平面等距螺线轨道上“板凳龙”运动路线设计与分析

摘要

本文针对“板凳龙”在阿基米德螺线上的运动问题，结合弧长积分、隐函数求解、碰撞检测等方法，建立了多个数学模型，分析了龙的运动状态和路径优化等问题。

针对问题一：首先，根据题意确定盘入螺线方程及进入点坐标。由**弧长积分公式**，结合龙头的恒定行进速度，求出任意时刻龙头的位置。接着，采用**迭代**方法建立龙身运动状态模型。通过极坐标与笛卡尔坐标的转换关系及板凳长度限制，推导出相邻把手的坐标递推隐函数式。同时，基于同一板凳两孔洞的速度在沿板凳方向上分量相同这一**物理约束**，推导出相邻把手的速度迭代公式。编写程序，采用**牛顿法**求解隐函数，计算相邻把手的坐标及速度，最终得到前 $300s$ “板凳龙”各把手的坐标位置和速度状态，分别见表 1、表 2，并对结果进行了理论与实际分析。

针对问题二：首先，根据板凳的运动学特点，结合曲线的**几何特性**，证明首次碰撞必为龙头板凳的顶点与外侧板凳相撞。然后结合螺线特点，建立**碰撞检测机制**，即判断龙头板凳靠外侧的两个顶角是否落入它们分别对应的两个外圈龙身板凳内。该机制有效降低**算法复杂度**。接下来，通过**变步长迭代法**得到**首次碰撞**发生在 **412.48 秒**，并利用问题一建立的模型，计算出此时舞龙队的运动状态，见表 3。

针对问题三：首先，基于问题二中建立的碰撞检测机制，推导了螺距与首次碰撞发生时龙头位置之间的关系。考虑到一定范围内螺纹间距与首次发生碰撞的半径并**不成单调关系**，本文通过设置螺纹间距步长为 $0.01m$ 的来绘制出二者关系曲线，并在各自**单调区间内**利用**二分法**求得**最小螺距为 0.451m。**

针对问题四：首先，证明在 $1.7m$ 的螺距条件下，板凳龙行进过程中**不会发生碰撞**。然后，根据几何对称性及圆弧相切条件，构建由两段圆弧组成的调头路径。为优化路径长度，找出保证龙头板凳能够顺利进出调头区域的**几何约束**，建立**单目标优化模型**。采用**模拟退火**算法，得到实际**最小调头区域半径为 4.28m，最短调头路径为 12.93m。**接下来，将各把手可能出现的位置分为盘入螺线、圆弧 1、圆弧 2 和盘出螺线 4 部分，**分段、分曲线讨论**，迭代求解出“板凳龙”相应时刻的位置信息，见表 4，再用**速度约束分阶段迭代**出速度信息，见表 5。

针对问题五：首先，基于两把手沿板凳方向速度余弦分量相同这一物理约束，确定任意特定形态下“板凳龙”各把手速度**成比例**。接下来，在问题四中“板凳龙”任意时刻速度信息结果中找寻最大速度的把手所在空间位置，将该位置龙把手速度调至 $2m/s$。同时，利用**速度比例关系**，得到龙头前把手的相应速度。最终计算出龙头前把手**最大且恒定的速度为 1.02m/s。**

关键词：螺线；隐函数求解；碰撞检测；局部迭代

一、 问题重述

“板凳龙”，又名“盘龙”，是中国浙闽地区的一项传统民俗活动，通常在元宵节期间举行。这项活动将多条板凳首尾相接，组成蜿蜒如龙的长队；在舞龙的过程中，龙头引领着龙身与龙尾环绕行进，形成螺旋状的盘旋图案，展现了独特的文化特色。

“板凳龙”结构：由龙头凳、龙身凳和龙尾凳组成。各凳宽均为 30 cm，每个板凳距两板头 27.5 cm 处均有一直径为 5.5 cm 的孔洞，用于插入把手，使凳子连接成龙。除了长度为 341 cm 的龙头凳之外，其他的凳子 (即龙身、龙尾) 长度均为 220 cm。

问题一：舞龙队沿螺距为 55 cm 的等距螺线从第 16 圈的 A 点盘入，龙头前把手保持 1 m/s 的速度不变。计算前 300 秒内每秒钟各把手（包括龙头、龙身和龙尾）的中心位置和速度，保留六位小数。

问题二：舞龙队按照问题一的方式盘入，在不发生碰撞的前提下，计算舞龙队可持续盘入的最长时间，并计算终止时刻舞龙队的位置与速度。

问题三：舞龙队沿某一螺距恒定的螺线盘入，到达一个圆心位于螺线中心、半径为 4.5 m 的圆形调头空间。计算使龙头在到达调头空间前不发生碰撞的最小螺距。

问题四：舞龙队须在问题三规定的圆形调头空间内调头，调头曲线为两段相互相切的圆弧拼接成的 S 型曲线，两端圆弧半径比为 2:1，且分别与盘入螺线、盘出螺线相切。在保证盘入螺线、盘出螺线关于螺线原点中心对称的前提下，实际的调头区域半径可小于 4.5 m。在满足上述条件的基础上优化调头路线，缩短调头曲线长度。当龙头前把手以 1 m/s 的速度匀速通过调头曲线时，记调头开始时刻为零时刻，给出-100 s 至 100 s 内每秒钟舞龙队的位置和速度。

问题五：舞龙队沿着问题四所设路线行进，龙头的速度保持不变。在确保舞龙队中每一把手的行进速度都不超过 2 m/s 的情况下，求出龙头的最大行进速度。

二、 问题分析

2.1 问题一的分析

首先，根据题中所给的参数可以确定阿基米德螺线方程以及“板凳龙”开始进入的坐标位置 A 点，基于龙头前把手的速度可以计算出经过任意 t 时刻龙头前把手经过的弧长，结合螺线弧长积分公式可以解出 t 时刻龙头前把手的坐标。考虑相邻两个把手之间欧氏距离已知，且均在螺线上，可借助求解隐函数零点的方式一一迭代出龙头前把手后每一个把手的坐标信息。

对于求解每一个把手位置的速度，可以通过每条板凳的两个把手沿板凳方向速度的余弦分量相同这一物理约束进行迭代求解，方向夹角可由板凳方向与把手所在螺线上切向（即速度方向）确定。

2.2 问题二的分析

考虑问题一中“板凳龙”在盘入螺线时，相邻两个把手的坐标需要向外扩展成一个矩形，这就导致在给定螺纹间距下，内圈板凳与外圈板凳可能发生碰撞，抽象成数学模型即为内圈矩形与外圈矩形发生重叠。本文首先判定出“板凳龙”的首次碰撞必定来自于龙头板凳的矩形顶点与外侧板凳相撞，随后进行编程模拟遍历求解，最后再次利用问题一中的方法求解此时各板凳把手的坐标位置与速度。

2.3 问题三的分析

本问要求当“板凳龙”的龙头板凳首次相撞时，龙头前把手的位置需在半径为4.5m的圆内，考虑到取问题二中螺纹间距参数时，首次相撞时龙头前把手位置已在圆内，为此，可以逐步下调螺纹间距，直至相撞时龙头前把手刚好落在题中所给圆周上，此时的螺纹间距即为最小螺纹间距。

2.4 问题四的分析

首先，考虑到螺纹间距为1.7m，可以判定舞龙队在掉头出来的部分不会与进去的部分、以及自身内外不会相撞。其次，基于中心对称以及两段圆弧与盘入、盘出螺线三处相切这一条件，可确定问题三规定调头空间下的调头路径。但实际若要使得调头路径最短，需要同比缩小调头曲线两圆弧半径，同时需要保证在该路径上龙头板凳能够通过，即不发生因路径曲率问题导致该板凳无法继续前行，最终确定实际掉头区域半径，并计算出最短调头曲线长度。

在确定龙头前把手的速度恒定为1m/s后，将龙头前把手进入调头区域的时刻记为0时刻，考虑到-100s至0时刻“板凳龙”还未进入调头区域，可借由问题一的方法计算出每秒钟舞龙队的位置与速度。对于0时刻至100s时刻，“板凳龙”的各个部分会出现在盘入螺线、两段调头圆弧曲线以及盘出螺线四部分，在进行求解时需确定任一时刻龙头前把手的位置，再针对特定曲线方程进行分类讨论，即对“板凳龙”进行分段迭代求解。

2.5 问题五的分析

基于问题四中所得到的任意时刻“板凳龙”每个把手的速度分布，对此可以理解为当中“板凳龙”的任意确定形态的每一个把手的速度已知，考虑到同一物理构型，相邻把手的速度需满足余弦分量相同，即对任意确定形态的“板凳龙”，各把手的速度各自呈一确定比例，为求满足题意的龙头速度，只需将问题四中得到的最大速度值调节为2m/s，在将该时刻的龙头速度同比进行缩放，即可求得所求最大行进速度。

三、 模型假设

1. 假设各把手中心均位于螺线上。
2. 假设各板凳不发生形变。
3. 假设整条板凳龙各节板凳均在同一平面内运动。
4. 假设把手从孔洞垂直插入，且在运动过程中不会松动；把手与板凳间无相对运动。

四、 符号说明

符号	说明
p	等距螺线的螺距
b	阿基米德螺旋线系数[2]
D_1	龙头板凳两孔洞中心的距离
D_2	龙身和龙尾板凳两孔洞中心的距离
W	板凳宽度
d	孔洞与最近板头的距离
(x, y)	笛卡尔坐标系下坐标
(r, θ)	极坐标系下坐标

五、 模型的建立与求解

5.1 问题一模型的建立与求解

问题一需解出 300 秒内每秒钟龙的运动状态。龙头保持 1 m/s 恒定速度前进，各节通过孔洞依次相连，跟随龙头，沿等距螺线相继盘入。因此首先应确立龙头的运动状态。

(i) 龙头运动模型的建立

根据题目，板凳龙盘入曲线为一阿基米德螺线，可由方程 (1)描述[1]

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

其中，r 是螺线的极径，θ 为极角，b 为阿基米德螺旋线系数，其值为 $b = \frac{p}{2\pi}$, p 为螺距，取 $0.55m$。首先确定龙头在任意时刻下的位置。借助阿基米德螺线弧长计算公式，螺旋线的弧长 $s(\theta)$ 可以通过以下积分公式求得，

$$s(\theta) = \int_{\theta_0}^{\theta} \sqrt{[r(\theta)]^2 + [r'(\theta)]^2} \cdot d\theta \tag{2}$$

即，

$$s(\theta)=\frac{b}{2}\left[\theta\sqrt{\theta^2+1}+\sinh^{-1}(\theta)\right]\Big|_{\theta_0}^{\theta} \tag{3}$$

结合龙头的恒定前进速度 v_{head} = 1 m/s，可知在任意时刻 t，龙头走过的螺线长度为，

$$s(\theta)=S(\theta)-S\left(\theta_0\right)=-v_{\text{head}}\cdot t \tag{4}$$

其中 θ_0 即为 A 点极角，为 $32\pi rad$。基于式 (4)可求解出任意 t 时刻下龙头前把手的极角 θ，代入式 (1)，可以最终解出龙头的笛卡尔坐标系坐标位置。

(ii) 迭代法求龙身运动状态

对于任意相邻的把手，其极坐标与笛卡尔坐标的转换关系分别如下，

$$\begin{cases} x_n=b\theta_n\cos\theta_n \\ y_n=b\theta_n\sin\theta_n \end{cases} \tag{5}$$

$$\begin{cases} x_{n+1}=b\theta_{n+1}\cos\theta_{n+1} \\ y_{n+1}=b\theta_{n+1}\sin\theta_{n+1} \end{cases} \tag{6}$$

板凳自身长度决定的限制可由式 (7)描述，其中 D_1，D_2 分别是龙头板凳、龙身和龙尾板凳两孔洞中心的距离，

$$\left(x_n-x_{n+1}\right)^2+\left(y_n-y_{n+1}\right)^2=D_{1,2}^2 \tag{7}$$

将极坐标代入该限制条件，可得到如下隐函数关系式，

$$f\left(\theta_n,\theta_{n+1}\right)=\theta_{n+1}^2+\theta_n^2-2\theta_n\theta_{n+1}\left(\cos\theta_n\cos\theta_{n+1}+\sin\theta_n\sin\theta_{n+1}\right)=0 \tag{8}$$

该隐函数可得到相邻把手的极角坐标递推关系。结合龙头前把手的坐标，便能推算出龙身和龙尾各把手的坐标。

对于速度的求解，由于板凳长度不发生改变，需满足同一板凳中两把手的速度在沿板凳方向的余弦分量相同这一物理约束，即，

$$v_{n+1}\cos\phi_{n+1}=v_n\cos\phi_n \tag{9}$$

其中，板凳方向的向量 $\vec{L}$ 表示为

$$\vec{L}=\left(x_{n+1}-x_n,y_{n+1}-y_n\right) \tag{10}$$

即，

$$\vec{L}=\left(b\theta_{n+1}\cos\theta_{n+1}-b\theta_n\cos\theta_n,b\theta_{n+1}\sin\theta_{n+1}-b\theta_n\sin\theta_n\right) \tag{11}$$

板凳上两把手速度方向分别如下，

$$\vec{T_n} = \left(1, \left.\frac{dy}{dx}\right|_{\theta_n}\right) = \left(1, \frac{\sin\theta_n + \theta_n\cos\theta_n}{\cos\theta_n - \theta_n\sin\theta_n}\right) \tag{12}$$

$$\overrightarrow{T_{n+1}} = \left(1, \left.\frac{dy}{dx}\right|_{\theta_{n+1}}\right) = \left(1, \frac{\sin\theta_{n+1} + \theta_{n+1}\cos\theta_{n+1}}{\cos\theta_{n+1} - \theta_{n+1}\sin\theta_{n+1}}\right) \tag{13}$$

代入向量夹角公式，

$$\cos\phi = \frac{\vec{T}\cdot\vec{L}}{|\vec{T}||\vec{L}|} \tag{14}$$

可得 $\cos\phi_n$ 以及 $\cos\phi_(n+1)$ 的值，最终得到相邻两把手的速度迭代关系，

$$v_{n+1} = \frac{v_n\cos\phi_n}{\cos\phi_{n+1}} = k\cdot v_n \tag{15}$$

基于龙头前把手的速度，可求解出任意时刻任意把手的速度。

(iii) “板凳龙”前 300 秒运动状态的分析与求解

编写 Python 程序，通过迭代求解出前 300 秒内每秒钟板凳龙的坐标和速度，见 result1.xlsx；部分时刻结果展示如表 1、表 2。[1]

表 1　问题一龙的各节点在不同时间点的坐标

	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)	-9.51873	-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)	2.913983	5.687116	5.361939	1.898794	-4.917371	-6.237722
第 101 节龙身 y (m)	-9.91831	-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)	1.828753	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	7.383896	3.241051	1.785033
龙尾（后）y (m)	-10.676584	-8.797992	0.843473	7.492370	9.469336	9.301164

[1]注：前 300 秒内“板凳龙”未全部进入盘入区域。为数据展示方便，本文假定剩余部分按螺线方程排列，实际解不唯一。

表 2　问题一不同时间点龙的各把手节点速度

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 (m/s)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身 (m/s)	0.999971	0.999961	0.999945	0.999917	0.999859	0.999709
第 51 节龙身 (m/s)	0.999742	0.999662	0.999538	0.999331	0.998941	0.998065
第 101 节龙身 (m/s)	0.999575	0.999453	0.999269	0.998971	0.998435	0.997302
第 151 节龙身 (m/s)	0.999448	0.999299	0.999078	0.998727	0.998115	0.996861
第 201 节龙身 (m/s)	0.999348	0.999180	0.998935	0.998551	0.997894	0.996574
龙尾（后）(m/s)	0.999311	0.999136	0.998883	0.998489	0.997816	0.996478

根据求解得到的位置结果，编写程序可视化 300 秒时刻“板凳龙”的盘入结构，如图5.1.1所示；在 300s 内以 0s 为起始点，60s 为步进长度取时间点，绘制出表2对应时刻下，“板凳龙”各把手的速度分布，如图5.1.2所示。

