基于数值模拟的“板凳龙”运动机理模型研究

摘要

传统民俗文化活动“板凳龙”以木制板凳串联成蜿蜒龙形，龙头引领，龙身和龙尾盘旋成圆盘状，在小空间内快速灵活地盘旋，极具观赏性。本文通过主要研究一 223 节板凳龙沿等距螺线的运动机理，建立了基于数值模拟的运动机理模型，利用运动学约束迭代、微分方程、分离轴定理、基于历史的迭代优化算法进行求解。

针对问题一：首先对题干给定等距螺线，基于几何知识建立极坐标下数学模型。再基于递推法与运动学原理，将板凳视为线段，建立板凳龙身把手的位置与速度递推模型。结合以上模型，确定初值条件，建立微分方程。

利用数值解法，解得从初始时刻到 300 s，舞龙队每秒的速度，以及位置对应的极径和旋转角。将极径和旋转角在直角坐标下表示，解得从初始到 300 s，舞龙队每秒的位置。相应结果已填入 $result1.xlsx$，同时论文内要求体现的结果附于表 2与表 3。

针对问题二：在问题一的基础上，考虑板凳为矩形。基于**分离轴定理**，建立碰撞检测模型。为了优化检测，以极点为圆心，待测板凳后把手极径与螺距的和为半径，划定碰撞检测范围。通过进一步优化检测策略，仅需对龙头与第 1 节龙身进行碰撞检测。

结合问题一的递推模型，利用上述检测方法，检测到碰撞发生时刻，即舞龙队的盘入终止时刻为 **413 s**。利用数值解法，解得从初始时刻到盘入终止时刻，舞龙队每秒的位置与速度。相应结果已填入 $result2.xlsx$，同时论文内要求体现的结果附于表 4。

针对问题三：建立螺距与碰撞半径的数学模型。基于问题二中的碰撞检测模型，首先建立螺距与碰撞半径的函数关系。再利用**基于历史的迭代优化——碰撞半径模型**，以 0.001 的精度在合适的区间内，搜素出碰撞半径小于 4.5 m 对应的最大螺距为 **0.45 m**，即为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的最小螺距。

针对问题四：首先证明在调头空间半径一定时，无法调整圆弧，仍保持各部分相切，使得调头曲线变短。再求解调头空间直径为 9 m 时，调头曲线各参数，进而构造调头曲线模型。结合问题一速度递推模型，构造全过程唯一状态标志 $\xi(\theta)$，修正速度递推模型。

利用数值解法，解得从-100 s 到 100 s，舞龙队每秒的位置与速度。相应结果已填入 $result4.xlsx$，同时论文内要求体现的结果附于表 7与表 8。

针对问题五：基于问题四，建立行进间最大速度与龙头速度的线性模型。在问题四的条件下，解得当龙头行进速度为 1 m/s 时，舞龙队在行进间出现的最大速度为 2.585907 m/s。利用上述模型，解得使得舞龙队各把手的最大速度为 2 m/s 时，对应龙头行进速度为 **0.773423 m/s**，即龙头的最大行进速度。

最后本文对所建模型进行了讨论分析，给出了综合评价。

关键词：板凳龙　数值模拟　碰撞检测　递推

1 问题重述

“板凳龙”，又称“盘龙”，是一种传统民俗文化活动，流行于浙江和福建地区。活动以木制板凳为材料，串联成蜿蜒龙形，龙头引领，龙身和龙尾盘旋成圆盘状。表演中，舞龙者需具备高超技巧，能在小空间内快速灵活地盘旋，以提高观赏性。

某板凳龙由 223 节板凳组成，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（孔的直径）为 5.5 cm，孔的中心距离最近的板头 27.5 cm（如图 1与图 2）。相邻两条板凳通过把手连接（如图 3）。

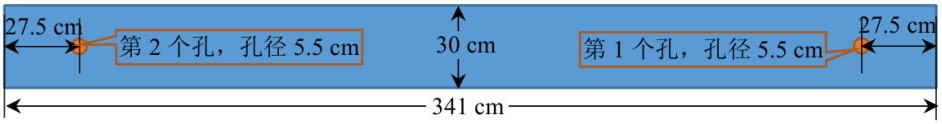

图 **1**　龙头的俯视图

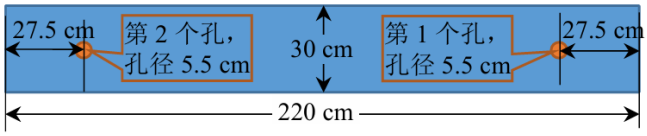

图 **2**　龙身和龙尾的俯视图

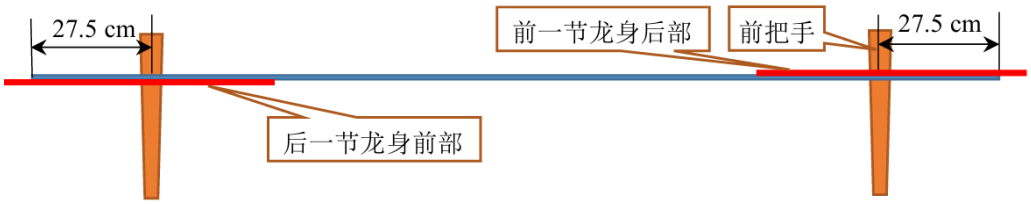

图 **3**　板凳的正视图

根据上述板凳龙的尺寸及表演场地的约束，建立舞龙队在等距螺线上运动的数学模型是本文的研究重点。针对由简至繁的运动约束，我们需要解决以下问题：

问题一：建立舞龙队从既定位置盘入，在已知等距螺线上运动的数学模型，并以此计算舞龙队的位置与速度；

问题二：在问题一的基础上，增加碰撞检测，并以此计算舞龙队盘入的终止时刻，以及舞龙队的位置与速度；

问题三：建立螺距与碰撞半径的数学模型，并以此计算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的最小螺距；

问题四：建立舞龙队在调头空间内调头路径的数学模型，并以此计算使得调头路径尽可能小时，舞龙队的位置与速度；

问题五：在问题四的基础上，计算使得舞龙队各把手的速度均不超过 $2\,m/s$ 时，龙头的最大行进速度。

2 问题分析

2.1 问题一的分析

对于问题一，需要建立舞龙队位置与速度的递推数学模型进行求解，属于机理建模问题。

问题一忽略板凳宽度，仅考虑板凳长度及连接关系，且已给定舞龙队路径、龙头前把手行进速度以及龙头起点。通过几何关系，利用等距螺线的数学描述与余弦定理，建立把手位置递推的数学模型；通过描述把手的运动性质，建立把手速度递推的数学模型。

基于以上两个模型，先求得初始时刻龙头前把手的位置与速度，再利用数值解法，求解从初始到 300 s 为止，每秒整个舞龙队的位置和速度。

2.2 问题二的分析

对于问题二，需要建立碰撞检测的数学模型进行求解，属于机理建模问题。

问题二在问题一的基础上，增加了板凳宽度，此时组成板凳龙的板凳以矩形进行建模。对于多边形的碰撞检测，宜使用分离轴定理。且考虑到对矩形选择方向轴时，两两一组的矩形仅需考虑四条方向轴，故问题二选用分离轴定理作为基本算法进行建模。

基于以上模型，按一定检测顺序，每时刻对矩形进行两两碰撞检测。当首次出现碰撞时，返回当前时刻即为盘入终止时刻；再利用问题一中的模型，求解此时舞龙队的位置和速度。

2.3 问题三的分析

对于问题三，需要建立螺距与碰撞点对应半径的数学模型，求解使得碰撞点位于调头空间边界的最小螺距，属于优化问题。

问题三在考虑问题二中碰撞问题的基础上，释放了固定螺距的约束。由于螺距越小，与碰撞点相切的圆半径应越大。为使舞龙队能顺利盘入至给定的调头空间，所求螺距应为碰撞点刚好落于调头空间边界时对应的螺距。

基于问题二的碰撞检测模型，对螺距遍历一定范围，即可求解一定精度下的最小螺距。

2.4 问题四的分析

对于问题四，需要建立调头空间内路径长度的数学模型，若该路径长不为定值，则考虑解决优化问题；若该路径在已知约束下已无法调整，则属于机理建模问题。

问题四要求基于几何原理，对调头空间内路径进行建模。确定全路径后，利用问题一中建立的位置与速度递推模型，求解从-100 s 到 100 s 为止，每秒整个舞龙队的位置和速度。

2.5 问题五的分析

对于问题五，需要建立舞龙队中最大速度与龙头行进速度的数学模型，属于机理建模问题。

问题五在问题四的基础上，找出当龙头保持匀速前进时，整个舞龙队出现的最大速度，建立该最大速度与龙头行进速度的数学模型。

基于以上模型，求解当最大速度为 2 m/s 时，龙头的前进速度。

3 符号说明

符号	含义	单位
r_i	某点极坐标下极径	m
θ_i	螺线上某点旋转角	rad
p	螺距	m
d	相邻把手距离	m

表 **1**　符号说明

4 模型假设

1. 假设给定等距螺线的螺线中心为坐标原点；
2. 对于问题一，假设板凳是一条没有宽度的线段，且舞龙队尚未通过入口的部分，按题设的等距螺线向外圈依次盘绕；
3. 对于问题二，假设舞龙队尚未通过入口的部分，按题设的等距螺线向外圈依次盘绕。

5 问题一的模型建立与求解

5.1 模型建立

如图 4，根据等距螺线基本数学描述，由螺距 $p = 0.55\,m$，建立盘入螺线路径的数学描述：

$$r = k\theta. \tag{1}$$

$$k = \frac{p}{2\pi} = \frac{0.55}{2\pi}(m). \tag{2}$$

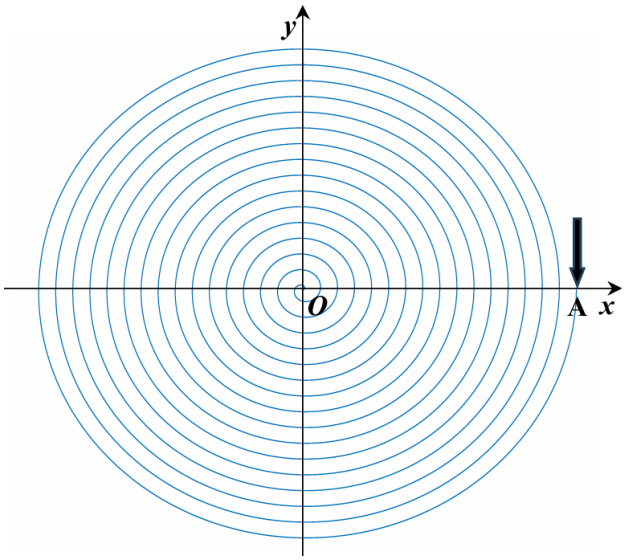

图 4　盘入螺线示意图

5.1.1 位置递推模型

考虑板凳龙前后连接的关系，从龙头前把手到龙尾后把手，设任一把手 i 位矢为 r_i，其下一把手位矢设为 r_{i+1}，两把手之间距离为 d，如图 5, 由余弦定理建立各把手位置递推模型：

$$r_i^2 + r_{i+1}^2 - 2r_i r_{i+1}\cos(\theta_{i+1} - \theta_i) = d^2. \tag{3}$$

由 $r_i = k\theta_i$、$r_{i+1} = k\theta_{i+1}$，可得：

$$\theta_i^2 + \theta_{i+1}^2 - 2\theta_i\theta_{i+1}\cos(\theta_{i+1} - \theta_i) = \frac{d^2}{k^2}. \tag{4}$$

