# 基于二分算法的板凳龙路径规划与速度控制模型

## 摘 要

“板凳龙”是浙闽地区的传统民俗活动，其舞龙表演通过将数百节板凳相连，形成蜿蜒盘旋的队伍。为了提升舞龙表演的观赏性，本文采用数学模型对其运动路径和速度进行优化分析，旨在为传统民俗活动提供科学支撑，提升文化传承效果。

**针对问题一**，本文通过极坐标方程建立了板凳龙沿螺线盘入的运动模型。首先，通过螺旋线方程与弧长公式计算龙头前把手在各时刻的位置。随后，利用**二分法递推**求解每个把手点的极角，确保把手点间距离符合已知长度，同时，利用极坐标与欧氏距离公式转换得到各把手的笛卡尔坐标。对于速度的计算，本文通过**位移与时间的比值近似求解**各把手的运动速度，最终记录 300 秒内每节板凳的位置与速度情况。分析结果数据可知：同一时刻，越接近螺线中心的把手点，运动速度越大。

**针对问题二，**在问题一的基础上引入碰撞因素，构建板凳龙的运动模型，并考虑实际板凳的大小，通过计算板凳矩形的四个顶点坐标，建立二维几何模型。利用**“分离轴定理”**判断板凳间是否发生碰撞，并结合二分查找算法迭代求解，最终计算出板凳群发生第一次碰撞的**临界时刻约为 412.473838** $s$ 。通过这一方法，我们确定了板凳龙在不发生碰撞时的最大行进时刻及对应的板凳位置和速度。

**针对问题三，**本文在问题二的基础上进一步分析了板凳龙的调头问题，并引入了调头空间的几何约束条件。通过二分法调整螺距，确保龙头在不超过调头空间半径的情况下顺利到达边界。利用“分离轴定理”判断板凳之间是否发生碰撞，并结合迭代算法逐步收敛求解最小螺距，最终确定了满足调头要求的**最小螺距值约为 0.420217** $m$ ，并通过可视化展示了螺线上各把手点位置与板凳群的排布情况。

**针对问题四，**本文研究了是否存在比题干给出的调头曲线更短的设计方案。通过分析调头路径的几何结构，证明**调头路径长度恒定，无法通过调整设计使路径变短**。此外，本问利用**“极角映射”**的方法将极角转换为平面坐标，依次计算出各把手点在调头区域内的精确位置与速度。最终，计算得出了**最短调头路径的长度约为 13.621245** $m$ ，并记录了从-100 秒至 100 秒内舞龙队各节的运动轨迹和速度。

**针对问题五，**需要计算龙头的最大行进速度，确保舞龙队每个板凳的把手点速度均不超过 $2\,m/s$ 。通过建立速度与位移比例关系的模型，确定了各把手点的速度随龙头速度变化的比例。具体求解中，首先固定龙头速度为 1 m/s，在指定范围内不断运动，同时计算所有把手点的速度并找出最大值。最终根据比例关系，将最大把手点速度控制在 2 m/s 以内，从而求解出**龙头的最大行进速度约为 1.246280** $m/s$ 。

**关键词：二分法迭代；分离轴定理；矩阵运算；极角极径；调头路径优化；**

# 一、问题重述

## 1.1 问题背景

“板凳龙”（即“盘龙”）是浙闽地区的传统民俗活动。舞龙队借助“把手”将几十到上百条板凳节节相连，组成一条蜿蜒前行的队伍，形似盘龙。通常舞龙队伍占据的空间越小、行进速度越快，观赏性越好。

随着中国的日益强大，国家对民俗传统文化活动的宣传、保护力度也大大加强，旨在提升公民的文化保护意识与民族文化自信。“板凳龙”作为一项经典的中国传统民俗活动，**如何通过数学模型改进舞龙队的搭建与行进问题以使观众得到更高质的观赏体验，更好地感受美好的民俗文化，是我们亟待解决的问题。**相信高质的表演定会使这些传统民俗文化焕发新的勃勃生机。

## 1.2 问题要求

已知某“板凳龙”由 223 节板凳组成（第 1 节为龙头，最后 1 节为龙尾，其余板凳为龙身）。龙头板凳长 $3.41\,m$，龙身与龙尾板长均为 $2.20\,m$，所有板凳的板宽均为 $0.3\,m$。每节板凳上方均钻有两小孔，小孔直径均为 $5.5\,cm$。小孔中心（即圆心）距离最近的板头 $27.5\,cm$。相邻两条板凳均通过“把手”相互连接，至此盘龙搭建完毕。

基于以上背景信息，拟建立模型解决以下问题：

**问题一：**舞龙队沿着螺距为 $55\,cm$ 的螺线顺时针盘入，龙头速度为 $1\,m/s$。需计算并记录从 0 到 $300\,s$ 内，每秒龙头、龙身和龙尾各把手（中心）的位置和速度。

**问题二：**在舞龙队沿问题一设定的螺线盘入的过程中，需找出不发生碰撞的情况下龙队可以盘入的最大时刻（即板凳间开始发生碰撞的时刻），并记录此时的龙头、龙身和龙尾各把手的位置和速度。

**问题三：**从盘入到盘出，舞龙队由顺时针盘入“调头”切换为逆时针盘出，需借助调头空间（以螺线中心为圆心、直径为 $9\,m$ 的圆形区域）完成切换。计算确定最小螺距，使得龙头前把手能够沿着相应的螺线顺利盘达调头空间的边界。

**问题四：**舞龙队沿螺距为 $1.7\,m$ 的螺线盘入，在调头空间内完成 S 形调头，S 形曲线由两段圆弧相切组成（前段圆弧的半径为后段的 2 倍）。需研究是否可以通过调整圆弧，在保持相切的情况下使调头曲线变短，并计算从调头开始前后各 $100\,s$ 内舞龙队的运动轨迹和速度。

**问题五：**在问题四设定的路径基础上，保持舞龙队各把手的速度不超过 $2\,m/s$，需计算龙头的最大行进速度，确保队伍能够顺利完成调头并安全行进。

# 二、问题分析

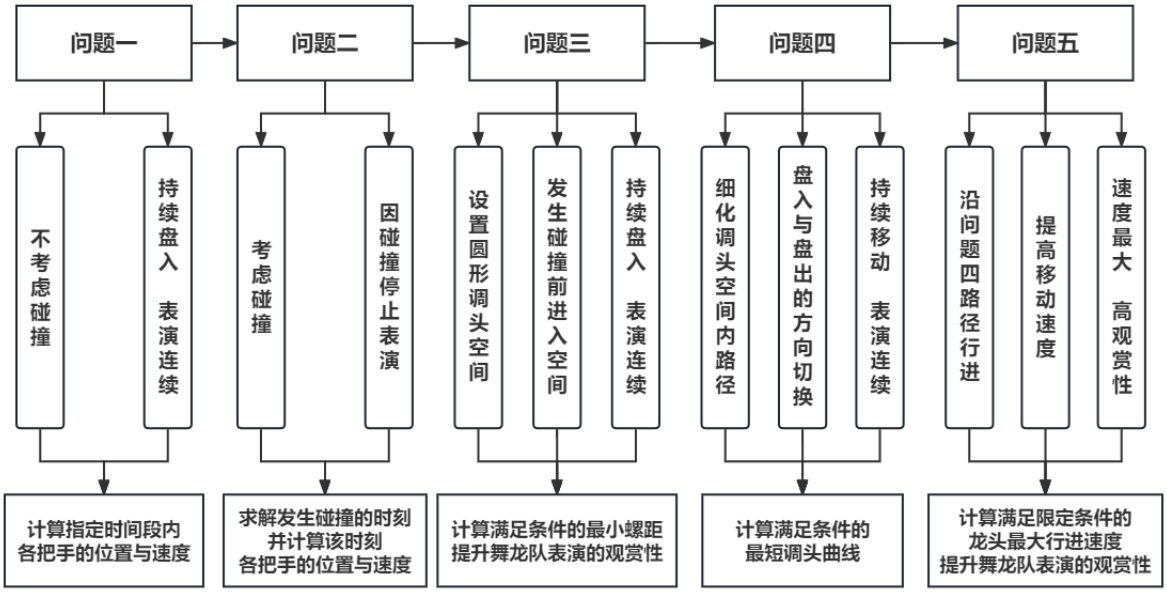



**图 1 各问题间的递进关系及求解流程图**

理解题意发现五道问题层层递进，通过全文的建模，能够帮助舞龙队合理布设路径、选择合适的运动速度，以提升盘龙表演的观赏性，助力弘扬传统民俗文化。

## 2.1 问题一的分析

**对于问题一**，要求分别建立把手点位置与速度的数学模型。在位置模型的建立中，能够通过螺旋线的极坐标方程确定龙头前把手在各时刻的位置，由于每节板凳间距离已知，采用二分法调整确定每节板凳的正确角度，不断递推即可依次计算得到其他节板凳的位置，为简化；在速度模型的建立中，可以利用 $\Delta t \to 0$ 内把手点的位移与 $\Delta t$ 的比值高精度近似求解把手点速度。

## 2.2 问题二的分析

**对于问题二**，需要建立板凳群的碰撞模型。相较于问题一，本问引入碰撞因素，需将问题一中的一维把手点和线段拓展到二维平面，即求解每节板凳矩形的顶点坐标。对于碰撞的判断，引入“分离轴定理”，同时进行二分运算的迭代，即可求解得到板凳群刚好发生碰撞的时刻。

## 2.3 问题三的分析

**对于问题三**，需要求解满足条件最小螺距。本问的解答需要利用问题二中建立的“二分查找和分离轴定理”算法（将二分的变量由极角改为螺距），并且在二分迭代过程中不断微调参数完善模型，最终求出理想题解。

### 2.4 问题四的分析

**对于问题四**，需要判断是否存在比题目给出的调头路径更短的路径设计方案并求解某些时刻把手点的位置与速度。利用多个三角形间的相互关系与几何运算，不难得出结论“调头路径上两圆弧所在圆的半径和不变”，即调头路径长度不变。又依据已知的半径倍数关系，即可解得多点坐标用于后文求解；对于后者的求解，本问仍采用“二分法”，但需要将极角$\theta$与调头路径中圆弧角进行关系映射，使得二分的变量$\theta$连续单调。

### 2.5 问题五的分析

**对于问题五**，需要求解满足条件的龙头最大行进速度。在问题一的基础上，已知把手点的速度互成比例关系（大小随时间、位置改变），只需让龙头保持$1\,m/s$在指定区间内运行，同时计算并记录最大把手点速度$v_{\max}$。最终得到最大龙头速度为$2/v_{\max}$。

## 三、模型假设

1、模型建立统一使用$m$为单位；

2、假设$t=0s$时，舞龙队整体由龙头至龙尾均按螺线方程分布，即所有把手点均已盘入螺线，螺线区域足以容纳整条板凳龙队列；

3、假设板凳龙的每个板凳节是刚性体，即不发生形变；

4、假设连接板凳的“把手”在转动或移动时没有摩擦力影响，连接不影响运动；

5、假设螺线为理想等距螺线，且保持螺距恒定（如题中给出的螺距$55\,cm$）；

6、忽略舞龙队表演者的体积；

7、假设龙头进入调头空间后直接开始调头。

## 四、符号说明

| 符号 | 解释说明 | 单位 |
|---|---|---|
| $pitch$ | 螺距 | $m$ |
| $v_{head}$ | 龙头前把手速度 | $m/s$ |
| $\theta$ | 极角 | $rad$ |
| $r(\theta)$ | 极径 | $m$ |
| $r_{\text{pitch}}$ | 二分上界 | \ |
| $m_{\text{pitch}}$ | 二分中值点 | \ |
| $l_{\text{pitch}}$ | 二分下界 | \ |

# 五、问题一模型的建立与求解

本问的求解需要模拟“板凳龙”沿着一个等距螺线运动的过程，其中板凳龙的龙头行进速度为 $1\,m/s$，螺距为 $55\,cm$。龙头、龙身及龙尾各把手中心点均沿螺线移动，要求记录 $300\,s$ 内每秒的各把手的运动状态（位置和速度）。

## 5.1 把手位置模型的建立与求解

设定等距螺线的极坐标方程为 $r(\theta)=\frac{pitch}{2\pi}\cdot\theta$。其中，$pitch$ 表示螺线的螺距，问题一中 $pitch=0.55m$，$\theta$ 是极角，$r(\theta)$ 是对应角度的半径。由于螺线上每点对应唯一极角，所以在极角已知的前提下，能够利用该螺线方程，求解得龙头、龙身和龙尾各把手中心点的极坐标。

### 5.1.1 计算螺旋线弧长求解龙头前把手位置

螺线的弧长是由极角 $\theta$ 随时间演变而产生的运动轨迹。为了计算某一时间段内龙头前把手的移动弧长，现对以下方程进行积分：

$$L=\int_{\theta_1}^{\theta_2}\sqrt{\left(\frac{dr}{d\theta}\right)^2+r^2(\theta)}\,d\theta \tag{1}$$

其中，$\frac{dr}{d\theta}=\frac{pitch}{2\pi}$，因此弧长的计算可转变为：

$$L=\int_{\theta_1}^{\theta_2}\sqrt{\left(\frac{pitch}{2\pi}\right)^2+\left(\frac{pitch}{2\pi}\cdot\theta\right)^2}\,d\theta \tag{2}$$