300s 时刻，“板凳龙”未全部盘入，为便于可视化，假定未盘入部分也沿式 (1)描述的螺线排布。实际情况下为盘入部分坐标不唯一。

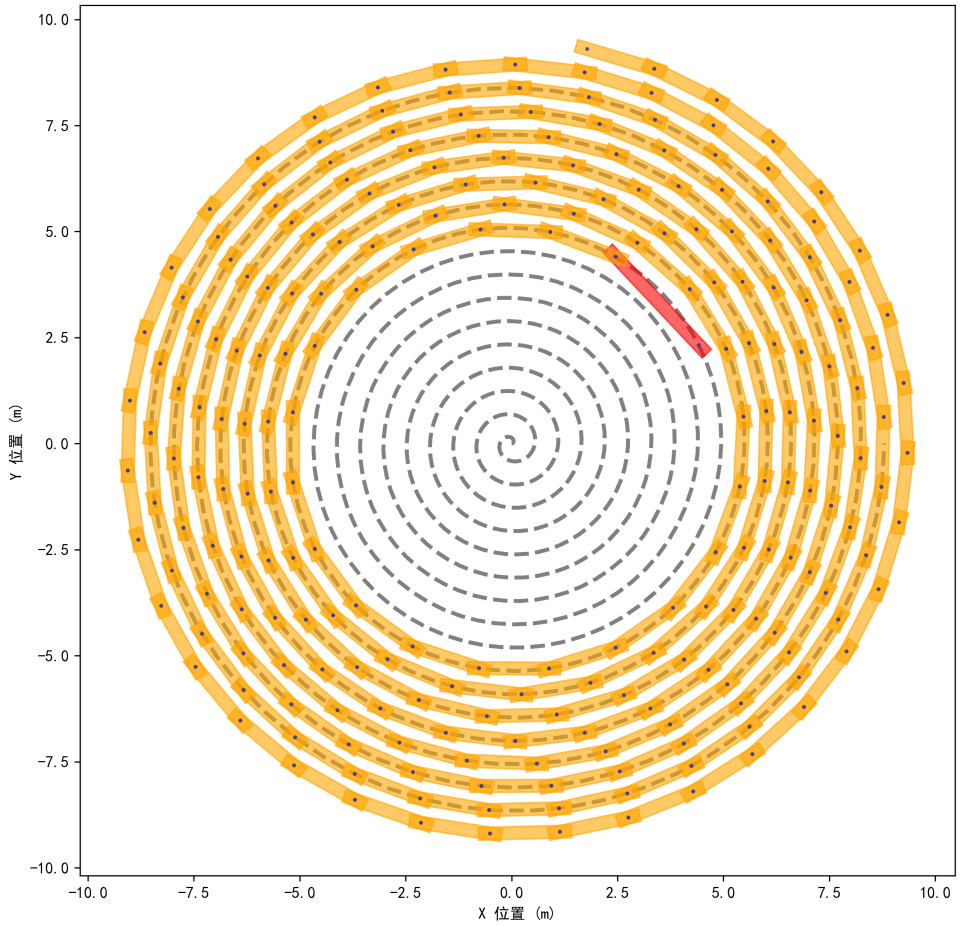


图 5.1.1　300 秒时刻龙的结构

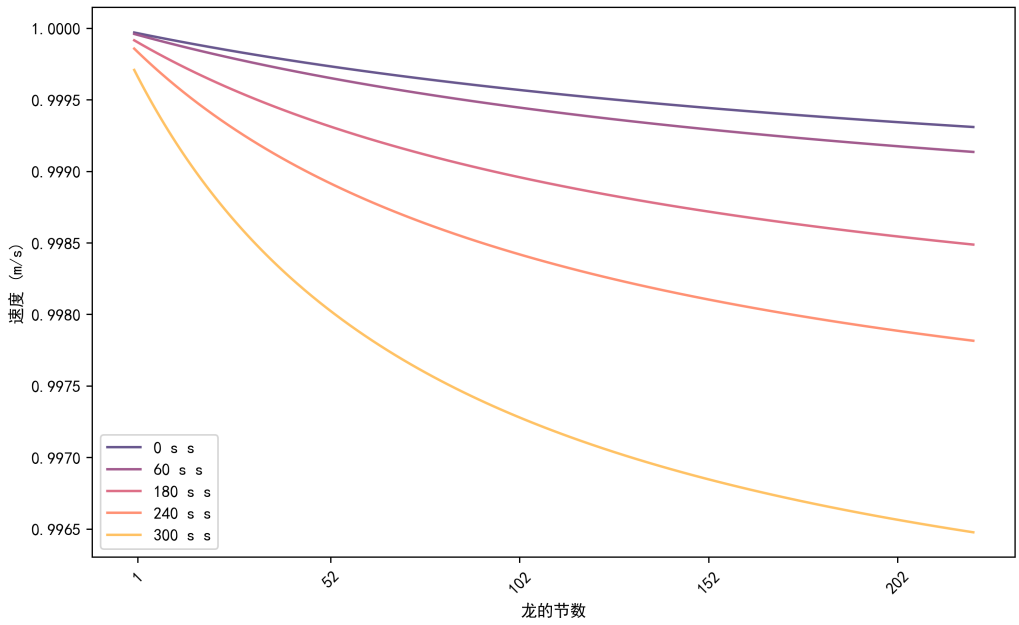


图 5.1.2　不同时间点、不同节的速度对比

从图5.1.2可以看出，同一时刻，龙的节数越靠后，速度越小。这是因为节数越靠后，对应的曲率半径越大，弦切角越小, 导致式 (15)中比例系数 k 值越小，所以越外侧的把手速度越小。对比 5 条曲线可知，随着时间的推移，各节“板凳龙”把手的速度差异越来越大，因为“板凳龙”向螺线内部行进时，螺线曲率半径变小，弦切角变大，结合余弦函数特性可知，相邻把手之间的 k 值变化率会逐渐增大。仿真结果与实际相符。

5.2 问题二模型的建立

问题二要解出舞龙队盘入的终止时刻，即需判断板凳间首次发生碰撞的时刻。首先，应当分析板凳龙的运动结构，建立合适的碰撞检测模型。

图5.2.1展示的是某单一板凳的左侧部分结构。凳宽 30cm，在距离两板头 27.5 cm 处各有一孔洞；孔洞始终沿螺线行进。板凳行进过程中，凳子经过空间的包络线由顶角确定，故顶角划定了矩形行进时占用的最大空间。

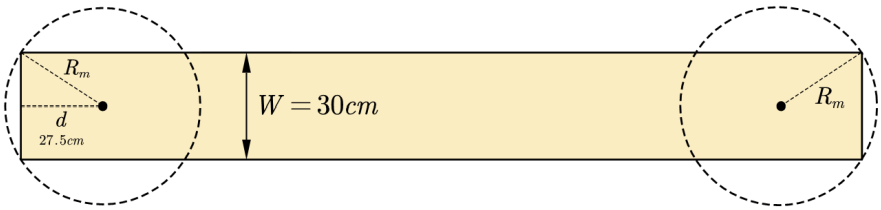


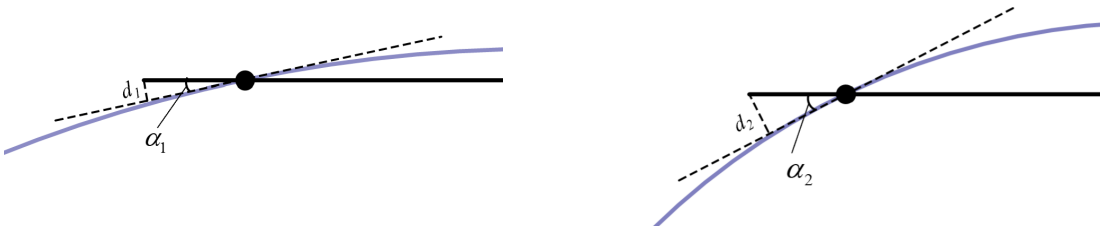
图 5.2.1　单节“板凳龙结构分析”

证明：首次碰撞必为龙头板凳的顶点与外侧板凳相撞

首先从简化的数学模型进行分析。考虑曲线曲率恒定，即为圆弧时的情况。将板凳简化为一条线段，孔洞为线段上一点，记板凳孔洞到线段较近端点的距离为 d，由题意，各节板凳 d 为定值。如图5.2.2所示，左图圆弧曲率半径更大，曲率较小。分别过交点作圆弧的切线，记切线与线段形成的锐角为 $\alpha_i\ (i=1,2)$，再分别过线段端点作切线的垂线，垂线段记为 d_1、d_2。显然，$\alpha_1 < \alpha_2$，根据几何关系，

$$d_i = \zeta \cdot \alpha_i \tag{16}$$

则 d_1 大于 d_2 。



(a) 较小曲率示意图　　(b) 较大曲率示意图

图 5.2.2　曲率对碰撞的影响

接下来考虑相同曲率下，割线长短对碰撞的影响。如图5.2.3所示，曲率相同的圆弧上有两条割线，长割线对应的 α_3 大于短割线对应的 α_4，由式 (16)可知，$d_3 > d_4$ 。

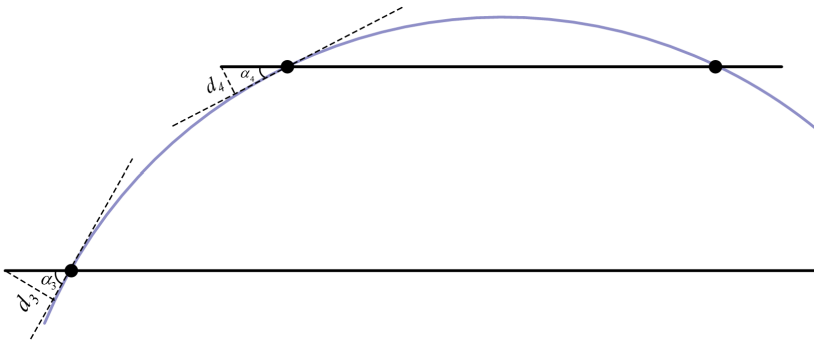


图 5.2.3　割线长度对碰撞的影响

在舞龙队盘入时，龙头位于最前方，龙头前把手所在螺线曲率最小；同时龙头凳长度大于其它板凳，产生的割线长度最大。综上，**龙头转向产生的 d_i 最大，最有可能发生碰撞**。

5.3 问题二模型的求解

基于上文提到的思路，可以判定出首次碰撞必然是龙头板凳外侧的两个矩形顶点之一与其外侧的矩形相撞，为了进一步简化，提高算法效率，本文模拟“板凳龙”行进时设定了如下的碰撞检测机制：

STEP1：获取龙头板凳的两个把手所在极角坐标 θ_1 与 θ_2

STEP2：找寻外圈中相邻把手的极角 θ_{11}，θ_{12}，θ_{21}，θ_{22} 满足

$$\begin{cases} \theta_{11} < \theta_1 + 2\pi < \theta_{12} \\ \theta_{21} < \theta_2 + 2\pi < \theta_{22} \end{cases} \tag{17}$$

式 (17)中，θ_{11} 与 θ_{12}，θ_{21} 与 θ_{22} 分别确定了外侧两块矩形。如图5.3.1

STEP3：判断龙头矩形的两个外侧顶点是否落入对应龙身矩形内。

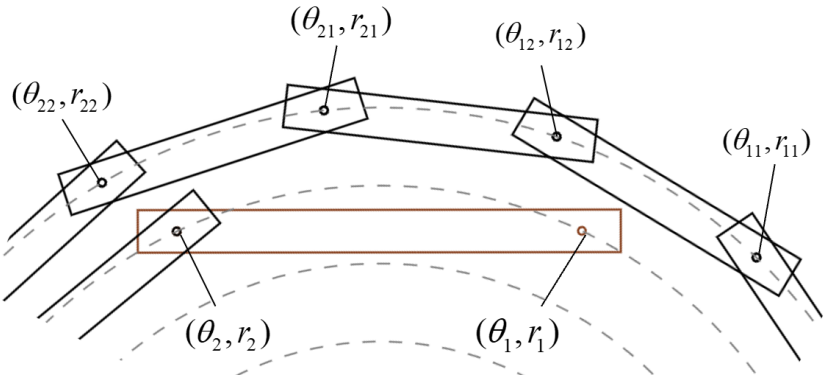


图 5.3.1　碰撞机制判定示意图

最终简化了大量计算，减少遍历算法的复杂度。

通过编写 Python 程序，在每次步进迭代的同时加入上述的条件判断，最终确定出“板凳龙”首次出现相撞的时刻在 412s 与 413s 之间，在该范围内进一步调小时间步长，最终确定首次出现相撞的时刻为

$$t = 412.48s \tag{18}$$

同时，获得该时刻下“板凳龙”的相关参数如下表所示，

表 3　终止时刻龙的各节点的坐标

	横坐标 x (m)	纵坐标 y (m)	速度 m/s
龙头	1.607860	1.600338	1.000000
第 1 节龙身	-1.216456	2.050716	0.991188
第 51 节龙身	1.761036	4.143629	0.976187
第 101 节龙身	-1.042985	-5.803967	0.973864
第 151 节龙身	0.458904	-7.003208	0.972919
第 201 节龙身	-7.950194	-0.722213	0.972405
龙尾（后）	1.461716	8.243548	0.972247

相关物理构型细节如下图所示，

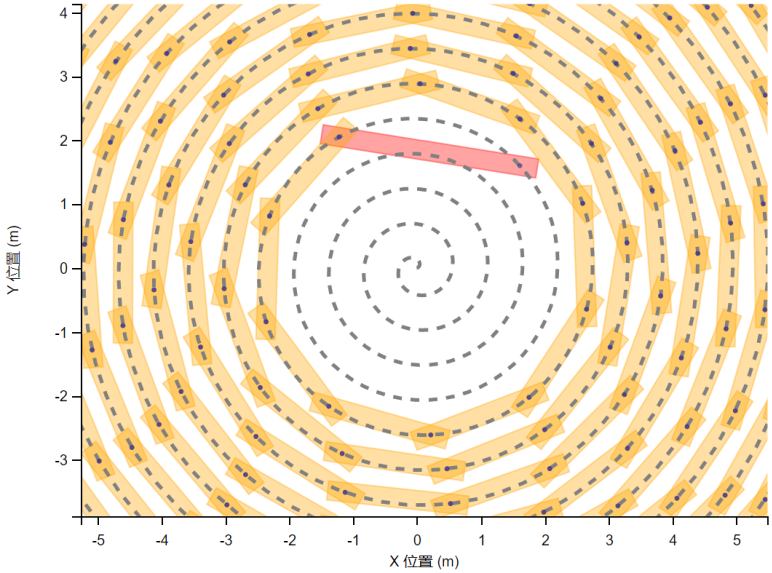
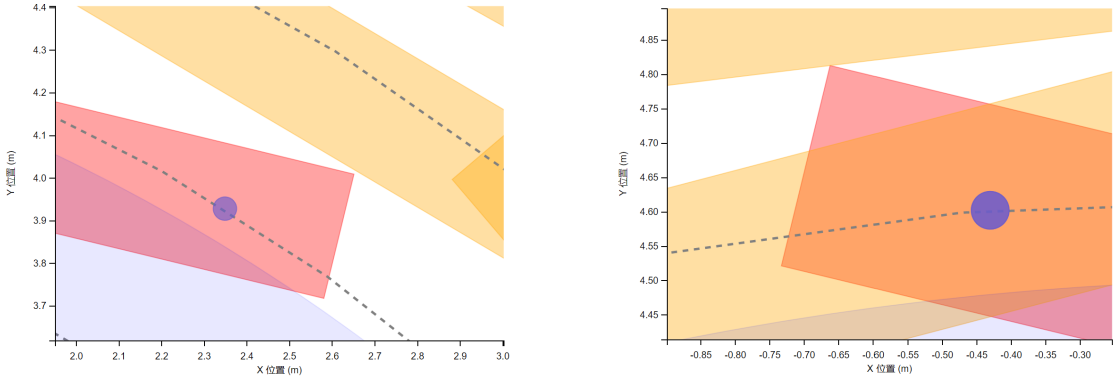


