基于几何模型的板凳龙运动模型

摘　要

这里是摘要

一、 问题背景与重述

1.1 问题背景

我国浙南地区有一项盛大的元宵习俗——“板凳龙”：村民们把一节节的板凳钻孔连接，组成几十至上百节的板凳长龙。演出时，龙头在前领头，龙身和龙尾相随着将长龙盘成圆形图案。在能够自如盘入盘出的条件下，若是能够减少盘龙面积、加快行进速度，观赏性将会进一步提升。

1.2 问题要求

假设有一 223 节板凳组成的板凳龙，其中第 1 节为龙头，后面 221 节为龙身，最后 1 节为龙尾。龙头的板长为 341 cm，龙身和龙尾的板长均为 220 cm，所有板凳的板宽均为 30 cm。每节板凳上均有两个孔，孔径为 5.5 cm，孔的中心距离最近的板头 27.5 cm。如下图 1、2、3。

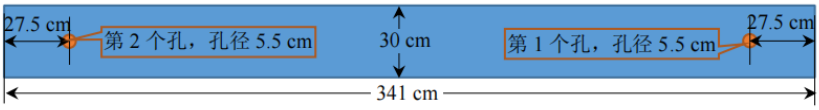

图 **1**

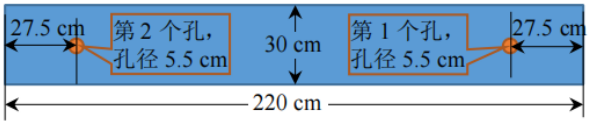

图 **2**

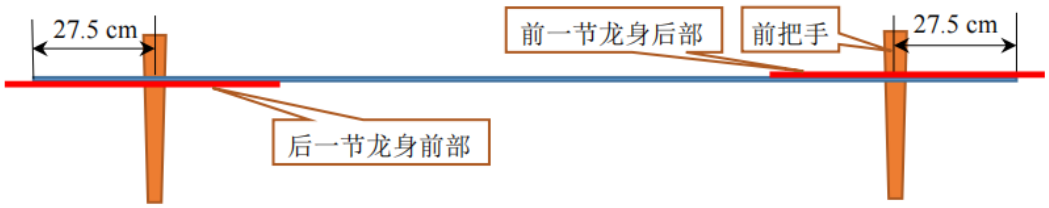

图 **3**

我们需要解决以下问题：

1. 舞龙队沿螺距为 55 cm 的等距螺线顺时针盘入，各把手中心均位于螺线上。龙头前把手的行进速度始终保持 1 m/s。初始时，龙头位于螺线第 16 圈 A 点处（见图 4）。请给出从初始时刻到 300 s 为止，每秒整个舞龙队的位置和速度（指龙头、龙身和龙尾各前把手及龙尾后把手中心的位置和速度，下同），将结果保存到文件 result1.xlsx 中（其中“龙尾（后）”表示龙尾后把手，其余的均是前把手，结果保留 6 位小数，下同）。同时在论文中给出 0 s、60 s、120 s、180 s、240 s、300 s 时，龙头前把手、龙头后面第 1、51、101、151、201 节龙身前把手和龙尾后把手的位置和速度。

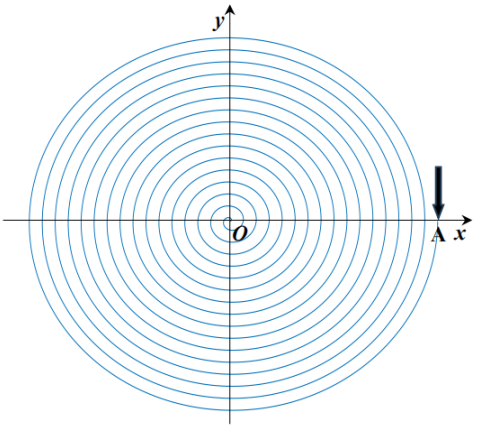

图 **4**

2. 舞龙队沿问题 1 设定的螺线盘入，请确定舞龙队盘入的终止时刻，使得板凳之间不发生碰撞（即舞龙队不能再继续盘入的时间），并给出此时舞龙队的位置和速度，将结果存放到文件 result2.xlsx 中。同时在论文中给出此时龙头前把手、龙头后面第 1、51、101、151、201 条龙身前把手和龙尾后把手的位置和速度。
3. 从盘入到盘出，舞龙队将由顺时针盘入调头切换为逆时针盘出，这需要一定的调头空间。若调头空间是以螺线中心为圆心、直径为 9 m 的圆形区域（见图 5），请确定最小螺距，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。

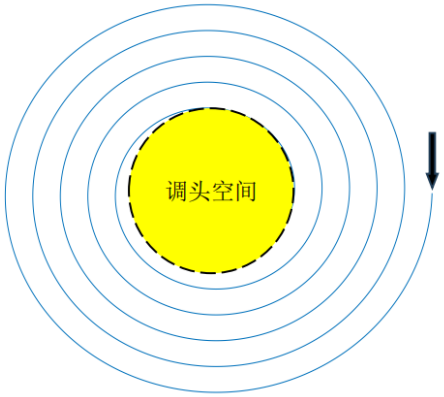

图 **5**

4. 盘入螺线的螺距为 1.7 m，盘出螺线与盘入螺线关于螺线中心呈中心对称，舞龙队在问题 3 设定的调头空间内完成调头，调头路径是由两段圆弧相切连接而成的 S 形曲线，前一段圆弧的半径是后一段的 2 倍，它与盘入、盘出螺线均相切。能否调整圆弧，仍保持各部分相切，使得调头曲线变短？

   龙头前把手的行进速度始终保持 1 m/s。以调头开始时间为零时刻，给出从 −100 s 开始到 100 s 为止，每秒舞龙队的位置和速度，将结果存放到文件 result4.xlsx 中。同时在论文中给出 −100 s、−50 s、0 s、50 s、100 s 时，龙头前把手、龙头后面第 1、51、101、151、201 节龙身前把手和龙尾后把手的位置和速度。
5. 舞龙队沿问题 4 设定的路径行进，龙头行进速度保持不变，请确定龙头的最大行进速度，使得舞龙队各把手的速度均不超过 2 m/s。

二、 问题分析

2.1 问题一的分析

对于问题一，已知盘入螺线的螺距和起点，本文能够建立该螺线对应的极坐标方程。由于龙头前把手沿盘入螺线的行进速度恒定，可以结合已经建立起的极坐标方程，采用积分的方式去确定某段时间内走过的路径长度与走过的角度的关系，进而得到每一时刻的位置坐标。由此，本文能够利用极坐标方程与每节板凳的数据，建立由前把手中心位置得到后把手中心位置的迭代公式。

在舞龙队盘入的过程中，虽然每个把手中心的前进速度各不相同，但不难看出，同一板凳上前后的把手中心沿板凳中心所在直线的速度是一样的。利用这一特性，本文通过几何方法得到前后把手中心速度方向与板凳中心所在直线的夹角，进而得到同一板凳上前后的把手中心速度的关系式，建立起由前把手中心速度得到后把手中心速度的迭代公式。

2.2 问题二的分析

对于问题二，在舞龙队沿着螺线盘入的过程中，随着龙头前把手逐渐接近圆心，由于每块板凳有自己固定的形状与体积，因此在接近于螺线中心的过程中，由于螺距的限制，可能会有一个时刻，舞龙队之间发生碰撞。

经过分析，由于龙身与龙尾有的板凳形状完全相同，并且在盘入过程中走过的路径也相同。因此这些板凳是否发生碰撞只需要考虑第一节龙身是否与其他板凳发生碰撞，因为若在走过的路径中，第一节龙身与其他板凳发生碰撞，后面的龙身自然也不会与其他板凳发生碰撞。故龙身前部部分只有龙头和第一节龙身可能会与其他板凳发生碰撞，并且显然只有这两块板外的四个角点可能会与其他板凳相撞，如下图所示：

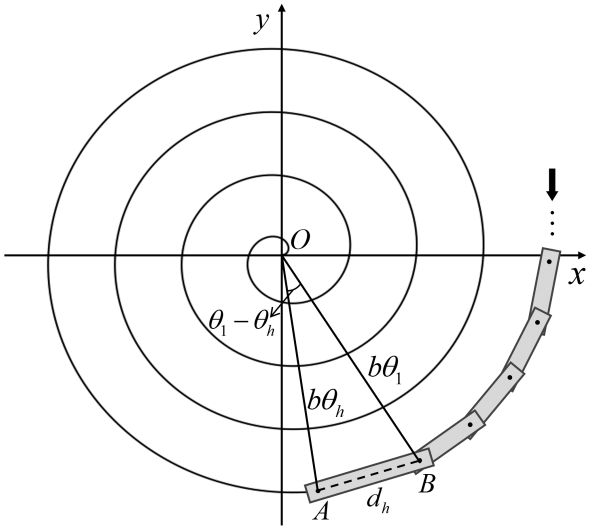

图 **6**

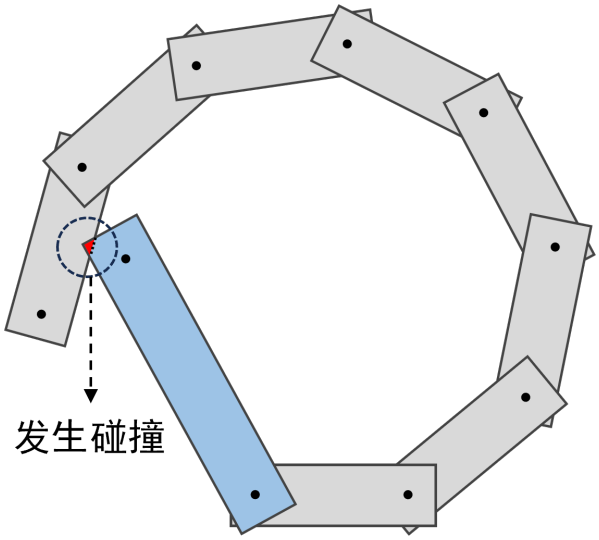

图 **7**

由于问题一中，我们已经确定了各个时刻各把手中心位置，并且，上面提到的两块板凳把手中心与板凳的角的相对位置是固定的，本文由此计算出每个时刻各个角点的位置坐标。同时，在龙头外一层螺线上分布着一圈板凳，板凳的两个把手中心连接形成的线段是此螺线的弦，并且可以利用问题一中所得到的坐标求出该弦所在直线的解析式。因此本文利用角点到弦所在直线的距离来刻画碰撞与否，当距离小于板凳的宽度时说明已经发生了碰撞。

2.3 问题三的分析

结合问题一、问题二，本文能够求解出当螺距已知时舞龙队盘入的终止时刻，以及此时舞龙队各把手的位置。对于问题三的问题要求，只需要通过调整螺距，使得在该螺

距对应的终止时刻时，龙头前把手中心的位置恰好位于调头空间的边界上。

由此，可以先确定出两个螺距，使得在其各自的终止时刻时，龙头前把手中心的位置分别位于题设要求的调头空间内和调头空间外。再采用十分法，精确求解满足要求的最小螺距。

2.4 问题四的分析

对于问题四的第一问，即能否通过调整圆弧使得调头曲线变短，由于调头路径是由两段圆弧构成的 S 型曲线，并且两段圆弧分别与盘入螺线、盘出螺线相切，通过分析其几何特性，本文通过计算判断是否能通过调整两段圆弧的半径比例使调头路径变短。

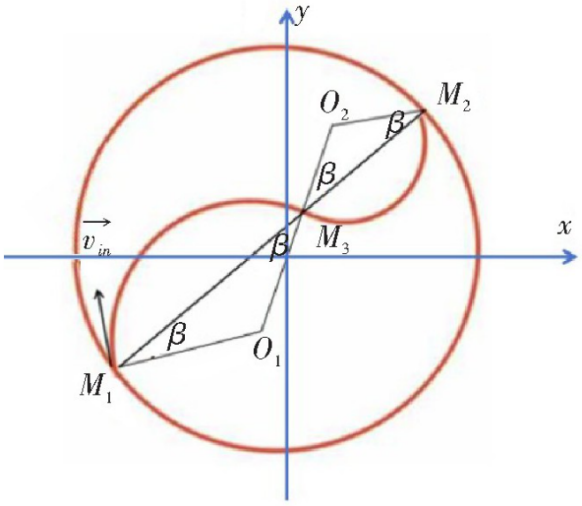

图 **8**

此路径有四段弧线，本文分别称作盘入螺线、第一段圆弧、第二段圆弧、盘出螺线。同时，此路径有三个关键节点，分别是盘入螺线与第一段圆弧的交点、两段圆弧的交点以及第二段圆弧与盘出螺线的交点。

首先，与问题一中的思路类似，当确定了调头路径的方程以后，根据龙头前把手的速度，可以得到各个时刻龙头前把手的位置坐标。本文先构建了一个判断函数：根据一块板凳的前把手的位置，判断该板凳后把手所位于的弧线。

当一块板凳的前后两个把手均位于螺线上时，位置坐标的推导公式与问题一中相似；当一块板凳的前后两个把手均位于同一段圆弧上时，或者当一块板凳的前后两个把手分别位于两段不同的弧线时，可根据前把手的位置，再利用几何知识，推导出后一把手位置。这样就可以通过前一个把手位置，通过判断函数判断后一把手所在圆弧，即这两个把手的位置情况，再利用上面的位置推导公式，将问题一中的位置迭代公式推广到在整个路径中运动时位置的迭代公式。进而，与问题一的思路类似，通过位置迭代公式，本文得到各个时刻，每块板凳前把手中心位置的坐标。