已知龙头前把手切向速度为 $1\,m/s$、沿螺线匀速运动，走过路径即为弧长 $L$，将弧长数值带入公式（2），即可解得龙头前把手极角 $\theta$ 的变化，并得到某一时刻 $t_i$ 龙头前把手的极角 $\theta_i$ 与极坐标 $r(\theta_i)$，从而根据转变定理公式求得相应笛卡尔坐标 $(x_i,y_i)$：

$$\begin{cases}x_i=r(\theta_i)\cdot\cos(\theta_i)\\ y_i=r(\theta_i)\cdot\sin(\theta_i)\end{cases} \tag{3}$$

### 5.1.2 各把手（排除龙头前把手）的位置计算与极坐标变换

已知每一节板凳相对于前一节有一个固定的长度，龙头的板长为 $341\,cm$，龙身和龙尾的板长均为 $220\,cm$。为了计算每节板凳在螺线上的位置，首先需要确定它们在极坐标中的角度 $\theta$ 和对应的半径 $r(\theta)$。

每一节板凳的前把手位置可以通过递推的方式计算：从龙头开始，根据龙头前把手的极角 $\theta$，递归生成下一节板凳前把手的极角。递归过程需要借助“二分法”找到

一个合适的极角，确保两节板凳之间的距离符合其实际长度（利用公式（3）转化为欧氏距离公式计算）。

**【二分法确定极角】**

考虑到二分法的特殊性质，需要确定初始二分（针对龙头）的下界；如图 **2** 所示，点 $A$ 为龙头前把手，点 $B$ 为龙头后把手，连接点 $A$ 与螺线中心点并延长得到虚线段 $AB_{\lim}$，当 $|AB_{\lim}|<$ 龙头长度 时，龙头无法继续盘入，故规定对于极角的二分范围为 $(\theta_0,\theta_0+\pi)$，其中 $\theta_0$ 表示龙头前把手的极角。

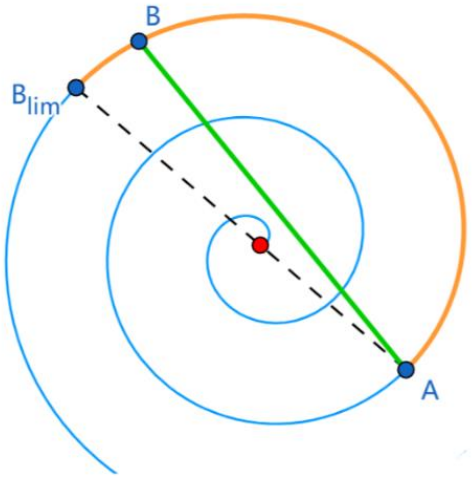



**图 2 确定二分下界**

二分法求解过程中，本问设置二分精度 $EPS=1\times10^{-12}$。计算每次二分所得点与上一已知把手中心点的距离，并将所得距离与已知龙头/龙身/龙尾的实际长度进行比较，若计算所得距离小于实际长度，则将二分区间向逆时针方向缩小（即选取该二分点的较大区间）。数学描述如下：

设上一节板凳的极角为 $\theta_1$，需要找到一个合适的 $\theta_2$ 使得极点 $(\theta_1,r(\theta_1)$ 和 $(\theta_2,r(\theta_2)$ 之间的极坐标距离等于当前板凳的长度。这个距离的计算公式为：

$$d=\sqrt{\left(r(\theta_2)\cdot\cos(\theta_2)-r(\theta_1)\cdot\cos(\theta_1)\right)^2+\left(r(\theta_2)\cdot\sin(\theta_2)-r(\theta_1)\cdot\sin(\theta_1)\right)^2} \qquad (4)$$

其中 $d$ 是两节板凳前把手之间的距离。当距离小于给定的板长时，调整极角 $\theta_2$，通过二分法逼近最佳解。

（*举例说明：计算龙头后把手时，已知龙头长 $LEN$ （即龙头前后把手中心点间距离），初始二分区间为 $(\theta_0,\theta_0+\pi)$，若第一次二分所得点与前把手中心点连线的欧氏距离 $len$ 满足 $len<LEN$ 时，则下一个二分区间为 $(\theta_0+\frac{\pi}{2},\theta_0+\pi)$。）

至此充分利用二分法进行递归运算，最终得到 224 个把手点的极角，同时借助公式（3）能够将这组极坐标点转换为笛卡尔坐标，记录于 result1.xlsx“位置”表中。

## 5.2 把手速度模型的建立与求解

龙头前把手的行进速度保持 $1\,m/s$，但由于板凳龙的每节段在螺线上的位置不同，龙身和龙尾各节的速度会有所变化。为了计算这些速度，本问推导得到如下计算公式：

$$v=\frac{\text{待解把手点位移}}{\text{龙头前把手位移}}\cdot v_{\text{head}} \tag{5}$$

**【公式（5）的推导】**

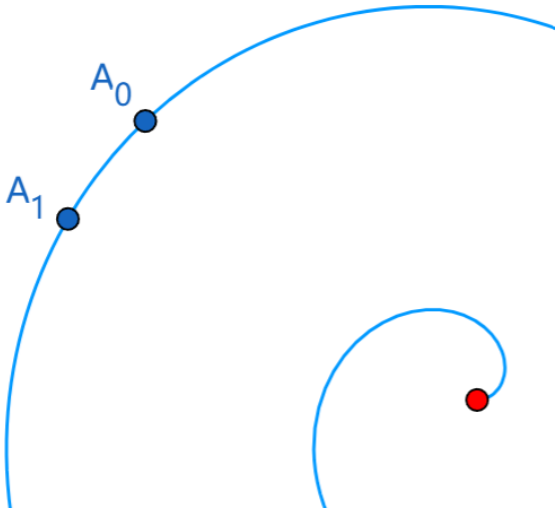



**图 3** $(t_1-t_0)\to 0$ 把手点位移示意图

如图 3 所示，$t_0$ 时刻某把手中心位于 $A_0$ 处，$t_1$ 时刻位于 $A_1$ 处，且 $(t_1-t_0)\to 0$，必然有 $A_0$ 与 $A_1$ 处的极角之差 $(\theta_1-\theta_2)\to 0$，由此能够利用两点间位移近似求解出该把手在 $A_1$ 处的运动速度，且精度极高，不必使用螺线的弧长公式进行求解，由是得：

$$v_1=\frac{|A_0A_1|}{\Delta t} \tag{6}$$

又因为螺线上的把手同步运动，在计算速度时 $\Delta t$ 相同，所以螺线上任意把手点的速度 $\propto$ 位移。题目已知龙头前把手运动速度 $v_{head}=1m/s$，只需求解出每个把手点（除去龙头前把手）在某一 $\Delta t$ 内产生的位移即可得到该点运动速度 $v_i$：

$$v_i=\frac{x_i}{x_{head}}\cdot v_{head} \tag{7}$$

其中，$x_i$ 为待求把手点产生的位移，$x_{head}$ 为龙头前把手的位移。

## 5.3 结果展示与分析

### 5.3.1 部分结果展示

**表 1 部分位置结果**

| | 0 s | 60 s | 120 s | 180 s | 240 s | 300 s |
|---|---|---|---|---|---|---|
| 龙头 x (m) | 8.8 | 5.79920903 | -4.08488744 | -2.963608686 | 2.59449372 | 4.420274064 |
| 龙头 y (m) | 0.0 | -5.771092341 | -6.304478922 | 6.09478024 | -5.356742803 | 2.320428674 |
| 第 1 节龙身 x (m) | 8.363823851 | 7.456757763 | -1.445472733 | -5.237117895 | 4.821220949 | 2.459488545 |
| 第 1 节龙身 y (m) | 2.826543891 | -3.44039884 | -7.405882575 | 4.359627256 | -3.561948706 | 4.402475779 |
| 第 51 节龙身 x (m) | -9.518732315 | -8.686317018 | -5.543149523 | 2.8904553 | 5.980010821 | -6.301345864 |
| 第 51 节龙身 y (m) | 1.341136873 | 2.540107836 | 6.37794586 | 7.24928898 | -3.827758216 | 0.465828976 |
| 第 101 节龙身 x (m) | 2.91398337 | 5.687115541 | 5.361938734 | 1.898794329 | -4.91737135 | -6.23772235 |
| 第 101 节龙身 y (m) | -9.91831102 | -8.001383839 | -7.557638257 | -8.471614101 | -6.379874255 | 3.936007686 |
| 第 151 节龙身 x (m) | 10.86172628 | 6.68231146 | 2.388757156 | 1.005153982 | 2.965378108 | 7.040740228 |
| 第 151 节龙身 y (m) | 1.828753459 | 8.134543947 | 9.727411332 | 9.424750976 | 8.399720549 | 4.393013132 |
| 第 201 节龙身 x (m) | 4.555102231 | -6.619663631 | -10.62721053 | -9.287719839 | -7.457151215 | -7.458662073 |
| 第 201 节龙身 y (m) | 10.72511774 | 9.025570323 | 1.359847195 | -4.246672902 | -6.18072612 | -5.26338411 |
| 龙尾（后）x (m) | -5.30544404 | 7.364557221 | 10.97434827 | 7.383895632 | 3.241051078 | 1.785032734 |
| 龙尾（后）y (m) | -10.67658431 | -8.79799165 | 0.84347322 | 7.492370456 | 9.469336433 | 9.301164015 |

**表 2 部分速度结果**

| | 0 s | 60 s | 120 s | 180 s | 240 s | 300 s |
|---|---|---|---|---|---|---|
| 龙头 (m/s) | 1 | 1 | 1 | 1 | 1 | 1 |
| 第 1 节龙身 (m/s) | 0.999971041 | 0.999961124 | 0.9999451 | 0.999916654 | 0.999858618 | 0.999709459 |
| 第 51 节龙身 (m/s) | 0.999741897 | 0.999662104 | 0.99953821 | 0.999330596 | 0.998941088 | 0.998064765 |
| 第 101 节龙身 (m/s) | 0.999574974 | 0.999453146 | 0.999269008 | 0.998970657 | 0.998435289 | 0.997301832 |
| 第 151 节龙身 (m/s) | 0.999447967 | 0.999298935 | 0.999077668 | 0.998727107 | 0.998114748 | 0.996861266 |
| 第 201 节龙身 (m/s) | 0.999348031 | 0.999180404 | 0.998934664 | 0.998551342 | 0.997893466 | 0.996574363 |
| 龙尾（后）(m/s) | 0.999310514 | 0.999136546 | 0.998882623 | 0.998488613 | 0.997816391 | 0.996477533 |

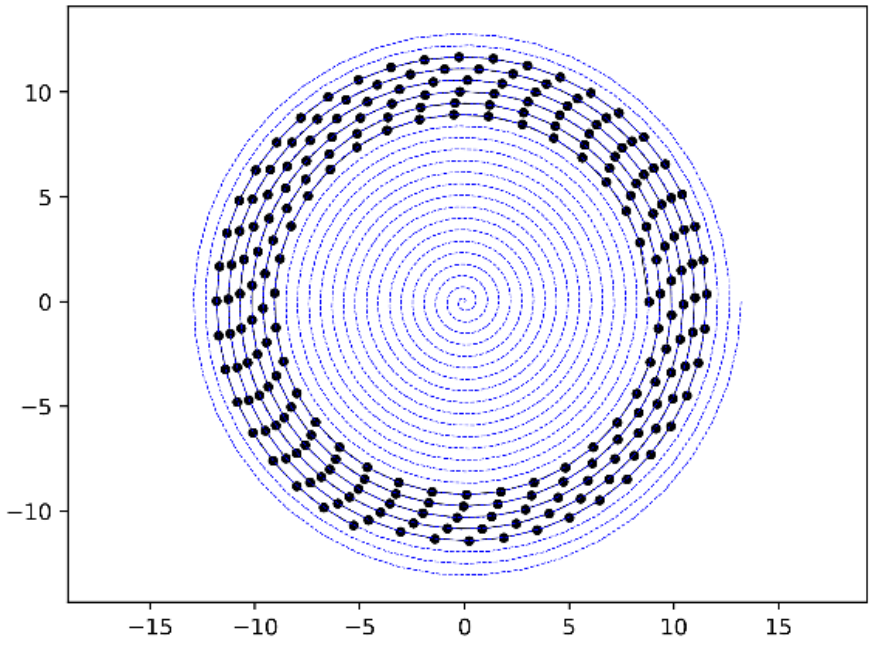
### 5.3.2 结果分析与可视化

**【结果分析】**

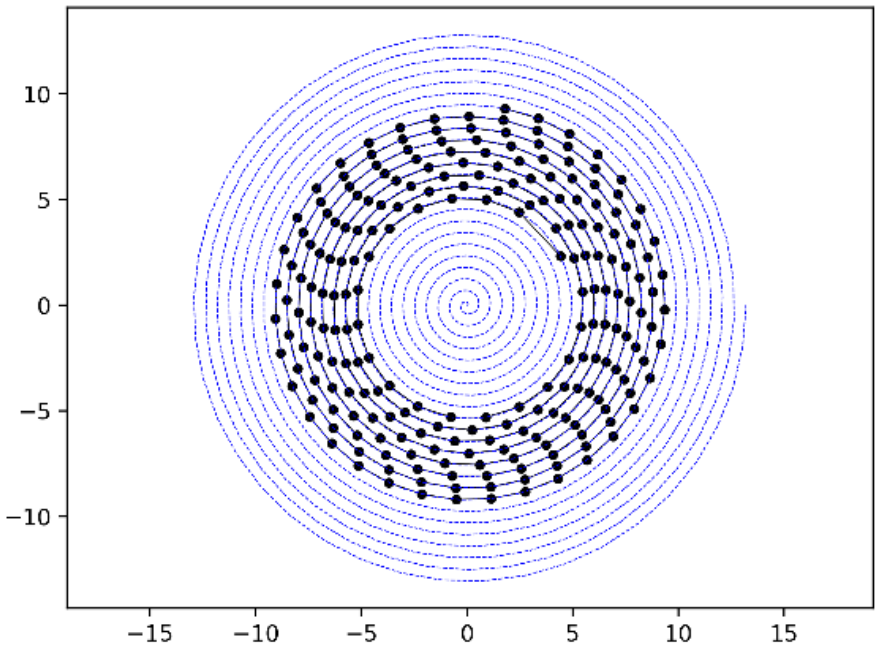
1、观察表 2 中数据易知，同一时刻，越接近螺线中心的把手点，运动速度越大；

2、随着龙头持续盘入螺线，逐渐靠近螺线中心，除龙头外其余所有把手速度随时间呈减小趋势。

【部分时刻把手位置可视化】




(a) $t = 0s$




(b) $t = 300s$

**图 4 部分时刻把手位置的可视图像**

# 六、问题二模型的建立与求解

在问题一的基础上，本问引入碰撞因素，需要计算板凳间不发生碰撞的最大时刻，即板凳群刚好发生碰撞的时刻。本问拟运用“向量和矩阵的变换运算”、“分离轴定理（SAT, Separating Axis Theorem）”和“二分查找算法”进行相关求解。

