量的加
    法
140.     Point res;
141.     res.x=a.x+b.x;
142.     res.y=a.y+b.y;
143.     return res;
144. }
145. Point operator-(const Point& a,const Point& b){ //向量的减
    法
146.     Point res;
147.     res.x=a.x-b.x;
148.     res.y=a.y-b.y;
149.     return res;
150. }
151. Point Rotate(const Point& a,ld angle){ //将向量a 顺时针旋转
    一个角度angle
152.     angle=2*PI-angle;
153.     ld cs=cos(angle),si=sin(angle);
154.     return {a.x*cs-a.y*si,a.x*si+a.y*cs};
155. }
156. ld operator*(const Point& a,const Point& b){ //向量的内积
157.     return a.x*b.x+a.y*b.y;
158. }
159. ld operator^(const Point& a,const Point& b){ //向量的叉积
160.     return a.x*b.y-a.y*b.x;
161. }
162. struct Line{ //线段类
163.     Point a,b;//两点确定一条线段
164.     Line(){};
165.     Line(const Point& a,const Point& b):a(a),b(b){};
166. };
167. bool dot_at_all_line(const Point& d,const Line& l){ //判断
    点是否在线段所在的直线上
168.     return sig((d-l.a)^(d-l.b))==0;
169. }

```

```

170. bool dot_at_line(const Point& d,const Line& l){ //判断点是
    否在线段上
171.     if(sig((d-l.a)^(d-l.b)))return false;
172.     ld mn=min(l.a.x,l.b.x);
173.     ld ma=max(l.a.x,l.b.x);
174.     if(d.x<mn||d.x>ma)return false;
175.     mn=min(l.a.y,l.b.y);
176.     ma=max(l.a.y,l.b.y);
177.     return d.y<=ma&& d.y>=mn;
178. }
179. bool intersection_line(const Line& l1,const Line& l2){ //判
    断两个线段是否有交点
180.     if(dot_at_line(l1.a,l2)|| dot_at_line(l1.b,l2)|| dot_at_
        line(l2.a,l1)|| dot_at_line(l2.b,l1))return true;
181.     if(dot_at_all_line(l1.a,l2)|| dot_at_all_line(l1.b,l2)||
        dot_at_all_line(l2.a,l1)|| dot_at_all_line(l2.b,l1))return fa
        lse;
182.     ld t1=(l1.a-l1.b)^(l1.a-l2.a);
183.     ld t2=(l1.a-l1.b)^(l1.a-l2.b);
184.     if(sig(t1)*sig(t2)>0)return false;
185.     t1=(l2.a-l2.b)^(l2.a-l1.a);
186.     t2=(l2.a-l2.b)^(l2.a-l1.b);
187.     if(sig(t1)*sig(t2)>0)return false;
188.     return true;
189. }
190. struct Rectangle{ //矩形类
191.     Point dot[4]; //四点确定一个矩形
192.     Rectangle(){};
193.     Rectangle(const Point& a,const Point& b,const Point& c,c
        onst Point& d){
194.         dot[0]=a;
195.         dot[1]=b;
196.         dot[2]=c;
197.         dot[3]=d;
198.     }
199. };
200. vector<Line> get_line_of_Rectangle(const Rectangle& a){ //
    获得给定矩形的四条边
201.     vector<Line> res;
202.     res.reserve(4);
203.     for(int i=0;i<4;i++){
204.         res.emplace_back(a.dot[i],a.dot[(i+1)%4]);
205.     }
206.     return res;

```

