

板凳龙盘绕路线的优化设计

摘要

板凳龙是浙闽地区的传统民俗文化活动，具有深厚的历史积淀和文化内涵。在板凳龙表演过程中，往往会受到各种条件限制。如何避免板凳碰撞、规划最优路径，让舞龙之美完全释放，对于保护传统文化具有重要意义，成为了我们亟待解决的问题。本论文通过数学建模方法，尝试为板凳龙活动提供科学方案。

针对问题一，我们构建了**把手位置情况依赖于运动时间的关系模型**。我们发现，极角能直接确定某点的直角坐标位置。首先，通过弧长这一桥梁，利用 matlab 内置 fsolve 函数以及螺线弧长公式，可以搭建起龙头前把手的极角依赖于时间的关系；接着，通过相邻把手距离恒定，以及等距螺线的轨迹方程，可以不断通过**数值迭代**推出后续把手的位置依赖于运动时间的关系；然后，根据速度的定义，计算给定时刻增加一个微小的时间变化量后的把手位置的变化量，利用**差分近似极限**，可以获得各把手沿等距螺旋线的线速度。最终我们得到了所有结果详见文件 **result1.xlsx**，部分呈现在后文中。

针对问题二，我们基于问题一的依赖关系模型构建了**板凳龙运动状态的几何仿真模型**，以及基于**叉乘法判定的二维几何图形的碰撞检测模型**。首先罗列足够长的时刻点数组，可以由问题一的结果直接获得各把手的位置情况；接着考虑板凳的实际长宽并通过相邻把手连接线段的平移和延长，实现了对板凳龙运动的几何模型仿真；然后，利用基于叉乘法的碰撞检测算法按时间顺序对图形中的线段逐一进行碰撞检测。最后我们得到了碰撞时刻为 **414s**，并获得了该时刻各个把手的位置与速度信息详见 **result2.xlsx**，以及可视化展示，详见后文与附录。

针对问题三，我们利用问题二的碰撞检测算法，构建了**二分法优化模型**对最小螺距进行查找，最终在**优化精度 0.002m** 下求出最小螺距为 **0.450625m**；接着利用碰撞检测算法对其附近值进行计算，可以验证所求最小螺距的可行性。

针对问题四，第一问是典型的**非线性规划问题**，目标是找到符合题意的最短的**S型路径**，我们给出该拼接曲线应满足自身内部约束条件，以及该曲线端点应满足的外部约束条件，通过 MATLAB 创建相应的优化问题，用 fmincon 函数求解，得出该路径的最短长度为 **13.621245m**，以及能准确描述该曲线的各个参数，详见后文。第二问求解方式与问题一的数值迭代和差分近似相似，我们根据组合曲线的参数特性精准地构建了**迭代方程切换模型**，使得基于距离约束的迭代方程能在光滑连接的分段曲线上连续运行，所有位置和速度信息详见 **result4.xlsx**，后文展示了其中部分信息。

针对问题五，在第四问中我们已经求得：在龙头行进速度保持为 **1m/s** 的条件下，行进过程中各个时刻各个把手的速度大小，由于**各个把手的速度大小与龙头速度大小线性相关**。故将各把手速度与龙头速度等比例放大，使得最大把手速度为 **2m/s**，最终得到龙头把手前进最大速度为 **1.405909 m/s**。

关键字：逐项递推 数值迭代 几何模型仿真 路径优化 可视化分析

一、问题重述

1.1 问题背景

板凳龙，作为闽浙地区重要的非物质文化遗产，承载着深厚的历史积淀与文化内涵。然而，传统表演形式往往受到场地和技术的制约，使得舞龙的美感难以完全释放。面对这一困境，正如习总书记所强调：“要把保护传承和开发利用有机结合起来，让中华优秀传统文化生生不息”，如何运用数学建模分析板凳龙的运动状态，规划盘龙路径以及盘龙速度，使得在舞龙队能够不发生碰撞，盘入盘出所需要的面积越小、行进速度越快，进而优化舞龙队在确定场地内展现最大程度美观的表演计划，成为了我们亟待解决的问题。

1.2 问题重述

板凳龙包括 223 节板凳，其中较长的龙首仅 1 节，余下是较短的 221 节龙身以及最后的 1 节龙尾。龙首、龙身和龙尾的宽度一致，而龙尾和龙身形状完全一致，不同之处仅在于龙尾的后把手只穿过龙尾自身一个板凳。舞龙队控制板凳龙的把手始终按照规定的曲线进行运动，其中龙首的位置和速度至关重要：随着龙首运动，各个把手随之运动，进而板凳上的所有点紧随对应把手的运动产生运动。现要求我们解决如下五个问题：

问题一：问题一给出了各个把手中心处于给定螺线的位置约束，龙头前把手的初始位置为第 16 圈 x 正半轴处，运动方向为向内盘入，以及沿给定螺线的运动速度 1 m/s。要求我们建立数学模型，计算从初始时刻到 300 s 每秒的一系列时刻下，所有把手中心的位置与速度，写入文件 result1.xlsx 中，同时将特定板凳的数据写入表一和表二呈现在本文中。

问题二：沿用问题一的条件，进一步要求我们计算板凳龙向内盘入多久后存在两个板凳相互碰撞，并给出此时所有把手中心的位置与速度，写入文件 result2.xlsx 中，同时将特定板凳的数据呈现在本文中。

问题三：问题三不再提供螺线的螺距，而是提供了一个以原点为圆心，直径为 9m 的圆为掉头空间。要求确定最小螺距，使得板凳龙的龙首可以向内盘入抵达给定掉头空间的边界，以便进入掉头空间进行掉头。

问题四：问题四重新规定了盘入螺线的螺距为 1.7m，规定了盘出螺线是盘入螺线的中心对称。要求设计一段落在掉头空间内的掉头路径，首先满足掉头路径曲线和掉头区域外的盘入螺线和盘出螺线是相切的（光滑连接的），而掉头路径自身是由两端圆弧组成，要求两段圆弧也是相切的（光滑连接的），同时使得靠近盘入口的圆弧半径是靠近盘出口的圆弧半径的两倍。在设计时希望掉头路径曲线最短。

在所设计出掉头曲线的基础上，以开始掉头为零时刻重新确立时间轴，给出掉头前 100 s 到开始掉头后 100 s 中每秒时刻所有把手中心的位置和速度，写入文件 result4.xlsx 中，同时将特定板凳的数据呈现在本文中。

问题五：在问题四的基础上，舞龙队沿着问题四所设计的路径运动。要求得到全时刻下，使得所有把手速度均不超过 2 m/s 的最大龙头把手沿曲线的前进速度。

二、问题分析

2.1 问题一分析

问题一给定了龙首前把手沿给定等距螺线的速度，为了方便计算，在极坐标系下考虑该问题。我们先假设出龙首前把手运动的总时间，结合给定的线速度，可以直接得到龙首前把手走过的曲线长度；接着从另一个方面再次计算曲线长度，已知极径是极角的函数，根据极坐标系下曲线长度的积分公式，将极角作为积分变量，可以得到曲线长度关于终点极角的值，进而得到终点极角。不难发现，螺线上某点对应的极角就能确定出该点的位置，于是可以得到龙首前把手位置依赖于运动时间的关系。通过某一节把手位置和后续节把手位置的距离约束关系，以及把手自身位置受到等距螺线的约束，可以不断地推出后续把手的位置依赖于运动时间的关系。

考察速度的定义，考虑按照以上方法再次计算给定时刻增加一个微小的时间变化量后的把手位置，比较这两种时刻各个把手的位置差即可得到在微小时内各个把手移动的微小距离，进而可以近似得到各个把手沿等距螺线的线速度。

2.2 问题二分析

问题二要求我们求出板凳龙按问题一设定的螺线盘入的中止时刻，即碰撞的时刻（此时不能再继续盘入），并给出碰撞时刻舞龙队的位置和速度信息。由于问题一已经求出了各时刻板凳龙各把手的坐标，我们可以以此为基础建立板凳龙的几何模型，例如各个板凳的边界线段，通过碰撞检测算法又积法时刻检测各个板凳的边界线段是否相交（即表示板凳发生碰撞），精确求出碰撞的时刻和位置。另外，在动画中显示出板凳龙外部边界的运动轨迹和潜在碰撞可以印证算法的计算结果，也能帮助我们认知碰撞发生的大致时刻以及碰撞时的状态。值得注意的是，最可能发生碰撞的是龙头，因此算法选择检测龙头向外的边界与其他龙身向内的边界是否相交。

2.3 问题三分析

问题三要求我们求解舞龙队能够安全移动至掉头空间边界的最小螺距，即在不发生碰撞的情况下，龙头前把手能够顺利到达边界的最小螺距。在问题二中，我们通过碰撞

检测算法得到了碰撞发生时刻舞龙队各节段的位置信息，这为我们提供了评估螺距可行性的依据。于是，问题三的螺距判断可以通过构建螺距判断模型来解决，这只需修改问题二算法中关于螺距的参数即可。为了寻找到最小螺距，我们采用二分法优化模型进行迭代寻找，取舍的标准是，若该螺距通过螺距判断模型得到的碰撞位置在掉头空间圆内，则说明龙头把手恰接触掉头空间边界时仍未发生碰撞，这样的螺距应该保留，反之应该舍去，最终可以求解出最小螺距。

2.4 问题四分析

问题四有两小问，第一问要求设计一段落在掉头空间内的最短掉头路径。首先，可以直接获得盘入螺线、盘出螺线和圆线在极坐标系下的方程，先计算出盘入、盘出螺线和圆线的交点，分别作为盘入口和盘出口。接着，可以仅仅关注掉头空间圆内的区域：我们设置两端圆弧的圆心、半径以及圆心角作为未知量，通过圆弧与外部螺线相切、圆弧过盘入口和盘出口、圆弧与圆弧相切等限制得到约束方程。最后希望目标函数掉头路径曲线长度为最小值，利用非线性规划问题的处理方法得到符合题意最佳的掉头路径，以及路径的具体参数。此外，我们需要验证板凳龙在这样的掉头路径上前进不会发生碰撞，检测方法与问题二类似。

第二小问要求得到一段时间内每个时刻舞龙队的把手的位置和速度，由于舞龙队行进的路径已经被确定，理论上已知龙头的速度，就能够得到龙头依赖于时刻的位置，并通过递推迭代的方法得到之后所有板凳把手的位置和速度。这是与问题一相类似的，不同之处在于问题一的行经曲线是单一曲线，而本问中的曲线是四段不同形式的曲线组合而成的复合曲线。在实际计算中，我们设计迭代切换算法来计算后续的板凳把手应该落在哪一段曲线上。切换算法可以实现如下功能：某把手应递推至盘出螺线上的后一个把手，切换至，某把手应递推至掉头曲线上的后一个把手，切换至，某把手应该递推至盘入螺线上的后一个把手。

按照以上算法得到各个板凳把手依赖于时刻的位置信息，之后又可以效仿第一问，考虑增加一个微小时时间量后的时刻，比较两个时刻的位置差进而得到各个把手前进的微小距离，根据速度的差分定义，可以近似求得各个板凳沿复合曲线路径前进的速度依赖于时刻的值。

2.5 问题五分析

问题五基于问题四，仍然沿用复合曲线作为舞龙队前进的路径，问题进行到现在，舞龙队应该如何舞龙，使得在舞龙队能在不发生碰撞的前提下，盘龙所需要的面积越小、行进速度越快，展现出最大程度的美观效果的问题，已经即将解决。问题五固定龙头行进速度保持不变，要求在各个时刻，板凳龙各把手的速度不超过 2 m/s ，确定龙头的最大行进速度。基于问题四第二问，先沿用龙头前进速度为 1 m/s ，与前几个问题同

理，可以直接获得各个时刻所有把手的速度信息，由于其余把手速度大致与龙头把手速度成线性关系，于是我们选择全时刻全把手中的最大速度，使其按某一放大倍数放大至2 m/s，则将1m/s乘以该放大倍数就是龙头的最大前进速度。

三、模型假设

1. 假设舞龙队成员能严格地使各个把手按照指定的轨道进行运动，并且各个板凳的把手离地高度完全一致
2. 假设板凳龙的各个板凳不会发生形变，板凳上任意两点之间运动状况的关系仅仅取决于它们之间的几何关系，这种状态的传导不需要时间
3. 忽略板凳龙所有板凳的厚度，所有板凳都处在与地面平行的某平面上
4. 将所有把手抽象为其中心一点

四、符号说明

符号	意义	符号	意义
$r(\theta)$	等距螺线	$\hat{r}(\theta)$	中心对称的等距螺线
p	等距螺线的螺距	r_0	初始螺距
x_0	龙头把手横坐标	y_0	龙头把手纵坐标
x_i	除龙头把手第 i 节横坐标	y_i	除龙头把手第 i 节纵坐标
L	曲线弧长	v	沿曲线前进的速度
$P_i(t)$	第 i 节的直角坐标	$L_i(t)$	第 i 节指向第 $i + 1$ 节的向量
c_0	龙头上两把手的距离	c_1	其余板凳上两把手的距离
$half_width$	板凳宽边的一半	$extension$	把手到较近宽边的距离
r_{small}	掉头区域内小圆的半径	r_{large}	掉头区域内大圆的半径
x_{small}	掉头区域内小圆圆心横坐标	x_{large}	掉头区域内大圆圆心横坐标
y_{small}	掉头区域内小圆圆心纵坐标	y_{large}	掉头区域内大圆圆心纵坐标
r_{circle}	掉头空间圆的半径	R	掉头区域内小圆半径
θ_{in}	盘入口极角	$theta_{out}$	盘出口极角
$k_1(\theta)$	盘入曲线直角坐标下的切线斜率		

五、模型的建立与求解

等距螺线的极坐标方程为：

$$r(\theta) = r_0 + \frac{p\theta}{2\pi}$$

其中， $r_0 = 0$ 是初始半径偏移， p 是螺距离， θ 为极角，不难发现，当极角确定时，等距螺线上点的位置也唯一确定了，可以说，知道极角，就知道了直角坐标系下的位置。

为了便于分析和绘制螺旋曲线，我们将极坐标下的位置信息转换为笛卡尔坐标，利用如下公式：

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

5.1 模型一的建立与求解

5.1.1 获得龙头把手的位置信息

已知有极坐标形式下曲线积分的表达式，积分项上下限为初始极角 $\theta_{start} = 32\pi$ ，终止极角 θ_0 ：

$$L = \int_{\theta_0}^{\theta_{start}} \sqrt{\frac{dr}{d\theta}^2 + r(\theta)^2}$$

我们假设出运动时间 t 以及已知速度 $v = 1m/s$ ，可以获得 L 的另一种表达方式：

$$L = v \times t$$

由此我们可以得出依赖关系 $\theta_0 = \theta_0(t)$ ，代表龙头的极角依赖运动时间的关系。

5.1.2 通过递推获得下一节把手的位置信息

已知 $\theta_0(t)$ ，希望得到第一节龙身把手位置 θ_0 依赖于运动时间的关系，

两点之间的欧几里得距离，其中 $c_0 = 3.41 - 2 \times 0.275(m)$ 是龙头把手和第一节龙身把手之间的距离：

$$\|L_0(t)\| = \sqrt{(x_1 - x_0(t))^2 + (y_1 - y_0(t))^2} = c_0$$

结合第一节龙身把手在等距螺线上的约束：

$$\begin{cases} r(\theta_1) = \frac{p\theta_1}{2\pi} \\ x_1 = r(\theta_1) \cdot \cos(\theta_1) \\ y_1 = r(\theta_1) \cdot \sin(\theta_1) \end{cases}$$

