基于牛顿二分法的“板凳龙”螺线轨迹模型

摘要

“板凳龙”是有着浓厚民俗文化的非物质文化遗产，深受闽浙一带居民的喜爱。本文通过建立螺线轨迹模型，利用数学物理方法，分析了“板凳龙”在二维平面上的运动情况，并利用**牛顿二分法**等算法对求解过程进行简化。

针对问题一，本文首先对需要求解的螺线轨迹建立了极坐标系和直角坐标系。然后利用等距螺线的定义，通过螺距求得极坐标系下的螺线方程，并可以对轨迹方程进行积分得到轨迹螺线长。之后通过龙头行进距离等于龙头轨迹螺线长，可以得到任意时刻龙头前把手在极坐标系上的坐标，然后利用**牛顿二分法**求得龙头后把手的位置，并将其作为第二节龙身前把手，依次递推得到各处把手的坐标。之后利用坐标转换公式可以将其转换至平面直角坐标系上，从而得到任意时刻“板凳龙”各把手的位置。针对速度的求解，本文利用利用了所需求解的时刻所在的一段足够小的时间间隔的前后位置，利用其前后位置的差分来近似求得其速度。最终通过 Python 编程求解，得到龙头在 300s 处的位置横坐标为 **4.420274m**，纵坐标为 **2.320429m**。

针对问题二，首先对问题一中构建的螺线轨迹模型进行分析，发现龙头部位的曲率最大，可以得出碰撞必先发生在龙头。然后利用各板凳四个角点相对于前后把手的位置，利用相关数学公式，可以求出各处角点的位置，并将碰撞抽象为龙头角点是否位于龙身板凳矩形内。之后利用射线相交法，对碰撞的求解过程进行简化。然后对碰撞发生的时间不断用**牛顿二分法**进行逼近，并最终二分求得满足最小阈值的碰撞时间为 **412.473838s**。通过第一问构建的轨迹模型即可得到该时间各处把手的位置和速度，其中龙头前把手的横坐标为 **1.209931m**，纵坐标为 **1.942784m**。

针对问题三，首先需要对碰撞发生的时间进行求解，首先大致估计一个较优的螺距值，同样采用**牛顿二分法**利用逼近二分的方法求得龙头前把手进入调头区域的时间，然后利用二分法求得在该时间下龙头与龙身恰好不相撞的螺距，再将该螺距作为初始值利用二分法求得龙头前把手进入调头区域的时间，再将该时间作为初值求螺距，不断迭代求解，直到相邻两次解得的螺距之差小于限差，最终计算得到螺距为 **0.441834m**。

针对问题四，首先需要利用问题一中的坐标系，利用几何关系推导出盘入盘出点的径向方向角与切向方向角与其极角的关系。之后建立“板凳龙”调头路线模型，利用其调头路径的路线的几何关系约束、螺线方程关系约束，以总调头路径最短为优化目标建立单目标优化模型。然后利用二分法和变步长搜索的方法求解出该模型的可行解集，在该解集里进行更为复杂的碰撞条件约束，即剔除龙头与龙身相撞的情况后，选择总调头路径最短的路径参数作为最终解。最后利用路径参数得到路径曲线方程，从而求解出“板凳龙”各部位在调头路径的位置和速度，其中第 100s 的龙头横坐标为 **-3.083475m**，纵坐标为 **7.576127m**，最短调头弧长为 **13.621245m**。

针对问题五，首先利用问题四中“板凳龙”各处龙身在调头空间的速度粗略估计盘出后龙头的最大行进速度。之后计算舞龙队从盘入到盘出调头空间的整个时间，遍历时间历元求出某处龙身在整个时间内的最大速度，然后利用牛顿二分法对龙头行进速度不断进行二分逼近，直到所有时间历元的最大龙身速度与 2m/s 的差小于阈值，最终可得到龙头最大行进速度为 **1.812835m/s**。

关键词：牛顿二分法　螺线轨迹方程　单目标优化　调头路线模型

一、 问题重述

“板凳龙”是闽浙地区用于迎神、祈福的一项民俗活动，蕴含着深厚的地域文化。其是由多条板凳首尾钻孔相连而成，整体呈圆盘状，表演者握住前后把手舞动，形似一条蜿蜒的龙，具有强烈的观赏价值和文化价值。

问题说明了“板凳龙”由龙头、龙身、龙尾三个部分组成及各组成部分的具体含义。同时给出了组成“板凳龙”的板凳数目、长度、宽度、孔径等具体参数。现要求解决如下五个问题：

问题一：该问题给出了“板凳龙”圆盘的螺距 55cm、龙头的行进速度 1m/s。现已知龙头初始时刻位于圆盘螺线第 16 圈与 x 轴交点，需要建立数学模型，求出 0-300s 内每秒时整个舞龙队前后把手的位置和速度。

问题二：该问题要求在问题一建立的模型的基础上继续求解，求出舞龙队板凳之间不发生碰撞的最大时间，并求出此时各个位置前后把手的位置和速度。

问题三：该问题给出了舞龙队的调头空间，即以螺线中心为圆心，直径 9m 的圆，在以上问题建立的模型的基础上，要求解出能使龙头前把手盘入调头空间的最小螺距。

问题四：现在给出了盘入螺距为 1.7m，已知舞龙队调头路径为两段圆弧组成的 S 形曲线（后一段圆弧半径为前一段的两倍，且与盘入盘出螺旋均相切）。现需要在问题三的基础上，保持调头弧段与螺线的相切关系不变，建立数学模型，求解出一条最短的调头弧段，并以掉头开始时刻为零时刻，计算出-100s-100s 内每秒舞龙队的位置。

问题五：在问题四基础上，舞龙队沿求解出的路径前进，龙头行进速度不变，现要求各把手的速度不超过 2m/s，需建模求解出龙头行进的最大速度。

二、 问题分析

2.1 问题一的分析

要求解出各处把手的位置和速度，需要建立起以螺线中心为极点的极坐标系[1]，利用题目中给出的圆盘螺距，基于阿基米德螺线公式，求出圆盘螺线的表达式[2]。然后对该表达式进行积分处理，计算出螺线长度，利用螺线长度等于龙头行进距离，计算出不同时刻龙头位于极坐标系的坐标，将其从极坐标系转换为直角坐标系，之后采用牛顿二分法，求得龙头后把手的位置并将其作为下一个龙身的前把手位置，依次递推，求出各处把手的位置坐标。对于各处把手处的速度，可以利用相邻时间间隔内位置差分的方法求解，当时间间隔足够小时，位置对时间的差分就无限趋近于所求速度。

2.2 问题二的分析

本问题需要求解出碰撞时刻“板凳龙”各处把手的位置和速度，利用第一问得到的螺线轨迹公式，可以求出任意时刻龙头的位置和速度。由对第一问运动状态的分析可知，龙头部位曲率最大，碰撞必定发生在龙头，根据把手的位置和相关参数，通过向量的累加，可以得到龙头四个角点的位置参数，当龙头的四个角点与龙身（除去第一节龙身，因为龙头和第一节龙身本身就有重叠部分）边界重叠时，即恰好发生碰撞。利用牛顿二分法，逐步二分求解出碰撞发生的时间，利用该时间和螺线轨迹，即可求得碰撞时刻内各处把手的位置和速度。

2.3 问题三的分析

本问题中给出了舞龙队调头空间的大小，需要求解出能够使龙头前把手切入调头区域的最小螺距。利用阿基米德螺线，将螺距转换为螺线参数，从而得到龙头前把手

运动的轨迹，利用该螺线轨迹与调头区域相交和在相交点处恰好碰撞的条件，先后对时间和螺距进行二分法：对时间二分，得到螺距一定时，队伍何时进入调头边界。恰好进入边界时利用问题二的思路分析是否碰撞，再对螺距进行二分，若碰撞，二分时应该将螺距增大，反之减小，直到在调头边界恰好发生碰撞，此时得到的螺距便是最小螺距。

2.4 问题四的分析

本问题给出了螺线螺距和龙头行进速度，需要求出在调头空间内调头的最短路径，可以利用优化模型对其求解。通过对调头“S”形曲线进行分析，可以得出该曲线内需要满足的几何关系，同时调头点位于上述建立的螺线轨迹上，还需要满足螺线轨迹约束，此外题目中还隐含了调头空间内不能相撞的约束条件。以调头曲线总长度最小为优化目标，可以建立该优化模型进行求解。

由于碰撞问题约束条件较为复杂且为简化计算过程，本文首先采用了二分法和变步长搜索的方法，在不考虑碰撞的约束下，对该优化模型进行求解，得到一个较为准确的可行解范围。之后在该可行解中剔除不满足碰撞约束条件的解，然后将剩余解中目标函数最小的解作为计算结果。

2.5 问题五的分析

本问题需要求解舞龙队在问题四给定的螺线和求得的调头曲线上运动时，龙头的速度最大为多少，可以保证龙身各个部位的运动速度均不超过 2m/s。本问可以利用第四问求到的结果，初步得到整个运动过程中出现的最大速度，再结合问题四中的龙头速度，将龙头速度与各个部位运动的最大速度的比值认为是近似相等的，来进行初始化，之后仍然利用二分法和时间遍历，得到整个运动过程中出现的最大速度为 2m/s 时龙头的运动速度。

三、 模型假设

1.假设“板凳龙”的螺线轨迹满足阿基米德螺线。
2.各处板凳为刚体，忽略其受力产生的变形。
3.各处板凳在竖直方向上的运动可以忽略。
4.忽略舞龙队挥舞板凳行进时受到的各种阻力。
5.忽略舞龙队队员的肩宽。
6.以板凳前把手的速度代替每个板凳的速度

四、 符号说明

符号	说明	单位
q	螺线轨迹的螺距	m
a	螺线轨迹的螺线参数	m
θ	螺线轨迹在极坐标系中的极角	rad
v	螺线轨迹中各处部位的运行速度	m/s
r	螺线上点的极径	m
x	螺线上的点在直角坐标系上的横坐标	m
y	螺线上的点在直角坐标系上的横坐标	m
t	点在螺线轨迹上的运行时间	s

五、 “板凳龙”模型的建立与求解

5.1 “板凳龙”轨迹模型的建立与求解

5.1.1 模型的建立

为计算出螺线上各把手的位置和速度，首先需要建立如图 1 所示的极坐标系，即以螺线中心为极点 O，从 O 到初始位置 A 点的 OA 方向为极轴，建立极坐标系。

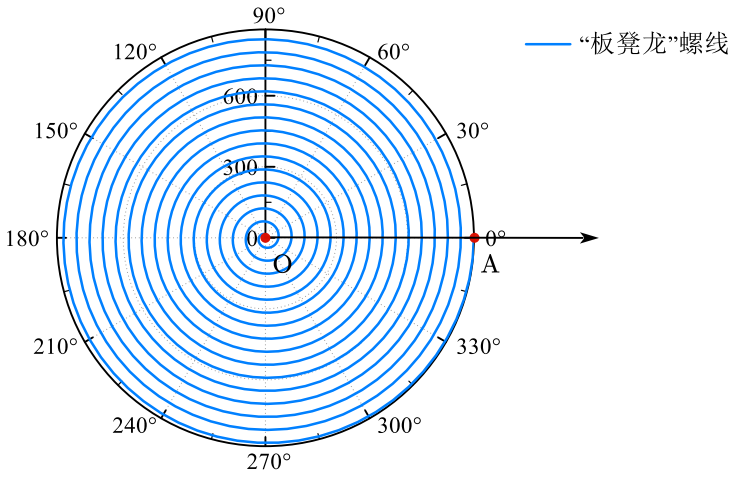

图1 极坐标系示意图

由于题目中需要求解的坐标为位于平面直角坐标系中，因此本文还建立了以螺线中心 O 为原点，OA 方向为 X 轴，垂直于 OA 方向且竖直向上方向为 Y 轴的平面直角坐标系，如图 2 所示。

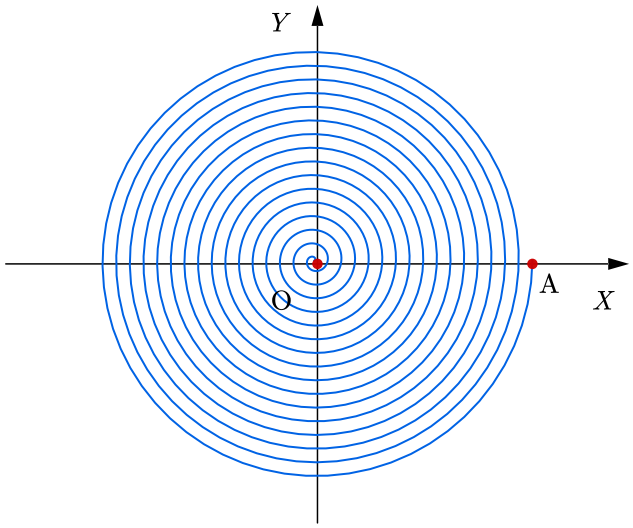

图2 直角坐标系示意图

由于“板凳龙”螺线为等距螺线，且螺距 q 为 55cm，可以将其认定为阿基米德螺线。由此可得其极坐标系内螺线公式为

$$r = a \cdot \theta \tag{1}$$

由阿基米德螺线定义可知，参数等于其径向速度与角速度的比值，故其计算方式如下：

$$a=\frac{v}{w}=\frac{v\cdot T}{w\cdot T}=\frac{q}{2\pi} \tag{2}$$

根据问题一中龙头的行进速度始终为 1m/s，可以求出龙头沿螺线的行进距离 $L=vt$。将龙头行进距离按照公式 1 进行线积分同样能够得到L，二者结合从初始时刻开始可求出积分步长θ_1、θ_2。

$$a\int_{\theta_1}^{\theta_2}\sqrt{\theta^2+1}\cdot d\theta=\nu\cdot t \tag{3}$$

$$\begin{cases} t_0=0 \\ \theta_1^{t_0}=0 \end{cases}\longrightarrow 求解\theta_2^{t_0}\quad \begin{cases} t_1=1 \\ \theta_1^{t1}=\theta_2^{t_0} \end{cases}\longrightarrow 求解\theta_2^{t_1}\longrightarrow\cdots\longrightarrow 得到所有时刻\theta$$

将求得的θ代入式 1 中，可以得到任意时刻龙头前把手在极坐标系内的坐标，利用如下的坐标转换公式，便可以将求得的坐标转换至要求的平面直角坐标系中，转换后的直角坐标系如图 2 所示。

$$\begin{cases} x=r\cdot\cos(\theta) \\ y=r\cdot\sin(\theta) \end{cases} \tag{4}$$

利用牛顿二分法，计算得到龙头后把手的坐标，将该后把手坐标作为下一个龙身的前把手，依次递推，即可求得每个把手的坐标。牛顿二分法的具体过程如图 3 所示，先估计出一个大致的位置，如图 3 中S_1，连接S_0S_1，计算其距离与板凳长l的差$d=S_0S_1-l$，若$d>0$，则将OS_1不断顺时针加上一个小角度，得到S_2。计算S_0S_2与板凳长的差d，直到该差值的符号发生了变化（如从S_0S_4到S_4S_5），之后对OS_4、OS_5之间不断进行二分（如图中红线OS'），直到S'满足约束条件。