```

207. }
208. bool intersection_Rectangle(const Rectangle& m1,const Rectangle& m2){ //判断两个矩形是否有交,对于等宽矩形,判断其边是否有交即可
209.     for(auto& l1: get_line_of_Rectangle(m1)){
210.         for(auto& l2: get_line_of_Rectangle(m2)){
211.             if(intersection_line(l1,l2)) {
212.                 return true;
213.             }
214.         }
215.     }
216.     return false;
217. }
218. const ld width=15;
219. const ld height=27.5;
220. Point zig_vector(const Point& a,ld len){ //将某个向量压缩(拉伸)至某个长度,保持方向不变
221.     ld old_len= Euler_distance(a,{0,0});
222.     return a*(len/old_len);
223. }
224. Rectangle dot_to_rec(const Point& a,const Point& b){ //给定两个把手坐标,求解板凳矩形
225.     ld md= Euler_distance(a,b);
226.     Point a_to_b=b-a;
227.     Point d1=a_to_b*(height/md);
228.     Point d2= zig_vector({d1.y,-d1.x},width);
229.     Rectangle res;
230.     res.dot[0]=(b+d1)+d2;
231.     res.dot[1]=(b+d1)-d2;
232.     res.dot[2]=(a-d1)-d2;
233.     res.dot[3]=(a-d1)+d2;
234.     return res;
235. }
236. vector<Rectangle> all_Rectangle_gene_from_dots(const vector<Point>& dots){ //给定所有把手位置,求解所有矩形
237.     vector<Rectangle> res;
238.     res.reserve(dots.size()-1);
239.     for(int i=0;i<dots.size()-1;i++){
240.         res.emplace_back(dot_to_rec(dots[i],dots[i+1]));
241.     }
242.     return res;
243. }
244. bool all_rectangle_intersection(const vector<Rectangle>& Recs){ //判断数组中是否有不相邻且相交的矩形(判断板凳碰撞)
245.     for(int i=0;i<Recs.size()-2;i++){

```

```

246.         for(int j=i+2;j<Recs.size();j++){
247.             if(intersection_Rectangle(Recs[i],Recs[j]))return
n true;
248.         }
249.     }
250.     return false;
251. }
252. Point angle_coordinate(ld o){    //螺旋线上,角度->坐标
253.     Point res;
254.     res.x=o*cos(o);
255.     res.y=o*sin(o);
256.     return res;
257. }
258. ld out_dis(ld o,ld x){    //螺旋线上,角度确定原先点位置,架上一条
    长度为x的线段后,确定新点的角度参数
259.     ld last_dis= per_dis(o);
260.     ld l=last_dis+x;
261.     ld r=last_dis+2*x;
262.     l= inv_per_dis(l),r= inv_per_dis(r);
263.     while(r-l>eps){
264.         ld m=(l+r)/2;
265.         if(Euler_distance(angle_coordinate(m), angle_coordin
    ate(o))>x)r=m;
266.         else l=m;
267.     }
268.     return l;
269. }
270. //224 个把手,223 个间隔
271. const ld dealt=1e-4;
272. vector<Point> calc_coor_at_time(ld t){//计算t时刻所有224个
    把手的坐标,下标从0开始
273.     vector<Point> res;
274.     ld tot_len=K*per_dis(16*2*PI);
275.     ld now_len=tot_len- t * velocity;
276.     ld angle= inv_per_dis(now_len/K);
277.     Point now_head= angle_coordinate(angle);
278.     now_head=now_head*K;
279.     //cout<<now_head<<'\\n';//计算龙头的坐标
280.     res.emplace_back(now_head);
281.     ld last_angel=angle;
282.     for(int j=2;j<=224;j++){
283.         ld now_angel= out_dis(last_angel,dis_handle[j-1]/K);
284.         Point now_pt= angle_coordinate(now_angel)*K;
285.         //cout<<now_pt<<'\\n';

```

```

286.         res.emplace_back(now_pt);
287.         last_angel=now_angel;
288.     }
289.     return res;
290. }
291. vector<Point> calc_coor_at_diff_k(ld d){ //Q3, 根据不同螺距
    确定龙头刚好转入调头区域时的 224 个把手位置
292.     ld k=d/(2*PI);
293.     ld angle=450/k;
294.     vector<Point> res;
295.     ld last_angel=angle;
296.     Point head= angle_coordinate(angle)*k;
297.     res.emplace_back(head);
298.     for(int j=2;j<=224;j++){
299.         ld now_angel= out_dis(last_angel,dis_handle[j-1]/k);
300.         Point now_pt= angle_coordinate(now_angel)*k;
301.         res.emplace_back(now_pt);
302.         last_angel=now_angel;
303.     }
304.     return res;
305. }
306. vector<ld> calc_velocity_at_time(ld t){ //计算 t 时刻每个把手
    的瞬时速度(标量), 下标从 0 开始
307.     vector<Point> la= calc_coor_at_time(t);
308.     vector<Point> ne= calc_coor_at_time(t+dealt);
309.     vector<ld> res(la.size());
310.     for(int i=0;i<la.size();i++){
311.         res[i]=Euler_distance(la[i],ne[i])/ dealt;
312.     }
313.     return res;
314. }
315. bool is_legit(ld t){ //Q2, 判断某个时间点所有把手位置是否合法
    (是否未发生碰撞)
316.     vector<Point> dots= calc_coor_at_time(t);
317.     vector<Rectangle> Recs=all_Rectangle_gene_from_dots(dots
    );
318.     return !all_rectangle_intersection(Recs);
319. }
320. bool is_d_legit(ld d){ //Q3, 判断螺距是否合法
321.     vector<Point> dots= calc_coor_at_diff_k(d);
322.     vector<Rectangle> Recs=all_Rectangle_gene_from_dots(dots
    );
323.     return !all_rectangle_intersection(Recs);
324. }

```