图 5.3.2　终止时刻“板凳龙”的结构



(a) 龙头凳前端点　　(b) 龙头凳后端点

图 5.3.3　问题二碰撞细节图

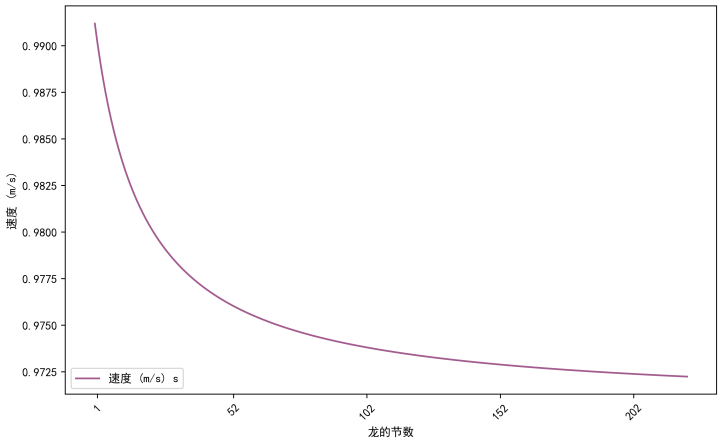


图 5.3.4　终止时刻“板凳龙”的速度分布

5.4 问题三模型的建立与求解

问题三需要调整螺距 p, 使得龙头前把手能够沿着相应的螺线盘入到调头空间的边界，即在到达调头空间前不发生碰撞。基于问题二计算碰撞时龙头前把手坐标的算法，本文抽象出龙头前把手与坐标原点距离 $f(p)$ 关于螺距 p 的关系函数，即，

$$f(p) = r_{crash} = \sqrt{x_{crash,head}^2 + y_{crash,head}^2} \tag{19}$$

由于函数 $f(p)$ 过于复杂，难以进行求导运算，本文并未计算出显式的导数，而是使用了数值法，即通过计算函数值在当前点及其附近点的差值来估计导数值。数值法的运算公式为

$$g(p) = f'(p) \approx \frac{f(p+\epsilon) - f(p)}{\epsilon} \tag{20}$$

其中，ϵ 在本文取 10^{-3}。

基于式 (20)，本文遍历了 $g(p)$ 的值，得到其值为先正后负，即 $f(p)$ 关于 p 先单调递增再单调递减。由于板凳宽 W 为 $0.3m$，实际盘入过程中必须保证相互之间留有空隙，故将 $p = 0.3m$ 设置为左边界。根据以上条件，绘出 $f(p)$ 关于 p 的变化曲线，如图5.4.1。可以得到在 $p = 0.3m$ 时，$f(p) = 4.8m$，递增到 $p = 0.41m$，$f(p) = 6.49m$，再继续递减。根据以上结果，本文在递减区间使用了二分法，以寻找最小螺距 p。在使用二分法时，将左边界设置为 $p = 0.41m$，同时，本文计算出在 $p = 0.55m$ 时，龙头前把手距离坐标原点的距离约为 $2.26m$，小于问题三所要求的调头半径 $4.5m$，故将 $p = 0.55m$ 设置为右边界。

基于以上算法，最终得到 p 的最小值为

$$p = 0.451m \tag{21}$$

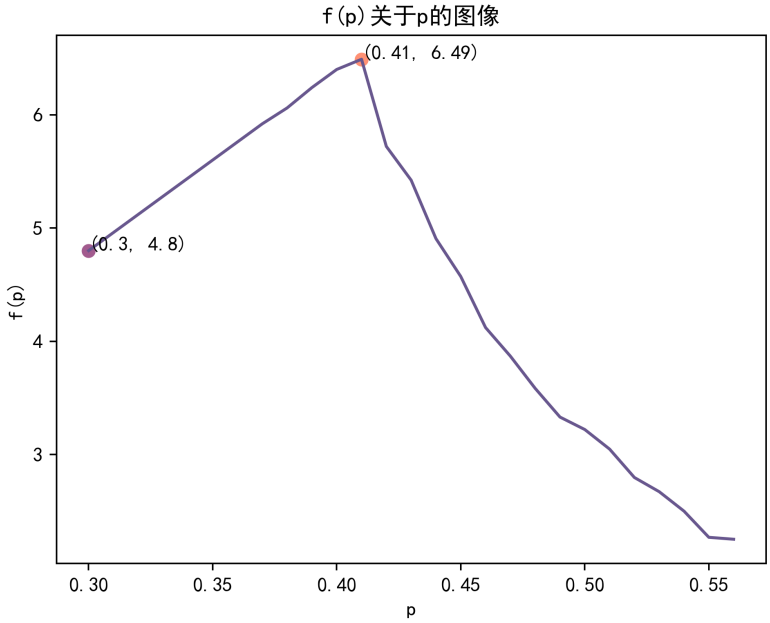


图 5.4.1　碰撞半径关于螺距的变化曲线

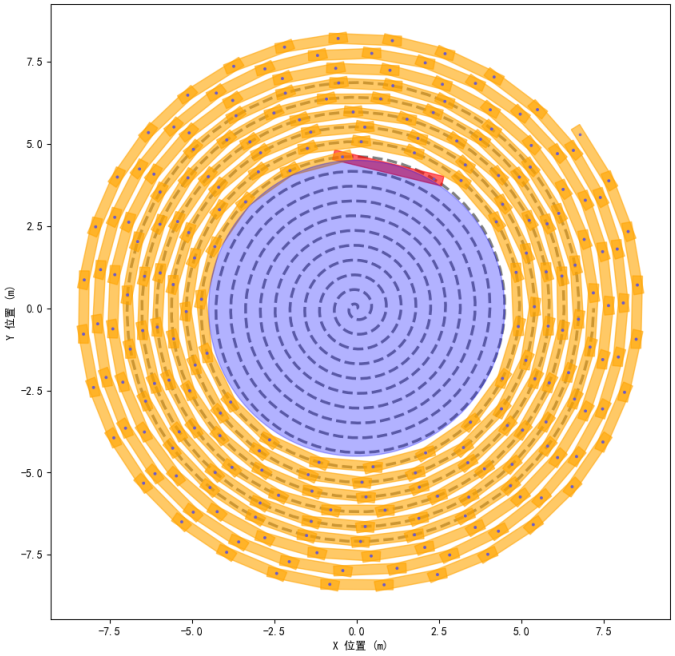


图 5.4.2　龙头到达调头空间边界

5.5 问题四模型的建立与求解

考虑本问中螺纹宽度为 1.7m，其中心对称后的盘入与盘出螺线如下图所示。由图5.2.1可得

$$R_m = \sqrt{(\frac{W}{2})^2 + d^2} \tag{22}$$

计算得到 $R_m \approx 0.31$ m，$2R_m < 1.7$m，故螺纹间距已足够大，为此，不再需要考虑相邻螺线间板凳的碰撞问题。

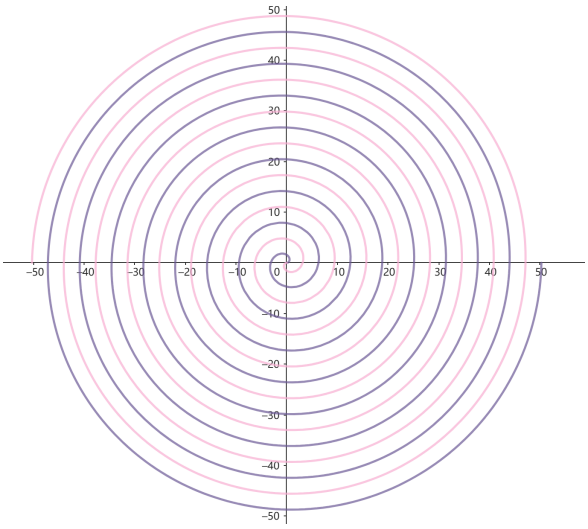


图 5.5.1　中心对称螺线结构

(i) 调头曲线的优化

对于给定调头区域半径 R，考虑中心对称以及两段圆弧与盘入、盘出螺线三处相切这一条件，且两段圆弧半径 r_1，r_2 满足

$$r_1 : r_2 = 2 : 1 \tag{23}$$

首先确定两段圆弧的圆心坐标 (x_1, y_1),(x_2, y_2) 以及在盘入盘出螺线上的对应切点 $(Rcos\beta, Rsin\beta)$，$(-Rcos\beta, -Rsin\beta)$，由于中心对称，两个切点处有着共同的斜率，即，

$$k = \frac{b\sin\beta + R\cos\beta}{b\cos\beta - R\sin\beta}, \quad R = b \cdot \beta \tag{24}$$

两段圆弧均满足与螺线相切，即需满足，

$$\begin{cases} \dfrac{y_1 - R\sin\beta}{x_1 - R\cos\beta} = -\dfrac{1}{k} \\ \dfrac{y_2 + R\sin\beta}{x_2 + R\cos\beta} = -\dfrac{1}{k} \end{cases} \tag{25}$$

两段圆弧满足半径比为 $2:1$，有

$$\sqrt{(x_1 - R\cos\beta)^2 + (y_1 - R\sin\beta)^2} = 2\sqrt{(x_2 + R\cos\beta)^2 + (y_2 + R\sin\beta)^2} \tag{26}$$

同时两段圆弧满足相切，有圆心距离等于半径之和，即

$$\begin{aligned} \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} = &\sqrt{(x_1 - R\cos\beta)^2 + (y_1 - R\sin\beta)^2} + \\ &\sqrt{(x_2 + R\cos\beta)^2 + (y_2 + R\sin\beta)^2} \end{aligned} \tag{27}$$

通过编写 Python 代码求解相关参数，得到相应结果如下，

$$\begin{cases} O_1(-0.76,-1.31) \\ O_2(1.74,2.45) \\ r_1 = 3\ \text{m} \\ r_2 = 1.5\ \text{m} \end{cases} \tag{28}$$

最终计算得到调头路径（即两段弧长之和），$l_0 = 13.62$ m。

具体调头路径如下图所示，

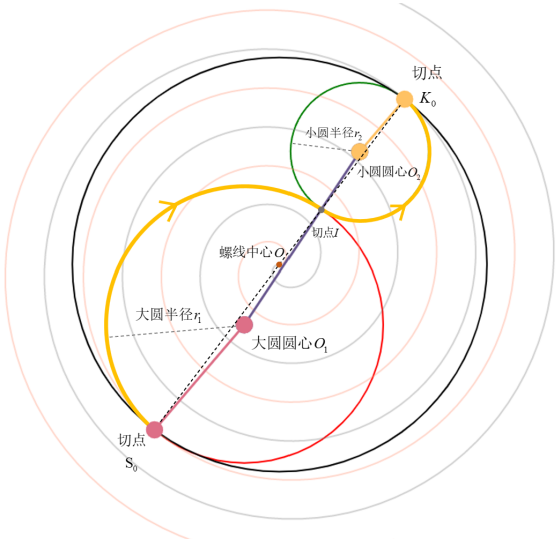


图 5.5.2　未优化调头路径（黄色曲线）

为了进一步优化，使得调头路径长度减小，可以减小实际调头区域的半径，但与此同时，需考虑因为小圆半径的进一步减小，可能会导致板凳无法通过，极端位置如下图所示，

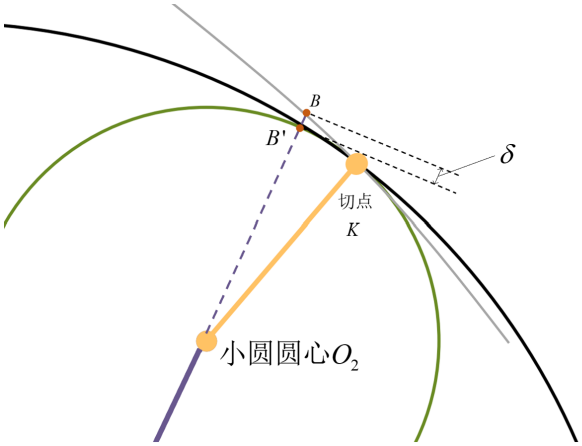


图 5.5.3　“板凳龙”通过调头曲线极限情况

此时龙头板凳的两个把手分别位于小圆与大圆的切点 I，以及 IO_2 延长线交于盘出螺纹于点 B，有

$$2r_2+\delta=D_1 \tag{29}$$

但此时点 I 没有沿板凳方向的速度分量，而点 B 有沿板凳方向的速度分量，不满足板凳长度始终不变这一物理约束，所以小圆半径 r_2 需满足

$$r_2>\frac{D_1-\delta}{2} \tag{30}$$

综合上述分析，本文给出“板凳龙”调头曲线的优化模型如下

$$\min r_2$$

$$\text{s.t.}\begin{cases}\dfrac{y_1-R\sin\beta}{x_1-R\cos\beta}=-\dfrac{1}{k}\\[2ex]\dfrac{y_2+R\sin\beta}{x_2+R\cos\beta}=-\dfrac{1}{k}\\[2ex]\sqrt{(x_1-R\cos\beta)^2+(y_1-R\sin\beta)^2}=2\sqrt{(x_2+R\cos\beta)^2+(y_2+R\sin\beta)^2}\\[2ex]\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}=\sqrt{(x_1-R\cos\beta)^2+(y_1-R\sin\beta)^2}+\\\qquad\qquad\sqrt{(x_2+R\cos\beta)^2+(y_2+R\sin\beta)^2}\\[2ex]\sqrt{(x_2+R\cos\beta)^2+(y_2+R\sin\beta)^2}\\[2ex]r_2>\dfrac{D_1-\delta}{2}\end{cases} \tag{31}$$

考虑到

$$\delta<<D_1 \tag{32}$$

取 $r_2=\frac{D_1}{2}$，并得到此时实际调头区域半径 $R=4.28\,\text{m}$。同时，本文根据式 (30)构建模拟退火算法来优化 R，并根据得到的 R 计算 r_2。两次得到的 r_2 相近，故本文取 $r_2=\frac{D_1}{2}$，再代回上述方程，最终得到优化后的相关参数，如下

$$\begin{cases}O_1(-1.40,-0.34)\\O_2(2.82,0.42)\\r_1=2.86\,\text{m}\\r_2=1.43\,\text{m}\end{cases} \tag{33}$$

得到优化后的两段弧长分别为

$$\begin{cases}l_1=8.62\,\text{m}\\l_2=4.31\,\text{m}\end{cases} \tag{34}$$

最终调头路径长度为

$$l = l_1 + l_2 = 12.93\ \text{m} \tag{35}$$

优化后的具体调头路径如下图所示，

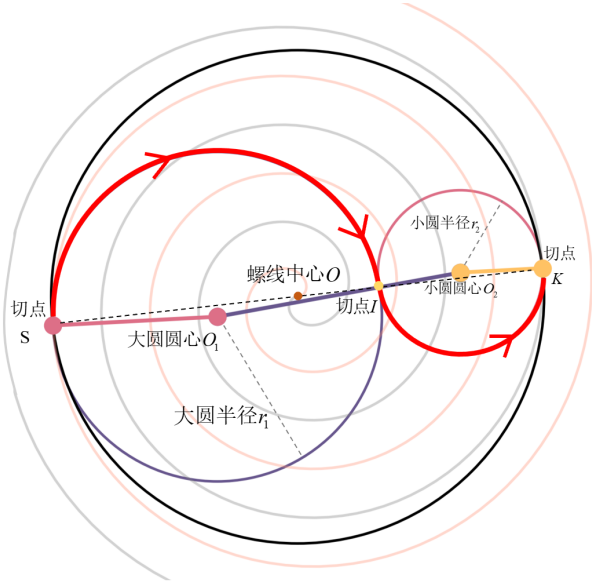


图 5.5.4　优化后调头路径示意图

(ii) “板凳龙”在-100 至 100s 运动状态的分析与求解

由于在-100s 至 0 时刻，“板凳龙”只在盘入螺线上进行运动，其位置信息与速度信息可以通过问题一中的方法得到。而在 0 至 100s 时刻，“板凳龙”会出现在至多四条曲线上，为此，本文对其进行了分段分类讨论来求解任意时刻“板凳龙”的坐标信息，四条曲线标号见下图，

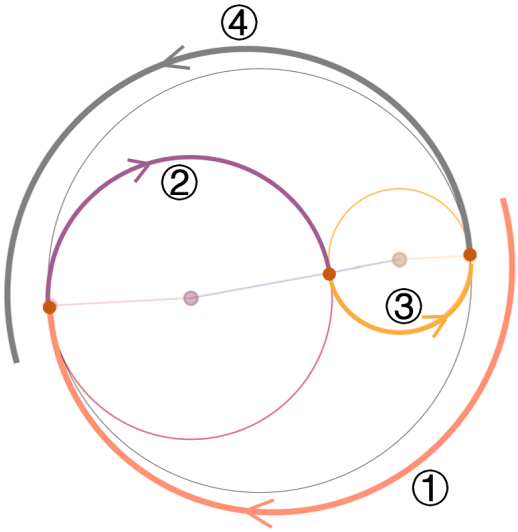


图 5.5.5　调头路径区域划分

为确定“板凳龙”各部分处于那些曲线段，首先对 0 至 100s 进行时间分段，基于曲线情况分出了如下三段，

$$\begin{cases} 0 < t \leqslant \dfrac{l_1}{V_{\text{head}}} & \text{，“板凳龙”只处于 1，2 曲线段} \\ \dfrac{l_1}{V_{\text{head}}} < t \leqslant \dfrac{l_1 + l_2}{V_{\text{head}}} & \text{，“板凳龙”只处于 1，2，3 曲线段} \\ \dfrac{l_1 + l_2}{V_{\text{head}}} < t \leqslant 100 & \text{，“板凳龙”处于 1，2，3，4 曲线段} \end{cases} \tag{36}$$

四段曲线各自的曲线方程为，

$$\begin{cases} r = b\theta, \quad \theta > 0 \\ (x - x_{01})^2 + (y - y_{01})^2 = r_1^2 \\ (x - x_{02})^2 + (y - y_{02})^2 = r_2^2 \\ r = -b\theta, \quad \theta < 0. \end{cases} \tag{37}$$