最终可以得到 $\theta_1 = \theta_1(t)$ 。

接着不断根据相同的方法进行递推，由于递推次数较大，本文使用 matlab 程序辅助运算，第一问的代码文件 question_1.m 提供了 get_location 函数。值得注意的是，在欧几里得距离约束除了以上龙头把手递推第一节龙身把手，之后应该将以上的 c_0 替换成 $c_1 = 2.20 - 2 \times 0.275(m)$ ，因为只有龙头的板长较长。最终可以获得所有板凳把手的位置信息。

	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)	8.363824	7.456758	-1.445473	-5.237118	4.821221	2.459489
第1节龙身y (m)	2.826544	-3.440399	-7.405883	4.359627	-3.561949	4.402476
第51节龙身x (m)	-9.518732	-8.686317	-5.543150	2.890455	5.980011	-6.301346
第51节龙身y (m)	1.341137	2.540108	6.377946	7.249289	-3.827758	0.465829
第101节龙身x (m)	2.913984	5.687116	5.361939	1.898794	-4.917371	-6.237722
第101节龙身y (m)	-9.918311	-8.001384	-7.557638	-8.471614	-6.379874	3.936008
第151节龙身x (m)	10.861726	6.682311	2.388757	1.005154	2.965378	7.040740
第151节龙身y (m)	1.828754	8.134544	9.727411	9.424751	8.399721	4.393013
第201节龙身x (m)	4.555102	-6.619664	-10.627211	-9.287720	-7.457151	-7.458662
第201节龙身y (m)	10.725118	9.025570	1.359847	-4.246673	-6.180726	-5.263384
龙尾 (后) x (m)	-5.305444	7.364557	10.974348	7.383895	3.241051	1.785033
龙尾 (后) y (m)	-10.676584	-8.797991	0.843473	7.492371	9.469337	9.301164

表一 各个把手的位置信息

5.1.3 通过位置信息获得速度信息

考虑速度的定义 $v = \lim_{\Delta t \rightarrow 0} \frac{\Delta x}{\Delta t}$

再次计算给定时刻增加一个微小的时间变化量后的所有把手位置，比较这两种时刻各个把手的位置差即可得到在微小时间內各个把手移动的微小距离，进而可以近似得到各个把手沿等距螺线的线速度。代码文件 question_1.m 提供了 get_speed 函数来完成这一想法。

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 (m/s)	0.999951	0.999943	0.999932	0.999917	0.999892	0.999846
第1节龙身 (m/s)	0.999922	0.999904	0.999878	0.999834	0.999752	0.999559
第51节龙身 (m/s)	0.999700	0.999615	0.999485	0.999268	0.998865	0.997969
第101节龙身 (m/s)	0.999539	0.999413	0.999224	0.998920	0.998376	0.997232
第151节龙身 (m/s)	0.999416	0.999264	0.999040	0.998685	0.998067	0.996806
第201节龙身 (m/s)	0.999320	0.999150	0.998901	0.998515	0.997853	0.996529
龙尾 (后) (m/s)	0.999284	0.999107	0.998851	0.998454	0.997778	0.996435

表二 各个把手的速度信息

5.2 模型二的建立与求解

问题二要求我们求出板凳龙按问题一设定的螺线盘入的终止时刻，即碰撞的时刻（此时不能再继续盘入），并给出碰撞时刻舞龙队的位置和速度信息。首先，基于问题一，利用问题一程序 question_1.m 中的函数 get_location，我们可以求出 0s 至 500s 之间各时刻舞龙队的位置信息，将其保存至文件 result1_500.xlsx 之中。接着，我们根据这些位置信息求解出各个板凳的几何模型，亦即组成它们边界的线段。然后，我们通过碰撞检测算法，检查不同板凳边界的线段是否会相交，我们尤其关注龙头的情况。最终得以算出龙头发生碰撞时的位置信息和时间信息。

5.2.1 板凳龙的几何模型

(1) 读取并提取坐标

假设在时刻 t ，舞龙队的每个节点 $P_i(t)$ 的二维坐标为：

$$P_i(t) = (x_i(t), y_i(t)), \quad i = 1, 2, \dots, N$$

其中， $x_i(t)$ 和 $y_i(t)$ 分别是第 i 个节点在时刻 t 的横坐标和纵坐标。

从 Excel 文件中读取数据后，对于第 i 个节点的 $x_i(t)$ 和 $y_i(t)$ ，可以从数据矩阵中提取如下：

$$x_i(t) = \text{data}(2i - 1, t), \quad y_i(t) = \text{data}(2i, t)$$

其中，奇数行表示横坐标，偶数行表示纵坐标。

(2) 原始线段的定义

任意两个相邻的节点 $P_i(t)$ 和 $P_{i+1}(t)$ 构成的线段 $L_i(t)$ 可以用向量表示为：

$$L_i(t) = (x_{i+1}(t) - x_i(t), y_{i+1}(t) - y_i(t))$$

这个向量代表从 $P_i(t)$ 指向 $P_{i+1}(t)$ 的线段。

(3) 单位方向向量和垂直向量

将线段归一化得到单位方向向量 u_i ：

$$u_i = \frac{P_{i+1}(t) - P_i(t)}{\|P_{i+1}(t) - P_i(t)\|} = \frac{L_i(t)}{\|L_i(t)\|}$$

其中，线段的长度 $\|L_i(t)\|$ 通过两点之间的欧几里得距离计算：

$$\|L_i(t)\| = \sqrt{(x_{i+1}(t) - x_i(t))^2 + (y_{i+1}(t) - y_i(t))^2}$$

为了平移线段，我们需要垂直于方向向量 u_i 的垂直向量 p_i ：

$$p_i = (-(y_{i+1}(t) - y_i(t)), x_{i+1}(t) - x_i(t))$$

将其归一化为单位垂直向量 p_i^{norm} :

$$p_i^{\text{norm}} = \frac{p_i}{\|p_i\|}$$

由于碰撞发生在板凳与板凳的边界上，而把手位于板凳的内部，所以不能单纯研究把手的连线进行碰撞的检测，需要将线段平移来模拟板凳的宽度，线段延长来模拟板凳实际的长度。

(4) 平移线段

将线段沿着单位垂直向量 p_i^{norm} 平移，平移距离为 `half_width`。第 i 个节点平移后的坐标为：

$$P_i^{\text{shifted}}(t) = P_i(t) + \text{half_width} \cdot p_i^{\text{norm}}$$

同样，对于相邻的节点 $P_{i+1}(t)$ ，其平移后的坐标为：

$$P_{i+1}^{\text{shifted}}(t) = P_{i+1}(t) + \text{half_width} \cdot p_i^{\text{norm}}$$

(5) 延长线段

为了延长线段，沿着原方向向量 u_i 对线段的两端点进行延长。延长后的端点为：

$$P_i^{\text{extended}}(t) = P_i^{\text{shifted}}(t) - \text{extension} \cdot u_i$$

$$P_{i+1}^{\text{extended}}(t) = P_{i+1}^{\text{shifted}}(t) + \text{extension} \cdot u_i$$

5.2.2 板凳龙的碰撞检测模型

碰撞检测的数学模型可以基于**向量几何**来判断两条线段是否相交。下面我将详细描述这个数学模型的构建，并给出相关的公式。

检测原理

假设 有两条线段，分别由端点 P_1, P_2 和 P_3, P_4 表示：第一条线段的端点是 $P_1(x_1, y_1)$ 和 $P_2(x_2, y_2)$ 。第二条线段的端点是 $P_3(x_3, y_3)$ 和 $P_4(x_4, y_4)$ 。

目标是确定这两条线段是否相交。

I. 线段表示为向量 每一条线段都可以表示为一个向量：

线段 1 的向量为：

$$\mathbf{v}_1 = P_2 - P_1 = (x_2 - x_1, y_2 - y_1)$$

线段 2 的向量为：

$$\mathbf{v}_2 = P_4 - P_3 = (x_4 - x_3, y_4 - y_3)$$

II. 叉积计算 为了判断两条线段是否相交，利用叉积来判断向量的位置关系。对于两个向量 \mathbf{v}_1 和 \mathbf{v}_2 ，其叉积可以表示为：

$$\text{cross}(\mathbf{v}_1, \mathbf{v}_2) = (x_2 - x_1) \cdot (y_4 - y_3) - (y_2 - y_1) \cdot (x_4 - x_3)$$

如果叉积等于零，则两条线段平行或共线，不会发生碰撞。如果叉积不为零，则有可能相交。

III. 判断点是否在线段的两侧 为了进一步判断两条线段是否相交，需要使用叉积来检测两条线段的端点是否分别位于对方的两侧。对于线段 1 和线段 2，有以下几个条件：

对线段 1 进行判断

首先，检查点 P_3 和 P_4 是否位于线段 1 的两侧：计算向量 $\mathbf{v}_3 = P_3 - P_1$ 和 $\mathbf{v}_4 = P_4 - P_1$ 。然后，计算以下两个叉积：

$$\text{cross}(\mathbf{v}_1, \mathbf{v}_3) = (x_2 - x_1) \cdot (y_3 - y_1) - (y_2 - y_1) \cdot (x_3 - x_1)$$

$$\text{cross}(\mathbf{v}_1, \mathbf{v}_4) = (x_2 - x_1) \cdot (y_4 - y_1) - (y_2 - y_1) \cdot (x_4 - x_1)$$

如果这两个叉积的符号相反，即：

$$\text{sign}(\text{cross}(\mathbf{v}_1, \mathbf{v}_3)) \neq \text{sign}(\text{cross}(\mathbf{v}_1, \mathbf{v}_4))$$

则说明点 P_3 和 P_4 位于线段 1 的两侧。

对线段 2 进行判断

同理，检查点 P_1 和 P_2 是否位于线段 2 的两侧：计算向量 $\mathbf{v}_5 = P_1 - P_3$ 和 $\mathbf{v}_6 = P_2 - P_3$ 。然后，计算以下两个叉积：

$$\text{cross}(\mathbf{v}_2, \mathbf{v}_5) = (x_4 - x_3) \cdot (y_1 - y_3) - (y_4 - y_3) \cdot (x_1 - x_3)$$

$$\text{cross}(\mathbf{v}_2, \mathbf{v}_6) = (x_4 - x_3) \cdot (y_2 - y_3) - (y_4 - y_3) \cdot (x_2 - x_3)$$

如果这两个叉积的符号相反，即：

$$\text{sign}(\text{cross}(\mathbf{v}_2, \mathbf{v}_5)) \neq \text{sign}(\text{cross}(\mathbf{v}_2, \mathbf{v}_6))$$

则说明点 P_1 和 P_2 位于线段 2 的两侧。

IV. 交点判断

如果两条线段满足上述条件，即线段 1 的两个端点分别位于线段 2 的两侧，线段 2 的两个端点分别位于线段 1 的两侧，则可以确定这两条线段相交。

V. 特殊情况：共线检测

如果线段共线（即叉积为 0），则需要进一步判断两条线段是否重叠。可以通过判断两个线段端点的投影是否重叠来解决。

总结 线段相交的数学模型可以归纳为以下步骤：

1. 计算两条线段的方向向量 \mathbf{v}_1 和 \mathbf{v}_2 。
2. 使用叉积判断线段是否平行，如果不平行，则继续判断。
3. 使用叉积和符号判断两条线段是否相交：即检查两个点是否位于对方的线段的两侧。
4. 如果符号不同，则这两条线段相交。

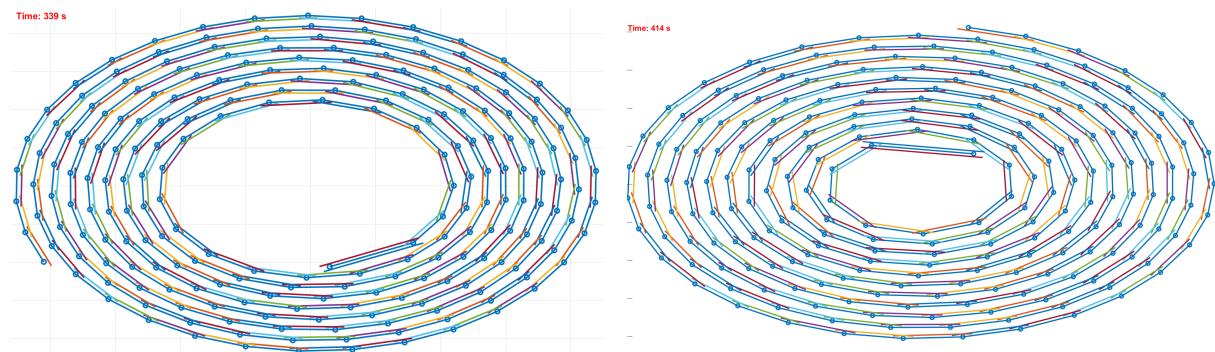
5.2.3 问题二模型的求解

(1) 数据获取

利用问题一代码中的 `get_location.m`, 将 T 设置为 $T = [0:1:500]$; 求出 0 到 500s 时板凳龙各把手的信息 $location = get_location(T)$; 得出结果保存至文件 `result1_500.xlsx` 之中。

(2) 几何模型求解

按照上述几何模型的构建配合之叉乘法，理论上我们可以检测每个时刻任意两个不相邻板凳的边界是否有交点。由于要检测每个时刻每个板凳之间的是否交叉性，计算量十分之大，考虑使用 matlab 编程获得程序 `animation_crash.m` 实现以上功能，该程序将读入文件 `result1_500.xlsx` 中的数据，一方面每秒进行严格的数值计算，检测是否存在两条边界线段相交，一方面借助其中的大量数据进行可视化，便于直观理解。最终获得碰撞时刻大致是 414s，如果想要获得更加精确的结果，则需要将 `get_location.m` 中的 T 设置为 0 至 500 分点更多、更加密集排布的数列进行进一步运算，本文在此不作更细致的运算。



图一 339s 时的位置信息

图二 414s 时的位置信息

(3) 根据碰撞时刻在表格中查找出

当碰撞时刻确定为 414s 时，立刻可以找到 `result1_500.xlsx` 中对应的列，其中记录了 224 个把手点的位置信息和速度信息，于是此时龙头前把手、龙头后面第 1、51、101、151、201 条龙身前把手和龙尾后把手的位置和速度都能展示在下表中。

T = 414s	横坐标x (m)	纵坐标y (m)	速度 (m/s)
龙头	2.102932	0.740969	0.999230
第1节龙身	-0.273863	2.331706	0.989741
第51节龙身	2.601151	3.651598	0.974627
第101节龙身	-1.978464	-5.539781	0.972354
第151节龙身	-0.512413	-6.987328	0.971434
第201节龙身	-7.968388	0.248727	0.970935
龙尾 (后)	2.403961	8.008969	0.970781

表三 撞击时刻信息

5.3 模型三的建立与求解

问题三要求我们确定最小螺距，使板凳龙能到达所谓的掉头空间，即龙头在到达掉头空间之前不发生碰撞。这里我们继续使用第二问的碰撞检测算法 `animation_crash.m`，利用第二问建立的板凳龙几何模型，不断改变螺距，检验相应螺距下板凳龙在进入掉头空间前是否发生碰撞，并根据螺距是否可行利用二分法优化模型进行取舍，多次迭代后逐渐逼近最小螺距。以下阐述了最小螺距的二分法求解优化模型。

5.3.1 优化二分法模型的介绍

二分法 \square *Binary Search or Bisection Method* \square 是一种广泛应用于数值计算中的根求解和优化算法。它通过逐步缩小问题的解区间来逼近解。二分法适用于单调递增或单调递减的函数，或者对于满足某些条件的问题，可以通过二分法在已知的区间内逐步逼近最优解。

