“盘龙”问题的研究

摘要

针对问题一，由于舞龙路径满足阿基米德螺旋线方程。因此，可以利用螺距和龙头前把手的初始位置构造龙头盘入路径的函数。通过对龙头路径进行微小极限处理，可以将龙头的前把手运动的轨迹近似为以坐标原点的圆心，通过龙头速度求得各时刻距离原点的距离，从而求得龙头的位置信息。进一步通过构造龙头和龙身与龙尾路径的几何关系，得到龙身与龙尾的位置，然后通过对位置函数求导得到螺线运动轨迹的瞬时速度。

针对问题二，首先根据第一问结果和龙头的长度与内圈直径的长度关系，缩小碰撞时刻所在圈数的位置，即第三圈到第十一圈；接着通过**龙格库塔法**拟合预测 300 秒后整个舞龙队的位置和速度，并绘制舞龙队碰撞轨迹过程仿真图；下面分为两种碰撞可能发生的情况，根据任意相邻板凳之间的距离与板凳长度的关系确定碰撞的基础条件，通过龙头一个点的轨迹是否再次经过龙身一条直线的轨迹作为内外层碰撞的条件，综合两个条件确定碰撞具体是时间为**第 413 秒**。

针对问题三，我们以螺线中心为圆心，半径为 4.5 米构建掉头空间边界圆，基于螺线方程，当螺线的半径达到调头空间的边界（半径为 4.5 米）时，此时恒速的龙头前把手对应极径为 4.5m,结合螺距及龙头初始位置（即题目中的 A 点），可以求得当前龙头前把手对应的螺旋角，进而求得最短螺距。

针对问题四，我们设计了一种由两段圆弧构成的 S 形调头路径的模型。通过分析圆弧的几何关系和切线条件，确定圆弧的参数；接着，我们利用参数方程计算了舞龙队在调头过程中的位置和速度；根据梯度下降算法，找到最优圆弧半径，我们确保了调头路径最短且龙头速度恒定。模拟结果显示，能够通过调整圆弧，在保持各部分相切的前提下，使得调头曲线变短。

针对问题五，通过构建动力学模型分析舞龙队在盘旋路径上的运动特性，我们提出了一种速度控制策略，并利用数学优化算法计算出龙头在不同盘旋阶段的允许速度上限。

关键词：阿基米德螺旋线，路径分析，余弦定理，龙格库塔法，数值差分

一、问题重述

1.1 问题背景

“板凳龙”，也被称作“盘龙”，是一种浙闽地区的传统民俗活动。在盘龙表演中，人们将数十甚至上百个板凳头尾相接，龙头引领在前，龙身和龙尾紧随其后，盘入成圆形，组成一条蜿蜒如龙的长队。通过对盘龙过程进行数学建模，分析不同状态下盘龙的运作过程，提升盘龙的观赏价值。

1.2 问题提出

某舞龙队由 223 个板凳组成，其中第 1 节是龙头，接下来的 221 节构成龙身，最后 1 节是龙尾。龙头部分的板凳长度为 341 厘米，龙身和龙尾的板凳长度为 220 厘米，所有板凳的宽度均为 30 厘米。每个板凳上都设有两个孔，孔的直径为 5.5 厘米，孔的中心距离板凳的一端 27.5 厘米，相邻的板凳通过连接杆相互连接。

问题一：在本题中，舞龙队伍沿着一个螺距为 55cm 的等距螺线顺时针盘入，每个把手的中心点都位于螺线上。龙头初始位置位于题中螺线的第 16 圈 A 处，龙头前把手以恒定的速度 1 m/s 前进。计算从开始到 300s 结束时，每一秒钟舞龙队的龙头、龙身和龙尾各前把手及龙尾后把手中心的位置和速度，将结果保存到文件 result1.xlsx 中，并按照表 1 和表 2 格式进行填写。

问题二：本题要求根据问题一中描述的螺线路径，确定舞龙队发生碰撞并停止盘入的具体时刻，并描述在这一时刻舞龙队伍的具体位置和速度，将结果存放到文件 result2.xlsx 中，并按照要求在论文中呈现。

问题三：在本题中，舞龙队从盘入转为盘出的调头过程中需要从顺时针方向转变为逆时针方向。其中调头所需的空间是一个以螺线的中心为圆心，直径为 9 米的圆形区域，要求计算出最小的螺距，以确保龙头的前把手能够顺利沿着相应的螺旋线进入调头空间的边缘。

问题四：在第 3 题的基础上，盘入螺距变为 1.7 米，以调头开始时间为零时刻，盘出的螺线与盘入的螺线呈中心对称。调头路径是由两个相切的圆弧组成的 S 形曲线，第 1 个圆弧的半径是第 2 个圆弧半径的 2 倍，并且这两个圆弧都与盘入和盘出的螺线相切。在以上条件下，判断能否在保持相切性的前提下，通过调整圆弧，缩短调头曲

线的长度。并计算出从−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，并按照要求在论文中呈现。

问题五：龙头行进速度保持不变，舞龙队沿问题四设定的路径行进，计算在此条件下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，由于盘龙过程按照等距螺线顺时针盘入，舞龙路径满足阿基米德螺旋线方程。因此，可以利用螺距和龙头前把手的初始位置构造龙头盘入路径的函数，求得龙头位置信息，通过各时刻龙头位置信息推导出其余节点位置信息。具体应将龙头路径进行微小极限处理，可以将龙头的前把手运动的轨迹绘制为以坐标原点的圆心，通过龙头速度求得各时刻距离原点的距离，从而求得龙头的位置信息。进一步通过构造龙头和龙身与龙尾路径的几何关系，得到龙身与龙尾的位置，然后通过对位置函数求导得到螺线运动轨迹的瞬时速度。

2.2 问题二的分析

问题二要求我们确定舞龙队盘入的终上时刻，即判断板凳发生碰撞的条件。我们可以初步定性分析，分析出不可能到达的圈层以及是否是龙头先发生碰撞，然后定量进行求解，考虑到舞龙队的运动为一个连续的过程，因此需要在问题一结果的基础上，选择适当的时间步长进行路径更新，这里则需要构建模型来求解更新策略。对于碰撞而言，由于舞龙队是沿着阿基米德螺旋线运动的，且板凳具有一定宽度，因此需要根据几何方法，确定何时板凳之间的最小距离小于安全距离，从而求得舞龙队盘入的终止时刻。

2.3 问题三的分析

问题三要求计算舞龙队盘入到调头空间边界的最小螺距，即在直径为 9 m 的调头空间里，龙头需要盘入到螺线的某一圈，使其达到调头空间的边界。可以通过设定一个极限半径（4.5 m），并建立极限半径和极角的模型函数，通过调节螺旋线的螺距，计算出对应的螺线方程，使得龙头的行进轨迹最终到达调头空间边界。并通过几何关系进行结果的验证，初步设想的几何关系为调转前后两圆半径比为 1：2。

2.4 问题四的分析

在问题四中，首先分析圆弧的几何关系以及切线条件，从而确定圆弧的参数，确保两段圆弧在连接点的切线方向相同，满足相切条件。然后，利用参数方程计算了舞龙队在调头过程中龙头前把手的位置和速度。运用梯度下降算法，进一步优化了圆弧半径，以最小化调头路径长度。

2.5 问题五的分析

在问题五的研究中，我们通过分析舞龙队在盘旋路径上的运动特性，设计一种速度控制策略。首先，构建一个动力学模型，用于模拟舞龙队在盘旋路径上的运动。通过这一模型，我们能够分析龙头在不同盘旋阶段的速度变化，并预测可能的速度范围。为了控制速度，我们运用数学优化算法，计算出在保证舞龙表演流畅性的同时，各把手速度不超过 2 m/s 的龙头允许速度上限。

三、 模型假设

1. 该螺旋轨迹的初始螺旋极径$a = 0$

2. 计算机底层的浮点运算误差忽略不计

3. 板凳龙整体钢性板凳均为刚性连接，形状不会变形。即每节板凳之间的长度保持不变，无论舞龙队如何移动，板凳的结构不会发生弯曲或变形。

4. 不考虑周围环境情况，如：风速、地形

四、 符号说明

符号	说明	单位
r	极径，即从原点到曲线上某点的距离	m
b	控制螺线收缩或膨胀的参数	m/rad
θ	板凳运动时的极角	rad
s	板凳所走路径的弧长	m
a	$\theta = 0$时的初始极径	m
v	板凳的行进速度	m/s
h	路径更新的时间步长	s
t_n	当前时间点	s
δ	安全距离	m