## 6.1 板凳龙运动模型的建立

问题一已做出“把手”点的运动路线模拟，能够计算得到“把手”点的位置坐标；而问题二的求解无法仅仅依赖点坐标完成，需要考虑实际的板凳大小，即需要对实际的板凳龙队列进行建模（考虑面积与板凳矩形的四顶点坐标）。此模型的建立，关键在于如何依赖两个“把手”点坐标和已知的板凳属性，计算得到板凳矩形的四个顶点坐标。

**【具体建模】**

取两相邻“把手”点A与点B，其笛卡尔坐标分别为$(x_1,y_1),(x_2,y_2)$。设线段$AB$的长度为$|\mathrm{AB}|$、向量$\overrightarrow{AB}$的单位向量$\alpha=(vec_x,vec_y)$，存在以下公式能够表示各参数之间的关系：

$$|AB|=\mathrm{dist}=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2} \tag{8}$$

$$\mathrm{vec}_x=\frac{x_2-x_1}{dist} \tag{9}$$

$$\mathrm{vec}_y=\frac{y_2-y_1}{dist} \tag{10}$$

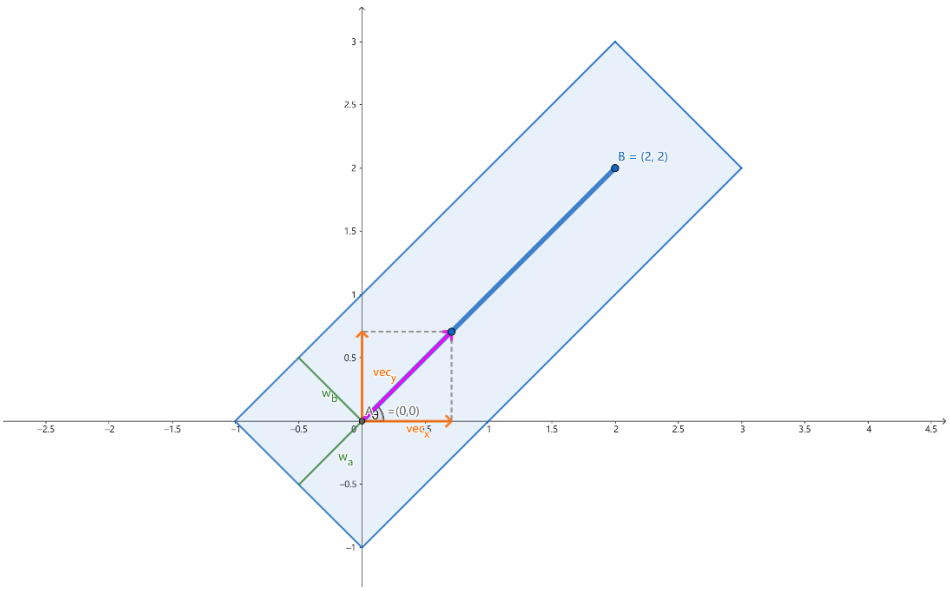



**图 5 矩形顶点坐标求解示意图**

具体运算中，定义一个旋转矩阵$V$（式（11））和表示长度的对焦矩阵$D$（式（12）），能够得到如公式（13）所示的偏移矩阵$Bias$：

$$V=\begin{pmatrix} vec_x & -vec_y \\ vec_y & vec_x \end{pmatrix}=\begin{pmatrix} \dfrac{x_2-x_1}{dist} & -\dfrac{y_2-y_1}{dist} \\ \dfrac{y_2-y_1}{dist} & \dfrac{x_2-x_1}{dist} \end{pmatrix} \tag{11}$$

$$D=\begin{pmatrix} w_A & 0 \\ 0 & w_B \end{pmatrix} \tag{12}$$

$$Bias=V\cdot D=\begin{pmatrix} w_A\cdot vec_x & -w_B\cdot vec_y \\ w_A\cdot vec_y & w_B\cdot vec_x \end{pmatrix} \tag{13}$$

其中$w_A$和$w_B$分别表示矩形内线段的两个端点（即“把手”点）到矩形两条相邻边（即“板凳边”）的距离。

最后，将偏移加到两个端点$A$、$B$上，能够得到四个顶点的位置：

$$P=\begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix}=\begin{pmatrix} x_1 & y_1 \\ x_1 & y_1 \\ x_2 & y_2 \\ x_2 & y_2 \end{pmatrix}+\begin{pmatrix} -w_A\cdot vec_x-w_B\cdot vec_y & -w_A\cdot vec_y+w_B\cdot vec_x \\ -w_A\cdot vec_x+w_B\cdot vec_y & -w_A\cdot vec_y-w_B\cdot vec_x \\ w_A\cdot vec_x+w_B\cdot vec_y & w_A\cdot vec_y-w_B\cdot vec_x \\ w_A\cdot vec_x-w_B\cdot vec_y & -w_A\cdot vec_y-w_B\cdot vec_x \end{pmatrix} \tag{14}$$

## 6.2 碰撞问题的模型建立与求解

首先将每节板凳抽象为二维平面内的矩形，以建立平面几何模型。本问借助二维层面的分离轴定理进行相关判断与求解。

### 6.2.1 板凳龙碰撞问题中分离轴定理的二维呈现

“分离轴定理”是检测凸多边形的碰撞问题领域中具有极大优势的一个算法，通过检测图形的每一条边对应的分离轴上“两个图形的投影重叠情况”，能够判断二者是否碰撞——只要存在一条分离轴，满足两凸多边形在其上的投影不重叠，即可判定二者是分离的。

分离轴定理要求被测多边形为凸多边形，本题中的板凳均可抽象为矩形，符合要求。每个矩形都具有两条分离轴（或投影轴），单条分离轴就是与“矩形某一边长的法向所在直线”平行的直线。如图 6 所示，直线$AB$即为矩形$A'B'C'D'$的边$A'B'$对应的分离轴（虚线所示即为边$A'B'$的法向所在直线，由是可得平行于虚线的分离轴$AB$，它也是边$C'D'$的分离轴；此时，将图中两凸多边形分别投影至分离轴$AB$，显然不存在重叠的情况，因此能够判定三角形与矩形是分离的）。

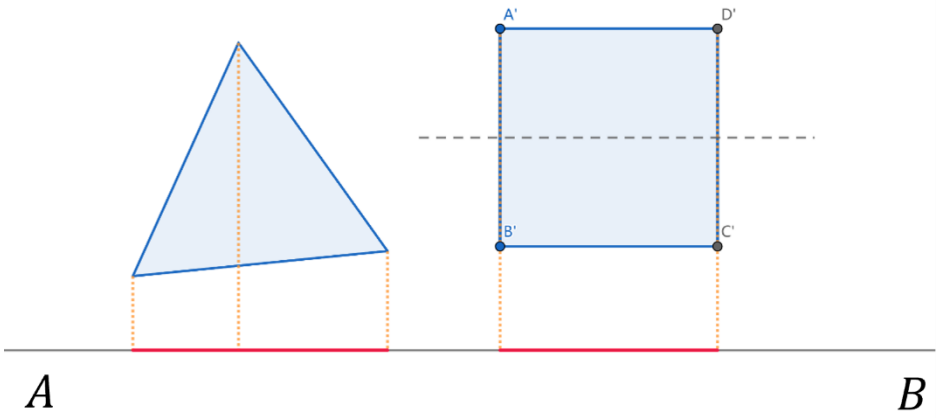
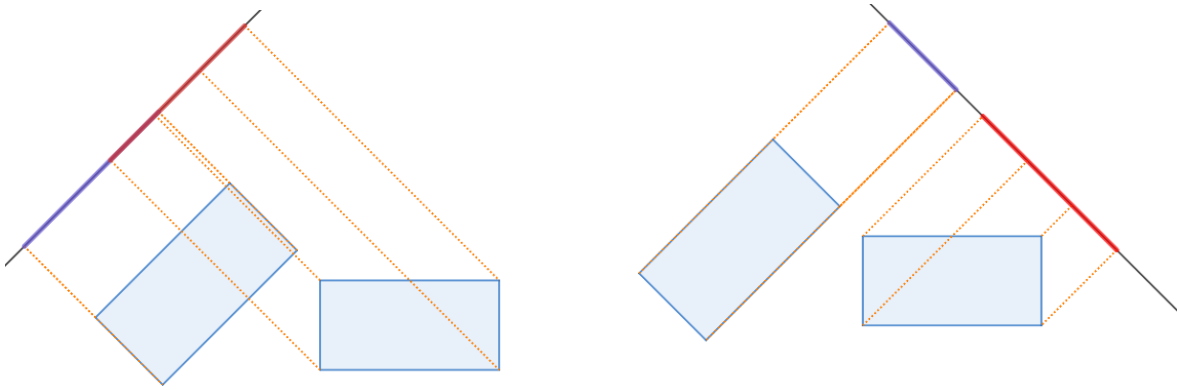



**图 6 分离轴定理的举例说明**

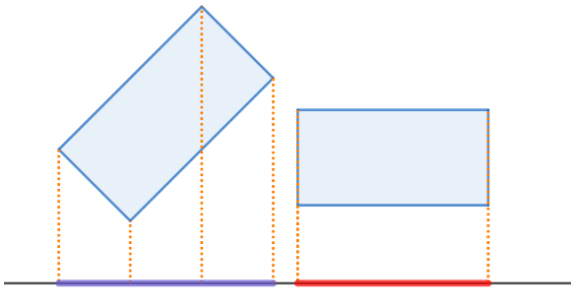
图 7、图 8 是分离轴定理应用于本问的具体情况，两图分别展示并说明了两矩形相交和分离两种情况下，分离轴定理的作用情况与判定。