5.3.2 优化模型的建立

在此优化问题中，需确定一个合适的螺距，使舞龙轨迹满足以下两个约束：

(1) 龙头进入调头空间

龙头（第一个节点）的欧几里得距离在某个时刻 t 小于等于 4.5 米，即满足

$$\sqrt{x_1(t)^2 + y_1(t)^2} \leq 4.5$$

(2) 没有发生碰撞

舞龙的各个节点沿着螺旋线运动，不应与其他部分的轨迹发生交叉碰撞。

(3) 优化问题的标准表示形式

$$\min_p p \quad \text{s.t.} \begin{cases} x_1(t)^2 + y_1(t)^2 \leq R_{\text{center}}^2 & (\text{调头空间约束}) \\ \text{No intersection between any two line segments at time } t & (\text{无碰撞约束}) \end{cases}$$

p 是螺距，需要最小化；

$(x_0(t), y_0(t))$ 是龙头（第零个节点）的坐标；

$r_{\text{circle}} = 4.5$ 米，是调头空间的半径；

调头空间约束：确保龙头能够在某个时间 t 进入调头空间，满足

$$\sqrt{x_0(t)^2 + y_0(t)^2} \leq 4.5$$

无碰撞约束：确保舞龙路径中的线段不会相互碰撞，线段之间不相交。

这是一种典型的带有非线性约束的优化问题，目标是找到最小的螺距 p ，在满足调头空间和无碰撞的条件下实现最优解。为了解决这个问题，算法选择了**二分法**来优化螺距 p ，因为螺距与舞龙轨迹的特性呈单调关系。具体来说：

1. 当螺距较大时，舞龙的轨迹会比较松散，龙头进入调头空间较慢，但碰撞的风险较小；
2. 当螺距较小时，轨迹紧凑，龙头更容易进入调头空间，但碰撞风险增加。

5.3.3 二分法的工作原理

二分法的思想

给定一个区间 $[p_{\min}, p_{\max}]$ ，初始的最小螺距为 p_{\min} ，最大的螺距为 p_{\max} ，我们希望找到最小的螺距 p^* 使得龙头刚好进入调头空间，同时没有发生碰撞。

二分法的基本步骤

1. **初始化：**设定螺距的上下界 p_{\min} 和 p_{\max} ，并设定优化精度（公差） ϵ 。比如，初始化 $p_{\min} = 0.43$ 和 $p_{\max} = 0.46$ ，公差 $\epsilon = 0.002$ 。

2. **计算中间值：**在每次迭代中，计算当前的中间螺距 p_{mid} ，即

$$p_{\text{mid}} = \frac{p_{\min} + p_{\max}}{2}$$

3. **判断条件：**根据当前的螺距 p_{mid} ，生成舞龙的轨迹，检查是否满足两个约束：1. 龙头是否进入调头空间：检查龙头（第一个节点）的位置 $(x_1(t), y_1(t))$ ，判断其是否进入调头空间：

$$\sqrt{x_1^2 + y_1^2} \leq 4.5$$

2. 是否发生碰撞：使用几何学上的碰撞检测算法检查舞龙各部分的线段是否相交。

4. **调整区间：**如果轨迹没有发生碰撞并且龙头进入调头空间，说明当前螺距 p_{mid} 是可行的，我们继续尝试更小的螺距，因此更新 $p_{\max} = p_{\text{mid}}$ 。如果发生碰撞，说明当前螺距 p_{mid} 过小，需要尝试更大的螺距，因此更新 $p_{\min} = p_{\text{mid}}$ 。

5. 终止条件：当区间的长度 $|p_{\max} - p_{\min}|$ 小于设定的精度 ϵ 时，停止迭代，此时的中间值 p_{mid} 就是逼近的最优解，即最小螺距 p^* 。

5.3.4 模型的求解

程序 min_p_recur.m 是以上算法描述的编程实现，直接运行 min_p_recur.m 即可获得最小螺距大约是 0.450625 m。

迭代过程与结果显示：

```
>> check
碰撞发生在时间: t = 395 秒, 对应的螺距: 0.445
龙头到达中心, 距离: 4.4853
    对应的螺距: 0.4525
碰撞发生在时间: t = 393 秒, 对应的螺距: 0.44875
龙头到达中心, 距离: 4.4957
    对应的螺距: 0.45062
最小螺距为: 0.45062
: >>
```

图三 螺距判断模型的迭代过程

图中如果发生碰撞则显示碰撞时间，如果在进入中心区域前没有发生任何碰撞则显示“龙头到达中心”。得到结果为 0.450625m，与上次迭代的螺距 0.448756m 相差小于误差允许范围 0.002m，故停止迭代，取 0.450625m 为最终结果。

5.3.5 模型的检验

利用问题二中的碰撞检测程序 animation_crash.m，不断修改该程序的螺距，可以画出不同螺距下的图像，确定出最小螺距的大致范围，这段范围不仅可以反过来修改 animation_crash.m 中螺距的初始上下界，减少算力消耗，还能与模型直接的求解的结果比对，完成验证过程。

螺距/cm	龙头X/m	龙头Y/m	龙头半径/m	碰撞时间/s
44.9	-1.20169	-4.46924	4.62798	393
45	-3.78004	-2.56419	4.56769	396
45.1	0.565689	4.35691	4.39348	406
45.2	-0.13354	4.39316	4.39519	405
45.5	-1.22425	4.20985	4.38425	403

表四 验证螺距判断模型

5.4 模型四的建立与求解

5.4.1 确定盘入口和盘出口

只需用到极径等于掉头区域的圆半径即可：

$$r(\theta) = r_{\text{circle}} = 4.5(m)$$

与原等距螺线 $r(\theta) = \frac{p\theta}{2\pi}$ 中心对成的等距螺线是 $\hat{r}(\theta) = -\frac{p\theta}{2\pi}$

解得适用于等距螺线 $r(\theta)$ 的入口极角 $\theta_{in} = 16.631961(\text{rad})$ ，对应盘入口的直角坐标为 $(-2.711856, -3.591078)(m)$

同理，适用于等距螺线 $\hat{r}(\theta)$ 的入口极角 $\theta_{out} = 16.631961(\text{rad})$ ，对应盘入口的直角坐标为 $(2.711856, 3.591078)(m)$

5.4.2 确定掉头区域内最短路径

我们只需关注圆内区域即可，为了方便描述题意中的约束，先假设待定量为大圆的圆心 (x_{large}, y_{large}) ，小圆的圆心 (x_{small}, y_{small}) ，小圆的半径 R ，大圆的半径 $2R$ ，大圆圆弧的圆心角 θ_{large} ，以及小圆圆弧的圆心角 θ_{small} 。

设置完如上待定量，接下来我们给出满足题意的约束方程：

约束 1 大圆过盘入点：

$$(x_{large} - x_0)^2 + (y_{large} - y_0)^2 - (2R)^2 = 0$$

约束 2 小圆过盘出点：

$$(x_{small} + x_0)^2 + (y_{small} + y_0)^2 - R^2 = 0$$

约束 3 大圆与小圆相切：

$$(x_{large} - x_{small})^2 + (y_{large} - y_{small})^2 - (3R)^2 = 0$$

约束 4 大圆圆弧圆心角的性质：

$$\cos(\theta_{large}) - \frac{(x_0 - x_1)(x_{small} - x_{large}) + (y_0 - y_{large})(y_{small} - y_{large})}{\|[x_0 - x_{large}, y_0 - y_{large}]\| \cdot \|[x_{small} - x_{large}, y_{small} - y_{large}]\|} = 0$$

约束 5 小圆圆弧圆心角的性质：

$$\cos(\theta_{small}) - \frac{(-x_0 - x_{small})(x_{large} - x_{small}) + (-y_0 - y_{small})(y_{large} - y_{small})}{\|[-x_0 - x_{small}, -y_0 - y_{small}]\| \cdot \|[x_{large} - x_{small}, y_{large} - y_{small}]\|} = 0$$

约束 6 大圆与盘入螺线相切:

$$\begin{bmatrix} 1 & k_1(\theta_{in}) \end{bmatrix} \begin{bmatrix} x_0 - x_{large} \\ y_0 - y_{large} \end{bmatrix} = 0$$

约束 7 小圆与盘出螺线相切:

$$\begin{bmatrix} 1 & k_1(\theta_{in}) \end{bmatrix} \begin{bmatrix} x_0 + x_{small} \\ y_0 + y_{small} \end{bmatrix} = 0$$

其中 $k_1(\theta)$ 是原等距螺线的斜率关于其极角的函数:

$$k_1(\theta) = \frac{\left(\frac{p}{2\pi} \sin(\theta) + \frac{p\theta}{2\pi} \cos(\theta) \right)}{\left(\frac{p}{2\pi} \cos(\theta) - \frac{p\theta}{2\pi} \sin(\theta) \right)}$$

希望求得最小值的目标函数

$$\min\{2R\theta_{large} + R\theta_{small}\}$$

观察该图形的几何性质，我们不难发现盘入口和盘出口是关于原点中心对称的，而大圆圆弧的圆心角等于小圆圆弧的圆心角，这些都将简化我们的运算。

以上的约束条件构成了一个非线性规划问题。目标是最小化调头路径的总长度，同时满足上述几何约束，可以通过 optimproblem 函数创建优化问题，利用 fmincon 来解决这个问题，这部分在程序 question_4_1.m 中得以实现。

求解得到最短调头路径长度 $L_{min} = 13.621245(m)$,

以及路径的具体参数,

$$x_{large} = -0.760009, \quad y_{large} = -1.305726,$$

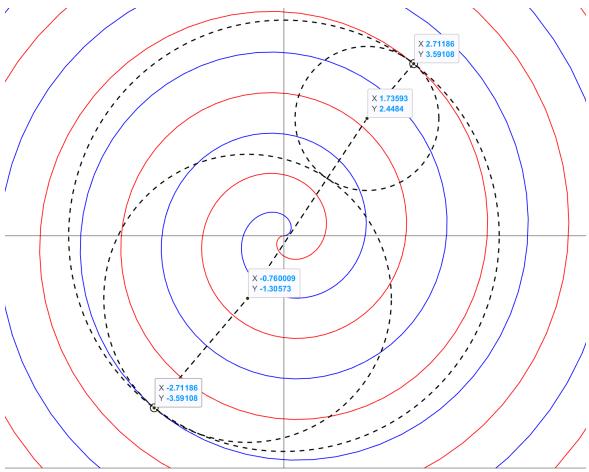
$$x_{small} = 1.735932, \quad y_{small} = 2.448402,$$

$$\theta_{small} = \theta_{large} = 3.021487,$$

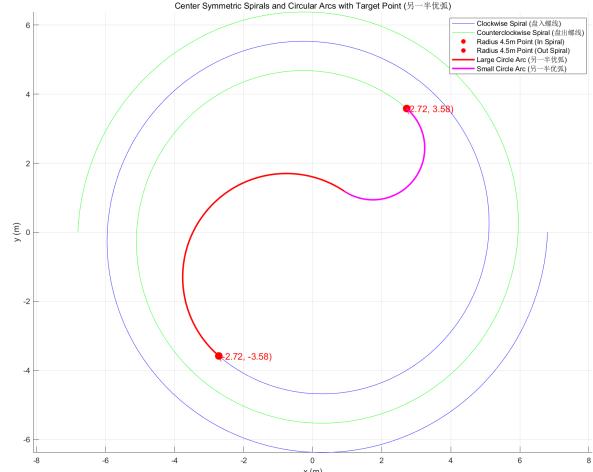
$$R = 1.502709$$

5.4.3 可视化展示

以下是上述两种解答的可视化，在一定误差范围内是符合题意的:



图四 最短路径的一个图像



图五 最短路径的另一种图像

5.4.4 建立各曲线内部迭代关系

轨迹可以分为盘入曲线, 大圆弧, 小圆弧, 盘出曲线各部分, 依次建立盘入曲线, 大圆弧, 小圆弧, 盘出曲线内部的迭代关系, 建立所用方程为:

1. 板凳龙的前后两节把手之间的距离

设第 i 节把手的位置为 $P_i = (x_i, y_i)$, 第 $i + 1$ 节把手的位置为 $P_{i+1} = (x_{i+1}, y_{i+1})$, 那么它们之间的距离 $\|L_i\|$ 可以表示为:

$$\|L_i\| = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

这个公式是两点之间距离的标准几何表达式, 用来描述前后两节把手之间的距离。

说明: x_i 和 y_i 是第 i 节把手的位置坐标; x_{i+1} 和 y_{i+1} 是第 $i + 1$ 节把手的位置坐标; d 是两节把手之间的距离。

2. 总的几何关系

大圆弧的标准方程:

$$(x - x_{large})^2 + (y - y_{large})^2 = (2R)^2$$

小圆弧的标准方程:

$$(x - x_{small})^2 + (y - y_{small})^2 = R^2$$

盘入螺旋线的方程 (顺时针):

$$x_{in}(\theta) = p * \theta / (2 * \pi) \cdot \cos(\theta)$$

$$y_{in}(\theta) = p * \theta / (2 * \pi) \cdot \sin(\theta)$$

盘出螺旋线的方程（逆时针）：

$$x_{\text{out}}(\theta) = -p * \theta / (2 * \pi) \cdot \cos(\theta)$$

$$y_{\text{out}}(\theta) = -p * \theta / (2 * \pi) \cdot \sin(\theta)$$

将距离方程与几何方程联立，用 `fsove` 函数求解即可表示出不同曲线内部的迭代规律

5.4.5 判断迭代过程中曲线的切换

由于迭代是从龙头开始逐渐向后迭代，所以轨迹变化只有以下三种：

变化 1：从盘出螺线到小圆弧的变化

变化 2：从小圆弧到大圆弧的变化

变化 3：从大圆弧到盘入螺线的变化

我们定义这些变化的数学条件如下：

1. 半径达到预定值：检测螺旋线的半径变化，当螺旋线的半径 $r \leq 4.5 \text{ m}$ 时，返回 $\text{exitflag} = -1$ ，说明该位置为无效位置，接下来的位置应该位于小圆弧上。

2. 角度达到最大值：当把手在小圆弧上运动时，若检测到某把手对应位置转过的角度大于最大圆心角 α_1 ，则该位置无效，接下来位置应该位于大圆弧上。

小圆弧的最大圆心角可以表示为：

$$\theta_{\text{small}} = 3.021487 \text{ rad}$$

我们通过跟踪旋转角度来判断是否超出最大值。

3. 角度达到最大值：当把手在大圆弧上运动时，若检测到某把手对应位置转过的角度大于最大圆心角 α_2 ，则该位置无效，接下来位置应该位于盘入螺线上。

大圆弧的最大圆心角可以表示为：

$$\theta_{\text{large}} = 3.021487 \text{ rad}$$

同样，通过检测旋转角度来判断轨迹是否需要从大圆弧转变到盘入螺线。

5.4.6 龙头前把手位置随时间的变化关系

1. 龙头前把手在盘入螺线的位置：

根据第一问，可以得到 $t \in [-100, 0]$ 时刻，龙头前把手位置的表达式为：

$$r_{\text{in}}(\theta) = A + B\theta$$

其中， $A = 0$ ， $B = \frac{1.70}{2\pi}$ 。这是在盘出螺线上的位置。

2. 掉头空间区域的时间计算:

根据第四问的结果，掉头空间区域的运动时间可以通过弧长计算公式：

$$L = R\theta$$

其中 R 为圆弧半径， θ 为该段圆弧对应的圆心角。根据该弧长和龙头的线速度 v ，可以计算龙头在掉头空间中运动的时间 T 。

3. 龙头前把手在盘出螺线的位置:

盘出螺线和盘入螺线是中心对称的，因此盘出螺线的位置与时间关系类似于盘入螺线的情况。盘出螺线位置方程为：

$$r_{out}(\theta) = A - B\theta$$

$$t \in [T, 100]$$

4. 龙头前把手在圆弧内部的位置信息:

两段圆弧均为已知弧长和参数，因此可以通过弧长公式推导出圆弧内部的位置信息，表示为时间的函数。

5.4.7 各把手不同时刻位置与速度的计算

由不同时刻龙头前把手的位置信息，迭代出该时刻各节把手的位置信息。并利用与第一问中相同的速度计算法，求出各节各时刻的速度。但我们在圆弧间迭代时，可能会出现多解的情况。因此，可以通过限制横纵坐标的范围等方式，避免迭代过程中的多解问题。