五、模型建立与求解

5.1 问题一模型的建立与求解

问题一的整体求解过程如图 1 所示

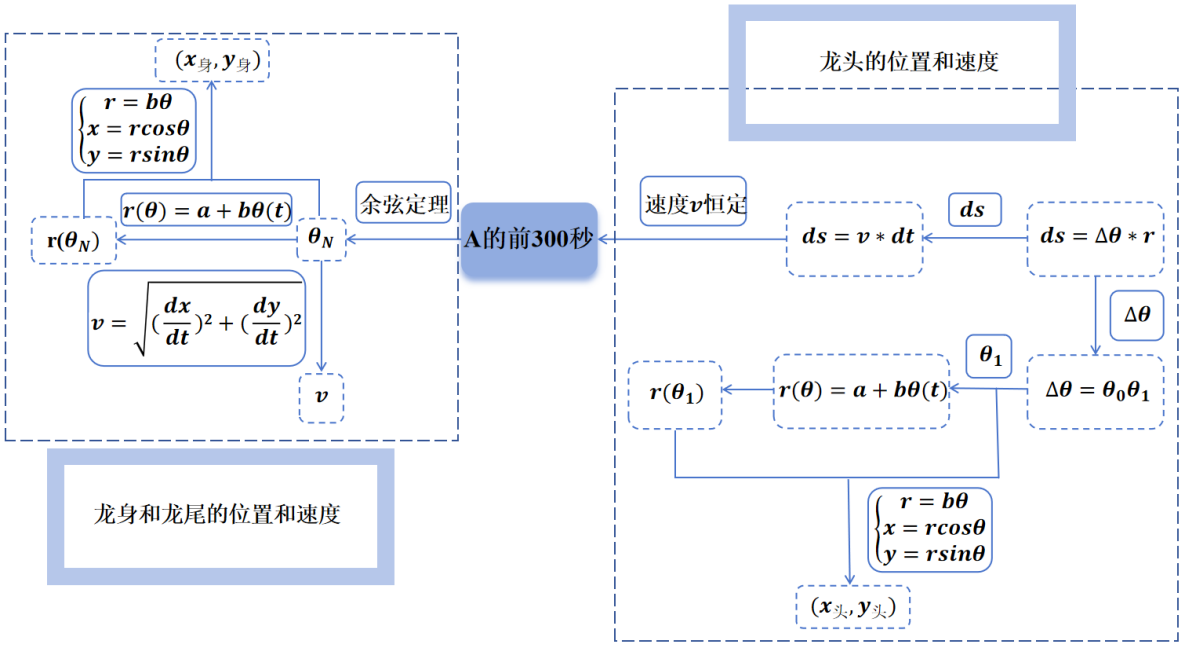

图 1 问题一的模型建立过程

5.1.1 龙头前把手位置的确定

本题中舞龙队沿螺距为 55 cm 的等距螺线顺时针盘入，各把手中心均位于螺线上，可确定舞龙队的运动轨迹为阿基米德螺旋线[1]，因此，舞龙队任一部分运动时的极坐标方程[2]为：

$$\begin{aligned} r(\theta) &= a + b\theta(t) \\ &= a + \frac{p}{2\Pi}\theta(t) \end{aligned} \tag{1}$$

其中，r 是极径，即从原点到曲线上某点的距离，θ是从极轴开始测量的角度，$\theta(t_0) = 32\Pi$， a是$\theta = 0$时的初始极径[3]，本题中$a = 0$； b是控制螺线收缩或膨胀的参数， p为螺距。

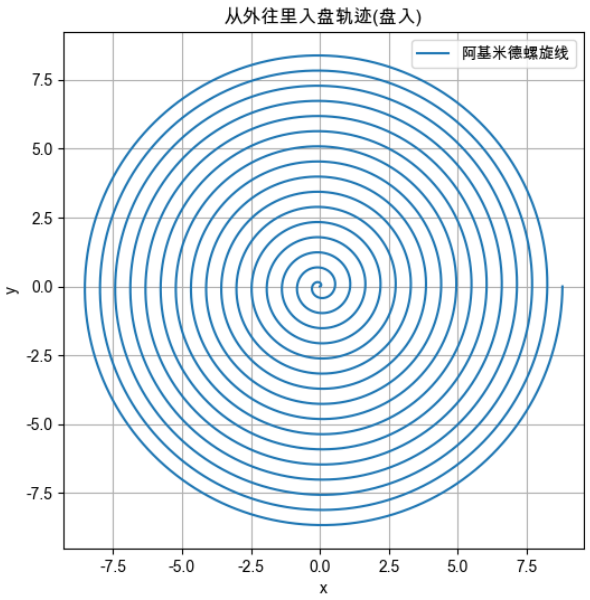

图 2 盘龙轨迹图

由于龙头的速度恒定不变，与龙身和龙尾的运动状态不同，因此模型建立时将舞龙队分成两部分，即第一部分是龙头部分，第二部分是龙身和龙尾部分。

- **龙头前把手运动的极坐标**

由龙头盘入的路径可知，为确定盘入各个时刻的位置，需要计算各个时刻的极径。因此，可以通过运动弧长和运动极角，进一步通过极坐标转换为直角系坐标确定龙头盘入路径的位置，具体流程如下。

首先计算龙头的前把手从 A 处运动一段微小距离ds后的情况，假设龙头的前把手运动的轨迹是以坐标原点为圆心，当前极径为半径的圆形，则

$$ds = \Delta\theta \cdot r \tag{2}$$

当龙头的前把手位于 A 点时，（1）式中的a为 0, $\theta(t_0)$为32Π，b 为$\frac{0.55}{2\Pi}$，则可确定$r(\theta_0) = 8.8m$；由于所建坐标系中，初始点 A 所在的任意同心圆为$p = 0.55m$的等距同心圆，故 OA 是该圆形的半径，即$16p = 8.8m$，该计算结果与代入阿基米德螺旋线极坐标方程的极径数值相同；此外，还可以由计算得知以坐标原点为圆心，16 个$p = 0.55m$的等距同心圆的总周长是$16 \cdot 2 \cdot \Pi \cdot 4.4 = 442.336m$,而总弧长是:

$$\sqrt{\left(a + b\theta_{初}\right)^2 + \left(a + b\theta_{末}\right)^2} = \sqrt{\left(0 + \frac{0.55}{2\Pi} \cdot 32\Pi\right)^2 + \left(0 + \frac{0.55}{2\Pi} \cdot 0\right)^2} = 442.590m$$

忽略由计算机底层浮点运算导致的误差，二者可以认为是相等的。通过上述计算结果，可以定量证明龙头的前把手运动的轨迹是以坐标原点为圆心，当前极径为半径

的圆形。绘制图形如下：

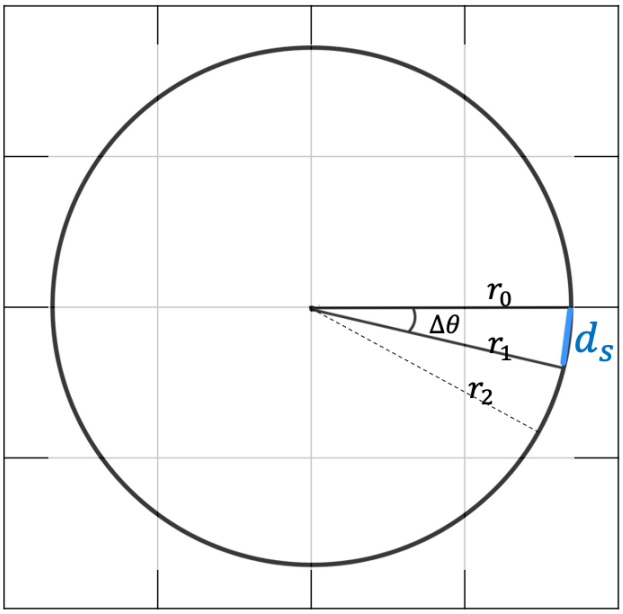

图 3 极角变化模型图

由于龙头的前把手保持$v_0 = 1m/s$的恒定速度，则在微小距离ds内，可以将龙头前把手的运动看作匀速直线运动，即：

$$ds = v \cdot dt \tag{3}$$

求得ds，将ds代入（2）中，求得$\Delta\theta$。如下图所示，此时$\Delta\theta = \theta_0 - \theta_1$。将$\theta_0$代入（1）式，求得此时龙头的前把手在阿基米德螺旋线上对应的极径$r(\theta_1)$。此时的极径可以作为下一次运动的初始极径。以此类推可得：

$$\theta_{t+1} = \theta_{\mathrm{t}} - \Delta\theta \tag{4}$$

根据上述步骤，我们求得从初始时刻到 300 s 为止，龙头的前把手每秒的极坐标位置。

- **龙头前把手运动的位置**

在完成龙头前把手的极径后，通过式（5）将极坐标转化为直角坐标，确定龙头的前把手从初始时刻到 300 s 为止每秒直角坐标的位置。

$$\begin{aligned} x(\theta) &= r(\theta)\cos(\theta) \\ y(\theta) &= r(\theta)\sin(\theta) \end{aligned} \tag{5}$$