上式可求出各把手在某时刻在极坐标下的极角，利用下式可求得直角坐标下各把手该时刻的位置：

$$x_i = r_i\cos\theta_i,\ y_i = r_i\sin\theta_i. \tag{5}$$

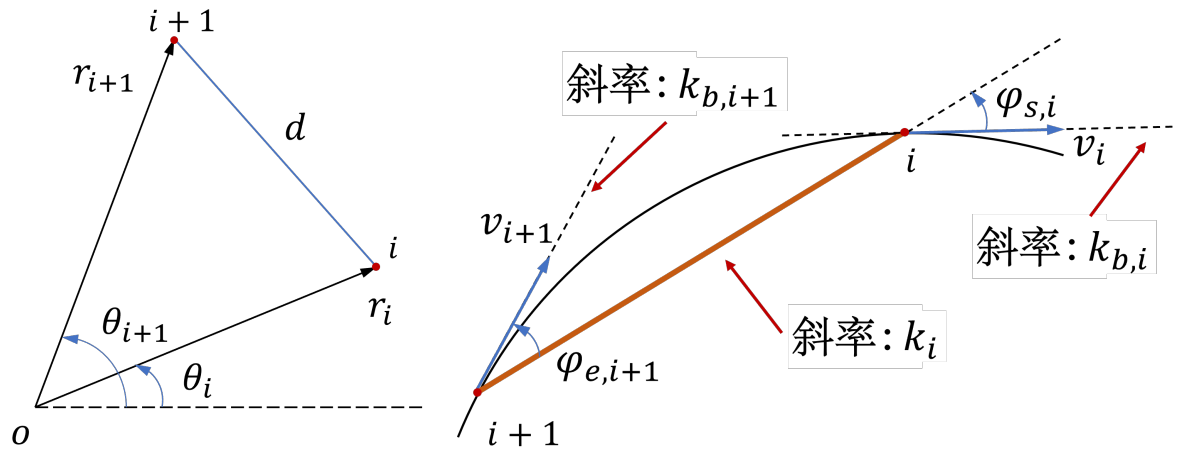

图 **5**　位置（左）与速度（右）递推示意图

5.1.2 速度递推模型

考虑板凳龙前后连接的关系，从龙头前把手到龙尾后把手，设任一把手 i 与把手 $i+1$ 连线, 同把手 i 速度 v_i 所成夹角为 $\varphi_{s,i}$, 同把手 $i+1$ 速度 v_{i+1} 所成夹角为 $\varphi_{e,i}$。把手 i 与把手 $i+1$ 连线斜率为 k_i，把手 i 所在螺线处切线斜率为 $k_{b,i}$，把手 $i+1$ 所在螺线处切线斜率为 $k_{b,i}=1$。把手速度沿同一板凳分量相等，根据运动关系，在位置递推模型的基础上，如图 5, 建立各把手速度递推模型:

根据位置递推模型，有

$$k_{b,i}=\frac{y_{i+1}-y_i}{x_{i+1}-x_i}. \tag{6}$$

$$k_i=\left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_i=\left.\frac{\mathrm{d}y}{\mathrm{d}\theta}\Big/\frac{\mathrm{d}x}{\mathrm{d}\theta}\right|_i=\frac{k\sin\theta_i+k\theta_i\cos\theta_i}{k\cos\theta_i-k\theta_i\sin\theta_i}=\frac{\sin\theta_i+\theta_i\cos\theta_i}{\cos\theta_i-\theta_i\sin\theta_i}. \tag{7}$$

速度关系：

$$v_i\left|\cos\varphi_{s,i}\right|=v_{i+1}\left|\cos\varphi_{e,i+1}\right|. \tag{8}$$

解得：

$$v_{i=1}=v_i\times\left|\frac{\cos\varphi_i}{\cos\varphi_{i+1}}\right|. \tag{9}$$

其中

$$\begin{cases}\varphi_{s,i}=\arctan(k_i)-\arctan(k_{b,i}). & (10)\\ \varphi_{e,i+1}=\arctan(k_{i+1})-\arctan(k_{b,i}). & (11)\end{cases}$$

即获得板凳龙各把手速度递推模型。

5.2 模型求解

首先考虑龙头前把手速度，即递推初值，有

$$v_{r,0}^2+v_{\theta,0}^2=v_0^2=1(m/s)^2. \tag{12}$$

由

$$v_r = \frac{k}{r_0} v_\theta. \tag{13}$$

可得

$$v_{\theta,0} = -\frac{v_0}{\sqrt{(\frac{k}{r_0})^2 + 1}}. \tag{14}$$

然后一般地，考虑龙身以及龙尾，有

$$\begin{aligned} v &= \sqrt{v_r^2 + v_\theta^2} = \sqrt{\dot{r}^2 + r^2\dot{\theta}^2} \\ &= \sqrt{(k^2 + r^2)\dot{\theta}^2} = -k\sqrt{1+\theta^2}\,\frac{\mathrm{d}\theta}{\mathrm{d}t}. \end{aligned} \tag{15}$$

将入口处角度记为最大角度，即 $\theta_m = 16 \times 2\pi$，则

$$\int_{\theta_m}^{\theta} \sqrt{1+\theta^2}\,\mathrm{d}\theta = -\int_0^t \frac{v\mathrm{d}t}{k}. \tag{16}$$

$$\frac{1}{2}\sinh^{-1}(\theta) + \frac{1}{2}\theta\sqrt{1+\theta^2}\Big|_{\theta_m}^{\theta} = -\frac{vt}{k}. \tag{17}$$

$$\frac{1}{2}\sinh^{-1}(\theta) + \frac{1}{2}\sqrt{1+\theta^2} - \frac{1}{2}\sinh^{-1}(\theta_m) - \frac{1}{2}\theta_m\sqrt{1+\theta_m^2} + \frac{vt}{k} = 0. \tag{18}$$

对式 (18) 使用数值解法，可解得 $\theta(t)$。代入递推模型，可解得从初始到 300s 舞龙队各时刻的速度与位置，对应结果已填入 $result1.xlsx$。第 300 s 时，舞龙队位置如图 6。

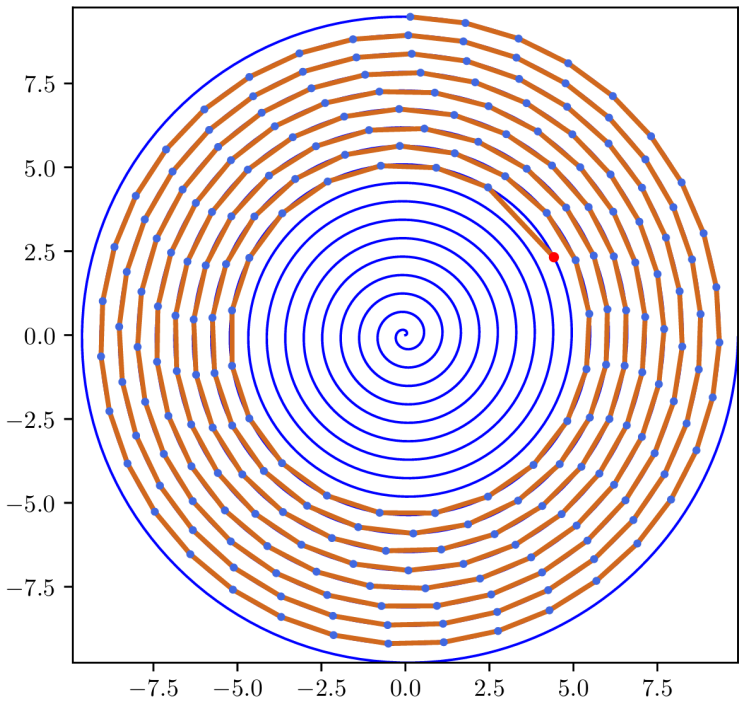

图 **6**　第 **300 s** 舞龙队位置

该问要求论文中体现的结果如表 2、表 3。

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 x (m)	8.800000	5.799209	-4.084887	-2.963609	2.594494	4.420274
龙头 y (m)	0.000000	-5.771092	-6.304479	6.094780	-5.356743	2.320429
第 1 节龙身 x (m)	8.363824	7.456758	-1.445473	-5.237118	4.821221	2.459489
第 1 节龙身 y (m)	2.826544	-3.440399	-7.405883	4.359627	-3.561949	4.402476
第 51 节龙身 x (m)	-9.518732	-8.686317	-5.543150	2.890455	5.980011	-6.301346
第 51 节龙身 y (m)	1.341137	2.540108	6.377946	7.249289	-3.827758	0.465829
第 101 节龙身 x (m)	2.913983	5.687116	5.361939	1.898794	-4.917371	-6.237722
第 101 节龙身 y (m)	-9.918311	-8.001384	-7.557638	-8.471614	-6.379874	3.936008
第 151 节龙身 x (m)	10.861726	6.682311	2.388757	1.005154	2.965378	7.040740
第 151 节龙身 y (m)	1.828754	8.134544	9.727411	9.424751	8.399721	4.393013
第 201 节龙身 x (m)	4.555102	-6.619664	-10.627211	-9.287720	-7.457151	-7.458662
第 201 节龙身 y (m)	10.725118	9.025570	1.359847	-4.246673	-6.180726	-5.263384
龙尾（后）x (m)	-5.305444	7.364557	10.974348	7.383896	3.241051	1.785033
龙尾（后）y (m)	-10.676584	-8.797992	0.843473	7.492371	9.469336	9.301164

表 **2**　问题一位置结果

	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	0.999951	0.999933	0.999901	0.999709
第 51 节龙身 (m/s)	0.999742	0.999662	0.999584	0.999445	0.999223	0.998065
第 101 节龙身 (m/s)	0.999575	0.999453	0.999336	0.999134	0.998821	0.997302
第 151 节龙身 (m/s)	0.999448	0.999299	0.999158	0.998918	0.998554	0.996861
第 201 节龙身 (m/s)	0.999348	0.999180	0.999024	0.998760	0.998364	0.996574
龙尾（后）(m/s)	0.999311	0.999136	0.998974	0.998703	0.998296	0.996478

表 **3**　问题一速度结果

6 问题二的模型建立与求解

6.1 模型建立

在问题一的基础上，使用分离轴定理建立碰撞检测模型。

对于代表任意两板凳的矩形，以矩形各边法线为方向轴，共可作出四条方向轴。将两矩形分别投影至四条方向轴上，若两矩形的投影在三条方向轴上都有重合部分，在剩余一条方向轴上投影线相连，则检测为碰撞；若存在一方向轴上，两矩形的投影无重合部分且不相连，则检测为分离，且该方向轴为分离轴。如图 7。