当一块板凳的前后两个把手均位于螺线上时，速度的推导公式与问题一中相似；当一块板凳的前后两个把手均位于同一段圆弧上时，速度推导相同；当一块板凳的前后两个把手分别位于两段不同的弧线时，可以根据前把手与后把手的位置坐标以及其所处的曲线，推导出前后把手中心速度的方向，以及连接前后把手中心位置线段所在直线的方向。利用直线夹角公式分别计算出前后把手中心速度方向与其所处弧线所在直线的夹角。最后根据问题一中的思路，采用关联速度的方法，根据前把手的速度得到后把手的速度。结合上述位置迭代公式，我们就将问题一中的速度迭代公式推广到了在整个路径中运动时速度的迭代公式。进而，与问题一的思路类似，通过速度的迭代公式，本文得到各个时刻，每个前把手中心的速度。

2.5 问题五的分析

在舞龙队调头过程中，龙头前把手行进速度始终保持不变，但其余各把手的速度会随着位置的变化而发生变化。经过分析问题四的结果发现，在龙头板凳由第二段调头圆弧进入螺线的过程中，第三块板凳的前把手中心会在运动过程中速度达到最大值。

本文通过分析问题四结果确定了第三块板凳的前把手中心速度达到最大值的大致时间区间，进而计算出此时龙头前把手大致位置区间。再采用十分法，利用编程求解第三块板凳的前把手中心速度达到最大值的精确时间，以及此时龙头前把手精确位置。进而利用问题四中速度的迭代公式，可以得到在这个位置时龙头前把手中心速度与第三块板凳的前把手中心速度的关系，再通过限制第三块板凳的前把手中心速度不超过 2 m/s，进而求得龙头最大行进速度。

三、 模型准备

3.1 模型假设

1. 在舞龙队盘入前，已经以相同的螺距排列为等距螺线列队在盘入螺线外。
2. 忽略板凳厚度带来的影响。
3. 各把手中心严格位于螺线上。
4. 忽略摩擦力带来的影响。

3.2 符号说明

所使用的符号及说明如表 3.1 所示。

表 **1**　符号说明

符号	说明	单位
d	螺距	m
ρ	板径	m
θ	极角	rad
v	龙头前把手行进速度	m/s
d_1	把手中心到最近板头的距离	m
d_2	半板宽	m
l_i	第 i 块凳前后把手中心所在直线	
l	板凳前后把手中心距离	m
A_1	龙头前外部角点	
A_2	龙头后外部角点	
A_3	第一节龙身前外部角点	
A_4	第一节龙身后外部角点	
di_j	角点 A_j 与直线 l_i 的距离	m
O_1	第一段圆弧圆心	
O_2	第二段圆弧圆心	
ϕ	第一段圆弧所对圆心角	rad
D	调头空间直径	m

注意：其他符号已在文章的相应部分给出说明。

四、 模型的建立与求解

4.1 问题一模型的建立与求解

4.1.1 模型的建立

STEP1 龙头前把手中心位置的确定

由已知的螺距 $d = 0.55\,\text{m}$ 和起点 $A(8,0)$，本文能够建立盘入螺线所对应的极坐标方程：

$$\rho(\theta) = \frac{d}{2\pi}\theta$$

由于龙头前把手沿盘入螺线的行进速度恒定为 $v = 1\,\text{m/s}$，从初始时刻 $t = 0\,\text{s}$ 至 $t = t_0 \in [0, 300]$ 时刻，龙头前把手走过的路径长度为 vt_0，极角的角度从 $\theta = 32\pi$ 变为 $\theta = \theta_0$。根据螺线长度积分公式，能够得到以下关系式：

$$vt_0 = \int_{32\pi}^{\theta_0} \sqrt{(\rho'(\theta))^2 + \rho^2(\theta)}\,\mathrm{d}\theta$$

通过已知的 t_0，可以解得对应的 θ_0，进而通过极坐标与直角坐标的转化公式：

$$\begin{cases} x_0 = \rho(\theta_0)\cos\theta_0 \\ y_0 = \rho(\theta_0)\sin\theta_0 \end{cases}$$

得到 $t = t_0$ 时刻龙头前把手中心坐标 (x_0, y_0)。

STEP2 建立位置迭代公式在得到了龙头前把手中心每个时刻的坐标后，本文建立了一个位置迭代式以计算各个把手在每一个时刻的坐标，建立过程如下：

假设在某一时刻下，某一板凳，其前把手中心距离为 l，其前把手中心位置坐标为 (x_1, y_1)，极角为 $\theta = \theta_1$，极径为 $\rho = \rho_1$。假设该板凳后把手中心位置此时的坐标为 (x_2, y_2)，极角为 $\theta = \theta_2$，根据螺线方程，其极径满足方程：

$$\rho_2 = \rho_1 + \frac{d}{2\pi}(\theta_2 - \theta_1) \tag{1}$$

如图 8 所示的三角形中，通过余弦定理可得到如下公式：

$$\rho_1^2 + \rho_2^2 - 2\rho_1\rho_2\cos(\theta_2 - \theta_1) = l^2 \tag{2}$$

联立式 (1)、式 (2)，采用二分求零点的方法可解得 ρ_2 与 θ_2，进而通过坐标转化公式：

$$\begin{cases} x_2 = \rho_2\cos\theta_2 \\ y_2 = \rho_2\sin\theta_2 \end{cases}$$

得到该板凳后把手中心位置的坐标 (x_2, y_2)。

通过 STEP1 中各个时刻龙头前把手中心坐标，利用该迭代公式，本文能够求解出各个时刻整个舞龙队各把手的位置。

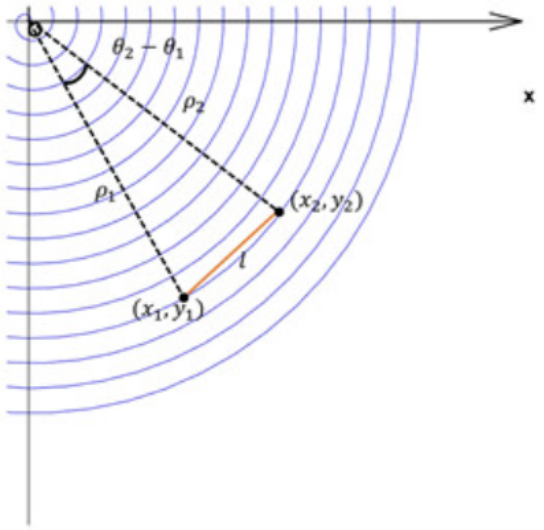

图 **9**

STEP3 建立速度迭代公式

不难发现，在舞龙队盘入等距螺线的过程中，同一板凳上前后把手中心沿板凳前后把手中心所在直线的速度是一样的。本文借助这一特性，利用前后把手关于板凳的关联速度建立迭代公式，建立过程如下：

假设在某一时刻下，如图 10 所示：

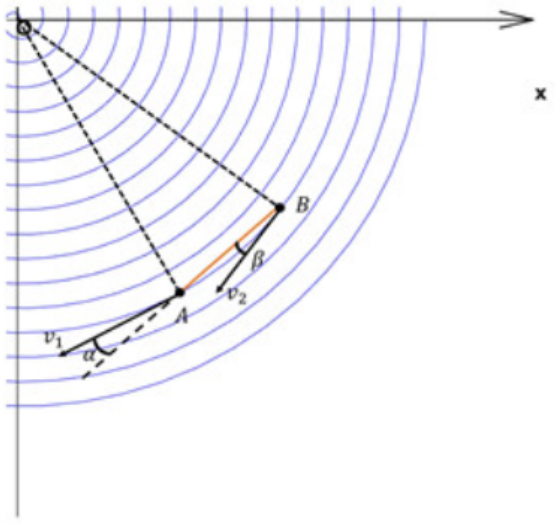

图 **10**

某一板凳，已知其前把手中心位置坐标为 $A(x_1,y_1)$，行进速度为 v_1，极角为 θ_1，螺线在该点的切线斜率为 k_1。由螺线极坐标方程，可以得到 k_1 为：

$$k_1=\frac{\sin\theta_1+\theta_1\cos\theta_1}{\cos\theta_1-\theta_1\sin\theta_1}$$

由上述位置迭代公式可以得到后把手中心 B 的坐标，记为 (x_2, y_2)，设其行进速度为 v_2。由螺线极坐标方程，可以得到其极角为 θ_2，设螺线在该点的切线斜率为 k_2，则 k_2 为：

$$k_2 = \frac{\sin\theta_2 + \theta_2\cos\theta_2}{\cos\theta_2 - \theta_2\sin\theta_2}$$

接下来，分别设螺线在点 A、点 B 处的切线与 A、B 所在直线的夹角分别为 α、β。由点 A、点 B 的坐标可以得到 A、B 所在直线的斜率 k 为：

$$k = \frac{y_1 - y_2}{x_1 - x_2}$$

从而可以得到 α、β 为：

$$\alpha = \arctan\left|\frac{k_1 - k}{1 + kk_1}\right|$$

$$\beta = \arctan\left|\frac{k_2 - k}{1 + kk_2}\right|$$

最后，利用前后把手关于板凳的关联速度建立速度迭代公式如下：

$$v_1\cos\alpha = v_2\cos\beta$$

通过已经求得的各个时刻整个舞龙队各把手的位置，以及龙头前把手的恒定速度，可以得到各个时刻整个舞龙队各把手的速度。

4.1.2 模型计算结果

将数据带入位置、速度迭代公式，通过程序得到结果如下：

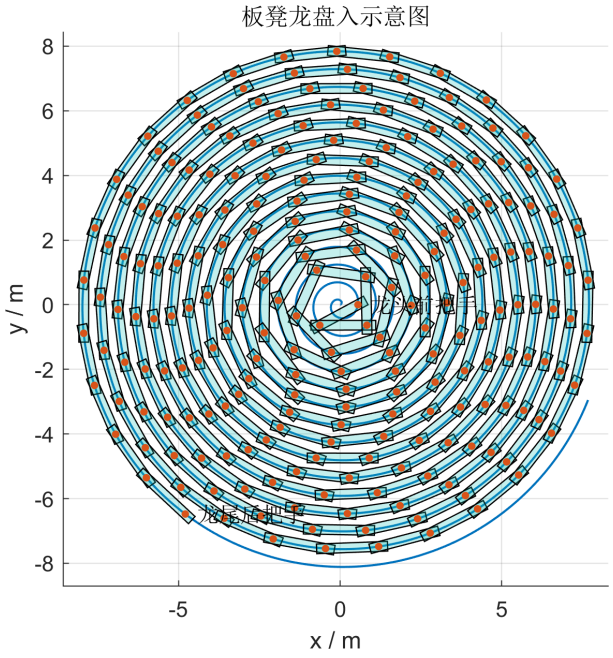

图 **11**

表 **2**　问题一已知参数

参数	d	v_h	d_h	d_b
数值	0.55m	1m/s	2.86m	1.65m

将龙头的初始运动状态代入建立的模型，即可求得所有把手运动状态。通过递推求出从初始时刻到 300 s 为止，每秒整个舞龙队的位置和速度，下述表3和表4分别为 0 s、60 s、120 s、180 s、240 s、300 s 时，龙头前把手、龙头后面第 1、51、101、151、201 节龙身前把手和龙尾后把手的位置和速度，具体求解结果见文件 result1.xlsx。

表 **3**　问题一舞龙队的位置

	0s	60s	120s	180s	240s	300s
龙头 x(m)	8.800000	5.796934	-4.090654	-2.953259	2.578971	4.431365
龙头 y(m)	0.000000	-5.773329	-6.300643	6.099638	-5.363954	2.298233
第 1 节龙身 x(m)	8.363824	7.455390	-1.452253	-5.229685	4.810833	2.480900
第 1 节龙身 y(m)	2.826544	-3.443281	-7.404474	4.368312	-3.575548	4.389951
第 51 节龙身 x(m)	-9.518736	-8.685398	-5.537887	2.901007	5.970603	-6.299130
第 51 节龙身 y(m)	1.341112	2.543141	6.382421	7.244935	-3.842026	0.490490
第 101 节龙身 x(m)	2.914039	5.684538	5.356285	1.887650	-4.930766	-6.224242
第 101 节龙身 y(m)	-9.918295	-8.003181	-7.561567	-8.473987	-6.369293	3.956744
第 151 节龙身 x(m)	10.861711	6.684695	2.395398	1.016445	2.981390	7.053566
第 151 节龙身 y(m)	1.828852	8.132552	9.725716	9.423434	8.393873	4.371895
第 201 节龙身 x(m)	4.554963	-6.617181	-10.626288	-9.292339	-7.467880	-7.472671
第 201 节龙身 y(m)	10.725178	9.027361	1.366600	-4.236322	-6.167517	-5.243064
龙尾（后）x(m)	-5.305286	7.362212	10.974813	7.391885	3.257083	1.809208
龙尾（后）y(m)	-10.676664	-8.799924	0.836694	7.484356	9.463677	9.296260