以上算法由 matlab 程序 question_4_2.m 实现。同问题一类似，该程序首先能够获得一系列时刻 $[-100 : 1 : 100]$ 下，各个把手的直角坐标位置；该程序还能获得一系列时刻偏移一段微小时间量 $0.001s$ 后各个把手偏移过的直角坐标位置；该程序最后通过差分近似极限的方法得到各个时刻下各个板凳把手的线速度。

我们将其各个时刻下各个把手的直角坐标位置保存至文件 result4.xlsx 的 sheet1 中以及文件 location_origin_5.xlsx 中，将各个时刻下各个把手偏移过的直角坐标位置保存到文件 location_delta_5.xlsx 中；将各个时刻下各个板凳把手的线速度保存至文件 result4.xlsx 的 sheet2 中以及文件 speed_5.xlsx 中。

按题目要求，此处给出部分时刻部分把手的位置和速度如下

	-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)	0.999904	0.999762	0.999878	1.000363	1.000124
第51节龙身 (m/s)	0.999346	0.998642	0.995116	0.950037	1.003966
第101节龙身 (m/s)	0.999091	0.998248	0.994429	0.948584	1.096325
第151节龙身 (m/s)	0.998944	0.998047	0.994137	0.948140	1.095368
第201节龙身 (m/s)	0.998849	0.997925	0.993976	0.947925	1.094996
龙尾 (后) (m/s)	0.998817	0.997885	0.993925	0.947862	1.094895

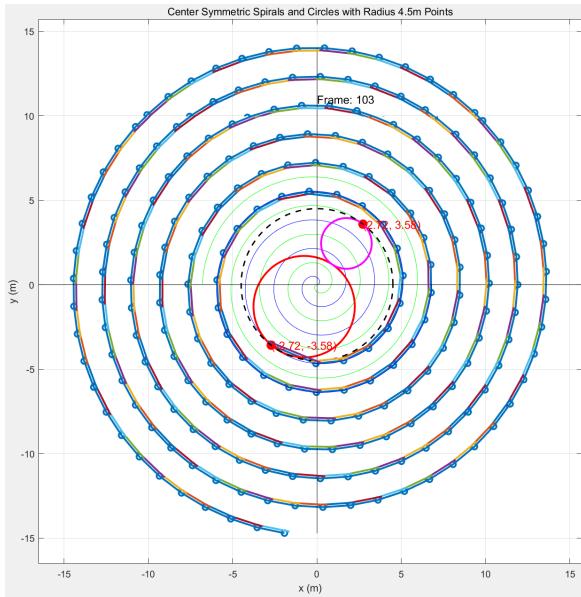
表五 部分把手部分时间速度信息

	-100 s	-50 s	0 s	50 s	100 s
龙头x (m)	7.778034	6.608301	-2.711856	1.332696	-3.157229
龙头y (m)	3.717164	1.898865	-3.591078	6.175324	7.548511
第1节龙身x (m)	6.209273	5.366911	-0.063534	3.862265	-0.346890
第1节龙身y (m)	6.108521	4.475403	-4.670888	4.840828	8.079166
第51节龙身x (m)	-10.025544	-5.079281	0.834188	-3.161956	0.549047
第51节龙身y (m)	4.375253	-8.175177	-8.059850	-5.369113	4.610352
第101节龙身x (m)	-12.406526	9.158729	4.532481	-6.506438	-6.614290
第101节龙身y (m)	-3.224889	-7.309146	9.476114	6.418248	3.569720
第151节龙身x (m)	-14.452392	12.383110	-5.560785	-6.042037	8.720119
第151节龙身y (m)	-0.334110	-5.350624	11.139433	-9.575879	-5.013907
第201节龙身x (m)	-10.777773	9.259818	-5.372540	-1.310306	9.586882
第201节龙身y (m)	11.725219	-11.869551	13.069337	-13.082537	7.344563
龙尾 (后) x (m)	-1.011059	0.189809	-1.933627	5.859094	-10.980157
龙尾 (后) y (m)	-16.527573	15.720588	-14.713128	12.612894	-6.770006

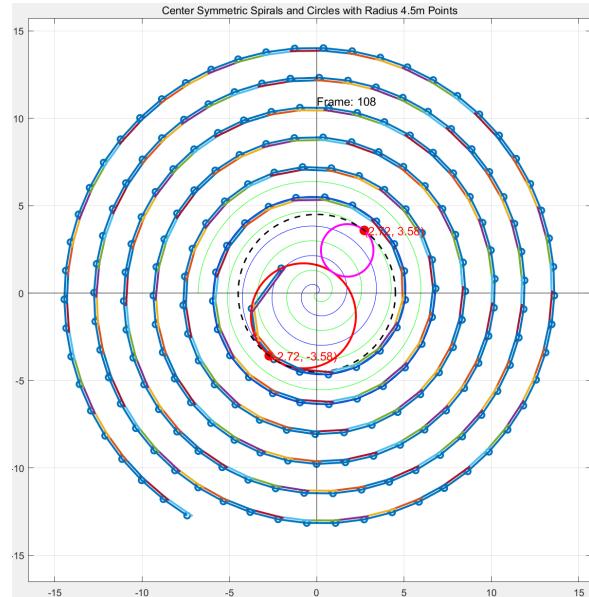
表六 部分把手部分时刻速度信息

5.4.8 模型的检验

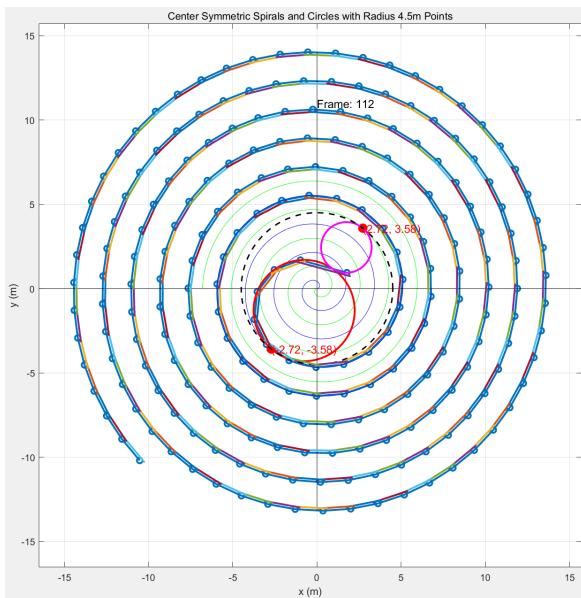
我们基于 matlab 编写了 animation_4.m 程序，该程序读取 result4.xlsx 的 sheet1 中各个时刻下各个把手的位置信息，并按照这些数据进行作图可视化舞龙队的行进场景。由于位置信息由 question_4_2.m 程序通过数值递推获得，如果 animation_4.m 的可视化结果大致符合理想状态下舞龙队的行进场景，那么就说明以上数值递推的算法是极其精确的。文件 animation_4.m 程序可以展示舞龙队的动态运动，以下是部分截图：



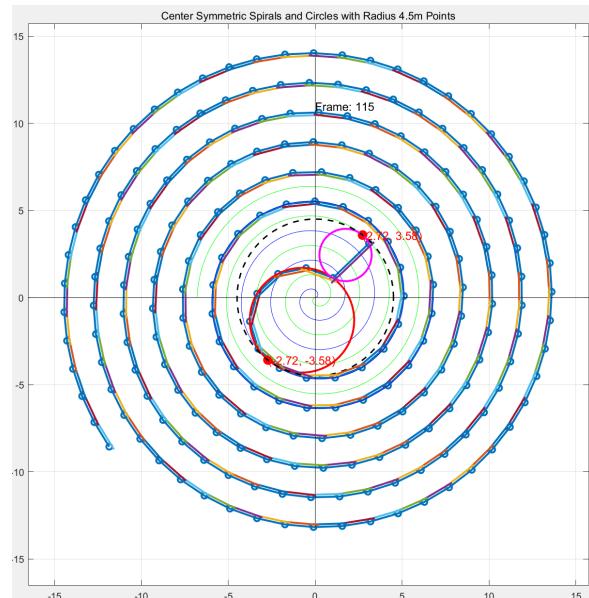
2s 时位置



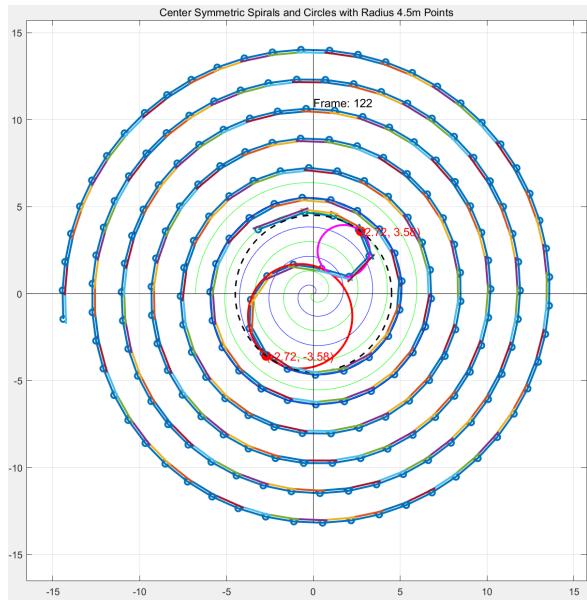
7s 时位置



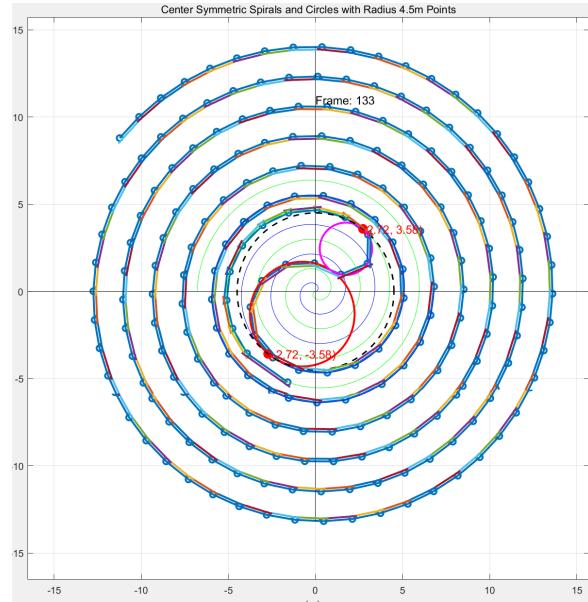
11s 时位置



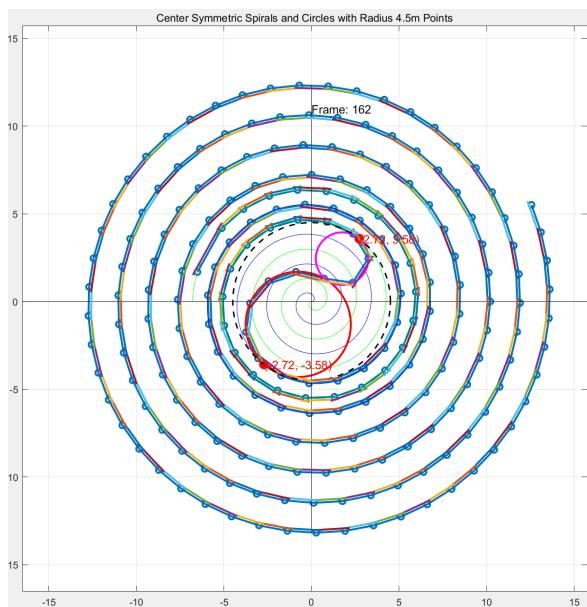
14s 时位置



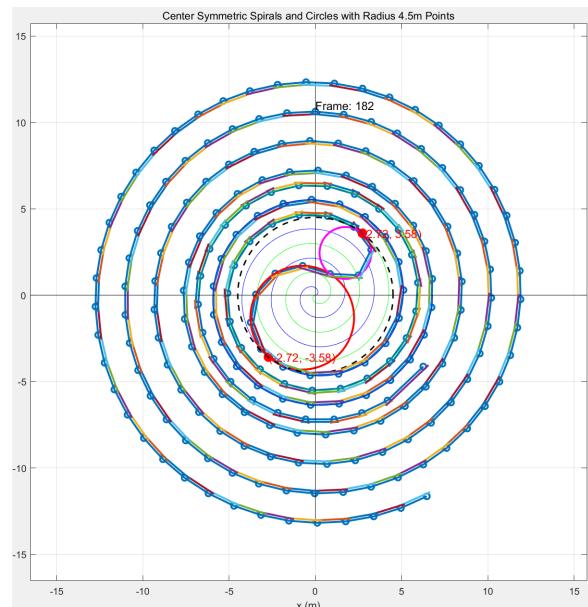
21s 时位置



32s 时位置

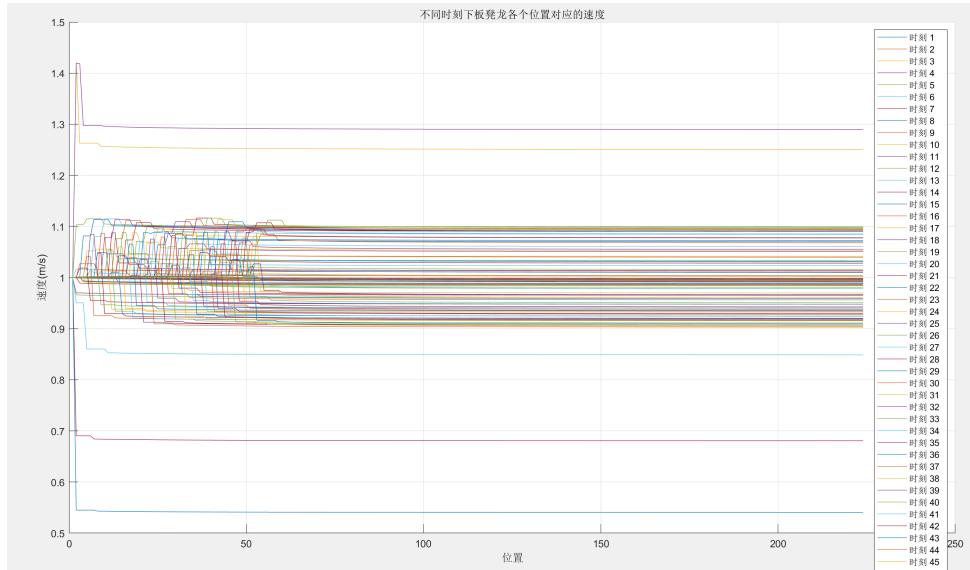


61s 时位置

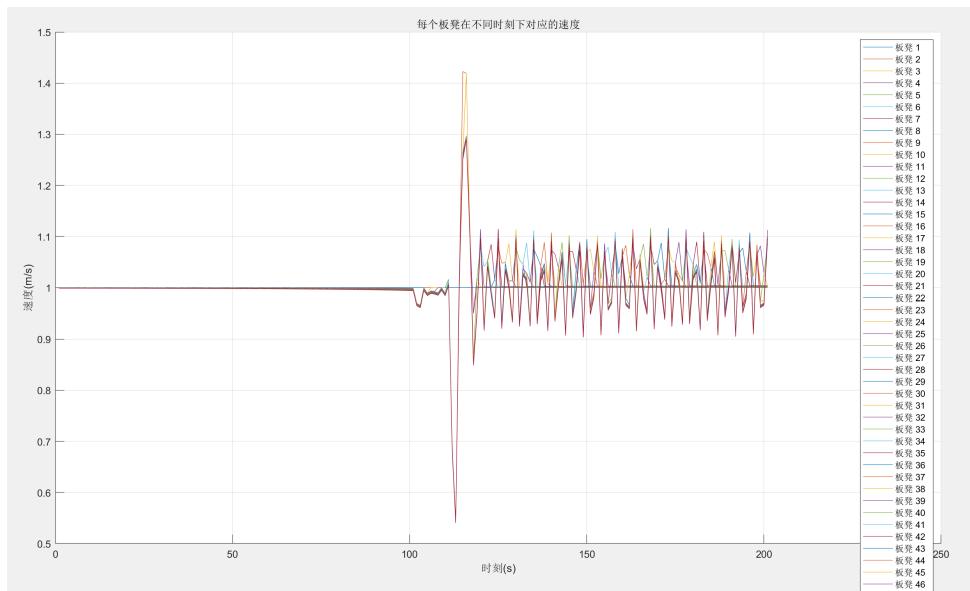


81s 时位置

如果能实际运行 animation_4.m 程序，一定能直观感受到所建立的模型是可靠的。如下的两幅可视化展示可以从两个维度集中展示板凳龙在行进时的特性



图六 不同时刻下各个位置对应的速度



图七 每个板凳在不同时刻下对应的速度

图六可以考察不同时刻板凳龙从前到后（对应图像从左到右）的速度变化趋势，不难发现不同时刻不同位置的板凳的速度都较为均匀地分布在 0.9m/s 和 1.1m/s 之间，而在某些时刻，个别把手和其它把手之间的速度相差较大，引起该现象的原因可能是舞龙队在切换路径时（例如从盘入螺线进入 S 型组合圆弧，或者是从 S 型组合圆弧进入盘出螺线）突入其来的几何特性改变（如曲率），使与龙头相连的板凳难以保持原速，受到龙头匀速前进的牵引而产生较大速度波动。