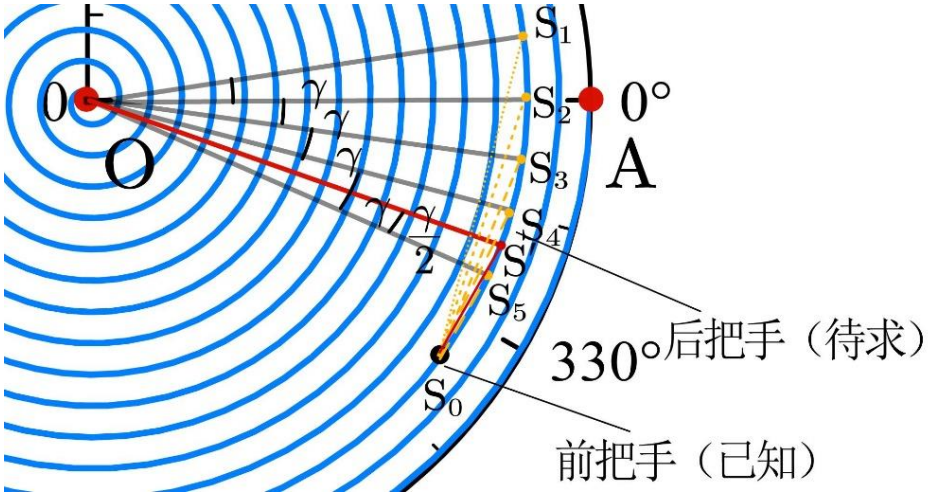

图3 牛顿二分法示意图

对于速度的求解可以采用极限的思想，利用一段较短的时间间隔dt（本文取

10^{-4}s），在该时间间隔内可以认为舞龙队的运动是匀速直线运动,从而可以得到如下的速度计算公式

$$v=\frac{\sqrt{(x_{t+dt}-x_t)^2+(y_{t+dt}-y_t)^2}+\sqrt{(x_t-x_{t-dt})^2+(y_t-y_{t-dt})^2}}{2\cdot dt} \tag{5}$$

5.1.2 模型的求解

在本问题中，需要利用螺距和龙头的行进速度求得各处把手在 0-300s 每个时刻的位置和速度。具体算法流程如下：

step1：从 Excel 文件中读取并存储初始的位置信息和速度数据，为后续计算做准备。

step2：计算出螺旋线的特定参数，确定龙的运动路径和初始位置。

step3：对每个时间步长（0-300 秒），依次进行位置和速度的计算。

step4：根据当前时间，使用积分方法计算龙头的极坐标和直角坐标。

step5：从龙头开始，利用牛顿二分法调整相邻节点的角度，确保相邻节点的距离满足要求。逐步计算每节龙身和龙尾的极坐标和直角坐标。

step6：计算前后微小时间间隔（±0.0001 秒）下各节的位置，通过位置差分计算龙头、龙身及龙尾的行进速度。

step7：输出计算结果。

为直观地展示算法流程，本文绘制了如图 4 所示的流程图

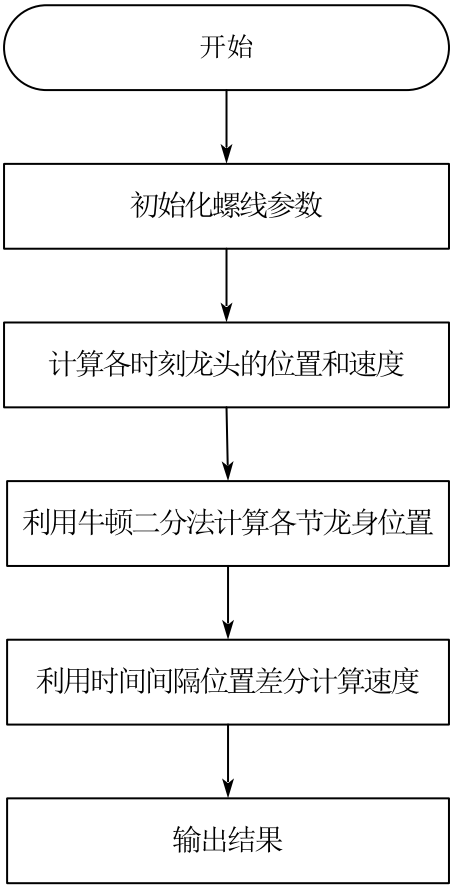

图4 问题一求解流程图

该算法利用了牛顿二分法的方法，通过设置小角度逼近和二分的方法，巧妙地将该连续的运动问题转换为非连续问题，从而大大降低了该问题的时间复杂度。最终求

出各处把手的位置如表 1 所示，速度如表 3 所示，空值表示此时当前部位没有进入螺线队列。

表1. 盘入螺线位置结果

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

表2. 盘入螺线速度结果

	0s	60s	120s	180s	240s	300s
龙头(m/s)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身(m/s)		0.999927	0.999955	0.999890	0.999850	0.999702
第 51 节龙身(m/s)			0.999076	0.999260	0.999172	0.997805
第 101 节龙身(m/s)				0.998481	0.998936	0.997103
第 151 节龙身(m/s)						0.996586
第 201 节龙身(m/s)						
龙尾（后）(m/s)						

求解得到的 0-300s 内各处把手的位置分布如图 5 所示：

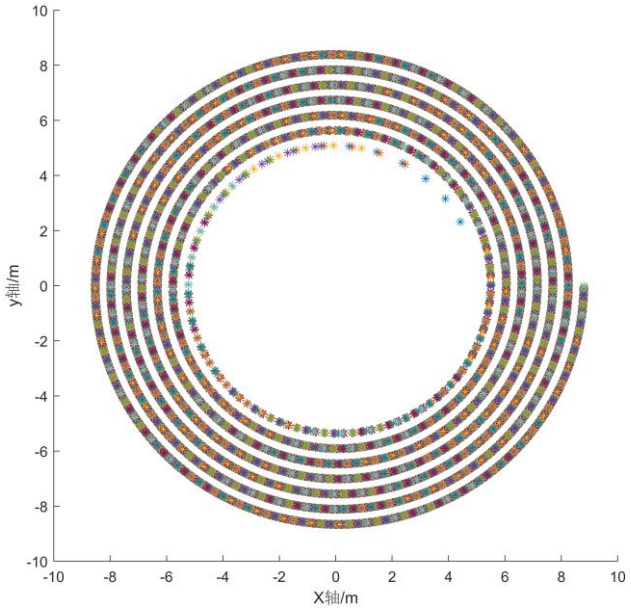

图5 “板凳龙”各处部位位置分布图

为更清晰地了解“板凳龙”的运动状态，本文将龙头与第一处龙身处的位置随时间分布单独分析，并加入了板凳长以更好地观察进行“板凳龙”表演过程中板凳的运动状态，如图 6 所示：

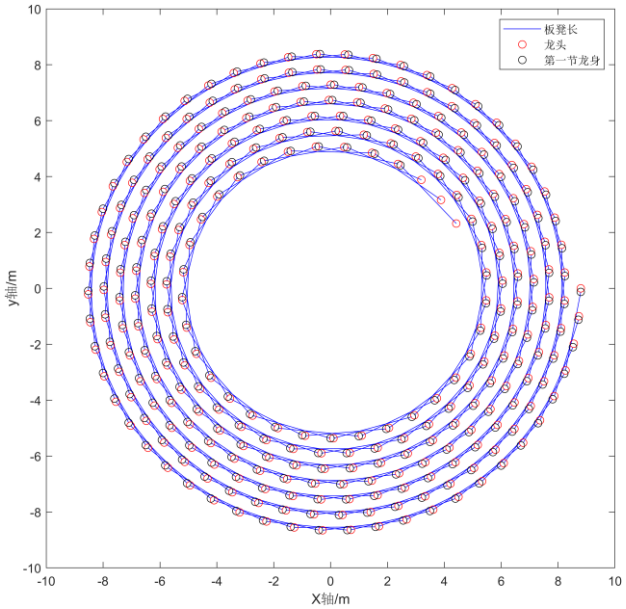

图6 “板凳龙”表演中板凳的运动状态

从上图中不难看出越往盘内进入，“板凳龙”的起伏越大，“板凳龙”圆盘的曲率越大，其与外环碰撞的可能性也会增大。

5.2 碰撞模型的建立与求解

由 5.1.2 中对图 6 的分析可知，“板凳龙”越向盘内进入，发生碰撞的概率越大。因此第一次发生碰撞的必然是龙头与外侧龙身，并且是发生在龙头前进方向的左侧，如图 7 所示：

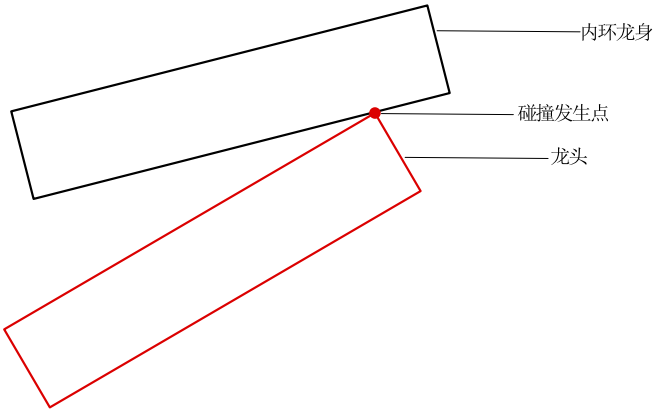

图7 碰撞发生示意图

在问题一中，本文已建模求解出了龙头前后把手的位置随时间变化的轨迹，根据题目给出的龙头把手相对于角点的具体位置，现假设前把手沿龙头方向距左边界的距离为l_1，龙头上下边界的距离为l_2，如图 8 所示：

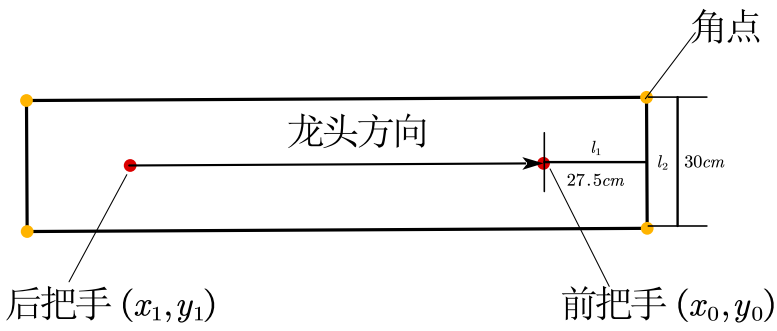

图8 角点位置示意图

为求得角点位置，现规定龙头方向为从龙头前把手指向后把手的方向，即为(x_0-x_1,y_0-y_1)，垂直龙头方向为(y_1-y_0,x_0-x_1)，已知角点相对于龙头方向和垂直龙头方向的距离，故可以得出四个角点的位置，以右上角点为例，具体公式如下：

$$(x_{右上},y_{右上})=(x_0,y_0)+\frac{l_2\cdot(y_1-y_0,x_0-x_1)}{2\sqrt{(x_1-x_0)^2+(y_1-y_0)^2}}+\frac{l_1\cdot(x_0-x_1,y_0-y_1)}{\sqrt{(x_1-x_0)^2+(y_1-y_0)^2}} \quad \textbf{(6)}$$

其中l_1、l_2均为定值，对于龙头，前后把手距离$\sqrt{(x_1-x_0)^2+(y_1-y_0)^2}$为 2.86m，对于龙身及龙尾，其前后把手的距离为 1.65m。因此可以根据式(5)计算出舞龙队各处部位的角点坐标。

该碰撞问题可以将其简化为判断龙头的左上角点是否会出现在内环龙身的矩形平面内，利用射线相交法就能够快速高效地对点与面的关系进行判断，如图 9 所示，从待判断点向任意方向引出一条射线，计算其与矩形边界的交点，若交点个数$n=0$或2则认为该点为矩形外部点；若交点个数$n=1$则认为该点为矩形内部点。

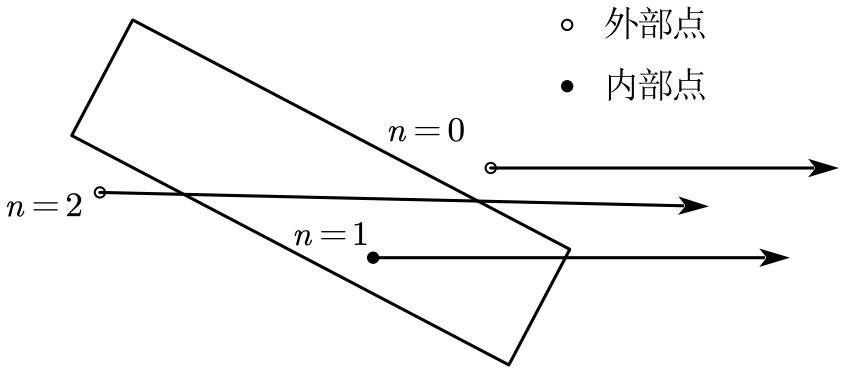

图9 射线相交法示意图

为将连续的运动问题转换为非连续的过程，本文仍选择采用牛顿二分法的方法进行求解，具体过程如图 10 所示：

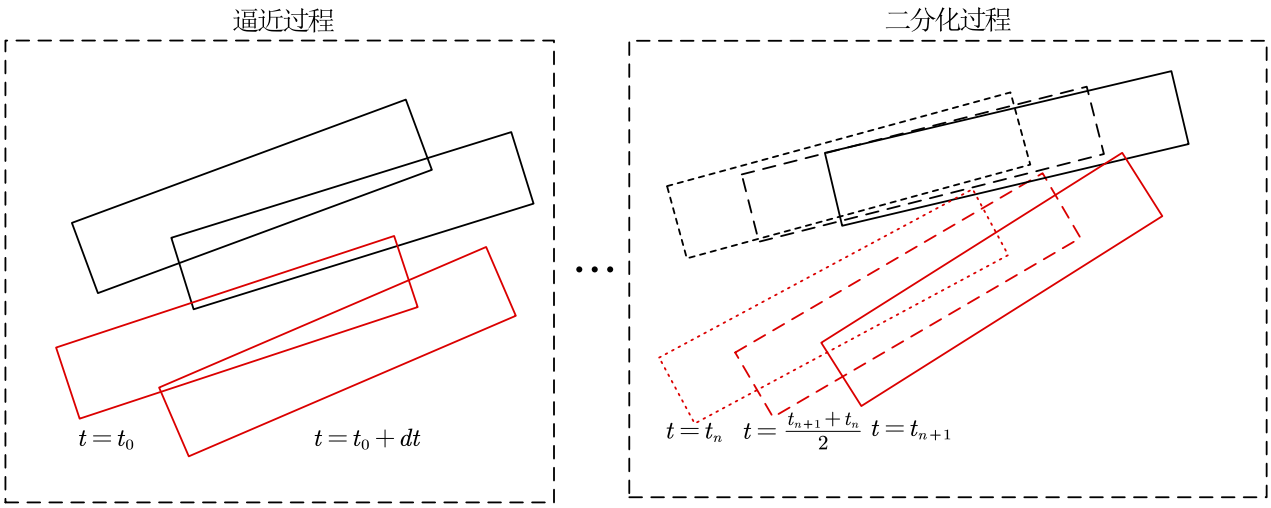

图10　　牛顿二分法解决龙头碰撞问题

1、大致估计一个龙头前把手所处的时间t_0，如第一问中的 300s。

2、利用轨迹方程求出龙头的四个角点在$t_1=t_0+dt$时的位置，并判断四个角点是否位于内环龙身凳内部，如果不位于其内部，则令时间$t_2=t_1+dt$，从而让当前时间不断逼近可能发生碰撞的时间点。

3、当时间处于t_{n+1}时发现已成功逼近，则取当前时间，计算龙头四个角点的位置，并将其与角点是否位于龙身内部状态相异的点继续进行时间二分，直到相邻两时间点的差小于10^{-6}s，则可以认为在时间上收敛了，输出当前时间t_i即为碰撞发生的时间点，再利用螺线轨迹方程即可求得各处把手在碰撞时间的位置。碰撞发生时速度的求法与第一问相同，均是采用极限的思想

最终计算得到龙头会在 412.473838s 发生碰撞，此时部分部位的位置和速度如表 3 所示：

表3. 碰撞时刻部分部位的位置和速度

	X(m)	Y(m)	V(m/s)
龙头	1.209931	1.942784	1
第 1 节龙身	-1.643792	1.753399	0.988684
第 51 节龙身	1.281201	4.326588	0.975675
第 101 节龙身	-0.536246	-5.880138	0.974893
第 151 节龙身	0.968840	-6.957479	0.972408
第 201 节龙身	-7.893161	-1.230764	0.975112
龙尾(后)	0.956217	8.322736	0.977140