表 **4**　问题一舞龙队的速度

	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)	0.999971	0.999961	0.999945	0.999917	0.999859	0.999709
第 51 节龙身 (m/s)	0.999744	0.999663	0.999537	0.999331	0.998939	0.998066
第 101 节龙身 (m/s)	0.999578	0.999450	0.999266	0.998972	0.998433	0.997302
第 151 节龙身 (m/s)	0.999452	0.999298	0.999078	0.998730	0.998113	0.996861
第 201 节龙身 (m/s)	0.999352	0.999183	0.998929	0.998556	0.997892	0.996573
龙尾（后）(m/s)	0.999308	0.999139	0.998883	0.998490	0.997816	0.996478

4.2 问题二模型的建立与求解

4.2.1 模型的建立

STEP1 龙头与第一节龙身外部四个角点位置的确定

对任一块板凳，记把手中心距最近的板头距离为 d_1，半板宽为 d_2，前后把手中心连线所在直线为 l_1，板外侧边所在直线为 a_3，前、后把手中心与最近的外部角点连线所在直线分别为 a_2、a_4，由对称性，l_1 与 a_2、a_4 的夹角均为 γ，如图 12 所示：

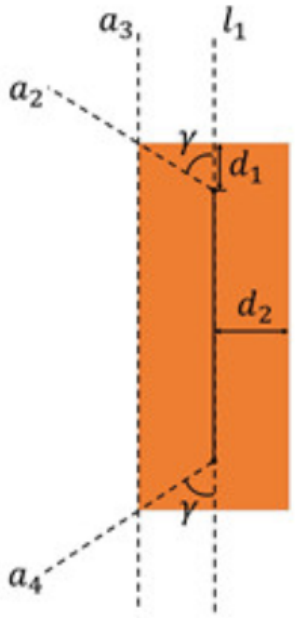

图 **12**

龙头前外部角点 A_1 位置的确定如下：

假设 t 时刻下龙头前把手中心位置坐标为 (x_1, y_1)，极角为 $\theta = \theta_1$。根据问题一的位置迭代公式，求龙头后把手中心位置坐标为 (x_2, y_2)，进而求得龙头两把手中心所在直线 l_1 的解析式：

$$l_1: \quad y - y_1 = k_1(x - x_1) \quad \text{其中} k_1 = \frac{y_1 - y_2}{x_1 - x_2}$$

由于 a_2 是由 l_1 绕前把手中心逆时针旋转 γ 得到的，根据直线旋转角公式，得到 a_2 的解析式：

$$a_2: \quad y - y_1 = k_2(x - x_1) \quad \text{其中} k_2 = \frac{\tan\gamma + k_1}{k_1 \tan\gamma - 1}, \quad \tan\gamma = \frac{d_2}{d_1}$$

由于 a_3 是由 l_1 向远离中心原点方向平移 d_2 得到的，由此得到 a_3 的解析式：

$$a_3: \quad y = k_1 x + b \quad \text{其中} b \text{ 满足条件：} \frac{|b - y_1 + k_1 x_1|}{\sqrt{1 + k_1^2}} = d_2 \text{ 且} |b| > |y_1 - k_1 x_2|$$

联立 a_2 与 a_3，即可解得 A_1 的坐标：

$$(x_{A_1}, y_{A_1}) = \left(\frac{y_1 - k_2 x_1 - b}{k_1 - k_2}, \frac{k_1 y_1 - k_1 k_2 x_1 - k_2 b}{k_1 - k_2} \right)$$

类似的，能够确定 t 时刻下龙头后外部角点 A_2、第一节龙身前外部角点 A_3、第一节龙身后外部角点 A_4 的坐标。

STEP2 四个角点碰撞情况的判断

在 STEP1 中，本文利用 t 时刻下确定的龙头前把手坐标 (x_1, y_1)，解得四个角点 A_1、A_2、A_3、A_4 的坐标。

同时，可以通过问题一所得的结果，得到 t 时刻下极角 $\theta \in \left[\theta_i + \frac{3\pi}{2}, \theta_i + \frac{5\pi}{2}\right]$ 的各前把手中心坐标 (x_i, y_i)（即为该前把手属于第 i 块板凳，龙头为第 1 块板凳），记满足前把手坐标 (x_i, y_i) 落在要求的极角范围内的 i 的集合为 I。通过两点间直线公式，能够求解出第 i 块板凳前后把手中心连线所在直线 l_i 的解析式：

$$l_i: \quad y - y_i = \frac{y_i - y_{i+1}}{x_i - x_{i+1}}(x - x_i)$$

得到了角点坐标与直线 l_i 的解析式后，记角点 A_j 与直线 l_i 的距离为 d_{ij}，即：

$$d_{ij} = \frac{\left| \frac{y_i - y_{i+1}}{x_i - x_{i+1}}(x_{A_j} - x_i) - y_{A_j} + y_i \right|}{\sqrt{\left(\frac{y_i - y_{i+1}}{x_i - x_{i+1}} \right)^2 + 1}}$$

对于是否发生碰撞，本文给出如下判断准则：

$$\begin{cases} \forall i, j \in I,\ j = 1,2,3,4,\ d_{ij} > d_2 & \text{则}t\text{ 时刻未发生碰撞} \\ \exists i, j (i \in I,\ j = 1,2,3,4) \text{ 使得 } d_{ij} \le d_2 & \text{则}t\text{ 时刻发生碰撞} \end{cases}$$

由此能够确定舞龙队盘入的终止时刻。

4.2.2 模型计算结果

代入数据后，本文通过程序得到终止时刻 412.473894 s，发生碰撞的点为龙头左前角点，位置、速度结果如下：

表 **5**　问题二舞龙队的位置和速度

	横坐标 x(m)	纵坐标 y(m)	速度 (m/s)
龙头 x	1.647511	1.556154	1.000000
第 1 节龙身	-1.164436	2.078219	0.991145
第 51 节龙身	1.803760	4.119592	0.976112
第 101 节龙身	-1.099658	-5.792623	0.973782
第 151 节龙身	0.401238	-7.006028	0.972839
第 201 节龙身	-7.954575	-0.664687	0.972327
龙尾（后）	1.518361	8.232690	0.972167

4.3 问题三模型的建立与求解

4.3.1 模型的建立

通过问题一、问题二中建立的模型，不难发现，在其他条件不变的情况下，每个时刻下舞龙队各把手所处的位置和舞龙队盘入的终止时刻都由螺距大小决定。

因此，本文在问题三中将沿用问题一、问题二的模型和公式，对于任一螺距 d，本文建立如下判定流程：

STEP1 对于该螺距 d，利用问题二的模型，求解出该螺距下舞龙队的盘入终止时刻。

STEP2 将 STEP1 中求得的终止时刻与螺距 d 代入问题一的模型，求解出此时龙头前把手中心的坐标。

STEP3 检查 STEP2 中得到的龙头前把手中心坐标落在题设要求调头空间的边界上/内部/外部。

本文首先通过调整螺距 d 得到两个分别使龙头前把手中心落在调头空间内部和外部的螺距，再通过十分法，精确求解使龙头前把手中心恰好位于边界上的螺距。

4.3.2 模型计算结果

代入数据后，本文通过程序求解得到能够满足要求的最小螺距为：0.450338 m。

在该最小螺距下，碰撞是由龙头板凳的左右角点造成的。

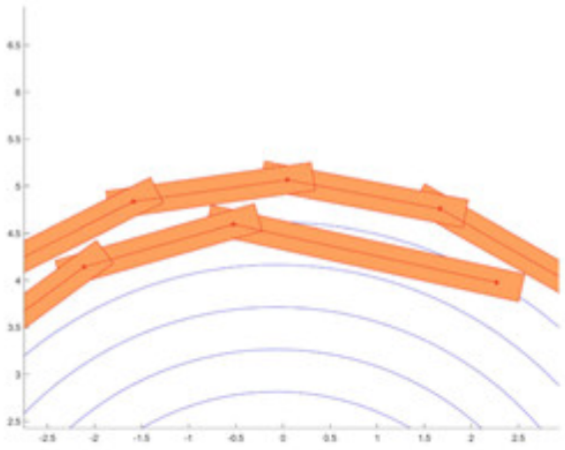

图 **13**

4.4 问题四模型的建立与求解

4.4.1 模型的建立

1. 调头曲线长度模型

舞龙队在问题三调头空间设定的基础上完成调头，由题目条件可知，调头路径是由两段圆弧相切连接而成的 S 形曲线，前段圆弧半径是后段的 2 倍，盘出螺线与盘入螺线关于螺线中心呈中心对称，S 形曲线与盘入、盘出螺线均相切。盘入盘出曲线（左）和调头曲线（右）如图14所示：

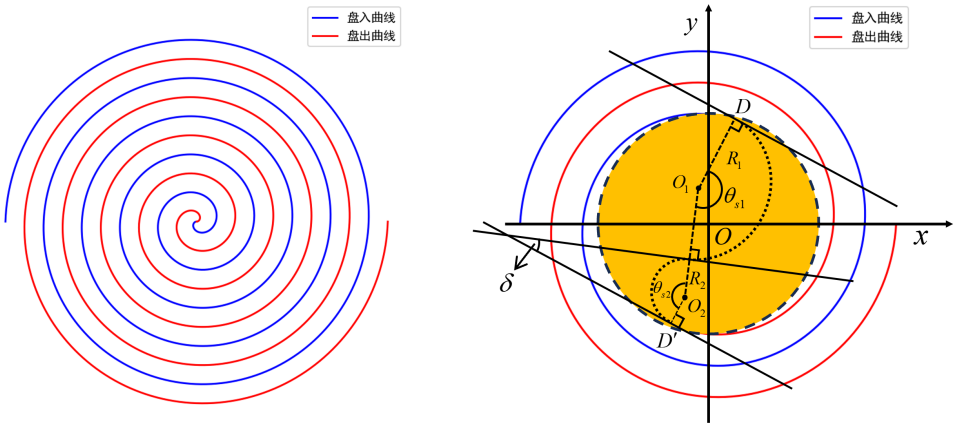

图 **14**　盘入盘出曲线和调头曲线几何关系示意图

对于调头曲线，由盘入曲线和盘出曲线成中心对称的几何关系与题设可知：

$$\begin{cases} R_1 = 2R_2 = R \\ \theta_{s1} = \theta_{s2} = \theta \end{cases} \tag{1}$$

调头曲线由两段圆弧组成，其长度可表示为：

$$s = \theta_{s1}R_1 + \theta_{s2}R_2 \tag{2}$$

2. 最短调头曲线优化模型

在第四问中舞龙队需要在直径为 9m 的圆内完成调头，可以证明调头曲线的长度仅与调头点坐标有关：设调头点坐标为 $D(x_m, y_m)$，盘出调头空间的坐标由中心对称可得 $D'(-x_m, -y_m)$。设于圆弧 $O_1(x_1, y_1)$、$O_2(x_2, y_2)$ 相切的直线斜率分别为 k_1、k_2，两圆弧公切线的斜率为 k_3，如图14（右）所示。

其中，直线 O_1D、直线 O_2D' 的方程可表示为：

$$\begin{cases} y_{O_1D} = -\dfrac{1}{k_1}(x - x_m) + y_m \\ y_{O_2D'} = -\dfrac{1}{k_2}(x + x_m) - y_m \end{cases} \tag{3}$$

由几何关系可得：

$$\begin{cases} O_1D = 2O_2D' \\ O_1O_2 = O_1D + O_2D' \end{cases} \tag{4}$$

用代数方程可表示为：

$$\begin{cases} \delta = tan^{-1}k_3 - tan^{-1}k2 \\ y_1 = -\dfrac{1}{k_1}(x_1 - x_m) + y_m \\ y_2 = -\dfrac{1}{k_2}(x_2 + x_m) - y_m \\ \sqrt{1 + (\frac{1}{k_1})^2}|x_m - x_1| = 2\sqrt{1 + (\frac{1}{k_2})^2}| - x_m - x_2| \\ \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} = \sqrt{1 + (\frac{1}{k_1})^2}|x_m - x_1| + \sqrt{1 + (\frac{1}{k_2})^2}| - x_m - x_2| \end{cases} \tag{5}$$

由式 (32) 可得圆弧运动的半径 R 和角度 θ 为：

$$\begin{cases} R = \dfrac{1}{3}\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\ \theta = \pi - \delta \end{cases} \tag{6}$$

舞龙队龙头沿螺线盘旋越接近螺线中心所需调头曲线长度越短，使用问题二碰撞模型模拟舞龙队在螺距 1.7m 的螺线上的运动，发现在盘出时龙头与龙身发生碰撞。因此当舞龙队从最晚调头点调头时恰好能顺利盘出（不发生碰撞），并且此时调头曲线最短。以最晚调头点到原点的距离为半径，原点为圆心作圆，此区域为最小调头空间。