将计算结果绘制其轨迹图如下图。

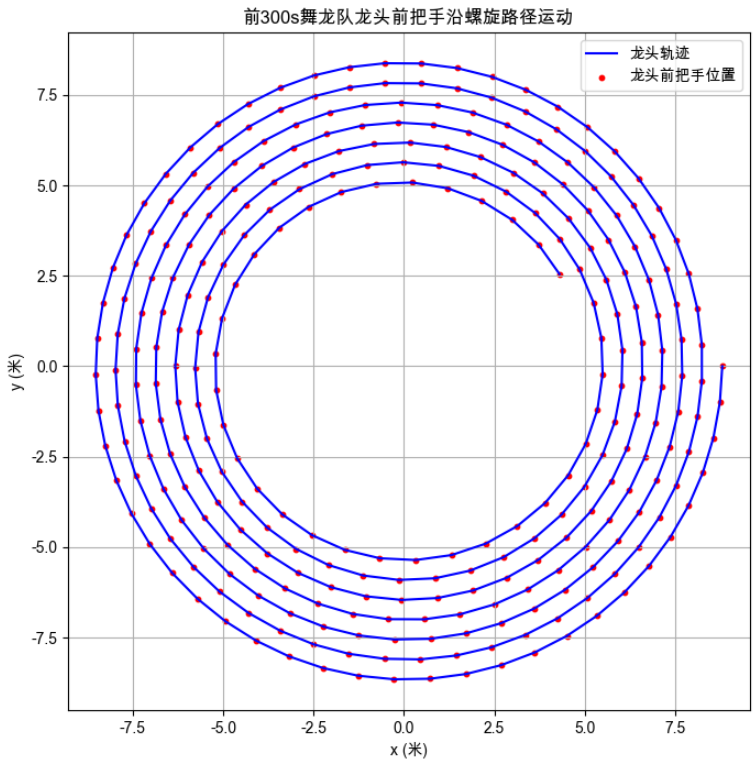

图 4 龙头的前把手从初始时刻到 300 s 为止每秒直角坐标的位置

5.1.2 龙身与龙尾前把手位置的确定

由于龙尾板凳长度与龙身板凳长度等长，通过在龙尾后多加一节等长的龙板凳，可得到如图 5 的几何关系

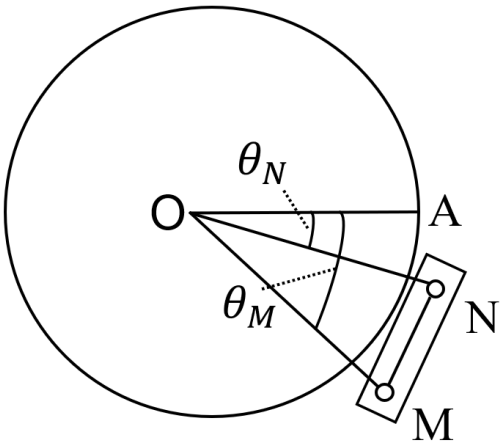

图 5 极角关系图

在三角形 OMN 中，OM（龙头前把手的极径）、ON（龙头前把手的极径）、MN（龙头前后把手的距离）以及∠MON（龙头前后把手角度的差值）可以用只含有一个未知量θ_N的方程表示，因此我们可以使用余弦定理：

$$l^2 = \left(a + b\theta_N(t)\right)^2 + \left(a + b\theta_M(t)\right)^2 - 2\left(a + b\theta_N(t)\right)\left(a + b\theta_M(t)\right) * \cos(\theta_M - \theta_N) \quad (6)$$

其中，l是龙身和龙尾前把手和后把手的距离。可求得θ_B。将结果代入（1）式，求得该时刻的极径，即可求得该时刻的极坐标。再根据（5）式转化为直角坐标，即可得龙身和龙尾直角坐标的位置。将计算得到的龙头、龙身和龙尾的运动路径结果成图 6 所示，由图可知，上述推导计算过程符合预期。

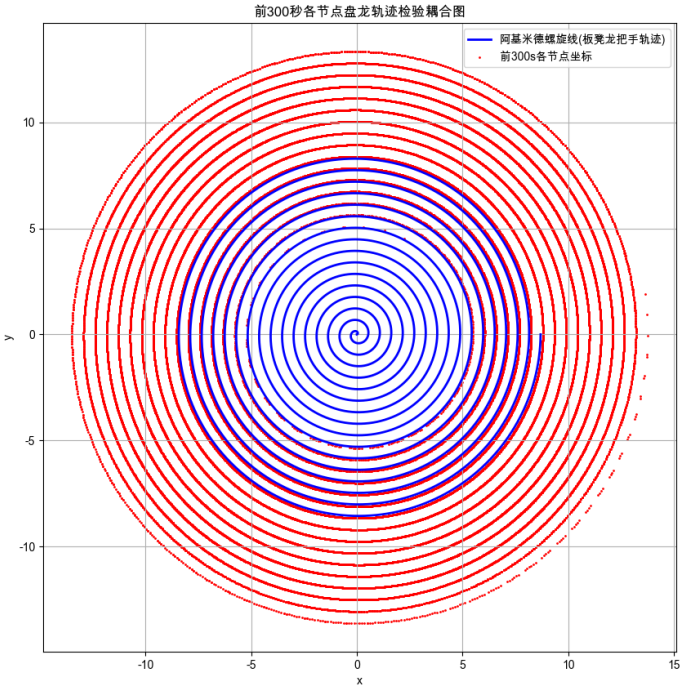

图 6 盘龙轨迹检验耦合图

5.1.3 龙身与龙尾前把手速度的确定

在求得龙身和龙尾的各时刻位置结果后，螺线运动轨迹的瞬时速度可以通过对位置函数求导得到。假设螺线的参数方程为$x(t)$和$y(t)$,其中t是参数，那么螺旋线的位置可以表示为：

$$\begin{aligned} x(t) &= r(t)\cos(t) \\ y(t) &= r(t)\sin(t) \end{aligned} \quad (7)$$

进一步对$x(t)$和$y(t)$求导可得：

$$\frac{dx}{dt} = \frac{d}{dt}\left(r(t)\cos(t)\right) = \frac{dr}{dt}\cos(t) - r(t)\sin(t) \quad (8)$$

$$\frac{dy}{dt} = \frac{d}{dt}\left(r(t)\sin(t)\right) = \frac{dr}{dt}\sin(t) + r(t)\cos(t) \quad (9)$$

瞬时速度的大小则通过以下公式计算

$$v = \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2} \tag{10}$$

将上述结果整理和化简后得到：

$$v = \sqrt{\left(\frac{dr}{dt}cos(t) - r(t)sin(t)\right)^2 + \left(\frac{dr}{dt}sin(t) + r(t)cos(t)\right)^2} \tag{11}$$

$$v = \sqrt{\left(\frac{dr}{dt}\right)^2 + \left(r(t)\right)^2} \tag{12}$$

最终得到龙身和龙尾的速度：

$$v = \sqrt{\left(\frac{p\theta}{2\Pi t}\right)^2 + \left(r(\theta)\right)^2} \tag{13}$$