碰撞时刻各处把手位置分布如图 11 所示：

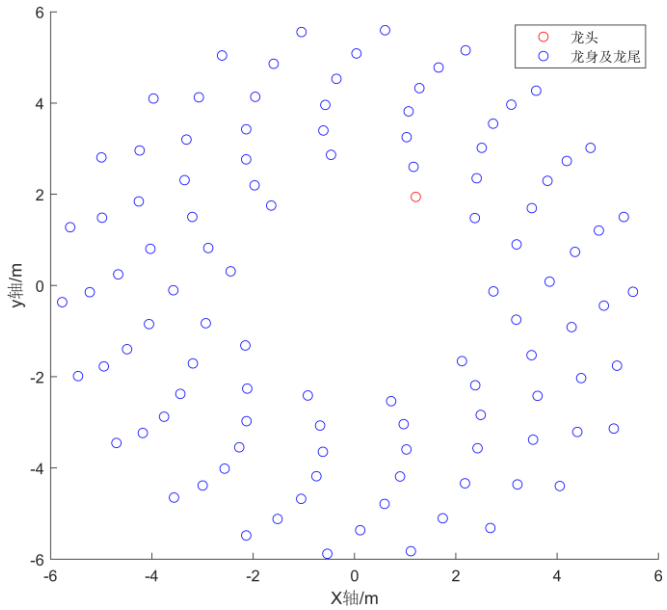

图11　　碰撞时刻各处把手分布

从图中不难看出内侧螺线曲率明显比外侧螺线大，这与模型的预期符合地很好，进一步证明了模型地准确性。

5. 3 调头区域模型的建立与求解

本问题需要求解出能使得“板凳龙”龙头前把手顺利切入调头区域内的最小螺距。对此，需要先构建与问题一中类似的阿基米德螺线，如上式(1)将螺线距转换为阿基米德螺线参数，从而将确定螺线距的问题转换为了确定龙头前把手运行轨迹的问题，如图 12 所示：

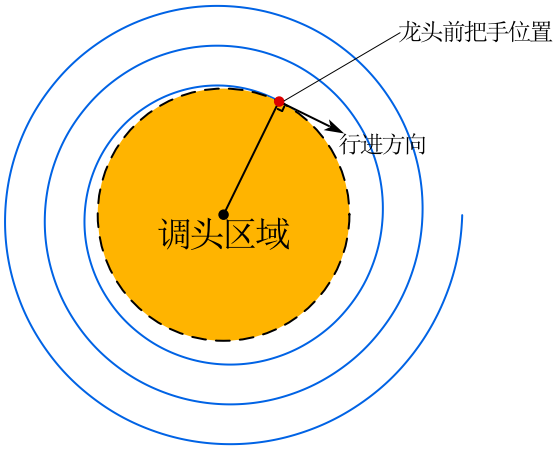

图12　　调头区域示意图

由于该问题中时间t与螺线螺距q均为未知值，同时确定二者的计算量过大，因此可以利用二分法对运算过程进行简化。首先应当估计一个螺距值q_0（即先估计一个值，由第二问计算得到当螺距为 0.55m 时，龙头在离原点 2.28m 处相撞，小于第三问中的 4.5m，故可令$q_0 = 0.55m$），并对时间t进行二分，同 5.2，即每次增加一个较小的时间间隔dt，然后不断逼近，直到发生龙头前把手进入调头区域，在碰撞发生的时间t_n

与上一次碰撞未发生的时间t_{n-1}之间不断进行二分，直到求出相邻时间间隔，直到得到龙头前把手进入调头区域的时间t_0。

之后，利用牛顿二分法对螺线参数进行二分。记龙头离原点的距离$D=\sqrt{x_{龙头}^2+y_{龙头}^2}$，在当前计算的碰撞时间$t_0$，将0.55m作为待估螺距的初值进行估计，不断给螺距减去一个较小值d_0，直到n次之后D大于4.5m，然后取螺距$q=\frac{(2n-1)d_0}{2}$，计算D，若D小于4.5m，取$q=\frac{q_n+q_{n+1}}{2}=\frac{(4n-1)d_0}{4}$；若$D$大于4.5m，取$q=\frac{q_{n-1}+q_{n+1}}{2}=\frac{(4n-3)d_0}{4}$，依次递推，直到相撞位置与4.5m的差小于阈值，则说明在当前螺距下恰好不相撞，输出当前计算的螺距q_1。

然后再利用计算得到的螺距q_1再次对时间进行二分，求出龙头前把手进入调头区域的时间t_1，之后利用时间t_1对螺距进行二分得到恰好不相撞时的螺距q_2，依次递推直到相邻两次计算的螺距大小之差小于阈值，则可以认为计算结果满足精度要求，输出螺距q_n。最终计算得到当前螺距$q=0.441834m$。

本模型在解题过程中利用了将时间与螺距先后进行二分的思想，先确定碰撞时间，然后在碰撞时间下二分逼近出切入螺距，并对该过程不断迭代求解，避免了同时对两个未知参数进行求解的庞大计算量，使模型具有良好的扩展性。

5.4 调头路径优化模型的建立与求解

5.4.1 径切角推导

本问题需要利用题目给出的螺线螺距、龙头行进速度，寻找到一条最短的且与盘入盘出螺线均相切的“S”形路径。

对此，需要先建立类似5.1.1中图1的极坐标系，通过数学关系推导出龙头前把手在轨迹螺线上的径向方向与切向方向的关系。如所图13示，记径向方向的单位向量为$\vec{a}$，螺旋线切向方向的方向向量为$\vec{b}$，径向方向与螺旋线切向方向的夹角（以下简称径切角）为φ。

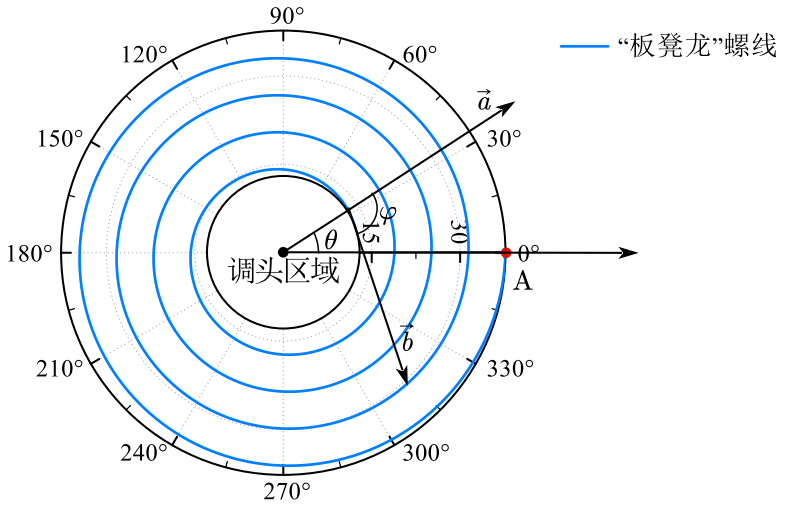

图13　　切向与径向示意图

对于$\vec{b}$可以通过对螺线轨迹方程（式 1）求导得到。需先利用式 4 将轨迹方程消去极径r，如下式：

$$\begin{cases} x = r\cdot\cos\theta = a\theta\cdot\cos\theta \\ y = r\cdot\sin\theta = a\theta\cdot\sin\theta \end{cases} \tag{7}$$

之后对$\vec{b}$的方向进行求导，不难得出$\dfrac{dy}{dx} = \dfrac{dy}{d\theta}\cdot\left(\dfrac{dx}{d\theta}\right)^{-1} = \dfrac{\sin\theta+\theta\cos\theta}{\cos\theta-\theta\sin\theta}$，由于$\vec{b}$为方向向量，故可以简单地将$\vec{b}$记为$(\cos\theta-\theta\sin\theta,\sin\theta+\theta\cos\theta)$，$\vec{a}$为径向方向的方向向量，在直角坐标系中可直接得出其为$(\cos\theta,\sin\theta)$，之后利用三角函数变换可以得到下式：

$$\begin{cases} \cos\left\langle\vec{a},\vec{b}\right\rangle = \dfrac{\vec{a}\cdot\vec{b}}{|\vec{a}|\,|\vec{b}|} = \dfrac{1}{1+\theta^2} \\ \sin\left\langle\vec{a},\vec{b}\right\rangle = \sqrt{1-\cos^2\varphi} \\ \left\langle\vec{a},\vec{b}\right\rangle+\varphi = 180^\circ \end{cases} \Longrightarrow \tan\varphi = -\theta\cdot\sqrt{\theta^2+2} \tag{8}$$

5.4.2 调头路径优化模型的建立

对于转向情况，如图 14 所示，龙头从 A 点进入调头空间开始转出，此时其径切角为φ，为保证转出螺线与转入螺线呈中心对称，其转出点 B 的径切角也为φ，且 AB 连线的中点为极坐标原点 O。记转角小圆弧半径为R，大圆弧半径为$2R$，圆心角$\angle AO_1P = \alpha$，$\angle BO_2P = \beta$，A、B 点处的极径为r、极角为θ，两圆弧相切点记为点 P，延长AO_1、BO_2交于点 Q。

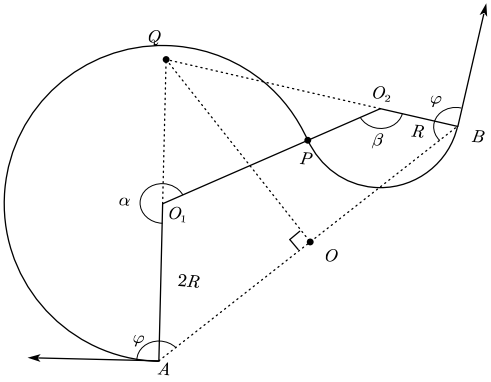

图14　　转向情况示意图

在调头路径形状、盘入螺线距、各曲线相关关系已知的情况下，需要建立单目标优化模型对调头最短路径进行规划求解。

- **约束条件**

1.螺线轨迹约束条件

由于 A、B 点盘入盘出均位于阿基米德螺线上，故其极径r和极角θ均满足如式(1)所示阿基米得螺线方程。

2.角条件

由于转向过程路线是从螺线中切入切出，可以得到$\angle O_1AB=\varphi-90°$、$\angle O_2BA=\varphi-90°$，又由于转入转出的圆弧相切，可以得到O_1、O_2、P 三点共线，利用四边形O_1O_2BA内角和为$360°$可以得到下式(9)。

$$2\varphi+\beta-\alpha=\pi \tag{9}$$

3.径切角关系

由 5.4.1 的推导结果，可以得到 A、B 点处的径切角满足如下关系：

$$\tan\varphi=-\theta\cdot\sqrt{\theta^2+2} \tag{10}$$

4.等腰三角形关系

由于$\angle QAB=\angle QBA=\varphi-90°$，可以得到$\triangle QAB$为等腰三角形，故有$QA=QB$，经过几何关系推导易得如下关系：

$$\sin\alpha+\sin\beta=\sin2\varphi \tag{11}$$

5.三角函数关系

由于$\triangle QAB$为等腰三角形，且 O 点为 AB 连线的中点，故在$\triangle QOA$中有$AQ\cdot\cos(\varphi-90°)=AO$，之后利用图中的几何关系进行数学推导可以得到如下关系：

$$\left(2R-\frac{3R\cdot\sin\beta}{\sin2\varphi}\right)\sin\varphi=r \tag{12}$$

6.**调头空间约束**

由于调头空间为一个半径为 4.5m 的圆，故 A、B 点处的极径r应该位于 0~4.5m 之间

$$0<r<4.5 \tag{13}$$

7.**碰撞条件约束**

舞龙队在沿“S”形曲线调头过程中还隐含着各处板凳不能相撞的条件，因此，现将 5.2 中有关碰撞的复杂关系式简化为$l_{板}=F(\theta,\alpha,\beta)$，其中$l_{板}$代表板凳间间距，$F$表示一定的函数关系，其还需要满足如下关系：

$$0<F(\theta,\alpha,\beta)<q \tag{14}$$

- **决策变量**

由上式可知，r、φ均是关于θ的函数（式（1），式（10）），因此只需要再确定α、β便可以解算出R，然后利用目标函数（式（15））得到最短路径，故决策变量为R、α、β。

- **目标函数**

问题的优化目标为舞龙队各部分沿“S”形曲线调头的路径长度最短。因此，其目标函数为

$$\min \quad f=2R\cdot\alpha+R\cdot\beta \tag{15}$$

综上，本文建立了舞龙队调头路径的单目标优化模型

$$\min \quad f=2R\cdot\alpha+R\cdot\beta$$

$$s.t.\begin{cases} r=\dfrac{q}{2\pi}\theta \\ 2\varphi+\beta-\alpha=\pi \\ \tan\varphi=-\theta\cdot\sqrt{\theta^2+2} \\ \sin\alpha+\sin\beta=\sin 2\varphi \\ \left(2R-\dfrac{3R\cdot\sin\beta}{\sin 2\varphi}\right)\sin\varphi=r \\ 0<r<4.5m \\ 0<F(\theta,\alpha,\beta)<p \end{cases} \tag{16}$$

由于有关碰撞的约束条件最为复杂，直接进行规划求解会导致计算量过大，因此在实际运算过程中，本文先利用前六个约束条件进行求解，计算得到若干组可行解，再从计算这些可行解是否会在调头过程中相撞，剔除不满足碰撞约束条件（式（14））的解后，在余下解中找到目标函数（式（15））最小的解，即为本文所求得的最优解。

5.4.3 调头路径优化模型的求解

考虑到可能存在多个可行解的区域，若是直接利用优化算法遍历求解决策变量（R、α、β）计算量过大。因此，本文先采用二分法的方法搜索关键变量θ的取值，再通过变步长搜索法寻找决策变量（R、α、β）的最佳取值，结合二分法和变步长搜索法的具体流程如图 15 所示：

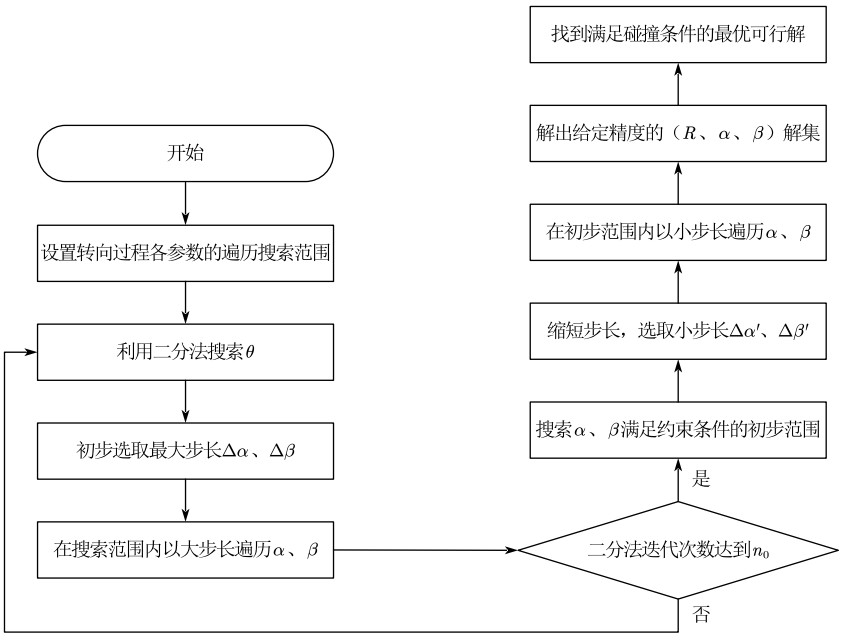

图15　　问题四算法求解流程

按照上述方法，最终求解得到的最短调头路径长度为 13.621245m，问题四求解结果如下表 4、表 5 所示：

表4. 调头过程中“板凳龙”部分部位位置

	-100s	-50s	0s	50s	100s
龙头 x(m)	7.778034	6.608301	-2.711856	1.408772	-3.083475
龙头 y(m)	3.717164	1.898864	-3.591078	6.154956	7.576127
第 1 节龙身 x(m)	6.209273	5.366912	-0.063534	3.921315	-0.268126
第 1 节龙身 y(m)	6.108521	4.475403	-4.670888	4.788674	8.079528
第 51 节龙身 x(m)	-10.608038	-3.629946	2.459962	-1.091642	2.162555
第 51 节龙身 y(m)	2.831491	-8.963800	-7.778145	-6.356925	3.992649
第 101 节龙身 x(m)	-11.922761	10.125787	3.008493	-8.011888	-7.703658
第 101 节龙身 y(m)	-4.802378	-5.972247	10.108539	4.700838	0.915322
第 151 节龙身 x(m)	-14.351032	12.974783	-7.002789	-4.057692	9.947604
第 151 节龙身 y(m)	-1.980993	-3.810358	10.337482	-10.700207	-2.429277
第 201 节龙身 x(m)	-11.952942	10.522508	-6.872842	0.964946	7.704519
第 201 节龙身 y(m)	10.566998	-10.807426	12.382609	-13.217101	9.495343
龙尾后把手 x(m)	-1.011059	0.189810	-1.933627	5.313002	-10.390173
龙尾后把手 y(m)	-16.527573	15.720588	-14.713128	12.924603	-7.824092