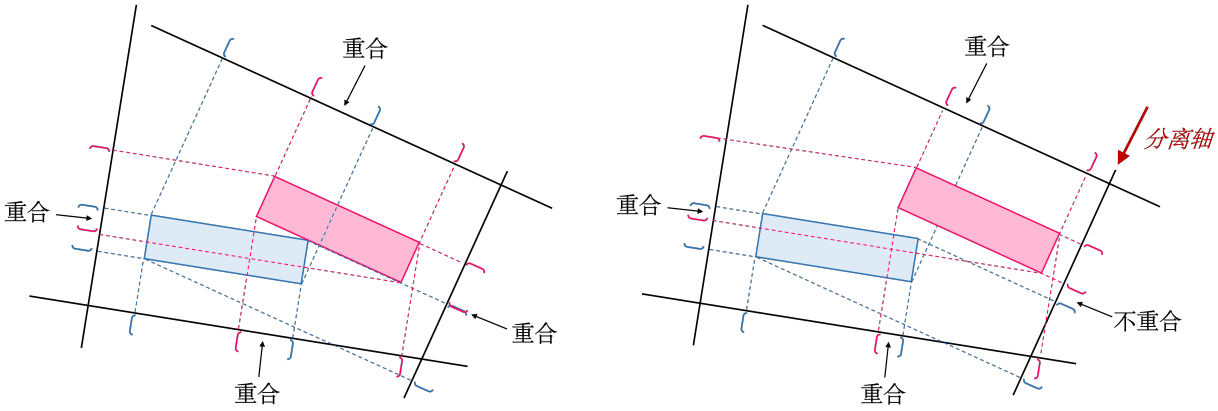

图 7　分离轴定理检测为碰撞（左）、分离（右）时示意图

首先证明碰撞一定发生于龙头或第一节龙身。

假设在某时刻，第 1 节龙身之后的部分于某点发生了碰撞，根据龙身与龙尾各节形态的一致性，其在确定位置上的分布是相同的。故在此时刻之前，当第 1 节龙身行进到该点时，必然已发生碰撞。因此，只需在每时刻对龙头与第 1 节龙身进行碰撞检测。

对于待检测的龙头与第 1 节龙身，需检测碰撞的范围为其所在螺线的本圈与其相邻外圈。为保证检测范围的完整性，以极点为圆心，选取检测半径为：

$$r_d = r + p. \tag{19}$$

其中 r 为待检测板凳后把手对应的极径。则检测范围内，龙身节数为：

$$n = \lceil \frac{2\pi r_d}{d} \rceil. \tag{20}$$

即对于龙头，需对其与第 2 至 $1+n_0$ 节龙身进行碰撞检测；对于第 1 节龙身，需对其与第 3 至 $1+n_1$ 节龙身进行碰撞检测.

6.2 模型求解

利用上述模型进行求解，检测到碰撞发生时刻，即舞龙队盘入终止时刻为 413 s，此时舞龙队的位置如图 8。

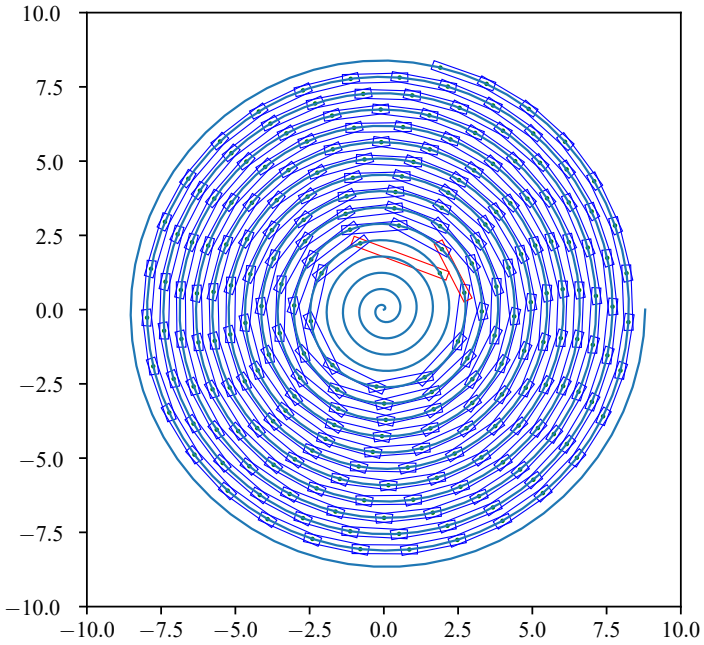

图 **8**　舞龙队终止盘入时位置

利用问题一中的模型求解盘入终止时刻，舞龙队的位置与速度，对应结果已填入 $result2.xlsx$。该问要求论文中体现的结果如表 4。

	横坐标 x (m)	纵坐标 y (m)	速度 (m/s)
龙头	1.296684	1.881044	1.000000
第 1 节龙身	-1.562686	1.820989	0.991479
第 51 节龙身	1.379975	4.293985	0.976725
第 101 节龙身	-0.639337	-5.868276	0.974414
第 151 节龙身	0.865897	-6.969738	0.973472
第 201 节龙身	-7.907337	-1.128122	0.972959
龙尾（后）	1.058929	8.309201	0.972801

表 **4**　问题二位置与速度结果

7 问题三的模型建立与求解

7.1 模型建立

基于问题二中的碰撞检测模型，对于一确定螺距 p，可求得碰撞点处对应半径 r_c。有

$$r_c = r_c(p). \tag{21}$$

进而有碰撞半径与碰撞点对应旋转角呈线性减小关系：

$$\theta_c = \theta_(p). \tag{22}$$

当螺距 $p = 0.41\ m$ 时，利用问题二中的碰撞模型解得碰撞半径 $r_c < 4.5\ m$。由问题二解答可知，当螺距 $p = 0.55\ m$ 时，碰撞半径 $r_c > 4.5\ m$。由于螺距与碰撞半径呈线性增长关系，则碰撞半径 $r_c = 4.5\ m$ 对应的螺距因满足 $0.41\ m < p < 0.55\ m$。

在 0.001 精度下，对 p 在 $(0.410, 0.550)$ 内进行遍历，遍历方法如表 5。

algorithm 基于历史的迭代优化

Def 迭代初始值碰撞半径求解 ( ):

最小螺距, 最大螺距, 步长 = 初始化参数

是否找到临界状态 = False

计算最小螺距的碰撞位置

for 螺距 **in** 从最小螺距到最大螺距 (步长):

获取上一次迭代计算的碰撞位置

碰撞半径 = 计算碰撞半径 (上一次迭代的碰撞位置)

if 碰撞半径第一次小于临界值:

保存临界状态 (螺距, 碰撞半径，碰撞位置)

是否找到临界状态 = True

存储结果 (螺距, 碰撞半径)

保存这一次迭代的碰撞位置

return 是否找到临界状态，临界状态

表 **5** 遍历搜索螺距 p 伪代码

7.2 模型求解

利用以上模型，求解出碰撞半径与螺距的关系如图 9，对于图线上的每一个“阶梯”，表征的是相同两个板凳碰撞的结果。在 0.01 精度下，当碰撞半径为 4.5 m 时，对应螺距 $p = 0.45\ m$，即问题三所求的最小螺距。

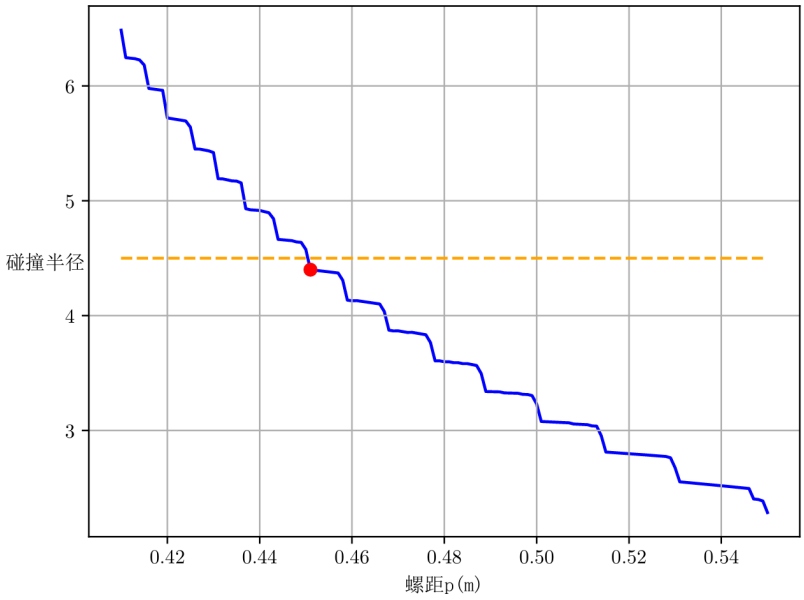

图 **9**　碰撞半径与螺距关系图

8 问题四的模型建立与求解

8.1 模型建立

8.1.1 圆弧路径长模型

由题，盘入与盘出螺线关于螺线中心呈中心对称分布，故二者分别与调头空间的切点也关于螺线中心对称。

如图 10，过盘入切点 B 与盘出切点 F 作螺线在该店切线的法线 l_1 与 l_2，作出由 B 至 F 的圆弧路径，圆弧路径对应圆心分别为 E 与 A，位于 l_1 与 l_2 上。设两圆弧半径分别为 mR 与 R，第一段圆弧对应圆心角为 θ，则：

$$\Delta ADE : (m+1)R\sin(\pi-\theta) = d_0. \tag{23}$$

$$\Delta BCF : (BE+ED+DC)^2 = D^2 - d_0^2. \tag{24}$$

$$[mR+(m+1)R\cos(\pi-\theta)+R]^2 = D^2 - d_0^2. \tag{25}$$

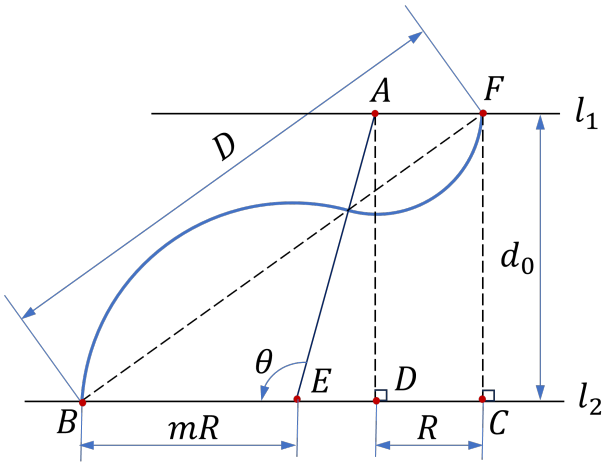

图 **10**　调头曲线示意图

由式 (23) 与式 (24) 化简得：

$$\frac{\sin^2\theta}{(1-\cos\theta)^2}=\frac{d_0^2}{D^2-d_0^2}. \tag{26}$$

由于调头空间位置与直径均为已知，故盘入与盘出螺线与调头空间的切点可确定，进而法线 l_1 与 l_2 可确定，则 l_1 与 l_2 之间的距离 d_0 为一定值。由式 (26) 可知，θ 的值只与 d_0 有关，故此时 θ 的值为定值。