此外，为了确保结果的准确性，我们还尝试使用公式$v = \frac{ds}{dt}$求得了龙身和龙尾对应的速度。结果表明，两者在数值上略有误差，但基本吻合。

5.1.4 问题一的求解结果

表 1 和表 2 分别为问题一的最终计算结果。

其中，假设各板凳在未盘入时，同样在实际轨迹的外围呈阿基米德螺旋线分布。

表 1 盘入时部分板凳的位置计算结果

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 x (m)	8.800000	5.822837	-4.024662	-3.070812	2.754696	4.298522
龙头 y (m)	0.000000	-5.747757	-6.344090	6.043200	-5.279101	2.547757
第 1 节龙身 x (m)	8.363824	7.470930	-1.374823	-5.313600	4.926958	2.233400
第 1 节龙身 y (m)	2.826544	-3.410367	-7.420173	4.268516	-3.418757	4.526360
第 51 节龙身 x (m)	-9.818249	-9.617586	-8.951886	-6.482784	1.402456	6.805048
第 51 节龙身 y (m)	2.583558	0.300164	1.404389	5.441570	7.694960	-2.106820
第 101 节龙身 x (m)	-9.746006	-1.645789	4.873760	7.426839	7.290584	4.767572
第 101 节龙身 y (m)	-5.761691	-10.723612	-9.136775	-6.451077	-5.760102	-7.293132
第 151 节龙身 x (m)	-12.355906	-1.686548	9.947436	9.892391	3.589331	-1.707795
第 151 节龙身 y (m)	-0.780308	-11.830493	-5.778325	4.901856	9.928231	9.906314
第 201 节龙身 x (m)	-2.694801	-12.078654	4.855802	11.679160	0.232610	-9.914932
第 201 节龙身 y (m)	13.080584	-4.690517	-11.569595	3.253192	11.683442	5.277879
龙尾（后）x (m)	13.627834	-5.769835	-10.526598	7.771320	10.293341	-3.426299
龙尾（后）y (m)	1.918724	12.068173	-7.593644	-9.880919	6.453356	11.201170

表 2 盘入时部分板凳的速度计算结果

	0 s	60 s	120 s	180 s	240 s	300 s
龙头(m/s)	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身(m/s)	0.000000	0.999328	0.999186	0.998970	0.998603	0.997850
第 51 节龙身(m/s)	0.000000	0.997952	0.997346	0.996381	0.994680	0.991156
第 151 节龙身(m/s)	0.000000	0.996657	0.995771	0.994422	0.992173	0.987831
第 201 节龙身(m/s)	0.000000	0.996301	0.995361	0.993946	0.991614	0.987166
龙尾（后）(m/s)	0.000000	0.996173	0.995217	0.993781	0.991423	0.986945

5.2 问题二模型的建立与求解

本题要求计算舞龙队沿问题一设定的螺线盘入过程的中止状态，终止盘入的时刻是板凳恰好不发生碰撞的临界值，我们分析有以下两种情况：

（1）任意龙身之间发生碰撞

（2）龙头与其所在层的前一层某龙身发生碰撞

5.2.1 盘入路径坐标的更新

在问题一中，确定了龙头和龙身在前 300s 每个时刻的位置和速度。因此，为求得是否发生碰撞，需要求得任意时刻所有板凳的运动路径，并考虑到板凳的实际尺寸限制。假设从第 300s 开始，时间每增加一个单位，路径信息更新一次。

首先，为简化计算过程，可以根据已知条件缩小碰撞时刻发生的圈数位置。

由问题一计算的位置结果可知，龙头在前 300s 不会发生碰撞，前 300s 移动的圈数在从外到内的第五圈和第六圈之间，即发生碰撞时的圈数在第十一圈之内。

其次，通过将阿基米德螺旋线近似看作是每个圆之间间隔 0.55m 的 16 个同心圆，根据龙头的长度为$3.41m$, 从内到外第i圈的半径为$i \cdot 0.55m$。在不考虑板宽的前提下，为使龙头不发生碰撞，则需要$3.41 > i \cdot 0.55 * 2$，即龙头板长应当至少大于所在圈层的同心圆直径，求得$i > 3$，即龙头最多只能进入从内到外的第四圈。

综上所述，发生碰撞的圈数位于第四圈到第十一圈之间。

设初始位置的直角坐标是(x_{head}, y_{head})和初始速度$\tilde{v}$，通过问题一的板凳位置计算其在不同时刻形成四个k_1, k_2, k_3, k_4路径斜率：

$$\begin{cases} k_1 = h \bullet f(t_n, x_n, y_n) \\ k_2 = h \bullet f\left(t_n + \dfrac{h}{2}, x_n + \dfrac{k_1}{2}, y_n + \dfrac{k_1}{2}\right) \\ k_3 = h \bullet f\left(t_n + \dfrac{h}{2}, x_n + \dfrac{k_2}{2}, y_n + \dfrac{k_2}{2}\right) \\ k_4 = h \bullet f\left(t_n + h, x_n + k_3, y_n + k_3\right) \end{cases} \tag{14}$$

其中，t_n是当前时间点，(x_n, y_n)是舞龙队当前位置坐标，h是时间步长，$f(t, x, y)$是位置随时间变化的微分方程，k_1, k_2, k_3, k_4是用于计算位置更新的四个斜率。

接着更新整个舞龙队的位置：

$$\begin{cases} x_{n+1} = x_n + \dfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ y_{n+1} = y_n + \dfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases} \tag{15}$$

其中，(x_{n+1}, y_{n+1})是舞龙队更新后的位置坐标。并根据龙格库塔法[4]，绘制整条舞龙队的动态仿真图，如图所示。

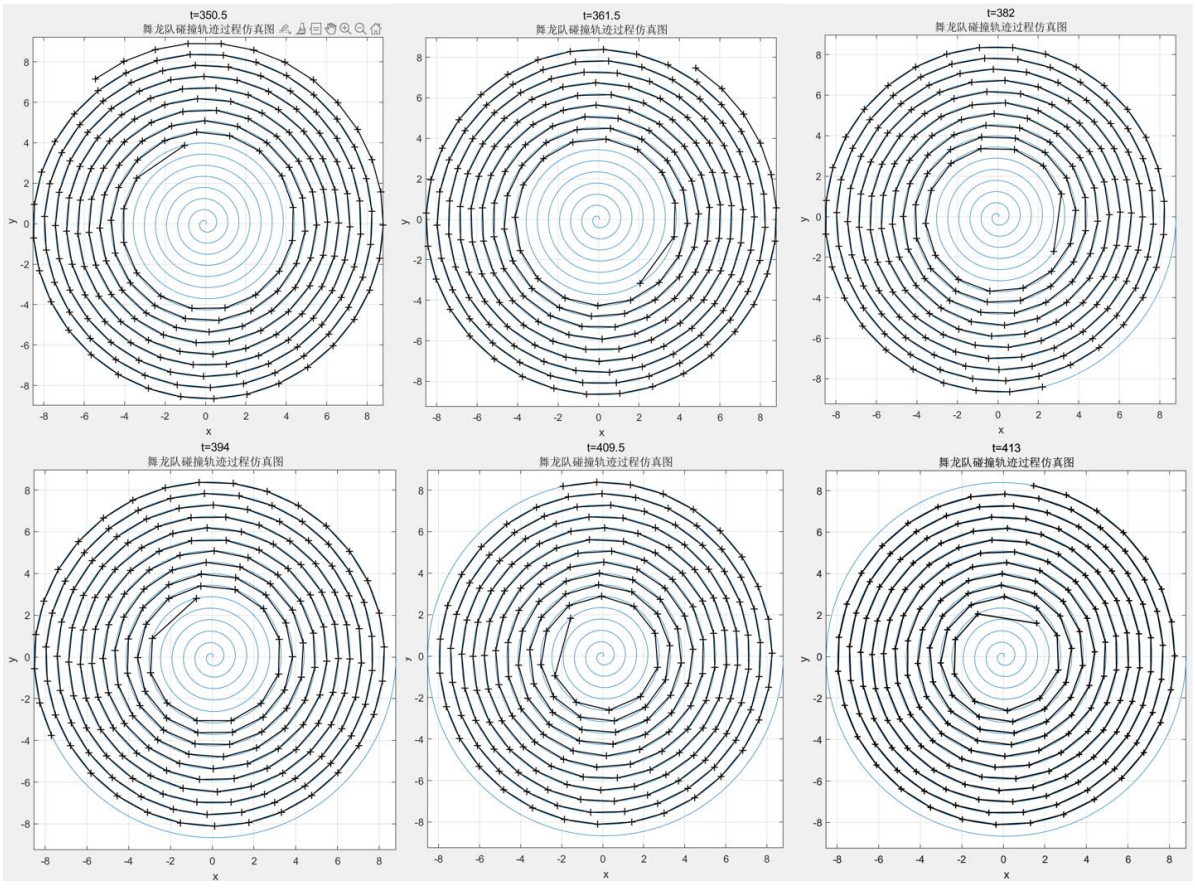

图 7 动态仿真图

5.2.2 碰撞条件判断

为求得碰撞时刻的具体时候和速度信息，以下进行碰撞的条件的分析。

- **基础碰撞条件（即任意相邻两个板凳的距离小于一个板凳的宽度）**

首先根据公式（5）确定龙头的位置，再确定第 i 节板凳的位置：

$$\begin{cases} x_i(t) = x\left(t - \dfrac{L_i}{v}\right) + L_i cos\left(\theta_i(t)\right) \\ y_i(t) = y\left(t - \dfrac{L_i}{v}\right) + L_i sin\left(\theta_i(t)\right) \end{cases} \tag{16}$$