3. 舞龙队在圆弧曲线上的运动模型

1) 位置坐标模型

板凳在调头圆弧曲线上运动状态如图15所示：

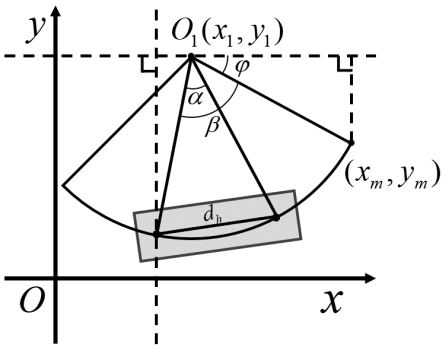

图 **15**　板凳在调头圆弧曲线上运动状态示意图

在圆弧上龙头以匀速运动，以下推导以前部分圆弧为例，则此时龙头的角速度 ω_h 为：

$$\omega_h = \frac{v_h}{R_1} \tag{7}$$

以调头时刻为零时刻，龙头从进入圆弧后沿圆心转过的角度 β_h 与时间 t 的关系为：

$$\beta_h = \omega_h t \tag{8}$$

调头点与圆心连线与 x 轴夹角为：

$$\varphi = tan^{-1}(\frac{y_1 - y_m}{x_m - x_1}) \tag{9}$$

其中，相邻两把手的节点与圆心连线夹角可由余弦定理求得，表示为：

$$\begin{cases} \alpha_h = cos^{-1}(\dfrac{2R_1^2 - d_h^2}{2R_1^2})，\text{龙头板凳} \\ \alpha_b = cos^{-1}(\dfrac{2R_1^2 - d_b^2}{2R_1^2})，\text{龙身板凳} \end{cases} \tag{10}$$

由式 (34)(35)(36) 可求得龙头前把手的坐标 [3]，即：

$$\begin{cases} x_h = x_1 + R_1 cos(\beta_h + \varphi) \\ y_h = y_1 - R_1 sin(\beta_h + \varphi) \end{cases} \tag{11}$$

龙头后第 $i+1$ 节龙身前把手坐标可表示为：

$$\begin{cases} x_{bi} = x_1 + R_1 cos(\beta_h + \varphi - \alpha_h - i\alpha_b) \\ y_{bi} = y_1 - R_1 sin(\beta_h + \varphi - \alpha_h - i\alpha_b) \end{cases} \tag{12}$$

2) 速度模型

当板凳沿圆弧运动时速度时刻相等，即：

$$v_{bi} = v_{b(i+1)} \tag{13}$$

4. 板凳运动状态的判断

由调头点坐标可求得板凳从螺线运动进入圆弧运动的临界旋转角，即：

$$\theta_{turn} = \frac{\sqrt{x_1^2 + y_1^2}}{b} \tag{14}$$

根据问题四步长 $\Delta t = 1s$ 判断，当 t 时刻满足 $\theta \leq \theta_{turn}$ 时表示从该时刻板凳从沿螺线运动转变为沿圆弧运动。

任一板凳从开始圆弧运动到结束时运动的时间可表示为：

$$T_i = \frac{s}{v_{bi}} \tag{15}$$

其中，T_i、v_{bi} 分别表示龙头后第 i 节龙身圆弧运动持续的时间和速度。

4.4.2 模型的求解

1. 最短调头曲线的求解

采用问题三所示的调头空间时，将调头空间半径 $r_d = 4.5$m 代入模型可得：

$$s = 14.854167\text{m} \tag{16}$$

对于最短调头曲线的求解，使用二分法和变步长遍历算法寻找调头曲线最小值的流程如下：

Step1：设置调头空间半径的遍历范围 $r_d \in [0, 4.5]$，取较大的遍历步长 $\Delta r_d = 0.1$，遍历，模糊检测到使舞龙队恰好不发生碰撞时 r_d 的取值区间；

Step2：根据得到的取值区间，利用二分法进行 n_0 次迭代，当步长之差小于 0.01m 时跳出循环，找到准确的 r_d；

经过二分法求解之后，得到最小调头空间半径 $r_d = 2.89$m 时，在盘入盘出过程中舞龙队恰好不会相撞。进而计算最短调头曲线长度为：

$$s_{min} = 9.879203\text{m} \tag{17}$$

2. 舞龙队位置和行进速度的求解

联立调头区域边界方程和盘入曲线方程：

$$\begin{cases} x^2 + y^2 = 4.5^2 \\ r = b\theta \end{cases} \tag{18}$$

得到调头点坐标 (x_m, y_m)，该点为零时刻龙头前把手初始坐标。根据临界旋转角度判断板凳运动状态，当在圆弧上运动时，龙身运动状态模型如式 (46)：

$$\begin{cases} x_{bi} = x_1 + R_1 cos(\beta_h + \varphi - \alpha_h - i\alpha_b) \\ y_{bi} = y_1 - R_1 sin(\beta_h + \varphi - \alpha_h - i\alpha_b) \\ v_{bi,(t+1)} = v_{bi,t} \end{cases} \tag{19}$$

其中，$v_{bi,(t+1)}$ 表示 $v_{bi,t}$ 下一秒运动速度。

在盘入螺线上运动时，由问题一建立的模型得：

$$\begin{cases} x_{bi} = r_i cos\theta_i \\ y_{bi} = r_i sin\theta_i \\ \dfrac{v_{b(i+1)}}{v_{bi}} = \dfrac{cos\alpha_{(i+1)}}{cos\beta_i} \end{cases} \tag{20}$$

龙头前把手的行进速度始终保持 1 m/s。以调头开始时间为零时刻，通过递推求出从 −100 s 开始到 100 s 为止，每秒整个舞龙队的位置和速度，下述表6和表7分别为 −100 s、−50 s、0 s、50 s、100 s 时，龙头前把手、龙头后面第 1、51、101、151、201 节龙身前把手和龙尾后把手的位置和速度，具体求解结果见文件 result4.xlsx。

表 **6**　问题四舞龙队的位置

	-100s	-50s	0s	50s	100s
龙头 x(m)	7.866065	6.664626	-2.943078	-1.783076	2.746366
龙头 y(m)	3.509739	1.649141	-3.379179	-6.039617	-7.692214
第 1 节龙身 x(m)	6.360066	5.519350	-0.375026	4.204898	0.088367
第 1 节龙身 y(m)	5.941111	4.269818	-4.638033	4.518307	8.071554
第 51 节龙身 x(m)	-10.542447	-3.862677	2.157886	-1.269492	2.458889
第 51 节龙身 y(m)	3.046781	-8.858199	-7.856570	-6.191475	3.791881
第 101 节龙身 x(m)	-12.000535	9.988706	3.303470	-7.829261	-6.992109
第 101 节龙身 y(m)	-4.591356	-6.187642	10.007598	4.832164	2.836937
第 151 节龙身 x(m)	-14.375875	12.895695	-6.739031	-4.224519	9.120762
第 151 节龙身 y(m)	-1.757570	-4.053001	10.503309	-10.558235	-4.293867
第 201 节龙身 x(m)	-11.800121	10.335096	-6.596106	0.754248	9.078184
第 201 节龙身 y(m)	10.731721	-10.980500	12.525488	-13.168898	7.992703
龙尾（后）x(m)	-1.234909	0.444688	-2.241075	5.481340	-11.376227
龙尾（后）y(m)	-16.508675	15.711052	-14.663702	12.790419	-6.044073

表 **7**　问题四舞龙队的速度

	-100s	-50s	0s	50s	100s
龙头 (m/s)	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身 (m/s)	0.999904	0.999761	0.998664	1.000368	1.000125
第 51 节龙身 (m/s)	0.999344	0.998635	0.995074	0.999998	1.004026
第 101 节龙身 (m/s)	0.999089	0.998241	0.994385	0.998482	1.000330
第 151 节龙身 (m/s)	0.998942	0.998039	0.994092	0.998017	0.999446
第 201 节龙身 (m/s)	0.998847	0.997917	0.993930	0.997791	0.999103
龙尾（后）(m/s)	0.998815	0.997877	0.993880	0.997725	0.999011

4.5 问题五模型的建立和求解

4.5.1 舞龙队运动速度分析

舞龙队在设定的路径中共有三段不同运动轨迹：沿等距螺线盘入轨迹 P_1，在调头区域中沿 S 型圆弧调头曲线运动轨迹 P_2、与盘入曲线中心对称的盘出曲线 P_3。在 P_1 上由问题一所求解数据可知：盘入螺线时，舞龙队中龙头速度最大，龙身速度逐节递减；P_2 上的运动为标准圆周运动，各节板凳的运动速度相等；P_3 为盘出曲线，为与盘入曲线相位相差为 π 的等距落线，舞龙队在 P_3 上的运动规律与 P_1 类似，其中龙头速度最小，龙身速度逐节递增。

4.5.2 龙头最大行进速度模型求解

由题目条件得，舞龙队各把手的速度均不超过 2m/s，可确定约束条件为：

$$\max\{v_h, v_{bi}\} \leq 2\text{m/s} \tag{21}$$

舞龙队在设定路径中行进时，由 5.5.1 分析可知龙尾后把手恰好进入盘出曲线时的速度即为舞龙队运动过程中的最大行进速度，使用二分法求解龙头速度不同时的龙尾速度，找到最大速度刚好小于 2m/s 时龙头的速度。计算龙头在约束条件下的最大行进速度为：

$$v_{h\_max} = 1.989837\text{m/s} \tag{22}$$

五、 灵敏度分析

在问题三中，调头空间为一直径恒定的圆，求解了最小螺距使舞龙队能够顺利盘入调头区域。为了探究调头空间圆半径 r_d 的取值对最小螺距 d_{min} 的影响，让调头空间半径 r_d 在 [2.5,5.5] 之间以 $r_d = 0.2$ 的步长变化，研究最小螺距 d_{min} 的变化程度，得到了最小螺距随调头空间圆半径 r_d 变化的折线（左）和其一阶差分折线（右）如图16。

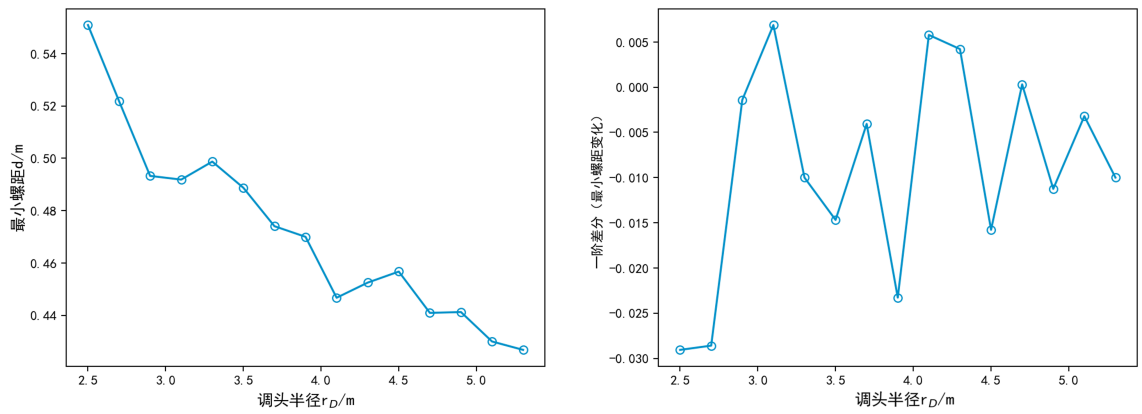

图 **16**　最小螺距随调头空间圆半径变化和其一阶差分折线图

从图16左中可已看出随着调头半径的增大，最小螺距整体呈减小趋势，调头空间半径在区间内变化幅度约为 17%，且由一阶差分折线图16右可以发现最小螺距在调头半径区间 [4.7,5.5] 逐渐趋于平稳。一般来说，在舞龙队可灵活盘入和盘出的前提下，缩小盘龙所需的面积，增加盘龙行进速度，即可提高观赏性。螺距越小盘龙所需面积越小，考虑到上述灵敏度分析，在问题三实际情况中在 [4.7,5.0] 中选择调头空间半径最优。

六、 模型的评价

6.1 模型的优点

1. 问题一和问题二求解速度时考虑到同一板凳上两把手沿板方向上速度分量相等的物理关系简化了速度求解过程；

2. 问题三遍历求解最小螺距时引入二分法的思想降低了时间复杂度；

3. 问题四中通过模拟舞龙队盘出运动找到了更小的调头空间和调头曲线。

6.2 模型的缺点

1. 在判断碰撞时，需要考虑每一节板凳四个顶点与其他非相邻板凳的位置状态，计算量较大且繁杂，使得求解时间较长；

2. 求解更短的调头曲线时仅考虑盘入点与盘出点中心对称的情况，可能导致求解结果不是最短曲线长度，存在可接受误差。

七、 模型的改进与推广

1. 考虑舞龙队发生碰撞的三维空间模型，使结果更贴合实际情况；

2. 提高二分法迭代次数，使用更小步长遍历，提高结果精确度；

3. 本论文所建立的运动模型可以应用于运动轨迹为阿基米德螺线的物理问题。

参考文献

[1] 包志轩, 王盈盈, 周逸轩, 鲍福良. 板凳龙调头路径模型 [J]. 台州学院学报, 2025, 47(3): 7–13.

[2] 李宇航, 巫如山, 张驰, 李明奇. 民俗活动“板凳龙”行进状态建模与路径优化研究 [J]. 实验科学与技术: 1–7 [2026-02-13].

[3] 蔡志杰. 板凳龙运动轨迹模型的分析研究 [J]. 数学建模及其应用, 2025, 14(1): 25–32. DOI:10.19943/j.2095-3070.jmmia.2025.01.03.