表5. 调头过程中“板凳龙”部分部位速度

	-100s	-50s	0s	50s	100s	-100s	-50s
龙头(m/s)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身(m/s)	0.999898	0.999753	0.997767	1.000353	1.000118	0.999898	0.999753
第 51 节龙身(m/s)	0.999342	0.998637	0.994231	0.423391	1.003963	0.999342	0.998637
第 101 节龙身(m/s)	0.999088	0.998245	0.993548	0.407564	0.463081	0.999088	0.998245
第 151 节龙身(m/s)	0.998942	0.998044	0.993257	0.402633	0.455381	0.998942	0.998044
第 201 节龙身(m/s)	0.998847	0.997923	0.993096	0.400226	0.452353	0.998847	0.997923
龙尾后把手(m/s)	0.998815	0.997883	0.993046	0.399518	0.451534	0.998815	0.997883

盘入盘出螺线轨迹分布如图 16 所示

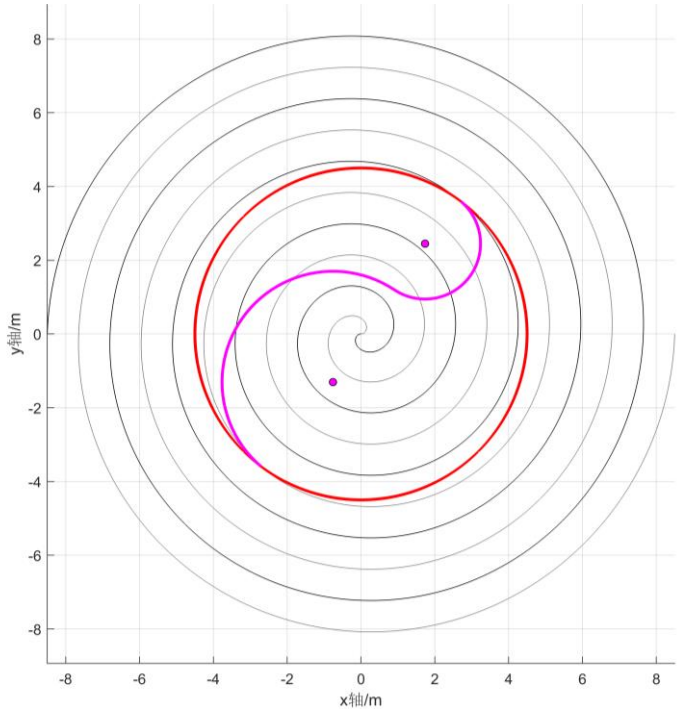

图16　　盘入盘出螺线轨迹

为清晰地表现出舞龙队调头过程的运动轨迹，图 17 表现了龙头前把手中心的运动轨迹，使得盘入盘出的运动过程更为直观。

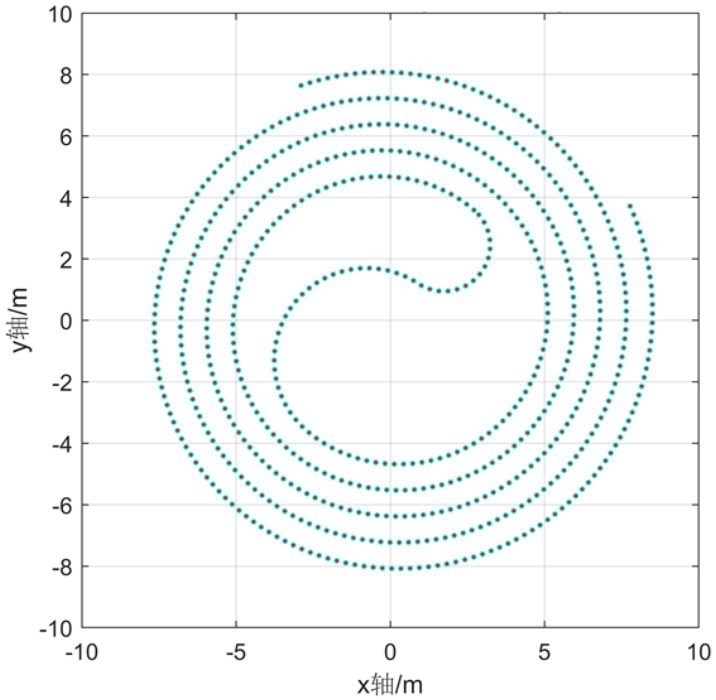

图17　　龙头前把手中心轨迹

5.5 舞龙队速度模型的建立与求解

本问题需要得到舞龙队从调头空间盘出时龙头的最大行进速度。对此，仍采用牛顿二分法的方法对龙头的运动速度进行二分逼近。逼近的初值可以利用第四问盘出的速度值求得，如下式所示：

$$\frac{v_{\max}}{v_{龙头}}=\frac{v'_{\max}}{v'_{龙头}} \tag{17}$$

其中$v_{\max}$为 5.4 中调头路径龙身部位行进的最大速度，$v_{龙头}$为第四问中龙头行进速度 1m/s，$v'_{\max}$为本问题中规定的最大速度 2m/s，$v'_{龙头}$为龙头盘出速度的初值。对该速度初值利用二分法进行逼近和二分，即利用一个较小的速度值dv不断改进龙头盘出速度$v'_{龙头}$，并求出整个舞龙队从盘入调头空间到盘出的总时间，遍历该时间求出舞龙队各部位的最大速度$v_{\max}^{body}$，若$v_{\max}^{body}$大于 2m/s，则开始进行二分，不断取最近的大于 2m/s 的$v_{\max}^{body}$和小于 2m/s 的$v_{\max}^{body}$对应的$v'_{龙头}$的中值，然后继续进行逼近，直到两次$v'_{龙头}$的差小于阈值，即可得到最终结果。最终得到龙头最大行进速度$v'_{龙头}=1.812835m/s$。

六、 模型求解结果分析

6.1 对问题一结果的分析

从求得的板凳龙各部分，在各个时刻的位置表格来分析，可以发现，表格是倒三角形状的，这是由于 0 时刻开始只有龙头进入给定的螺线，队伍其他部分的分布无从得知，随着时间推移，龙头深入螺线，此时有更多的龙身进入，才能求解位置，对于速度表格的形状也是同理，因为速度需要位置对时间求导或者差分。从求得的板凳龙各部分，在各个时刻的速度大小表格来分析，可以发现，随着时间推移以及队内板凳编号数的增加，板凳龙各部分的移动速度大致上是减少的。换言之，板凳龙同一部分速度大小对时间的差分和板凳龙不同部分速度大小对队内板凳编号数的差分，在总体上来看是小于零的。对这一结果进行分析：首先，当板凳龙在螺线外侧运动时，由于螺线的极径较大，单个板凳对应的极角较小，这时板凳长度与螺线长度差异不大。这一结论类似于：当一条线段的两个端点都位于一个圆上，圆的半径越大，线段所对圆心角越小，线段与其对应的弧段差值也就越小。而当板凳龙随着时间推移进入到螺线内侧，这时板凳长度就开始较为明显的小于螺线长度，又知道板凳龙的把手都位于螺线上，且板凳龙的总长是固定的，可以明显的推断出，随着时间推移，板凳龙的队伍长度是在逐渐增加的（仅限于盘入，盘出则与这个过程刚好相反），这说明，队伍靠后的位置在远离龙头，即随着时间推移，队伍的速度小于龙头，且越来越慢；对于编号数这一结论来说也是同理，因为随着时间推移，队内板凳编号大的开始进入螺线。

6.2 对问题二结果的分析

根据表格可以得知速度的规律分布与问题一类似，间接印证了问题一采用模型的正确性。

6.3 对问题三结果的分析

分析可知，螺距越小，队伍越容易在外侧碰撞（极限情况就是螺距小于 0.3m，即凳子宽度，此时螺距不足以容纳螺线上相邻两圈队伍的凳子运动），当螺距越大，队伍就越容易走到螺线内部而晚碰撞，这与实际板凳龙的情况是一致的：当队伍走到内部

要发生碰撞时，会有号令哨响起，此时舞龙队整体外移，也就是增大螺距，以避免发生碰撞。如图 16 所示，当前把手的极径相等时，螺距越大的曲线，相同长度的板凳离其越近（这与问题一的结果分析也类似），此时板凳必然不容易与其他板凳相撞。

在第二问中，螺距 p=0.55m 的条件下，队伍在 r=2.2864m 处发生碰撞，该问中优化的临界点是在 r=4.5m 处恰好碰撞，那么必然可以得知，螺距在 0.3m 与 0.55m 之间，而结果求得的是 0.441834m，在这一范围内，与前文分析一致。

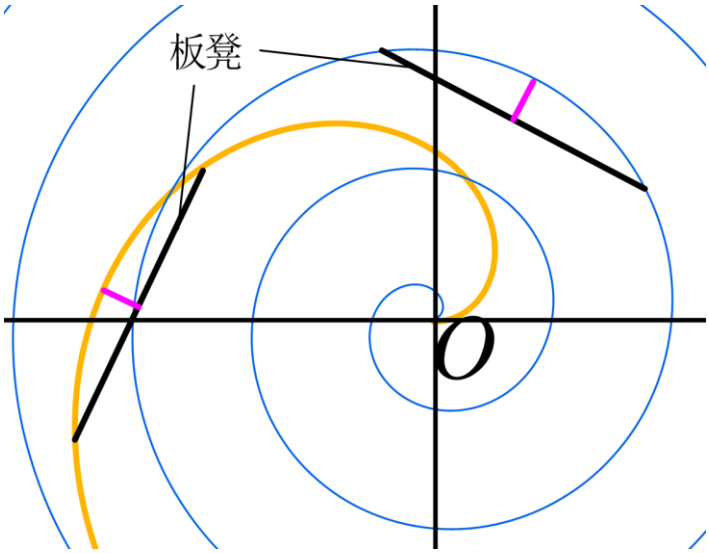

图18　　不同螺距下板凳偏离螺线程度示意图

七、 模型的评价、改进与推广

7.1 模型的优点

1.在构建“板凳龙”螺线模型中多次利用了牛顿二分法的思想，该方法能够将连续的运动问题转换为较短时间间隔内的非连续的运动问题，同时能够在保证运算精度的同时大大加快模型求解问题的速度，使得模型在求解的过程中能大大降低复杂度，同时具有良好地扩展性。

2.本文对于几个问题的模型建立具有可传递性，从问题一到问题五是对模型的层层递进，通过“板凳龙”盘入的整个过程：0-300s 的运动状态、板凳碰撞、板凳进入调头区域、寻找板凳的调头路径、板凳调头后行进速度，以一个完整的“板凳龙”表演过程进行建模，逻辑更为严密。

3.本文在建立模型时只考虑了影响该情况的主要因素，将复杂的运动情况转换为简单且直观的几何模型，从而使模型更加易于理解。

7.2 模型的缺点

构建模型过程中未考虑利用螺线方程推导出螺线的速度，使得最终计算出的速度结果精度不够高。

7.3 模型的推广

1.本文对“板凳龙”的研究中的螺旋运动和调头操作可以作为群体运动控制问题的一个案例，为自动化系统或多机器人协作提供参考。

2.本文对于“板凳龙”调头的研究为在有限空间内规划行驶路线、减少交通拥堵、避免事故等问题提供了思路。

八、 参考文献

[1]. 刘崇军.等距螺旋的原理与计算[J].数学的实践与认识,2018,48(11):165-174
[2]. 陈远宁.关于等距曲线的若干研究[D].合肥工业大学,2007.

附录

附录 1

支撑材料说明

支撑材料包含问题一、问题二、问题三、问题四、问题五、文献引用，共 6 个文件夹，使用时先运行 qm_1.n，再运行 qm_2.n。前五个文件夹分别对应 A 题的求解代码和求解结果，最后一个文件夹是 pdf 格式的引用文献。

问题一的文件夹包含运行代码 q1.py（计算每秒每个把手的位置和速度并保存至 result1.xlsx）、运行结果 result1.xlsx；问题二的文件夹包含运行代码 q2_1.py（计算碰撞时的龙头运动时间）、q2_2.py（计算此时的位置和速度并保存至 result2.xlsx）、运行结果 result2.xlsx；问题一的文件夹包含运行代码 q3.py（计算掉头的最短螺距）；问题四的文件夹包含运行代码 q4_1.m（计算碰撞时的龙头运动时间）、q4_2.py（计算速度并合并 matlab 计算结果到 result4.xlsx）、运行结果 result4.xlsx；问题五的文件夹包含运行代码 q5.m（计算龙头的最大行进速度）；文献引用的文件夹包含论文所引用的文献。