其中，$\theta_i(t)$是第 i 节板凳相对于龙头的相对角度，则

$$\theta_i(t) = \theta\left(t - \frac{L_i}{v}\right) \tag{17}$$

接着计算相邻板凳之间的距离，相邻第 i 节板凳和第 i+1 节板凳的距离为

$$D_{i,i+1}(t) = \sqrt{\left(x_{i+1}(t) - x_i(t)\right)^2 + \left(y_{i+1}(t) - y_i(t)\right)^2} \tag{18}$$

即发生碰撞的条件是相邻板凳之间的距离小于它们的总长度加上安全距离：

$$D_{i,i+1}(t) < L_i + L_{i+1} + \delta \tag{19}$$

则认为在时间 t 发生了碰撞，其中δ是安全距离。

- **内外层碰撞条件**

根据已知条件，以板凳运动方向为正，由于龙头位于舞龙队的最前端，其路径每时刻均发生更新，因此只需判断龙头前把手左前方的端点是否会再次经过龙身右侧所在直线走过的轨迹，即简化为判断一个点的轨迹是否会再次与一条线的轨迹有交点。

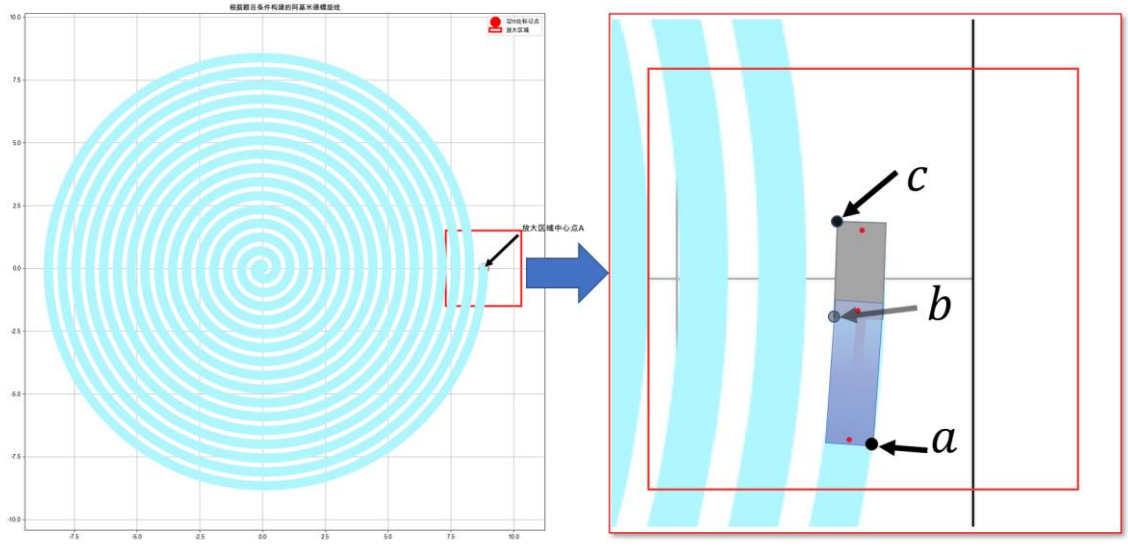

图 8 碰撞机制分析图

龙头前把手和后把手的位置已知，由图所示由于板凳是长方形，即可以在直角坐标系下确定 a 点、b 点、c 点的坐标，分别是$(x_a, y_a), (x_b, y_b), (x_c, y_c)$。此处 b 点和 c 点的坐标并不特指第一节龙身坐标，而是所有可能与龙头发展碰撞的龙身右侧两点的坐标。

求出 b，c 两点的坐标，就可以知道 b，c 两点所在直线的坐标，即：

$$y = mx + k \tag{20}$$

其中，$m = \frac{y_2 - y_1}{x_2 - x_1}$，$k = y_1 - mx_1$

根据公式（5）可以将 b，c 两点所在直线的直角坐标方程，转化为极坐标方程：

$$r = \frac{k}{\sin(\theta) - m\cos(\theta)} \tag{21}$$

将直线的极坐标方程代入公式（1）可得：

$$a + b\theta = \frac{k}{\sin(\theta) - m\cos(\theta)} \tag{22}$$

求得 b，c 两点所在直线的运动轨迹后，与 a 点运动的轨迹对比即可找到碰撞的时刻。

5.2.3 盘龙碰撞模型的结果求解

通过对盘入路径的更新，得到 300s 后，时间步长为 0.5s 的轨迹仿真图，当$t = 413s$时，碰撞模型的条件处理，表明发生了碰撞情况，此时的舞龙队位置如下图 9 所示。因此盘入的碰撞时间为$412.5s$至$413s$之间。

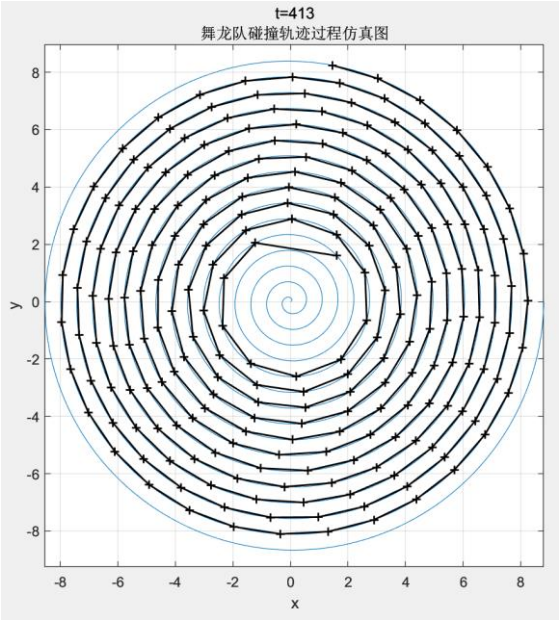

图 9 碰撞时刻轨迹图

六、模型优缺点评价

6.1 模型的优点

多方法求解速度：模型采用了两种方法（几何关系和微分方程）求解舞龙队的速度，两者求得结果几乎相等。几何关系法能够快速估计速度分布，而微分方程法则提供了更精确的速度随时间变化的解析。

基准模型建立：所有计算都基于阿基米德螺线模型，使用阿基米德螺线方程描述舞龙队的盘入路径，精确捕捉了龙头和龙身的轨迹变化。这一螺线模型能够较好地描述舞龙队的运动规律，保证了路径精度。

仿真验证模型准确性：在模型构建的基础上，通过仿真对舞龙队的路径和速度进行了验证。仿真过程能够动态展示舞龙队在不同时间的运动情况，直观清晰，确保了理论推导的可靠性。

碰撞检测仿真：通过仿真分析舞龙队在盘入过程中的可能碰撞时刻，有效验证了模型在实际操作中的可行性。通过仿真，可以直观地观测到队伍中的板凳是否会在某一时刻相撞，优化了模型的设计，避免了实际中的错误。

6.2 模型的缺点

基于理想条件：所有的求解都是理想化的模型，考虑实际的其他因素较少。如刚性连接、同步运动等。实际表演中，队员的动作不同步、场地的摩擦力变化、空气阻力等因素都会影响舞龙队的实际运行效果。

S 形调头路径过于理想：虽然 S 形曲线能够提供一个调头路径，但实际舞龙队的调头过程受到队员同步性和路径复杂性的影响，S 形曲线可能无法完全反映实际情况。尤其是调头过程中，龙身和龙尾的转向调整较慢，模型未考虑这种情况。

曲线运动细节缺失：在阿基米德螺线或 S 形曲线等复杂路径上，1 秒时间片可能无法捕捉到龙头和龙尾的曲线运动细节，尤其是龙尾在盘入、盘出时的路径精细变化，曲线运动的细节可能被简化，一定程度上影响模型的准确性。

6.3 模型的改进

使用更小的时间步长，尤其是在曲线转向或调头的关键阶段，能够更好地捕捉龙头和龙尾的轨迹变化。

参考文献

[1]王栋,周可璞.基于阿基米德螺线走法的全区域覆盖路径规划[J].工业控制计算机,2018,31(05):83-84+87.