[4] 全国大学生数学建模竞赛组委会. 2024 高教社杯全国大学生数学建模竞赛 A 题论文展示（A163）[J]. 实验科学与技术, 2024, 22(4).

[5] JEREMATH. 2024 全国大学生数学建模竞赛 A 题——板凳龙 [EB/OL]. (2024) [2026-02-13]. https://github.com/Jeremath/2024-A.

附录

A 支撑材料文件列表

· 问题一

result1.xlsx——题目要求结果
Question1.py——求解问题一的程序

· 问题二

result2.xlsx——题目要求结果
Question2.py——求解问题二的程序

· 问题三

Question3.py——求解问题三的程序
灵敏度分析.py——探究调头空间圆的半径对最小螺距的影响

· 问题四

result4.xlsx——题目要求结果
Question4.py——求解问题四的程序

· 问题五

Question5.py——求解问题五的程序

· 论文相关材料

2024A.tex——撰写论文的 Tex 代码

B 源代码

Question1.py

```
import numpy as np
import pandas as pd
#参数
n=223
b=0.55
pi=np.pi
t_max=301
v_h=1
d_h=2.86
d_b=1.65

#计算每一时刻龙头位置
def theta(t):
theta=np.sqrt(-4*pi*v_h*t/b+1024*pi*pi)
return theta

theta_head=[]
r_head=[]
head_x=[]
head_y=[]
for t in range(t_max+1):
theta_now=theta(t)
theta_head.append(theta_now)
r_head.append(b*theta_now/(2*pi))
x=r_head[t]*np.cos(theta_now)
y=r_head[t]*np.sin(theta_now)
x=round(x,6)
y=round(y,6)
head_x.append(x)
head_y.append(y)

#牛顿法解方程,求解各点的位置
def f(theta_now,theta_next,d):
return (b*b/4/pi/pi)*(theta_now**2+theta_next**2-2*theta_now*theta_next*
np.cos(theta_now-theta_next))-d**2
def f_prime(theta_now,theta_next,d):
return (b*b/4/pi/pi)*(2*theta_next+2*theta_now*(np.cos(theta_now-theta_next)-
theta_next*np.sin(theta_now-theta_next)))

def newton_method(f,f_prime,d,theta_now,x0,tol=1e-6,max_iter=100):
for i in range(max_iter):
x = x0 - f(theta_now,x0,d) / f_prime(theta_now,x0,d)
```

```
if abs(x - x0) < tol:
break
x0 = x
return x

theta_body_i=np.empty((n,t_max+1))
r_body_i=np.empty((n,t_max+1))
body_i_x=np.empty((n,t_max+1))
body_i_y=np.empty((n,t_max+1))
for t in range(t_max+1):
theta_next=newton_method(f,f_prime,d_h,theta_head[t],theta_head[t]+pi/2)
theta_body_i[0][t]=theta_next
r_body_i[0][t]=theta_next*b/(2*pi)
x=r_body_i[0][t]*np.cos(theta_next)
y=r_body_i[0][t]*np.sin(theta_next)
x=round(x,6)
y=round(y,6)
body_i_x[0][t]=x
body_i_y[0][t]=y

for i in range(1,n):
for t in range(t_max+1):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]+pi/2)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b/(2*pi)
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
x=round(x,6)
y=round(y,6)
body_i_x[i][t]=x
body_i_y[i][t]=y

#求解各点速度
v_body_i=np.empty((n,t_max+1))
for t in range(t_max):
m_0=(head_y[t]-body_i_y[0][t])/(head_x[t]-body_i_x[0][t])
m_now=(theta_head[t]*np.cos(theta_head[t])+np.sin(theta_head[t]))/
(-theta_head[t]*np.sin(theta_head[t])+np.cos(theta_head[t]))
m_next=(theta_body_i[0][t]*np.cos(theta_body_i[0][t])+np.sin(theta_body_i[0][t]))/
(-theta_body_i[0][t]*np.sin(theta_body_i[0][t])+np.cos(theta_body_i[0][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_h*np.cos(alpha_1)/np.cos(alpha_2)
v_next=round(v_next,6)
v_body_i[0][t]=v_next

for i in range(1,n):
```

```
for t in range(t_max):
m_0=(body_i_y[i][t]-body_i_y[i-1][t])/(body_i_x[i][t]-body_i_x[i-1][t])
m_now=(theta_body_i[i-1][t]*np.cos(theta_body_i[i-1][t])+np.sin(theta_body_i[i-1][t]))/
(-theta_body_i[i-1][t]*np.sin(theta_body_i[i-1][t])+np.cos(theta_body_i[i-1][t]))
m_next=(theta_body_i[i][t]*np.cos(theta_body_i[i][t])+np.sin(theta_body_i[i][t]))/
(-theta_body_i[i][t]*np.sin(theta_body_i[i][t])+np.cos(theta_body_i[i][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_body_i[i-1][t]*np.cos(alpha_1)/np.cos(alpha_2)
v_next=round(v_next,6)
v_body_i[i][t]=v_next

#输出固定时间，固定位置的位置速度
time_point=[0,60,120,180,240,300]
num_point=[0,50,100,150,200,222]

for t in time_point:
print(f"time:{t}")
print(f"head:x:{head_x[t]},y:{head_y[t]},v:{v_h}")
for i in num_point:
print(f"body{i}:x:{body_i_x[i][t]},y:{body_i_y[i][t]},v:{v_body_i[i][t]}")

#将结果写入excel
data=[]
data.append(head_x)
data.append(head_y)
for i in range(n):
data.append(body_i_x[i])
data.append(body_i_y[i])
df=pd.DataFrame(data)
df.to_excel('solution1-1.xlsx')

data_v=v_body_i
df=pd.DataFrame(data_v)
df.to_excel('solution1-2.xlsx')
```

Question2.py

```
import numpy as np
import pandas as pd
#参数
n=223
b=0.55
pi=np.pi
t_max=300
```

```
v_h=1
d_h=2.86
d_b=1.65
l_h=3.41
l_b=2.2

#计算每一时刻龙头位置
def theta(t):
theta=np.sqrt(-4*pi*v_h*t/b+1024*pi*pi)
return theta

#牛顿法解方程,求解各点的位置
def f(theta_now,theta_next,d):
return (b*b/4/pi/pi)*(theta_now**2+theta_next**2-2*theta_now*theta_next*
np.cos(theta_now-theta_next))-d**2
def f_prime(theta_now,theta_next,d):
return (b*b/4/pi/pi)*(2*theta_next+2*theta_now*(np.cos(theta_now-theta_next)-
theta_next*np.sin(theta_now-theta_next)))

def newton_method(f,f_prime,d,theta_now,x0,tol=1e-6,max_iter=100):
for i in range(max_iter):
x = x0 - f(theta_now,x0,d) / f_prime(theta_now,x0,d)
if abs(x - x0) < tol:
break
x0 = x
return x

#考虑矩形的4个角
def corner(x1,y1,x2,y2,l):
k=(y2-y1)/(x2-x1)
k_p=-1/k
db=np.sqrt(1+k**2)*0.15
b_0=(x1*y2-x2*y1)/(x1-x2)
b_1=b_0-db
b_2=b_0+db
mid_x=(x1+x2)/2
mid_y=(y1+y2)/2
c_0=mid_y-k_p*mid_x
dc=np.sqrt(1+k_p**2)*(l/2)
c_1=c_0-dc
c_2=c_0+dc
x_a=(c_1-b_1)/(k-k_p)
y_a=k*x_a+b_1
x_b=(c_1-b_2)/(k-k_p)
y_b=k*x_b+b_2
x_c=(c_2-b_1)/(k-k_p)
y_c=k*x_c+b_1
```

```
x_d=(c_2-b_2)/(k-k_p)
y_d=k*x_d+b_2
return x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d

#判断点是否在矩形内
def is_in_rectangle(x,y,x1,y1,x2,y2,l):
k=(y2-y1)/(x2-x1)
b=(x1*y2-x2*y1)/(x1-x2)
d=np.abs(k*x-y+b)/np.sqrt(1+k**2)
if d>0.15:
return False
else:
x0=(k*(y-b)+x)/(1+k**2)
dis_sum=(1+k**2)*(np.abs(x1-x0)+np.abs(x2-x0))
if dis_sum>l:
return False
else:
return True

theta_head=[]
r_head=[]
head_x=[]
head_y=[]
theta_body_i=np.empty((n,2*t_max))
r_body_i=np.empty((n,2*t_max))
body_i_x=np.empty((n,2*t_max))
body_i_y=np.empty((n,2*t_max))
t=0
flag=False
while flag==False:
#计算龙头位置
theta_now=theta(t)
theta_head.append(theta_now)
r_head.append(b*theta_now/(2*pi))
x=r_head[t]*np.cos(theta_now)
y=r_head[t]*np.sin(theta_now)
x=round(x,6)
y=round(y,6)
head_x.append(x)
head_y.append(y)
#计算下一节位置
theta_next=newton_method(f,f_prime,d_h,theta_head[t],theta_head[t]+pi/2)
theta_body_i[0][t]=theta_next
r_body_i[0][t]=theta_next*b/(2*pi)
x=r_body_i[0][t]*np.cos(theta_next)
y=r_body_i[0][t]*np.sin(theta_next)
x=round(x,6)
```

```
y=round(y,6)
body_i_x[0][t]=x
body_i_y[0][t]=y
for i in range(1,n):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]+pi/2)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b/(2*pi)
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
x=round(x,6)
y=round(y,6)
body_i_x[i][t]=x
body_i_y[i][t]=y
#判断是否与龙头碰撞
x1=head_x[t]
y1=head_y[t]
x2=body_i_x[0][t]
y2=body_i_y[0][t]
x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d=corner(x1,y1,x2,y2,l_h)
for i in range(1,n-1):
if(is_in_rectangle(x_a,y_a,body_i_x[i][t],
body_i_y[i][t],body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
print(f"在t={t}时刻，龙头和第{i+1}个板子碰撞")
flag=True
break
elif(is_in_rectangle(x_b,y_b,body_i_x[i][t],
body_i_y[i][t],body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
print(f"在t={t}时刻，龙头和第{i+1}个板子碰撞")
flag=True
break
elif(is_in_rectangle(x_c,y_c,body_i_x[i][t],
body_i_y[i][t],body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
print(f"在t={t}时刻，龙头和第{i+1}个板子碰撞")
flag=True
break
elif(is_in_rectangle(x_d,y_d,body_i_x[i][t],
body_i_y[i][t],body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
print(f"在t={t}时刻，龙头和第{i+1}个板子碰撞")
flag=True
break

#判断是否与龙身碰撞
for i in range(0,n-1):
x1=body_i_x[i][t]
y1=body_i_y[i][t]
x2=body_i_x[i+1][t]
y2=body_i_y[i+1][t]
```

```
x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d=corner(x1,y1,x2,y2,l_b)
for j in range(i+2,n-1):
if(is_in_rectangle(x_a,y_a,body_i_x[j][t],body_i_y[j][t],
body_i_x[j+1][t],body_i_y[j+1][t],l_b)):
print(f"在t={t}时刻，第{i}块板子和第{j+1}个板子碰撞")
flag=True
break
elif(is_in_rectangle(x_b,y_b,body_i_x[j][t],body_i_y[j][t],
body_i_x[j+1][t],body_i_y[j+1][t],l_b)):
print(f"在t={t}时刻，第{i}块板子和第{j+1}个板子碰撞")
flag=True
break
elif(is_in_rectangle(x_c,y_c,body_i_x[j][t],body_i_y[j][t],
body_i_x[j+1][t],body_i_y[j+1][t],l_b)):
print(f"在t={t}时刻，第{i}块板子和第{j+1}个板子碰撞")
flag=True
break
elif(is_in_rectangle(x_d,y_d,body_i_x[j][t],body_i_y[j][t],
body_i_x[j+1][t],body_i_y[j+1][t],l_b)):
print(f"在t={t}时刻，第{i}块板子和第{i+1}个板子碰撞")
flag=True
break
t=t+1
t=t-1
#求解该时刻速度
v_body_i=[]
m_0=(head_y[t]-body_i_y[0][t])/(head_x[t]-body_i_x[0][t])
m_now=(theta_head[t]*np.cos(theta_head[t])+np.sin(theta_head[t]))/(-theta_head[t]*
np.sin(theta_head[t])+np.cos(theta_head[t]))
m_next=(theta_body_i[0][t]*np.cos(theta_body_i[0][t])+np.sin(theta_body_i[0][t]))/
(-theta_body_i[0][t]*np.sin(theta_body_i[0][t])+np.cos(theta_body_i[0][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_h*np.cos(alpha_1)/np.cos(alpha_2)
v_next=round(v_next,6)
v_body_i.append(v_next)
for i in range(1,n):
m_0=(body_i_y[i][t]-body_i_y[i-1][t])/(body_i_x[i][t]-body_i_x[i-1][t])
m_now=(theta_body_i[i-1][t]*np.cos(theta_body_i[i-1][t])+np.sin(theta_body_i[i-1][t]))/
(-theta_body_i[i-1][t]*np.sin(theta_body_i[i-1][t])+np.cos(theta_body_i[i-1][t]))
m_next=(theta_body_i[i][t]*np.cos(theta_body_i[i][t])+np.sin(theta_body_i[i][t]))/
(-theta_body_i[i][t]*np.sin(theta_body_i[i][t])+np.cos(theta_body_i[i][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_body_i[i-1]*np.cos(alpha_1)/np.cos(alpha_2)
v_next=round(v_next,6)
v_body_i.append(v_next)
```

```
print(f"t={t}时,龙头的位置x={head_x[t]},y={head_y[t]},龙头的速度v={v_h}")

data={
   'x': body_i_x[:,t],
   "y": body_i_y[:,t],
   "v": v_body_i
}
df=pd.DataFrame(data)
df.to_excel("solution2.xlsx")

#打印固定位置的信息
num_point=[0,50,100,150,200,222]
print(f"time:{t}")
print(f"head:x:{head_x[t]},y:{head_y[t]},v:1")
for i in num_point:
print(f"body{i}:x:{body_i_x[i][t]},y:{body_i_y[i][t]},v:{v_body_i[i]}")
```