图七的横坐标应修正为 [-100:100]。我们发现，在龙头从盘入螺线进入 S 型组合圆弧段（0s 处，对应图中 100s 处），各个把手之间的速度相差不大，但当龙头从 S 型组合

圆弧进入盘出螺线段时（14s 处，对应图中 114s 处），各个把手之间的速度开始出现较大差异。这印证了我们对图六现象作出的猜测：因为盘入螺线与调转过程中半径较大的圆弧相连，盘出螺线与调转过程中半径较小的圆弧相连，而等距螺线和小圆圆弧（大曲率）的几何特性相差过大，使与龙头直接相连的板凳速度产生了较大波动，并将该波动依次向后传递给相连板凳，这导致了其它板凳在龙头盘出之后的时间里产生了参差不齐的速度波动趋势。

5.5 计算龙头的最大行进速度

根据龙头速度与龙身各把手速度之间的线性关系，可知在螺线的相同位置处，增大龙头速度，各个把手速度也会同比例增大。通过问题四，可以得到在龙头速度为 1m/s 条件下各个把手各个时刻的速度，其中的速度最大值便是整个行进过程中各把手所能达到的最大速度，且达到速度最大值的时刻和把手位置对于不同的龙头速度是相同的。故令该最大速度扩大 m 倍，使其达到 2m/s，对应的龙头速度 $m * 1m/s$ 即为龙头速度的最大值。

根据问题四的计算结果，易得速度的最大值出现在第 14 秒的第二个把手处，在龙头速度为 1m/s 的情况下最大值为 1.422567m/s。由此可得龙头速度最大值：

$$m = 2/1.422567$$

$$headspeed_{max} = m * 1 = 1.405909m/s$$

六、总结

模型优点：

1. 基于几何定理建立数学模型，真实准确反映舞龙队每节把手舞龙的轨迹、位置、速度等行进规律
2. 通过数值递推和二分逼近等方法求解位置信息，不失逻辑严密性的同时避免了繁复的解析推导，使得求解过程更易于理解，最后通过可视化分析证明了我们模型的正确性和可靠性
3. 提供了大量可视化方法，能直观地展现建模效果，使模型更易于使用

模型缺点：

1. 通过计算位置差与时间差得到速度，分母为极小量可能会造成数值计算相关问题
2. 在建模时忽略了大部分动力学因素，在实际的盘入盘出过程中，靠近龙尾处可能会由于惯性使得舞龙者难以保持稳定的前进状态。为防止板凳之间相互碰撞造成损伤或意

外，还需要在转弯幅度较大处预留一些额外的螺距，并相应地减慢龙头速度

参考文献

- [1] 司守奎, 孙玺菁, 数学建模算法与应用 [M]. 北京: 国防工业出版社, 2024
- [2] 李谋涛, 守正创新: 基于非物质文化遗产“徽州板凳龙”传承人的口述史考察 [J]. 通化师范学院学报, 2024, 45(03)
- [3] 陈纪修, 於崇华, 金路, 数学分析上册 [M]. 北京: 高等教育出版社, 2019

附录1 文件清单

```
question_1.m % 问题一 数值迭代与差分近似模型  
anamation_crash.m % 问题二 基于叉积法的碰撞检测模型  
min_p_recur.m % 问题三 最小螺距的二分法求解优化模型  
question_4_1.m % 问题四第一小问 非线性规划模型  
question_4_2.m % 问题四第二小问 迭代切换模型  
anamation_4 % 问题四动态可视化  
  
result1.xlsx % 问题一结果  
result1_500.xlsx % 问题一结果的拓展，用于问题二  
location_speed_for_writing.xlsx % 问题一结果（部分位置与速度）  
result2.xlsx % 问题二结果  
crush_location_loc_speed.xlsx % 问题二结果（撞击时刻位置与速度）  
animation_crash.xlsx % 问题三结果（检验）  
result4.xlsx % 问题四结果  
result4_for_writing.xlsx % 问题四结果（部分位置与速度）  
location_origin_5.xlsx % 问题四结果（位置）  
location_delta_5.xlsx % 问题四结果（偏移位置）  
speed_5.xlsx % 问题四结果，用于问题五
```

附录2 question_1.m

```
%% 基础作图  
% 参数设置  
r0 = 0; % 初始半径偏移  
p = 0.55; % 螺旋增长的速度  
theta_max = 32 * pi; % 螺旋的终止角度  
  
% 生成螺旋的角度和半径数据  
theta = linspace(0, theta_max, 1000);  
r = r0 + (p * theta) / (2 * pi);
```

```

% 转换为笛卡尔坐标系
x = r .* cos(theta);
y = r .* sin(theta);

% 绘图
figure;
plot(x, y);
axis equal; % 保持横纵坐标刻度一致
xlabel('x');
ylabel('y');
title('等距螺线');
grid on; % 显示网格

% 去掉横纵坐标上的数字
xticks([]);
yticks([]);

% 坐标轴
hold on;
line(xlim, [0 0], 'Color', 'k'); % X轴
line([0 0], ylim, 'Color', 'k'); % Y轴
hold off;

%% 获得所有位置
% 设置时间的步长

i = 1;

T = [0:i:500];
location = get_location(T);

% %% 获得所有速度
% % 设置时间的步长
%
% i = 1;

```

```

%
% T = [0:i:500];
% speed = get_speed(T);

%% 问题一论文写作提交格式的位置表格

% part_location = zeros(7*2, length(T));
%
% for i = 1:length(T)
%     part_location(1,i) = location(1,i);
%     part_location(2,i) = location(2,i);
%     part_location(3,i) = location(3,i);
%     part_location(4,i) = location(4,i);
%     part_location(5,i) = location(103,i);
%     part_location(6,i) = location(104,i);
%     part_location(7,i) = location(203,i);
%     part_location(8,i) = location(204,i);
%     part_location(9,i) = location(303,i);
%     part_location(10,i) = location(304,i);
%     part_location(11,i) = location(403,i);
%     part_location(12,i) = location(404,i);
%     part_location(13,i) = location(447,i);
%     part_location(14,i) = location(448,i);
%
% end

%
% %% 问题一论文写作提交格式的速度表格

% part_speed = zeros(7, length(T));
%
% for i = 1:length(T)
%     part_speed(1,i) = speed(1,i+1);
%     part_speed(2,i) = speed(2,i+1);
%     part_speed(3,i) = speed(52,i+1);
%     part_speed(4,i) = speed(102,i+1);
%     part_speed(5,i) = speed(152,i+1);

```

```

%      part_speed(6,i) = speed(202,i+1);
%      part_speed(7,i) = speed(224,i+1);
%
% end
%
% %% 问题二论文写作提交格式的位置和速度表格
%
% T = [414];
% location2 = get_location(T);
%
% speed2 = get_speed(T);
%
% result2 = zeros(224, 3);
%
% result2(:,3) = speed2(:,2);
%
% for i = 1:224
%     result2(i,1) = location2(2*i-1, 1);
%     result2(i,2) = location2(2*i, 1);
%
% end
%% 获得所有速度的函数
% r(theta) * (\delta(theta)) / (\delta(t)) = (\delta(l)) / (\delta(t))
function result = get_speed(T)

flag = 224;

location_theta = get_location_theta(T);

delta_t = 0.001;
delta_T = T + delta_t;

delta_location_theta = get_location_theta(delta_T);

d_location_theta = location_theta - delta_location_theta;

```

```

dtheta_dt = d_location_theta ./ delta_t;
r_theta = get_r( (0.5 .* T + 0.5 .* delta_T) );

speed = r_theta .* dtheta_dt;

for i = 1:flag
    speed(i, 1) = i - 1;
end

result = speed;
end

%% 获得所有极径的函数

function result = get_r(T)

a = 16 * 0.55;
b = 0.55 / (2 * pi);

location_theta = get_location_theta(T);
result = a + b .* location_theta;

end

%% 获得所有位置(单参数)的函数
function result = get_location_theta(T)

% 定义基本参数及函数
a = 16 * 0.55;
b = 0.55 / (2 * pi);

% 龙头板登前把手和后把手的固定距离
c0 = 3.41 - 2 * 0.275;
% 其余板登前把手和后把手的固定距离
c1 = 2.20 - 2 * 0.275;

```

```

% 龙头的线速度
v = 1;

% 定义迭代次数
flag = 224;

theta = zeros(flag, length(T)+1);

for i = 1:flag
    theta(i, 1) = i - 1;
end

for i = 2:(length(T)+1)
    theta(1, i) = get_theta0(v, T(i-1));
    theta(2, i) = location_rec(c0, theta(1, i));

    for j = 2:(flag-1)
        theta(j+1, i) = location_rec(c1, theta(j, i));
    end
end

result = theta;
end

%% 获得所有位置(直角坐标系)的函数

function result = get_location(T)

% 定义基本参数及函数
a = 16 * 0.55;
b = 0.55 / (2 * pi);

% 龙头板登前把手和后把手的固定距离

```

```

c0 = 3.41 - 2 * 0.275;
% 其余板登前把手和后把手的固定距离
c1 = 2.20 - 2 * 0.275;

% 龙头的线速度
v = 1;

% 定义迭代次数
flag = 224;

theta = zeros(flag, length(T)+1);

for i = 1:flag
    theta(i, 1) = i - 1;
end

for i = 2:(length(T)+1)
    theta(1, i) = get_theta0(v, T(i-1));
    theta(2, i) = location_rec(c0, theta(1, i));

    for j = 2:(flag-1)
        theta(j+1, i) = location_rec(c1, theta(j, i));
    end
end

all_location = zeros(flag*2, length(T));

for i = 1:length(T)
    for j = 1:flag
        [all_location(2*j-1, i), all_location(2*j, i)] = change(theta(j, i+1));
    end
end

result = all_location;

```

```
end
```

```
%% 计算经过时间t龙头的位置函数
```

```
function theta0 = get_theta0(v, t)
```

```
% 等距螺线基本参数
```

```
a = 16 * 0.55;
```

```
b = 0.55 / (2 * pi);
```

```
% 定义 r(theta)
```

```
r = @(theta) a + b * theta;
```

```
% 它的导数 dr/dtheta
```

```
dr_dtheta = @(theta) b;
```

```
% 龙头在t时间内走过的弧长
```

```
L = v * t;
```

```
% 定义被积函数
```

```
integrand = @(theta) sqrt((dr_dtheta(theta)).^2 + (r(theta)).^2);
```

```
% 定义积分方程：目标是求解 theta0，使得积分结果等于 v*t
```

```
integral_eq = @(theta0) integral(integrand, theta0, 0) - L;
```

```
% 使用 fsolve 求解 theta0
```

```
theta0_initial_guess = 0;
```

```
result = fzero(integral_eq, theta0_initial_guess);
```

```
theta0 = result;
```

```
end
```

```
%% 极坐标系转换成直角坐标系函数
```

```

function [x, y] = change(theta)

% 等距螺线基本参数
a = 16 * 0.55;
b = 0.55 / (2 * pi);

% 定义 r(theta)
r = @(theta) a + b * theta;

x = r(theta) * cos(theta);
y = r(theta) * sin(theta);

end

%% 通过距离约束得到位置递推函数

function [theta_new] = location_rec(c, theta)

% 等距螺线基本参数
a = 16 * 0.55;
b = 0.55 / (2 * pi);

% 定义 r(theta)
r = @(theta) a + b * theta;

% 计算已知 theta 对应的坐标
[x_known, y_known] = change(theta);

% 方程定义
eq_fun = @(vars) [
    (vars(2) * cos(vars(1)) - x_known)^2 + ...
    (vars(2) * sin(vars(1)) - y_known)^2 - c^2;
    vars(2) - (a + b * vars(1))
];

```

```

% 初始猜测
guess = [theta+0.1, r(theta)];

% 使用 fsolve 求解方程
options = optimoptions('fsolve', 'Display', 'off'); % 可选项
result = fsolve(eq_fun, guess, options);

% 输出结果
theta_new = result(1);

end

```

附录3 anamation_crash.m

```

% 读取 Excel 数据
filename = 'result1_500.xlsx'; % Excel 文件名
data = readmatrix(filename); % 读取 Excel 文件为矩阵

% 设置平移的距离 (half_width) 和延长长度 (extension)
half_width = 0.15; % 设定平移的距离，可以根据实际调整
extension = 0.275; % 设定延长长度

% 获取点的数量
num_points = size(data, 1) / 2; % 假设是2N行，每个点有 x 和 y 坐标

% 创建动画
figure;

for t = 1:size(data, 2) % 遍历每个时刻
    % 提取当前时刻的 x 和 y 坐标
    x = data(1:2:end, t); % 奇数行是 x 坐标
    y = data(2:2:end, t); % 偶数行是 y 坐标

```

```

clf; % 清除旧图形

% 原始线段绘制
plot(x, y, '-o', 'LineWidth', 2, 'MarkerSize', 6);
hold on;

% 添加显示时间的文本
time_str = ['Time: ', num2str(t), ' s'];
text(min(x), max(y), time_str, 'FontSize', 12, 'Color', 'r', 'FontWeight', 'bold');

EX1 = [];
EY1 = [];
EX2 = [];
EY2 = [];

% 遍历每一对相邻点，计算线段并检测碰撞
for i = num_points - 1:-1:1
    % 获取线段的端点
    x1 = x(i); y1 = y(i);
    x2 = x(i+1); y2 = y(i+1);

    % 计算线段方向
    dx = x2 - x1;
    dy = y2 - y1;
    length_original = sqrt(dx^2 + dy^2); % 线段长度
    unit_vec = [dx, dy] / length_original; % 方向单位向量

    % 计算垂直向量
    perp_vec = [-dy, dx]; % 垂直向量
    perp_unit_vec = perp_vec / norm(perp_vec); % 垂直单位向量

    % 计算平移后的点
    shift_x = half_width * perp_unit_vec(1);
    shift_y = half_width * perp_unit_vec(2);