[2]三种等距螺线的统一与差异 - 知乎 (zhihu.com)

[3]刘崇军.等距螺旋的原理与计算[J].数学的实践与认识,2018,48(11):165-174.

[4]江山,张岩,孙美玲.常微分方程初值问题的高阶泰勒法与龙格-库塔法之应用对比[J].高师理科学刊,2019,39(12):12-15.

附录

python代码及数据结果树结构图

```
.
└── math-modeling
    ├── __init__.py
    ├── imgs
    │   ├── _02_根据题目条件构建的阿基米德螺旋线.png
    │   ├── _03_从外往里入盘轨迹(盘入).png
    │   ├── _05_前300s龙头运动轨迹及定位.png
    │   ├── _08_根据螺旋角求得坐标绘制的前300s轨迹图.png
    │   └── _10_前300秒各节点盘龙轨迹检验耦合图(余弦定理求得个节点各时间theta验证).png
    ├── imgs.zip
    ├── output
    │   ├── _05_前300s龙头位置信息.csv
    │   ├── _07_前300s各个节点的坐标信息_方法1.csv
    │   ├── _08_根据螺旋角求得坐标.csv
    │   ├── _08_余弦定理求得其余点螺旋角(根据螺旋角定位出x,y).csv
    │   ├── _11_检测(灵敏度分析)_根据前300s龙头theta求龙头每秒弧长.csv
    │   ├── _14_表格转换结构.csv
    │   ├── _15_选择舍弃(某处公式推导出错)_各点速度结果_方法1(推导公式).csv
    │   ├── _17_已舍弃_各点速度结果.csv
    │   ├── _17_求速度前的表格转换.csv
    │   ├── _18_各点速度结果(方法1).csv
    │   └── _18_各点速度结果(方法2_更靠谱).csv
    ├── result
    │   ├── _01_result01_各个节点的坐标信息.py
    │   ├── _02_result01_2各节点速度信息.py
    │   ├── _03_论文表格_01.py
    │   ├── _04_查的论文表一数据.py
    │   ├── res.csv
    │   ├── result01.csv
```

```
30      │   ├── result02.csv
31      │   ├── speed_result.csv
32      │   ├── 论文表格_01(修改).csv
33      │   └── 论文表格_01(自建).csv
34      ├── test
35      │   ├── _07的结果绘图.py
36      │   ├── demo3.py
37      │   ├── my_test_01.py
38      │   ├── test_碰撞.py
39      │   ├── 求解thetaB下的弧长ds_13.py
40      │   ├── 测试代码01.py
41      │   ├── 龙头弧长累计.py
42      │   └── 拟合曲率半径05.py
43      └── work
44          ├── _02_尝试绘制螺旋线.py
45          ├── _03_从外往里入盘轨迹(出盘).py
46          ├── _05_前300s龙头运动轨迹及定位.py
47          ├── _06_已舍弃_相同案例.py
48          ├── _06_已舍弃_拟合曲率半径.py
49          ├── _07_数据表格转换(007的).py
50          ├── _07_第一问求解前300s各节点位置_方法1.py
51          ├── _08_根据theta求得坐标位置_方法2第二步.py
52          ├── _08_余弦定理求其他点theta_方法2第一步.py
53          ├── _09_第一问结果图(存在延长弧).py
54          ├── _10_板凳龙把手轨迹与前300秒各节点耦合图.py
55          ├── _11_检测(灵敏度分析)_根据前300s龙头theta求龙头每秒弧
    长.py
56          ├── _12_弧长公式(靠谱版)这里标记做题顺序(公式已封装到
    base_tools中).py
57          ├── _14_表格转换_方便后续论文和求解速度.py
58          ├── _15_第一问各个点速度求解_方法1.py
59          ├── _16_已舍弃_节点每一秒的delta_theta和速度.py
60          ├── _17_利用速度公式.py
61          ├── _18_最终速度求解(方法1).py
62          ├── _18_最终速度求解(方法2_靠谱).py
63          ├── _19_起点A标注图.py
64          ├── _20_龙格库塔法求解前进策略.py
```

```
        ├── _22_最短螺距建模求解.py
        ├── __init__.py
        ├── utils
        │   ├── __init__.py
        │   ├── __pycache__
        │   │   ├── __init__.cpython-38.pyc
        │   │   └── base_tools.cpython-38.pyc
        │   ├── base_tools.py
        │   └── copy_tools.py
        └── 龙格库塔仿真
            ├── __pycache__
            │   ├── code01.cpython-38.pyc
            │   └── find_if_intersect.cpython-38.pyc
            ├── code01.py
            ├── code02.py
            └── find_if_intersect.py
```

python具体关键代码：

1. utils.base_tools.py

```python
"""
定义了一些常用的函数工具(比如阿基米德螺旋公式, 极坐标与直角坐标的相互转化等)
"""
import warnings
import numpy as np
from scipy.integrate import quad
from scipy.optimize import fsolve


def r(theta, a, b):
    """
    阿基米德螺旋线极坐标方程
    :param theta: 螺旋角
    :param a: 螺旋线初始点半径
```

```
    :param b: 径增长比 P/(2*pi)
    :return: 极径r
    """
    return a + b * theta


def calc_position(a, b, theta):
    """
    计算螺旋线上的位置 (x, y)
    :param a:
    :param b:
    :param theta:
    :return:
    """
    r = a + b * theta
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y


def calc_arc_length(a, b, theta1, theta2):
    """
    计算阿基米德螺旋线的弧长，已废弃
    :param a: 螺旋线初始点半径
    :param b: 径增长比 P/(2*pi)
    :param theta1: 要求弧长的起点极角(弧头螺旋角)
    :param theta2: 要求弧长的终点极角(弧尾螺旋角)
    :return: 某一段的弧长
    """
    warnings.warn("此方法已废弃",
                  DeprecationWarning)
    integrand = lambda theta: np.sqrt(b ** 2 + (a + b *
theta) ** 2)
    arc_length, _ = quad(integrand, theta1, theta2)
    return arc_length


```

```python
def find_theta_by_arc_length(a, b, current_theta, target_arc_length):
    """
    给定目标弧长，找到对应的螺旋角度
    :param a:
    :param b:
    :param current_theta:
    :param target_arc_length:
    :return:
    """
    warnings.warn("此方法过于粗略,误差极大",
                  DeprecationWarning)

    def equation(theta):
        return calc_arc_length(a, b, theta, current_theta) - target_arc_length

    theta_solution = fsolve(equation, current_theta - 1)[0]
    return theta_solution


# ----------
# 计算弧长(theta与l的等量关系)，需定义螺旋线的 r(θ) 和其导数 dr/dθ
def dr_dtheta(theta, b):
    """
    极径关于theta的导数
    :param theta:
    :param b:
    :return:
    """
    return b


def arc_length_integrand(theta, a, b):
    """
     定义计算弧长的被积函数
    :param theta: pass
```

```
    :param a: pass
    :param b: pass
    :return: 弧长被积函数
    """
    return np.sqrt(dr_dtheta(theta, b) ** 2 + r(theta, a, b) ** 2)


def compute_arc_length(a, b, theta1, theta2):
    """ 数值积分方法计算弧长
        计算从 θ1 到 θ2 的弧长(阿基米德螺旋线的弧长，当然通用于所有极坐标的弧长计算)
        :param a: 螺旋线初始点半径
        :param b: 径增长比 P/(2*pi)
        :param theta1: 要求弧长的起点极角(弧头螺旋角)
        :param theta2: 要求弧长的终点极角(弧尾螺旋角)
        :return: 某一段θ1 到 θ2的弧长
        """
    length, error = quad(arc_length_integrand, theta1, theta2, args=(a, b))
    return length


# -----------


# -----------
# 余弦定理求解thetaB (B表示龙头的后一个节点，即第一个龙身前把手所经过的theta)
# 根据余弦定理求得其他的位置与龙头位置的关系结果，可求得theta_others
def determine_theta_other(l, a, b, theta_per):  # l就是板长，a=0, b=0.55/(2*pi), theta_per就是前一个点的螺旋角(龙头的已求出)
    # 绘图后建立的余弦定理方程，theta_i就是要求解的当前theta角
    # l ** 2 = (a + b * theta_i)**2 + (a + b * theta_per)**2 - 2*(a + b * theta_i) * (a + b * theta_i) * cos(theta_per - theta_i)