由 (23)

$$(m+1)R=\frac{d_0}{\sin(\pi-\theta)}. \tag{27}$$

可知 $(m+1)R$ 为定值。而弧长

$$L=\theta(m+1)R. \tag{28}$$

其中 θ 与 $(m+1)R$ 均为定值，故弧长 L 的值为定值。当 $m=2$ 时，解得中各参数如表 6。

8.1.2 全过程唯一状态标志描述模型

对于盘入盘出全过程，舞龙队各把手在极坐标下 $\theta\to r$ 不满足双射。考虑构建另一指标 ξ 满足 $\xi\to r$ 双射，以此将对间断多值函数的处理，优化为对连续单值函数的处理。

如图 11，将盘入螺线与调头空间切点、大圆路径与小圆路径切点、过原点作小圆切线的切点、调头空间与盘出螺线切点，分别记为 C_1、C_2、C_3、C_4。

其中，盘入螺线与盘出螺线的方程分别为 $r=k\theta$、$r=k(\theta-\pi)$。由调头空间半径 r_0 为 4.5 m，代入上述式子可求得螺线上两切点 C_1、C_4 对应的 θ_1、θ_4，为唯一确定值：

$$\theta_1=\frac{r_0}{k},\ \theta_4=\frac{r_0}{k}+\pi. \tag{29}$$

参数	数学描述
C_1（大圆与螺线切点）	$(-2.711855863706647, -3.591077522761084)$
C_2（大圆与小圆切点）	$(0.903951954500684, 1.1970258408177847)$
C_3（过原点作小圆切线的切点）	$(2.361918498354993, 1.082285086397807)$
C_4（小圆与螺线切点）	$(2.711855863706647, 3.591077522761084)$
l_1 （C_1 处切线）	$y = 1.17086605x - 0.41585755$
l_2 （C_4 处切线）	$y = 1.17086605x + 0.41585755$
R_1（大圆半径）	3.005417668
R_2（小圆半径）	1.502708834
O_1（大圆圆心）	$(-0.7600091139658178, -1.3057264286867012)$
O_2（小圆圆心）	$(1.7359324888362324, 2.4484019757238924)$

表 **6**　调头路径参数表

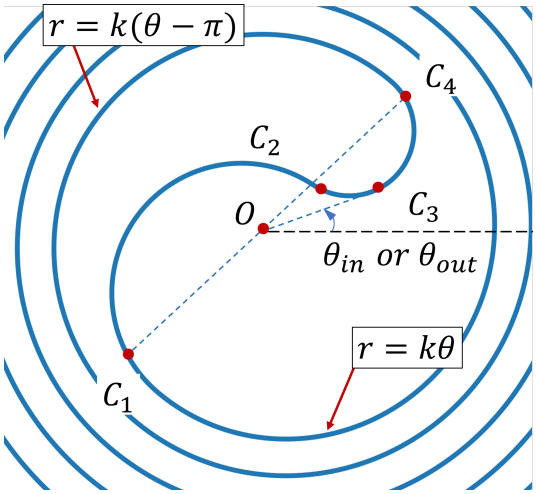

图 **11**　总调头曲线局部示意图

对于 θ_3，经计算，由盘入螺线与盘出螺线出发所得的值不同，记由盘入螺线求得值为 θ_{in}，由盘出螺线求得值为 θ_{out}：

$$\theta_{in} = \theta_1 - [ang(C_3) - ang(C_1) + 2\pi]. \tag{30}$$

$$\theta_{out} = \theta_4 - [ang(C_4) - ang(C_3)]. \tag{31}$$

定义变量 ξ：

$$\xi = \begin{cases} -(\theta - \theta_{in}), \text{ 盘入}. & (32) \\ \theta - \theta_{out}, \text{ 盘出}. & (33) \end{cases}$$

此时 $\xi \to r$ 满足双射

$$\theta = \begin{cases} \xi + \theta_{out},\ \xi \geq 0. & (34) \\ -\xi + \theta_{in},\ \xi < 0. & (35) \end{cases}$$

由此构建了以 ξ 为唯一状态标志的描述模型（如图 12）。

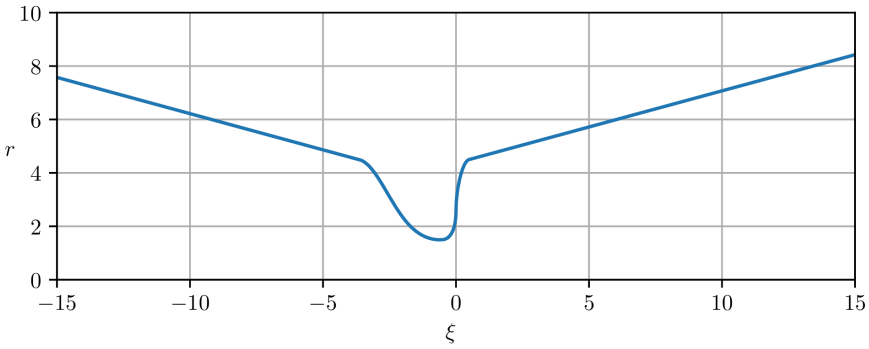

图 **12**　ξ 与 r 关系图

8.1.3 调头曲线模型

对于平面内一以 (x_0, y_0) 为圆心，R 为半径的圆，建立其在极坐标下的数学描述：

由

$$(x - x_0)^2 + (y - y_0)^2 = R^2. \tag{36}$$

有

$$r^2 - 2rx_0\cos\theta - 2ry_0\sin\theta + x_0^2 + y_0^2 - R^2 = 0. \tag{37}$$

$$\begin{aligned} r &= \frac{2(x_0\cos\theta - y_0\sin\theta) \pm \sqrt{4(x_0\cos\theta - y_0\sin\theta)^2 - 4(x_0^2 + y_0^2 - R^2)}}{2} \\ &= (x_0\cos\theta - y_0\sin\theta) \pm \sqrt{R^2 - (x_0\sin\theta - y_0\cos\theta)}. \end{aligned} \tag{38}$$

8.1.4 速度递推模型

根据图 11，将路径分为盘入螺线至 C_1、C_1 至 C_2、C_2 至 C_3、C_3 至 C_4、C_4 至盘出螺线五段，定义路径函数 $f(\xi)$：

$$f(\xi) = \begin{cases} k\theta(\xi),\ \xi \leq 0,\ \theta \geq \theta_1 & (39) \\ r_{O_1}\cos(\theta(\xi) - \theta_{O_1}) + \sqrt{R_1^2 - r_{O_1}^2 + [r_{O_1}\cos(\theta(\xi) - \theta_{O_1})]^2},\ \xi \leq 0,\ \theta_2 \leq \theta \leq \theta_1 & (40) \\ r_{O_2}\cos(\theta(\xi) - \theta_{O_2}) - \sqrt{R_2^2 - r_{O_2}^2 + [r_{O_2}\cos(\theta(\xi) - \theta_{O_2})]^2},\ \xi \leq 0,\ \theta_{in} < \theta < \theta_2 & (41) \\ r_{O_2}\cos(\theta(\xi) - \theta_{O_2}) + \sqrt{R_2^2 - r_{O_2}^2 + [r_{O_2}\cos(\theta(\xi) - \theta_{O_2})]^2},\ \xi > 0,\ \theta_{out} \leq \theta < \theta_4 & (42) \\ k[\theta(\xi) - \pi],\ \xi > 0,\ \theta > \theta_4 & (43) \end{cases}$$

基于问题一的速度递推模型，使用新定义的状态标志 ξ，更改其中 k_i 的值：

位置坐标：

$$\begin{cases} y = f(\xi)\sin\theta(\xi). & (44) \\ x = f(\xi)\cos\theta(\xi). & (45) \end{cases}$$

k_i 更改为：

$$k_i = \left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_i = \left.\frac{\mathrm{d}y}{\mathrm{d}\theta}\Big/\frac{\mathrm{d}x}{\mathrm{d}\theta}\right|_i = \left.\frac{f'(\xi)\xi'(\theta)\sin\theta(\xi) + f(\xi)\cos\theta(\xi)}{f'(\xi)\xi'(\theta)\cos\theta(\xi) - f(\xi)\sin\theta(\xi)}\right|_i. \tag{46}$$

由式 (34) 可知，$\xi'(\theta) = sign(\xi)$。则

$$k_i = \left.\frac{f'(\xi)sign(\xi)\sin\theta(\xi) + f(\xi)\cos\theta(\xi)}{f'(\xi)sign(\xi)\cos\theta(\xi) - f(\xi)\sin\theta(\xi)}\right|_i. \tag{47}$$

对于非螺线段 $f'(\theta)$：

$$f'(\theta) = \begin{cases} -r_{O_1}\sin(\theta-\theta_{O_1}) - \dfrac{r_{O_1}\cos(\theta-\theta_{O_1})\sin(\theta-\theta_{O_1})}{\sqrt{R_1^2 - r_{O_1}^2 + [r_{O_1}\cos(\theta-\theta_{O_1})]}}, \ \xi \le 0, \ \theta_2 \le \theta(\xi) \le \theta_1 & (48) \\ -r_{O_2}\sin(\theta-\theta_{O_2}) + \dfrac{r_{O_2}\cos(\theta-\theta_{O_2})\sin(\theta-\theta_{O_2})}{\sqrt{R_2^2 - r_{O_2}^2 + [r_{O_2}\cos(\theta-\theta_{O_2})]}}, \ \xi \le 0, \ \theta_{in} < \theta(\xi) < \theta_2 & (49) \\ -r_{O_2}\sin(\theta-\theta_{O_2}) - \dfrac{r_{O_2}\cos(\theta-\theta_{O_2})\sin(\theta-\theta_{O_2})}{\sqrt{R_2^2 - r_{O_2}^2 + [r_{O_2}\cos(\theta-\theta_{O_2})]}}, \ \xi > 0, \ \theta_{out} \le \theta(\xi) < \theta_4 & (50) \end{cases}$$

8.2 模型求解

由圆弧路径长模型可知，当调头空间半径一定时，路径长度为一定值，即无法调整圆弧，且仍保持各部分相切时，使得调头曲线变短。

总路径如图 13。

利用上述全过程唯一状态标志描述模型，结合问题一中位置递推模型，以及更新后的速度递推模型，可解得从-100 s 到 100 s 舞龙队各时刻的位置与速度，对应结果已填入 $result4.xlsx$。舞龙队在调头空间内某时刻位置如图 14。