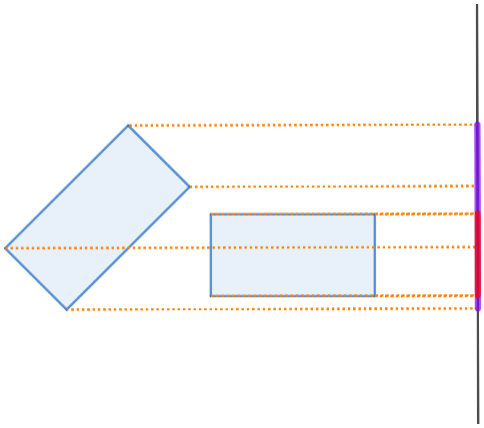
(a) 左侧矩形的分离轴 1



(b) 左侧矩形的分离轴 2



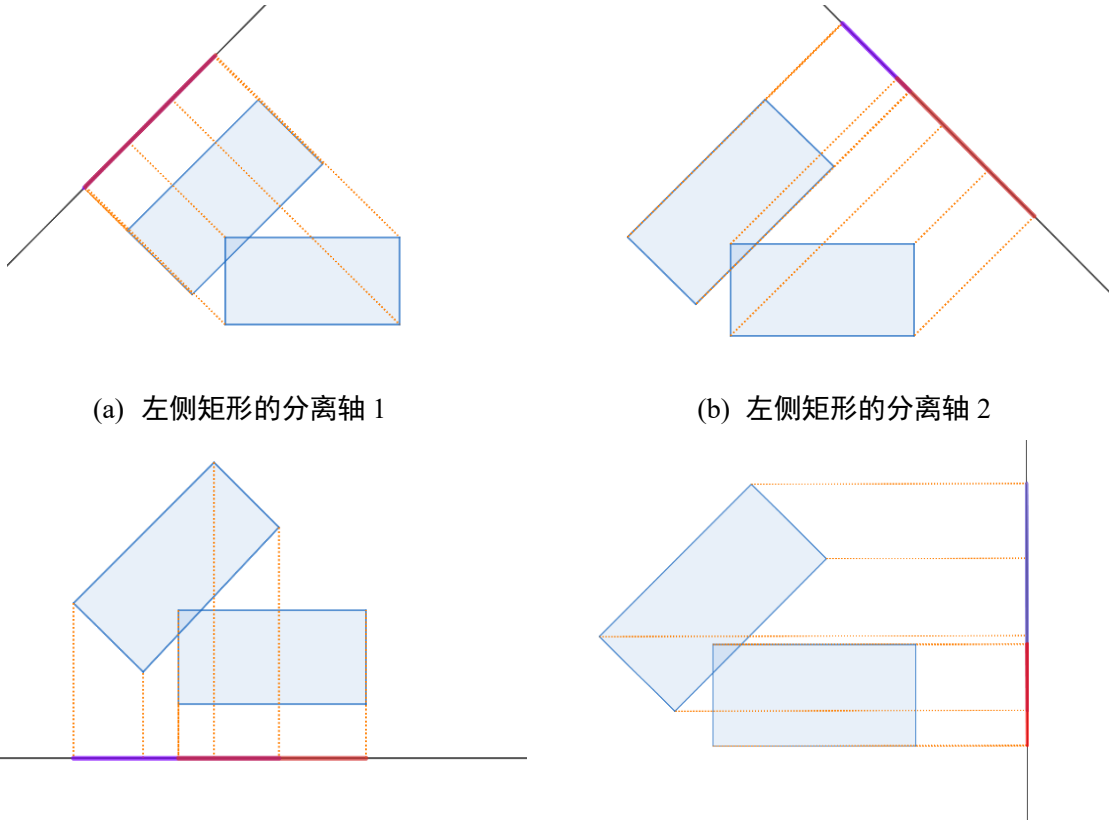
(c) 右侧矩形的分离轴 1



(d) 右侧矩形的分离轴 2

**图 7 两矩形分离的情况**

（两矩形在(b)、(c)图中分离轴上的投影均不重叠，任意一条都可推知两矩形是分离的）



(a) 左侧矩形的分离轴 1　　(b) 左侧矩形的分离轴 2

(c) 右侧矩形的分离轴 1　　(d) 右侧矩形的分离轴 2

**图 8 两矩形相交的情况**

（所有情况中分离轴上的两矩形投影均重叠，可推知两矩形是相交的）

### 6.2.2 碰撞问题的求解

本问的解答同样借助二分查找算法，需要不断迭代直至找到一个临界值$t_x$，此时板凳群在运动中发生第一次碰撞。

二分查找的上下界设定尤为重要，本问初始上界已知为$32\pi$，初始下界的计算与问题一相同。求解过程中，本问设置二分精度$EPS=1\times10^{-12}$。利用分离轴定理对每次二分所得板凳群进行判定，若不发生碰撞（即所有板凳矩形均分离），则将二分区间向顺时针方向缩小（即选取该二分点的较小区间）。若发生碰撞，则选取较大的半区间。

## 6.3 结果展示与可视化

程序计算得：**盘入终止时刻为 412.4738376821458s**（此时龙头前把手极角为 26.14653475889224rad）。

### 6.3.1 部分结果展示

龙头前把手、第 1、51、101、151、201 条龙身前把手及龙尾后把手的位置和速度如表 3 所示。（所有要求数据均已记录在 result2.xlsx 中）

**表 3 部分位置与速度结果**

| | 横坐标 x (m) | 纵坐标 y (m) | 速度 (m/s) |
|---|---|---|---|
| 龙头 | 1.209931025 | 1.942784028 | 1 |
| 第 1 节龙身 | -1.643791702 | 1.753399236 | 0.991550932 |
| 第 51 节龙身 | 1.281201114 | 4.32658823 | 0.976858129 |
| 第 101 节龙身 | -0.536246214 | -5.880137964 | 0.974550555 |
| 第 151 节龙身 | 0.968840233 | -6.95747913 | 0.973608544 |
| 第 201 节龙身 | -7.89316124 | -1.230763703 | 0.973096586 |
| 龙尾（后） | 0.95621705 | 8.322735966 | 0.972938491 |

### 6.3.2 结果可视化

**【碰撞发生时的板凳排布可视化】**

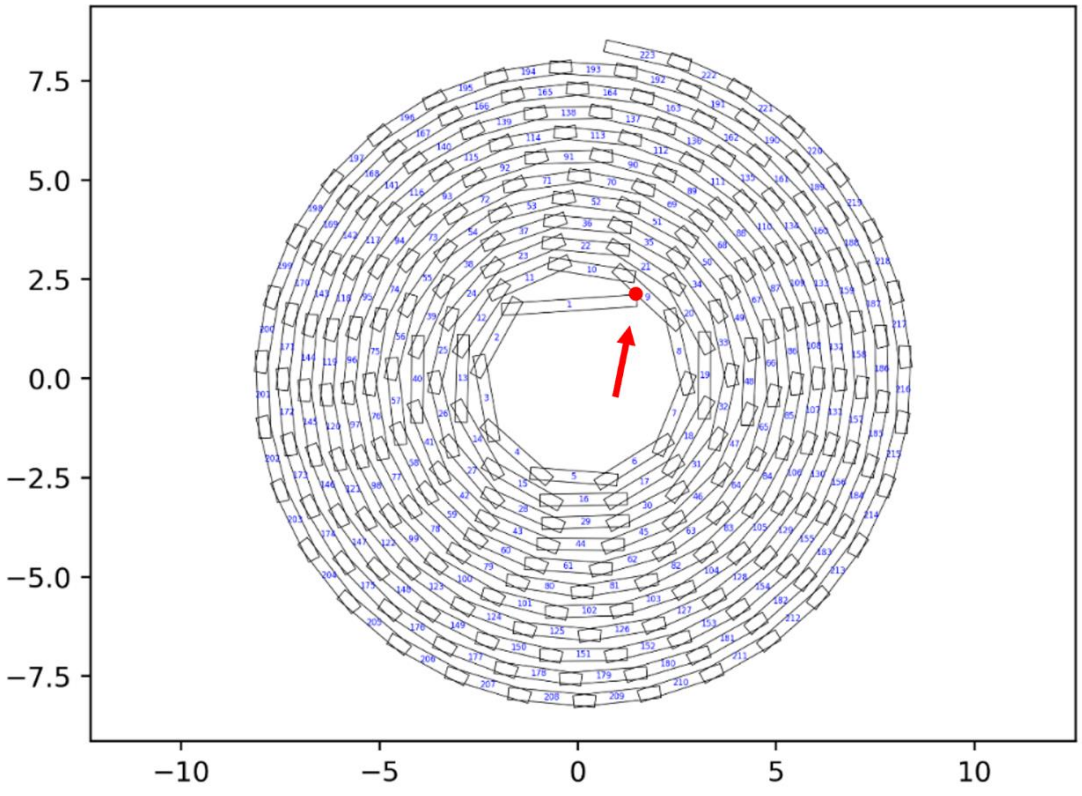



**图 9 碰撞发生时的板凳排布可视化图**

（图中红点所示即为发生碰撞的位置）

# 七、问题三模型的建立与求解

问题三在问题二的基础上针对碰撞问题提出进一步的要求——设定调头空间，并需要求解满足题意的最小螺距。本问继续使用上文的“二分法”和“分离轴定理”，进行相关模型的建立与求解。

## 7.1 最小螺距的求解

### 7.1.1 调头空间的几何约束

易知当龙头到达调头空间时，其在螺线上的极径 $r_{\max}$ 不能超过调头空间的半径长，否则龙头就会离开调头区域。现已知调头空间直径为 $9\,m$，因此调头时龙头的最大极径 $r_{\max}$ 必须小于等于 $4.5\,m$（即调头空间的半径）。

为了找到满足这一条件的最小螺距，螺线的极径 $r(\theta)=\dfrac{pitch}{2\pi}\cdot\theta$ 应满足：

$$r=4.5\text{ 时 }\theta=\frac{4.5\cdot 2\pi}{pitch} \tag{15}$$

通过调整螺距，可以控制龙头历经多久到达调头空间的边界。

### 7.1.2 二分法求解最小螺距

本问沿用了问题二中建立的“板凳龙运动模型”（见 6.1）和“分离轴定理”，并通过二分法来搜索满足条件的最小螺距。搜索逻辑如下：

初始搜索范围为 $[l_{\text{pitch}},r_{\text{pitch}}]$，其中 $l_{\text{pitch}}=0.1m$，$r_{\text{pitch}}=2.0m$，每次二分通过取中间值 $m_{\text{pitch}}=\dfrac{l_{\text{pitch}}+r_{\text{pitch}}}{2}$，计算螺旋线的半径增长情况以及是否会在螺旋线盘入的过程中产生板凳之间的碰撞。

**【迭代收敛求解最小螺距】**

当存在碰撞时，代码将下界 $l_{\text{pitch}}$ 更新为当前中间值 $m_{\text{pitch}}$，即加大螺距以避免碰撞；当不存在碰撞时，将上界 $r_{\text{pitch}}$ 更新为 $m_{\text{pitch}}$，即缩小螺距，直到找到满足条件的最小螺距。迭代停止的条件为上界与下界之差小于精度值 $EPS=1\times10^{-12}$，此时的 $l_{\text{pitch}}$ 即为最小螺距。

## 7.2 结果展示与可视化

### 7.2.1 结果展示

程序计算得：**能够到达调头区边缘的最小螺距约为 $0.420216898051877\,m$ 。**

### 7.2.2 结果可视化

图 10 和图 11 分别为，螺距取最小值时，螺线上各把手点位置示意图和板凳群实际排布图。

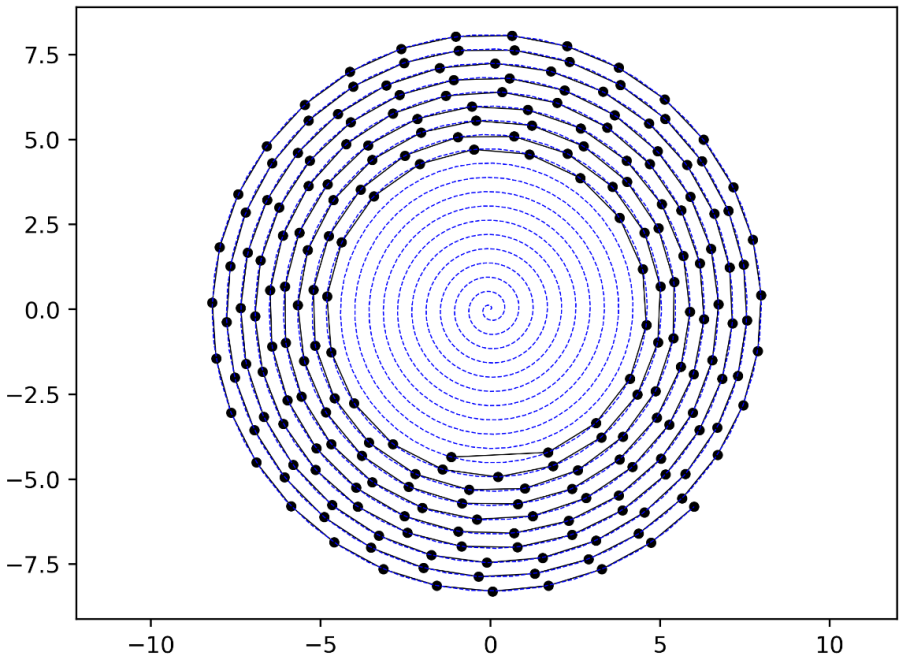



**图 10 各把手点位置示意图**

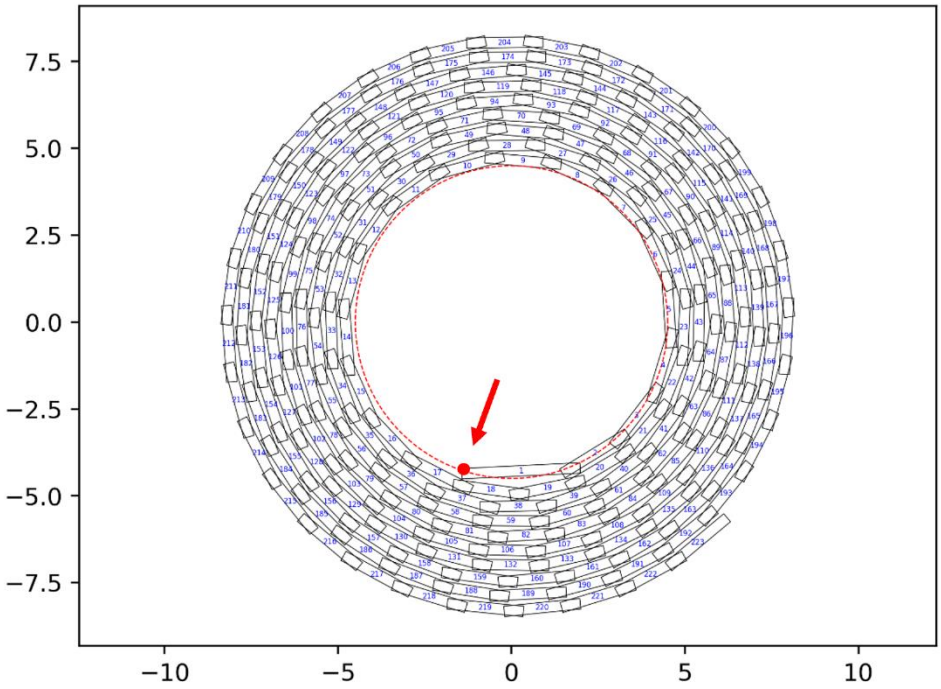



**图 11 板凳群实际排布图**

（图中红色虚线圆内部即为调头空间，龙首位于红点处）

# 八、问题四模型的建立与求解

本问需要判断是否存在比题干中给出的调头曲线方案更短的调头曲线设计，且该路径符合本问的相切要求，最终记录要求的位置与速度。通过本问题干给出的“盘入螺线与盘出螺线关于螺线中心呈中心对称”，易知“盘入螺线末端点与盘出螺线起始端点关于中心对称”，以下建模过程不再对其进行解释说明。