```

325. void testcase1(){
326.     vector<vector<bool>> recd(301,vector<bool>(224));
327.
328.     console_buf = std::cout.rdbuf();//将程序输出重定向到文件
329.     std::ofstream file("../projects/dot_place_q1");
330.     std::cout.rdbuf(file.rdbuf());
331.
332.     for(int i=0;i<=300;i++){
333.         auto dots= calc_coor_at_time(i);
334.         for(int j=0;j<dots.size();j++){
335.             auto& dot=dots[j];
336.             if(Euler_distance(dot,{0,0})>rA)cout<<"--- ---
\n",recd[i][j]=true;
337.             else cout<<dot.x<<' '<<dot.y<<'\n';
338.         }
339.     }
340.     file.close();
341.     std::cout.rdbuf(console_buf);
342.
343.     console_buf = std::cout.rdbuf();
344.     std::ofstream file2("../projects/dot_place2_q1");
345.     std::cout.rdbuf(file2.rdbuf());
346.
347.     for(int i=0;i<=300;i++){
348.         auto dots= calc_coor_at_time(i);
349.         for(auto& dot:dots){
350.             if(Euler_distance(dot,{0,0})>rA)cout<<"---\n---
\n";
351.             else cout<<dot.x<<'\n'<<dot.y<<'\n';
352.         }
353.     }
354.     file2.close();
355.     std::cout.rdbuf(console_buf);
356.
357.     console_buf = std::cout.rdbuf();
358.     std::ofstream file3("../projects/velocity_q1");
359.     std::cout.rdbuf(file3.rdbuf());
360.     for(int i=0;i<=300;i++){
361.         auto vecs= calc_velocity_at_time(i);
362.         for(int j=0;j<vecs.size();j++){
363.             if(recd[i][j])cout<<"---\n";
364.             else cout<<vecs[j]<<'\n';
365.         }
366.     }

```

```

367.     file3.close();
368.     std::cout.rdbuf(console_buf);
369. }
370. void testcase2(){
371.     ld l=0,r=800;
372.     while(r-l>1e-2){
373.         ld m=(l+r)/2;
374.         if(is_legit(m))l=m;
375.         else r=m;
376.     }
377.
378.     cout<<"Q2: 发生板凳碰撞的时间 "<<l<<endl;
379.
380.     console_buf = std::cout.rdbuf();
381.     std::ofstream file("../projects/dot_place2_q2");
382.     std::cout.rdbuf(file.rdbuf());
383.
384.     auto dots= calc_coor_at_time(1);
385.     vector<bool> recd(dots.size());
386.     for(int i=0;i<dots.size();i++){
387.         if(Euler_distance(dots[i],{0,0})>880)cout<<"---\n---
388.         \n",recd[i]=true;
389.         else cout<<dots[i].x<<'\\n'<<dots[i].y<<'\\n';
390.     }
391.     file.close();
392.     std::cout.rdbuf(console_buf);
393.
394.     console_buf = std::cout.rdbuf();
395.     std::ofstream file2("../projects/velocity_q2");
396.     std::cout.rdbuf(file2.rdbuf());
397.
398.     auto vecs= calc_velocity_at_time(1);
399.     for(int i=0;i<vecs.size();i++){
400.         if(recd[i])cout<<"---\n";
401.         else cout<<vecs[i]<<'\\n';
402.     }
403.     file2.close();
404.     std::cout.rdbuf(console_buf);
405.
406.     console_buf = std::cout.rdbuf();
407.     std::ofstream file3("../projects/dot_place_q2");
408.     std::cout.rdbuf(file3.rdbuf());
409.     for(int i=0;i<dots.size();i++){

```