```

```
    # 初始化一个数组来存储每个时间步的结果
    def equation(theta_i, l, a, b, theta_p):
        return l ** 2 - (a + b * theta_i) ** 2 - (a + b * theta_p) ** 2 + 2 * (a + b * theta_i) * (
                a + b * theta_p) * np.cos(theta_p - theta_i)

    # 初始化一个数组来存储每个时间步的结果
    theta_others = np.zeros(len(theta_per))

    for i, theta_p in enumerate(theta_per):
        # 使用 fsolve 求解 theta_i
        initial_guess = theta_p  # 初始猜测值可以设为 theta_p
        theta_i_solution, = fsolve(equation, initial_guess, args=(l, a, b, theta_p))
        theta_others[i] = theta_i_solution


# _______
# 通过弧长变化计算速度
def calculate_velocity(a, b, theta_values):
    velocities = [1]  # 起始速度为 1 m/s
    for i in range(1, len(theta_values)):
        s1 = compute_arc_length(a, b, 0, theta_values[i - 1])
        s2 = compute_arc_length(a, b, 0, theta_values[i])
        ds = s2 - s1
        velocities.append(ds)  # 因为 dt = 1, 速度 v = ds/dt = ds
    return velocities
```

2. 余弦定理求其他点theta_方法2第一步.py

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

# 根据余弦定理求得其他的位置与龙头位置的关系结果，可求得theta_others
```

```python
def determine_theta_other(l, a, b, theta_per):  # l就是板长, a=0, b=0.55/(2*pi), theta_per就是前一个点的螺旋角(龙头的已求出)
    # 绘图后建立的余弦定理方程, theta_i就是要求解的当前theta角
    # l ** 2 = (a + b * theta_i)**2 + (a + b * theta_per)**2 - 2*(a + b * theta_i) * (a + b * theta_i) * cos(theta_per - theta_i)

    def equation(theta_i, l, a, b, theta_p):
        return l ** 2 - (a + b * theta_i) ** 2 - (a + b * theta_p) ** 2 + 2 * (a + b * theta_i) * (
                a + b * theta_p) * np.cos(theta_p - theta_i)

    # 初始化一个数组来存储每个时间步的结果
    theta_others = np.zeros(len(theta_per))

    for i, theta_p in enumerate(theta_per):
        # 使用 fsolve 求解 theta_i
        initial_guess = theta_p  # 初始猜测值可以设为 theta_p
        theta_i_solution, = fsolve(equation, initial_guess, args=(l, a, b, theta_p))
        theta_others[i] = theta_i_solution

    return theta_others


if __name__ == '__main__':
    l = (220 - 27.5 - 27.5) / 100  # 板长(两接点距离)
    a = 0
    b = 0.55 / (2 * np.pi)

    # 读取初始的龙头螺旋角信息
    df = pd.read_csv("../output/_05_前300s龙头位置信息.csv")
    theta_per = df.iloc[3, 1:].values

    # 初始化空 DataFrame, 用于保存所有节点的计算结果
    all_results = []

```

```
    # 遍历每个节点
    sum_point = 223 #尾部多加一块等长板
    for point in tqdm(range(sum_point)):
        if point == 0:
            l = (341 - 27.5 - 27.5) / 100  # 龙头特殊长，也是两孔径

        # 计算当前节点的螺旋角 theta_i
        theta_per = determine_theta_other(l, a, b, theta_per)

        # 将每个节点的编号和对应的 theta_i 结果存入 all_results 列表
        for theta in theta_per:
            all_results.append([point, theta])

        print(f"当前节点 {point} 的螺旋角 theta_i: {theta_per}")

    all_results_df = pd.DataFrame(all_results, columns=["节点编号", "theta_i"])
    all_results_df.to_csv("../output/_08_余弦定理求得其余点螺旋角(根据螺旋角定位出x,y).csv", index=False, encoding="utf-8")

```

3. 第一问各个点速度求解_方法1.py

```
import pandas as pd
import numpy as np

# 读取之前保存的转换后的表格数据
input_file_path = '../output/_14_表格转换结构.csv'
data = pd.read_csv(input_file_path, index_col=0)

# 定义常量 b，用于计算半径 r
b = 0.55 / (2 * np.pi)


# 计算速度的函数，确保节点1的速度始终为 1 m/s
def compute_velocity(theta_vals, b, node_index):

```

```
    if node_index == 0:
        return [1] * len(theta_vals)  # 节点1的所有速度为 1 m/s

    velocities = []
    for i in range(1, len(theta_vals)):
        r = b * theta_vals[i]  # 根据 theta 计算半径 r
        ds = r * abs(theta_vals[i] - theta_vals[i - 1])  # 计算 ds = r * dθ
        velocities.append(ds)  # 存储速度 (dt = 1秒)

    return [1] + velocities  # 第一个速度设为 1，保持一致性


# 创建一个新的 DataFrame 来存储计算后的速度
corrected_velocities_df = pd.DataFrame()

# 遍历每个节点，计算速度并重命名为“节点i_v”
for index, column in enumerate(data.columns):
    theta_vals = data[column].values  # 获取每个节点的 theta 值
    velocities = compute_velocity(theta_vals, b, index)  # 计算速度
    corrected_velocities_df[f'节点{index + 1}_v'] = velocities  # 将速度存储并命名为“节点i_v”

# 保存计算后的速度数据到CSV文件
output_file_path = '../output/_18_各点速度结果(方法1).csv'
corrected_velocities_df.to_csv(output_file_path, index=False)

print(f"已将速度数据保存到 {output_file_path}")
```

4. 最终速度求解(方法2_靠谱).py

```
import pandas as pd
import numpy as np
from utils import base_tools


```

```python
# 读取之前保存的转换后的表格数据
input_file_path = '../output/_17_求速度前的表格转换.csv'
data = pd.read_csv(input_file_path, index_col=0)

a = 0
# 定义常量 b，用于计算半径 r
b = 0.55 / (2 * np.pi)
dt = 1

"""分部积分"""

# 计算速度的函数，确保节点1的速度始终为 1 m/s
def compute_velocity(theta_vals, b, node_index):
    if node_index == 0:
        return [1] * len(theta_vals)  # 节点1的所有速度为 1 m/s

    velocities = []
    for i in range(1, len(theta_vals)):
        r = b * theta_vals[i]  # 根据 theta 计算半径 r
        # ds = r * abs(theta_vals[i] - theta_vals[i - 1])  # 计算 ds = r * dθ
        ds = base_tools.compute_arc_length(a, b, theta_vals[i], theta_vals[i - 1])
        velocities.append(ds/dt)  # 存储速度 (dt = 1秒)

    return [1] + velocities  # 第一个速度设为 1，保持一致性


# 创建一个新的 DataFrame 来存储计算后的速度
corrected_velocities_df = pd.DataFrame()

# 遍历每个节点，计算速度并重命名为“节点i_v”
for index, column in enumerate(data.columns):
    theta_vals = data[column].values  # 获取每个节点的 theta 值
    velocities = compute_velocity(theta_vals, b, index)  # 计算速度
```

```
    corrected_velocities_df[f'节点{index + 1}_v'] = velocities  # 将速度存储并命名为“节点i_v”

# 保存计算后的速度数据到csv文件
output_file_path = '../output/_18_各点速度结果(方法2_更靠谱).csv'
corrected_velocities_df.to_csv(output_file_path, encoding="gbk", index=False)