## 8.1 调头路径长的建模与求解

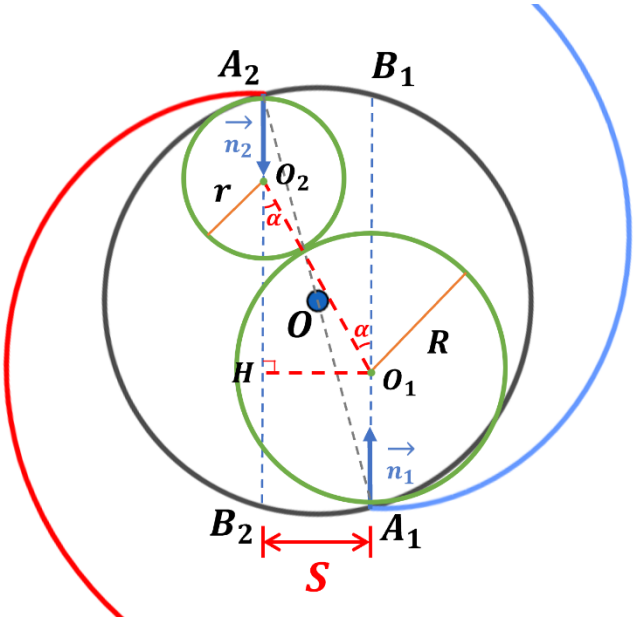



**图 12 辅助证明图(a)**

如图 12 所示，其中右侧蓝色曲线表示“盘入螺线”，左侧红色曲线表示“盘出螺线”，绿色圆表示“调头路径圆弧所在圆”（此处称为“路径圆”），蓝色箭头 $\vec{n_1}$ 、 $\vec{n_2}$ 表示“盘入、盘出螺线与调头路径圆弧切线的法向”，蓝色虚线 $A_1B_1$ 、 $A_2B_2$ 表示“两法向所在直线”。

由于路径圆与盘入、盘出螺线均相切，即两切点 $A_1$ 、 $A_2$ 处切线的法向 $\vec{n_1}$ 、 $\vec{n_2}$ 既满足螺线，又满足路径圆 $\odot O_1$ 与 $\odot O_2$，因此路径圆的圆心 $O_1$ 、 $O_2$ 必然位于两法向所在直线（即蓝色虚线 $A_1B_1$ 、 $A_2B_2$ ）。

【$\triangle O_1O_2H$】

连接两路径圆的圆心 $O_1$ 、 $O_2$ 必然经过两圆切点，得到图中红色虚线段 $O_1O_2$ 。假设两路径圆半径分别为 $R$ 、 $r$，易知线段 $O_1O_2$ 长为 $(R+r)$ 。设两法向所在直线间距离为 $S$，即线段 $O_1H$ 长为 $S$，在 $\triangle O_1O_2H$ 中，设 $\angle HO_2O_1=\alpha$，由三角形基本性质易知：

$$(r+R)\cdot\sin\alpha=S \tag{16}$$

由题意知，需求解出最短的调头曲线，即调头圆弧长度之和最小。根据圆的基本性质易知“弧长＝弧度×半径”，令调头曲线长为$L$，可知：

$$L=(r+R)\cdot(\pi-\alpha) \tag{17}$$

联立式（16）、（17）可得：

$$L=\frac{S}{\sin\alpha}\cdot(\pi-\alpha) \tag{18}$$

【$\triangle OA_1D$】

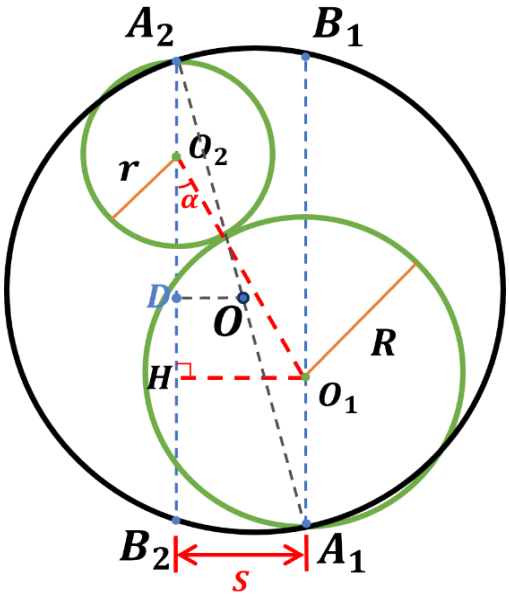



**图 13 辅助证明图(b)**

（图中$A_1$为盘入点，$A_2$为盘出点）

由上文推导知$|O_1O_2|=R+r$，又$\because |A_1O_1|=R,|A_2O_2|=r$，$\therefore |O_1O_2|=|A_1O_1|+|A_2O_2|$。

已知调头空间直径为$9\,m$（即$|OA_2|=4.5m$），假设线段$A_1B_1$、$A_2B_2$长度均为$len$，过点$O$向线段$A_2B_2$作垂线，垂足为点$D$，得到$\triangle OA_1D$，且$|O_1O_2|=|A_1O_1|+|A_2O_2|$恒成立，可在该三角形内解得：

$$len=2\cdot\sqrt{(4.5)^2-(\frac{S}{2})^2} \tag{19}$$

在$\triangle O_1O_2H$中，根据勾股定理易知：

$$|O_1O_2|=\sqrt{(S)^2+(len-r-R)^2} \tag{20}$$

联立式（19）、（20）解得：

$$R+r=\left|O_1O_2\right|=\frac{(4.5)^2}{\sqrt{(4.5)^2-\frac{S^2}{4}}} \tag{21}$$

由式（21）易知，$(R+r)$ 为定值，从而根据公式（16）推得 $\alpha$ 为定值，带入公式（17），**即可解得调头曲线长度 $L$ 恒定，不存在更短的调头曲线设计。**

## 8.2 把手位置与速度的建模与求解

本问题干已经给出“调头曲线上前一段圆弧的半径 $R$ 是后一段 $r$ 的 2 倍”，结合式（21）可以解得：

$$\begin{cases} R=\dfrac{2\cdot(4.5)^2}{3\cdot\sqrt{(4.5)^2-\dfrac{S^2}{4}}} \\ r=\dfrac{(4.5)^2}{3\cdot\sqrt{(4.5)^2-\dfrac{S^2}{4}}} \end{cases} \tag{22}$$

其中 $R$ 、 $r$ 为图 13 中所示相应变量。

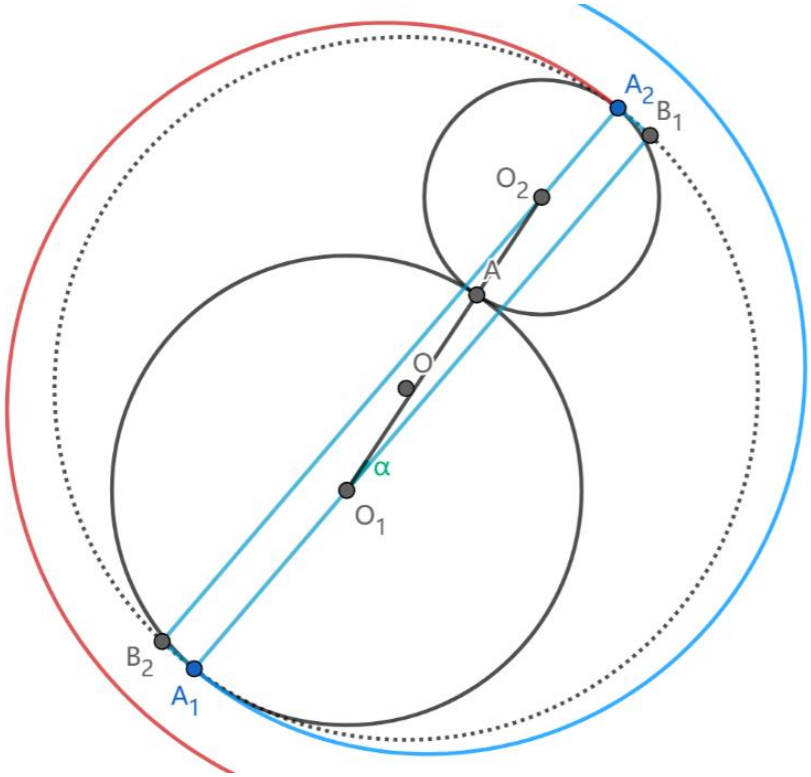



**图 14 高精度位置示意图**

**【基于图 14 进行相关求解方法说明】**

本问沿用前文求解中借助的极角$\theta$，能够通过程序计算得到盘入点$A_1$、盘出点$A_2$的极角$\theta_1$、$\theta_2$，且存在关系$\theta_2 = -\theta_1$（$\because$关于螺线中心对称）。为了满足“二分法”求解的必要条件——单调性，需要将两路径圆的切点$A$处的极角设定为 0，进一步将$(\theta_1, 0)$与$(0, \theta_2)$这两段角度区间内的极角值分别分配给弧线$A_1A$与$AA_2$上的点，即给定某把手点处极角$\theta$，可以通过旋转变换来计算出相应的圆弧坐标：

$$x = x_{\text{center}} + r \cdot \cos(\alpha \cdot \frac{\theta}{\theta_{\max}}) \tag{23}$$

$$y = y_{\text{center}} + r \cdot \sin(\alpha \cdot \frac{\theta}{\theta_{\max}}) \tag{24}$$

其中$(x_{center}, y_{center})$为路径圆圆心，$\alpha$为某段圆弧总的旋转角，$\theta_{\max}$为极角的上限。通过式（23）、（24）可以将极角$\theta$转换为舞龙队在圆弧调头区域的平面坐标计算距离。

处理完上述极角分配后，问题四的位置与速度求解便可简化为问题一中的求解，能够借助问题一中建立的位置、速度模型进行最终的数值求解计算。

### 8.2.1 位置的求解

通过递推的方式计算板凳龙上每个把手的位置：

对于每个新的板凳节，其前把手位置即为前一板凳的后把手位置，直到生成所有把手的位置。在生成每个把手点时，均使用二分法搜索，使得两相邻板凳的把手点间距离等于板凳的长度。数学表示如下：

$$\text{distance}(point_a, point_b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{25}$$

其中，$point_a$和$point_b$分别为两个把手点的坐标，通过控制相邻两点间的距离来生成下一把手点的坐标。

### 8.2.2 速度的求解

首先计算相邻两节板凳在时间微元中的位置变化量，进而能够计算各出各把手点的速度大小。

令$s_i$为第$i$节板凳在时间$\Delta t$内的位移，速度计算公式为：

$$v_i = \frac{s_i}{\Delta t} \tag{26}$$

其中，$\Delta t$通过调整时间步长控制误差，使得速度计算尽可能精确。

由于所有点的$\Delta t$相同，所以可以将速度求解简化为与问题一中一样的比例关系。

## 8.3 结果展示与可视化

程序计算得：**最短调头路径长度为 13.621244906820971 $m$ 。**

题目要求以调头开始时间为零时刻，给出从−100 s 开始到 100 s 为止，每秒舞龙队各把手点的位置和速度。现已将相应结果存放到文件 result4.xlsx 中。

### 8.3.1 部分结果展示

**表 4 部分位置坐标结果**

| | -100 s | -50 s | 0 s | 50 s | 100 s |
|---|---|---|---|---|---|
| 龙头 x (m) | 7.778033587 | 6.608300812 | -2.711855864 | 1.332695842 | -3.157228766 |
| 龙头 y (m) | 3.717164162 | 1.898864648 | -3.591077523 | 6.175323746 | 7.548511231 |
| 第 1 节龙身 x (m) | 6.209272664 | 5.366911313 | -0.063533847 | 3.862265026 | -0.346889766 |
| 第 1 节龙身 y (m) | 6.108521342 | 4.475403433 | -4.670887922 | 4.840827682 | 8.079166211 |
| 第 51 节龙身 x (m) | -10.60803788 | -3.629944995 | 2.459962156 | -1.67138482 | 2.095033158 |
| 第 51 节龙身 y (m) | 2.831491266 | -8.963799845 | -7.77814506 | -6.076713001 | 4.033787146 |
| 第 101 节龙身 x (m) | -11.92276108 | 10.12578705 | 3.008493276 | -7.591815738 | -7.288774194 |
| 第 101 节龙身 y (m) | -4.802378316 | -5.972246515 | 10.10853913 | 5.175486731 | 2.063874897 |
| 第 151 节龙身 x (m) | -14.35103222 | 12.97478364 | -7.002788579 | -4.605164593 | 9.462513461 |
| 第 151 节龙身 y (m) | -1.980993344 | -3.81035721 | 10.33748222 | -10.38698844 | -3.540356771 |
| 第 201 节龙身 x (m) | -11.95294238 | 10.52250858 | -6.872841753 | 0.336952219 | 8.524374193 |
| 第 201 节龙身 y (m) | 10.56699779 | -10.80742496 | 12.38260876 | -13.17760957 | 8.606932612 |
| 龙尾（后）x (m) | -1.011058616 | 0.189809149 | -1.933627477 | 5.85909427 | -10.98015715 |
| 龙尾（后）y (m) | -16.52757274 | 15.72058757 | -14.7131281 | 12.61289424 | -6.770006019 |