```

410.         if(recd[i])cout<<"--- ---\n";
411.         else cout<<dots[i].x<<' '<<dots[i].y<<' \n';
412.     }
413.     file3.close();
414.     std::cout.rdbuf(console_buf);
415. }
416. void testcase3(){
417.     ld l=10,r=200;
418.     while(r-l>eps){
419.         ld m=(l+r)/2;
420.         if(is_d_legit(m))r=m;
421.         else l=m;
422.     }
423.
424.     cout<<"Q3: 最小螺距 "<<r<<endl;
425. }
426. Point coor_angle_distance(ld angle,ld distance){ //确定
    angle 之后运行轨迹确定,输出从标定起点开始向后沿着轨迹走 distance 路
    程后到达的坐标
427.     if(distance<=0){
428.         distance=-distance;
429.         ld st_len= per_dis(angle);
430.         ld now_angle= inv_per_dis(st_len+distance/K);
431.         Point res= angle_coordinate(now_angle)*K;
432.         return res;
433.     }
434.     Point start= angle_coordinate(angle)*K;
435.     Point sys_start=start*(-1);
436.     Point mi_2_3=start*(1.0/3)+sys_start*(2.0/3);
437.     Point o1,o2;
438.     ld w=atan(angle);//w:1.50626
439.     ld r1,r2;//Q4 决策下:r1:279.731 r2:139.865 随 angle 递增
    而递增,这是 angle 为 15.476 时的 val
440.     {
441.         ld len1= Euler_distance(mi_2_3,start)/(2.0*angle);
442.         r1=len1*sqrt(angle*angle+1);
443.         Point tp1=(start*0.5)+(mi_2_3*0.5);
444.         Point ve1=start-mi_2_3;
445.         Point pt1= zig_vector({ve1.y, -ve1.x},len1);
446.         if((ve1^pt1)<0)pt1=pt1*(-1);
447.         o1=tp1+pt1;
448.
449.         ld len2= Euler_distance(mi_2_3,sys_start)/(2.0*angle
    );

```



```

450.         r2=len2*sqrt(angle*angle+1);
451.         Point tp2=(sys_start*0.5)+(mi_2_3*0.5);
452.         Point ve2=sys_start-mi_2_3;
453.         Point pt2= zig_vector({ve2.y,-ve2.x},len2);
454.         if((ve2^pt2)<0)pt2=pt2*(-1);
455.         o2=tp2+pt2;
456.     }
457.     if(distance<=2*w*r1){
458.         ld w1=distance/r1;
459.         Point tp1=start-o1;
460.         tp1= Rotate(tp1,w1);
461.         return o1+tp1;
462.     }else if(distance<=2*w*(r1+r2)){
463.         distance=2.0*w*(r1+r2)-distance;
464.         ld w2=distance/r2;
465.         Point tp2=sys_start-o2;
466.         tp2= Rotate(tp2,w2);
467.         return o2+tp2;
468.     }else{
469.         distance-=2.0*w*(r1+r2);
470.         ld st_len= per_dis(angle);
471.         ld now_angle= inv_per_dis(st_len+distance/K);
472.         Point res= angle_coordinate(now_angle)*K;
473.         return res*(-1);
474.     }
475. }
476. ld step=10;
477. ld run_back_distance(ld angle,ld old_distance,ld len){ //由
    angle 确定曲线后,从old_distance 位置往前找第一个与old_distance 代
    表的点的直线距离为len 的点的distance
478.     Point old_pt= coor_angle_distance(angle,old_distance);
479.     ld r=old_distance-len;
480.     ld l=old_distance-len;
481.     while(Euler_distance(old_pt, coor_angle_distance(angle,l
        ))<len)r=l,l-=step;
482.     while(r-l>eps){
483.         ld m=(l+r)/2;
484.         if(Euler_distance(old_pt, coor_angle_distance(angle,
            m))>len)l=m;
485.         else r=m;
486.     }
487.     return r;
488. }

```