```

```

% 平移后的线段
shifted_x1 = x1 + shift_x; shifted_y1 = y1 + shift_y;
shifted_x2 = x2 + shift_x; shifted_y2 = y2 + shift_y;

% 延长平移后的线段
extended_x1 = shifted_x1 - extension * unit_vec(1);
extended_y1 = shifted_y1 - extension * unit_vec(2);
extended_x2 = shifted_x2 + extension * unit_vec(1);
extended_y2 = shifted_y2 + extension * unit_vec(2);

EX1(end + 1) = extended_x1;
EY1(end + 1) = extended_y1;
EX2(end + 1) = extended_x2;
EY2(end + 1) = extended_y2;

% 绘制延长后的线段
plot([extended_x1, extended_x2], [extended_y1, extended_y2], 'LineWidth', 2);

% 对于首条线段m的检测，检查m与延长后的线段是否相交
if i == 1
    m_x1 = [x1, x2] - shift_x;
    m_y1 = [y1, y2] - shift_y;
    extended_m_x1 = m_x1(1) - extension * unit_vec(1);
    extended_m_y1 = m_y1(1) - extension * unit_vec(2);
    extended_m_x2 = m_x1(2) + extension * unit_vec(1);
    extended_m_y2 = m_y1(2) + extension * unit_vec(2);

    % 绘制延长后的线段
    plot([extended_m_x1, extended_m_x2], [extended_m_y1, extended_m_y2], 'Lin

    % 只检测平移并延长的线段
    for j = 2:num_points-1
        p1 = [EX1(j), EY1(j)];
        p2 = [EX2(j), EY2(j)];

```

```

        if check_intersection([extended_m_x1, extended_m_y1], [extended_m_x2,
            disp(['碰撞发生在时刻: t = ', num2str(t)]);
            return; % 停止动画
        end
    end
end

% 设置 x 和 y 轴比例相同
axis([min(data(:)) max(data(:)) min(data(:)) max(data(:))]); % 设置动态坐标轴范围
grid on;

% 控制动画速度
pause(0.1); % 暂停 0.1 秒
end

% 定义交点检测函数，考虑延长后的线段
function intersects = check_intersection(p1, p2, q1, q2)
    % 使用向量叉积法计算线段是否相交
    cross1 = (q1(1) - p1(1)) * (p2(2) - p1(2)) - (q1(2) - p1(2)) * (p2(1) - p1(1));
    cross2 = (q2(1) - p1(1)) * (p2(2) - p1(2)) - (q2(2) - p1(2)) * (p2(1) - p1(1));
    cross3 = (p1(1) - q1(1)) * (q2(2) - q1(2)) - (p1(2) - q1(2)) * (q2(1) - q1(1));
    cross4 = (p2(1) - q1(1)) * (q2(2) - q1(2)) - (p2(2) - q1(2)) * (q2(1) - q1(1));

    % 判断叉积符号是否相异，若相异则两线段相交
    intersects = (cross1 * cross2 < 0) && (cross3 * cross4 < 0);
end

```

附录4 min_p_recur.m

```

% 初始螺距参数
min_pitch = 0.43; % 最小螺距
max_pitch = 0.46; % 已知数据螺距为 0.55 cm

```

```

tolerance = 0.002; % 优化的精度
center_radius = 4.5; % 中心区域半径
half_width = 0.15; % 设定平移的距离
extension = 0.275; % 设定延长长度

% 优化螺距
while (max_pitch - min_pitch > tolerance)
    current_pitch = (min_pitch + max_pitch) / 2;

% 生成轨迹数据
data = get_trajectory(current_pitch); % 调用 get_trajectory 生成轨迹
collision_detected = false;

% 获取点的数量
num_points = size(data, 1) / 2;

% 遍历时间步骤
for t = 1:size(data, 2)
    x = data(1:2:end, t); % 奇数行是 x 坐标
    y = data(2:2:end, t); % 偶数行是 y 坐标

    % 清除旧图形
    clf;

    % 绘制原始线段
    plot(x, y, '-o', 'LineWidth', 2, 'MarkerSize', 6);
    hold on;

    EX1 = [];
    EY1 = [];
    EX2 = [];
    EY2 = [];

    % 遍历每一对相邻点，计算线段并检测碰撞
    for i = num_points - 1:-1:1

```

```

% 获取线段的端点
x1 = x(i); y1 = y(i);
x2 = x(i+1); y2 = y(i+1);

% 计算线段方向
dx = x2 - x1;
dy = y2 - y1;
length_original = sqrt(dx^2 + dy^2); % 线段长度
unit_vec = [dx, dy] / length_original; % 方向单位向量

% 计算垂直向量
perp_vec = [-dy, dx]; % 垂直向量
perp_unit_vec = perp_vec / norm(perp_vec); % 垂直单位向量

% 计算平移后的点
shift_x = half_width * perp_unit_vec(1);
shift_y = half_width * perp_unit_vec(2);

% 平移后的线段
shifted_x1 = x1 + shift_x; shifted_y1 = y1 + shift_y;
shifted_x2 = x2 + shift_x; shifted_y2 = y2 + shift_y;

% 延长平移后的线段
extended_x1 = shifted_x1 - extension * unit_vec(1);
extended_y1 = shifted_y1 - extension * unit_vec(2);
extended_x2 = shifted_x2 + extension * unit_vec(1);
extended_y2 = shifted_y2 + extension * unit_vec(2);

EX1(end + 1) = extended_x1;
EY1(end + 1) = extended_y1;
EX2(end + 1) = extended_x2;
EY2(end + 1) = extended_y2;

% 绘制延长后的线段
plot([extended_x1, extended_x2], [extended_y1, extended_y2], 'LineWidth',

```

```

% 对于首条线段m的检测，检查m与延长后的线段是否相交
if i == 1
    m_x1 = [x1, x2] - shift_x;
    m_y1 = [y1, y2] - shift_y;
    extended_m_x1 = m_x1(1) - extension * unit_vec(1);
    extended_m_y1 = m_y1(1) - extension * unit_vec(2);
    extended_m_x2 = m_x1(2) + extension * unit_vec(1);
    extended_m_y2 = m_y1(2) + extension * unit_vec(2);

% 绘制延长后的线段
plot([extended_m_x1, extended_m_x2], [extended_m_y1, extended_m_y2], 'r');

% 只检测平移并延长的线段
for j = 2:num_points-1
    p1 = [EX1(j), EY1(j)];
    p2 = [EX2(j), EY2(j)];

    if check_intersection([extended_m_x1, extended_m_y1], [extended_m_x2, extended_m_y2])
        disp(['碰撞发生在时间: t = ', num2str(t), ' 秒, 对应的螺距: ', num2str(current_pitch)]);
        collision_detected = true;
        break;
    end
end
end

% 检查龙头是否到达中心
if sqrt(x(1)^2 + y(1)^2) <= center_radius
    disp(['龙头到达中心, 距离: ', num2str(sqrt(x(1)^2 + y(1)^2))]);
    disp([' 对应的螺距: ', num2str(current_pitch)]);
    break;
end

if collision_detected

```

```

        break;
    end

end

% 调整螺距
if collision_detected
    min_pitch = current_pitch;
else
    max_pitch = current_pitch;
end

end

disp(['最小螺距为： ', num2str(current_pitch)]);

%% 轨迹生成函数 get_trajectory
function location = get_trajectory(current_pitch)
% 参数设置
T = 0:1:500; % 时间范围
location = get_location(T, current_pitch); % 调用 get_location 生成轨迹
end

%% get_location 函数
function result = get_location(T, current_pitch)
% 基础参数设置
a = 8.8;
b = current_pitch / (2 * pi);
c0 = 3.41 - 2 * 0.275; % 龙头板的前把手和后把手的固定距离
c1 = 2.20 - 2 * 0.275; % 其他板凳的前把手和后把手的固定距离
v = 1; % 线速度
flag = 224; % 龙身节数

% 计算角度 theta
theta = zeros(flag, length(T)+1);
for i = 2:(length(T)+1)
    theta(1, i) = get_theta0(v, T(i-1), current_pitch); % 龙头

```

```

theta(2, i) = location_rec(c0, theta(1, i), current_pitch); % 龙头板
for j = 2:(flag-1) % 其余板凳
    theta(j+1, i) = location_rec(c1, theta(j, i), current_pitch);
end
end

% 计算每个板凳的位置
all_location = zeros(flag*2, length(T));
for i = 1:length(T)
    for j = 1:flag
        [all_location(2*j-1, i), all_location(2*j, i)] = change(theta(j, i+1), cu
    end
end

result = all_location;
end

%% get_theta0 函数
function theta0 = get_theta0(v, t, current_pitch)
    % 计算龙头在 t 时刻的角度 theta0
    a = 8.8;
    b = current_pitch / (2 * pi);
    r = @(theta) a + b * theta;
    dr_dtheta = b;
    L = v * t; % 龙头走过的弧长
    integrand = @(theta) sqrt((dr_dtheta).^2 + (r(theta)).^2); % 计算弧长的积分函数
    integral_eq = @(theta0) integral(integrand, theta0, 0) - L;
    theta0 = fzero(integral_eq, 0); % 求解 theta0
end

%% 极坐标转换为直角坐标
function [x, y] = change(theta, current_pitch)
    a = 8.8;
    b = current_pitch / (2 * pi);
    r = a + b * theta;

```

```

x = r * cos(theta);
y = r * sin(theta);
end

%% 位置递推函数
function theta_new = location_rec(c, theta, current_pitch)
    % 根据当前点位置计算下一点的角度 theta_new
    a = 8.8;
    b = current_pitch / (2 * pi);
    r = @(theta) a + b * theta;

    [x_known, y_known] = change(theta, current_pitch);
    eq_fun = @(vars) [
        (vars(2) * cos(vars(1)) - x_known)^2 + ...
        (vars(2) * sin(vars(1)) - y_known)^2 - c^2;
        vars(2) - (a + b * vars(1))
    ];

    result = fsolve(eq_fun, [theta+0.1, r(theta)], optimoptions('fsolve', 'Display',
    theta_new = result(1);
end

%% 线段相交检测函数
function intersects = check_intersection(p1, p2, q1, q2)
    % 使用向量叉积法计算线段是否相交
    cross1 = (q1(1) - p1(1)) * (p2(2) - p1(2)) - (q1(2) - p1(2)) * (p2(1) - p1(1));
    cross2 = (q2(1) - p1(1)) * (p2(2) - p1(2)) - (q2(2) - p1(2)) * (p2(1) - p1(1));
    cross3 = (p1(1) - q1(1)) * (q2(2) - q1(2)) - (p1(2) - q1(2)) * (q2(1) - q1(1));
    cross4 = (p2(1) - q1(1)) * (q2(2) - q1(2)) - (p2(2) - q1(2)) * (q2(1) - q1(1));

    % 判断叉积符号是否相异，若相异则两线段相交
    intersects = (cross1 * cross2 < 0) && (cross3 * cross4 < 0);
end

```

附录5 question_4_1.m

%% 基本图形

```
r0 = 0;
p = 1.7;
theta_max = 32 * pi;
num_points = 1000;

theta = linspace(0, theta_max, num_points);

r1 = r0 + (p * theta) / (2 * pi);
r2 = r0 - (p * theta) / (2 * pi);

x1 = r1 .* cos(theta);
y1 = r1 .* sin(theta);

x2 = r2 .* cos(theta);
y2 = r2 .* sin(theta);

figure;
plot(x1, y1, 'b', 'LineWidth', 1);
hold on;
plot(x2, y2, 'r', 'LineWidth', 1);
axis equal;

xticks([]);
yticks([]);

xlabel('');
ylabel('');

legend off;
```

```

hold on;
line(xlim, [0 0], 'Color', 'k');
line([0 0], ylim, 'Color', 'k');

% 添加一个圆心在原点，半径为4.5米的圆
r_circle = 4.5;
theta_circle = linspace(0, 2*pi, num_points);
x_circle = r_circle * cos(theta_circle);
y_circle = r_circle * sin(theta_circle);
plot(x_circle, y_circle, 'k--', 'LineWidth', 1.5); % 绘制圆

hold off;

%% 求解进入掉头区域坐标

% 基本参数定义
r0 = 0;
p = 1.7;
r1_circle = 4.5;

r1 = @(theta) r0 + (p * theta) / (2 * pi);
r2 = @(theta) -r0 - (p * theta) / (2 * pi);

f1 = @(theta) r1(theta) - r1_circle;
f2 = @(theta) r2(theta) - r1_circle;

guess = 10;

theta0 = fzero(f1, guess);
theta3 = fzero(f2, -guess);

x0 = r1(theta0) * cos(theta0);
y0 = r1(theta0) * sin(theta0);

x3 = r2(-theta3) * cos(-theta3);

```

```

y3 = r2(-theta3) * sin(-theta3);

fprintf('螺线与圆相切点的坐标: (%f, %f), %f\n', x0, y0, theta0);
fprintf('螺线与圆相切点的坐标: (%f, %f), %f\n', x3, y3, theta3);

% 在图上标注相切点
hold on;
plot(x0, y0, 'ko', 'MarkerSize', 10, 'LineWidth', 1);
plot(x3, y3, 'ko', 'MarkerSize', 10, 'LineWidth', 1);
hold off;

%% 主程序
% 定义初始猜测值
xmin = [-1, 1, -1, 1, 1.5, 3*pi/4, 3*pi/4]; % [x1, x2, y1, y2, R, theta1, theta2] 初如

% 调用 fmincon 优化
options = optimoptions('fmincon', 'Display','iter', 'Algorithm', 'sqp');

% 最小化 2 * R * theta1 + R * theta2
[x_opt, fval] = fmincon(@objective, xmin, [], [], [], [], [], [], @constraints, options);

% 输出最优结果
disp('最优解:');
disp(x_opt);
disp('最小化目标函数的值:');
disp(fval);

%% 进一步作图

hold on;

x1 = x_opt(1);
x2 = x_opt(2);
y1 = x_opt(3);

```

```

y2 = x_opt(4);
R = x_opt(5);
theta1 = x_opt(6);
theta2 = x_opt(7);

num_points = 1000;
theta_circle = linspace(0, 2*pi, num_points);

r1_circle = 2 * R;
x1_circle = x1 + r1_circle * cos(theta_circle);
y1_circle = y1 + r1_circle * sin(theta_circle);
plot(x1_circle, y1_circle, 'k--', 'LineWidth', 1.5);

r2_circle = R;
x2_circle = x2 + r2_circle * cos(theta_circle);
y2_circle = y2 + r2_circle * sin(theta_circle);
plot(x2_circle, y2_circle, 'k--', 'LineWidth', 1.5);

plot([x0, x1], [y0, y1], 'k--', 'LineWidth', 1.5);
plot([x2, x1], [y2, y1], 'k--', 'LineWidth', 1.5);
plot([x3, x2], [y3, y2], 'k--', 'LineWidth', 1.5);

hold off;

%% 目标函数
function obj = objective(x)
    R = x(5);
    theta1 = x(6);
    theta2 = x(7);

    % 目标函数: 2 * R * theta1 + R * theta2
    obj = 2 * R * theta1 + R * theta2;
end

%% 约束方程

```

```

function [c, ceq] = constraints(x)
    x1 = x(1);
    x2 = x(2);
    y1 = x(3);
    y2 = x(4);
    R = x(5);
    theta1 = x(6);
    theta2 = x(7);
    x0 = -2.711855863706671;
    y0 = -3.591077522761064;
    theta0 = 16.631961107240077;
    p = 1.7;

    % k1(theta0) 函数定义
    k1 = @(theta) ( (p / (2 * pi)) * sin(theta) + (p * theta) / (2 * pi) * cos(theta)
                    ( (p / (2 * pi)) * cos(theta) - (p * theta) / (2 * pi) * sin(theta))

    % 约束1: (x1 - x0)^2 + (y1 - y0)^2 = (2 * R)^2
    ceq(1) = (x1 - x0)^2 + (y1 - y0)^2 - (2 * R)^2;

    % 约束2: (x2 + x0)^2 + (y2 + y0)^2 = R^2
    ceq(2) = (x2 + x0)^2 + (y2 + y0)^2 - R^2;

    % 约束3: [1, k1(theta0)] .* [x0 - x1, y0 - y1] = 0
    ceq(3) = [1, k1(theta0)] * [x0 - x1; y0 - y1];

    % 约束4: cos(theta1) = ([x0-x1, y0-y1] .* [x2-x1, y2-y1]) / (norm([x0-x1, y0-y1]))
    ceq(4) = cos(theta1) - dot([x0 - x1, y0 - y1], [x2 - x1, y2 - y1]) / ...
              (norm([x0 - x1, y0 - y1]) * norm([x2 - x1, y2 - y1]));

    % 约束5: cos(theta2) = ([-x0-x2, -y0-y2] .* [x1-x2, y1-y2]) / (norm([-x0-x2, -y0-y2]))
    ceq(5) = cos(theta2) - dot([-x0 - x2, -y0 - y2], [x1 - x2, y1 - y2]) / ...
              (norm([-x0 - x2, -y0 - y2]) * norm([x1 - x2, y1 - y2]));

    ceq(6) = (x1 - x2)^2 + (y1 - y2)^2 - (3 * R)^2;