**表 5 部分速度结果**

| | -100 s | -50 s | 0 s | 50 s | 100 s |
|---|---|---|---|---|---|
| 龙头 (m/s) | 1 | 1 | 1 | 1 | 1 |
| 第 1 节龙身 (m/s) | 0.999904005 | 0.999762306 | 0.998686476 | 1.0003628 | 1.000123803 |
| 第 51 节龙身 (m/s) | 0.999342363 | 0.99864215 | 0.995134497 | 0.949934308 | 1.003967811 |
| 第 101 节龙身 (m/s) | 0.999084798 | 0.998251015 | 0.994451176 | 0.948483993 | 1.096264848 |
| 第 151 节龙身 (m/s) | 0.998939513 | 0.998046152 | 0.994156746 | 0.948040957 | 1.095311918 |
| 第 201 节龙身 (m/s) | 0.998843741 | 0.997925103 | 0.993995663 | 0.947828533 | 1.094941159 |
| 龙尾（后）(m/s) | 0.998809737 | 0.997886447 | 0.993945966 | 0.947766491 | 1.094842377 |

### 8.3.2 结果可视化

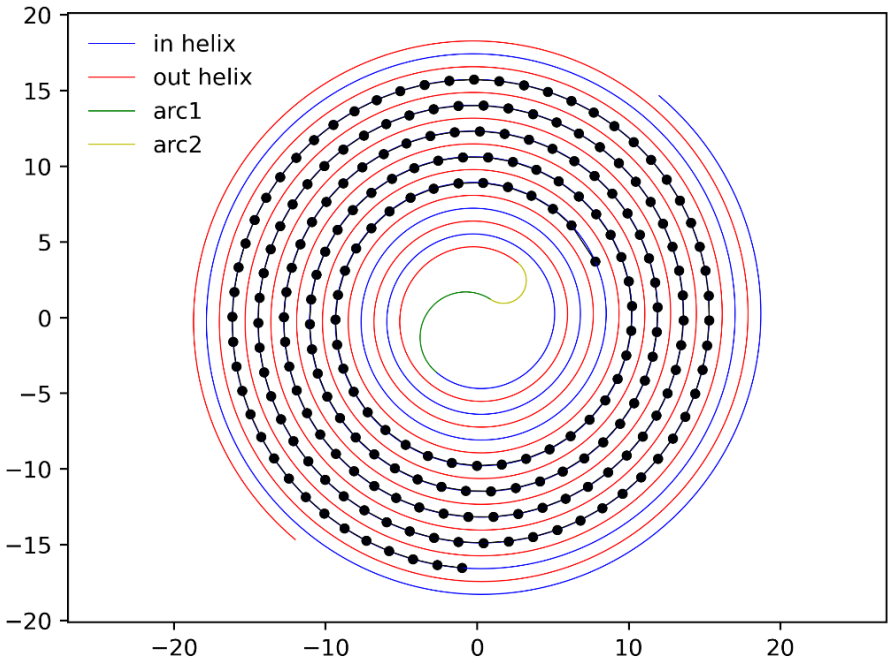



**图 15** $t=-100s$

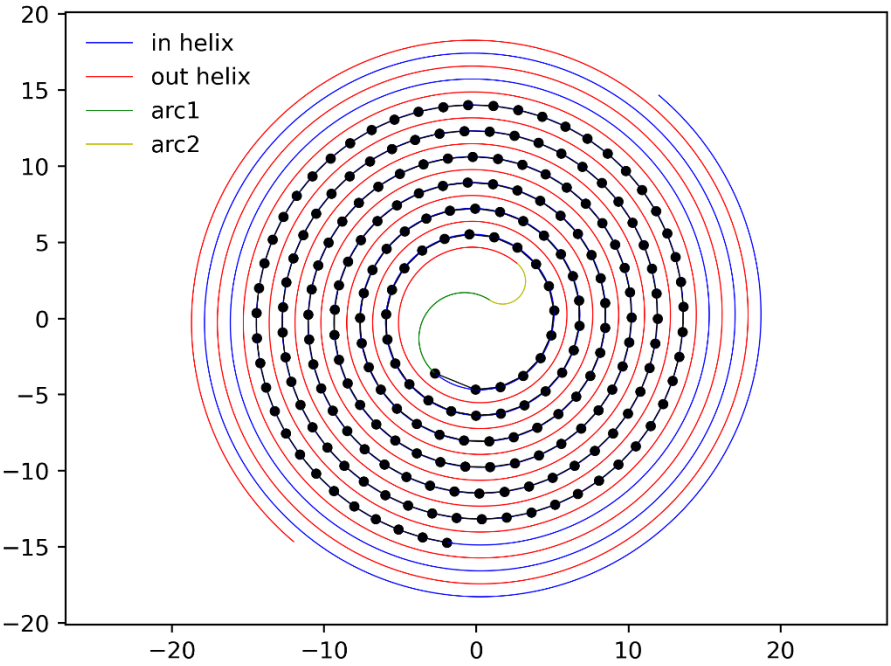



**图 16** $t=0s$

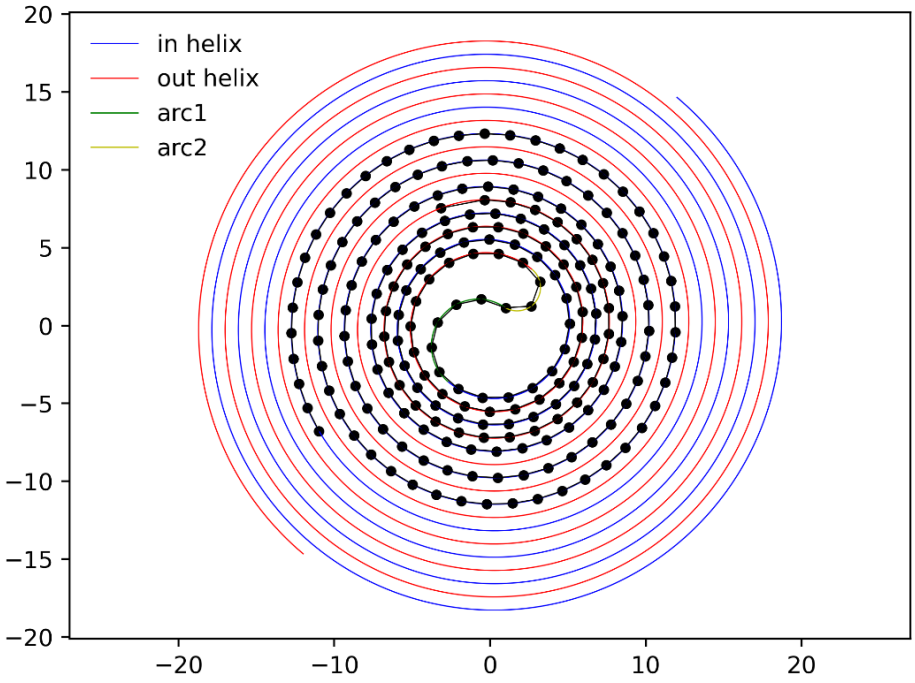



**图 17** $t=100s$

图 15、16、17 中，如图例所示，蓝色弧线表示 $in\_helix$，即“盘入螺线”；红色弧线表示 $out\_helix$，即“盘出螺线”；绿色圆弧表示第一段调头路径弧（属于半径为 $R$ 的路径圆）；黄色圆弧表示第二段调头路径弧（属于半径为 $r$ 的路径圆）。

# 九、问题五模型的建立与求解

根据问题五的要求，需要计算龙头的最大行进速度，确保舞龙队中每个板凳的把手点的速度均不超过 $2\,m/s$。本问模型需要根据路径调整速度，使得龙头的行进速度不会超过此限制。

## 9.1 最大行进速度的模型建立于求解

本问可以采取最简单的做法——固定龙头速度为 $1\,m/s$，使龙头沿着某一路径不断运动，同时求解并记录每个把手点在每个时刻的速度大小。运行完毕后，只需查找记录的最大速度。

问题一中已经证明（见 5.2），螺线上任意把手点的速度 $\propto$ 位移，可知无论速度如何改变，任意两把手点的速度比必为定值。

## 9.2 结果展示

程序计算得：当龙头保持 $1\,m/s$ 运动时，最大把手点速度为 1.6047758042912914 $m/s$。由于速度成比例关系，即可解得龙头最大行进速度为（2/1.6047758042912914）$m/s$，**即 1.246280006622638** $m/s$。

# 十、模型的评价与推广

## 10.1 模型的优点

1、**高精确性：**模型通过二分思想进行递归计算，能够以高精度逼近每个板凳在螺线轨迹上的位置和运动轨迹。通过递推，能够确保相邻板凳之间的距离符合题目给定的板凳长度约束。这种递推过程结合二分法保证了计算的精确度。

2、**有效的碰撞检测：**模型利用分离轴定律进行碰撞检测。对于每一张板凳，都将其抽象为一个矩形，分离轴定理能够判断两个凸多边形（本文为矩形）在运动过程中是否相交，从而避免了板凳在螺旋路径上发生碰撞。这种方法不仅有效，而且计算复杂度较低，能够适用于大规模的多板凳系统。

3、**几何建模直观、适用性广：**螺旋线和圆弧调头区域的几何建模简单清晰。模型基于极坐标和笛卡尔坐标的转换，能够直观地描述舞龙队的整体路径，这种几何建模方式具有普遍性，可推广应用于其他类似的轨迹规划问题，如路径优化等。

4、**高效的速度优化策略：**通过二分法进行龙头速度的优化，确保过程中模型所有单元符合要求。计算复杂度低，且快速收敛，能够高效地求出最优解。

## 10.2 模型的缺点

1、**忽略了部分物理因素：**模型没有考虑到板凳之间的物理作用，如摩擦力、惯性等因素。在实际应用中，考虑这些力的影响会使模型更加复杂，但也会更加真实。

2、**忽视了板凳与表演者的交互：**模型中忽略了表演者的体积和运动，假设板凳完全沿螺旋线运动。然而，实际表演过程中，表演者的行走路径、速度、动作协调等因素可能会影响板凳的运动。忽略这些交互因素可能导致模型与实际表演的差距。

## 10.1 模型的推广与改进

1、针对本文提出的模型，对螺线轨道、板块队列等加以更换，能够运用于更多的动态场景，如交通轨道、机器人路径规划等。

# 十一、参考文献


[1] 李玓.福清板凳龙非物质文化遗产的保护与传承研究[D].华中师范大学,2022.DOI:10.27159/d.cnki.ghzsu.2022.002708.

[2] 刘娜,毛晓菊.基于分离轴定理的碰撞检测算法[J].数字技术与应用,2012,(08):102.DOI:10.19695/j.cnki.cn12-1369.2012.08.070.

[3] 刘永昌.基于分离轴定理的堆场防碰撞系统设计[J].科技传播,2015,7(19):131-132.DOI:10.16607/j.cnki.1674-6708.2015.19.082.

[4] 张应中,范超,罗晓芳.凸多面体连续碰撞检测的运动轨迹分离轴算法[J].计算机辅助设计与图形学学报,2013,25(01):7-14.

[5] 张晓勇,王仲君.二分法和牛顿迭代法求解非线性方程的比较及应用[J].教育教学论坛,2013,(25):139.

[6] 何天荣.非线性方程的数值解法中的二分法[J].科技风,2016,(13):28.DOI:10.19392/j.cnki.1671-7341.201613027.

[7] 肖息桂,唐生福,石宝.“用二分法求方程的近似解”的教学思考[J].桂林师范高等专科学校学报,2008,(03):157-159.

[8] 苏金源.二分法逼近解题的数学思想方法[J].中国科技信息,2005,(22):70+85.

# 附录

## 附录一 支撑文件列表

文件一　　工作簿：result1.xlsx
　　工作表 1：位置
　　工作表 2：速度
文件二　　工作簿：result2.xlsx
文件三　　工作簿：result4.xlsx
　　工作表 1：位置
　　工作表 2：速度
文件四　　程序一（自定义 helix 类）Python 代码文件：helix.py
文件五　　程序二（自定义 sat 类）Python 代码文件：sat.py
文件六　　程序三（第一问）Python 代码文件：Q1.py
文件七　　程序四（第二问）Python 代码文件：Q2.py
文件八　　程序五（第三问）Python 代码文件：Q3.py
文件九　　程序六（第四问）Python 代码文件：Q4.py
文件十　　程序七（第五问）Python 代码文件：Q5.py
文件十一　程序输出的可视化图像集