```

489. void Print_dis_dots(const vector<Point>& dots){ //调试用,打印点坐标
490.     for(int i=0;i<dots.size()-1;i++)cout<<Euler_distance(dots[i],dots[i+1])<<endl;
491. }
492. bool test_dots_legit(const vector<Point>& dots){ //调试用,判断点坐标数组(把手位置数组)是否合法(得到的板凳矩形数组是否有交)
493.     for(int i=1;i<=223;i++){
494.         if(abs(Euler_distance(dots[i-1],dots[i])-dis_handle[i])>1) {
495.             return false;
496.         }
497.     }
498.     return true;
499. }
500. vector<Point> calc_plc_angle_time(ld angle,ld t,vector<ld>& dist){ //Q4 根据传入angle 确定运动轨迹,根据t 确定运动时间(龙头到达转入点为0时刻),返回把手们的坐标,dist 向外传出距离
501.     vector<Point> res;
502.     ld d0=t*velocity;
503.     dist.emplace_back(d0);
504.     res.emplace_back(coor_angle_distance(angle,d0));
505.     for(int j=2;j<=224;j++){
506.         ld d1= run_back_distance(angle,d0,dis_handle[j-1]);
507.         // if(abs(Euler_distance(coor_angle_distance(angle,d0),coor_angle_distance(angle,d1))-dis_handle[j-1])>1){
508.         //     cout<<"Oops"<<endl;
509.         //     exit(0);
510.         // }
511.         res.emplace_back(coor_angle_distance(angle,d1));
512.         d0=d1;
513.         dist.emplace_back(d0);
514.     }
515.     return res;
516. }
517. vector<Point> calc_plc_angle_time(ld angle,ld t){ //上一个函数去掉dist
518.     vector<Point> res;
519.     ld d0=t*velocity;
520.     res.emplace_back(coor_angle_distance(angle,d0));
521.     for(int j=2;j<=224;j++){
522.         ld d1= run_back_distance(angle,d0,dis_handle[j-1]);
523.         res.emplace_back(coor_angle_distance(angle,d1));
524.         d0=d1;

```

```

525.     }
526.     return res;
527. }
528. vector<ld> calc_velocity_angle_time(ld angle,ld t){ //Q4 根
    据传入 angle 确定运动轨迹, 根据 t 确定运动时间(龙头到达转入点为0 时
    刻), 返回把手们的速度
529.     vector<Point> la= calc_plc_angle_time(angle,t);
530.     vector<Point> ne= calc_plc_angle_time(angle,t+dealt);
531.     vector<ld> res(la.size());
532.     for(int i=0;i<la.size();i++)res[i]= Euler_distance(la[i]
        ,ne[i])/dealt;
533.     return res;
534. }
535. bool is_plc_legit(const vector<Point>& dots){ //判断把手位
    置是否合法, 产生的矩形是否有交
536.     vector<Rectangle> Recs= all_Rectangle_gene_from_dots(dot
        s);
537.     return !all_rectangle_intersection(Recs);
538. }
539. bool is_angle_legit(ld angle){ //Q4 判断某个转入角度是否合
    法
540.     vector<ld> last_dist;
541.     for(ld t=0;t<=100;t+=0.2){
542.         vector<ld> dist;
543.         vector<Point> dots= calc_plc_angle_time(angle,t,dist
            );
544.         if(!is_plc_legit(dots))return false;
545.         if(t>0){
546.             for(int i=0;i<224;i++){
547.                 if(dist[i]<last_dist[i])return false;
548.             }
549.         }
550.         last_dist=dist;
551.     }
552.     return true;
553. }
554. void testcase4(){
555.     ld d=170;
556.     K=d/(2*PI);
557.     ld r=10*PI;
558.     // ld l=PI*4;
559.     // l=15.566371;
560.     // r=16.66666;
561.     // while(r-l>1e-3){

```