附录 2

第一问代码

```
import pandas as pd
import sympy as sm
import math


def length(x1, y1, x2, y2):
    return sm.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)


def f(theta, X, Y, D):
    return sm.sqrt((a[0] * theta * sm.cos(theta) - X) ** 2 + (a[0] * theta * sm.sin(theta) - Y) ** 2) - D


# 定义符号
x = sm.symbols("x")  # 方位角
b = sm.symbols("b")  # 积分下限，第 t 秒的方位角
a = sm.symbols("a")  # 螺线参数

# 求解螺线参数 a
a = sm.solve(2 * sm.pi * a - 0.55, a)
```

```
21.
22.# 指定 Excel 文件的路径
23.file_path = 'result1.xlsx'
24.
25.# 使用 pandas 读取 Excel 文件
26.df_1 = pd.read_excel(file_path, index_col=0, sheet_name="位置")
27.df_2 = pd.read_excel(file_path, index_col=0, sheet_name="速度")
28.df_3 = df_1.copy()  # 存放第-0.0001 秒的运动坐标
29.df_4 = df_1.copy()  # 存放第0.0001 秒的运动坐标
30.
31.# 先计算0-300s 的龙头位置
32.# 循环遍历 t 值，并计算对应的积分下限 b
33.for t in range(0, 301):
34.    print('第' + str(t) + 's 计算')
35.    # 进行积分，传递正确的积分上下限
36.    fx = sm.integrate(a[0] * sm.sqrt(x ** 2 + 1), (x, b, sm.pi * 32))
    - t
37.    # 解方程，求解 b
38.    res = sm.nsolve(fx, b, 16)
39.    # 求解 r x y
40.    r = res * a[0]
41.    x_t = r * sm.cos(res)
42.    y_t = r * sm.sin(res)
43.    df_1.loc['龙头x (m)', str(t) + ' s'] = x_t
44.    df_1.loc['龙头y (m)', str(t) + ' s'] = y_t
45.    flag = 0  # 判断是不是龙头的标志
46.    theta0 = res
47.    while r < a[0] * 32 * sm.pi:
48.        # 定义距离差
49.        if flag == 0:
50.            delta_d = 2.86
51.            theta1 = theta0  # 初始角度，可以根据需要调整
52.            flag += 1
53.            x0 = x_t
54.            y0 = y_t
55.        else:
56.            delta_d = 1.65
57.            theta1 = theta0
58.            flag += 1
59.        delta = 0.5
60.        # 牛顿法求解
61.        while abs(f(theta1, x0, y0, delta_d)) > 1e-8:
62.            if f(theta1, x0, y0, delta_d) > 0:
63.                theta1 -= delta
```

```
                delta /= 2
            else:
                theta1 += delta
        # 计算相邻点的极坐标
        r1 = a[0] * theta1
        x1 = r1 * sm.cos(theta1)
        y1 = r1 * sm.sin(theta1)
        if r1 <= a[0] * 32 * sm.pi:
            if flag <= 221:
                df_1.loc['第' + str(flag) + '节龙身
   x (m)', str(t) + ' s'] = x1
                df_1.loc['第' + str(flag) + '节龙身
   y (m)', str(t) + ' s'] = y1
            elif flag == 222:
                df_1.loc['龙尾 x (m)', str(t) + ' s'] = x1
                df_1.loc['龙尾 y (m)', str(t) + ' s'] = y1
            elif flag == 223:
                df_1.loc['龙尾（后）x (m)', str(t) + ' s'] = x1
                df_1.loc['龙尾（后）y (m)', str(t) + ' s'] = y1
            else:
                break
        x0 = r1 * sm.cos(theta1)
        y0 = r1 * sm.sin(theta1)
        r = r1
        theta0 = theta1

df_2.loc[['龙头 (m/s)'], :] = 1.000000

# 下面开始计算-0.0001s 的（点是否在螺线内不管，在前面df_1 中已经控制）
# 这里只是为了计算-0.0001s 时各点位置便于速度计算，最终某时刻某点的速度是否
   存在以df_1 中对应时间的该点是否有坐标来判断
for t in range(0, 301):
    print('第' + str(t) + 's 计算')
    # 进行积分，传递正确的积分上下限
    fx = sm.integrate(a[0] * sm.sqrt(x ** 2 + 1), (x, b, sm.pi * 32))
    - t + 0.0001  # 0.0001s 为设定的间隔时间
    # 解方程，求解 b
    res = sm.nsolve(fx, b, 16)
    # 求解 r x y
    r = res * a[0]
      x_t = r * sm.cos(res)
      y_t = r * sm.sin(res)
      df_3.loc['龙头 x (m)', str(t) + ' s'] = x_t
      df_3.loc['龙头 y (m)', str(t) + ' s'] = y_t
```

```
    flag = 0  # 判断是不是龙头的标志
    theta0 = res
    while r < a[0] * 32 * sm.pi + 0.55:  # 这里0.55为手动添加的微小扰动，防止某点在正秒时在螺线范围内，但0.0001s前在螺线外
        # 定义距离差
        if flag == 0:
            delta_d = 2.86
            theta1 = theta0  # 初始角度，可以根据需要调整
            flag += 1
            x0 = x_t
            y0 = y_t
        else:
            delta_d = 1.65
            theta1 = theta0
            flag += 1
        delta = 0.5
        # 牛顿法求解
        while abs(f(theta1, x0, y0, delta_d)) > 1e-8:
            if f(theta1, x0, y0, delta_d) > 0:
                theta1 -= delta
                delta /= 2
            else:
                theta1 += delta
        # 计算相邻点的极坐标
        r1 = a[0] * theta1
        x1 = r1 * sm.cos(theta1)
        y1 = r1 * sm.sin(theta1)
        if r1 <= a[0] * 32 * sm.pi + 0.55:  # 这里的微小扰动原理同上
            if flag <= 221:
                df_3.loc['第' + str(flag) + '节龙身x (m)', str(t) + ' s'] = x1
                df_3.loc['第' + str(flag) + '节龙身y (m)', str(t) + ' s'] = y1
            elif flag == 222:
                df_3.loc['龙尾x (m)', str(t) + ' s'] = x1
                df_3.loc['龙尾y (m)', str(t) + ' s'] = y1
            elif flag == 223:
                df_3.loc['龙尾（后）x (m)', str(t) + ' s'] = x1
                df_3.loc['龙尾（后）y (m)', str(t) + ' s'] = y1
            else:
                break
        x0 = r1 * sm.cos(theta1)
        y0 = r1 * sm.sin(theta1)
        r = r1
```

```
        theta0 = theta1

# 下面开始计算0.0001s的
for t in range(0, 301):
    print('第' + str(t) + 's计算')
    # 进行积分，传递正确的积分上下限
    fx = sm.integrate(a[0] * sm.sqrt(x ** 2 + 1), (x, b, sm.pi * 32)) - t - 0.0001  # 0.0001s为设定的间隔时间
    # 解方程，求解 b
    res = sm.nsolve(fx, b, 16)
    # 求解 r x y
    r = res * a[0]
    x_t = r * sm.cos(res)
    y_t = r * sm.sin(res)
    df_4.loc['龙头x (m)', str(t) + ' s'] = x_t
    df_4.loc['龙头y (m)', str(t) + ' s'] = y_t
    flag = 0  # 判断是不是龙头的标志
    theta0 = res
    while r < a[0] * 32 * sm.pi + 0.55:  # 这里0.55为手动添加的微小扰动，防止某点在正秒时在螺线范围内，但0.0001s前在螺线外
        # 定义距离差
        if flag == 0:
            delta_d = 2.86
            theta1 = theta0  # 初始角度，可以根据需要调整
            flag += 1
            x0 = x_t
            y0 = y_t
        else:
            delta_d = 1.65
            theta1 = theta0
            flag += 1
        delta = 0.5
        # 牛顿法求解
        while abs(f(theta1, x0, y0, delta_d)) > 1e-8:
            if f(theta1, x0, y0, delta_d) > 0:
                theta1 -= delta
                delta /= 2
            else:
                theta1 += delta
        # 计算相邻点的极坐标
        r1 = a[0] * theta1
        x1 = r1 * sm.cos(theta1)
        y1 = r1 * sm.sin(theta1)
        if r1 <= a[0] * 32 * sm.pi + 0.55:  # 这里的微小扰动原理同上
```

```
            if flag <= 221:
                df_4.loc['第' + str(flag) + '节龙身x (m)', str(t) + ' s'] = x1
                df_4.loc['第' + str(flag) + '节龙身y (m)', str(t) + ' s'] = y1
            elif flag == 222:
                df_4.loc['龙尾x (m)', str(t) + ' s'] = x1
                df_4.loc['龙尾y (m)', str(t) + ' s'] = y1
            elif flag == 223:
                df_4.loc['龙尾（后）x (m)', str(t) + ' s'] = x1
                df_4.loc['龙尾（后）y (m)', str(t) + ' s'] = y1
            else:
                break
        x0 = r1 * sm.cos(theta1)
        y0 = r1 * sm.sin(theta1)
        r = r1
        theta0 = theta1

# 开始用df_1、df_3、df_4对应点的坐标距离差分计算第1节龙身到221节龙身速度
for i in range(1, 222):
    for t in range(0, 301):
        if math.isnan(df_1.at['第' + str(i) + '节龙身x (m)', str(t) + ' s']) is not True:
            x_a = df_1.at['第' + str(i) + '节龙身x (m)', str(t) + ' s']  # 某时刻某点的x坐标
            y_a = df_1.at['第' + str(i) + '节龙身y (m)', str(t) + ' s']  # 某时刻某点的y坐标
            x_b = df_3.at['第' + str(i) + '节龙身x (m)', str(t) + ' s']  # 某时刻某点前0.0001s的x坐标
            y_b = df_3.at['第' + str(i) + '节龙身y (m)', str(t) + ' s']  # 某时刻某点前0.0001s的y坐标
            x_c = df_4.at['第' + str(i) + '节龙身x (m)', str(t) + ' s']  # 某时刻某点后0.0001s的x坐标
            y_c = df_4.at['第' + str(i) + '节龙身y (m)', str(t) + ' s']  # 某时刻某点后0.0001s的y坐标
            D1 = length(x_a, y_a, x_b, y_b)
            D2 = length(x_a, y_a, x_c, y_c)
            v = (D2 + D1) / 0.0002  # 某时刻某点的速度
            df_2.loc['第' + str(i) + '节龙身 (m/s)', str(t) + ' s'] = v

# 计算两个龙尾速度
for t in range(0, 301):
```

```
    if math.isnan(df_1.at['龙尾x (m)', str(t) + ' s']) is not True:
        x_a = df_1.at['龙尾x (m)', str(t) + ' s']  # 某时刻前龙尾的x坐标
        y_a = df_1.at['龙尾y (m)', str(t) + ' s']  # 某时刻前龙尾的y坐标
        x_b = df_3.at['龙尾x (m)', str(t) + ' s']  # 某时刻前龙尾前0.0001s的x坐标
        y_b = df_3.at['龙尾y (m)', str(t) + ' s']  # 某时刻前龙尾前0.0001s的y坐标
        x_c = df_4.at['龙尾x (m)', str(t) + ' s']  # 某时刻前龙尾后0.0001s的x坐标
        y_c = df_4.at['龙尾y (m)', str(t) + ' s']  # 某时刻前龙尾后0.0001s的y坐标
        D1 = length(x_a, y_a, x_b, y_b)
        D2 = length(x_a, y_a, x_c, y_c)
        v = (D2 + D1) / 0.0002  # 某时刻前龙尾的速度
        df_2.loc['龙尾  (m/s)', str(t) + ' s'] = v
for t in range(0, 301):
    if math.isnan(df_1.at['龙尾（后）x (m)', str(t) + ' s']) is not True:
        x_a = df_1.at['龙尾（后）x (m)', str(t) + ' s']  # 某时刻后龙尾的x坐标
        y_a = df_1.at['龙尾（后）y (m)', str(t) + ' s']  # 某时刻后龙尾的y坐标
        x_b = df_3.at['龙尾（后）x (m)', str(t) + ' s']  # 某时刻后龙尾前0.0001s的x坐标
        y_b = df_3.at['龙尾（后）y (m)', str(t) + ' s']  # 某时刻后龙尾前0.0001s的y坐标
        x_c = df_4.at['龙尾（后）x (m)', str(t) + ' s']  # 某时刻后龙尾后0.0001s的x坐标
        y_c = df_4.at['龙尾（后）y (m)', str(t) + ' s']  # 某时刻后龙尾后0.0001s的y坐标
        D1 = length(x_a, y_a, x_b, y_b)
        D2 = length(x_a, y_a, x_c, y_c)
        v = (D2 + D1) / 0.0002  # 某时刻后龙尾的速度
        df_2.loc['龙尾（后） (m/s)', str(t) + ' s'] = v

# 写入文件
with pd.ExcelWriter(file_path, engine='xlsxwriter') as writer:
    df_1.to_excel(writer, sheet_name="位置", float_format="%.6f")
    df_2.to_excel(writer, sheet_name="速度", float_format="%.6f")
```

附录 3

第二问第一部分代码（计算碰撞时间）

```
import sympy as sm


# 将列表中所有元素求和，便于后续统计是否有相交
def sum_up(l):
    add_all = 0
    for i in l:
        add_all += i
    if add_all > 0:
        return False
    else:
        return True


# 根据板凳两个把手获取该板凳的四个顶点以便于后续相交分析
def get_point_of_rectangle(x0, y0, x1, y1, delta):
    p1 = (x0 + 0.15 * (y1 - y0) / delta + 0.275 * (x0 - x1) / delta,
          y0 + 0.15 * (x0 - x1) / delta + 0.275 * (y0 - y1) / delta)
    p2 = (x1 + 0.15 * (y1 - y0) / delta - 0.275 * (x0 - x1) / delta,
          y1 + 0.15 * (x0 - x1) / delta - 0.275 * (y0 - y1) / delta)
    p3 = (x1 - 0.15 * (y1 - y0) / delta - 0.275 * (x0 - x1) / delta,
          y1 - 0.15 * (x0 - x1) / delta - 0.275 * (y0 - y1) / delta)
    p4 = (x0 - 0.15 * (y1 - y0) / delta + 0.275 * (x0 - x1) / delta,
          y0 - 0.15 * (x0 - x1) / delta + 0.275 * (y0 - y1) / delta)
    return [p1, p2, p3, p4]


def f(theta, X, Y, D):
    return sm.sqrt((a * theta * sm.cos(theta) - X) ** 2 + (a * theta
    * sm.sin(theta) - Y) ** 2) - D


# 判断点是否在四边形内（射线相交法）
def point_in_irregular_quadrilateral(point, vertices):
    x, y = point
    num_vertices = len(vertices)
    num_crossings = 0
    for i in range(num_vertices):
        x1, y1 = vertices[i]
        x2, y2 = vertices[(i + 1) % num_vertices]
```

```
        if (y1 > y) != (y2 > y) and x < ((x2 - x1) * (y - y1) / (y2 - y1) + x1):
            num_crossings += 1
    return num_crossings % 2 == 1


# 定义符号
x = sm.symbols("x")  # 方位角
b = sm.symbols("b")  # 积分下限，第 t 秒的方位角
a = sm.symbols("a")  # 螺线参数

# 求解螺线参数 a
a = sm.solve(2 * sm.pi * a - 0.55, a)[0]

time = 300  # 初始化时间
delta_time = 20  # 二分法时间变化

while abs(delta_time >= 1e-8):
    results = []
    # 进行积分，传递正确的积分上下限
    fx = sm.integrate(a * sm.sqrt(x ** 2 + 1), (x, b, sm.pi * 32)) - time
    # 解方程，求解 b
    theta0 = sm.nsolve(fx, b, 16)
    # 求解 r x y
    r = theta0 * a
    x_t1 = r * sm.cos(theta0)
    y_t1 = r * sm.sin(theta0)
    flag = 0
    while flag <= 30:
        theta1 = theta0  # 初始角度，可以根据需要调整
        # 定义距离差
        if flag == 0:
            delta_d = 2.86
            x0 = x_t1
            y0 = y_t1
        else:
            delta_d = 1.65
        flag += 1
        delta = 0.5
        # 牛顿法求解
        while abs(f(theta1, x0, y0, delta_d)) > 1e-8:
            if f(theta1, x0, y0, delta_d) > 0:
                theta1 -= delta
```

```
                delta /= 2
            else:
                theta1 += delta
        # 计算相邻点的极坐标
        r1 = a * theta1
        x1 = r1 * sm.cos(theta1)
        y1 = r1 * sm.sin(theta1)
        if flag == 1:
            x_t2 = x1
            y_t2 = y1
            [p1, p2, p3, p4] = get_point_of_rectangle(x_t1, y_t1, x_t2, y_t2, delta_d)
        else:
            [pt1, pt2, pt3, pt4] = get_point_of_rectangle(x0, y0, x1, y1, delta_d)
            result1 = point_in_irregular_quadrilateral(p1, [pt1, pt2, pt3, pt4])
            result2 = point_in_irregular_quadrilateral(p2, [pt1, pt2, pt3, pt4])
            result3 = point_in_irregular_quadrilateral(p3, [pt1, pt2, pt3, pt4])
            result4 = point_in_irregular_quadrilateral(p4, [pt1, pt2, pt3, pt4])
            if result1 is not True and result2 is not True and result3 is not True and result4 is not True:
                results.append(0)
             else:
                if flag == 2:
                    results.append(0)  # 龙头和第一节龙身一定相交，为方便记作 0
                else:
                    results.append(1)  # 相交为 1
        x0 = r1 * sm.cos(theta1)
        y0 = r1 * sm.sin(theta1)
        r = r1
        theta0 = theta1
    print(results)
    if sum_up(results):
        time += delta_time
    else:
        time -= delta_time
        delta_time /= 2
    print(delta_time)