该问要求论文中体现的结果如表 7、表 8。

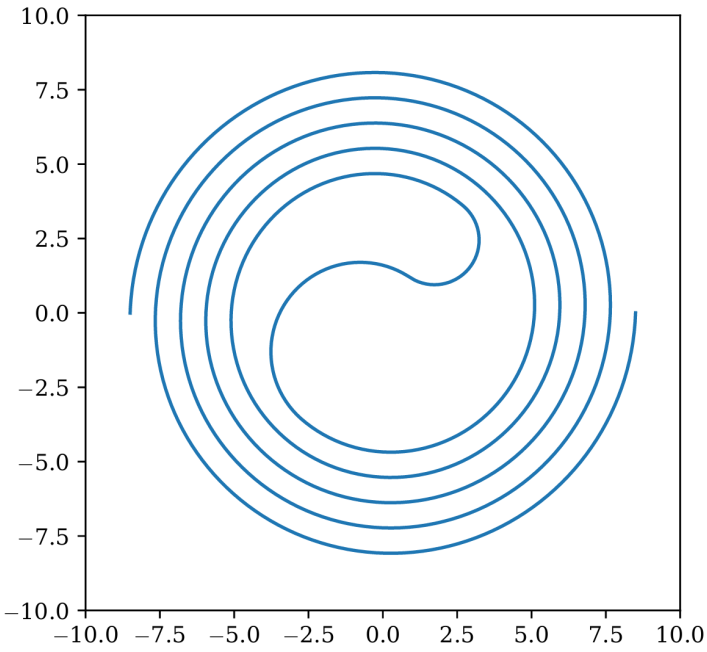

图 **13**　路径示意图

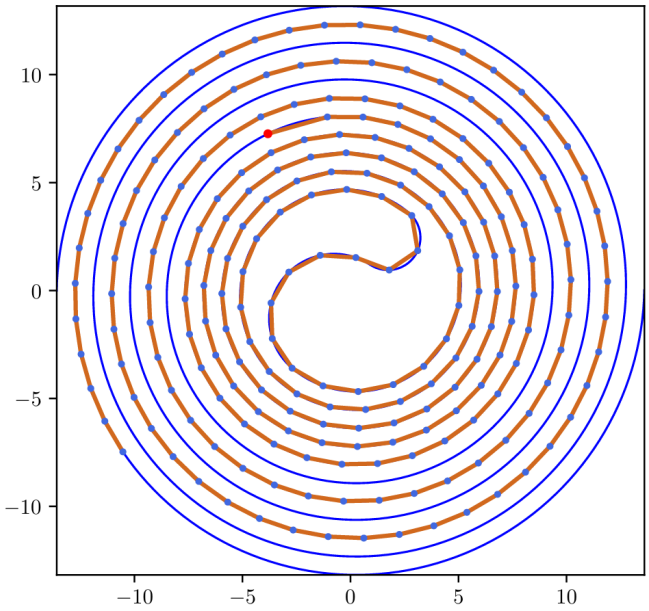

图 **14**　舞龙队在调头空间内某时刻位置示意图

	-100 s	-50 s	0 s	50 s	100 s
龙头 x (m)	7.773796	6.607624	-2.711856	0.628895	-3.822741
龙头 y (m)	3.726780	1.901624	-3.591078	6.316924	7.261329
第 1 节龙身 x (m)	6.202135	5.365178	-0.063534	3.294548	-1.070737
第 1 节龙身 y (m)	6.116233	4.477653	-4.670888	5.280636	8.039836
第 51 节龙身 x (m)	-10.610990	-3.627346	2.459962	-2.438604	1.443005
第 51 节龙身 y (m)	2.821413	-8.964938	-7.778145	-5.772933	4.355633
第 101 节龙身 x (m)	-11.919040	10.127283	3.008493	-7.078237	-6.997242
第 101 节龙身 y (m)	-4.812196	-5.969837	10.108539	5.820132	2.825458
第 151 节龙身 x (m)	-14.349787	12.975637	-7.002789	-5.336959	9.126366
第 151 节龙身 y (m)	-1.991417	-3.807653	10.337482	-10.008402	-4.282732
第 201 节龙身 x (m)	-11.960024	10.524575	-6.872842	-0.486548	9.070249
第 201 节龙身 y (m)	10.559250	-10.805484	12.382609	-13.156004	8.002124
龙尾（后）x (m)	-1.000594	0.186975	-1.933627	6.588237	-11.370684
龙尾（后）y (m)	-16.528381	15.720670	-14.713128	12.229616	-6.055045

表 **7**　问题四位置结果

	-100 s	-50 s	0 s	50 s	100 s
龙头 (m/s)	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身 (m/s)	0.999904	0.999762	0.996529	1.000355	1.000123
第 51 节龙身 (m/s)	0.999346	0.998642	0.992984	1.136741	1.003869
第 101 节龙身 (m/s)	0.999091	0.998248	0.992299	1.134973	1.007580
第 151 节龙身 (m/s)	0.998944	0.998047	0.992008	1.134437	1.006690
第 201 节龙身 (m/s)	0.998849	0.997925	0.991846	1.134179	1.006345
龙尾（后）(m/s)	0.998817	0.997885	0.991796	1.134103	1.006252

表 **8**　问题四速度结果

9 问题五的模型建立与求解

9.1 模型建立

将全路径分为五段，对相邻两节板凳考虑，以龙头与第 1 节龙身为例（余下同理）：

(1) 如图 15，全部位于盘入螺线上时，设螺线上的弦斜率为 k_0，弦两端的切线斜率为 k_1、$k-2$，弦切角为 φ_1、φ_2。

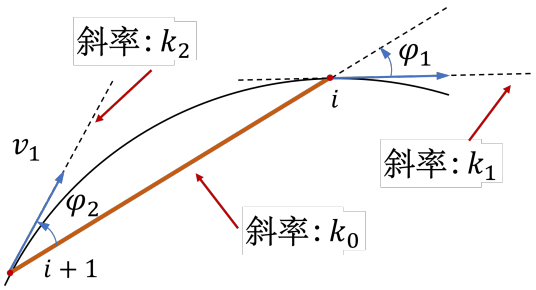

图 **15**　第一段路径示意图

$$\begin{cases} \varphi_1 = \arctan k_0 - \arctan k_1. & (51) \\ \varphi_2 = \arctan k_0 - \arctan k_2. & (52) \end{cases}$$

$$v_0 = \frac{\cos\varphi_1}{\cos\varphi_2} v_1 \tag{53}$$

对于盘入螺线，$\theta_0 < \theta_1$，则

$$\varphi_1 - \varphi_2 = |\arctan k_2| - |\arctan k_1| . \tag{54}$$

螺线切线斜率：

$$k = \frac{\sin\theta + \theta\cos\theta}{\cos\theta - \theta\sin\theta}. \tag{55}$$

由于 $k' > 0$，故 $k_1 > k_2$，则有 $\varphi_1 > \varphi_2$，因此速度 v 递减。

(2) 部分位于大圆上，部分位于盘入螺线上时，v_1 逐渐增大。直到末状态完全进入大圆上时，此时有 $v_0 = v_1$，即二者速度相等。

(3) 二者均位于大圆或小圆上时，根据圆的性质，此时弦切角相等，即 $\varphi_1 = \varphi_2$，则有 $v_0 = v_1$，二者速度始终相等。

(4) 部分位于大圆，部分位于小圆上时，初态为 $\varphi_1 = \varphi_2$。随着行进过程，k_1 与 k_2 均先增大后减小，无法直观判断 $\frac{\cos\varphi_1}{\cos\varphi_2}$ 与 1 的大小关系，故采用模拟的方法求解该段的速度变化规律。

(5) 全部位于盘出螺线上时，同理 (1)，第 1 节龙身速度始终大于龙头速度，但具体变化幅度难以用直观的数学关系进行刻画，故也采用模拟的方法求解。

9.2 模型求解

在问题四的基础上，当龙头以 1 m/s 匀速行进时，求得最大速度为 2.585907 m/s，出现在第 12.25 s。

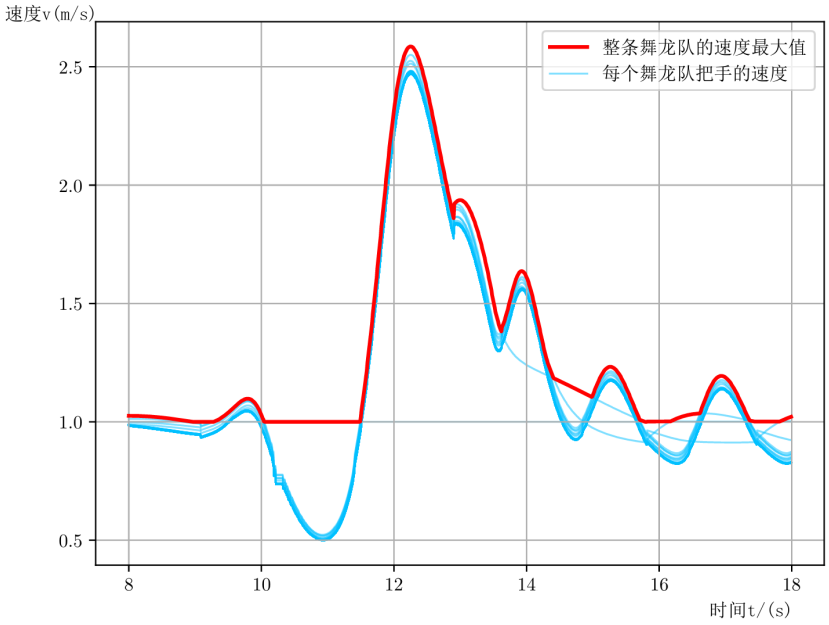

图 **16**　舞龙队速度随时间变化

最大速度与龙头行进速度呈线性关系，故最大速度为 2 m/s 时，龙头的行进速度为：

$$v_0 = \frac{2}{2.585907}\, m/s \approx 0.773423\, m/s. \tag{56}$$

10 模型评价与推广

10.1 模型优点

1. 建立碰撞检测模型时，充分考虑矩形的几何对称特性，选用分离轴定理进行碰撞检测。
2. 选取碰撞检测范围时，选用合适的检测半径对算法进行了优化；通过数学证明碰撞出现的节点，进一步优化了检测过程，使得运行时间大大缩短。
3. 对于问题三，首先粗略确定了最小螺距所在区间，再利用迭代初始值碰撞半径求解模型，遍历搜素解得答案，大大减少了不必要的计算量。
4. 对于问题四，构建了全过程唯一状态指标 ξ，将间断多值函数映射到连续单值函数上，使得求解更简便。