Question3.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
np.seterr(divide='ignore',invalid='ignore')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
#参数
n=223
pi=np.pi
t_max=300
v_h=1
d_h=2.86
d_b=1.65
l_h=3.41
l_b=2.2

#计算每一时刻龙头位置
def theta(t,b):
theta=np.sqrt(-4*pi*v_h*t/b+(32*pi*0.55/b)**2)
return theta

#牛顿法解方程,求解各点的位置
def f(theta_now,theta_next,d,b):
return (b*b/4/pi/pi)*(theta_now**2+theta_next**2-2*theta_now*theta_next*
np.cos(theta_now-theta_next))-d**2
def f_prime(theta_now,theta_next,d,b):
```

```
return (b*b/4/pi/pi)*(2*theta_next+2*theta_now*(np.cos(theta_now-theta_next)-theta_next*
np.sin(theta_now-theta_next)))

def newton_method(f,f_prime,d,theta_now,x0,b,tol=1e-6,max_iter=100):
for i in range(max_iter):
x = x0 - f(theta_now,x0,d,b) / f_prime(theta_now,x0,d,b)
if abs(x - x0) < tol:
break
x0 = x
return x

#考虑矩形的4个角
def corner(x1,y1,x2,y2,l):
k=(y2-y1)/(x2-x1)
k_p=-1/k
db=np.sqrt(1+k**2)*0.15
b_0=(x1*y2-x2*y1)/(x1-x2)
b_1=b_0-db
b_2=b_0+db
mid_x=(x1+x2)/2
mid_y=(y1+y2)/2
c_0=mid_y-k_p*mid_x
dc=np.sqrt(1+k_p**2)*(l/2)
c_1=c_0-dc
c_2=c_0+dc
x_a=(c_1-b_1)/(k-k_p)
y_a=k*x_a+b_1
x_b=(c_1-b_2)/(k-k_p)
y_b=k*x_b+b_2
x_c=(c_2-b_1)/(k-k_p)
y_c=k*x_c+b_1
x_d=(c_2-b_2)/(k-k_p)
y_d=k*x_d+b_2
return x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d

#判断点是否在矩形内
def is_in_rectangle(x,y,x1,y1,x2,y2,l):
k=(y2-y1)/(x2-x1)
b=(x1*y2-x2*y1)/(x1-x2)
d=np.abs(k*x-y+b)/np.sqrt(1+k**2)
if d>0.15:
return False
else:
x0=(k*(y-b)+x)/(1+k**2)
dis_sum=(1+k**2)*(np.abs(x1-x0)+np.abs(x2-x0))
if dis_sum>l:
return False
```

```
else:
return True

#计算碰撞时距离
def dis(b_test):
theta_head=[]
r_head=[]
head_x=[]
head_y=[]
theta_body_i=np.empty((n,2*t_max))
r_body_i=np.empty((n,2*t_max))
body_i_x=np.empty((n,2*t_max))
body_i_y=np.empty((n,2*t_max))
t=0
flag=False
while flag==False:
#计算龙头位置
theta_now=theta(t,b_test)
theta_head.append(theta_now)
r_head.append(b_test*theta_now/(2*pi))
x=r_head[t]*np.cos(theta_now)
y=r_head[t]*np.sin(theta_now)
x=round(x,6)
y=round(y,6)
head_x.append(x)
head_y.append(y)
#计算下一节位置
theta_next=newton_method(f,f_prime,d_h,theta_head[t],theta_head[t]+pi/2,b_test)
theta_body_i[0][t]=theta_next
r_body_i[0][t]=theta_next*b_test/(2*pi)
x=r_body_i[0][t]*np.cos(theta_next)
y=r_body_i[0][t]*np.sin(theta_next)
x=round(x,6)
y=round(y,6)
body_i_x[0][t]=x
body_i_y[0][t]=y
for i in range(1,n):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]+pi/2,b_test)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b_test/(2*pi)
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
x=round(x,6)
y=round(y,6)
body_i_x[i][t]=x
body_i_y[i][t]=y
#判断是否与龙头碰撞
```

```
x1=head_x[t]
y1=head_y[t]
x2=body_i_x[0][t]
y2=body_i_y[0][t]
x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d=corner(x1,y1,x2,y2,l_h)
for i in range(1,n-1):
if(is_in_rectangle(x_a,y_a,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
elif(is_in_rectangle(x_b,y_b,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
elif(is_in_rectangle(x_c,y_c,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
elif(is_in_rectangle(x_d,y_d,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break

#判断是否与龙身碰撞
for i in range(0,n-1):
x1=body_i_x[i][t]
y1=body_i_y[i][t]
x2=body_i_x[i+1][t]
y2=body_i_y[i+1][t]
x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d=corner(x1,y1,x2,y2,l_b)
for j in range(i+2,n-1):
if(is_in_rectangle(x_a,y_a,body_i_x[j][t],body_i_y[j][t],
body_i_x[j+1][t],body_i_y[j+1][t],l_b)):
flag=True
break
elif(is_in_rectangle(x_b,y_b,body_i_x[j][t],body_i_y[j][t],
body_i_x[j+1][t],body_i_y[j+1][t],l_b)):
flag=True
break
elif(is_in_rectangle(x_c,y_c,body_i_x[j][t],body_i_y[j][t],
body_i_x[j+1][t],body_i_y[j+1][t],l_b)):
flag=True
break
elif(is_in_rectangle(x_d,y_d,body_i_x[j][t],body_i_y[j][t],
body_i_x[j+1][t],body_i_y[j+1][t],l_b)):
flag=True
break
```

```
t=t+1
t=t-1
return np.sqrt(head_x[t]**2+head_y[t]**2)

#二分答案
L=0.4
R=0.5
step=0.001
while L<=R:
mid=(L+R)/2
if dis(mid)>4.5:
L=mid+step
else:
R=mid-step
print(R)
#遍历
# b_list=np.arange(0,0.55,0.01)
# r_list=[]
# whlie b in b_list:
#    r_list.append(dis(b))
# plt.scatter(b_list,r_list)
# plt.xlabel('螺距d/m')
# plt.ylabel('碰撞半径r/m')
# plt.title('碰撞半径随螺距变化的关系')
# plt.show()
```

Question4.py

```
import numpy as np
import pandas as pd
import math
#参数
n=223
b=1.7
pi=np.pi
t_max=300
v_h=1
d_h=2.86
d_b=1.65
l_h=3.41
l_b=2.2
R=4.5

#计算盘入每一时刻龙头位置
def theta(t):
```

```
if -4*pi*v_h*t/b+(32*pi*0.55/b)**2<0:
return -1
theta=np.sqrt(-4*pi*v_h*t/b+(32*pi*0.55/b)**2)
return theta

#计算盘出每一时刻龙头位置
def theta_out(t):
theta=np.sqrt(4*pi*v_h*t/b+(2*pi*R/b)**2)
return theta

#牛顿法解方程,求解各点的位置
def f(theta_now,theta_next,d):
return (b*b/4/pi/pi)*(theta_now**2+theta_next**2-2*theta_now*theta_next*
np.cos(theta_now-theta_next))-d**2
def f_prime(theta_now,theta_next,d):
return (b*b/4/pi/pi)*(2*theta_next+2*theta_now*(np.cos(theta_now-theta_next)-
theta_next*np.sin(theta_now-theta_next)))

def newton_method(f,f_prime,d,theta_now,x0,tol=1e-6,max_iter=100):
for i in range(max_iter):
x = x0 - f(theta_now,x0,d) / f_prime(theta_now,x0,d)
if abs(x - x0) < tol:
break
x0 = x
return x

#考虑矩形的4个角
def corner(x1,y1,x2,y2,l):
k=(y2-y1)/(x2-x1)
k_p=-1/k
db=np.sqrt(1+k**2)*0.15
b_0=(x1*y2-x2*y1)/(x1-x2)
b_1=b_0-db
b_2=b_0+db
mid_x=(x1+x2)/2
mid_y=(y1+y2)/2
c_0=mid_y-k_p*mid_x
dc=np.sqrt(1+k_p**2)*(l/2)
c_1=c_0-dc
c_2=c_0+dc
x_a=(c_1-b_1)/(k-k_p)
y_a=k*x_a+b_1
x_b=(c_1-b_2)/(k-k_p)
y_b=k*x_b+b_2
x_c=(c_2-b_1)/(k-k_p)
y_c=k*x_c+b_1
x_d=(c_2-b_2)/(k-k_p)
```

```
y_d=k*x_d+b_2
return x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d

#判断点是否在矩形内
def is_in_rectangle(x,y,x1,y1,x2,y2,l):
k=(y2-y1)/(x2-x1)
b=(x1*y2-x2*y1)/(x1-x2)
d=np.abs(k*x-y+b)/np.sqrt(1+k**2)
if d>0.15:
return False
else:
x0=(k*(y-b)+x)/(1+k**2)
dis_sum=(1+k**2)*(np.abs(x1-x0)+np.abs(x2-x0))
if dis_sum>l:
return False
else:
return True

#根据掉头半径确定两圆弧
def turning(R):
theta_turn=2*pi*R/b
x_1=R*np.cos(theta_turn)
y_1=R*np.sin(theta_turn)
x_2=-R*np.cos(theta_turn)
y_2=-R*np.sin(theta_turn)
x_mid=(x_1+x_2*2)/3
y_mid=(y_1+y_2*2)/3
o_1_x=(x_1+x_mid)/2
o_1_y=(y_1+y_mid)/2
o_2_x=(x_2+x_mid)/2
o_2_y=(y_2+y_mid)/2
r_2=R/3
r_1=r_2*2
return o_1_x,o_1_y,r_1,o_2_x,o_2_y,r_2

#计算掉头空间为R时是否会相撞
#盘出曲线点的递推方程
def g(theta_now,theta_next,d):
return (b*b/4/pi/pi)*((theta_now-pi)**2+(theta_next-pi)**2-2*(theta_now-pi)*
(theta_next-pi)*np.cos(theta_now-theta_next))-d**2
def g_prime(theta_now,theta_next,d):
return (b*b/4/pi/pi)*(2*(theta_next-pi)+2*(theta_now-pi)*(np.cos(theta_now-theta_next)-
(theta_next-pi)*np.sin(theta_now-theta_next)))

def crash(R):
theta_turn=2*pi*R/b
x_2=-R*np.cos(theta_turn)
```

```
y_2=-R*np.sin(theta_turn)
theta_3=newton_method(g,g_prime,d_b,theta_turn+pi,theta_turn+pi*3/2)
x_3=b*(theta_3-pi)*np.cos(theta_3)/(2*pi)
y_3=b*(theta_3-pi)*np.sin(theta_3)/(2*pi)
x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d=corner(x_2,y_2,x_3,y_3,l_h)
theta_head=[]
r_head=[]
head_x=[]
head_y=[]
theta_body_i=np.empty((n,2*t_max))
r_body_i=np.empty((n,2*t_max))
body_i_x=np.empty((n,2*t_max))
body_i_y=np.empty((n,2*t_max))
t=0
flag=False
while True:
#计算龙头位置
theta_now=theta(t)
if theta_now==-1:
t=t+1
break
theta_head.append(theta_now)
r_head.append(b*theta_now/(2*pi))
x=r_head[t]*np.cos(theta_now)
y=r_head[t]*np.sin(theta_now)
x=round(x,6)
y=round(y,6)
head_x.append(x)
head_y.append(y)
#计算下一节位置
theta_next=newton_method(f,f_prime,d_h,theta_head[t],theta_head[t]+pi/2)
theta_body_i[0][t]=theta_next
r_body_i[0][t]=theta_next*b/(2*pi)
x=r_body_i[0][t]*np.cos(theta_next)
y=r_body_i[0][t]*np.sin(theta_next)
x=round(x,6)
y=round(y,6)
body_i_x[0][t]=x
body_i_y[0][t]=y
for i in range(1,n):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]+pi/2)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b/(2*pi)
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
x=round(x,6)
y=round(y,6)
```

```
body_i_x[i][t]=x
body_i_y[i][t]=y
#判断是否与龙头碰撞
for i in range(1,n-1):
if(is_in_rectangle(x_a,y_a,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
elif(is_in_rectangle(x_b,y_b,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
elif(is_in_rectangle(x_c,y_c,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
elif(is_in_rectangle(x_d,y_d,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
t=t+1
t=t-1
return flag

left=0
right=4.5
while left<=right:
mid=(left+right)/2
if crash(mid):
left=mid+0.01
else:
right=mid-0.01
print(f"掉头半径为{left}时，龙身不会相撞")