```

```

ceq(7) = [1, k1(theta0)] * [x0 + x2; y0 + y2];

% 不等式约束为空
c = [];

end

```

附录6 question_4_2.m

```

clc, clear

% 基础参数设置
A = 0;
B = 1.7 / (2 * pi);

r_in = @(theta) A + B * theta;
r_out = @(theta) A - B * theta;

L = 13.621245;
v = 1;

loc = zeros(224 * 2, 200);

% 龙头随时间迭代
T1 = linspace(-100, -1, 100);
theta1 = get_theta1(v, T1);
for i = 1:length(theta1)
    loc(1, i) = r_in(theta1(i)) * cos(theta1(i));
    loc(2, i) = r_in(theta1(i)) * sin(theta1(i));
end

T = floor(L / v);

```

```

T2 = linspace(0, T, T + 1);
for i = 1:length(T2)
    [result_x, result_y] = get_location_stage_s(v, T2(i));
    loc(1, 100 + i) = result_x;
    loc(2, 100 + i) = result_y;
end

T3 = linspace(T + 1, 100, 100 - T);
for i = 1:length(T3)
    theta2 = get_theta2(v, T3(i));
    loc(1, 101 + T + i) = r_out(theta2) * cos(theta2);
    loc(2, 101 + T + i) = r_out(theta2) * sin(theta2);
end

% 龙身随空间迭代
for t = 1:201
    loc = iterative_process(loc(1, t), loc(2, t), t, loc);
end

%% 求速度
delta_loc = zeros(224 * 2, 200);
speed = zeros(224*2, 200);

delta_t = 0.001;
delta_T1 = linspace(-100 + delta_t, -1 + delta_t, 100);
theta1 = get_theta1(v, delta_T1);
for i = 1:length(theta1)
    delta_loc(1, i) = r_in(theta1(i)) * cos(theta1(i));
    delta_loc(2, i) = r_in(theta1(i)) * sin(theta1(i));
end

T = 13;
delta_T2 = linspace(0 + delta_t, T + delta_t, T + 1);
for i = 1:length(delta_T2)
    [result_x, result_y] = get_location_stage_s(v, delta_T2(i));

```

```

delta_loc(1, 100 + i) = result_x;
delta_loc(2, 100 + i) = result_y;
end

delta_T3 = linspace(T + 1 + delta_t, 100 + delta_t, 100 - T);
for i = 1:length(delta_T3)
    theta2 = get_theta2(v, delta_T3(i));
    delta_loc(1, 101 + T + i) = r_out(theta2) * cos(theta2);
    delta_loc(2, 101 + T + i) = r_out(theta2) * sin(theta2);
end

for t = 1:201
    delta_loc = iterative_process(delta_loc(1, t), delta_loc(2, t), t, delta_loc);
end

[rows, cols] = size(loc);
for i = 1:rows
    for j = 1:cols
        speed(i, j) = (delta_loc(i, j) - loc(i, j)) / delta_t;
    end
end

[m, n] = size(speed);

speed_magnitudes = zeros(m/2, n);

% 计算速度大小
for col = 1:n
    vx = speed(1:2:end, col); % 获取 x 方向速度 (奇数行)
    vy = speed(2:2:end, col); % 获取 y 方向速度 (偶数行)

    % 速度大小 = sqrt(vx^2 + vy^2)
    speed_magnitudes(:, col) = sqrt(vx.^2 + vy.^2);
end

```

```

%% 不同时刻的龙头迭代
%% 龙头在 -100-0 时间内在第一段等距螺线位置函数
function x = get_theta1(v, t)
% 等螺线基本参数
b = 1.7 / (2 * pi);
theta0 = 16.631961107240077;
r = @(theta) b * theta;
dr_dtheta = @(theta) b;
%龙头在t时间内走过的弧长
for i = 1:length(t)
    L = -v .* t(i);
    % 定义被积函数
    integrand = @(theta) sqrt((dr_dtheta(theta)).^2 + (r(theta)).^2);
    integral_eq = @(x) integral(integrand, theta0, x) - L;
    % 使用 fsolve 求解 theta0
    theta0_initial_guess = 16*pi;
    result = fzero(integral_eq, theta0_initial_guess);
    x(1, i) = result;
end
end

%% 龙头在 0-T 时间内在S形圆弧的直角坐标位置函数
function [result_x, result_y] = get_location_stage_s(v, t)
x0 = -2.711855863706671;
y0 = -3.591077522761064;
x1 = -0.760009116655550;
y1 = -1.305726426346273;
x2 = 1.735932490181134;
y2 = 2.448401974553573;
% l <= l_long + l_short
l = v * t;
R = 1.502708833894531;
theta = 3.021486841980670;
l_long = 2 * R * theta;
l_short = R * theta;
if l <= l_long

```

```

delta_theta = l / l_long * theta;
% 顺时针转动delta_theta
Location = [x1; y1] + [cos(delta_theta), -sin(-delta_theta); sin(-delta_theta), cos(d
result_x = Location(1);
result_y = Location(2);
else
l_extra = l - l_long;
delta_theta = theta - l_extra / l_short * theta;
% 顺时针转动delta_theta
Location = [x2; y2] + [cos(delta_theta), -sin(-delta_theta); sin(-delta_theta), cos(d
result_x = Location(1);
result_y = Location(2);
end
end

%% 龙头在 T-100 时间内在第二段等距螺线位置函数
function theta0 = get_theta2(v, t)
% 等螺线基本参数
b = 1.7 / (2 * pi);
thetak = 16.631961107240077+pi; % 给定的终点角度
T = 13.621244906821248;
% 定义 r(theta) 螺旋线的半径
r = @(theta) b * (theta-pi);
% 定义 r(theta) 的导数 dr/dtheta
dr_dtheta = @(theta) b;
% 龙头在 t 时间内走过的弧长 L (负号表示从外向内运动)
L = v * (t-T);
% 定义被积函数，用于计算弧长
integrand = @(theta) sqrt((dr_dtheta(theta)).^2 + (r(theta)).^2);
% 定义积分方程，目标是求解 theta0
integral_eq = @(x) integral(integrand, thetak, x) - L;
% 使用 fzero 求解 theta0
theta0_initial_guess = 0; % 初始猜测点
result = fzero(integral_eq, theta0_initial_guess);
% 计算出的 theta0
theta0 = result - pi;

```

```

end

%% 不同位置的龙身迭代
%% 逆螺线迭代函数1

function [d, e, exitflag] = spiral_spiral1(c, a, b, a_last, b_last)
exitflag = 1;
% 等距螺线基本参数
A = 0;
B = 1.7 / (2 * pi);

eqn = @(var)[
    (var(1) * cos(var(2)) - a)^2 + (var(1) * sin(var(2)) - b)^2 - c^2;
    var(1) - A - (-B * var(2));
];

% 初始猜测值
guess = [-sqrt(a^2 + b^2), -sqrt(a^2 + b^2) / (-B) + 1];

% 创建 fsolve 的选项
options = optimoptions('fsolve', 'Display', 'off'); % 关闭显示

% 调用 fsolve 进行求解
result = fsolve(eqn, guess, options);

% 结果
d_temp = result(1) * cos(result(2));
e_temp = result(1) * sin(result(2));

% 检查是否需要调整解
if [d_temp - a, e_temp - b] * [a_last - a, b_last - b].' < 0
    % "option A"
    d = d_temp;
    e = e_temp;
else
    % "option B"

```

```

% 更新猜测值， 并再次求解
% guess = [a + c, b + c];
guess = [-sqrt(a^2 + b^2), -sqrt(a^2 + b^2) / (-B) - 1];
result = fsolve(eqn, guess, options);
d = result(1) * cos(result(2));
e = result(1) * sin(result(2));

end
hold on;
plot(d, e);

% 如果解的模小于等于 4.5， 设置 exitflag 为 -1
if sqrt(d^2 + e^2) <= 4.5
    sqrt(d^2 + e^2);
    exitflag = -1;
end

function x = check_spiral(exitflag)
if exitflag <= 0
    fprintf(" option 1 ")
    x = 0;
else
    x = 1;
end
end

%% 内部圆弧迭代函数
function [d,e,exitflag] = smallcircle_smallcircle(c,a,b,a_last,b_last)
exitflag = 1;
% 已知坐标
x0 = -2.711855863706671;
y0 = -3.591077522761064;
x1 = -0.760009116655550;
y1 = -1.305726426346273;
x2 = 1.735932490181134;
y2 = 2.448401974553573 ;

```

```

r = 1.502708833894531;

% 定义方程组，注意这里的 vars 是一个向量 [x, y]
eq_fun = @(vars) [
    (a - vars(1))^2 + (b - vars(2))^2 - c^2;
    (vars(1) - x2)^2 + (vars(2) - y2)^2 - r^2;
];

```

% 初始猜测

```

guess = [a - c, b - c];

```

% 优化选项

```

options = optimoptions('lsqnonlin', 'Display', 'off'); % 使用 lsqnonlin 的选项
% 如果没有上下限，可以设为 []
lb = [0.90395, -inf];
ub = [inf, inf];

```

% 使用 lsqnonlin 求解

```

[result, ~, ~, exitflag] = lsqnonlin(eq_fun, guess, lb, ub, options);

```

% 结果

```

d_temp = result(1);
e_temp = result(2);

% size([d_temp - a, e_temp - b])
% size([a_last - a, b_last - b])
if [d_temp - a, e_temp - b] * [a_last - a, b_last - b].' < 0
    % "option A"
    % [d_temp - a, e_temp - b] * [a_last - a, b_last - b].'
    d = d_temp;
    e = e_temp;
else
    % "option B"
    % [d_temp - a, e_temp - b] * [a_last - a, b_last - b].'
    guess = [a + c, b + c];
    % result = fsolve(eq_fun, guess, options);

```

```

[result, ~, ~, exitflag] = lsqnonlin(eq_fun, guess, lb, ub, options);
d = result(1);
e = result(2);
% [d - a, e - b] * [a_last - a, b_last - b].'

if d == d_temp && e == e_temp
    exitflag = -1;
end
end

function [d,e,exitflag] = bigcircle_bigcircle(c,a,b,a_last,b_last)
% 已知坐标
exitflag = 1;
x0 = -2.711855863706671;
y0 = -3.591077522761064;
x1 = -0.760009116655550;
y1 = -1.305726426346273;
x2 = 1.735932490181134;
y2 = 2.448401974553573 ;
r = 1.502708833894531 * 2; % 大圆的半径

% 定义方程组，注意这里的 vars 是一个向量 [x, y]
eq_fun = @(vars) [
    (a - vars(1))^2 + (b - vars(2))^2 - c^2;
    (vars(1) - x1)^2 + (vars(2) - y1)^2 - r^2;
];
guess = [a+c, b+c];

% 优化选项
options = optimoptions('lsqnonlin', 'Display', 'off'); % 使用 lsqnonlin 的选项
% 如果没有上下限，可以设为 []
lb = [-4, -inf];

```

```

ub = [0.90395, inf];
% 使用 lsqnonlin 求解
[result, ~, ~, exitflag] = lsqnonlin(eq_fun, guess, lb, ub, options);

% 结果
d_temp = result(1);
e_temp = result(2);

% size([d_temp - a, e_temp - b])
% size([a_last - a, b_last - b])
if [d_temp - a, e_temp - b] * [a_last - a, b_last - b].' < 0
    d = d_temp;
    e = e_temp;
else
    guess = [a-c, b-c];
    % result = fsolve(eq_fun, guess, options);
    [result, ~, ~, exitflag] = lsqnonlin(eq_fun, guess, lb, ub, options);
    d = result(1);
    e = result(2);
    if d == d_temp && e == e_temp
        exitflag = -1;
    end
end
end

function [x, check_degree] = check_small(check_degree,c,d,e,exitflag)
x0 = -2.711855863706671;
y0 = -3.591077522761064;
x1 = -0.760009116655550;
y1 = -1.305726426346273;
x2 = 1.735932490181134;
y2 = 2.448401974553573 ;
r = 1.502708833894531;
theta1 = 3.021486841980670;

```

```

delta_degree = 2 * asin(c / (2 * r));
if check_degree == -114514
    check_degree = acos( ((d-x2)*(-x0-x2)+(e-y2)*(-y0-y2))/(r*r));
else
    check_degree = check_degree + delta_degree;
end

if exitflag<=0
    fprintf(" option 1 ");
    x=0;
    return;
end

if check_degree > theta1
    fprintf(" option 2 ");
    x=0;
    return;
end
x=1;
end

function [x, check_degree] = check_big(check_degree,c,d,e,exitflag)
x0 = -2.711855863706671;
y0 = -3.591077522761064;
x1 = -0.760009116655550;
y1 = -1.305726426346273;
x2 = 1.735932490181134;
y2 = 2.448401974553573 ;
r = 1.502708833894531 * 2;
theta1 = 3.021486841980670;

delta_degree = 2 * asin(c / (2 * r));
if check_degree == -114514
    check_degree = theta1 - acos( ((d-x1)*(x0-x1)+(e-y1)*(y0-y1))/(r*r));
else

```

```

check_degree = check_degree + delta_degree;
end

if exitflag<=0
    fprintf(" option 1\n");
    x=0;
    return;
end

if check_degree > theta1
    fprintf(" option 2 ");
    x=0;
    return;
end
x=1;
end

%% 螺线迭代函数2
function [d, e, exitflag] = spiral_spiral2(c, a, b, a_last, b_last)
    exitflag = 1;
    % 等距螺线基本参数
    A = 0;
    B = 1.7 / (2 * pi);

    eqn = @(var)[
        (var(1) * cos(var(2)) - a)^2 + (var(1) * sin(var(2)) - b)^2 - c^2;
        var(1) - A - (B * var(2));
    ];

    % 初始猜测值
    guess = [sqrt(a^2 + b^2), sqrt(a^2 + b^2) / (B - 1)];

    % 创建 fsolve 的选项
    options = optimoptions('fsolve', 'Display', 'off'); % 关闭显示

```

```

% 调用 fsolve 进行求解
result = fsolve(eqn, guess, options);
% 结果
d_temp = result(1) * cos(result(2));
e_temp = result(1) * sin(result(2));

% 检查是否需要调整解
if [d_temp - a, e_temp - b] * [a_last - a, b_last - b].' < 0
    % "option A"
    d = d_temp;
    e = e_temp;
else
    % "option B"
    % 更新猜测值，并再次求解

        guess = [sqrt(a^2 + b^2), sqrt(a^2 + b^2) / (B) + 1];
        result = fsolve(eqn, guess, options);
        d = result(1) * cos(result(2));
        e = result(1) * sin(result(2));
end
hold on;
plot(d, e);

% 如果解的模小于等于 4.5，设置 exitflag 为 -1
if sqrt(d^2 + e^2) <= 4.5
    sqrt(d^2 + e^2);
    exitflag = -1;
end
end

%% 整体迭代过程
function loc = iterative_process(a, b, t, loc_in)
max_iterations = 223; % 设置最大迭代次数防止无限循环
% 设定初始默认变量值
a_last = a;