10.2 模型缺点

由于数值模拟的局限性，递推会导致累计误差，使结果与理论值之间存在偏差。

附录 A　问题一——python 源代码

```
import numpy as np
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import pandas as pd
from tqdm import tqdm

class Problem1:

    d_body = 2.2 - 2 * 0.275
    d_head = 3.41 - 2 * 0.275
    v0 = 1
    num = 223
    init_theta0 = 16 * 2 * np.pi

    def __init__(self):
        self.k = np.array(0.55 / (2 * np.pi))

    def set_k(self, k):
        self.k = k

    def next_theta(self, theta, is_head=False):
        """ 角递推"""
        k, d_head, d_body = self.k, self.d_head, self.d_body
        d = d_head if is_head else d_body
        def h(theta_prime):
            return theta ** 2 + theta_prime ** 2 - 2 * theta * theta_prime * np.cos(theta_prime
                - theta) - d ** 2 / k ** 2
        theta_next = fsolve(h, theta + 1)
        theta_next = theta_next[0]
        return theta_next

    def next_v(self, v, theta, theta_next):
        """速度v递推"""
        x, y = self.get_xy(theta)
        x_next, y_next = self.get_xy(theta_next)
        k_board = (y_next - y) / (x_next - x)
        k_curve = (np.sin(theta) + theta * np.cos(theta)) / (np.cos(theta) - theta *
            np.sin(theta))
        k_curve_next = (np.sin(theta_next) + theta_next * np.cos(theta_next)) /
            (np.cos(theta_next) - theta_next * np.sin(theta_next))
        phi = np.arctan(k_curve) - np.arctan(k_board)
        phi_next = np.arctan(k_curve_next) - np.arctan(k_board)
        v_next = np.abs((np.cos(phi) / np.cos(phi_next)) * v)
        return v_next
```

```
def next_state(self, theta, v, is_head=False):
    """递推下一把手状态"""
    theta_next = self.next_theta(theta, is_head)
    v_next = self.next_v(v, theta, theta_next)
    return theta_next, v_next

def get_xy(self, theta):
    """极坐标求直角坐标"""
    k = self.k
    x = k * theta * np.cos(theta)
    y = k * theta * np.sin(theta)
    return x, y

def get_positions_and_velocities(self, theta0):
    """求每个节点的位置和速度"""
    k, v0, num = self.k, self.v0, self.num
    x0, y0 = self.get_xy(theta0)
    result_x = np.array([x0])
    result_y = np.array([y0])
    result_v = np.array([v0])

    theta = theta0
    v = v0

    for i in range(num + 1):
        theta_next, v_next = self.next_state(theta, v, is_head=(i == 0))
        x_next, y_next = self.get_xy(theta_next)
        result_x = np.append(result_x, x_next)
        result_y = np.append(result_y, y_next)
        result_v = np.append(result_v, v_next)
        theta, v = theta_next, v_next

    return result_x, result_y, result_v

def t_to_theta0(self, t):
    """根据时间求龙头位置"""
    k, init_theta0, v0 = self.k, self.init_theta0, self.v0
    def f(theta0_prime):
        return (1/2) * np.arcsinh(theta0_prime) + (1/2) * theta0_prime * np.sqrt(1 +
            theta0_prime ** 2) - (1/2) * np.arcsinh(init_theta0) - (1/2) * init_theta0 *
            np.sqrt(1 + init_theta0 ** 2) + v0 * t / k
    theta0 = fsolve(f, 0)
    return theta0

def get_curve(self, round_num):
    """求螺旋曲线"""
```

```
    k = self.k
    theta = np.arange(0, round_num * 2 * np.pi, 0.01)
    x = k * theta * np.cos(theta)
    y = k * theta * np.sin(theta)
    return x, y

def save_t_fig(self, t, direct):
    """保存时间t的状态图像"""
    theta0 = self.t_to_theta0(t)
    x, y ,v = self.get_positions_and_velocities(theta0)
    x_curve, y_curve = self.get_curve(np.max(np.abs(x)) // (2 * np.pi * self.k) + 2)

    fig, ax = plt.subplots()
    ax.plot(x_curve, y_curve, linewidth=1, zorder=1, color='blue')
    ax.plot(x, y, linewidth=2, zorder=2, color='chocolate')
    ax.scatter(x[1:], y[1:], s=5, zorder=3, color='royalblue')
    ax.scatter(x[0], y[0], s=10, zorder=3, color='red')
    ax.set_xlim(-np.max(np.abs(x_curve)), np.max(np.abs(x_curve)))
    ax.set_ylim(-np.max(np.abs(y_curve)), np.max(np.abs(y_curve)))
    ax.set_aspect('equal', adjustable='box')
    plt.savefig(f"{direct}/state_{t}s.pdf")
    plt.cla()
    plt.clf()
    plt.close()
    print(f"保存t={t}s的图像为state_{t}s.pdf，存放在{direct}文件夹里。")

def save_result(self):
    """保存结果"""
    df_positions = pd.read_excel("result/result1.xlsx", sheet_name="位置")
    df_velocities = pd.read_excel("result/result1.xlsx", sheet_name="速度")

    print("开始计算时间t的状态。")
    for t in tqdm(range(300 + 1), desc="计算时间t状态"):
        theta0 = self.t_to_theta0(t)
        x, y ,v = self.get_positions_and_velocities(theta0)
        for i in range(223 + 1):
            df_positions.loc[2 * i, f"{t} s"] = x[i]
            df_positions.loc[2 * i + 1, f"{t} s"] = y[i]
            df_velocities.loc[i, f"{t} s"] = v[i]
    print("计算完成。")

    df_positions.columns.values[0] = ''
    df_velocities.columns.values[0] = ''

    with pd.ExcelWriter("result/result1.xlsx") as writer:
        df_positions.to_excel(writer, sheet_name="位置", index=False, float_format="%.6f")
        df_velocities.to_excel(writer, sheet_name="速度", index=False, float_format="%.6f")
```

```
        print("已将计算结果保存到problem1/result1.xlsx中。")
```

附录 B　问题二——python 源代码

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np
from shapely.geometry import Polygon
from problem1.problem1 import Problem1
from tqdm import tqdm
import pandas as pd

class Problem2(Problem1):

    def __init__(self):
        super().__init__()
        self.theta0_collision = None
        self.rectangles_collision = None
        self.x_collision = None
        self.y_collision = None
        self.v_collision = None

    def get_rectangle(self, x, y, x_next, y_next):
        nx = (x_next - x) / np.sqrt((x_next - x) ** 2 + (y_next - y) ** 2)
        ny = (y_next - y) / np.sqrt((x_next - x) ** 2 + (y_next - y) ** 2)
        mx = - ny
        my = nx
        anchor_x = x - nx * 0.275 - mx * 0.15
        anchor_y = y - ny * 0.275 - my * 0.15
        width = np.sqrt((x_next - x) ** 2 + (y_next - y) ** 2) + 2 * 0.275
        height = 0.3
        angle = np.rad2deg(np.arctan2(y_next - y, x_next - x))
        rect = patches.Rectangle((anchor_x, anchor_y), width=width, height=height, angle=angle,
            rotation_point="xy", linewidth=0.5, edgecolor='b', facecolor='none', zorder=3)
        return rect

    def get_rectangles(self, x, y):
        rectangles = []
        for i in range(len(x) - 1):
            rect = self.get_rectangle(x[i], y[i], x[i + 1], y[i + 1])
            rectangles.append(rect)
        return rectangles

    def is_overlap(self, rect1, rect2):
        rect1 = Polygon(rect1.get_corners())
```

```
        rect2 = Polygon(rect2.get_corners())
        return rect1.intersects(rect2)

    def check_collision(self, rectangles, theta0):
        for i in range(2, int((2 * np.pi * self.k * theta0 + 4 * np.pi ** 2) / self.d_body)):
            if self.is_overlap(rectangles[0], rectangles[i]):
                rectangles[0].set_edgecolor('r')
                rectangles[i].set_edgecolor('r')
                return True

        for i in range(3, int((2 * np.pi * self.k * self.next_theta(theta0, is_head=True) + 4 *
            np.pi ** 2) / self.d_body)):
            if self.is_overlap(rectangles[1], rectangles[i]):
                rectangles[1].set_edgecolor('r')
                rectangles[i].set_edgecolor('r')
                return True

        return False

    def calc_collision_state(self, previous=False, desc=True):
        start_theta0 = self.theta0_collision if previous else self.init_theta0
        if start_theta0 is None: start_theta0 = self.init_theta0
        it = tqdm(np.flip(np.arange(0, start_theta0, 0.1)), desc=f"计算螺距为{float(self.k * 2 *
            np.pi):.2f}的碰撞点") if desc else np.flip(np.arange(0, start_theta0, 0.1))
        for theta0 in it:
            x, y, v = self.get_positions_and_velocities(theta0)
            rectangles = self.get_rectangles(x, y)
            if self.check_collision(rectangles, theta0):
                self.theta0_collision, self.rectangles_collision, self.x_collision,
                    self.y_collision, self.v_collision = theta0, rectangles, x, y, v
                return

    def get_curve(self, round_num):
        """求螺旋曲线"""
        k = self.k
        theta = np.arange(0, round_num * 2 * np.pi, 0.01)
        x = k * theta * np.cos(theta)
        y = k * theta * np.sin(theta)
        return x, y

    def save_collision_fig(self, direct):
        fig, ax = plt.subplots()
        x_curve, y_curve = self.get_curve(np.max(np.abs(self.x_collision)) // (2 * np.pi *
            self.k) + 2)
        ax.plot(x_curve, y_curve, linewidth=1, zorder=1)
        ax.scatter(self.x_collision, self.y_collision, s=1, color='green', zorder=2)
        for rectangle in self.rectangles_collision:
```

```
            ax.add_patch(rectangle)
        ax.set_xlim(-np.max(np.abs(x_curve)), np.max(np.abs(x_curve)))
        ax.set_ylim(-np.max(np.abs(y_curve)), np.max(np.abs(y_curve)))
        ax.set_aspect('equal', adjustable='box')
        plt.savefig(f"{direct}/collision_state_{self.k * 2 * np.pi : .2f}.pdf")
        plt.cla()
        plt.clf()
        plt.close()
        print(f"保存螺距为{self.k * 2 * np.pi : .2f}碰撞状态图像为collision_state_{self.k * 2 *
            np.pi : .2f}.pdf，存放在{direct}文件夹里。")

    def save_result(self):
        df = pd.read_excel("result/result2.xlsx", sheet_name="Sheet1")

        for i in range(224):
            df.iloc[i, 1] = self.x_collision[i]
            df.iloc[i, 2] = self.y_collision[i]
            df.iloc[i, 3] = self.v_collision[i]

        df.columns.values[0] = ''

        with pd.ExcelWriter("result/result2.xlsx") as writer:
            df.to_excel(writer, sheet_name="Sheet1", index=False, float_format="%.6f")

        print("已将碰撞结果保存到result2.xlsx中。")
```

附录 C　问题三——python 源代码

```
import numpy as np
import time
from tqdm import tqdm
import matplotlib.pyplot as plt
from problem2.problem2 import Problem2


class Problem3(Problem2):

    def __init__(self):
        super().__init__()
        self.pitch_critical = None
        self.collision_radius_critical = None
        self.theta0_critical = None
        self.rectangles_critical = None
        self.x_critical = None
        self.y_critical = None
```

```
        self.v_critical = None
        self.collision_radius = {}

    def get_collision_radius(self, p, savefig=False):
        k = p / (2 * np.pi)
        self.set_k(k)
        self.calc_collision_state(previous=True, desc=False)
        if savefig: self.save_collision_fig("problem3/savefig")
        time.sleep(0.1)
        collision_radius = k * self.theta0_collision
        return collision_radius

    def calc_collision_states(self, pmin, pmax, interval, savefig=False):
        pass_critical = False
        for p in tqdm(np.arange(pmin, pmax, interval), desc="对每个螺距求解碰撞半径"):
            collision_radius = self.get_collision_radius(p, savefig)
            if collision_radius < 4.5 and not pass_critical:
                pass_critical = True
                self.pitch_critical = p
                self.collision_radius_critical = collision_radius
                self.theta0_critical = self.theta0_collision
                self.rectangles_critical = self.rectangles_collision
                self.x_critical = self.x_collision
                self.y_critical = self.y_collision
                self.v_critical = self.v_collision
            self.collision_radius[str(p)] = collision_radius

    def save_pitch_fig(self, direct):
        pitch, collision_radius = [], []
        for key, value in self.collision_radius.items():
            pitch.append(float(key))
            collision_radius.append(value)
        plt.plot(pitch, collision_radius, color="blue", zorder=1)
        plt.scatter(self.pitch_critical, self.collision_radius_critical, color="red", zorder=2)
        plt.plot(pitch, np.ones_like(pitch) * 4.5, linestyle='--', color="orange", zorder=3)
        plt.grid()
        plt.xlabel("螺距p(m)")
        plt.ylabel("碰撞半径", rotation="horizontal", labelpad=10)
        plt.savefig(f"{direct}/pitch_and_collision_radius.pdf")
        plt.cla()
        plt.clf()
        plt.close()
        print(f"保存螺距和碰撞半径关系的图像，存放在{direct}文件夹里。")
```

附录 D　问题四——python 源代码