#计算掉头的时间
t=0
t_0=0
while True:
#计算龙头位置
theta_now=theta(t)
r_now=b*theta_now/(2*pi)
if(r_now<=4.5):
t_0=t
break
t=t+1
print(f"掉头的时间为{t_0}")
```

```
#计算各点的两圆的弧长
theta_turn=2*pi*R/b
x_turn_in=R*np.cos(theta_turn)
y_turn_in=R*np.sin(theta_turn)
x_turn_out=-R*np.cos(theta_turn)
y_turn_out=-R*np.sin(theta_turn)
o_1_x,o_1_y,r_1,o_2_x,o_2_y,r_2=turning(4.5)
mid_x=(o_1_x+2*o_2_x)/3
mid_y=(o_1_y+2*o_2_y)/3
alpha_c1=pi
alpha_c2=pi
s_1=r_1*alpha_c1
s_2=r_2*alpha_c2



#计算点位置旋转后位置
def rotate_point(o_x, o_y, a, b, r, theta):
angle = np.arctan2(o_y - b, o_x - a) + theta
x_prime = a + r * np.cos(angle)
y_prime = b + r * np.sin(angle)
return x_prime, y_prime

#计算各点位置速度
theta_head=np.empty(t_max)
r_head=np.empty(t_max)
head_x=np.empty(t_max)
head_y=np.empty(t_max)
theta_body_i=np.empty((n,t_max))
r_body_i=np.empty((n,t_max))
body_i_x=np.empty((n,t_max))
body_i_y=np.empty((n,t_max))
v_body_i=np.empty((n,t_max))
alpha_t_1=2*np.arcsin(d_h/(2*r_1))
alpha_t_2=2*np.arcsin(d_b/(2*r_1))
alpha_t_3=2*np.arcsin(d_h/(2*r_2))
alpha_t_4=2*np.arcsin(d_b/(2*r_2))
for t in range(6,207):
#盘入阶段
if t<=t_0:
#龙头
theta_now=theta(t)
theta_head[t]=theta_now
r_head[t]=b*theta_now/(2*pi)
head_x[t]=r_head[t]*np.cos(theta_now)
head_y[t]=r_head[t]*np.sin(theta_now)
#龙身
```

```
theta_next=newton_method(f,f_prime,d_h,theta_head[t],theta_head[t]+pi/2)
theta_body_i[0][t]=theta_next
r_body_i[0][t]=theta_next*b/(2*pi)
x=r_body_i[0][t]*np.cos(theta_next)
y=r_body_i[0][t]*np.sin(theta_next)
body_i_x[0][t]=x
body_i_y[0][t]=y
for i in range(1,n):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]+pi/2)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b/(2*pi)
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
body_i_x[i][t]=x
body_i_y[i][t]=y
#速度
m_0=(head_y[t]-body_i_y[0][t])/(head_x[t]-body_i_x[0][t])
m_now=(theta_head[t]*np.cos(theta_head[t])+np.sin(theta_head[t]))/
(-theta_head[t]*np.sin(theta_head[t])+np.cos(theta_head[t]))
m_next=(theta_body_i[0][t]*np.cos(theta_body_i[0][t])+np.sin(theta_body_i[0][t]))/
(-theta_body_i[0][t]*np.sin(theta_body_i[0][t])+np.cos(theta_body_i[0][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_h*np.cos(alpha_1)/np.cos(alpha_2)
v_body_i[0][t]=v_next
for i in range(1,n):
m_0=(body_i_y[i][t]-body_i_y[i-1][t])/(body_i_x[i][t]-body_i_x[i-1][t])
m_now=(theta_body_i[i-1][t]*np.cos(theta_body_i[i-1][t])+np.sin(theta_body_i[i-1][t]))/
(-theta_body_i[i-1][t]*np.sin(theta_body_i[i-1][t])+np.cos(theta_body_i[i-1][t]))
m_next=(theta_body_i[i][t]*np.cos(theta_body_i[i][t])+np.sin(theta_body_i[i][t]))/
(-theta_body_i[i][t]*np.sin(theta_body_i[i][t])+np.cos(theta_body_i[i][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_body_i[i-1][t]*np.cos(alpha_1)/np.cos(alpha_2)
v_body_i[i][t]=v_next
# print(v_next,t)
#圆弧1
elif t>t_0 and t<=t_0+s_1/v_h:
#龙头
s_move=v_h*(t-t_0)
alpha_move=s_move/r_1
head_x[t],head_y[t]=rotate_point(o_1_x,o_1_y,x_turn_in,y_turn_in,r_1,-alpha_move)
#龙身
num1=math.ceil((alpha_move)/alpha_t_2)
body_i_x[0][t],body_i_y[0][t]=rotate_point(o_1_x,o_1_y,head_x[t],head_y[t],r_1,alpha_t_1)
v_body_i[0][t]=v_h
for i in range(1,num1+1):
```

```
body_i_x[i][t],body_i_y[i][t]=rotate_point(o_1_x,o_1_y,body_i_x[i-1][t],
body_i_y[i-1][t],r_1,alpha_t_2)
v_body_i[i][t]=v_body_i[i-1][t]
theta_body_i[num1][t]=theta_turn+np.abs(alpha_move-alpha_t_1-num1*alpha_t_2)
r_body_i[num1][t]=b*theta_body_i[num1][t]/(2*pi)
body_i_x[num1][t]=r_body_i[num1][t]*np.cos(theta_body_i[num1][t])
body_i_y[num1][t]=r_body_i[num1][t]*np.sin(theta_body_i[num1][t])
for i in range(num1+1,n):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]+pi/2)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b/(2*pi)
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
body_i_x[i][t]=x
body_i_y[i][t]=y
m_0=(body_i_y[i][t]-body_i_y[i-1][t])/(body_i_x[i][t]-body_i_x[i-1][t])
m_now=(theta_body_i[i-1][t]*np.cos(theta_body_i[i-1][t])+np.sin(theta_body_i[i-1][t]))/
(-theta_body_i[i-1][t]*np.sin(theta_body_i[i-1][t])+np.cos(theta_body_i[i-1][t]))
m_next=(theta_body_i[i][t]*np.cos(theta_body_i[i][t])+np.sin(theta_body_i[i][t]))/
(-theta_body_i[i][t]*np.sin(theta_body_i[i][t])+np.cos(theta_body_i[i][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_body_i[i-1][t]*np.cos(alpha_1)/np.cos(alpha_2)
v_body_i[i][t]=v_next
#圆弧2
elif t>t_0+s_1/v_h and t<=t_0+s_1/v_h+s_2/v_h:
#龙头
s_move=v_h*(t-t_0-s_1/v_h)
alpha_move=s_move/r_2
head_x[t],head_y[t]=rotate_point(o_2_x,o_2_y,mid_x,mid_y,r_2,-alpha_move)
#龙身
num2=math.ceil((alpha_move)/alpha_t_4)
body_i_x[0][t],body_i_y[0][t]=rotate_point(o_2_x,o_2_y,head_x[t],head_y[t],r_2,alpha_t_3)
v_body_i[0][t]=v_h
for i in range(1,num2+1):
body_i_x[i][t],body_i_y[i][t]=rotate_point(o_2_x,o_2_y,
body_i_x[i-1][t],body_i_y[i-1][t],r_2,alpha_t_4)
v_body_i[i][t]=v_body_i[i-1][t]
num3=math.ceil(alpha_c1/alpha_t_2)
for i in range(num2+1,num2+num3+2):
body_i_x[i][t],body_i_y[i][t]=rotate_point(o_1_x,o_1_y,mid_x,mid_y,r_1,alpha_t_2)
v_body_i[i][t]=v_body_i[i-1][t]
theta_body_i[num2+num3+1][t]=theta_turn+np.abs(alpha_move-alpha_t_3-num2*alpha_t_4)
r_body_i[num2+num3+1][t]=b*theta_body_i[num2+num3+1][t]/(2*pi)
body_i_x[num2+num3+1][t]=r_body_i[num2+num3+1][t]*np.cos(theta_body_i[num2+num3+1][t])
body_i_y[num2+num3+1][t]=r_body_i[num2+num3+1][t]*np.sin(theta_body_i[num2+num3+1][t])
for i in range(num2+num3+2,n):
```

```
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]+pi/2)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b/(2*pi)
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
body_i_x[i][t]=x
body_i_y[i][t]=y
m_0=(body_i_y[i][t]-body_i_y[i-1][t])/(body_i_x[i][t]-body_i_x[i-1][t])
m_now=(theta_body_i[i-1][t]*np.cos(theta_body_i[i-1][t])+np.sin(theta_body_i[i-1][t]))/
(-theta_body_i[i-1][t]*np.sin(theta_body_i[i-1][t])+np.cos(theta_body_i[i-1][t]))
m_next=(theta_body_i[i][t]*np.cos(theta_body_i[i][t])+np.sin(theta_body_i[i][t]))/
(-theta_body_i[i][t]*np.sin(theta_body_i[i][t])+np.cos(theta_body_i[i][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_body_i[i-1][t]*np.cos(alpha_1)/np.cos(alpha_2)
v_body_i[i][t]=v_next
#print(v_next,t)
#盘出阶段:
else:
#龙头
theta_now=theta_out(t-t_0-(s_1+s_2)/v_h)
theta_head[t]=theta_now
r_head[t]=b*theta_now/(2*pi)
head_x[t]=r_head[t]*np.cos(theta_now)
head_y[t]=r_head[t]*np.sin(theta_now)
#龙身
theta_next=newton_method(f,f_prime,d_h,theta_head[t],theta_head[t]-pi/2)
theta_body_i[0][t]=theta_next
r_body_i[0][t]=theta_next*b/(2*pi)
x=r_body_i[0][t]*np.cos(theta_next)
y=r_body_i[0][t]*np.sin(theta_next)
body_i_x[0][t]=x
body_i_y[0][t]=y
m_0=(head_y[t]-body_i_y[0][t])/(head_x[t]-body_i_x[0][t])
m_now=(theta_head[t]*np.cos(theta_head[t])+np.sin(theta_head[t]))/
(-theta_head[t]*np.sin(theta_head[t])+np.cos(theta_head[t]))
m_next=(theta_body_i[0][t]*np.cos(theta_body_i[0][t])+np.sin(theta_body_i[0][t]))/
(-theta_body_i[0][t]*np.sin(theta_body_i[0][t])+np.cos(theta_body_i[0][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_h*np.cos(alpha_1)/np.cos(alpha_2)
v_body_i[0][t]=v_next
num4=0
for i in range(1,n):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]-pi/2)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b/(2*pi)
```

```
if r_body_i[i][t]<R:
num4=i
break
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
body_i_x[i][t]=x
body_i_y[i][t]=y
m_0=(body_i_y[i][t]-body_i_y[i-1][t])/(body_i_x[i][t]-body_i_x[i-1][t])
m_now=(theta_body_i[i-1][t]*np.cos(theta_body_i[i-1][t])+np.sin(theta_body_i[i-1][t]))/
(-theta_body_i[i-1][t]*np.sin(theta_body_i[i-1][t])+np.cos(theta_body_i[i-1][t]))
m_next=(theta_body_i[i][t]*np.cos(theta_body_i[i][t])+np.sin(theta_body_i[i][t]))/
(-theta_body_i[i][t]*np.sin(theta_body_i[i][t])+np.cos(theta_body_i[i][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_body_i[i-1][t]*np.cos(alpha_1)/np.cos(alpha_2)
v_body_i[i][t]=v_next
body_i_x=np.multiply(body_i_x,-1)
body_i_y=np.multiply(body_i_y,-1)
num5=math.ceil(alpha_c2/alpha_t_4)
for i in range(num4,num4+num5):
body_i_x[i][t],body_i_y[i][t]=rotate_point(o_2_x,o_2_y,x_turn_out,y_turn_out,r_2,alpha_t_4)
v_body_i[i][t]=v_body_i[i-1][t]
num3=math.ceil(alpha_c1/alpha_t_2)
for i in range(num4+num5,num4+num5+num3):
body_i_x[i][t],body_i_y[i][t]=rotate_point(o_1_x,o_1_y,mid_x,mid,r_1,alpha_t_2)
v_body_i[i][t]=v_body_i[i-1][t]
theta_body_i[num4+num5+num3-1][t]=theta_turn
r_body_i[num4+num5+num3-1][t]=b*theta_body_i[num4+num5+num3-1][t]/(2*pi)
body_i_x[num4+num5+num3-1][t]=r_body_i[num4+num5+num3-1][t]*
np.cos(theta_body_i[num4+num5+num3-1][t])
body_i_y[num4+num5+num3-1][t]=r_body_i[num4+num5+num3-1][t]*
np.sin(theta_body_i[num4+num5+num3-1][t])
for i in range(num4+num5+num3,n):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]+pi/2)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b/(2*pi)
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
body_i_x[i][t]=x
body_i_y[i][t]=y
m_0=(body_i_y[i][t]-body_i_y[i-1][t])/(body_i_x[i][t]-body_i_x[i-1][t])
m_now=(theta_body_i[i-1][t]*np.cos(theta_body_i[i-1][t])+np.sin(theta_body_i[i-1][t]))/
(-theta_body_i[i-1][t]*np.sin(theta_body_i[i-1][t])+np.cos(theta_body_i[i-1][t]))
m_next=(theta_body_i[i][t]*np.cos(theta_body_i[i][t])+np.sin(theta_body_i[i][t]))/
(-theta_body_i[i][t]*np.sin(theta_body_i[i][t])+np.cos(theta_body_i[i][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
```