随后，本文确定“板凳龙”调头的迭代模型如下。

STEP1：对于某一时刻 t，基于式 (4)可得任意时刻走过的弧长，进而解出龙头前把手 (n=1) 的坐标，判定此刻龙头所在弧线编号为 k。

STEP2：根据把手 n 所在弧线 k 的方程向后迭代，得到把手 $n+1$ 的坐标。

STEP3：判断得到的把手 $n+1$ 是否仍在弧线段 k 内。如果把手 $n+1$ 超出螺线段 k 的范围，则需要切换至下一个螺线段 $k+1$，并在新的螺线段方程上计算与前一节距离为 $D_{1,2}$ 的点。将该点作为起点，继续迭代计算。

STEP4：重复步骤 1 - 3 ，直至迭代出时刻 t 下最后一个点的坐标。

STEP5：对时刻 $t+1$，重复步骤 1-4 的过程。

获得任意时刻“板凳龙”的点坐标信息后，再次通过在曲线上分类的方式求出每点的速度方向，单位化后用 $\vec{V_i}$ 表示，同时根据相邻两点计算出板凳方向，单位化后用 $\vec{A_i}$ 表示，得到每一点速度在其板凳方向的余弦分量，

$$cos\phi_i = \vec{V_i} \cdot \vec{A_i} \tag{38}$$

再次由式 (15)中提到的物理约束，来进行求解相邻两点的速度之比 k_i，最终一一迭代，求出任意时刻“板凳龙”各点的速度。

通过编写 Python 代码，通过迭代求解出-100s 至 100s 每秒钟板凳龙的坐标和速度，见 result4.xlsx；部分时刻结果展示如表 4、表 5。

表 4　问题四龙的各节点在不同时间点的坐标

	-100 s	-50 s	0 s	50 s	100s
龙头 x (m)	8.504254	6.531955	-4.253717	3.098031	-1.287447
龙头 y (m)	0.241823	-1.635590	-0.473593	5.410044	8.015352
第 1 节龙身 x (m)	8.034496	6.739352	-3.182018	5.097351	1.568696
第 1 节龙身 y (m)	3.062980	1.216880	-3.125208	3.364973	7.866868
第 51 节龙身 x (m)	-9.064496	-6.564402	-1.063993	-5.950182	3.497837
第 51 节龙身 y (m)	6.037329	-6.964413	-7.968363	-0.536709	2.716258
第 101 节龙身 x (m)	-12.708913	7.812838	6.140269	-1.370861	-4.176240
第 101 节龙身 y (m)	-1.328747	-8.673385	8.462125	8.861116	6.093850
第 151 节龙身 x (m)	-14.333166	11.465798	-3.784833	-9.845330	6.384542
第 151 节龙身 y (m)	1.582389	-7.033371	11.817566	-5.305203	-7.648714
第 201 节龙身 x (m)	-9.267818	7.657458	-3.558299	-6.714151	11.228854
第 201 节龙身 y (m)	12.911588	-12.921150	13.637482	-11.165231	4.224885
龙尾（后）x (m)	-4.515046	3.714405	-5.351109	11.367060	-12.649196
龙尾（后）y (m)	-15.870242	15.213327	-13.771645	7.757803	-1.894048

表 5　问题四龙的各节点在不同时间点的速度

	-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)	0.999899	0.999741	0.998391	1.000383	1.000128
第 51 节龙身 (m/s)	0.999317	0.998550	0.994373	0.680021	1.004246
第 101 节龙身 (m/s)	0.999054	0.998142	0.993655	0.736015	1.936996
第 151 节龙身 (m/s)	0.998905	0.997936	0.993354	0.735942	1.945279
第 201 节龙身 (m/s)	0.998808	0.997812	0.993189	0.735907	1.948443
龙尾（后）(m/s)	0.998776	0.997771	0.993138	0.735897	1.949290

在 $t = 100s$ 时刻，“板凳龙”的构型如下图所示。

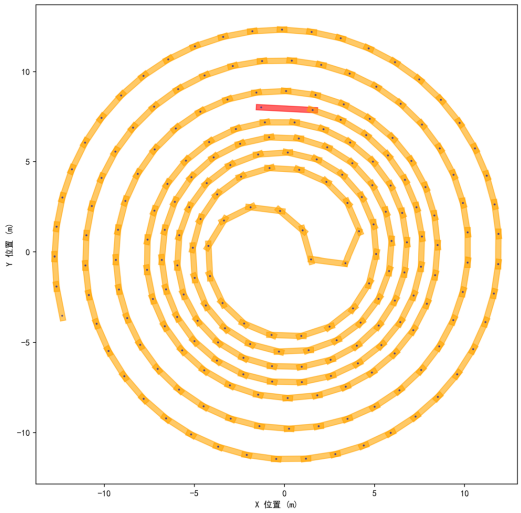


图 5.5.6　“板凳龙”通过调头曲线

5.6 问题五模型的建立与求解

在问题四中已经得到了在该路线下“板凳龙”在给定时间与空间条件下各点的速度分布，同时可以发现在给定“板凳龙”自身物理构型的情况下，各点速度均成一定比例。基于此前提下，为保证各把手的速度均不超过 $v_{ref}=2m/s$ 的前提下，只需将问题四中最大速度 v_{max} 调至 $2m/s$，再根据比例求得此时的龙头前把手速度 v'_{head} 即，

$$\frac{v_{max}}{v_{ref}}=\frac{v_{head}}{v'_{head}} \tag{39}$$

在遍历问题四关于速度的结果后，找出 $v_{max}=1.97m/s$ 代入上式，得到所需的龙头前把手速度 $v'_{head}=1.02m/s$

六、 模型的评价与推广

模型的优点

1. 牛顿法解决隐函数求解问题，而非遍历自变量去使目标函数逼近解，精度较高。

模型的缺点

1. 本文涉及的隐函数都存在多解问题，应用牛顿法求解时，为保证准确收敛到目标解，初始值的确定难度高。

模型的推广

1. 本文重点涉及了定长线段在阿基米德螺线上动态滑动的问题，能够对阿基米德螺线的性质有进一步研究与认识。

参考文献

[1] 同济大学数学科学学院. 高等数学 (第八版, 上册)[M]. 北京: 高等教育出版社, 2023, 276-279.

[2] 王明华, 杨继绪. 阿基米德螺线的性质与应用[J]. 数学通报, 1989, (07):11-12.

附录 A　支撑材料

名称	类型	简介
q1.py	Python 代码文件	问题一求解代码
q2_cal.py	Python 代码文件	问题二计算碰撞时间代码
q2_dis.py	Python 代码文件	问题二求解分布代码
q3.py	Python 代码文件	问题三求解代码
q4_normal.py	Python 代码文件	问题四未优化路径计算代码
q4_optimize.py	Python 代码文件	问题四优化路径计算代码
q4_cal-100to0.py	Python 代码文件	问题四计算前 101 秒分布代码
q4_cal1to100.py	Python 代码文件	问题四计算后 100 秒分布代码
result1.xlsx	Excel 结果文件	问题一结果
result2.xlsx	Excel 结果文件	问题二结果
result4.xlsx	Excel 结果文件	问题四结果

附录 B　代码

第一问 python 源代码

```python
# q1.py
import numpy as np
import pandas as pd
from scipy.optimize import fsolve

# 参数设置
p = 0.55 # 螺距（米）
b = p / (2 * np.pi) # 增长系数
v_head = 1 # 龙头速度（m/s）
time_duration = 300 # 模拟时间 300 秒
time_steps = np.arange(0, time_duration + 1) # 时间序列
length_head = 3.41 # 龙头长度（米）
length_body = 2.2 # 龙身和龙尾长度（米）
num_segments = 223 # 总节数（1 个龙头 + 221 个龙身 + 1 个龙尾）
theta_0 = 32 * np.pi # 初始角度（弧度）


def line_vector(theta_n, theta_n1, b):
    x_n = b * theta_n * np.cos(theta_n)
    y_n = b * theta_n * np.sin(theta_n)
```

```
    x_n1 = b * theta_n1 * np.cos(theta_n1)
    y_n1 = b * theta_n1 * np.sin(theta_n1)

    return np.array([x_n1 - x_n, y_n1 - y_n])


def tangent_vector(theta):
    tan_slope = (np.sin(theta) + theta * np.cos(theta)) / (np.cos(theta) - theta *
        np.sin(theta))
    return np.array([1, tan_slope])


def cos_phi(theta_n, theta_n1, b):
    line_vec = line_vector(theta_n, theta_n1, b)
    tangent_vec_n = tangent_vector(theta_n)
    tangent_vec_n1 = tangent_vector(theta_n1)
    cos_phi_n = np.dot(tangent_vec_n, line_vec) / (np.linalg.norm(tangent_vec_n) *
        np.linalg.norm(line_vec))
    cos_phi_n1 = np.dot(tangent_vec_n1, line_vec) / (np.linalg.norm(tangent_vec_n1) *
        np.linalg.norm(line_vec))
    return np.abs(cos_phi_n), np.abs(cos_phi_n1)


def calculate_v_n1(v_n, theta_n, theta_n1, b):
    cos_phi_n, cos_phi_n1= cos_phi(theta_n, theta_n1, b)
    v_n1 = (v_n * cos_phi_n) / cos_phi_n1
    return v_n1


def arc_length(theta):
    return 0.5 * b * (theta * np.sqrt(theta ** 2 + 1) + np.arcsinh(theta))


def func_head(y, x):
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (0.55 ** 2) *
        (length_head - 0.55) ** 2


def func_body(y, x):
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (0.55 ** 2) *
        (length_body - 0.55) ** 2



s_0 = arc_length(theta_0)
s_head = s_0 - v_head * time_steps
```

```
# 先计算头部
theta_head = []
for s in s_head:
    func = lambda theta: arc_length(theta) - s
    theta_initial_guess = theta_0 + (s - s_0) / (b * np.sqrt(theta_0 ** 2 + 1))
    theta_solution = fsolve(func, theta_initial_guess)
    theta_head.append(theta_solution[0])
theta_head = np.array(theta_head)

r_head = b * theta_head
x_head = r_head * np.cos(theta_head)
y_head = r_head * np.sin(theta_head)

positions_x = [x_head]
positions_y = [y_head]
velocities = [v_head]

# 第一节
theta_initial = []
for theta in theta_head:
    theta_initial_guess = theta + 0.1
    theta_initial_solution = fsolve(func_head, theta_initial_guess, args=theta)
    theta_initial.append(theta_initial_solution[0])
theta_initial = np.array(theta_initial)

r_initial = b * theta_initial
x_initial = r_initial * np.cos(theta_initial)
y_initial = r_initial * np.sin(theta_initial)
v_initial = []
for i in range(0, time_duration + 1):
    v_n1 = calculate_v_n1(v_head, theta_head[i], theta_initial[i], b)
    v_initial.append(v_n1)
v_initial = np.array(v_initial)

positions_x.append(x_initial)
positions_y.append(y_initial)
velocities.append(v_initial)

# 迭代求后面部分
for i in range(2, num_segments + 1):
    theta_l = []
    for theta in theta_initial:
        theta_guess = theta + 0.1
        theta_solution = fsolve(func_body, theta_guess, args=theta)
        theta_l.append(theta_solution[0])
    theta_l = np.array(theta_l)
    r = b * theta_l
```

```
    x = r * np.cos(theta_l)
    y = r * np.sin(theta_l)
    v = []
    for j in range(0, time_duration + 1):
        v_n1 = calculate_v_n1(v_initial[j], theta_initial[j], theta_l[j], b)
        v.append(v_n1)
    v = np.array(v)
    positions_x.append(x)
    positions_y.append(y)
    velocities.append(v)
    theta_initial = theta_l
    v_initial = v

data_dict1 = {
    'Time (s)': time_steps
}
data_dict2 = {
    'Time (s)': time_steps
}
for i in range(num_segments + 1):
    data_dict1[f'第{i+1}节x (m)'] = positions_x[i]
    data_dict1[f'第{i+1}节y (m)'] = positions_y[i]
for i in range(num_segments + 1):
    data_dict2[f'第{i+1}节速度 (m/s)'] = velocities[i]
df1 = pd.DataFrame(data_dict1)
df2 = pd.DataFrame(data_dict2)
df1_transposed = df1.transpose()
df2_transposed = df2.transpose()
with pd.ExcelWriter('result.xlsx') as writer:
    df1_transposed.to_excel(writer, sheet_name='Sheet1', header=True)
    df2_transposed.to_excel(writer, sheet_name='Sheet2', header=True)
```