```python
import numpy as np
from scipy.optimize import fsolve

def Problem4_1():
    # 定义螺线和圆的参数
    p = 1.7 # 螺距
    R = 4.5 # 大圆的半径
    D = 2 * R # D = 2 * R

    # 定义螺线的参数方程
    def spiral(theta):
        r = p * theta / (2 * np.pi)
        x = r * np.cos(theta)
        y = r * np.sin(theta)
        return x, y

    # 定义螺线的导数方程
    def spiral_derivative(theta):
        # r( ) 的导数
        dr_dtheta = p / (2 * np.pi)
        # 计算导数
        dx_dtheta = dr_dtheta * np.cos(theta) - (p * theta / (2 * np.pi)) * np.sin(theta)
        dy_dtheta = dr_dtheta * np.sin(theta) + (p * theta / (2 * np.pi)) * np.cos(theta)
        return dx_dtheta, dy_dtheta

    # 定义求解方程: 螺线与圆的交点
    def equations(theta):
        x, y = spiral(theta)
        return x**2 + y**2 - R**2

    # 初始猜测 theta 值
    theta_guess = 1.0

    # 使用 fsolve 求解交点
    theta_solution = fsolve(equations, theta_guess)
    theta_solution = theta_solution[0]

    # 计算交点坐标
    x_solution, y_solution = spiral(theta_solution)

    # 计算交点处的导数 (dx/d , dy/d )
    dx_dtheta, dy_dtheta = spiral_derivative(theta_solution)

    # 计算交点处的切线斜率 dy/dx
```

```
slope = dy_dtheta / dx_dtheta

# 法线的斜率
normal_slope = -1 / slope

# 法线与x轴夹角
beta = np.arctan(normal_slope)

# 输出螺线的切点坐标
print(f"螺线切点坐标: ({x_solution}, {y_solution})")

# 计算对称点坐标
x_sym, y_sym = -x_solution, -y_solution
print(f"对称点坐标: ({x_sym}, {y_sym})")

# 法线方程: y = m * x + c
# 交点处的法线
c_solution = y_solution - normal_slope * x_solution
print(f"法线方程1: y = {normal_slope} * x + {c_solution}")

# 对称点处的法线
c_sym = y_sym - normal_slope * x_sym
print(f"法线方程2: y = {normal_slope} * x + {c_sym}")

# 计算对称点到法线的距离
distance = abs(normal_slope * x_sym - y_sym) / np.sqrt(normal_slope**2 + 1)

# 定义方程以求解 alpha
def alpha_equation(alpha):
    return (np.sin(alpha)**2) / (1 - np.cos(alpha))**2 - (distance**2) / (D**2 - distance**2)

# 初始猜测 alpha 值
alpha_guess = np.pi / 4

# 使用 fsolve 求解 alpha
alpha_solution = fsolve(alpha_equation, alpha_guess)

# 计算 alpha 对应的弧度值
alpha = alpha_solution[0]
alpha_degrees = np.degrees(alpha)
print(f"alpha 为: {alpha} 弧度, 即 {alpha_degrees} 度")

# 使用 3 * R_1 * sin(alpha) = distance 计算 R_1（小圆半径）
R_1 = distance / (3 * np.sin(alpha))
R_2 = 2 * R_1 # 大圆半径
print(f"小圆半径 R_1: {R_1}")
print(f"大圆半径 R_2: {R_2}")
```

```
    # 计算小圆圆心坐标
    x_small = x_sym - R_1 * np.cos(beta)
    y_small = y_sym - R_1 * np.sin(beta)
    print(f"小圆圆心坐标: ({x_small}, {y_small})")

    # 计算大圆圆心坐标
    x_large = x_solution + R_2 * np.cos(beta)
    y_large = y_solution + R_2 * np.sin(beta)
    print(f"大圆圆心坐标: ({x_large}, {y_large})")

    # 计算大圆和小圆相切点坐标
    x_tangent_large = x_large + R_2 * np.cos(alpha)
    y_tangent_large = y_large + R_2 * np.sin(alpha)
    print(f"大圆和小圆之间的切点坐标: ({x_tangent_large}, {y_tangent_large})")

    # 计算原点处引出的直线与小圆的切点坐标
    # 斜率 m_tangent 为原点到小圆的切线斜率，利用直线和圆相切的条件
    def tangent_slope_equation(m):
        return np.abs(m * x_small - y_small) / np.sqrt(m**2 + 1) - R_1

    # 初始猜测斜率
    m_guess = 1.0 # 下方的切点，选择负斜率

    # 求解斜率 m_tangent
    m_tangent = fsolve(tangent_slope_equation, m_guess)[0]
    print(f"原点到小圆的切线斜率: {m_tangent}")

    # 联立圆的方程和直线方程求解切点坐标
    def circle_line_intersection(x):
        y = m_tangent * x
        return (x - x_small)**2 + (y - y_small)**2 - R_1**2

    # 初始猜测的 x 值
    x_tangent_guess = x_small

    # 使用 fsolve 求解切点的 x 坐标
    x_tangent_small = fsolve(circle_line_intersection, x_tangent_guess)[0]
    y_tangent_small = m_tangent * x_tangent_small

    print(f"原点到小圆的切点坐标: ({x_tangent_small}, {y_tangent_small})")
```

```
import matplotlib.pyplot as plt
import numpy as np

class Problem4_2:
```

```
slope = np.array(1.17086605) # 两条直线的斜率
intercept1 = np.array(-0.41585755) # 直线1的截距
intercept2 = np.array(0.41585755) # 直线2的截距
cut_point1 = np.array((-2.711855863706647, -3.591077522761084)) # 第一个圆和第一段螺线的切点
cut_point2 = np.array((0.903951954500684, 1.1970258408177847)) # 两个圆的切点
cut_point3 = np.array((2.361918498354993, 1.082285086397807)) # 坐标原点对第二个圆的切点
cut_point4 = np.array((2.711855863706647, 3.591077522761084)) # 第二个圆和第二段螺线的切点
R1 = np.array(3.0054176677561504) # 第一个圆的半径
R2 = np.array(1.5027088338780752) # 第二个圆的半径
O1 = np.array((-0.7600091139658178, -1.3057264286867012)) # 第一个圆的圆心
O2 = np.array((1.7359324888362324, 2.4484019757238924)) # 第二个圆的圆心
k = np.array(1.7 / (2 * np.pi))

r_o1 = np.sqrt(O1[0] ** 2 + O1[1] ** 2)
theta_o1 = np.arctan2(O1[1], O1[0])
r_o2 = np.sqrt(O2[0] ** 2 + O2[1] ** 2)
theta_o2 = np.arctan2(O2[1], O2[0])

r1 = np.sqrt(cut_point1[0] ** 2 + cut_point1[1] ** 2)
theta1 = np.array(r1 / k)
r2 = np.sqrt(cut_point2[0] ** 2 + cut_point2[1] ** 2)
theta2 = theta1 - (np.arctan2(cut_point1[1], cut_point1[0]) + 2 * np.pi -
    np.arctan2(cut_point2[1], cut_point2[0]))
r4 = np.sqrt(cut_point4[0] ** 2 + cut_point4[1] ** 2)
theta4 = np.array(r4 / k + np.pi)
r3 = np.sqrt(cut_point3[0] ** 2 + cut_point3[1] ** 2)
theta3_in = theta2 - (np.arctan2(cut_point2[1], cut_point2[0]) - np.arctan2(cut_point3[1],
    cut_point3[0]))
theta3_out = theta4 - (np.arctan2(cut_point4[1], cut_point4[0]) - np.arctan2(cut_point3[1],
    cut_point3[0]))

theta_in = theta3_in
theta_out = theta3_out

def __init__(self):
    pass

def xi_to_theta_value(self, xi):
    theta_in, theta_out = self.theta_in, self.theta_out
    if xi <= 0:
        return - xi + theta_in
    elif xi >= 0:
        return xi + theta_out

def xi_to_theta(self, xi):
    if xi.shape == ():
```

```
        return self.xi_to_theta_value(xi)
    return np.array([self.xi_to_theta_value(xi) for xi in xi])

def theta_to_xi_value(self, theta, state="in"):
    theta_in, theta_out = self.theta_in, self.theta_out
    if state == "in" and theta >= theta_in:
        return - (theta - theta_in)
    elif state == "out" and theta >= theta_out:
        return theta - theta_out
    return np.nan

def theta_to_xi(self, theta, state="in"):
    if theta.shape == ():
        return self.theta_to_xi_value(theta, state)
    return np.array([self.theta_to_xi_value(theta, state) for theta in theta])

def f_value(self, xi):
    theta1, theta2, theta3_in, theta3_out, theta4 = self.theta1, self.theta2,
        self.theta3_in, self.theta3_out, self.theta4
    R1, R2 = self.R1, self.R2
    r_o1, theta_o1, r_o2, theta_o2 = self.r_o1, self.theta_o1, self.r_o2, self.theta_o2
    k = self.k
    r3 = self.r3

    if xi < 0:
        theta = self.xi_to_theta_value(xi)
        if theta > theta1:
            return k * theta
        elif theta2 <= theta <= theta1:
            return r_o1 * np.cos(theta - theta_o1) + np.sqrt(
                R1 ** 2 - r_o1 ** 2 + (r_o1 * np.cos(theta - theta_o1)) ** 2)
        elif theta3_in <= theta <= theta2:
            return r_o2 * np.cos(theta - theta_o2) - np.sqrt(
                R2 ** 2 - r_o2 ** 2 + (r_o2 * np.cos(theta - theta_o2)) ** 2)
    elif xi > 0:
        theta = self.xi_to_theta_value(xi)
        if theta3_out <= theta <= theta4:
            return r_o2 * np.cos(theta - theta_o2) + np.sqrt(
                R2 ** 2 - r_o2 ** 2 + (r_o2 * np.cos(theta - theta_o2)) ** 2)
        elif theta >= theta4:
            return k * (theta - np.pi)
    else:
        return r3

def f(self, xi):
    if xi.shape == ():
        return self.f_value(xi)
```

```
        return np.array([self.f_value(xi) for xi in xi])

    def get_xy_value(self, xi):
        theta = self.xi_to_theta_value(xi)
        f_xi = self.f_value(xi)
        x = f_xi * np.cos(theta)
        y = f_xi * np.sin(theta)
        return x, y

    def get_xy(self, xi):
        if xi.shape == ():
            return self.get_xy_value(xi)
        x = np.array([])
        y = np.array([])
        for xi in xi:
            x_now, y_now = self.get_xy_value(xi)
            x = np.append(x, x_now)
            y = np.append(y, y_now)
        return x, y