```

```
118. print("time:", time)
```

附录 4

第二问第二部分代码（计算位置和速度）

```
1. import pandas as pd
2. import sympy as sm
3. import math
4.
5.
6. def length(x1, y1, x2, y2):
7.     return sm.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
8.
9.
10.def f(theta, X, Y, D):
11.    return sm.sqrt((a[0] * theta * sm.cos(theta) - X) ** 2 + (a[0] *
    theta * sm.sin(theta) - Y) ** 2) - D
12.
13.
14.file_path = 'result2.xlsx'
15.time = 412.47383765876293  # q2(1).py 运行出的盘入终止时间
16.
17.# 定义符号
18.x = sm.symbols("x")  # 方位角
19.b = sm.symbols("b")  # 积分下限，第 t 秒的方位角
20.a = sm.symbols("a")  # 螺线参数
21.
22.# 求解螺线参数 a
23.a = sm.solve(2 * sm.pi * a - 0.55, a)
24.
25.df = pd.read_excel(file_path, index_col=0, sheet_name='Sheet1')
26.df_2 = df.copy()  # 存放第-0.0001 秒的运动坐标
27.df_3 = df.copy()  # 存放第 0.0001 秒的运动坐标
28.
29.print('第' + str(time) + 's 计算')
30.# 进行积分，传递正确的积分上下限
31.fx = sm.integrate(a[0] * sm.sqrt(x ** 2 + 1), (x, b, sm.pi * 32)) -
   time
32.# 解方程，求解 b
33.res = sm.nsolve(fx, b, 16)
34.# 求解 r x y
35.r = res * a[0]
36.print('半径：', r)
```

```
x_t = r * sm.cos(res)
y_t = r * sm.sin(res)
df.loc['龙头', '横坐标 x (m)'] = x_t
df.loc['龙头', '纵坐标 y (m)'] = y_t
flag = 0  # 判断是不是龙头的标志
theta0 = res
while r < a[0] * 32 * sm.pi:
    # 定义距离差
    if flag == 0:
        delta_d = 2.86
        theta1 = theta0  # 初始角度，可以根据需要调整
        flag += 1
        x0 = x_t
        y0 = y_t
    else:
        delta_d = 1.65
        theta1 = theta0
        flag += 1
    delta = 0.5
    # 牛顿法求解
    while abs(f(theta1, x0, y0, delta_d)) > 1e-8:
        if f(theta1, x0, y0, delta_d) > 0:
            theta1 -= delta
            delta /= 2
        else:
            theta1 += delta
    # 计算相邻点的极坐标
    r1 = a[0] * theta1
    x1 = r1 * sm.cos(theta1)
    y1 = r1 * sm.sin(theta1)
    if r1 <= a[0] * 32 * sm.pi:
        print(flag)
        if flag <= 221:
            df.loc['第' + str(flag) + '节龙身', '横坐标 x (m)'] = x1
            df.loc['第' + str(flag) + '节龙身', '纵坐标 y (m)'] = y1
        elif flag == 222:
            df.loc['龙尾', '横坐标 x (m)'] = x1
            df.loc['龙尾', '纵坐标 y (m)'] = y1
        elif flag == 223:
            df.loc['龙尾（后）', '横坐标 x (m)'] = x1
            df.loc['龙尾（后）', '纵坐标 y (m)'] = y1
        else:
            break
    x0 = r1 * sm.cos(theta1)
```

```
    y0 = r1 * sm.sin(theta1)
    r = r1
    theta0 = theta1

# 显示数据框内容
print(df)

# 下面开始计算-0.0001s 的（点是否在螺线内不管，在前面 df 中已经控制）
# 这里只是为了计算-0.0001s 时各点位置便于速度计算，最终某时刻某点的速度是否存在以 df 中对应时间的该点是否有坐标来判断
print('第' + str(time) + 's 计算')
# 进行积分，传递正确的积分上下限
fx = sm.integrate(a[0] * sm.sqrt(x ** 2 + 1), (x, b, sm.pi * 32)) - time + 0.0001  # 0.0001s 为设定的间隔时间
# 解方程，求解 b
res = sm.nsolve(fx, b, 16)
# 求解 r x y
r = res * a[0]
x_t = r * sm.cos(res)
y_t = r * sm.sin(res)
df_2.loc['龙头', '横坐标 x (m)'] = x_t
df_2.loc['龙头', '纵坐标 y (m)'] = y_t
flag = 0  # 判断是不是龙头的标志
theta0 = res
while r < a[0] * 32 * sm.pi:
    # 定义距离差
    if flag == 0:
        delta_d = 2.86
        theta1 = theta0  # 初始角度，可以根据需要调整
        flag += 1
        x0 = x_t
        y0 = y_t
    else:
        delta_d = 1.65
        theta1 = theta0
        flag += 1
    delta = 0.5
    # 牛顿法求解
    while abs(f(theta1, x0, y0, delta_d)) > 1e-8:
        if f(theta1, x0, y0, delta_d) > 0:
            theta1 -= delta
            delta /= 2
        else:
            theta1 += delta
```

```
    # 计算相邻点的极坐标
    r1 = a[0] * theta1
    x1 = r1 * sm.cos(theta1)
    y1 = r1 * sm.sin(theta1)
    if r1 <= a[0] * 32 * sm.pi:
        print(flag)
        if flag <= 221:
            df_2.loc['第' + str(flag) + '节龙身', '横坐标x (m)'] = x1
            df_2.loc['第' + str(flag) + '节龙身', '纵坐标y (m)'] = y1
        elif flag == 222:
            df_2.loc['龙尾', '横坐标x (m)'] = x1
            df_2.loc['龙尾', '纵坐标y (m)'] = y1
        elif flag == 223:
            df_2.loc['龙尾（后）', '横坐标x (m)'] = x1
            df_2.loc['龙尾（后）', '纵坐标y (m)'] = y1
        else:
            break
    x0 = r1 * sm.cos(theta1)
    y0 = r1 * sm.sin(theta1)
    r = r1
    theta0 = theta1

# 下面开始计算0.0001s的（点是否在螺线内不管，在前面df中已经控制）
# 这里只是为了计算0.0001s时各点位置便于速度计算，最终某时刻某点的速度是否存在以df中对应时间的该点是否有坐标来判断
print('第' + str(time) + 's计算')
# 进行积分，传递正确的积分上下限
fx = sm.integrate(a[0] * sm.sqrt(x ** 2 + 1), (x, b, sm.pi * 32)) - time - 0.0001  # 0.0001s为设定的间隔时间
# 解方程，求解 b
res = sm.nsolve(fx, b, 16)
# 求解 r x y
r = res * a[0]
x_t = r * sm.cos(res)
y_t = r * sm.sin(res)
df_3.loc['龙头', '横坐标x (m)'] = x_t
df_3.loc['龙头', '纵坐标y (m)'] = y_t
flag = 0  # 判断是不是龙头的标志
theta0 = res
while r < a[0] * 32 * sm.pi:
    # 定义距离差
    if flag == 0:
```

```
            delta_d = 2.86
            theta1 = theta0  # 初始角度，可以根据需要调整
            flag += 1
            x0 = x_t
            y0 = y_t
        else:
            delta_d = 1.65
            theta1 = theta0
            flag += 1
        delta = 0.5
        # 牛顿法求解
        while abs(f(theta1, x0, y0, delta_d)) > 1e-8:
            if f(theta1, x0, y0, delta_d) > 0:
                theta1 -= delta
                delta /= 2
            else:
                theta1 += delta
        # 计算相邻点的极坐标
        r1 = a[0] * theta1
        x1 = r1 * sm.cos(theta1)
        y1 = r1 * sm.sin(theta1)
        if r1 <= a[0] * 32 * sm.pi:
            print(flag)
            if flag <= 221:
                df_3.loc['第' + str(flag) + '节龙身', '横坐标
    x (m)'] = x1
                df_3.loc['第' + str(flag) + '节龙身', '纵坐标
    y (m)'] = y1
            elif flag == 222:
                df_3.loc['龙尾', '横坐标 x (m)'] = x1
                df_3.loc['龙尾', '纵坐标 y (m)'] = y1
            elif flag == 223:
                df_3.loc['龙尾（后）', '横坐标 x (m)'] = x1
                df_3.loc['龙尾（后）', '纵坐标 y (m)'] = y1
            else:
                break
        x0 = r1 * sm.cos(theta1)
        y0 = r1 * sm.sin(theta1)
        r = r1
        theta0 = theta1

df.loc['龙头', '速度 (m/s)'] = 1.000000
# 开始用 df、df_2、df_3 对应点的坐标距离差分计算第 1 节龙身到 221 节龙身速
    度
```

```
for i in range(1, 222):
    if math.isnan(df.at['第' + str(i) + '节龙身', '横坐标 x (m)']) is not True:
        x_a = df.at['第' + str(i) + '节龙身', '横坐标 x (m)']  # 某点的x坐标
        y_a = df.at['第' + str(i) + '节龙身', '纵坐标 y (m)']  # 某点的y坐标
        x_b = df_2.at['第' + str(i) + '节龙身', '横坐标 x (m)']  # 某点前0.0001s的x坐标
        y_b = df_2.at['第' + str(i) + '节龙身', '纵坐标 y (m)']  # 某点前0.0001s的y坐标
        x_c = df_3.at['第' + str(i) + '节龙身', '横坐标 x (m)']  # 某点后0.0001s的x坐标
        y_c = df_3.at['第' + str(i) + '节龙身', '纵坐标 y (m)']  # 某点后0.0001s的y坐标
        D1 = length(x_a, y_a, x_b, y_b)
        D2 = length(x_a, y_a, x_c, y_c)
        v = (D2 + D1) / 0.0002  # 某点的速度
        df.loc['第' + str(i) + '节龙身', '速度 (m/s)'] = v

# 计算两个龙尾速度
# 前龙尾
if math.isnan(df.at['龙尾', '横坐标 x (m)']) is not True:
    x_a = df.at['龙尾', '横坐标 x (m)']  # 前龙尾的x坐标
    y_a = df.at['龙尾', '纵坐标 y (m)']  # 前龙尾的y坐标
    x_b = df_2.at['龙尾', '横坐标 x (m)']  # 前龙尾前0.0001s的x坐标
    y_b = df_2.at['龙尾', '纵坐标 y (m)']  # 前龙尾前0.0001s的y坐标
    x_c = df_3.at['龙尾', '横坐标 x (m)']  # 前龙尾后0.0001s的x坐标
    y_c = df_3.at['龙尾', '纵坐标 y (m)']  # 前龙尾后0.0001s的y坐标
    D1 = length(x_a, y_a, x_b, y_b)
    D2 = length(x_a, y_a, x_c, y_c)
    v = (D2 + D1) / 0.0002  # 某时刻前龙尾的速度
    df.loc['龙尾', '速度 (m/s)'] = v

# 后龙尾
if math.isnan(df.at['龙尾（后）', '横坐标 x (m)']) is not True:
    x_a = df.at['龙尾（后）', '横坐标 x (m)']  # 后龙尾的x坐标
    y_a = df.at['龙尾（后）', '纵坐标 y (m)']  # 后龙尾的y坐标
    x_b = df_2.at['龙尾（后）', '横坐标 x (m)']  # 后龙尾前0.0001s的x坐标
    y_b = df_2.at['龙尾（后）', '纵坐标 y (m)']  # 后龙尾前0.0001s的y坐标
    x_c = df_3.at['龙尾（后）', '横坐标 x (m)']  # 后龙尾后0.0001s的x坐标
```

```
    y_c = df_3.at['龙尾（后）', '纵坐标 y (m)']  # 后龙尾后0.0001s 的 y 坐标
    D1 = length(x_a, y_a, x_b, y_b)
    D2 = length(x_a, y_a, x_c, y_c)
    v = (D2 + D1) / 0.0002  # 某时刻后龙尾的速度
    df.loc['龙尾（后）', '速度 (m/s)'] = v

# 保存数据
df.to_excel(file_path, sheet_name="Sheet1")
```

附录 5

第三问代码

```
import sympy as sm


def sum_up(l):
    add_all = 0
    for i in l:
        add_all += i
    if add_all > 0:
        return False
    else:
        return True


# 根据板凳两个把手获取该板凳的四个顶点以便于后续相交分析
def get_point_of_rectangle(x0, y0, x1, y1, delta):
    p1 = (x0 + 0.15 * (y1 - y0) / delta + 0.275 * (x0 - x1) / delta,
          y0 + 0.15 * (x0 - x1) / delta + 0.275 * (y0 - y1) / delta)
    p2 = (x1 + 0.15 * (y1 - y0) / delta - 0.275 * (x0 - x1) / delta,
          y1 + 0.15 * (x0 - x1) / delta - 0.275 * (y0 - y1) / delta)
    p3 = (x1 - 0.15 * (y1 - y0) / delta - 0.275 * (x0 - x1) / delta,
          y1 - 0.15 * (x0 - x1) / delta - 0.275 * (y0 - y1) / delta)
    p4 = (x0 - 0.15 * (y1 - y0) / delta + 0.275 * (x0 - x1) / delta,
          y0 - 0.15 * (x0 - x1) / delta + 0.275 * (y0 - y1) / delta)
    return [p1, p2, p3, p4]


def f(theta, X, Y, D):
    return sm.sqrt((a * theta * sm.cos(theta) - X) ** 2 + (a * theta * sm.sin(theta) - Y) ** 2) - D

```

```
30.
31.# 判断点是否在四边形内（射线相交法）
32.def point_in_irregular_quadrilateral(point, vertices):
33.    x, y = point
34.    num_vertices = len(vertices)
35.    num_crossings = 0
36.    for i in range(num_vertices):
37.        x1, y1 = vertices[i]
38.        x2, y2 = vertices[(i + 1) % num_vertices]
39.        if (y1 > y) != (y2 > y) and x < ((x2 - x1) * (y - y1) / (y2
   - y1) + x1):
40.            num_crossings += 1
41.    return num_crossings % 2 == 1
42.
43.
44.def calculate_intersection_true_or_false(a):
45.    time = 300  # 初始化时间
46.    delta_time = 50  # 二分法时间变化
47.    x = sm.symbols("x")  # 方位角
48.    b = sm.symbols("b")  # 积分下限，第 t 秒的方位角
49.    while abs(delta_time >= 1e-8):
50.        results = []
51.        # 进行积分，传递正确的积分上下限
52.        fx = sm.integrate(a * sm.sqrt(x ** 2 + 1), (x, b, sm.pi * 32
   )) - time
53.        # 解方程，求解 b
54.        theta0 = sm.nsolve(fx, b, 16)
55.        # 求解 r x y
56.        r = theta0 * a
57.        x_t1 = r * sm.cos(theta0)
58.        y_t1 = r * sm.sin(theta0)
59.        flag = 0
60.        while flag <= 100:
61.            theta1 = theta0  # 初始角度，可以根据需要调整
62.            # 定义距离差
63.            if flag == 0:
64.                delta_d = 2.86
65.                x0 = x_t1
66.                y0 = y_t1
67.            else:
68.                delta_d = 1.65
69.            flag += 1
70.            delta = 0.5
71.            # 牛顿法求解
```

```
72.             while abs(f(theta1, x0, y0, delta_d)) > 1e-8:
73.                 if f(theta1, x0, y0, delta_d) > 0:
74.                     theta1 -= delta
75.                     delta /= 2
76.                 else:
77.                     theta1 += delta
78.             # 计算相邻点的极坐标
79.             r1 = a * theta1
80.             x1 = r1 * sm.cos(theta1)
81.             y1 = r1 * sm.sin(theta1)
82.             if flag == 1:
83.                 x_t2 = x1
84.                 y_t2 = y1
85.                 [p1, p2, p3, p4] = get_point_of_rectangle(x_t1, y_t1
    , x_t2, y_t2, delta_d)
86.             else:
87.                 [pt1, pt2, pt3, pt4] = get_point_of_rectangle(x0, y0
    , x1, y1, delta_d)
88.                 result1 = point_in_irregular_quadrilateral(p1, [pt1,
     pt2, pt3, pt4])
89.                 result2 = point_in_irregular_quadrilateral(p2, [pt1,
     pt2, pt3, pt4])
90.                 result3 = point_in_irregular_quadrilateral(p3, [pt1,
     pt2, pt3, pt4])
91.                 result4 = point_in_irregular_quadrilateral(p4, [pt1,
     pt2, pt3, pt4])
92.                 if result1 is not True and result2 is not True and r
    esult3 is not True and result4 is not True:
93.                     results.append(0)
94.                 else:
95.                     if flag == 2:
96.                         results.append(0)  # 龙头和第一节龙身一定相
    交，为方便记作 0
97.                     else:
98.                         results.append(1)  # 相交为 1
99.             x0 = r1 * sm.cos(theta1)
100.             y0 = r1 * sm.sin(theta1)
101.             theta0 = theta1
102.         print(delta_time)
103.         if sum_up(results):
104.             time += delta_time
105.         else:
106.             time -= delta_time
107.             delta_time /= 2
```

```
108.
109.     print("time:", time)
110.     # 定义符号
111.     xp = sm.symbols("x")  # 方位角
112.     bp = sm.symbols("b")  # 积分下限，第 t 秒的方位角
113.     print('第' + str(time) + 's 计算')
114.     # 进行积分，传递正确的积分上下限
115.     fxp = sm.integrate(a * sm.sqrt(xp ** 2 + 1), (xp, bp, sm.pi * 3
    2)) - time
116.     # 解方程，求解 b
117.     resp = sm.nsolve(fxp, bp, 16)
118.     # 求解 r x y
119.     rp = resp * a
120.     print("极轴:", rp)
121.     return rp
122.
123.
124. p = 0.55
125. delta_p = 0.05
126. # 定义符号螺线参数
127. a = sm.symbols("a")
128. # 求解螺线参数 a
129. a = sm.solve(2 * sm.pi * a - p, a)[0]
130. distance = calculate_intersection_true_or_false(a)
131. while abs(distance - 4.5) > 1e-8:
132.     print("p:", p)
133.     if distance <= 4.5:
134.         p -= delta_p
135.         # 定义符号螺线参数
136.         a = sm.symbols("a")
137.         # 求解螺线参数 a
138.         a = sm.solve(2 * sm.pi * a - p, a)[0]
139.         distance = calculate_intersection_true_or_false(a)
140.     else:
141.         p += delta_p
142.         delta_p /= 2
143.         # 定义符号螺线参数
144.         a = sm.symbols("a")
145.         # 求解螺线参数 a
146.         a = sm.solve(2 * sm.pi * a - p, a)[0]
147.         distance = calculate_intersection_true_or_false(a)
148.
149. print(p)
```