第二问 python 源代码

```
# q2_cal.py
import numpy as np
from scipy.optimize import fsolve

# 参数设置
p = 0.55 # 螺距（米）
b = p / (2 * np.pi) # 增长系数
v_head = 1 # 龙头速度（m/s）
time_duration = 430 # 模拟时间 400 秒
time_steps = np.arange(0, time_duration + 1) # 时间序列
length_head = 3.41 # 龙头长度（米）
length_body = 2.2 # 龙身和龙尾长度（米）
num_segments = 223 # 总节数（1 个龙头 + 221 个龙身 + 1 个龙尾）
```

```
theta_0 = 32 * np.pi # 初始角度（弧度）


def line_vector(theta_n, theta_n1, b):
    x_n = b * theta_n * np.cos(theta_n)
    y_n = b * theta_n * np.sin(theta_n)
    x_n1 = b * theta_n1 * np.cos(theta_n1)
    y_n1 = b * theta_n1 * np.sin(theta_n1)

    return np.array([x_n1 - x_n, y_n1 - y_n])


def tangent_vector(theta):
    tan_slope = (np.sin(theta) + theta * np.cos(theta)) / (np.cos(theta) - theta *
        np.sin(theta))
    return np.array([1, tan_slope])


def cos_phi(theta_n, theta_n1, b):
    line_vec = line_vector(theta_n, theta_n1, b)
    tangent_vec_n = tangent_vector(theta_n)
    tangent_vec_n1 = tangent_vector(theta_n1)
    cos_phi_n = np.dot(tangent_vec_n, line_vec) / (np.linalg.norm(tangent_vec_n) *
        np.linalg.norm(line_vec))
    cos_phi_n1 = np.dot(tangent_vec_n1, line_vec) / (np.linalg.norm(tangent_vec_n1) *
        np.linalg.norm(line_vec))
    return np.abs(cos_phi_n), np.abs(cos_phi_n1)


def calculate_v_n1(v_n, theta_n, theta_n1, b):
    cos_phi_n, cos_phi_n1= cos_phi(theta_n, theta_n1, b)
    # 计算v_(n+1)
    v_n1 = (v_n * cos_phi_n) / cos_phi_n1
    return v_n1


def arc_length(theta):
    return 0.5 * b * (theta * np.sqrt(theta ** 2 + 1) + np.arcsinh(theta))


def func_head(y, x):
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (0.55 ** 2) *
        (length_head - 0.55) ** 2


def func_body(y, x):
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (0.55 ** 2) *
```

```
        (length_body - 0.55) ** 2


def calculate_rectangle(x1, y1, x2, y2, extend_length=0.275, width=0.15):
    vec = np.array([x2 - x1, y2 - y1])
    distance = np.linalg.norm(vec)
    unit_vec = vec / distance
    p1 = np.array([x1, y1]) - extend_length * unit_vec
    p2 = np.array([x2, y2]) + extend_length * unit_vec
    perp_vec = np.array([-unit_vec[1], unit_vec[0]])
    top_left = p1 + width * perp_vec
    bottom_left = p1 - width * perp_vec
    top_right = p2 + width * perp_vec
    bottom_right = p2 - width * perp_vec
    return top_left, bottom_left, top_right, bottom_right


def is_point_in_rectangle(point, rect_points):
    Px, Py = point
    A, B, C, D = rect_points


    def cross_product(O, A, B):
        return (A[0] - O[0]) * (B[1] - O[1]) - (A[1] - O[1]) * (B[0] - O[0])


    AB_P = cross_product(A, B, (Px, Py))
    BC_P = cross_product(B, C, (Px, Py))
    CD_P = cross_product(C, D, (Px, Py))
    DA_P = cross_product(D, A, (Px, Py))
    if (AB_P >= 0 and BC_P >= 0 and CD_P >= 0 and DA_P >= 0) or \
            (AB_P <= 0 and BC_P <= 0 and CD_P <= 0 and DA_P <= 0):
        return 1
    else:
        return 0


s_0 = arc_length(theta_0)
s_head = s_0 - v_head * time_steps

theta_head = []
for s in s_head:
    func = lambda theta: arc_length(theta) - s
    theta_initial_guess = theta_0 + (s - s_0) / (b * np.sqrt(theta_0 ** 2 + 1))
    theta_solution = fsolve(func, theta_initial_guess)
    theta_head.append(theta_solution[0])
theta_head = np.array(theta_head)
```

```
r_head = b * theta_head
x_head = r_head * np.cos(theta_head)
y_head = r_head * np.sin(theta_head)

positions_x = [x_head]
positions_y = [y_head]

theta_initial = []
for theta in theta_head:
    theta_initial_guess = theta + 0.5
    theta_initial_solution = fsolve(func_head, theta_initial_guess, args=theta)
    theta_initial.append(theta_initial_solution[0])
theta_initial = np.array(theta_initial)

r_initial = b * theta_initial
x_initial = r_initial * np.cos(theta_initial)
y_initial = r_initial * np.sin(theta_initial)

positions_x.append(x_initial)
positions_y.append(y_initial)

theta_every = [theta_head, theta_initial]
theta_first = theta_initial
for i in range(2, num_segments + 1):
    theta_l = []
    for theta in theta_initial:
        theta_guess = theta + 0.5
        theta_solution = fsolve(func_body, theta_guess, args=theta)
        theta_l.append(theta_solution[0])
    theta_l = np.array(theta_l)
    r = b * theta_l
    x = r * np.cos(theta_l)
    y = r * np.sin(theta_l)
    positions_x.append(x)
    positions_y.append(y)
    theta_initial = theta_l
    theta_every.append(theta_l)

theta_get_final = []
for i in range(360, time_duration + 1):
    theta_cal = []
    theta_guess1 = theta_head[i] + 2 * np.pi
    theta_guess2 = theta_first[i] + 2 * np.pi
    for j in range(1, num_segments):
        if theta_every[j][i] < theta_guess1 < theta_every[j + 1][i]:
            theta_cal.append((theta_every[j][i], theta_every[j + 1][i]))
```

```
        if theta_every[j][i] < theta_guess2 < theta_every[j + 1][i]:
            theta_cal.append((theta_every[j][i], theta_every[j + 1][i]))
            break
    theta_get_final.append(theta_cal)
theta_get_final = np.array(theta_get_final)

crash_point_large = 0
for i in range(0, len(theta_get_final)):
    top_left_head, bottom_left_head, top_right_head, bottom_right_head =
        calculate_rectangle(x_head[i + 360], y_head[i + 360], x_initial[i + 360], y_initial[i +
        360])
    r_impact1 = b * theta_get_final[i][0]
    x_impact1 = r_impact1 * np.cos(theta_get_final[i][0])
    y_impact1 = r_impact1 * np.sin(theta_get_final[i][0])
    top_left1, bottom_left1, top_right1, bottom_right1 = calculate_rectangle(x_impact1[0],
        y_impact1[0], x_impact1[1], y_impact1[1])
    r_impact2 = b * theta_get_final[i][1]
    x_impact2 = r_impact2 * np.cos(theta_get_final[i][1])
    y_impact2 = r_impact2 * np.sin(theta_get_final[i][1])
    top_left2, bottom_left2, top_right2, bottom_right2 = calculate_rectangle(x_impact2[0],
        y_impact2[0], x_impact2[1], y_impact2[1])
    vertices1 = np.array([top_left1, bottom_left1, bottom_right1, top_right1])
    vertices2 = np.array([top_left2, bottom_left2, bottom_right2, top_right2])
    if is_point_in_rectangle(top_left_head, vertices1) or
        is_point_in_rectangle(bottom_left_head, vertices1) or \
          is_point_in_rectangle(top_right_head, vertices1) or
              is_point_in_rectangle(bottom_right_head, vertices1) or \
          is_point_in_rectangle(top_left_head, vertices2) or
              is_point_in_rectangle(bottom_left_head, vertices2) or \
          is_point_in_rectangle(top_right_head, vertices2) or
              is_point_in_rectangle(bottom_right_head, vertices2):
        crash_point_large = i + 360
        break

# 细分时间范围
time_steps_fine = np.arange(crash_point_large - 1, crash_point_large, 0.01)

# 重新计算头部位置和初始位置
s_head_fine = s_0 - v_head * time_steps_fine
theta_head_fine = []
for s in s_head_fine:
    func = lambda theta: arc_length(theta) - s
    theta_initial_guess = theta_0 + (s - s_0) / (b * np.sqrt(theta_0 ** 2 + 1))
    theta_solution = fsolve(func, theta_initial_guess)
    theta_head_fine.append(theta_solution[0])
theta_head_fine = np.array(theta_head_fine)
```

```
r_head_fine = b * theta_head_fine
x_head_fine = r_head_fine * np.cos(theta_head_fine)
y_head_fine = r_head_fine * np.sin(theta_head_fine)

theta_initial_fine = []
for theta in theta_head_fine:
    theta_initial_guess = theta + 0.5
    theta_initial_solution = fsolve(func_head, theta_initial_guess, args=theta)
    theta_initial_fine.append(theta_initial_solution[0])
theta_initial_fine = np.array(theta_initial_fine)

r_initial_fine = b * theta_initial_fine
x_initial_fine = r_initial_fine * np.cos(theta_initial_fine)
y_initial_fine = r_initial_fine * np.sin(theta_initial_fine)

# 重新细分计算每一节的位置
theta_every_fine = [theta_head_fine, theta_initial_fine]
theta_first_fine = theta_initial_fine
for i in range(2, num_segments + 1):
    theta_l_fine = []
    for theta in theta_initial_fine:
        theta_guess_fine = theta + 0.5
        theta_solution_fine = fsolve(func_body, theta_guess_fine, args=theta)
        theta_l_fine.append(theta_solution_fine[0])
    theta_l_fine = np.array(theta_l_fine)
    theta_every_fine.append(theta_l_fine)
    theta_initial_fine = theta_l_fine

# 计算细分时间段的碰撞检测
theta_get_final_fine = []
for i in range(len(time_steps_fine)):
    theta_cal_fine = []
    theta_guess1_fine = theta_head_fine[i] + 2 * np.pi
    theta_guess2_fine = theta_first_fine[i] + 2 * np.pi
    for j in range(1, num_segments - 1):
        if theta_every_fine[j][i] < theta_guess1_fine < theta_every_fine[j + 1][i]:
            theta_cal_fine.append((theta_every_fine[j][i], theta_every_fine[j + 1][i]))
        if theta_every_fine[j][i] < theta_guess2_fine < theta_every_fine[j + 1][i]:
            theta_cal_fine.append((theta_every_fine[j][i], theta_every_fine[j + 1][i]))
            break
    theta_get_final_fine.append(theta_cal_fine)
theta_get_final_fine = np.array(theta_get_final_fine)

# 进行碰撞检测
for i in range(len(theta_get_final_fine)):
    top_left_head_fine, bottom_left_head_fine, top_right_head_fine, bottom_right_head_fine =
        calculate_rectangle(
```

```
        x_head_fine[i], y_head_fine[i], x_initial_fine[i], y_initial_fine[i])
    r_impact1_fine = b * theta_get_final_fine[i][0]
    x_impact1_fine = r_impact1_fine * np.cos(theta_get_final_fine[i][0])
    y_impact1_fine = r_impact1_fine * np.sin(theta_get_final_fine[i][0])
    top_left1_fine, bottom_left1_fine, top_right1_fine, bottom_right1_fine =
        calculate_rectangle(
        x_impact1_fine[0], y_impact1_fine[0], x_impact1_fine[1], y_impact1_fine[1])

    r_impact2_fine = b * theta_get_final_fine[i][1]
    x_impact2_fine = r_impact2_fine * np.cos(theta_get_final_fine[i][1])
    y_impact2_fine = r_impact2_fine * np.sin(theta_get_final_fine[i][1])
    top_left2_fine, bottom_left2_fine, top_right2_fine, bottom_right2_fine =
        calculate_rectangle(
        x_impact2_fine[0], y_impact2_fine[0], x_impact2_fine[1], y_impact2_fine[1])

    vertices1_fine = np.array([top_left1_fine, bottom_left1_fine, bottom_right1_fine,
        top_right1_fine])
    vertices2_fine = np.array([top_left2_fine, bottom_left2_fine, bottom_right2_fine,
        top_right2_fine])

    if is_point_in_rectangle(top_left_head_fine, vertices1_fine) or \
            is_point_in_rectangle(bottom_left_head_fine, vertices1_fine) or \
            is_point_in_rectangle(top_right_head_fine, vertices1_fine) or \
            is_point_in_rectangle(bottom_right_head_fine, vertices1_fine) or \
            is_point_in_rectangle(top_left_head_fine, vertices2_fine) or \
            is_point_in_rectangle(bottom_left_head_fine, vertices2_fine) or \
            is_point_in_rectangle(top_right_head_fine, vertices2_fine) or \
            is_point_in_rectangle(bottom_right_head_fine, vertices2_fine):
        crash_point_fine = time_steps_fine[i]
        print(f"碰撞时间为: {round(crash_point_fine, 2)} 秒")
        break


# q2_dis.py
import numpy as np
import pandas as pd
from scipy.optimize import fsolve

# 参数设置
p = 0.55 # 螺距 (米)
b = p / (2 * np.pi) # 增长系数
v_head = 1 # 龙头速度 (m/s)
time_duration = 430 # 模拟时间 300 秒
time_steps = np.arange(0, time_duration + 1) # 时间序列
length_head = 3.41 # 龙头长度 (米)
length_body = 2.2 # 龙身和龙尾长度 (米)
num_segments = 223 # 总节数 (1 个龙头 + 221 个龙身 + 1 个龙尾)
```

```
theta_0 = 32 * np.pi # 初始角度（弧度）


def line_vector(theta_n, theta_n1, b):
    x_n = b * theta_n * np.cos(theta_n)
    y_n = b * theta_n * np.sin(theta_n)
    x_n1 = b * theta_n1 * np.cos(theta_n1)
    y_n1 = b * theta_n1 * np.sin(theta_n1)

    return np.array([x_n1 - x_n, y_n1 - y_n])


def tangent_vector(theta):
    tan_slope = (np.sin(theta) + theta * np.cos(theta)) / (np.cos(theta) - theta *
        np.sin(theta))
    return np.array([1, tan_slope])


def cos_phi(theta_n, theta_n1, b):
    line_vec = line_vector(theta_n, theta_n1, b)
    tangent_vec_n = tangent_vector(theta_n)
    tangent_vec_n1 = tangent_vector(theta_n1)
    cos_phi_n = np.dot(tangent_vec_n, line_vec) / (np.linalg.norm(tangent_vec_n) *
        np.linalg.norm(line_vec))
    cos_phi_n1 = np.dot(tangent_vec_n1, line_vec) / (np.linalg.norm(tangent_vec_n1) *
        np.linalg.norm(line_vec))
    return np.abs(cos_phi_n), np.abs(cos_phi_n1)


def calculate_v_n1(v_n, theta_n, theta_n1, b):
    cos_phi_n, cos_phi_n1= cos_phi(theta_n, theta_n1, b)
    # 计算v_(n+1)
    v_n1 = (v_n * cos_phi_n) / cos_phi_n1
    return v_n1


def arc_length(theta):
    return 0.5 * b * (theta * np.sqrt(theta ** 2 + 1) + np.arcsinh(theta))


def func_head(y, x):
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (0.55 ** 2) *
        (length_head - 0.55) ** 2


def func_body(y, x):
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (0.55 ** 2) *
```

```
        (length_body - 0.55) ** 2


s_0 = arc_length(theta_0)
s_head = s_0 - v_head * time_steps

theta_head = []
for s in s_head:
    func = lambda theta: arc_length(theta) - s
    theta_initial_guess = theta_0 + (s - s_0) / (b * np.sqrt(theta_0 ** 2 + 1))
    theta_solution = fsolve(func, theta_initial_guess)
    theta_head.append(theta_solution[0])
theta_head = np.array(theta_head)

r_head = b * theta_head
x_head = r_head * np.cos(theta_head)
y_head = r_head * np.sin(theta_head)

positions_x = [x_head]
positions_y = [y_head]
v_head_array = np.full_like(x_head, 1)
velocities = [v_head_array]

theta_initial = []
for theta in theta_head:
    theta_initial_guess = theta + 0.5
    theta_initial_solution = fsolve(func_head, theta_initial_guess, args=theta)
    theta_initial.append(theta_initial_solution[0])
theta_initial = np.array(theta_initial)



r_initial = b * theta_initial
x_initial = r_initial * np.cos(theta_initial)
y_initial = r_initial * np.sin(theta_initial)
v_initial = []
for i in range(0, time_duration + 1):
    v_n1 = calculate_v_n1(v_head, theta_head[i], theta_initial[i], b)
    v_initial.append(v_n1)
v_initial = np.array(v_initial)

positions_x.append(x_initial)
positions_y.append(y_initial)
velocities.append(v_initial)

for i in range(2, num_segments + 1):
    theta_l = []
    for theta in theta_initial:
```

```
        theta_guess = theta + 0.5
        theta_solution = fsolve(func_body, theta_guess, args=theta)
        theta_l.append(theta_solution[0])
    theta_l = np.array(theta_l)
    r = b * theta_l
    x = r * np.cos(theta_l)
    y = r * np.sin(theta_l)
    v = []
    for j in range(0, time_duration + 1):
        v_n1 = calculate_v_n1(v_initial[j], theta_initial[j], theta_l[j], b)
        v.append(v_n1)
    v = np.array(v)
    positions_x.append(x)
    positions_y.append(y)
    velocities.append(v)
    theta_initial = theta_l
    v_initial = v

data_dict = {'节点': []}
x_values = []
y_values = []
v_values = []
for i in range(num_segments + 1):
    data_dict['节点'].append(f'第{i+1}节')
    x_values.append(positions_x[i][413])
    y_values.append(positions_y[i][413])
    v_values.append(velocities[i][413])
data_dict['x (m)'] = x_values
data_dict['y (m)'] = y_values
data_dict['速度 (m/s)'] = v_values
df = pd.DataFrame(data_dict)
with pd.ExcelWriter('result2_distribution.xlsx') as writer:
    df.to_excel(writer, sheet_name='节点数据', index=False)
```