## 附录二 封装 helix 类

```
import numpy as np
from scipy.integrate import quad

NUMS = 223  # 板凳数量
LEN1 = 2.86  # 龙头长度
LEN2 = 1.65  # 龙身和龙尾长度
EPS = 1e-12  # 二分精度
OFFSET = 0.0001  # 计算速度时的偏移量


# 坐标转换函数（极坐标到笛卡尔坐标）
def polar_to_cartesian(polar_point):
    return [polar_point[0] * np.cos(polar_point[1]), polar_point[0] * np.sin(polar_point[1])]


# 欧几里得距离计算函数
def distance(point_a, point_b):
    x1, y1 = point_a
    x2, y2 = point_b
```

```
20.     return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5
21.
22.
23. # 极坐标距离计算函数（转换为笛卡尔坐标后计算距离）
24. def polar_distance(polar_point_a, polar_point_b):
25.     x1, y1 = polar_to_cartesian(polar_point_a)
26.     x2, y2 = polar_to_cartesian(polar_point_b)
27.     return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5
28.
29.
30. # 龙头最小极角计算函数（若不满足则不能够在该极角处放置龙头）（连接螺线中心与龙头前把手）
31. def min_head_theta(pitch):
32.     return LEN1 * np.pi / pitch - np.pi / 2 # 龙头最小极角
33.
34.
35. class Helix:
36.     def __init__(self, pitch, head_theta=None, head_speed=None):
37.         # r = pitch / (2 * pi) * θ
38.         self.pitch = pitch
39.
40.         if head_theta is not None:
41.             self.head_theta = head_theta
42.             self.generate_points()
43.
44.         if head_speed is not None:
45.             self.head_speed = head_speed
46.             self.generate_offset_points()
47.             self.speed = []
48.             standard = distance(self.cartesian_points[0], self.offset_cartesian_points[0])
49.             for i in range(NUMS + 1):
50.                 self.speed.append(distance(self.cartesian_points[i], self.offset_cartesian_points[i]) / standar
d * head_speed)
51.
52.     def r(self, theta):
53.         return self.pitch / (2 * np.pi) * theta
54.
55.     # 生成 224 个点的极坐标列表和笛卡尔坐标列表
56.     def generate_points(self):
57.         def generate_next_point(self, distance):
58.             l_theta = self.polar_points[-1][1]
59.             r_theta = l_theta + np.pi
60.             while l_theta + EPS < r_theta:
61.                 m_theta = (l_theta + r_theta) / 2
62.                 if polar_distance(self.polar_points[-1], [self.r(m_theta), m_theta]) < distance:
```

```
                l_theta = m_theta
            else:
                r_theta = m_theta
        self.polar_points.append([self.r(l_theta), l_theta])

    self.polar_points = [[self.r(self.head_theta), self.head_theta]]
    generate_next_point(self, LEN1)
    for _ in range(NUMS - 1):
        generate_next_point(self, LEN2)
    self.cartesian_points = np.array([[r * np.cos(theta), r * np.sin(theta)] for r, theta in self.polar_points])

# 生成 224 个点的偏移坐标列表和偏移坐标列表
def generate_offset_points(self):
    def generate_next_offset_point(distance):
        l_theta = self.offset_polar_points[-1][1]
        r_theta = l_theta + np.pi
        while l_theta + EPS < r_theta:
            m_theta = (l_theta + r_theta) / 2
            if polar_distance(self.offset_polar_points[-1], [self.r(m_theta), m_theta]) < distance:
                l_theta = m_theta
            else:
                r_theta = m_theta
        self.offset_polar_points.append([self.r(l_theta), l_theta])

    self.offset_polar_points = [[self.r(self.head_theta + OFFSET), self.head_theta + OFFSET]]
    generate_next_offset_point(LEN1)
    for _ in range(NUMS - 1):
        generate_next_offset_point(LEN2)
    self.offset_cartesian_points = np.array([[r * np.cos(theta), r * np.sin(theta)] for r, theta in self.offset_polar_points])

# 积分计算指定范围内的弧长（用于得到龙头前把手位置）
def length(self, start, end):
    # 螺线的方程式：r = pitch / (2 * pi) * θ
    def integrand(theta):
        r = self.r(theta)
        dr_dtheta = self.pitch / (2 * np.pi)
        return np.sqrt(dr_dtheta**2 + r**2)

    # 积分计算弧长
    arc_length, _ = quad(integrand, start, end)
    return arc_length

# 绘制图形
```

```
    def plot(self, filename):
        import matplotlib.pyplot as plt

        plt.figure(dpi=400)
        plt.axis("equal")

        # 绘制所有点
        plt.plot(self.cartesian_points[:, 0], self.cartesian_points[:, 1], color="black", linewidth=0.5)
        plt.scatter(self.cartesian_points[:, 0], self.cartesian_points[:, 1], color="black", s=12)

        # 绘制 r = pitch / 2pi * θ
        thetas = np.linspace(0, self.polar_points[-1][1], 1000)
        rs = self.r(thetas)
        x = rs * np.cos(thetas)
        y = rs * np.sin(thetas)
        plt.plot(x, y, color="blue", linestyle="--", linewidth=0.5)

        plt.savefig(filename)


if __name__ == "__main__":
    helix = Helix(0.55, 5 * np.pi, 1)
    helix.plot("helix.png")
    print(helix.length(np.pi * 10, np.pi * 25))
```

## 附录三 封装 sat 类

```
WIDTH_A = 0.275
WIDTH_B = 0.15


class SAT:
    class Rectangle:
        def __init__(self, point_a, point_b):
            x1, y1 = point_a
            x2, y2 = point_b

            self.axes = [] # 获取矩形分离轴
            self.axes.append((x2 - x1, y2 - y1))
            self.axes.append((y2 - y1, x1 - x2))

            self.points = [] # 获取矩形顶点
            dist = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
            vecx = (x2 - x1) / dist
            vecy = (y2 - y1) / dist
```

```
            self.points.append((-WIDTH_A * vecx + WIDTH_B * vecy + x1, -WIDTH_A * vecy - WIDTH_B * vecx + y1))
            self.points.append((-WIDTH_A * vecx - WIDTH_B * vecy + x1, -WIDTH_A * vecy + WIDTH_B * vecx + y1))
            self.points.append((WIDTH_A * vecx - WIDTH_B * vecy + x2, WIDTH_A * vecy + WIDTH_B * vecx + y2))
            self.points.append((WIDTH_A * vecx + WIDTH_B * vecy + x2, WIDTH_A * vecy - WIDTH_B * vecx + y2))

    def __init__(self, points):
        self.points = points
        self.rectangles = []
        for i in range(len(points) - 1):
            self.rectangles.append(self.Rectangle(points[i], points[i + 1]))

    def is_intersecting(self, rectangle_a, rectangle_b):
        def project(rectangle, axis):
            # 投影函数
            min_proj = float("inf")
            max_proj = float("-inf")
            for point in rectangle.points:
                projection = point[0] * axis[0] + point[1] * axis[1]
                min_proj = min(min_proj, projection)
                max_proj = max(max_proj, projection)
            return min_proj, max_proj

        def overlap(proj_a, proj_b):
            # 判断投影是否重叠
            return not (proj_a[1] < proj_b[0] or proj_b[1] < proj_a[0])

        axes = rectangle_a.axes + rectangle_b.axes
        for axis in axes:
            proj_a = project(rectangle_a, axis)
            proj_b = project(rectangle_b, axis)
            if not overlap(proj_a, proj_b):
                return False  # 有分离，矩形不相交
        return True  # 所有轴上的投影都有重叠，矩形相交

    def exist_intersecting(self):
        # 判断不相邻矩形是否相交
        for i in range(len(self.rectangles) - 2):
            for j in range(i + 2, len(self.rectangles)):
                if self.is_intersecting(self.rectangles[i], self.rectangles[j]):
                    return True
        return False
```

```
60.
61.     def plot(self, filename, include_index=False):
62.         import matplotlib.pyplot as plt
63.
64.         plt.figure(dpi=400)
65.         plt.axis("equal")
66.
67.         for idx, rectangle in enumerate(self.rectangles):
68.             # 绘制矩形
69.             x = [point[0] for point in rectangle.points + [rectangle.points[0]]]
70.             y = [point[1] for point in rectangle.points + [rectangle.points[0]]]
71.             plt.plot(x, y, color="black", linewidth=0.3)
72.
73.             if include_index:
74.                 # 标注矩形序号
75.                 center_x = sum(point[0] for point in rectangle.points) / len(rectangle.points)
76.                 center_y = sum(point[1] for point in rectangle.points) / len(rectangle.points)
77.                 plt.text(center_x, center_y, str(idx + 1), color="blue", fontsize=3, ha="center", va="center")
78.
79.         plt.savefig(filename)
```

## 附录四 问题一的求解代码

```
1.  import numpy as np
2.  from helix import *
3.  import openpyxl
4.
5.  if __name__ == "__main__":
6.      base = Helix(0.55)
7.
8.      wb = openpyxl.load_workbook("result1.xlsx")
9.
10.     temp = Helix(0.55, 32 * np.pi, 1)
11.     for i in range(len(temp.cartesian_points)):
12.         wb["位置"].cell(row=i * 2 + 2, column=2, value=temp.cartesian_points[i][0])
13.         wb["位置"].cell(row=i * 2 + 3, column=2, value=temp.cartesian_points[i][1])
14.         wb["速度"].cell(row=i + 2, column=2, value=temp.speed[i])
15.
16.     for t in range(1, 301):
17.         l_theta = min_head_theta(0.55)
18.         r_theta = 32 * np.pi
19.         while l_theta + EPS < r_theta:
20.             m_theta = (l_theta + r_theta) / 2
21.             if base.length(m_theta, 32 * np.pi) > t:
22.                 l_theta = m_theta
```

```
            else:
                r_theta = m_theta

        temp = Helix(0.55, l_theta, 1)
        for i in range(len(temp.cartesian_points)):
            wb["位置"].cell(row=i * 2 + 2, column=t + 2, value=temp.cartesian_points[i][0])
            wb["位置"].cell(row=i * 2 + 3, column=t + 2, value=temp.cartesian_points[i][1])
            wb["速度"].cell(row=i + 2, column=t + 2, value=temp.speed[i])

    wb.save("result1_new.xlsx")
```

## 附录五 问题二的求解代码

```
import numpy as np
from helix import *
from sat import SAT
import openpyxl


if __name__ == "__main__":
    l_head_theta = min_head_theta(0.55)
    r_head_theta = 32 * np.pi

    while l_head_theta + EPS < r_head_theta:
        m_head_theta = (l_head_theta + r_head_theta) / 2
        helix = Helix(0.55, m_head_theta)
        points = helix.cartesian_points
        sat = SAT(points)
        if sat.exist_intersecting():
            l_head_theta = m_head_theta
        else:
            r_head_theta = m_head_theta

    helix = Helix(0.55, l_head_theta, 1.0)

    print(f"碰撞时的极角：{l_head_theta}")
    print(f"已经行进的距离：{helix.length(l_head_theta, 32 * np.pi)} m")

    wb = openpyxl.load_workbook("result2.xlsx")
    for i in range(len(helix.cartesian_points)):
        wb["Sheet1"].cell(row=i + 2, column=2, value=helix.cartesian_points[i][0])
        wb["Sheet1"].cell(row=i + 2, column=3, value=helix.cartesian_points[i][1])
        wb["Sheet1"].cell(row=i + 2, column=4, value=helix.speed[i])
    wb.save("result2_new.xlsx")

```

```
    helix.plot("Q2_1.png")
    sat = SAT(helix.cartesian_points)
    sat.plot("Q2_2.png", include_index=True)
```

## 附录六 问题三的求解代码

```
import numpy as np
from helix import *
from sat import *

DIAMETER = 9.0 # 调头空间直径


if __name__ == "__main__":
    l_pitch = 0.1
    r_pitch = 2.0

    while l_pitch + EPS < r_pitch:
        m_pitch = (l_pitch + r_pitch) / 2
        head_theta = 9 * np.pi / m_pitch
        helix = Helix(m_pitch, head_theta)
        points = helix.cartesian_points
        sat = SAT(points)
        if sat.exist_intersecting():
            l_pitch = m_pitch
        else:
            r_pitch = m_pitch

    print(f"最小螺距：{l_pitch} m")

    head_theta = DIAMETER * np.pi / l_pitch
    helix = Helix(l_pitch, head_theta, 1.0)
    helix.plot("Q3_1.png")
    sat = SAT(helix.cartesian_points)
    sat.plot("Q3_2.png", include_index=True)
```