print(f"已将速度数据保存到 {output_file_path}")
```

5. 龙格库塔法求解前进策略.py

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


def plot_spiral(luoju, k):
    """绘制初始螺旋线图"""
    theta = np.arange(20 * 2 * np.pi, 0, -0.01)
    r = k * theta
    x = r * np.cos(theta)
    y = r * np.sin(theta)

    plt.figure(figsize=(6, 6))
    plt.plot(x, y, '--')
    plt.axis('equal')
    plt.grid(True)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('螺线图')
    plt.show()


def mydtheta(theta, t, k):
```

```
    """微分方程：螺旋线的角速度变化"""
    return -1 / (k * np.sqrt(1 + theta ** 2))


def solve_theta(luoju, x1, y1, theta1, d):
    """求解theta值的函数，给定位置和距离约束"""
    k = luoju / (2 * np.pi)
    fun = lambda theta: (k * theta * np.cos(theta) - x1) ** 2 + (k * theta * np.sin(theta) - y1) ** 2 - d ** 2
    q = 0.01
    theta = fsolve(fun, theta1 + q)[0]
    while theta <= theta1 or abs(k * theta - k * theta1) > luoju / 2:
        q += 0.1
        theta = fsolve(fun, theta + q)[0]
    return theta


def compute_positions(tspan, theta0, luoju, D1, D2, N):
    """计算所有孔的位置"""
    k = luoju / (2 * np.pi)
    theta_sol = odeint(mydtheta, theta0, tspan, args=(k,)).flatten()

    X1 = k * theta_sol * np.cos(theta_sol)
    Y1 = k * theta_sol * np.sin(theta_sol)

    X = np.zeros((N + 1, len(X1)))
    Y = np.zeros((N + 1, len(X1)))
    Theta = np.zeros((N + 1, len(X1)))

    X[0, :] = X1
    Y[0, :] = Y1
    Theta[0, :] = theta_sol

    for j in range(len(tspan)):
        for i in range(1, N + 1):
```

```
            d = D1 if i <= 2 else D2
            theta_ij = solve_theta(luoju, X[i - 1, j], Y[i - 1, j], Theta[i - 1, j], d)
            Theta[i, j] = theta_ij
            X[i, j] = k * theta_ij * np.cos(theta_ij)
            Y[i, j] = k * theta_ij * np.sin(theta_ij)

    return X, Y, Theta


def compute_velocity(Theta, k, dt):
    """计算速度数据"""
    V = np.zeros(Theta.shape)
    V[:, 0] = -k * np.sqrt(1 + Theta[:, 0] ** 2) * (Theta[:, 1] - Theta[:, 0]) / dt
    V[:, -1] = -k * np.sqrt(1 + Theta[:, -1] ** 2) * (Theta[:, -1] - Theta[:, -2]) / dt
    V[:, 1:-1] = -k * np.sqrt(1 + Theta[:, 1:-1] ** 2) * (Theta[:, 2:] - Theta[:, :-2]) / (2 * dt)
    return V


def export_data(X, Y, V, tspan, dt, filename):
    """导出位置和速度数据到Excel文件"""
    nn = int(1 / dt)
    index = np.arange(0, len(tspan), nn)
    Dataxy = np.zeros((2 * (X.shape[0]), len(index)))
    Dataxy[0::2, :] = np.round(X[:, index], 6)
    Dataxy[1::2, :] = np.round(Y[:, index], 6)
    Datav = np.round(V[:, index], 6)

    with pd.ExcelWriter(filename) as writer:
        pd.DataFrame(Dataxy).to_excel(writer, sheet_name='位置', startcol=1, index=False)
        pd.DataFrame(Datav).to_excel(writer, sheet_name='速度', startcol=1, index=False)

```

```python

def main():
    # 参数设置
    luoju = 55e-2  # 螺距
    k = luoju / (2 * np.pi)  # 螺旋常数

    L1 = 341e-2
    D1 = L1 - 27.5e-2 * 2
    L2 = 220e-2
    D2 = L2 - 27.5e-2 * 2

    # 绘制螺旋线
    plot_spiral(luoju, k)

    # 第一步：求解第一个把手的轨迹
    theta0 = 2 * np.pi * 16
    dt = 0.1
    tspan = np.arange(0, 300, dt)
    N = 223  # 龙头+龙身+龙尾的总个数

    X, Y, Theta = compute_positions(tspan, theta0, luoju, D1, D2, N)

    # 计算速度数据
    V = compute_velocity(Theta, k, dt)

    # 绘制头把手的速度随时间变化
    plt.figure()
    plt.plot(tspan, V[0, :], 'b-', linewidth=1.3)
    plt.ylim([0, 1.1])
    plt.xlabel('时间')
    plt.ylabel('头把手的速度')
    plt.title('验证数值计算得到的速度')
    plt.show()

    # 导出数据
    export_data(X, Y, V, tspan, dt, 'result1_test.xlsx')
```

```
127
128     print("全部计算完成！")
129
130
131 if __name__ == "__main__":
132     main()
```

6. 龙格库塔仿真

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.integrate import odeint
4  from scipy.optimize import fsolve
5  from find_if_intersect import find_if_intersect  # 假设模块已存在
6  from code01 import solve_theta  # 假设模块已存在
7
8  # 参数初始化
9  luoju = 55e-2  # 螺距
10 k = luoju / (2 * np.pi)  # 螺线方程系数
11 L1 = 341e-2
12 D1 = L1 - 27.5e-2 * 2  # 龙头把手两个孔之间的距离
13 L2 = 220e-2
14 D2 = L2 - 27.5e-2 * 2  # 其他把手两个孔之间的距离
15
16 def plot_spiral(k):
17     """绘制初始螺旋线"""
18     theta = np.arange(16 * 2 * np.pi, 0, -0.01)
19     r = k * theta
20     x = r * np.cos(theta)
21     y = r * np.sin(theta)
22
23     plt.figure(figsize=(6, 6))
24     plt.plot(x, y, '--')
25     plt.gca().set_aspect('equal', adjustable='box')
26     plt.xlabel('x')
27     plt.ylabel('y')
```

```
    plt.grid(True)
    plt.show()

def mydtheta(t, theta, k):
    """定义微分方程：角度随时间的变化"""
    return -1 / (k * np.sqrt(1 + theta**2))

def compute_positions(X, Y, Theta, tspan, N, k, luoju, D1, D2):
    """计算每个孔在不同时间点的位置"""
    for j in range(1, len(tspan)):
        for i in range(1, N + 1):
            d = D1 if i <= 2 else D2
            theta_ij = solve_theta(luoju, X[i - 1, j], Y[i - 1, j], Theta[i - 1, j], d)
            Theta[i, j] = theta_ij
            X[i, j] = k * theta_ij * np.cos(theta_ij)
            Y[i, j] = k * theta_ij * np.sin(theta_ij)

def dynamic_plot(X, Y, step, dt):
    """动态绘制轨迹"""
    plt.plot(X[:, -1], Y[:, -1], 'k-', linewidth=1.2, marker='o', markersize=6, markerfacecolor='r')
    plt.title(f't = {300 + step * dt}')
    plt.grid(True)
    plt.draw()
    plt.pause(0.01)

def detect_collision(X, Y, Theta, N, L1, L2):
    """检测孔与孔之间是否发生碰撞"""
    for i in range(N):
        if find_if_intersect(X, Y, Theta, i, L1, L2):
            return True
    return False

def compute_velocity(Theta, k, dt):
    """计算速度"""
```

```
    return -k * np.sqrt(1 + Theta[:, -1]**2) * (Theta[:, -1] - Theta[:, -2]) / (dt / 2)

def plot_velocity(V):
    """绘制速度图"""
    plt.figure()
    plt.plot(V, 'b-', linewidth=1.3)
    plt.ylim([0, 1.1])
    plt.xlabel('把手位置')
    plt.ylabel('把手的速度')
    plt.title('数值计算得到的速度分布')
    plt.show()

def main():
    """主函数，执行完整逻辑"""
    theta0 = 57.032076651015522  # 初始角度
    dt = 0.2
    step = 0
    N = 223  # 总数

    # 初始化位置矩阵
    X = np.full((N+1, 3), np.nan)
    Y = np.full((N+1, 3), np.nan)
    Theta = np.full((N+1, 3), np.nan)
    Theta[0, 2] = theta0
    flag = False

    plot_spiral(k)  # 绘制初始螺旋线

    # 开始动态模拟
    while not flag:
        step += 1
        X[:, 0], Y[:, 0], Theta[:, 0] = X[:, 2], Y[:, 2], Theta[:, 2]

        # 使用 odeint 求解微分方程
        tspan = [0, dt / 2, dt]
```

```
        theta_sol = odeint(mydtheta, Theta[0, 0], tspan, args=(k,))
        X1 = k * theta_sol * np.cos(theta_sol)
        Y1 = k * theta_sol * np.sin(theta_sol)

        # 更新位置和角度
        X[0, :] = X1.flatten()
        Y[0, :] = Y1.flatten()
        Theta[0, :] = theta_sol.flatten()

        # 计算每个孔的位置
        compute_positions(X, Y, Theta, tspan, N, k, luoju, D1, D2)

        # 动态绘制轨迹
        dynamic_plot(X, Y, step, dt)

        # 碰撞检测
        if detect_collision(X, Y, Theta, N, L1, L2):
            flag = True
            break

    # 计算速度并绘制速度分布图
    V = compute_velocity(Theta, k, dt)
    plot_velocity(V)

if __name__ == "__main__":
    main()

```