```

562. //      ld m=(l+r)/2;
563. //      cout<<l<<' '<<r<<endl;
564. //      if(is_angle_legit(m))r=m;
565. //      else l=m;
566. //  }
567.      r=15.6964;
568.
569.      cout<<"Q4: 决策的最小调头弧度值 "<<r<<endl;//决策的开始调头
        角度
570.
571.      ld angle=r;
572.
573.      console_buf = std::cout.rdbuf();
574.      std::ofstream file("../projects/dot_place_q4");
575.      std::cout.rdbuf(file.rdbuf());
576.      for(int t=-100;t<=100;t++){//外循环时间,内循环点
577.          vector<Point> dots= calc_plc_angle_time(angle,t);
578.          for(int i=0;i<dots.size();i++){
579.              cout<<dots[i].x<<' '<<dots[i].y<<'\n';
580.          }
581.      }
582.      file.close();
583.      std::cout.rdbuf(console_buf);
584.
585.
586.      console_buf = std::cout.rdbuf();
587.      std::ofstream file2("../projects/dot_place2_q4");
588.      std::cout.rdbuf(file2.rdbuf());
589.      for(int t=-100;t<=100;t++){//外循环时间,内循环点
590.          vector<Point> dots= calc_plc_angle_time(angle,t);
591.          for(int i=0;i<dots.size();i++){
592.              cout<<dots[i].x<<'\n'<<dots[i].y<<'\n';
593.          }
594.      }
595.      file2.close();
596.      std::cout.rdbuf(console_buf);
597.
598.
599.      console_buf = std::cout.rdbuf();
600.      std::ofstream file3("../projects/velocity_q4");
601.      std::cout.rdbuf(file3.rdbuf());
602.      for(int t=-100;t<=100;t++){//外循环时间,内循环点
603.          vector<ld> vecs= calc_velocity_angle_time(angle,t);
604.          for(int i=0;i<vecs.size();i++){

```

```

605.         cout<<vecs[i]<<'\n';
606.     }
607. }
608.     file3.close();
609.     std::cout.rdbuf(console_buf);
610. }
611. void testcase5(){
612.     ld d=170;
613.     K=d/(2*PI);
614.     ld angle=15.6964;
615.     ld l=100,r=200;
616.
617.     while(r-l>1e-3){
618.         cout<<l<<' '<<r<<endl;
619.         ld m=(l+r)/2;
620.         velocity=m;
621.         bool ok=true;
622.         for(int t=0;t<=100;t++){
623.             vector<ld> vecs= calc_velocity_angle_time(angle,
        t);
624.             for(ld vv:vecs){
625.                 if(vv>200){
626.                     ok=false;
627.                     break;
628.                 }
629.             }
630.             if(!ok)break;
631.         }
632.         if(ok)l=m;
633.         else r=m;
634.     }
635.     cout<<"Q5: Q4 前提下最大的龙头速度(cm/s) "<<l<<endl;
636. }
637. signed main(){
638.     cout<<fixed<<setprecision(6);// 设置输出精度
639.     ld tot_len=K*per_dis(16*2*PI);
640.     dis_handle.resize(224);
641.     dis_handle[1]=286;
642.     for(int i=2;i<=223;i++)dis_handle[i]=165;// 设置把手间距
643.     testcase1();
644.     testcase2();
645.     testcase3();
646.     testcase4();
647.     testcase5();

```

```
648. }
```

### 附录 3

介绍：该代码为 matlab 代码，作用是画出灵敏度分析时的图像和求出平均距离以便进行灵敏度分析。

```
1. %原始数据
2. data1 = [% 原始数据坐标];
3.
4. %测试数据
5. data2 = [% 灵敏度测试数据];
6.
7. %x 和 y 坐标
8. x1=data1(:,1);
9. y1=data1(:,2);
10. x2=data2(:,1);
11. y2=data2(:,2);
12.
13. ED1=sqrt(x1.^2+y1.^2);
14. ED2=sqrt(x2.^2+y2.^2);
15.
16. %计算平均距离和标准差
17. MD1=mean(ED1);
18. ST1=std(ED1);
19. MD2=mean(ED2);
20. SD2=std(ED2);
21.
22. %打印
23. fprintf('原始数据的平均距离: %.4f\n',MD1);
24. fprintf('原始数据的标准差: %.4f\n',ST1);
25. fprintf('测试数据的平均距离: %.4f\n',MD2);
26. fprintf('测试数据的标准差: %.4f\n',SD2);
27.
28. %绘制差异图 1
29. figure;
30. plot(ED1,'o-b');
31. hold on;
32. plot(ED2,'x-r');
33. title('两组数据的距离差异图');
```

```
34.xlabel('数据点索引');
35.ylabel('距离');
36.legend('原始数据','灵敏度测试数据');
37.grid on;
38.
39.%距离差异
40.disdif=ED1-ED2;
41.
42.%绘制距离差异图 2
43.figure;
44.plot(disdif,'o-k');
45.title('两组数据的距离差异');
46.xlabel('数据点索引');
47.ylabel('距离差异');
48.grid on;
```