附录 6

第四问第一部分代码

```
p = 1.7; % 螺距
a = p / (2 * pi); % 螺线系数
d1 = 2.86; % 龙头两把手间距
d2 = 1.65; % 除龙头板外其他板两把手间距
v0 = 1; % 龙头运动速度
R=4.5; % 掉头区域半径

% 绘制中心对称螺线
theta=5*2*pi:-0.01:0*pi;
r=a*theta;
x=r.*cos(theta);
y=r.*sin(theta);
figure(1)
hold on
set(gcf,'Position',[200 200 600 600]);
title('盘入、盘出螺线及掉头圆弧')
plot(x,y,'-','Color',[0.5, 0.5, 0.5],'DisplayName','盘入螺线')
axis equal
grid on
xlabel('x 轴/m')
ylabel('y 轴/m')

% 中心对称图形，方位角相差 180°
theta=theta-pi;
ra=a*(theta+pi);
x2=ra.*cos(theta);
y2=ra.*sin(theta);
plot(x2,y2,'m','Color','black','DisplayName','盘出螺线')
% 绘制掉头区域
x_diao=R*cos(theta);
y_diao=R*sin(theta);
plot(x_diao,y_diao,'Color','red','LineStyle','--
','LineWidth',2,'DisplayName','掉头区域')

% 求两段圆弧的位置、切入切出点
theta_ru=R/a;
theta_chu=R/a-pi; % 另一个螺线
slope=(a*sin(theta_ru)+R*cos(theta_ru))/(a*cos(theta_ru)-
R*sin(theta_ru));
theta_max_a=atan(-1/slope)+pi;
```

```
theta_equal=atan(tan(theta_ru))+pi-theta_max_a;
rC1_C2=R/cos(theta_equal);
r2=rC1_C2/3;
r1=r2*2;
phi=2*theta_equal;
SC1=r1*(pi-phi); SC2=r2*(pi-phi);
theta_min_a=theta_max_a-SC1/r1;
theta_min_2=theta_min_a-pi;
theta_max_2=theta_min_2+SC2/r2;
% 两段弧圆心坐标
x_C1=R*cos(theta_ru)+r1*cos(theta_max_a-pi);
y_C1=R*sin(theta_ru)+r1*sin(theta_max_a-pi);
x_C2=R*cos(theta_chu)-r2*cos(theta_max_2);
y_C2=R*sin(theta_chu)-r2*sin(theta_max_2);

% 继续完善图，添加绘制两段弧
hold on
plot(x_C1+r1*cos(linspace(theta_min_a,theta_max_a,50)),y_C1+r1*sin(linspace(theta_min_a,theta_max_a,50)),'Color','magenta','LineWidth',2)
plot(x_C1,y_C1,'ko','MarkerSize',5,'MarkerFaceColor','magenta')
plot(x_C2+r2*cos(linspace(theta_min_2,theta_max_2,50)),y_C2+r2*sin(linspace(theta_min_2,theta_max_2,50)),'Color','magenta','LineWidth',2)
plot(x_C2,y_C2,'ko','MarkerSize',5,'MarkerFaceColor','magenta')
axis equal
set(gca, 'looseInset', [0 0 0 0]);

% 盘入曲线上头节点的位置求解
my_theta=@(t,theta)1./(a*sqrt(1+theta.^2));
theta0=theta_ru;
dt=0.1; % 时间步长
t_span=0:dt:100;
[ttl,theta]=ode45(my_theta,t_span,theta0); % 龙格库塔法求解
X1=a*theta.*cos(theta);
Y1=a*theta.*sin(theta);
tt_ru=ttl(end:-1:1);
X=zeros(224,200/dt+1);
Y=zeros(224,200/dt+1);
Ting=zeros(224,200/dt+1);
X(1,1:length(X1))=X1(end:-1:1);
Y(1,1:length(Y1))=Y1(end:-1:1);
Ting(1,1:length(theta))=theta(end:-1:1);
% 圆弧上切点的位置信息
```

```
td_c1=dt:dt:SC1;
theta_C1=-td_c1/r1+theta_max_a;
Ting(1,length(theta)+(1:length(td_c1)))=theta_C1;
X(1,length(X1)+(1:length(td_c1)))=r1*cos(theta_C1)+x_C1;
Y(1,length(Y1)+(1:length(td_c1)))=r1*sin(theta_C1)+y_C1;
td_c2=td_c1(end)+dt:dt:SC1+SC2;
theta_C2=(td_c2-SC1)/r2+theta_min_a-pi;
Ting(1,length(theta)+length(theta_C1)+(1:length(td_c2)))=theta_C2;
X(1,length(X1)+length(theta_C1)+(1:length(td_c2)))=r2*cos(theta_C2)+
    x_C2;
Y(1,length(Y1)+length(theta_C1)+(1:length(td_c2)))=r2*sin(theta_C2)+
    y_C2;
my_theta=@(t,theta)1./(a*sqrt(1+(theta+pi).^2));
theta0=theta_chu;
t_span=td_c2(end)+dt:dt:100;
[ttl,theta2]=ode45(my_theta,t_span,theta0); % 龙格库塔法求解
X2=a*(theta2+pi).*cos(theta2);
Y2=a*(theta2+pi).*sin(theta2);
Ting(1,length(theta)+length(theta_C1)+length(td_c2)+1:end)=theta2;
X(1,length(theta)+length(theta_C1)+length(td_c2)+1:end)=X2;
Y(1,length(theta)+length(theta_C1)+length(td_c2)+1:end)=Y2;
set(gcf,'Position',[300 300 600 600])
waitfor(gcf);

% 绘制第二张图
figure(2)
for i=1:3:length(Ting(1,:))
    title('头把手中心的轨迹(-100 到 100s)')
    xlabel('x 轴/m')
    ylabel('y 轴/m')
    plot(X(1,i),Y(1,i),'Marker','o','MarkerSize',2,'MarkerFaceColor
    ','r','Color','cyan')
    hold on
    axis equal
    axis([-10 10 -10 10])
    grid on
    drawnow
end
set(gca, 'looseInset', [0 0 0 0]);
waitfor(gcf);

% 求各段龙身的位置
N=223;
t_total=-100:dt:100;
```

```
for j=1:length(t_total)
    if t_total(j)<=0
        for i=2:N+1
            d=d1*(i<=2)+d2*(i>2);
            theta_xy=solve_theta1(p,X(i-1,j),Y(i-1,j),Ting(i-1,j),d);
            Ting(i,j)=theta_xy;
            X(i,j)=a*theta_xy*cos(theta_xy);
            Y(i,j)=a*theta_xy*sin(theta_xy);
        end
    elseif t_total(j)>0 && t_total(j)<=SC1
        f=2;
        for i=2:N+1
            d=d1*(i<=2)+d2*(i>2);
            if f==2
                [xi,yi,arf,f]=solve_point_2_1(p,X(i-1,j),Y(i-1,j),Ting(i-1,j),d,r1,x_C1,y_C1,theta_max_a);
                Ting(i,j)=arf;
                X(i,j)=xi;
                Y(i,j)=yi;
            else
                theta_xy=solve_theta1(p,X(i-1,j),Y(i-1,j),Ting(i-1,j),d);
                Ting(i,j)=theta_xy;
                X(i,j)=a*theta_xy*cos(theta_xy);
                Y(i,j)=a*theta_xy*sin(theta_xy);
            end
        end
    elseif t_total(j)>SC1 && t_total(j)<=SC1+SC2
        f=3;
        for i=2:N+1
            d=d1*(i<=2)+d2*(i>2);
            if f==3
                [xi,yi,arf,f]=solve_point_3_2(X(i-1,j),Y(i-1,j),Ting(i-1,j),d,r1,x_C1,y_C1,r2,x_C2,y_C2,theta_min_2);
                Ting(i,j)=arf;
                X(i,j)=xi;
                Y(i,j)=yi;
            elseif f==2
                [xi,yi,arf,f]=solve_point_2_1(p,X(i-1,j),Y(i-1,j),Ting(i-1,j),d,r1,x_C1,y_C1,theta_max_a);
                Ting(i,j)=arf;
                X(i,j)=xi;
                Y(i,j)=yi;
```

```
            else
                theta_xy=solve_theta1(p,X(i-1,j),Y(i-1,j),Ting(i-1,j),d);
                Ting(i,j)=theta_xy;
                X(i,j)=a*theta_xy*cos(theta_xy);
                Y(i,j)=a*theta_xy*sin(theta_xy);
            end
        end
    else
        f=4;
        for i=2:N+1
            d=d1*(i<=2)+d2*(i>2);
            if f==4
                [xi,yi,arf,f]=solve_point_4_3(p,X(i-1,j),Y(i-1,j),Ting(i-1,j),d,r2,x_C2,y_C2,theta_max_2);
                Ting(i,j)=arf;
                X(i,j)=xi;
                Y(i,j)=yi;
            elseif f==3
                [xi,yi,arf,f]=solve_point_3_2(X(i-1,j),Y(i-1,j),Ting(i-1,j),d,r1,x_C1,y_C1,r2,x_C2,y_C2,theta_min_2);
                Ting(i,j)=arf;
                X(i,j)=xi;
                Y(i,j)=yi;
            elseif f==2
                [xi,yi,arf,f]=solve_point_2_1(p,X(i-1,j),Y(i-1,j),Ting(i-1,j),d,r1,x_C1,y_C1,theta_max_a);
                Ting(i,j)=arf;
                X(i,j)=xi;
                Y(i,j)=yi;
            else
                theta_xy=solve_theta1(p,X(i-1,j),Y(i-1,j),Ting(i-1,j),d);
                Ting(i,j)=theta_xy;
                X(i,j)=a*theta_xy*cos(theta_xy);
                Y(i,j)=a*theta_xy*sin(theta_xy);
            end
        end
    end
end

% 调头过程可视化
figure(3)
clf;
```

```
set(gcf,'Position',[200 200 600 600])
for j=1:size(X,2)
    plot(X(:,j),Y(:,j),'k-',Marker','*','MarkerSize',4,'MarkerFaceColor','cyan')
    title('100s 时板凳龙的轨迹')
    axis equal
    grid on
    xlabel('x 轴/m')
    ylabel('y 轴/m')
    axis([-25 25 -25 25])
    drawnow
end
set(gca, 'looseInset', [0 0 0 0]);

% 保存至相应位置的 excel 文件
xlswrite('X.xlsx',X)
xlswrite('Y.xlsx',Y)

function theta=solve_theta1(p,x,y,theta,d)
a=p/2/pi;
fun=@(theta)(a*theta.*cos(theta)-x).^2+(a*theta.*sin(theta)-y).^2-d^2;
q=0.01;
options = optimoptions('fsolve','Display','off');
theta=fsolve(fun,theta+q,options);
while theta<=theta || abs(a*theta-a*theta)>a/2
    q=q+0.1;
    theta=fsolve(fun,theta+q,options);
end
end

function theta=solve_theta2(p,x,y,theta,d)
a=p/2/pi;
fun=@(theta)(a*(theta+pi).*cos(theta)-x).^2+(a*(theta+pi).*sin(theta)-y).^2-d^2;
q=-0.1;
options = optimoptions('fsolve','Display','off');
theta=fsolve(fun,theta+q,options);
while theta>=theta || abs(a*theta-a*theta)>p/2
    q=q-0.1;
    theta=fsolve(fun,theta+q,options);
end
end
```

```
function [x,y,theta,flag]=solve_point_2_1(p,x,y,theta,d,r1,xc,yc,th
   eta_m)
a=p/2/pi;
delta_theta=2*asin(d/2/r1);
if delta_theta<=theta_m-theta
    flag=2;
    theta=theta+delta_theta;
    x=xc+r1*cos(theta);
    y=yc+r1*sin(theta);
else
    theta=solve_theta1(p,x,y,4.5/a,d);
    flag=1;
    x=a*theta*cos(theta);
    y=a*theta*sin(theta);
end
end

function [x,y,theta,flag]=solve_point_3_2(x,y,theta,d,r1,x1,y1,r2,x
   2,y2,theta_m)
delta_theta=2*asin(d/2/r2);
if delta_theta<=theta-theta_m
    flag=3;
    theta=theta-delta_theta;
    x=x2+r2*cos(theta);
    y=y2+r2*sin(theta);
else
    di=sqrt((x-x1)^2+(y-y1)^2);
    delta_theta=acos((di^2+r1^2-d^2)/2/di/r1);
    theta_C1_di=atan((y-y1)/(x-x1));
    theta=theta_C1_di+delta_theta;
    flag=2;
    x=x1+r1*cos(theta);
    y=y1+r1*sin(theta);
end
end

function [x,y,theta,flag]=solve_point_4_3(p,x,y,theta,d,r,xc,yc,the
   ta_m)
a=p/2/pi;
theta=solve_theta2(p,x,y,theta,d);
if theta>=4.5/a-pi
    flag=4;
    x=a*(theta+pi)*cos(theta);
    y=a*(theta+pi)*sin(theta);
```

```
else
    fun=@(t)(xc+r*cos(theta_m-t)-x).^2+(yc+r*sin(theta_m-t)-y).^2-d^2;
    q=0.1;
    options = optimoptions('fsolve','Display','off');
    delta_theta=fsolve(fun,theta+q,options);
    theta=theta_m-delta_theta;
    flag=3;
    x=xc+r*cos(theta);
    y=yc+r*sin(theta);
end
end
```