```
v_next=v_body_i[i-1][t]*np.cos(alpha_1)/np.cos(alpha_2)
v_body_i[i][t]=v_next
#print(v_next,t)

for t in range(6,207):
head_x[t]=round(head_x[t],6)
head_y[t]=round(head_y[t],6)
for i in range(n):
body_i_x[i][t]=round(body_i_x[i][t],6)
body_i_y[i][t]=round(body_i_y[i][t],6)
v_body_i[i][t]=round(v_body_i[i][t],6)
#输出固定时间，固定位置的位置速度
time_point=[6,56,106,156,206]
num_point=[0,50,100,150,200,222]

for t in time_point:
print(f"time:{t}")
print(f"head:x:{head_x[t]},y:{head_y[t]},v:{v_h}")
for i in num_point:
print(f"body{i}:x:{body_i_x[i][t]},y:{body_i_y[i][t]},v:{v_body_i[i][t]}")

#将结果写入excel
data=[]
data.append(head_x[6:207])
data.append(head_y[6:207])
for i in range(n):
data.append(body_i_x[i][6:207])
data.append(body_i_y[i][6:207])
df=pd.DataFrame(data)
df.to_excel('solution4-1.xlsx')

data_v=[]
for i in range(n):
data_v.append(v_body_i[i][6:207])
df=pd.DataFrame(data_v)
df.to_excel('solution4-2.xlsx')
```

Question5.py

```
import numpy as np
import pandas as pd
#参数
n=223
b=1.7
pi=np.pi
```

```
t_max=301
d_h=2.86
d_b=1.65
R=4.5

#计算每一时刻龙头位置
def theta(t,v_h):
theta=np.sqrt(4*pi*v_h*t/b+(2*pi*R/b)**2)
return theta

#牛顿法解方程,求解各点的位置
def f(theta_now,theta_next,d,v_h):
return (b*b/4/pi/pi)*(theta_now**2+theta_next**2-2*theta_now*theta_next*
np.cos(theta_now-theta_next))-d**2
def f_prime(theta_now,theta_next,d,v_h):
return (b*b/4/pi/pi)*(2*theta_next+2*theta_now*(np.cos(theta_now-theta_next)-
theta_next*np.sin(theta_now-theta_next)))

def newton_method(f,f_prime,d,theta_now,x0,v_h,tol=1e-6,max_iter=100):
for i in range(max_iter):
x = x0 - f(theta_now,x0,d,v_h) / f_prime(theta_now,x0,d,v_h)
if abs(x - x0) < tol:
break
x0 = x
return x

#计算v为v_h时的最大速度
def v_max(v_test):
theta_head=[]
r_head=[]
head_x=[]
head_y=[]
theta_body_i=np.empty((n,2*t_max))
r_body_i=np.empty((n,2*t_max))
body_i_x=np.empty((n,2*t_max))
body_i_y=np.empty((n,2*t_max))
t=0
flag=False
while flag==False:
#计算龙头位置
theta_now=theta(t,v_test)
theta_head.append(theta_now)
r_head.append(b*theta_now/(2*pi))
x=r_head[t]*np.cos(theta_now)
y=r_head[t]*np.sin(theta_now)
head_x.append(x)
head_y.append(y)
```

```
#计算下一节位置
theta_next=newton_method(f,f_prime,d_h,theta_head[t],theta_head[t]-pi/2,v_test)
theta_body_i[0][t]=theta_next
r_body_i[0][t]=theta_next*b/(2*pi)
x=r_body_i[0][t]*np.cos(theta_next)
y=r_body_i[0][t]*np.sin(theta_next)
body_i_x[0][t]=x
body_i_y[0][t]=y
for i in range(1,n):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]-pi/2,v_test)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b/(2*pi)
if r_body_i[i][t]<R:
flag=False
break
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
body_i_x[i][t]=x
body_i_y[i][t]=y
flag=True
t=t+1
t=t-1
# print(f"t={t}")
#计算最大速度
v_body_i=[]
m_0=(head_y[t]-body_i_y[0][t])/(head_x[t]-body_i_x[0][t])
m_now=(theta_head[t]*np.cos(theta_head[t])+np.sin(theta_head[t]))/
(-theta_head[t]*np.sin(theta_head[t])+np.cos(theta_head[t]))
m_next=(theta_body_i[0][t]*np.cos(theta_body_i[0][t])+np.sin(theta_body_i[0][t]))/
(-theta_body_i[0][t]*np.sin(theta_body_i[0][t])+np.cos(theta_body_i[0][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_test*np.cos(alpha_1)/np.cos(alpha_2)
v_body_i.append(v_next)
for i in range(1,n):
m_0=(body_i_y[i][t]-body_i_y[i-1][t])/(body_i_x[i][t]-body_i_x[i-1][t])
m_now=(theta_body_i[i-1][t]*np.cos(theta_body_i[i-1][t])+np.sin(theta_body_i[i-1][t]))/
(-theta_body_i[i-1][t]*np.sin(theta_body_i[i-1][t])+np.cos(theta_body_i[i-1][t]))
m_next=(theta_body_i[i][t]*np.cos(theta_body_i[i][t])+np.sin(theta_body_i[i][t]))/
(-theta_body_i[i][t]*np.sin(theta_body_i[i][t])+np.cos(theta_body_i[i][t]))
alpha_1=np.arctan(np.abs((m_0-m_now)/(1+m_0*m_now)))
alpha_2=np.arctan(np.abs((m_0-m_next)/(1+m_0*m_next)))
v_next=v_body_i[i-1]*np.cos(alpha_1)/np.cos(alpha_2)
v_body_i.append(v_next)
return v_body_i[-1]
#二分答案
left=1
```

```
right=2
step=0.00001
while left<=right:
mid=(left+right)/2
# print(mid)
if v_max(mid)>2:
right=mid-step
else:
left=mid+step
print(f"龙头最大速度为{right}")
```

灵敏度分析.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
np.seterr(divide='ignore',invalid='ignore')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
#参数
n=223
pi=np.pi
t_max=300
v_h=1
d_h=2.86
d_b=1.65
l_h=3.41
l_b=2.2

#计算每一时刻龙头位置
def theta(t,b):
theta=np.sqrt(-4*pi*v_h*t/b+(32*pi*0.55/b)**2)
return theta

#牛顿法解方程,求解各点的位置
def f(theta_now,theta_next,d,b):
return (b*b/4/pi/pi)*(theta_now**2+theta_next**2-2*theta_now*theta_next*
np.cos(theta_now-theta_next))-d**2
def f_prime(theta_now,theta_next,d,b):
return (b*b/4/pi/pi)*(2*theta_next+2*theta_now*(np.cos(theta_now-theta_next)-
theta_next*np.sin(theta_now-theta_next)))

def newton_method(f,f_prime,d,theta_now,x0,b,tol=1e-6,max_iter=100):
for i in range(max_iter):
x = x0 - f(theta_now,x0,d,b) / f_prime(theta_now,x0,d,b)
```

```
if abs(x - x0) < tol:
break
x0 = x
return x

#考虑矩形的4个角
def corner(x1,y1,x2,y2,l):
k=(y2-y1)/(x2-x1)
k_p=-1/k
db=np.sqrt(1+k**2)*0.15
b_0=(x1*y2-x2*y1)/(x1-x2)
b_1=b_0-db
b_2=b_0+db
mid_x=(x1+x2)/2
mid_y=(y1+y2)/2
c_0=mid_y-k_p*mid_x
dc=np.sqrt(1+k_p**2)*(l/2)
c_1=c_0-dc
c_2=c_0+dc
x_a=(c_1-b_1)/(k-k_p)
y_a=k*x_a+b_1
x_b=(c_1-b_2)/(k-k_p)
y_b=k*x_b+b_2
x_c=(c_2-b_1)/(k-k_p)
y_c=k*x_c+b_1
x_d=(c_2-b_2)/(k-k_p)
y_d=k*x_d+b_2
return x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d

#判断点是否在矩形内
def is_in_rectangle(x,y,x1,y1,x2,y2,l):
k=(y2-y1)/(x2-x1)
b=(x1*y2-x2*y1)/(x1-x2)
d=np.abs(k*x-y+b)/np.sqrt(1+k**2)
if d>0.15:
return False
else:
x0=(k*(y-b)+x)/(1+k**2)
dis_sum=(1+k**2)*(np.abs(x1-x0)+np.abs(x2-x0))
if dis_sum>l:
return False
else:
return True

#计算碰撞时距离
def dis(b_test):
theta_head=[]
```

```
r_head=[]
head_x=[]
head_y=[]
theta_body_i=np.empty((n,2*t_max))
r_body_i=np.empty((n,2*t_max))
body_i_x=np.empty((n,2*t_max))
body_i_y=np.empty((n,2*t_max))
t=0
flag=False
while flag==False:
#计算龙头位置
theta_now=theta(t,b_test)
theta_head.append(theta_now)
r_head.append(b_test*theta_now/(2*pi))
x=r_head[t]*np.cos(theta_now)
y=r_head[t]*np.sin(theta_now)
x=round(x,6)
y=round(y,6)
head_x.append(x)
head_y.append(y)
#计算下一节位置
theta_next=newton_method(f,f_prime,d_h,theta_head[t],theta_head[t]+pi/2,b_test)
theta_body_i[0][t]=theta_next
r_body_i[0][t]=theta_next*b_test/(2*pi)
x=r_body_i[0][t]*np.cos(theta_next)
y=r_body_i[0][t]*np.sin(theta_next)
x=round(x,6)
y=round(y,6)
body_i_x[0][t]=x
body_i_y[0][t]=y
for i in range(1,n):
theta_next=newton_method(f,f_prime,d_b,theta_body_i[i-1][t],theta_body_i[i-1][t]+pi/2,b_test)
theta_body_i[i][t]=theta_next
r_body_i[i][t]=theta_next*b_test/(2*pi)
x=r_body_i[i][t]*np.cos(theta_next)
y=r_body_i[i][t]*np.sin(theta_next)
x=round(x,6)
y=round(y,6)
body_i_x[i][t]=x
body_i_y[i][t]=y
#判断是否与龙头碰撞
x1=head_x[t]
y1=head_y[t]
x2=body_i_x[0][t]
y2=body_i_y[0][t]
x_a,y_a,x_b,y_b,x_c,y_c,x_d,y_d=corner(x1,y1,x2,y2,l_h)
for i in range(1,n-1):
```

```
if(is_in_rectangle(x_a,y_a,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
elif(is_in_rectangle(x_b,y_b,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
elif(is_in_rectangle(x_c,y_c,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
elif(is_in_rectangle(x_d,y_d,body_i_x[i][t],body_i_y[i][t],
body_i_x[i+1][t],body_i_y[i+1][t],l_h)):
flag=True
break
t=t+1
t=t-1
return np.sqrt(head_x[t]**2+head_y[t]**2)

#灵敏度分析
b_list=[]
r_list=np.arange(2.5,5.5,0.2)
for r in r_list:
L=0
R=r
step=0.01
while L<=R:
mid=(L+R)/2
# print(mid)
if dis(mid)>r:
L=mid+step
else:
R=mid-step
print(f"调头半径为{r}时,最小螺距{R}")
b_list.append(R)

plt.plot(r_list, b_list, marker='o', markerfacecolor='none', markeredgecolor=(0, 0.57,
    0.79), color=(0, 0.57, 0.79))
plt.xlabel('调头半径r$_D$/m',fontsize=12)
plt.ylabel('最小螺距d/m',fontsize=12)
# plt.title('最小螺距随调头半径变化的关系图',fontsize=16)
plt.show()

# 调头半径和最小螺距数据
r_list_result = [2.5, 2.7, 2.9, 3.1, 3.3, 3.5, 3.7, 3.9, 4.1, 4.3, 4.5, 4.7, 4.9, 5.1, 5.3,
    5.5]
```

```
b_list_result = [0.5509375, 0.521875, 0.4932812500000001, 0.49187500000000006,
    0.49875000000000014,
0.48875000000000013, 0.47406250000000016, 0.47000000000000014, 0.44671875000000016,
0.45250000000000024, 0.4567187500000002, 0.4409375000000002, 0.44125000000000025,
0.43000000000000027, 0.4267968750000002, 0.4167968750000002]

# 计算最小螺距的一阶差分
diff_b_list = np.diff(b_list_result)
diff_r_list = r_list_result[:-1] # 差分后r_list会少一个元素

# 绘制一阶差分图
plt.plot(diff_r_list, diff_b_list, marker='o', markerfacecolor='none', color=(0, 146/255,
    202/255))
plt.xlabel('调头半径r$_D$/m', fontsize=10)
plt.ylabel('一阶差分（最小螺距变化）', fontsize=10)
# plt.title('最小螺距随调头半径变化的一阶差分图', fontsize=14)
# plt.grid(True)
plt.show()
```