第三问 python 源代码

```
# q3.py
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import fsolve
plt.rcParams['font.sans-serif'] = ['STZhongsong'] # 指定默认字体：解决plot不能显示中文问题
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题
# 参数设置
v_head = 1 # 龙头速度 (m/s)
length_head = 3.41 # 龙头长度 (米)
length_body = 2.2 # 龙身和龙尾长度 (米)
num_segments = 223 # 总节数 (1 个龙头 + 221 个龙身 + 1 个龙尾)
```

```
theta_0 = 32 * np.pi # 初始角度（弧度）


def line_vector(theta_n, theta_n1, b):
    x_n = b * theta_n * np.cos(theta_n)
    y_n = b * theta_n * np.sin(theta_n)
    x_n1 = b * theta_n1 * np.cos(theta_n1)
    y_n1 = b * theta_n1 * np.sin(theta_n1)
    return np.array([x_n1 - x_n, y_n1 - y_n])


def tangent_vector(theta):
    tan_slope = (np.sin(theta) + theta * np.cos(theta)) / (np.cos(theta) - theta *
        np.sin(theta))
    return np.array([1, tan_slope])


def cos_phi(theta_n, theta_n1, b):
    line_vec = line_vector(theta_n, theta_n1, b)
    tangent_vec_n = tangent_vector(theta_n)
    tangent_vec_n1 = tangent_vector(theta_n1)
    cos_phi_n = np.dot(tangent_vec_n, line_vec) / (np.linalg.norm(tangent_vec_n) *
        np.linalg.norm(line_vec))
    cos_phi_n1 = np.dot(tangent_vec_n1, line_vec) / (np.linalg.norm(tangent_vec_n1) *
        np.linalg.norm(line_vec))
    return np.abs(cos_phi_n), np.abs(cos_phi_n1)


def calculate_v_n1(v_n, theta_n, theta_n1, b):
    cos_phi_n, cos_phi_n1= cos_phi(theta_n, theta_n1, b)
    v_n1 = (v_n * cos_phi_n) / cos_phi_n1
    return v_n1


def arc_length(theta, b):
    return 0.5 * b * (theta * np.sqrt(theta ** 2 + 1) + np.arcsinh(theta))


def calculate_rectangle(x1, y1, x2, y2, extend_length=0.275, width=0.15):
    vec = np.array([x2 - x1, y2 - y1])
    distance = np.linalg.norm(vec)
    unit_vec = vec / distance
    p1 = np.array([x1, y1]) - extend_length * unit_vec
    p2 = np.array([x2, y2]) + extend_length * unit_vec
    perp_vec = np.array([-unit_vec[1], unit_vec[0]])
    top_left = p1 + width * perp_vec
    bottom_left = p1 - width * perp_vec
```

```
    top_right = p2 + width * perp_vec
    bottom_right = p2 - width * perp_vec
    return top_left, bottom_left, top_right, bottom_right


def is_point_in_rectangle(point, rect_points):
    Px, Py = point
    A, B, C, D = rect_points


    def cross_product(O, A, B):
        return (A[0] - O[0]) * (B[1] - O[1]) - (A[1] - O[1]) * (B[0] - O[0])


    AB_P = cross_product(A, B, (Px, Py))
    BC_P = cross_product(B, C, (Px, Py))
    CD_P = cross_product(C, D, (Px, Py))
    DA_P = cross_product(D, A, (Px, Py))
    if (AB_P >= 0 and BC_P >= 0 and CD_P >= 0 and DA_P >= 0) or \
            (AB_P <= 0 and BC_P <= 0 and CD_P <= 0 and DA_P <= 0):
        return 1
    else:
        return 0


def calculate_crash_point(p):
    b = p / (2 * np.pi)
    s_0 = arc_length(theta_0, b)
    time_duration = int(1000 * p - 120)
    time_steps = np.arange(0, time_duration + 1)
    s_head = s_0 - v_head * time_steps
    theta_head = []
    for s in s_head:
        func = lambda theta: arc_length(theta, b) - s
        theta_initial_guess = theta_0 + (s - s_0) / (b * np.sqrt(theta_0 ** 2 + 1))
        theta_solution = fsolve(func, theta_initial_guess)
        theta_head.append(theta_solution[0])
    theta_head = np.array(theta_head)

    r_head = b * theta_head
    x_head = r_head * np.cos(theta_head)
    y_head = r_head * np.sin(theta_head)
    positions_x = [x_head]
    positions_y = [y_head]

    theta_initial = []
    def func_head(y, x):
```

```
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) *
        (length_head - 0.55) ** 2

for theta in theta_head:
    theta_initial_guess = theta + 0.5
    theta_initial_solution = fsolve(func_head, theta_initial_guess, args=theta)
    theta_initial.append(theta_initial_solution[0])
theta_initial = np.array(theta_initial)

r_initial = b * theta_initial
x_initial = r_initial * np.cos(theta_initial)
y_initial = r_initial * np.sin(theta_initial)

positions_x.append(x_initial)
positions_y.append(y_initial)

theta_every = [theta_head, theta_initial]
theta_first = theta_initial
def func_body(y, x):
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) *
        (length_body - 0.55) ** 2

for i in range(2, num_segments + 1):
    theta_l = []
    for theta in theta_initial:
        theta_guess = theta + 0.5
        theta_solution = fsolve(func_body, theta_guess, args=theta)
        theta_l.append(theta_solution[0])
    theta_l = np.array(theta_l)
    r = b * theta_l
    x = r * np.cos(theta_l)
    y = r * np.sin(theta_l)
    positions_x.append(x)
    positions_y.append(y)
    theta_initial = theta_l
    theta_every.append(theta_l)

theta_get_final = []
for i in range(0, time_duration + 1):
    theta_cal = []
    theta_guess1 = theta_head[i] + 2 * np.pi
    theta_guess2 = theta_first[i] + 2 * np.pi
    for j in range(1, num_segments):
        if theta_every[j][i] < theta_guess1 < theta_every[j + 1][i]:
            theta_cal.append((theta_every[j][i], theta_every[j + 1][i]))
        if theta_every[j][i] < theta_guess2 < theta_every[j + 1][i]:
            theta_cal.append((theta_every[j][i], theta_every[j + 1][i]))
```

```
                break
        theta_get_final.append(theta_cal)
    theta_get_final = np.array(theta_get_final)

    crash_point = 0
    for i in range(0, len(theta_get_final)):
        top_left_head, bottom_left_head, top_right_head, bottom_right_head =
            calculate_rectangle(x_head[i], y_head[i], x_initial[i], y_initial[i])
        r_impact1 = b * theta_get_final[i][0]
        x_impact1 = r_impact1 * np.cos(theta_get_final[i][0])
        y_impact1 = r_impact1 * np.sin(theta_get_final[i][0])
        top_left1, bottom_left1, top_right1, bottom_right1 = calculate_rectangle(x_impact1[0],
            y_impact1[0], x_impact1[1], y_impact1[1])
        r_impact2 = b * theta_get_final[i][1]
        x_impact2 = r_impact2 * np.cos(theta_get_final[i][1])
        y_impact2 = r_impact2 * np.sin(theta_get_final[i][1])
        top_left2, bottom_left2, top_right2, bottom_right2 = calculate_rectangle(x_impact2[0],
            y_impact2[0], x_impact2[1], y_impact2[1])
        vertices1 = np.array([top_left1, bottom_left1, bottom_right1, top_right1])
        vertices2 = np.array([top_left2, bottom_left2, bottom_right2, top_right2])
        if is_point_in_rectangle(top_left_head, vertices1) or
            is_point_in_rectangle(bottom_left_head, vertices1) or \
               is_point_in_rectangle(top_right_head, vertices1) or
                   is_point_in_rectangle(bottom_right_head, vertices1) or \
               is_point_in_rectangle(top_left_head, vertices2) or
                   is_point_in_rectangle(bottom_left_head, vertices2) or \
               is_point_in_rectangle(top_right_head, vertices2) or
                   is_point_in_rectangle(bottom_right_head, vertices2):
            crash_point = i
            break
    r_head_final = b * theta_head[crash_point]
    return r_head_final


def compute_gradient(p, epsilon=1e-3):
    f_p = calculate_crash_point(p)
    f_p_plus_epsilon = calculate_crash_point(p + epsilon)
    gradient = (f_p_plus_epsilon - f_p) / epsilon
    return gradient


def binary(left, right):
    left = left * 1000
    right = right * 1000
    while right > left:
        mid = (left + right) // 2
        r_head_final = calculate_crash_point(mid / 1000)
```

```
        if r_head_final <= 4.5:
            right = mid - 1
        else:
            left = mid + 1
    return left / 1000


g = []
p = np.arange(0.3, 0.56, 0.01)
for p_x in p:
    g.append(calculate_crash_point(p_x))
g = np.array(g)

x_point1 = p[0]
y_point1 = g[0]
plt.scatter(x_point1, y_point1, color='red')
plt.annotate(f'({x_point1}, {y_point1})', xy=(x_point1, y_point1), xytext=(x_point1, y_point1))
max_index = np.argmax(g)
x_point2 = round(p[max_index], 2)
y_point2 = round(g[max_index], 2)
plt.scatter(x_point2, y_point2, color='red')
plt.annotate(f'({x_point2}, {y_point2})', xy=(x_point2, y_point2), xytext=(x_point2, y_point2))
plt.plot(p, g)
plt.title('f(p)关于p的图像')
plt.xlabel('p')
plt.ylabel('f(p)')
plt.show()

p_max = 0.55
p_min = p[max_index]
p_final = binary(p_min, p_max)
print("最小螺距为(m)", p_final)
```