    def get_in_and_out(self, theta_max, interval):
        theta_in, theta_out = self.theta_in, self.theta_out
        xi_in = self.theta_to_xi(np.flip(np.arange(theta_in, theta_max + interval, interval)),
            state="in")
        xi_out = self.theta_to_xi(np.arange(theta_out, theta_max + np.pi + interval, interval),
            state="out")
        xi = np.hstack((xi_in, xi_out))
        return xi

    def save_curve(self, direct):
        fig, ax = plt.subplots()

        xi = self.get_in_and_out(5 * 2 * np.pi, 0.001)
        x, y = self.get_xy(xi)
        ax.plot(x, y)

        ax.set_xlim(-np.max(np.abs(x)), np.max(np.abs(x)))
        ax.set_ylim(-np.max(np.abs(y)), np.max(np.abs(y)))
        ax.set_aspect('equal', adjustable='box')
        plt.savefig(f"{direct}/curve.pdf")
        plt.cla()
        plt.clf()
        plt.close()

    def save_r_xi(self, direct):

        xi = self.get_in_and_out(5 * 2 * np.pi, 0.001)
```

```
        r = self.f(xi)

        fig, ax = plt.subplots()
        ax.plot(xi, r)
        ax.set_xlim(-15, 15)
        ax.set_ylim(0, 10)
        ax.set_xlabel("$\\xi$")
        ax.set_ylabel("$r$", rotation="horizontal")
        ax.set_aspect('equal', adjustable='box')
        ax.grid()
        plt.savefig(f"{direct}/r_xi.pdf")
        plt.cla()
        plt.clf()
        plt.close()
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.optimize import fsolve
from tqdm import tqdm
from problem4.problem4_2 import Problem4_2


class Problem4_3(Problem4_2):

    d_head = 3.41 - 2 * 0.275
    d_body = 2.2 - 2 * 0.275
    v0 = 1
    num = 223

    def __init__(self):
        super().__init__()

    def next_xi(self, xi, is_head=False):
        """ 迭代"""
        d = self.d_head if is_head else self.d_body
        def g(xi_prime):
            f_xi = self.f(xi)
            f_xi_prime = self.f(xi_prime)
            return f_xi ** 2 + f_xi_prime ** 2 - 2 * f_xi * f_xi_prime * np.cos(xi_prime - xi) -
                d ** 2
        xi_next = fsolve(g, xi - 1)
        return xi_next

    def next_v(self, v, xi, xi_next):
        """速度v迭代，用显式求导公式"""
```

```
        theta, theta_next = self.xi_to_theta(xi), self.xi_to_theta(xi_next)
        x, y = self.get_xy(xi)
        x_next, y_next = self.get_xy(xi_next)
        f, f_next = self.f(xi), self.f(xi_next)
        diff_f, diff_f_next = self.get_diff_theta(xi), self.get_diff_theta(xi_next)
        k_board = (y_next - y) / (x_next - x)
        k_curve = (diff_f * np.sin(theta) + f * np.cos(theta)) / (diff_f * np.cos(theta) - f *
            np.sin(theta))
        k_curve_next = (diff_f_next * np.sin(theta_next) + f_next * np.cos(theta_next)) /
            (diff_f_next * np.cos(theta_next) - f_next * np.sin(theta_next))
        phi = np.arctan(k_curve) - np.arctan(k_board)
        phi_next = np.arctan(k_curve_next) - np.arctan(k_board)
        v_next = np.abs((np.cos(phi) / np.cos(phi_next)) * v)
        return v_next

    def next_state(self, xi, v, is_head=False):
        """下一个状态迭代"""
        xi_next = self.next_xi(xi, is_head)
        v_next = self.next_v(v, xi, xi_next)
        return xi_next, v_next

    def get_diff_theta_value(self, xi):
        """求对 的导数"""
        theta1, theta2, theta3_in, theta3_out, theta4 = self.theta1, self.theta2,
            self.theta3_in, self.theta3_out, self.theta4
        R1, R2 = self.R1, self.R2
        r_o1, theta_o1, r_o2, theta_o2 = self.r_o1, self.theta_o1, self.r_o2, self.theta_o2
        k = self.k

        if xi < 0:
            theta = self.xi_to_theta_value(xi)
            if theta > theta1:
                return k
            elif theta2 <= theta <= theta1:
                return -r_o1 * np.sin(theta - theta_o1) + (- 2 * r_o1 * np.cos(theta - theta_o1) *
                        np.sin(theta - theta_o1)) / np.sqrt(R1 ** 2 - r_o1 ** 2 + (r_o1 *
                            np.cos(theta - theta_o1)) ** 2)
            elif theta3_in <= theta <= theta2:
                return -r_o2 * np.sin(theta - theta_o2) - (- 2 * r_o2 * np.cos(theta - theta_o2) *
                        np.sin(theta - theta_o2)) / np.sqrt(R2 ** 2 - r_o2 ** 2 + (r_o2 *
                            np.cos(theta - theta_o2)) ** 2)
        elif xi > 0:
            theta = self.xi_to_theta_value(xi)
            if theta3_out <= theta <= theta4:
                return -r_o2 * np.sin(theta - theta_o2) + (- 2 * r_o2 * np.cos(theta - theta_o2) *
                        np.sin(theta - theta_o2)) / np.sqrt(R2 ** 2 - r_o2 ** 2 + (r_o2 *
                            np.cos(theta - theta_o2)) ** 2)
```

```
        elif theta >= theta4:
            return k
    else:
        return 0

def get_diff_theta(self, xi):
    """求对 的导数"""
    if xi.shape == ():
        return self.get_diff_theta_value(xi)
    return np.array([self.get_diff_theta_value(xi) for xi in xi])

def get_diff_xi(self, xi):
    """求对 的导数"""
    return np.sign(xi) * self.get_diff_theta(xi)

def get_positions_and_velocities(self, xi0):
    """求对 的导数"""
    v0, num = self.v0, self.num
    x0, y0 = self.get_xy(xi0)
    result_x = np.array([x0])
    result_y = np.array([y0])
    result_v = np.array([v0])

    xi = xi0
    v = v0

    for i in range(num + 1):
        xi_next, v_next = self.next_state(xi, v, is_head=(i == 0))
        x_next, y_next = self.get_xy(xi_next)
        result_x = np.append(result_x, x_next)
        result_y = np.append(result_y, y_next)
        result_v = np.append(result_v, v_next)
        xi, v = xi_next, v_next

    return result_x, result_y, result_v

def t_to_xi0(self, t):
    xi = self.theta_to_xi(self.theta1, state="in")
    if t == 0: return xi
    target = self.v0 * t
    direct = np.sign(target)
    integ = 0
    d_xi = direct * 0.001
    while direct * integ < direct * target:
        integ += np.sqrt(self.get_diff_xi(xi) ** 2 + self.f(xi) ** 2) * d_xi
        xi += d_xi
    return xi
```

```
def get_t_state(self, t):
    xi0 = self.t_to_xi0(t)
    x, y, v = self.get_positions_and_velocities(xi0)
    return x, y, v

def save_t_state(self, t, direct):
    x, y, v = self.get_t_state(t)
    xi = self.get_in_and_out((np.max(np.abs(x)) // 1.7 + 1) * 2 * np.pi, 0.001)
    x_curve, y_curve = self.get_xy(xi)
    fig, ax = plt.subplots()
    ax.plot(x_curve, y_curve, linewidth=1, zorder=1, color='blue')
    ax.plot(x, y, linewidth=2, zorder=2, color='chocolate')
    ax.scatter(x[1:], y[1:], s=5, zorder=3, color='royalblue')
    ax.scatter(x[0], y[0], s=10, zorder=3, color='red')
    ax.set_xlim(-np.max(np.abs(x_curve)), np.max(np.abs(x_curve)))
    ax.set_ylim(-np.max(np.abs(y_curve)), np.max(np.abs(y_curve)))
    ax.set_aspect('equal', adjustable='box')
    plt.savefig(f"{direct}/state_{t}s.pdf")
    print(f"保存t={t}s的图像为state_{t}s.pdf，存放在{direct}文件夹里。")
    plt.cla()
    plt.clf()
    plt.close()

def save_result(self):
    df_positions = pd.read_excel("result/result4.xlsx", sheet_name="位置")
    df_velocities = pd.read_excel("result/result4.xlsx", sheet_name="速度")

    for t in tqdm(np.arange(-100, 100 + 1, 1),
        desc="计算-100s到100s情况并保存到result4.xlsx中"):
        xi0 = self.t_to_xi0(t)
        x, y, v = self.get_positions_and_velocities(xi0)
        for i in range(223 + 1):
            df_positions.loc[2 * i, str(t) + " s"] = x[i]
            df_positions.loc[2 * i + 1, str(t) + " s"] = y[i]
            df_velocities.loc[i, str(t) + " s"] = v[i]

    df_positions.columns.values[0] = ''
    df_velocities.columns.values[0] = ''

    with pd.ExcelWriter("result/result4.xlsx") as writer:
        df_positions.to_excel(writer, sheet_name="位置", index=False, float_format="%.6f")
        df_velocities.to_excel(writer, sheet_name="速度", index=False, float_format="%.6f")
```

附录 E　问题五——python 源代码

```
import numpy as np
from panel.models.vtk import vtk_cdn
from tqdm import tqdm
import matplotlib.pyplot as plt
from problem4.problem4_3 import Problem4_3

class Problem5(Problem4_3):

    def __init__(self):
        super().__init__()
        self.t = None
        self.v_t = None
        self.vmax_t = None

    def calc_v_t(self, tmin, tmax, interval):
        """计算tmin到tmax的速度情况（也就是每一个t对应的225个节点的v）"""
        self.t = np.arange(tmin, tmax, interval)
        _, _, self.v_t = self.get_t_state(self.t[0])
        for t_now in tqdm(self.t[1:],
                          desc=f"计算从{tmin}s到{tmax}s的速度情况，模拟时间间隔为{interval}"):
            _, _, v_now = self.get_t_state(t_now)
            self.v_t = np.vstack((self.v_t, v_now))
        self.v_t = self.v_t.T

    def calc_vmax_t(self):
        "计算t时刻速度v的最大值"
        self.vmax_t = np.max(self.v_t, axis=0)

    def save_v_and_vmax_t_fig(self, direct):
        plt.plot(self.t, self.vmax_t, color="red", linewidth=2, zorder=2,
                 label="整条舞龙队的速度最大值")
        plt.plot(self.t, self.v_t[0], color="deepskyblue", linewidth=1, alpha=0.5, zorder=1,
                 label="每个舞龙队把手的速度")
        for v_n_t in self.v_t[1:]:
            plt.plot(self.t, v_n_t, color="deepskyblue", linewidth=1, alpha=0.5, zorder=1)
        plt.grid()
        plt.legend()
        plt.xlabel("时间t/(s)", x=0.9)
        plt.ylabel("速度v(m/s)", rotation="horizontal", y = 1)
        plt.tight_layout()
        plt.savefig(f"{direct}/v_and_vmax_t.pdf")
        plt.cla()
        plt.clf()
        plt.close()
```