## 附录七 问题四的求解代码

```
import numpy as np
from scipy.integrate import quad
import openpyxl

NUMS = 223  # 板凳数量
LEN1 = 2.86  # 龙头长度
```

```
LEN2 = 1.65  # 龙身和龙尾长度
EPS = 1e-12  # 二分精度
DIAMETER = 9.0  # 调头空间直径
PITCH = 1.7  # 螺距
# 盘入螺线 r = PITCH / (2 * pi) * theta
# 盘出螺线 r = -PITCH / (2 * pi) * theta
LIM = DIAMETER * np.pi / PITCH  # 极角临界值
ALPHA = 3.0214868419806535  # 圆弧圆心角上限
P_X = 0.9039519545689  # 两圆弧切点
P_Y = 1.1970258409204
R1 = 3.005417667789  # 圆弧1半径
O1_X = -0.7600091166555  # 圆弧1圆心
O1_Y = -1.3057264263463
R2 = 1.5027088338945  # 圆弧2半径
O2_X = 1.7359324901811  # 圆弧2圆心
O2_Y = 2.4484019745537


def polar_to_cartesian(polar_point):
    return [polar_point[0] * np.cos(polar_point[1]), polar_point[0] * np.sin(polar_point[1])]


def distance(point_a, point_b):
    x1, y1 = point_a
    x2, y2 = point_b
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5


def get_point(theta):  # 返回theta对应的坐标
    if theta >= LIM:
        polar_point = [PITCH / (2 * np.pi) * theta, theta]
        return polar_to_cartesian(polar_point)
    elif theta <= -LIM:
        polar_point = [PITCH / (2 * np.pi) * theta, -theta]
        return polar_to_cartesian(polar_point)
    elif theta >= 0:
        dx = P_X - O1_X
        dy = P_Y - O1_Y
        angle = ALPHA * theta / LIM
        # 将向量dx dy逆时针旋转angle
        x = dx * np.cos(angle) - dy * np.sin(angle) + O1_X
        y = dx * np.sin(angle) + dy * np.cos(angle) + O1_Y
        return [x, y]
```

```
    else:
        dx = P_X - O2_X
        dy = P_Y - O2_Y
        angle = -ALPHA * theta / LIM
        # 将向量 dx dy 逆时针旋转 angle
        x = dx * np.cos(angle) - dy * np.sin(angle) + O2_X
        y = dx * np.sin(angle) + dy * np.cos(angle) + O2_Y
        return [x, y]


def length(start, end):
    def integrand(theta):
        r = PITCH / (2 * np.pi) * theta
        dr_dtheta = PITCH / (2 * np.pi)
        return np.sqrt(dr_dtheta**2 + r**2)

    # 分段求长度
    res = 0
    if end >= LIM:
        l = max(start, LIM)
        r = end
        res += quad(integrand, l, r)[0]
    if start <= -LIM:
        l = abs(min(end, -LIM))
        r = abs(start)
        res += quad(integrand, l, r)[0]
    if end >= 0 and start < LIM:
        l = max(start, 0)
        r = min(end, LIM)
        res += R1 * ALPHA * (r - l) / LIM
    if end > -LIM and start < 0:
        l = max(start, -LIM)
        r = min(end, 0)
        res += R2 * ALPHA * (r - l) / LIM
    return res


class Helix:
    def __init__(self, head_theta=None):
        if head_theta is not None:
            self.head_theta = head_theta
            self.generate_points()

    def generate_points(self):
        def generate_next_point(dist):
```

```
95.             l_theta = self.theta_list[-1]
96.             r_theta = l_theta + np.pi
97.             if -LIM <= r_theta <= LIM:
98.                 r_theta += 4 * np.pi
99.                 r_theta = min(r_theta, LIM + 0.8 * np.pi)
100.                while l_theta + EPS < r_theta:
101.                    m_theta = (l_theta + r_theta) / 2
102.                    if distance(self.points[-
1], get_point(m_theta)) < dist:
103.                        l_theta = m_theta
104.                    else:
105.                        r_theta = m_theta
106.                self.theta_list.append(l_theta)
107.                self.points.append(get_point(l_theta))
108.
109.            self.theta_list = [self.head_theta]
110.            self.points = [get_point(self.head_theta)]
111.            generate_next_point(LEN1)
112.            for _ in range(NUMS - 1):
113.                generate_next_point(LEN2)
114.
115.        def plot(self, filename):
116.            import matplotlib.pyplot as plt
117.
118.            plt.figure(dpi=400)
119.            plt.axis("equal")
120.
121.            # 绘制路径
122.            helix_points = np.linspace(-70, 70, 10000)
123.            x_in = []
124.            y_in = []
125.            x_out = []
126.            y_out = []
127.            x_arc0 = []
128.            y_arc0 = []
129.            x_arc1 = []
130.            y_arc1 = []
131.            for point in helix_points:
132.                x, y = get_point(point)
133.                if point >= LIM:
134.                    x_in.append(x)
135.                    y_in.append(y)
136.                elif point <= -LIM:
137.                    x_out.append(x)
138.                    y_out.append(y)
```

```
139.                 elif point >= 0:
140.                     x_arc0.append(x)
141.                     y_arc0.append(y)
142.                 else:
143.                     x_arc1.append(x)
144.                     y_arc1.append(y)
145.             plt.plot(x_in, y_in, label="in helix", color="b", linewidth=0.5
)
146.             plt.plot(x_out, y_out, label="out helix", color="r", linewidth=
0.5)
147.             plt.plot(x_arc0, y_arc0, label="arc1", color="g", linewidth=0.5
)
148.             plt.plot(x_arc1, y_arc1, label="arc2", color="y", linewidth=0.5
)
149.
150.             self.points = np.array(self.points)
151.             plt.plot(self.points[:, 0], self.points[:, 1], color="black", l
inewidth=0.5)
152.             plt.scatter(self.points[:, 0], self.points[:, 1], color="black"
, s=12)
153.             plt.legend(frameon=False)
154.
155.             plt.savefig(filename)
156.
157.
158.     def generate_speed(helix, head_speed):
159.         l = helix.head_theta
160.         r = l + 0.1
161.         while l + EPS < r:
162.             m = (l + r) / 2
163.             if length(helix.head_theta, m) > 0.00001:
164.                 r = m
165.             else:
166.                 l = m
167.         offset_helxi = Helix(l)
168.         standard = distance(helix.points[0], offset_helxi.points[0])
169.         res = []
170.         for i in range(NUMS + 1):
171.             res.append(distance(helix.points[i], offset_helxi.points[i]) /
standard * head_speed)
172.         return res
173.
174.
175.     if __name__ == "__main__":
176.
```

```
177.         print(f"最短掉头路径长度：{length(-LIM, LIM)} m")
178.
179.         wb = openpyxl.load_workbook("result4.xlsx")
180.
181.         temp = Helix(LIM)
182.         temp.plot("Q4_0.png")
183.         speed = generate_speed(temp, 1)
184.         for i in range(len(temp.points)):
185.             wb["位置
"].cell(row=i * 2 + 2, column=102, value=temp.points[i][0])
186.             wb["位置
"].cell(row=i * 2 + 3, column=102, value=temp.points[i][1])
187.             wb["速度"].cell(row=i + 2, column=102, value=speed[i])
188.
189.         for t in range(1, 101):
190.             l = LIM
191.             r = 32 * np.pi
192.             while l + EPS < r:
193.                 m = (l + r) / 2
194.                 if length(LIM, m) > t:
195.                     r = m
196.                 else:
197.                     l = m
198.
199.             temp = Helix(l)
200.             if t == 100:
201.                 temp.plot("Q4_-100.png")
202.             speed = generate_speed(temp, 1)
203.             for j in range(len(temp.points)):
204.                 wb["位置
"].cell(row=j * 2 + 2, column=102 - t, value=temp.points[j][0])
205.                 wb["位置
"].cell(row=j * 2 + 3, column=102 - t, value=temp.points[j][1])
206.                 wb["速度"].cell(row=j + 2, column=102 - t, value=speed[j])
207.
208.         for t in range(1, 101):
209.             l = -32 * np.pi
210.             r = LIM
211.             while l + EPS < r:
212.                 m = (l + r) / 2
213.                 if length(m, LIM) > t:
214.                     l = m
215.                 else:
216.                     r = m
217.             temp = Helix(l)
```

```
            if t == 100:
                temp.plot("Q4_100.png")
            speed = generate_speed(temp, 1)
            for j in range(len(temp.points)):
                wb["位置
"].cell(row=j * 2 + 2, column=102 + t, value=temp.points[j][0])
                wb["位置
"].cell(row=j * 2 + 3, column=102 + t, value=temp.points[j][1])
                wb["速度"].cell(row=j + 2, column=102 + t, value=speed[j])

        wb.save("result4.xlsx")
```

## 附录八 问题五的求解代码

```
import numpy as np
from scipy.integrate import quad
import openpyxl

NUMS = 223  # 板凳数量
LEN1 = 2.86  # 龙头长度
LEN2 = 1.65  # 龙身和龙尾长度
EPS = 1e-12  # 二分精度
DIAMETER = 9.0  # 调头空间直径
PITCH = 1.7  # 螺距
# 盘入螺线 r = PITCH / (2 * pi) * theta
# 盘出螺线 r = -PITCH / (2 * pi) * theta
LIM = DIAMETER * np.pi / PITCH  # 极角临界值
ALPHA = 3.0214868419806535  # 圆弧圆心角上限
P_X = 0.9039519545689  # 两圆弧切点
P_Y = 1.1970258409204
R1 = 3.005417667789  # 圆弧 1 半径
O1_X = -0.7600091166555  # 圆弧 1 圆心
O1_Y = -1.3057264263463
R2 = 1.5027088338945  # 圆弧 2 半径
O2_X = 1.7359324901811  # 圆弧 2 圆心
O2_Y = 2.4484019745537


def polar_to_cartesian(polar_point):
    return [polar_point[0] * np.cos(polar_point[1]), polar_point[0] * np.s
in(polar_point[1])]


```

```
29. def distance(point_a, point_b):
30.     x1, y1 = point_a
31.     x2, y2 = point_b
32.     return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5
33.
34.
35. def get_point(theta):  # 返回theta对应的坐标
36.     if theta >= LIM:
37.         polar_point = [PITCH / (2 * np.pi) * theta, theta]
38.         return polar_to_cartesian(polar_point)
39.     elif theta <= -LIM:
40.         polar_point = [PITCH / (2 * np.pi) * theta, -theta]
41.         return polar_to_cartesian(polar_point)
42.     elif theta >= 0:
43.         dx = P_X - O1_X
44.         dy = P_Y - O1_Y
45.         angle = ALPHA * theta / LIM
46.         # 将向量dx dy逆时针旋转angle
47.         x = dx * np.cos(angle) - dy * np.sin(angle) + O1_X
48.         y = dx * np.sin(angle) + dy * np.cos(angle) + O1_Y
49.         return [x, y]
50.     else:
51.         dx = P_X - O2_X
52.         dy = P_Y - O2_Y
53.         angle = -ALPHA * theta / LIM
54.         # 将向量dx dy逆时针旋转angle
55.         x = dx * np.cos(angle) - dy * np.sin(angle) + O2_X
56.         y = dx * np.sin(angle) + dy * np.cos(angle) + O2_Y
57.         return [x, y]
58.
59.
60. def length(start, end):
61.     def integrand(theta):
62.         r = PITCH / (2 * np.pi) * theta
63.         dr_dtheta = PITCH / (2 * np.pi)
64.         return np.sqrt(dr_dtheta**2 + r**2)
65.
66.     # 分段求长度
67.     res = 0
68.     if end >= LIM:
69.         l = max(start, LIM)
70.         r = end
71.         res += quad(integrand, l, r)[0]
72.     if start <= -LIM:
73.         l = abs(min(end, -LIM))
```

```
74.         r = abs(start)
75.         res += quad(integrand, l, r)[0]
76.     if end >= 0 and start < LIM:
77.         l = max(start, 0)
78.         r = min(end, LIM)
79.         res += R1 * ALPHA * (r - l) / LIM
80.     if end > -LIM and start < 0:
81.         l = max(start, -LIM)
82.         r = min(end, 0)
83.         res += R2 * ALPHA * (r - l) / LIM
84.     return res
85.
86.
87. class Helix:
88.     def __init__(self, head_theta=None):
89.         if head_theta is not None:
90.             self.head_theta = head_theta
91.             self.generate_points()
92.
93.     def generate_points(self):
94.         def generate_next_point(dist):
95.             l_theta = self.theta_list[-1]
96.             r_theta = l_theta + np.pi
97.             if -LIM <= r_theta <= LIM:
98.                 r_theta += 4 * np.pi
99.                 r_theta = min(r_theta, LIM + 0.8 * np.pi)
100.                while l_theta + EPS < r_theta:
101.                    m_theta = (l_theta + r_theta) / 2
102.                    if distance(self.points[-
1], get_point(m_theta)) < dist:
103.                        l_theta = m_theta
104.                    else:
105.                        r_theta = m_theta
106.                self.theta_list.append(l_theta)
107.                self.points.append(get_point(l_theta))
108.
109.            self.theta_list = [self.head_theta]
110.            self.points = [get_point(self.head_theta)]
111.            generate_next_point(LEN1)
112.            for _ in range(NUMS - 1):
113.                generate_next_point(LEN2)
114.
115.        def plot(self, filename):
116.            import matplotlib.pyplot as plt
117.
```

```
118.             plt.figure(dpi=400)
119.             plt.axis("equal")
120.
121.             # 绘制路径
122.             helix_points = np.linspace(-70, 70, 10000)
123.             x_in = []
124.             y_in = []
125.             x_out = []
126.             y_out = []
127.             x_arc0 = []
128.             y_arc0 = []
129.             x_arc1 = []
130.             y_arc1 = []
131.             for point in helix_points:
132.                 x, y = get_point(point)
133.                 if point >= LIM:
134.                     x_in.append(x)
135.                     y_in.append(y)
136.                 elif point <= -LIM:
137.                     x_out.append(x)
138.                     y_out.append(y)
139.                 elif point >= 0:
140.                     x_arc0.append(x)
141.                     y_arc0.append(y)
142.                 else:
143.                     x_arc1.append(x)
144.                     y_arc1.append(y)
145.             plt.plot(x_in, y_in, label="in helix", color="b", linewidth=0.5
)
146.             plt.plot(x_out, y_out, label="out helix", color="r", linewidth=
0.5)
147.             plt.plot(x_arc0, y_arc0, label="arc1", color="g", linewidth=0.5
)
148.             plt.plot(x_arc1, y_arc1, label="arc2", color="y", linewidth=0.5
)
149.
150.             self.points = np.array(self.points)
151.             plt.plot(self.points[:, 0], self.points[:, 1], color="black", l
inewidth=0.5)
152.             plt.scatter(self.points[:, 0], self.points[:, 1], color="black"
, s=12)
153.             plt.legend(frameon=False)
154.
155.             plt.savefig(filename)
156.
```

```
157.
158.    def generate_speed(helix, head_speed):
159.        l = helix.head_theta
160.        r = l + 0.1
161.        while l + EPS < r:
162.            m = (l + r) / 2
163.            if length(helix.head_theta, m) > 0.00001:
164.                r = m
165.            else:
166.                l = m
167.        offset_helxi = Helix(l)
168.        standard = distance(helix.points[0], offset_helxi.points[0])
169.        res = []
170.        for i in range(NUMS + 1):
171.            res.append(distance(helix.points[i], offset_helxi.points[i]) /
standard * head_speed)
172.        return res
173.
174.
175.    if __name__ == "__main__":
176.        max_speed = 1
177.        for i in range(-500, 500):
178.            helix = Helix(-LIM + 0.01 * i)
179.            speeds = generate_speed(helix, 1)
180.            max_speed = max(max_speed, max(speeds))
181.
182.        for i in range(-500, 500):
183.            helix = Helix(i)
184.            speeds = generate_speed(helix, 1)
185.            max_speed = max(max_speed, max(speeds))
186.
187.        print(f"最大速度：{max_speed} m/s")
188.        print(f"换算龙头 1 最大速度：{2/max_speed} m/s")
```