第四问 python 源代码

```
# q4_normal.py
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import fsolve
plt.rcParams['font.sans-serif'] = ['STZhongsong'] # 指定默认字体：解决plot不能显示中文问题
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题

# 参数设置
v_head = 1 # 龙头速度 (m/s)
p = 1.7 # 螺距 (米)
R_ratio = 2 # 半径之比 R1:R2 = 2:1
R = 4.5 # 螺旋线在 r=4.5 处与圆相切
```

```
b = p / (2 * np.pi) # 增长系数
length_head = 3.41 # 龙头长度（米）
length_body = 2.2 # 龙身和龙尾长度（米）
num_segments = 223 # 总节数（1 个龙头 + 221 个龙身 + 1 个龙尾）
theta_0 = 32 * np.pi # 初始角度（弧度）

def equations(vars):
    x1, y1, x2, y2 = vars
    k = (b * np.sin(R / b) + R * np.cos(R / b)) / (b * np.cos(R / b) - R * np.sin(R / b))
    eq1 = y1 - R * np.sin(R / b) + (x1 - R * np.cos(R / b)) / k
    eq2 = y2 + R * np.sin(R / b) + (x2 + R * np.cos(R / b)) / k
    eq3 = (x1 - R * np.cos(R / b)) ** 2 + (y1 - R * np.sin(R / b)) ** 2 - 4 * ((x2 + R *
        np.cos(R / b)) ** 2 + (y2 + R * np.sin(R / b)) ** 2)
    eq4 = np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2) - np.sqrt((x1 - R * np.cos(R / b)) ** 2 +
        (y1 - R * np.sin(R / b)) ** 2) \
         - np.sqrt((x2 + R * np.cos(R / b)) ** 2 + (y2 + R * np. sin(R / b)) ** 2)
    return [eq1, eq2, eq3, eq4]


def plot_circle(ax, x_center, y_center, radius, color='b', label=None):
    circle = plt.Circle((x_center, y_center), radius, color=color, fill=False, label=label)
    ax.add_patch(circle)


def spiral(b, theta):
    r = b * theta
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y


initial_guess = [-1, 3, 1, -2]
solution = fsolve(equations, initial_guess)
x1, y1, x2, y2 = solution
R1 = np.sqrt((x1 - R * np.cos(R / b)) ** 2 + (y1 - R * np.sin(R / b))** 2)
R2 = np.sqrt((x2 + R * np.cos(R / b)) ** 2 + (y2 + R * np.sin(R / b)) ** 2)
print(f"圆半径：({R1})，圆心坐标：({x1}，{y1})")
print(f"圆半径：({R2})，圆心坐标：({x2}，{y2})")
o1_to_o2 = np.array([x1 - x2, y1 - y2])
vec1 = np.array([x1 - R * np.cos(R / b), y1 - R * np.sin(R / b)])
vec2 = np.array([x2 + R * np.cos(R / b), y2 + R * np.sin(R / b)])
sigma1 = np.arccos(np.dot(o1_to_o2, vec1) / (np.linalg.norm(o1_to_o2) * np.linalg.norm(vec1)))
sigma2 = np.arccos(-np.dot(o1_to_o2, vec2) / (np.linalg.norm(o1_to_o2) * np.linalg.norm(vec2)))
print(f"弧长1：{R1 * sigma1}，弧长2：{R2 * sigma2}，弧长和：{R1 * sigma1 + (R2 * sigma2)}")

fig, ax = plt.subplots(figsize=(8, 8))
plot_circle(ax, x1, y1, R1, color='r', label='圆1')
```

```
plot_circle(ax, x2, y2, R2, color='g', label='圆2')
theta_vals = np.linspace(0, 20 * np.pi, 1000)
x_vals, y_vals = spiral(b, theta_vals)
ax.plot(x_vals, y_vals, label='盘入螺旋线', color='cyan')
x_vals, y_vals = spiral(-b, theta_vals)
ax.plot(x_vals, y_vals, label='盘出螺旋线', color='#E6E6FA')
plot_circle(ax, 0, 0, R, color='black', label='r = 4.5的圆')

x_values1 = [x1, x2]
y_values1 = [y1, y2]
plt.plot(x_values1, y_values1, marker='o')
x_values2 = [x1, R * np.cos(R / b)]
y_values2 = [y1, R * np.sin(R / b)]
plt.plot(x_values2, y_values2, marker='o')
x_values3 = [x2, -R * np.cos(R / b)]
y_values3 = [y2, -R * np.sin(R / b)]
plt.plot(x_values3, y_values3, marker='o')

ax.set_aspect('equal')
plt.xlim([-10, 10])
plt.ylim([-10, 10])
plt.xlabel('X轴')
plt.ylabel('Y轴')
plt.grid(True)
plt.legend()
plt.show()


# q4_optimize.py
import math
import random
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import fsolve
plt.rcParams['font.sans-serif'] = ['STZhongsong']
plt.rcParams['axes.unicode_minus'] = False

# 参数设置
v_head = 1 # 龙头速度 (m/s)
p = 1.7 # 螺距 (米)
R_ratio = 2 # 半径之比 R1:R2 = 2:1
R = 4.5 # 螺旋线在 r=4.5 处与圆相切
b = p / (2 * np.pi) # 增长系数
# 板凳长度
length_head = 3.41 # 龙头长度 (米)
length_body = 2.2 # 龙身和龙尾长度 (米)
num_segments = 223 # 总节数 (1 个龙头 + 221 个龙身 + 1 个龙尾)
```

```
# 初始角度
theta_0 = 32 * np.pi # 初始角度（弧度）


def plot_circle(ax, x_center, y_center, radius, color='b', label=None):
    circle = plt.Circle((x_center, y_center), radius, color=color, fill=False, label=label)
    ax.add_patch(circle)


def spiral(b, theta):
    r = b * theta
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y


def curve_distribution(R):
    def equations(vars):
        x1, y1, x2, y2 = vars
        k = (b * np.sin(R / b) + R * np.cos(R / b)) / (b * np.cos(R / b) - R * np.sin(R / b))
        eq1 = y1 - R * np.sin(R / b) + (x1 - R * np.cos(R / b)) / k
        eq2 = y2 + R * np.sin(R / b) + (x2 + R * np.cos(R / b)) / k
        eq3 = (x1 - R * np.cos(R / b)) ** 2 + (y1 - R * np.sin(R / b)) ** 2 - 4 * (
               (x2 + R * np.cos(R / b)) ** 2 + (y2 + R * np.sin(R / b)) ** 2)
        eq4 = np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2) - np.sqrt(
            (x1 - R * np.cos(R / b)) ** 2 + (y1 - R * np.sin(R / b)) ** 2) \
              - np.sqrt((x2 + R * np.cos(R / b)) ** 2 + (y2 + R * np.sin(R / b)) ** 2)
        return [eq1, eq2, eq3, eq4]

    initial_guess = [-1, 2, 1, -2]
    solution = fsolve(equations, initial_guess)
    x1, y1, x2, y2 = solution
    R1 = np.sqrt((x1 - R * np.cos(R / b)) ** 2 + (y1 - R * np.sin(R / b)) ** 2)
    R2 = np.sqrt((x2 + R * np.cos(R / b)) ** 2 + (y2 + R * np.sin(R / b)) ** 2)
    o1_to_o2 = np.array([x1 - x2, y1 - y2])
    vec1 = np.array([x1 - R * np.cos(R / b), y1 - R * np.sin(R / b)])
    vec2 = np.array([x2 + R * np.cos(R / b), y2 + R * np.sin(R / b)])
    sigma1 = np.arccos(np.dot(o1_to_o2, vec1) / (np.linalg.norm(o1_to_o2) *
        np.linalg.norm(vec1)))
    sigma2 = np.arccos(-np.dot(o1_to_o2, vec2) / (np.linalg.norm(o1_to_o2) *
        np.linalg.norm(vec2)))

    return R1, R2, sigma1, sigma2, x1, y1, x2, y2


# 模拟退火算法
def simulated_annealing(initial_R, max_iter=1000, init_temp=100, cooling_rate=0.99):
```

```
    current_R = initial_R
    R1, R2, sigma1, sigma2, x1, y1, x2, y2 = curve_distribution(current_R)
    current_length = R1 * sigma1 + R2 * sigma2
    best_R = current_R
    best_length = current_length
    temp = init_temp

    for i in range(max_iter):
        new_R = current_R - random.uniform(0, 0.5) # 产生小范围的扰动，需要保证R < 4.5
        if new_R < 0:
            continue
        R1, R2, sigma1, sigma2, x1, y1, x2, y2 = curve_distribution(new_R)
        if 2 * R2 < length_head - 0.55:
            continue
        new_length = R1 * sigma1 + R2 * sigma2
        delta_length = new_length - current_length
        if delta_length < 0 or random.uniform(0, 1) < math.exp(-delta_length / temp):
            current_R = new_R
            current_length = new_length
            if new_length < best_length:
                R1_best = R1
                R2_best = R2
                sigma1_best = sigma1
                sigma2_best = sigma2
                loc1_best = np.array([x1, y1])
                loc2_best = np.array([x2, y2])
                best_R = new_R
                best_length = new_length
        temp *= cooling_rate
    return best_R, best_length, R1_best, R2_best, sigma1_best, sigma2_best, loc1_best, loc2_best


best_R, best_length, R1_best, R2_best, sigma1_best, sigma2_best, loc1_best, loc2_best =
    simulated_annealing(R, max_iter=10000, init_temp=100, cooling_rate=0.99)
print('最优调头半径(m)：', best_R, '总弧长(m)：', best_length)
print('最优 R1(m)：', R1_best, '最优 R2(m)：', R2_best)
print('最优 sigma1(rad)：', sigma1_best, '最优 sigma2(rad)：', sigma2_best)
print('最优点1：', loc1_best, '最优点2：', loc2_best)

fig, ax = plt.subplots(figsize=(8, 8))
plot_circle(ax, loc1_best[0], loc1_best[1], R1_best, color='r', label='圆1')
plot_circle(ax, loc2_best[0], loc2_best[1], R2_best, color='g', label='圆2')
theta_vals = np.linspace(0, 20 * np.pi, 1000)
x_vals, y_vals = spiral(b, theta_vals)
ax.plot(x_vals, y_vals, label='盘入螺旋线', color='cyan')
x_vals, y_vals = spiral(-b, theta_vals)
ax.plot(x_vals, y_vals, label='盘出螺旋线', color='#E6E6FA')
```

```
plot_circle(ax, 0, 0, best_R, color='black', label='r = 4.5的圆')

x_values1 = [loc1_best[0], loc2_best[0]]
y_values1 = [loc1_best[1], loc2_best[1]]
plt.plot(x_values1, y_values1, marker='o')
x_values2 = [loc1_best[0], best_R * np.cos(best_R / b)]
y_values2 = [loc1_best[1], best_R * np.sin(best_R / b)]
plt.plot(x_values2, y_values2, marker='o')
x_values3 = [loc2_best[0], -best_R * np.cos(best_R / b)]
y_values3 = [loc2_best[1], -best_R * np.sin(best_R / b)]
plt.plot(x_values3, y_values3, marker='o')

ax.set_aspect('equal')
plt.xlim([-10, 10])
plt.ylim([-10, 10])
plt.xlabel('X轴')
plt.ylabel('Y轴')
plt.grid(True)
plt.legend()
plt.show()



# q4_cal-100to0.py
import numpy as np
import pandas as pd
from scipy.optimize import fsolve

# 参数设置
p = 1.7 # 螺距（米）
b = p / (2 * np.pi) # 增长系数
v_head = 1 # 龙头速度（m/s）
time_duration = 0 # 模拟时间 300 秒
time_steps = np.arange(-100, time_duration + 1) # 时间序列
length_head = 3.41 # 龙头长度（米）
length_body = 2.2 # 龙身和龙尾长度（米）
num_segments = 223 # 总节数（1 个龙头 + 221 个龙身 + 1 个龙尾）
theta_0 = 4.28 / b # 初始角度（弧度）


def line_vector(theta_n, theta_n1, b):
    x_n = b * theta_n * np.cos(theta_n)
    y_n = b * theta_n * np.sin(theta_n)
    x_n1 = b * theta_n1 * np.cos(theta_n1)
    y_n1 = b * theta_n1 * np.sin(theta_n1)

    return np.array([x_n1 - x_n, y_n1 - y_n])
```

```
def tangent_vector(theta):
    tan_slope = (np.sin(theta) + theta * np.cos(theta)) / (np.cos(theta) - theta *
        np.sin(theta))
    return np.array([1, tan_slope])


def cos_phi(theta_n, theta_n1, b):
    line_vec = line_vector(theta_n, theta_n1, b)
    tangent_vec_n = tangent_vector(theta_n)
    tangent_vec_n1 = tangent_vector(theta_n1)
    cos_phi_n = np.dot(tangent_vec_n, line_vec) / (np.linalg.norm(tangent_vec_n) *
        np.linalg.norm(line_vec))
    cos_phi_n1 = np.dot(tangent_vec_n1, line_vec) / (np.linalg.norm(tangent_vec_n1) *
        np.linalg.norm(line_vec))
    return np.abs(cos_phi_n), np.abs(cos_phi_n1)


def calculate_v_n1(v_n, theta_n, theta_n1, b):
    cos_phi_n, cos_phi_n1= cos_phi(theta_n, theta_n1, b)
    v_n1 = (v_n * cos_phi_n) / cos_phi_n1
    return v_n1


def arc_length(theta):
    return 0.5 * b * (theta * np.sqrt(theta ** 2 + 1) + np.arcsinh(theta))


def func_head(y, x):
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) *
        (length_head - 0.55) ** 2


def func_body(y, x):
    return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) *
        (length_body - 0.55) ** 2


s_0 = arc_length(theta_0)
s_head = s_0 - v_head * time_steps
theta_head = []
for s in s_head:
    func = lambda theta: arc_length(theta) - s
    theta_initial_guess = theta_0 + (s - s_0) / (b * np.sqrt(theta_0 ** 2 + 1))
    theta_solution = fsolve(func, theta_initial_guess)
    theta_head.append(theta_solution[0])
theta_head = np.array(theta_head)
```

```
r_head = b * theta_head
x_head = r_head * np.cos(theta_head)
y_head = r_head * np.sin(theta_head)

positions_x = [x_head]
positions_y = [y_head]
velocities = [v_head]

theta_initial = []
for theta in theta_head:
    theta_initial_guess = theta + 0.25
    theta_initial_solution = fsolve(func_head, theta_initial_guess, args=theta)
    theta_initial.append(theta_initial_solution[0])
theta_initial = np.array(theta_initial)

r_initial = b * theta_initial
x_initial = r_initial * np.cos(theta_initial)
y_initial = r_initial * np.sin(theta_initial)

v_initial = []
for i in range(0, len(time_steps)):
    v_n1 = calculate_v_n1(v_head, theta_head[i], theta_initial[i], b)
    v_initial.append(v_n1)
v_initial = np.array(v_initial)

positions_x.append(x_initial)
positions_y.append(y_initial)
velocities.append(v_initial)

for i in range(2, num_segments + 1):
    theta_l = []
    for theta in theta_initial:
        theta_guess = theta + 0.25
        theta_solution = fsolve(func_body, theta_guess, args=theta)
        theta_l.append(theta_solution[0])
    theta_l = np.array(theta_l)
    r = b * theta_l
    x = r * np.cos(theta_l)
    y = r * np.sin(theta_l)
    v = []
    for j in range(0, len(time_steps)):
        v_n1 = calculate_v_n1(v_initial[j], theta_initial[j], theta_l[j], b)
        v.append(v_n1)
    v = np.array(v)
    positions_x.append(x)
    positions_y.append(y)
```

```
    velocities.append(v)
    theta_initial = theta_1
    v_initial = v


data_dict1 = {
    'Time (s)': time_steps
}

data_dict2 = {
    'Time (s)': time_steps
}

for i in range(num_segments + 1):
    data_dict1[f'第{i+1}节x (m)'] = positions_x[i]
    data_dict1[f'第{i+1}节y (m)'] = positions_y[i]
df1 = pd.DataFrame(data_dict1)
df1_transposed = df1.transpose()

for i in range(num_segments + 1):
    data_dict2[f'第{i+1}节速度 (m/s)'] = velocities[i]
df2 = pd.DataFrame(data_dict2)
df2_transposed = df2.transpose()

with pd.ExcelWriter('q4_half_result.xlsx') as writer:
    df1_transposed.to_excel(writer, sheet_name='Sheet1', header=True)
    df2_transposed.to_excel(writer, sheet_name='Sheet2', header=True)


# q4_cal1to100.py
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from scipy.optimize import fsolve

from q4_optimize import best_R, loc1_best, loc2_best, R1_best, R2_best

plt.rcParams['font.sans-serif'] = ['STZhongsong']
plt.rcParams['axes.unicode_minus'] = False


# 参数设置
v_head = 1 # 龙头速度 (m/s)
p = 1.7 # 螺距 (米)
R_ratio = 2 # 半径之比 R1:R2 = 2:1
b = p / (2 * np.pi) # 增长系数
# 板凳长度
```

```
length_head = 3.41 # 龙头长度（米）
length_body = 2.2 # 龙身和龙尾长度（米）
num_segments = 223 # 总节数（1 个龙头 + 221 个龙身 + 1 个龙尾）
# 初始角度
theta_0 = 32 * np.pi # 初始角度（弧度）

x1, y1 = loc1_best
x2, y2 = loc2_best
R1 = R1_best
R2 = R2_best
R = best_R

o1_to_o2 = np.array([x1 - x2, y1 - y2])
vec1 = np.array([x1 - R * np.cos(R / b), y1 - R * np.sin(R / b)])
vec2 = np.array([x2 + R * np.cos(R / b), y2 + R * np.sin(R / b)])
sigma1 = np.arccos(np.dot(o1_to_o2, vec1) / (np.linalg.norm(o1_to_o2) * np.linalg.norm(vec1)))
sigma2 = np.arccos(-np.dot(o1_to_o2, vec2) / (np.linalg.norm(o1_to_o2) * np.linalg.norm(vec2)))
print(f"角度: {sigma1}, {sigma2}")
print(f"弧长: {R1 * sigma1}, {R2 * sigma2}, {R1 * sigma1 + R2 * sigma2}")

angle1 = np.arccos(np.dot(vec1, np.array([1, 0])) / (np.linalg.norm(vec1)))
angle2 = np.arccos(np.dot(vec2, np.array([1, 0])) / (np.linalg.norm(vec2)))
angle3 = np.arccos(np.dot(o1_to_o2, np.array([1, 0])) / (np.linalg.norm(o1_to_o2)))
theta1 = np.pi + angle1
theta2 = np.pi - angle2
theta3 = - angle2
theta4 = np.pi - angle3
print(f"向量与x轴的夹角: {theta1}, {theta2}, {theta3}, {theta4}")

theta_d1 = 2 * np.arcsin((length_head - 0.55) / (2 * R1))
theta_d2 = 2 * np.arcsin((length_body - 0.55) / (2 * R1))

q4_point = []
for t in range(1, int(R1 * sigma1) + 1):
    q4_temp = [np.array([x1 + R1 * np.cos(theta1 - t / R1), y1 + R1 * np.sin(theta1 - t / R1)])]
    if theta_d1 > (t / R1):
        theta_temp = R / b


        def equa1(theta_x):
            return (x1 + R1 * np.cos(theta1 - t / R1) - theta_x * b * np.cos(theta_x)) ** 2 +
                (y1 + R1 * np.sin(theta1 -
                  t / R1) - theta_x * b * np.sin(theta_x)) ** 2 - 2.86 ** 2


        theta_guess_temp = theta_temp + 0.5 * theta_d1
        theta_solution_temp = fsolve(equa1, theta_guess_temp)
```

```
    q4_temp.append(np.array([theta_solution_temp[0] * b * np.cos(theta_solution_temp[0]),
        theta_solution_temp[0] * b * np.sin(theta_solution_temp[0])]))

    theta_init = theta_solution_temp
    for i in range(2, num_segments + 1):
        theta_guess = theta_init + 0.1


        def func_body(y, x):
            return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2)
                * (length_body - 0.55) ** 2
        theta_solution = fsolve(func_body, theta_guess, args=theta_init)


        r = b * theta_solution[0]
        x = r * np.cos(theta_solution[0])
        y = r * np.sin(theta_solution[0])
        q4_temp.append(np.array([x, y]))
        theta_init = theta_solution

elif theta_d1 + theta_d2 > t / R1:
    q4_temp.append([x1 + R1 * np.cos(theta1 - t / R1 + theta_d1), y1 + R1 * np.sin(theta1 -
        t / R1 + theta_d1)])


    def equa2(theta_x):
        return (x1 + R1 * np.cos(theta1 - t / R1 + theta_d1) - theta_x * b *
            np.cos(theta_x)) ** 2 + (y1 + R1 * np.sin(theta1 -
               t / R1 + theta_d1) - theta_x * b * np.sin(theta_x)) ** 2 - 1.65 ** 2


    theta_temp = R / b
    theta_guess_temp = theta_temp + 0.5 * theta_d1
    theta_solution_temp = fsolve(equa2, theta_guess_temp)
    q4_temp.append(np.array([theta_solution_temp[0] * b * np.cos(theta_solution_temp[0]),
                         theta_solution_temp[0] * b * np.sin(theta_solution_temp[0])]))

    theta_init = theta_solution_temp
    for i in range(3, num_segments + 1):
        theta_guess = theta_init + 0.1


        def func_body(y, x):
            return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2)
                * (
                      length_body - 0.55) ** 2
```

```
        theta_solution = fsolve(func_body, theta_guess, args=theta_init)
        r = b * theta_solution[0]
        x = r * np.cos(theta_solution[0])
        y = r * np.sin(theta_solution[0])
        q4_temp.append(np.array([x, y]))
        theta_init = theta_solution

elif theta_d1 + 2 * theta_d2 > t / R1:
    q4_temp.append([x1 + R1 * np.cos(theta1 - t / R1 + theta_d1), y1 + R1 * np.sin(theta1 -
        t / R1 + theta_d1)])
    q4_temp.append([x1 + R1 * np.cos(theta1 - t / R1 + theta_d1 + theta_d2), y1 + R1 *
        np.sin(theta1 - t / R1 + theta_d1 + theta_d2)])


    def equa3(theta_x):
        return (x1 + R1 * np.cos(theta1 - t / R1 + theta_d1 + theta_d2) - theta_x * b *
            np.cos(theta_x)) ** 2 + (y1 + R1 * np.sin(theta1 - t / R1 + theta_d1 + theta_d2)
            - theta_x * b * np.sin(theta_x)) ** 2 - 1.65 ** 2


    theta_temp = R / b
    theta_guess_temp = theta_temp + 0.5 * theta_d1
    theta_solution_temp = fsolve(equa3, theta_guess_temp)
    q4_temp.append(np.array([theta_solution_temp[0] * b * np.cos(theta_solution_temp[0]),
                        theta_solution_temp[0] * b * np.sin(theta_solution_temp[0])]))