```

```

b_last = b;
check_degree = -114514;

R0 = sqrt(a^2 + b^2);

if t <= 103
    use_spiral1 = false;
    use_smallcircle = false;
    use_bigcircle = false;
    use_spiral2 = true;
elseif t <= 113
    use_spiral1 = false;
    use_smallcircle = false;
    use_bigcircle = true;
    use_spiral2 = true;
elseif t <= 118
    use_spiral1 = false;
    use_smallcircle = true;
    use_bigcircle = true;
    use_spiral2 = true;
else
    use_spiral1 = true;
    use_smallcircle = true;
    use_bigcircle = true;
    use_spiral2 = true;
end

for iter = 1:max_iterations
    c = 1.65;
    if iter == 1
        c = 2.86;
    end

    if use_spiral1

```

```

[d, e, exitflag] = spiral_spiral1(c, a, b, a_last, b_last);

% 检查 smallcircle_smallcircle 结果
x = check_spiral(exitflag);

if x == 0
    [d, e, exitflag] = smallcircle_smallcircle(c, a, b, a_last, b_last);
    use_spiral1 = false;
end

% 打印当前迭代结果
fprintf('Iteration %d (Spiral1): d = %.6f, e = %.6f, x = %d\n', iter, d,
plot(d, e, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'b');

a_last = a;
b_last = b;
a = d;
b = e;

elseif use_smallcircle
    % 使用 smallcircle_smallcircle 函数
    [d, e, exitflag] = smallcircle_smallcircle(c, a, b, a_last, b_last);
    % 检查 smallcircle_smallcircle 结果
    [x, check_degree] = check_small(check_degree, c, d, e, exitflag);

    if x == 0
        [d, e, exitflag] = bigcircle_bigcircle(c, a, b, a_last, b_last);
        check_degree = -114514;
        use_smallcircle = false;
    end

    % 打印当前迭代结果
    fprintf('Iteration %d (Smallcircle): d = %.6f, e = %.6f, x = %d\n', iter,
plot(d, e, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'b');

```

```

a_last = a;
b_last = b;
a = d;
b = e;

elseif use_bigcircle
    % 使用 bigcircle_bigcircle 函数，不再进行 check
    [d, e, exitflag] = bigcircle_bigcircle(c, a, b, a_last, b_last);
    % 检查 bigcircle_bigcircle 结果
    [x, check_degree] = check_big(check_degree, c, d, e, exitflag);

if x == 0
    [d, e, exitflag] = spiral_spiral2(c, a, b, a_last, b_last);
    check_degree = -114514;
    use_bigcircle = false;
end

% 打印 bigcircle 阶段的迭代结果
fprintf('Iteration %d (Bigcircle): d = %.6f, e = %.6f, x = %d\n', iter, d
plot(d, e, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'b');

% 更新 a, b, 继续 bigcircle_bigcircle 迭代
a_last = a;
b_last = b;
a = d;
b = e;

elseif use_spiral2
    [d, e, exitflag] = spiral_spiral2(c, a, b, a_last, b_last);

    x = check_spiral(exitflag);

if x == 0
    [d, e, exitflag] = smallcircle_smallcircle(c, a, b, a_last, b_last);
    % use_spiral2 = false;

```

```

    end

    % 打印当前迭代结果
    fprintf('Iteration %d (Spiral2): d = %.6f, e = %.6f, x = %d\n', iter, d,
    plot(d, e, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'b');

    a_last = a;
    b_last = b;
    a = d;
    b = e;

    end

    loc_in(iter * 2 + 1, t) = d;
    loc_in(iter * 2 + 2, t) = e;
    loc = loc_in;
    % 可根据需求添加终止条件
end

%% 坐标系转化函数(等距螺线上)

function [x, y] = theta2loc1(theta, a, b)

    % % 等距螺线基本参数
    % a = 16 * 0.55;
    % b = 0.55 / (2 * pi);

    % 定义 r(theta)
    r = @(theta) a + b * theta;

    x = r(theta) * cos(theta);
    y = r(theta) * sin(theta);

end

```

```

function theta = loc2theta1(loc, a, b)
    r = sqrt(loc(1)^2 + loc(2)^2);
    theta = (r - a) / (-b);
end

function theta = loc2theta2(loc, a, b)
    r = sqrt(loc(1)^2 + loc(2)^2);
    theta = (r - a) / (b);
end

function [r, theta] = loc2polar(x, y)
    % 计算 r
    r = sqrt(x^2 + y^2);

    % 计算 theta (弧度)
    theta = atan2(y, x);
end

%% 绘图函数
function draw(X0, Y0)
% 螺距设置
a = 1.7/(2*pi); % 螺距
theta = linspace(0, 8 * pi, 1000); % 角度范围

% 两条中心对称的等距螺旋线
r_in = a * theta; % 顺时针盘入螺线
r_out = -a * theta; % 逆时针盘出螺线 (中心对称)

% 将极坐标转换为直角坐标
x_in = r_in .* cos(theta);
y_in = r_in .* sin(theta);

x_out = r_out .* cos(theta);
y_out = r_out .* sin(theta);

```

```

% 找到半径为4.5m的点
radius_target = 4.5;
[~, idx_in] = min(abs(r_in - radius_target));
[~, idx_out] = min(abs(r_out + radius_target));

% 对应半径为4.5m的点的坐标
x_in_4_5 = x_in(idx_in);
y_in_4_5 = y_in(idx_in);
x_out_4_5 = x_out(idx_out);
y_out_4_5 = y_out(idx_out);

% 圆弧参数
a1=3.2611;
a2= 3.0221;
r= 1.5014;
x1= -0.7639;
x2= 1.7420;
y1= -1.3018;
y2= 2.4409;

% 创建大圆和小圆的参数
theta_circle = linspace(0, 2*pi, 1000); % 完整圆的角度范围

x_circle1 = x1 +2 *r * cos(theta_circle); % 大圆的x坐标
y_circle1 = y1 + 2*r * sin(theta_circle); % 大圆的y坐标

x_circle2 = x2 + r * cos(theta_circle); % 小圆的x坐标
y_circle2 = y2 + r * sin(theta_circle); % 小圆的y坐标

% 绘制螺旋线
figure;
hold on;
plot(x_in, y_in, 'b', 'DisplayName', 'Clockwise Spiral (盘入螺线)');
plot(x_out, y_out, 'g', 'DisplayName', 'Counterclockwise Spiral (盘出螺线)');

```

```

% 标记半径为4.5m的点
scatter(x_in_4_5, y_in_4_5, 100, 'filled', 'r', 'DisplayName', 'Radius 4.5m Point (In)
scatter(x_out_4_5, y_out_4_5, 100, 'filled', 'r', 'DisplayName', 'Radius 4.5m Point (Out)

% 标注半径为4.5m的点
text(x_in_4_5, y_in_4_5, sprintf('%.2f, %.2f)', x_in_4_5, y_in_4_5), 'Color', 'r', 'FontSize', 12);
text(x_out_4_5, y_out_4_5, sprintf('%.2f, %.2f)', x_out_4_5, y_out_4_5), 'Color', 'r', 'FontSize', 12);

% 绘制完整的大圆和小圆
plot(x_circle1, y_circle1, 'r', 'LineWidth', 2, 'DisplayName', 'Large Circle');
plot(x_circle2, y_circle2, 'm', 'LineWidth', 2, 'DisplayName', 'Small Circle');

% 添加图例和标签
title('Center Symmetric Spirals and Circles with Radius 4.5m Points');
xlabel('x (m)');
ylabel('y (m)');
% legend('show');
axis equal;
grid on;

hold on;
line(xlim, [0 0], 'Color', 'k');
line([0 0], ylim, 'Color', 'k');

% 添加一个圆心在原点，半径为4.5米的圆
r1_circle = 4.5;
theta_circle = linspace(0, 2*pi, 1000);
x1_circle = r1_circle * cos(theta_circle);
y1_circle = r1_circle * sin(theta_circle);
plot(x1_circle, y1_circle, 'k--', 'LineWidth', 1.5); % 绘制圆

% 坐标轴
hold on;
line(xlim, [0 0], 'Color', 'k'); % X轴

```

```

line([0 0], ylim, 'Color', 'k'); % Y轴

plot(X0, Y0, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'b');

iterative_process(1.65,X0,Y0);

hold off;
grid on; % 显示网格
end

```

附录7 anamation_4

```

% 读取 Excel 数据
filename = 'result4.xlsx'; % Excel 文件名
data = readmatrix(filename); % 读取 Excel 文件为矩阵

% 设置平移的距离 (half_width) 和延长长度 (extension)
half_width = 0.15; % 设定平移的距离，可以根据实际调整
extension = 0.275; % 设定延长长度

% 获取点的数量
global num_points;
num_points = size(data, 1) / 2; % 假设是2N行，每个点有 x 和 y 坐标

% 创建动画
figure;

for t = 90:size(data, 2) % 遍历每个时刻
    % 提取当前时刻的 x 和 y 坐标
    x = data(1:2:end, t); % 奇数行是 x 坐标
    y = data(2:2:end, t); % 偶数行是 y 坐标

```

```

% 原始线段绘制
plot(x, y, '-o', 'LineWidth', 2, 'MarkerSize', 6);
hold on;
text(0, 11, num_points, ['Frame: ', num2str(t)], 'FontSize', 12);
%%%
a = 1.7/(2*pi); % 螺距
theta = linspace(0, 8 * pi, 1000); % 角度范围

% 两条中心对称的等距螺旋线
r_in = a * theta; % 顺时针盘入螺线
r_out = -a * theta; % 逆时针盘出螺线 (中心对称)

% 将极坐标转换为直角坐标
x_in = r_in .* cos(theta);
y_in = r_in .* sin(theta);

x_out = r_out .* cos(theta);
y_out = r_out .* sin(theta);

% 找到半径为4.5m的点
radius_target = 4.5;
[~, idx_in] = min(abs(r_in - radius_target));
[~, idx_out] = min(abs(r_out + radius_target));

% 对应半径为4.5m的点的坐标
x_in_4_5 = x_in(idx_in);
y_in_4_5 = y_in(idx_in);
x_out_4_5 = x_out(idx_out);
y_out_4_5 = y_out(idx_out);

% 圆弧参数
a1=3.2611;
a2= 3.0221;
r= 1.5014;

```

```

x1= -0.7639;
x2= 1.7420;
y1= -1.3018;
y2= 2.4409;

% 创建大圆和小圆的参数
theta_circle = linspace(0, 2*pi, 1000); % 完整圆的角度范围

x_circle1 = x1 + 2 *r * cos(theta_circle); % 大圆的x坐标
y_circle1 = y1 + 2*r * sin(theta_circle); % 大圆的y坐标

x_circle2 = x2 + r * cos(theta_circle); % 小圆的x坐标
y_circle2 = y2 + r * sin(theta_circle); % 小圆的y坐标

% 绘制螺旋线

hold on;
plot(x_in, y_in, 'b', 'DisplayName', 'Clockwise Spiral (盘入螺线)');
plot(x_out, y_out, 'g', 'DisplayName', 'Counterclockwise Spiral (盘出螺线)');

% 标记半径为4.5m的点
scatter(x_in_4_5, y_in_4_5, 100, 'filled', 'r', 'DisplayName', 'Radius 4.5m Point');
scatter(x_out_4_5, y_out_4_5, 100, 'filled', 'r', 'DisplayName', 'Radius 4.5m Point');

% 标注半径为4.5m的点
text(x_in_4_5, y_in_4_5, sprintf('(% .2f, % .2f)', x_in_4_5, y_in_4_5), 'Color', 'r');
text(x_out_4_5, y_out_4_5, sprintf('(% .2f, % .2f)', x_out_4_5, y_out_4_5), 'Color', 'g');

% 绘制完整的大圆和小圆
plot(x_circle1, y_circle1, 'r', 'LineWidth', 2, 'DisplayName', 'Large Circle');
plot(x_circle2, y_circle2, 'm', 'LineWidth', 2, 'DisplayName', 'Small Circle');

% 添加图例和标签
title('Center Symmetric Spirals and Circles with Radius 4.5m Points');
xlabel('x (m)');

```

```

ylabel('y (m)');
% legend('show');
axis equal;
grid on;

hold on;
line(xlim, [0 0], 'Color', 'k');
line([0 0], ylim, 'Color', 'k');

% 添加一个圆心在原点，半径为4.5米的圆
r1_circle = 4.5;
theta_circle = linspace(0, 2*pi, 1000);
x1_circle = r1_circle * cos(theta_circle);
y1_circle = r1_circle * sin(theta_circle);
plot(x1_circle, y1_circle, 'k--', 'LineWidth', 1.5); % 绘制圆

% 坐标轴
hold on;
line(xlim, [0 0], 'Color', 'k'); % X轴
line([0 0], ylim, 'Color', 'k'); % Y轴

% plot(X0, Y0, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'b');

%%

EX1 = [];
EY1 = [];
EX2 = [];
EY2 = [];

% 遍历每一对相邻点，计算线段并检测碰撞
for i = num_points - 1:-1:1
    % 获取线段的端点
    x1 = x(i); y1 = y(i);
    x2 = x(i+1); y2 = y(i+1);

```

```

% 计算线段方向
dx = x2 - x1;
dy = y2 - y1;
length_original = sqrt(dx^2 + dy^2); % 线段长度
unit_vec = [dx, dy] / length_original; % 方向单位向量

% 计算垂直向量
perp_vec = [-dy, dx]; % 垂直向量
perp_unit_vec = perp_vec / norm(perp_vec); % 垂直单位向量

% 计算平移后的点
shift_x = half_width * perp_unit_vec(1);
shift_y = half_width * perp_unit_vec(2);

% 平移后的线段
shifted_x1 = x1 + shift_x; shifted_y1 = y1 + shift_y;
shifted_x2 = x2 + shift_x; shifted_y2 = y2 + shift_y;

% 延长平移后的线段
extended_x1 = shifted_x1 - extension * unit_vec(1);
extended_y1 = shifted_y1 - extension * unit_vec(2);
extended_x2 = shifted_x2 + extension * unit_vec(1);
extended_y2 = shifted_y2 + extension * unit_vec(2);

EX1(end + 1) = extended_x1;
EY1(end + 1) = extended_y1;
EX2(end + 1) = extended_x2;
EY2(end + 1) = extended_y2;
% 绘制延长后的线段
plot([extended_x1, extended_x2], [extended_y1, extended_y2], 'LineWidth', 2);
end

axis equal; % 设置 x 和 y 轴比例相同
axis([min(data(:)) max(data(:)) min(data(:)) max(data(:))]); % 设置动态坐标轴范围
grid on;

```

```

% 控制动画速度
pause(0.1); % 暂停 0.1 秒
clf; % 清除旧图形
end

%% 定义交点检测函数，考虑延长后的线段
function intersects = check_intersection(p1, p2, q1, q2)
    % 使用向量叉积法计算线段是否相交
    cross1 = (q1(1) - p1(1)) * (p2(2) - p1(2)) - (q1(2) - p1(2)) * (p2(1) - p1(1));
    cross2 = (q2(1) - p1(1)) * (p2(2) - p1(2)) - (q2(2) - p1(2)) * (p2(1) - p1(1));
    cross3 = (p1(1) - q1(1)) * (q2(2) - q1(2)) - (p1(2) - q1(2)) * (q2(1) - q1(1));
    cross4 = (p2(1) - q1(1)) * (q2(2) - q1(2)) - (p2(2) - q1(2)) * (q2(1) - q1(1));

    % 判断叉积符号是否相异，若相异则两线段相交
    intersects = (cross1 * cross2 < 0) && (cross3 * cross4 < 0);
end

```