附录 7

第四问第二部分代码

```
# 运行此代码前，在X.xlsx 和Y.xlsx 中第一行前插一行，用(-100,100)，步长0.1 填充
import pandas as pd
import sympy as sm


# 计算两点距离
def length(x1, y1, x2, y2):
    return sm.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)


file_path_1 = "X.xlsx"
file_path_2 = "Y.xlsx"
file_path = "result4.xlsx"
df_1 = pd.read_excel(file_path_1)
df_2 = pd.read_excel(file_path_2)
df_3 = pd.read_excel(file_path, index_col=0, sheet_name="位置")
df_4 = pd.read_excel(file_path, index_col=0, sheet_name="速度")

print(df_1)
print(df_2)
print(df_3)
print(df_4)

for t in range(-100, 101):
    for i in range(0, 224):
        if i == 0:
```

```
            df_3.loc['龙头
   x (m)', str(t) + ' s'] = df_1.iat[i, 10 * t + 1000]
            df_3.loc['龙头
   y (m)', str(t) + ' s'] = df_2.iat[i, 10 * t + 1000]
            df_4.loc['龙头 (m/s)', str(t) + ' s'] = 1.000000
        elif i == 222:
            df_3.loc['龙尾
   x (m)', str(t) + ' s'] = df_1.iat[i, 10 * t + 1000]
            df_3.loc['龙尾
   y (m)', str(t) + ' s'] = df_2.iat[i, 10 * t + 1000]
            if t == -100:
                x_a = df_1.iat[i, 10 * t + 1000]
                y_a = df_2.iat[i, 10 * t + 1000]
                x_b = df_1.iat[i, 10 * t + 1001]
                y_b = df_2.iat[i, 10 * t + 1001]
                D1 = length(x_a, y_a, x_b, y_b)
                v = D1 / 0.1
                df_4.loc['龙尾  (m/s)', str(t) + ' s'] = v
            elif t == 100:
                x_a = df_1.iat[i, 10 * t + 1000]
                y_a = df_2.iat[i, 10 * t + 1000]
                x_b = df_1.iat[i, 10 * t + 999]
                y_b = df_2.iat[i, 10 * t + 999]
                D1 = length(x_a, y_a, x_b, y_b)
                v = D1 / 0.1
                df_4.loc['龙尾  (m/s)', str(t) + ' s'] = v
            else:
                x_a = df_1.iat[i, 10 * t + 1000]
                y_a = df_2.iat[i, 10 * t + 1000]
                x_b = df_1.iat[i, 10 * t + 999]
                y_b = df_2.iat[i, 10 * t + 999]
                x_c = df_1.iat[i, 10 * t + 1001]
                y_c = df_2.iat[i, 10 * t + 1001]
                D1 = length(x_a, y_a, x_b, y_b)
                D2 = length(x_a, y_a, x_c, y_c)
                v = (D2 + D1) / 0.2  # 某时刻后龙尾的速度
                df_4.loc['龙尾  (m/s)', str(t) + ' s'] = v
        elif i == 223:
            df_3.loc['龙尾（后）
   x (m)', str(t) + ' s'] = df_1.iat[i, 10 * t + 1000]
            df_3.loc['龙尾（后）
   y (m)', str(t) + ' s'] = df_2.iat[i, 10 * t + 1000]
            if t == -100:
                x_a = df_1.iat[i, 10 * t + 1000]
```

```
                y_a = df_2.iat[i, 10 * t + 1000]
                x_b = df_1.iat[i, 1]
                y_b = df_2.iat[i, 1]
                D1 = length(x_a, y_a, x_b, y_b)
                v = D1 / 0.1
                df_4.loc['龙尾（后） (m/s)', str(t) + ' s'] = v
            elif t == 100:
                x_a = df_1.iat[i, 10 * t + 1000]
                y_a = df_2.iat[i, 10 * t + 1000]
                x_b = df_1.iat[i, 10 * t + 999]
                y_b = df_2.iat[i, 10 * t + 999]
                D1 = length(x_a, y_a, x_b, y_b)
                v = D1 / 0.1
                df_4.loc['龙尾（后） (m/s)', str(t) + ' s'] = v
            else:
                x_a = df_1.iat[i, 10 * t + 1000]
                y_a = df_2.iat[i, 10 * t + 1000]
                x_b = df_1.iat[i, 10 * t + 999]
                y_b = df_2.iat[i, 10 * t + 999]
                x_c = df_1.iat[i, 10 * t + 1001]
                y_c = df_2.iat[i, 10 * t + 1001]
                D1 = length(x_a, y_a, x_b, y_b)
                D2 = length(x_a, y_a, x_c, y_c)
                v = (D2 + D1) / 0.2
                df_4.loc['龙尾（后） (m/s)', str(t) + ' s'] = v
        else:
            df_3.loc['第' + str(i) + '节龙身
   x (m)', str(t) + ' s'] = df_1.iat[i, 10 * t + 1000]
            df_3.loc['第' + str(i) + '节龙身
   y (m)', str(t) + ' s'] = df_2.iat[i, 10 * t + 1000]
            if t == -100:
                x_a = df_1.iat[i, 10 * t + 1000]
                y_a = df_2.iat[i, 10 * t + 1000]
                x_b = df_1.iat[i, 1]
                y_b = df_2.iat[i, 1]
                D1 = length(x_a, y_a, x_b, y_b)
                v = D1 / 0.1
                 df_4.loc['第' + str(i) + '节龙
   身 (m/s)', str(t) + ' s'] = v
             elif t == 100:
                x_a = df_1.iat[i, 10 * t + 1000]
                y_a = df_2.iat[i, 10 * t + 1000]
                x_b = df_1.iat[i, 10 * t + 999]
                y_b = df_2.iat[i, 10 * t + 999]
```

```
                D1 = length(x_a, y_a, x_b, y_b)
                v = D1 / 0.1
                df_4.loc['第' + str(i) + '节龙身  (m/s)', str(t) + ' s'] = v
            else:
                x_a = df_1.iat[i, 10 * t + 1000]
                y_a = df_2.iat[i, 10 * t + 1000]
                x_b = df_1.iat[i, 10 * t + 999]
                y_b = df_2.iat[i, 10 * t + 999]
                x_c = df_1.iat[i, 10 * t + 1001]
                y_c = df_2.iat[i, 10 * t + 1001]
                D1 = length(x_a, y_a, x_b, y_b)
                D2 = length(x_a, y_a, x_c, y_c)
                v = (D2 + D1) / 0.2
                df_4.loc['第' + str(i) + '节龙身  (m/s)', str(t) + ' s'] = v

# 写入文件
with pd.ExcelWriter(file_path, engine='xlsxwriter') as writer:
    df_3.to_excel(writer, sheet_name="位置", float_format="%.6f")
    df_4.to_excel(writer, sheet_name="速度", float_format="%.6f")
```

附录 8

第五问代码

```
p=1.7; % 螺距
a = p / (2 * pi); % 螺线系数
d1 = 2.86; % 龙头两把手间距
d2 = 1.65; % 除龙头板外其他板两把手间距
V_0=1:0.1:2; % 龙头运动速度
V_max=zeros(1,length(V_0));

for v=1:length(V_0)
    v0=V_0(v);
    R=4.5;
    theta_ru=R/a;
    theta_chu=R/a-pi;
    slope=(a*sin(theta_ru)+R*cos(theta_ru))/(a*cos(theta_ru)-R*sin(theta_ru));
    theta_max_a=atan(-1/slope)+pi;
    theta_d=atan(tan(theta_ru))+pi-theta_max_a;
    rC1_C2=R/cos(theta_d);
```

```
    rc2=rC1_C2/3;
    rc1=rc2*2;
    phi=2*theta_d;
    Sc1=rc1*(pi-phi);
    Sc2=rc2*(pi-phi);
    theta_min_a=theta_max_a-Sc1/rc1;
    theta_min_b=theta_min_a-pi;
    theta_max_b=theta_min_b+Sc2/rc2;
    % 得到两个圆圆心坐标
    x_C1=R*cos(theta_ru)+rc1*cos(theta_max_a-pi);
    y_C1=R*sin(theta_ru)+rc1*sin(theta_max_a-pi);
    x_C2=R*cos(theta_chu)-rc2*cos(theta_max_b);
    y_C2=R*sin(theta_chu)-rc2*sin(theta_max_b);
    dt=0.1;
    Ttotal=(Sc1+Sc2)/v0;
    T=0:dt:Ttotal;
    X=nan*zeros(224,length(T));
    Y=nan*zeros(224,length(T));
    T=nan*zeros(224,length(T));
    tt_c1=0:dt:Sc1/v0;
    theta_C1=-v0*tt_c1/rc1+theta_max_a;
    T(1,(1:length(tt_c1)))=theta_C1;
    X(1,(1:length(tt_c1)))=rc1*cos(theta_C1)+x_C1;
    Y(1,(1:length(tt_c1)))=rc1*sin(theta_C1)+y_C1;
    tt_c2=tt_c1(end)+dt:dt:(Sc1+Sc2)/v0;
    theta_C2=v0*(tt_c2-Sc1/v0)/rc2+theta_min_a-pi;
    T(1,length(theta_C1)+(1:length(tt_c2)))=theta_C2;
    X(1,length(theta_C1)+(1:length(tt_c2)))=rc2*cos(theta_C2)+x_C2;
    Y(1,length(theta_C1)+(1:length(tt_c2)))=rc2*sin(theta_C2)+y_C2;

    % 循环求解其余板凳位置
    N=223; % 板凳总个数
    ttotal=0:dt:(Sc1+Sc2)/v0;
    for jj=(1:length(ttotal))
        j=jj;
        if ttotal(jj)<0
            for i=2:N+1
                d=d1*(i<=2)+d2*(i>2);
                thetaij=solve_theta1(p,X(i-1,j),Y(i-1,j),T(i-
  1,j),d);
                T(i,j)=thetaij;
                X(i,j)=a*thetaij*cos(thetaij);
                Y(i,j)=a*thetaij*sin(thetaij);
            end
```

```
        elseif ttotal(jj)>=0 && ttotal(jj)<=Sc1/v0
            flag=2;
            for i=2:N+1
                d=d1*(i<=2)+d2*(i>2);
                if flag==2
                    [xi,yi,thetai,flag]=solve_point_2_1(p,X(i-1,j),Y(i-1,j),T(i-1,j),d,rc1,x_C1,y_C1,theta_max_a);
                    T(i,j)=thetai;
                    X(i,j)=xi;
                    Y(i,j)=yi;
                else
                    thetaij=solve_theta1(p,X(i-1,j),Y(i-1,j),T(i-1,j),d);
                    T(i,j)=thetaij;
                    X(i,j)=a*thetaij*cos(thetaij);
                    Y(i,j)=a*thetaij*sin(thetaij);
                end
            end
        elseif ttotal(jj)>Sc1/v0 && ttotal(jj)<=Sc1/v0+Sc2/v0
            flag=3;
            for i=2:N+1
                d=d1*(i<=2)+d2*(i>2);
                if flag==3
                    [xi,yi,thetai,flag]=solve_point_3_2(X(i-1,j),Y(i-1,j),T(i-1,j),d,rc1,x_C1,y_C1,rc2,x_C2,y_C2,theta_min_b);
                    T(i,j)=thetai;
                    X(i,j)=xi;
                    Y(i,j)=yi;
                elseif flag==2
                    [xi,yi,thetai,flag]=solve_point_2_1(p,X(i-1,j),Y(i-1,j),T(i-1,j),d,rc1,x_C1,y_C1,theta_max_a);
                    T(i,j)=thetai;
                    X(i,j)=xi;
                    Y(i,j)=yi;
                else
                    thetaij=solve_theta1(p,X(i-1,j),Y(i-1,j),T(i-1,j),d);
                    T(i,j)=thetaij;
                    X(i,j)=a*thetaij*cos(thetaij);
                    Y(i,j)=a*thetaij*sin(thetaij);
                end
            end
        else
            flag=4;
```

```
            for i=2:N+1
                d=d1*(i<=2)+d2*(i>2);
                if flag==4
                    [xi,yi,thetai,flag]=solve_point_4_3(p,X(i-1,j),Y(i-1,j),T(i-1,j),d,rc2,x_C2,y_C2,theta_max_b);
                    T(i,j)=thetai;
                    X(i,j)=xi;
                    Y(i,j)=yi;
                elseif flag==3
                    [xi,yi,thetai,flag]=solve_point_3_2(X(i-1,j),Y(i-1,j),T(i-1,j),d,rc1,x_C1,y_C1,rc2,x_C2,y_C2,theta_min_b);
                    T(i,j)=thetai;
                    X(i,j)=xi;
                    Y(i,j)=yi;
                elseif flag==2
                    [xi,yi,thetai,flag]=solve_point_2_1(p,X(i-1,j),Y(i-1,j),T(i-1,j),d,rc1,x_C1,y_C1,theta_max_a);
                    T(i,j)=thetai;
                    X(i,j)=xi;
                    Y(i,j)=yi;
                else
                    thetaij=solve_theta1(p,X(i-1,j),Y(i-1,j),T(i-1,j),d);
                    T(i,j)=thetaij;
                    X(i,j)=a*thetaij*cos(thetaij);
                    Y(i,j)=a*thetaij*sin(thetaij);
                end
            end
        end
        waitbar((length(ttotal)*(v-1)+jj)/(length(ttotal)*length(V_0)),hwait,'已经完成...')
    end

    Vx=zeros(size(X));
    Vy=Vx;
    Vx(:,1)=(X(:,2)-X(:,1))/dt;
    Vx(:,end)=(X(:,end)-X(:,end-1))/dt;
    Vx(:,2:end-1)=(X(:,3:end)-X(:,1:end-2))/2/dt;
    Vy(:,1)=(Y(:,2)-Y(:,1))/dt;
    Vy(:,end)=(Y(:,end)-Y(:,end-1))/dt;
    Vy(:,2:end-1)=(Y(:,3:end)-Y(:,1:end-2))/2/dt;
    V=sqrt(Vx.^2+Vy.^2);
    V_max(v)=max(max(V));

```

```
end

% 下面可视化
p=polyfit(V_0,V_max,1); % 线性拟合得到 Vmax 与 v0 的解析关系
figure
p1=plot(V_0,V_max,'ro','LineWidth',1);
xlabel('龙头速度')
ylabel('队伍把手最高速度')
hold on
color=rand(1,3);
p2=plot(linspace(V_0(1),V_0(end),100),polyval(p,linspace(V_0(1),V_0(end),100)),'Color',color);
plot([V_0(1) V_0(end)],[2 2],'k--')
legend('队伍最大速度与龙头速度数据点','线性拟合关系')
v0max=(2-p(2))/p(1);
disp(['龙头最大行进速度为:',num2str(v0max)])

function theta=solve_theta1(p,x,y,theta,d)
a=p/2/pi;
fun=@(theta)(a*theta.*cos(theta)-x).^2+(a*theta.*sin(theta)-y).^2-d^2;
q=0.01;
options = optimoptions('fsolve','Display','off');
theta=fsolve(fun,theta+q,options);
while theta<=theta || abs(a*theta-a*theta)>a/2
    q=q+0.1;
    theta=fsolve(fun,theta+q,options);
end
end

function theta=solve_theta2(p,x,y,theta,d)
a=p/2/pi;
fun=@(theta)(a*(theta+pi).*cos(theta)-x).^2+(a*(theta+pi).*sin(theta)-y).^2-d^2;
q=-0.1;
options = optimoptions('fsolve','Display','off');
theta=fsolve(fun,theta+q,options);
while theta>=theta || abs(a*theta-a*theta)>p/2
    q=q-0.1;
    theta=fsolve(fun,theta+q,options);
end
end

```

```
function [x,y,theta,flag]=solve_point_2_1(p,x1,y1,theta,d,rC1,x,y,t
   heta_m)
a=p/2/pi;
delta_theta=2*asin(d/2/rC1);
if delta_theta<=theta_m-theta
    flag=2;
    theta=theta+delta_theta;
    x=x+rC1*cos(theta);
    y=y+rC1*sin(theta);
else
    theta=solve_theta1(p,x1,y1,4.5/a,d);
    flag=1;
    x=a*theta*cos(theta);
    y=a*theta*sin(theta);
end
end

function [x,y,theta,flag]=solve_point_3_2(x,y1,theta1,d,rC1,x_c1,y_
   c1,rC2,x2,y2,theta_m)
delta_theta=2*asin(d/2/rC2);
if delta_theta<=theta1-theta_m
    flag=3;
    theta=theta1-delta_theta;
    x=x2+rC2*cos(theta);
    y=y2+rC2*sin(theta);
else
    di=sqrt((x-x_c1)^2+(y1-y_c1)^2);
    delta_theta=acos((di^2+rC1^2-d^2)/2/di/rC1);
    theta_C1_di=atan((y1-y_c1)/(x-x_c1));
    theta=theta_C1_di+delta_theta;
    flag=2;
    x=x_c1+rC1*cos(theta);
    y=y_c1+rC1*sin(theta);
end
end

function [x,y,theta,flag]=solve_point_4_3(p,x,y,theta,d,rc2,xc,yc,t
   heta_m)
a=p/2/pi;
theta=solve_theta2(p,x,y,theta,d);
if theta>=4.5/a-pi
    x=a*(theta+pi)*cos(theta);
    y=a*(theta+pi)*sin(theta);
else
```

```
219.     fun=@(t)(xc+rc2*cos(theta_m-t)-x).^2+(yc+rc2*sin(theta_m-t)-
   y).^2-d^2;
220.     q=-0.1;
221.     options = optimoptions('fsolve','Display','off');
222.     delta_theta=fsolve(fun,theta_m+q,options);
223.     theta=theta_m-delta_theta;
224.     flag=3;
225.     x=xc+rc2*cos(theta);
226.     y=yc+rc2*sin(theta);
227. end
228. end
```