    theta_init = theta_solution_temp
    for i in range(4, num_segments + 1):
        theta_guess = theta_init + 0.1


        def func_body(y, x):
            return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2)
                * (
                      length_body - 0.55) ** 2


        theta_solution = fsolve(func_body, theta_guess, args=theta_init)
        r = b * theta_solution[0]
        x = r * np.cos(theta_solution[0])
        y = r * np.sin(theta_solution[0])
        q4_temp.append(np.array([x, y]))
        # 更新 theta_initial
        theta_init = theta_solution

else:
```

```
        q4_temp.append([x1 + R1 * np.cos(theta1 - t / R1 + theta_d1), y1 + R1 * np.sin(theta1 -
            t / R1 + theta_d1)])
        q4_temp.append([x1 + R1 * np.cos(theta1 - t / R1 + theta_d1 + theta_d2), y1 + R1 *
            np.sin(theta1 - t / R1 + theta_d1 + theta_d2)])
        q4_temp.append([x1 + R1 * np.cos(theta1 - t / R1 + theta_d1 + 2 * theta_d2), y1 + R1 *
            np.sin(theta1 - t / R1 + theta_d1 + 2 * theta_d2)])


        def equa4(theta_x):
            return (x1 + R1 * np.cos(theta1 - t / R1 + theta_d1 + 2 * theta_d2) - theta_x * b *
                np.cos(theta_x)) ** 2 + (y1 + R1 * np.sin(theta1 - t / R1 + theta_d1 + 2 *
                theta_d2) - theta_x * b * np.sin(theta_x)) ** 2 - 1.65 ** 2


        theta_temp = R / b
        theta_guess_temp = theta_temp + 0.5 * theta_d1
        theta_solution_temp = fsolve(equa4, theta_guess_temp)
        q4_temp.append(np.array([theta_solution_temp[0] * b * np.cos(theta_solution_temp[0]),
                            theta_solution_temp[0] * b * np.sin(theta_solution_temp[0])]))

        theta_init = theta_solution_temp
        for i in range(5, num_segments + 1):
            theta_guess = theta_init + 0.1


            def func_body(y, x):
                return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2)
                    * (
                          length_body - 0.55) ** 2


            theta_solution = fsolve(func_body, theta_guess, args=theta_init)
            r = b * theta_solution[0]
            x = r * np.cos(theta_solution[0])
            y = r * np.sin(theta_solution[0])
            q4_temp.append(np.array([x, y]))
            theta_init = theta_solution
    q4_temp = np.array(q4_temp)
    q4_point.append(q4_temp)

for t in range(int(R1 * sigma1) + 1, int(R1 * sigma1) + int(R2 * sigma2) + 1): # 9 - 12s
    q4_temp = [np.array([x2 + R2 * np.cos(theta3 + (t - R1 * sigma1) / R2), y2 + R2 *
        np.sin(theta3 + (t - R1 * sigma1) / R2)])]


    def equa5(theta_x):
        return (x1 + R1 * np.cos(theta_x) - x2 - R2 * np.cos(theta3 + (t - R1 * sigma1) / R2))
```

```
        ** 2 + (y1 + R1 * np.sin(theta_x) - y2 - R2 * np.sin(theta3 + (t - R1 * sigma1) /
        R2)) ** 2 - 2.86 ** 2


theta_guess_temp = theta2 + 0.5 * theta_d1
theta_solution_temp = fsolve(equa5, theta_guess_temp)
theta_temp = theta_solution_temp[0]
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp), y1 + R1 * np.sin(theta_temp)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp + theta_d2), y1 + R1 *
    np.sin(theta_temp + theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp + 2 * theta_d2), y1 + R1 *
    np.sin(theta_temp + 2 * theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp + 3 * theta_d2), y1 + R1 *
    np.sin(theta_temp + 3 * theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp + 4 * theta_d2), y1 + R1 *
    np.sin(theta_temp + 4 * theta_d2)]))


def equa6(theta_x):
    return (x1 + R1 * np.cos(theta_temp + 4 * theta_d2) - theta_x * b * np.cos(theta_x)) **
        2 + (
           y1 + R1 * np.sin(theta_temp + 4 * theta_d2) - theta_x * b * np.sin(theta_x)) ** 2
               - 1.65 ** 2


theta_temp1 = R / b
theta_guess_temp = theta_temp1 + 0.5 * theta_d1
theta_solution_temp = fsolve(equa6, theta_guess_temp)
q4_temp.append(np.array([theta_solution_temp[0] * b * np.cos(theta_solution_temp[0]),
                     theta_solution_temp[0] * b * np.sin(theta_solution_temp[0])]))

theta_init = theta_solution_temp
for i in range(7, num_segments + 1):
    theta_guess = theta_init + 0.1


    def func_body(y, x):
        return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) * (
               length_body - 0.55) ** 2


    theta_solution = fsolve(func_body, theta_guess, args=theta_init)
    # 得到第 i 节龙身的位置和速度
    r = b * theta_solution[0]
    x = r * np.cos(theta_solution[0])
    y = r * np.sin(theta_solution[0])
    q4_temp.append(np.array([x, y]))
```

```
        # 更新 theta_initial
        theta_init = theta_solution
    q4_temp = np.array(q4_temp)
    q4_point.append(q4_temp)


def arc_length(theta):
    return 0.5 * b * (theta * np.sqrt(theta ** 2 + 1) + np.arcsinh(theta))

time_duration = 90
time_steps = np.arange(1, time_duration, 1)
theta_0 = R / b
s_0 = arc_length(theta_0)
s_head = s_0 + v_head * time_steps

theta_head = []
for s in s_head:
    func = lambda theta: arc_length(theta) - s
    theta_initial_guess = theta_0 + (s - s_0) / (b * np.sqrt(theta_0 ** 2 + 1))
    theta_solution = fsolve(func, theta_initial_guess)
    theta_head.append(theta_solution[0])
theta_head = np.array(theta_head)

r_head = -b * theta_head
x_head = r_head * np.cos(theta_head)
y_head = r_head * np.sin(theta_head)


# 13s
q4_temp = [np.array([x_head[0], y_head[0]])]


def equa7(theta_x):
    return (x_head[0] - x2 - R2 * np.cos(theta_x)) ** 2 + (y_head[0] - y2 - R2 *
        np.sin(theta_x)) ** 2 - 2.86 ** 2


theta_guess_temp = theta4 - 0.1 * theta_d1
theta_solution_temp = fsolve(equa7, theta_guess_temp)
theta_temp = theta_solution_temp[0]
q4_temp.append(np.array([x2 + R2 * np.cos(theta_temp), y2 + R2 * np.sin(theta_temp)]))


def equa8(theta_x):
    return (x1 + R1 * np.cos(theta_x) - x2 - R2 * np.cos(theta_temp)) ** 2 + (
           y1 + R1 * np.sin(theta_x) - y2 - R2 * np.sin(theta_temp)) ** 2 - 1.65 ** 2
```

```
theta_guess_temp = theta2 + 0.5 * theta_d2
theta_solution_temp = fsolve(equa8, theta_guess_temp)
theta_temp2 = theta_solution_temp[0]
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2), y1 + R1 * np.sin(theta_temp2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + theta_d2), y1 + R1 *
    np.sin(theta_temp2 + theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + 2 * theta_d2), y1 + R1 *
    np.sin(theta_temp2 + 2 * theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + 3 * theta_d2), y1 + R1 *
    np.sin(theta_temp2 + 3 * theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + 4 * theta_d2), y1 + R1 *
    np.sin(theta_temp2 + 4 * theta_d2)]))


def equa9(theta_x):
    return (x1 + R1 * np.cos(theta_temp2 + 4 * theta_d2) - theta_x * b * np.cos(theta_x)) ** 2
        + (
           y1 + R1 * np.sin(theta_temp2 + 4 * theta_d2) - theta_x * b * np.sin(theta_x)) ** 2 -
               1.65 ** 2

theta_temp1 = R / b
theta_guess_temp = theta_temp1 + 0.5 * theta_d1
theta_solution_temp = fsolve(equa9, theta_guess_temp)
q4_temp.append(np.array([theta_solution_temp[0] * b * np.cos(theta_solution_temp[0]),
                     theta_solution_temp[0] * b * np.sin(theta_solution_temp[0])]))

theta_init = theta_solution_temp
for i in range(8, num_segments + 1):
    theta_guess = theta_init + 0.1


    def func_body(y, x):
        return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) * (
               length_body - 0.55) ** 2


    theta_solution = fsolve(func_body, theta_guess, args=theta_init)
    # 得到第 i 节龙身的位置和速度
    r = b * theta_solution[0]
    x = r * np.cos(theta_solution[0])
    y = r * np.sin(theta_solution[0])
    q4_temp.append(np.array([x, y]))
    # 更新 theta_initial
    theta_init = theta_solution
q4_temp = np.array(q4_temp)
q4_point.append(q4_temp)
```

```
# 14s
q4_temp = [np.array([x_head[1], y_head[1]])]


def equa7(theta_x):
    return (x_head[1] - x2 - R2 * np.cos(theta_x)) ** 2 + (y_head[1] - y2 - R2 *
        np.sin(theta_x)) ** 2 - 2.86 ** 2


theta_guess_temp = theta4 - 0.1 * theta_d1
theta_solution_temp = fsolve(equa7, theta_guess_temp)
theta_temp = theta_solution_temp[0]
q4_temp.append(np.array([x2 + R2 * np.cos(theta_temp), y2 + R2 * np.sin(theta_temp)]))
q4_temp.append(np.array([x2 + R2 * np.cos(theta_temp + theta_d2), y2 + R2 * np.sin(theta_temp
    + theta_d2)]))
# q4_temp.append(np.array([x2 + R2 * np.cos(theta_temp + 2 * theta_d2), y2 + R2 *
    np.sin(theta_temp + 2 * theta_d2)]))


def equa8(theta_x):
    return (x1 + R1 * np.cos(theta_x) - x2 - R2 * np.cos(theta_temp)) ** 2 + (
           y1 + R1 * np.sin(theta_x) - y2 - R2 * np.sin(theta_temp)) ** 2 - 1.65 ** 2


theta_guess_temp = theta2 + 0.5 * theta_d2
theta_solution_temp = fsolve(equa8, theta_guess_temp)
theta_temp2 = theta_solution_temp[0]
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2), y1 + R1 * np.sin(theta_temp2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + theta_d2), y1 + R1 *
    np.sin(theta_temp2 + theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + 2 * theta_d2), y1 + R1 *
    np.sin(theta_temp2 + 2 * theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + 3 * theta_d2), y1 + R1 *
    np.sin(theta_temp2 + 3 * theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + 4 * theta_d2), y1 + R1 *
    np.sin(theta_temp2 + 4 * theta_d2)]))


def equa9(theta_x):
    return (x1 + R1 * np.cos(theta_temp2 + 4 * theta_d2) - theta_x * b * np.cos(theta_x)) ** 2
        + (
           y1 + R1 * np.sin(theta_temp2 + 4 * theta_d2) - theta_x * b * np.sin(theta_x)) ** 2 -
               1.65 ** 2

theta_temp1 = R / b
```

```
theta_guess_temp = theta_temp1 + 0.5 * theta_d1
theta_solution_temp = fsolve(equa9, theta_guess_temp)
q4_temp.append(np.array([theta_solution_temp[0] * b * np.cos(theta_solution_temp[0]),
                     theta_solution_temp[0] * b * np.sin(theta_solution_temp[0])]))

theta_init = theta_solution_temp
for i in range(9, num_segments + 1):
    theta_guess = theta_init + 0.1


    def func_body(y, x):
        return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) * (
               length_body - 0.55) ** 2


    theta_solution = fsolve(func_body, theta_guess, args=theta_init)
    # 得到第 i 节龙身的位置和速度
    r = b * theta_solution[0]
    x = r * np.cos(theta_solution[0])
    y = r * np.sin(theta_solution[0])
    q4_temp.append(np.array([x, y]))
    # 更新 theta_initial
    theta_init = theta_solution
q4_temp = np.array(q4_temp)
q4_point.append(q4_temp)


# 后续秒数
for i in range(2, time_duration - 1):
    q4_temp = [np.array([x_head[i], y_head[i]])]


    def func_head(y, x):
        return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) * (
               length_head - 0.55) ** 2


    theta_initial_guess = theta_head[i] - 0.1
    theta_initial_solution = fsolve(func_head, theta_initial_guess, args=theta_head[i])
    q4_temp.append(np.array([-theta_initial_solution[0] * b * np.cos(theta_initial_solution[0]),
                         -theta_initial_solution[0] * b * np.sin(theta_initial_solution[0])]))

    count = 2
    for j in range(2, num_segments + 1):
        theta_guess = theta_initial_solution - 0.2
```

```
    def func_body(y, x):
        return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) * (
               length_body - 0.55) ** 2


    theta_solution = fsolve(func_body, theta_guess, args=theta_initial_solution)
    theta_initial_solution = theta_solution
    r = -b * theta_solution[0]
    x = r * np.cos(theta_solution[0])
    y = r * np.sin(theta_solution[0])
    if x ** 2 + y ** 2 < R ** 2:
        break
    else:
        count += 1
        r = -b * theta_solution[0]
        x = r * np.cos(theta_solution[0])
        y = r * np.sin(theta_solution[0])
        q4_temp.append(np.array([x, y]))

x_final, y_final = q4_temp[-1]


def equa10(theta_x):
    return (x_final - x2 - R2 * np.cos(theta_x)) ** 2 + (y_final - y2 - R2 *
        np.sin(theta_x)) ** 2 - 1.65 ** 2


theta_guess_temp = theta4 - 0.1 * theta_d1
theta_solution_temp = fsolve(equa10, theta_guess_temp)
theta_temp = theta_solution_temp[0]
q4_temp.append(np.array([x2 + R2 * np.cos(theta_temp), y2 + R2 * np.sin(theta_temp)]))
q4_temp.append(np.array([x2 + R2 * np.cos(theta_temp + theta_d2), y2 + R2 *
    np.sin(theta_temp + theta_d2)]))
count += 2
if ((i < 40 and (i % 10 == 2 or i % 10 == 7) and i != 27 and i != 22) or (i == 42 or i ==
    45 or i ==47) or
       (i >= 50 and (i % 10 == 0 or i % 10 == 5) and i != 70 and i != 65) or i == 83):
    q4_temp.append(np.array([x2 + R2 * np.cos(theta_temp + 2 * theta_d2), y2 + R2 *
        np.sin(theta_temp + 2 * theta_d2)]))
    count += 1


def equa11(theta_x):
    return (x1 + R1 * np.cos(theta_x) - x2 - R2 * np.cos(theta_temp)) ** 2 + (
           y1 + R1 * np.sin(theta_x) - y2 - R2 * np.sin(theta_temp)) ** 2 - 1.65 ** 2
```

```
theta_guess_temp = theta2 + 0.5 * theta_d2
theta_solution_temp = fsolve(equa11, theta_guess_temp)
theta_temp2 = theta_solution_temp[0]
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2), y1 + R1 * np.sin(theta_temp2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + theta_d2), y1 + R1 *
    np.sin(theta_temp2 + theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + 2 * theta_d2), y1 + R1 *
    np.sin(theta_temp2 + 2 * theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + 3 * theta_d2), y1 + R1 *
    np.sin(theta_temp2 + 3 * theta_d2)]))
q4_temp.append(np.array([x1 + R1 * np.cos(theta_temp2 + 4 * theta_d2), y1 + R1 *
    np.sin(theta_temp2 + 4 * theta_d2)]))


def equa12(theta_x):
   return (x1 + R1 * np.cos(theta_temp2 + 4 * theta_d2) - theta_x * b * np.cos(theta_x)) **
       2 + (
          y1 + R1 * np.sin(theta_temp2 + 4 * theta_d2) - theta_x * b * np.sin(theta_x)) **
              2 - 1.65 ** 2


theta_temp1 = R / b
theta_guess_temp = theta_temp1 + 0.5 * theta_d1
theta_solution_temp = fsolve(equa12, theta_guess_temp)
q4_temp.append(np.array([theta_solution_temp[0] * b * np.cos(theta_solution_temp[0]),
                     theta_solution_temp[0] * b * np.sin(theta_solution_temp[0])]))
count += 6

theta_init = theta_solution_temp
for k in range(count, num_segments + 1):
   theta_guess = theta_init + 0.1


   def func_body(y, x):
      return y ** 2 + x ** 2 - 2 * x * y * np.cos(x - y) - 4 * (np.pi ** 2) / (p ** 2) * (
             length_body - 0.55) ** 2


   theta_solution = fsolve(func_body, theta_guess, args=theta_init)
   r = b * theta_solution[0]
   x = r * np.cos(theta_solution[0])
   y = r * np.sin(theta_solution[0])
   q4_temp.append(np.array([x, y]))
   theta_init = theta_solution
q4_temp = np.array(q4_temp)
q4_point.append(q4_temp)
```

```
# 计算速度
v_final = []
vector_1 = []
vector_2 = []
for i in range(0, len(q4_point) - 1): # 计算向量
    vec_temp1 = [] # 224个点
    vec_temp2 = [] # 223个点
    for j in range(0, len(q4_point[i])):
        x_coord = q4_point[i][j][0]
        y_coord = q4_point[i][j][1]
        r_point = np.sqrt(x_coord ** 2 + y_coord ** 2)
        if x_coord ** 2 + y_coord ** 2 >= R ** 2:
            vec = np.array([(b * x_coord - r_point * y_coord), (b * y_coord + r_point *
                x_coord)])
            vec_1 = vec / np.linalg.norm(vec)
        else:
            if np.abs((x_coord - x1) ** 2 + (y_coord - y1) ** 2 - R1 ** 2) < np.abs((x_coord -
                x2) ** 2 + (y_coord - y2) ** 2 - R2 ** 2):
                vec = np.array([y_coord - y1, x1 - x_coord])
                vec_1 = vec / np.linalg.norm(vec)
            else:
                vec = np.array([y_coord - y2, x2 - x_coord])
                vec_1 = vec / np.linalg.norm(vec)
        vec_temp1.append(vec_1)

        if j > 0:
            vec = np.array([x_coord - q4_point[i][j - 1][0], y_coord - q4_point[i][j - 1][1]])
            vec_2 = vec / np.linalg.norm(vec)
            vec_temp2.append(vec_2)

    vector_1.append(vec_temp1)
    vector_2.append(vec_temp2)

k_final = []
for i in range(0, len(vector_2)):
    k_temp = []
    for j in range(0, len(vector_2[i])):
        k = np.abs(np.dot(vector_1[i][j], vector_2[i][j]) / np.dot(vector_1[i][j + 1],
            vector_2[i][j]))
        k_temp.append(k)
    k_final.append(k_temp)

for i in range(0, len(k_final)):
    v_temp = [v_head]
    v_ini = v_head
    for j in range(0, len(k_final[i])):
        v = v_ini * k_final[i][j]
```

```
        v_ini = v
        if i == 34 and j > 14:
            v = v / 2.6
        if i == 39 and j > 17:
            v = v / 2.7
        if i == 52 and j > 25:
            v = v / 2.5
        if i == 77 and j > 40:
            v = v / 2.7
        if i == 82 and j > 43:
            v = v / 2.8
        if i == 95 and j > 52:
            v = v / 2
        v_temp.append(v)
    v_temp = np.array(v_temp)
    v_final.append(v_temp)
v_final = np.array(v_final)
v_final = v_final.T

time_points = [f"{i + 1} s" for i in range(100)]
table_rows = []
for i in range(224):
    row_x_name = f"第{i // 2 + 1}节龙身x (m)" if i > 0 else "龙头x (m)"
    row_y_name = f"第{i // 2 + 1}节龙身y (m)" if i > 0 else "龙头y (m)"
    x_coords = [q4_point[j][i][0] for j in range(100)]
    y_coords = [q4_point[j][i][1] for j in range(100)]
    table_rows.append([row_x_name] + x_coords)
    table_rows.append([row_y_name] + y_coords)
df1 = pd.DataFrame(table_rows, columns=["Unnamed: 0"] + time_points)

time_steps = np.arange(1, 101, 1)
data_dict2 = {
    'Time (s)': time_steps
}

for i in range(100):
    data_dict2[f'第{i+1}s速度 (m/s)'] = v_final[i]
df2 = pd.DataFrame(data_dict2)
df2_transposed = df2.transpose()

with pd.ExcelWriter('q4_point_generated.xlsx') as writer:
    df1.to_excel(writer, sheet_name='Sheet1', header=True)
    df2_transposed.to_excel(writer, sheet_name='Sheet2', header=True)
```