摘要

“板凳龙”，亦称作“盘龙”，在浙江与福建地区广为流传，作为一种传统的民俗文化活动而备受推崇。本论文旨在深入探讨并优化“板凳龙”表演的行进路径与速度控制，以提升其表演的艺术性和观赏性。

针对问题一，我们针对“板凳龙”的运动构建了一个数学模型，通过螺线轨迹描述“板凳龙”的行进过程。首先，我们基于螺线的极坐标方程，确定各把手的运动轨迹。接着利用微元法得到龙头运动的微分方程，然后基于板凳把手均位于螺线上、板凳不可伸长等假设得到各把手位置及速度的递推关系。最终，将问题一的参数代入计算，利用数值方法求解 0 s~300 s 每秒的各节龙身位置与速度，并**将 0s、60s、120s、180s、240s、300s 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运动数据记录在表（2）与表（3）中。**

针对问题二，我们首先将“板凳龙”的螺线轨迹简化为两个同心圆，利用几何关系得到碰撞的大致时刻。接着建立碰撞判断模型，并利用叉乘法判断点是否位于三角形内部，从而判断两板凳是否相撞。在模型求解时，我们首先根据同心圆近似碰撞模型计算出碰撞时间大约为 410 s，然后在[400, 420]秒的时间区间内进行变步长搜索，最终确定**碰撞时间为 412.4739s**。我们将该时刻龙身的位置和速度数据保存在文件中，并**将龙头、第 1、51、101、151、201 节龙身以及龙尾的运动数据记录在表（4）与表（5）中。**

针对问题三，为简化计算，我们仅考虑龙头与第一节龙身的碰撞情况，并只计算进入调头空间前一段时间是否会发生碰撞。接着，采用二分法求解最小螺距，通过不断缩小搜索区间并判断碰撞情况，最终得到满足精度要求的最小螺距 **0.4000m。**

针对问题四，首先根据调头空间，我们将运动过程划分为四个阶段：盘入，调头第一段圆弧，调头第二段圆弧，盘出。针对每个阶段，分别建立了“板凳龙”的运动方程。由于运动轨迹关于极角为多值函数，在求解位置时，对极角的约束十分复杂，因此我们使用单值函数用于龙身位置和速度的求解。此外，我们证明了**调头曲线长度不变性**。最后，我们求解了特殊节点的位置和速度，并分析了龙身各节点的运动规律，并**将−100s、−50s、0s、50s、100s 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运动数据记录在表（6）与表（7）中。**

针对问题五，我们首先探究了“板凳龙”运动中把手最大速度的分布规律，分析曲率半径、速度与板凳方向夹角之间的关系，猜想最大速度出现位置不随龙头行进速度变化，出现在第二段小圆弧末端附近。为了验证这一规律，我们针对不同的龙头行进速度进行了数值计算，并选取了 200 个时间点进行分析。结果表明，最大速度的变化呈现周期性，且最大速度峰值均出现在第 100 至 125 个时间点之间的特定位置范围内。最终我们采用了二分法对该时间段进行搜索，通过不断迭代逼近，得到了龙头的最大行进速度为 **1.2462m/s**，误差控制在 0.0001m/s 之内。

关键词：**等距螺线；仿真模拟；微元法；二分法**

中国大学生在线 dxs.moe.gov.cn

一、 问题重述

1.1 问题背景

“板凳龙”是浙闽地区元宵节期间的传统民俗活动之一，表演形式类似舞龙，但使用板凳串联模拟龙形。整个表演过程中，“板凳龙”的行进需要呈现出蜿蜒曲折的形状，具有强烈的观赏性。表演的核心在于控制队伍的行进路线，使得整个龙队的盘入盘出流畅自然，避免出现队伍拥堵或碰撞的情况。此外，合理的速度控制和路径规划也是保证表演顺利进行的关键。

1.2 问题提出

“板凳龙”由 223 节板凳组成，包括 1 节龙头、221 节龙身、1 节龙尾。其中，龙头板长为 341cm，龙身和龙尾板长均为 220cm，所有板凳的板宽为 30cm；每节板凳上有两个孔，孔径为 5.5cm，孔的中心均距离板头 27.5cm；板凳通过把手连接，每节板凳上的前把手和后把手位于板凳的两端，连接处通过孔固定。

针对问题一，模拟“板凳龙”沿螺距为 55 cm 的等距螺线顺时针盘入的过程，给出从初始时刻到 300 秒为止，每秒”板凳龙”各把手的位置和速度，并将结果保存到文件 result1.xlsx 中。同时，提供 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数据。

针对问题二，在确保“板凳龙”不发生碰撞的情况下，计算能够盘入的最晚时刻并计算此时刻龙头及龙身各关键节点（同问题一）的前把手、龙尾后把手的位置和速度。

针对问题三，在“板凳龙”表演中，盘入后需要进行调头，调头空间为以螺线中心为圆心、直径为 9 米的圆形区域。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。

针对问题四，“板凳龙”沿螺距为 1.7m 的螺线盘入和盘出，调头空间为直径 9 m 的圆形区域，调头路径为由两段圆弧相切连接而成的 S 形曲线。确定是否可以调整圆弧，仍保持各部分相切，使得调头曲线变短。给出从-100s 开始到 100s 为止，每秒“板凳龙”各把手的位置和速度，将结果存放到文件 result4.xlsx 中，同时提供-100s、-50s、0s、50s、100s 时，龙头及龙身各关键节点（同问题一）的前把手、龙尾后把手的位置和速度。

针对问题五，“板凳龙”沿问题四设计的路径行进，要求计算龙头的最大行进速度，使得“板凳龙”各把手的速度均不超过 2m/s。

二、 问题分析

2.1 问题一的分析

针对问题一，我们假设龙头把手沿等距螺线运动，通过螺距确定螺线的极坐标方程，并建立极角与时间的微分方程描述运动；随后，我们利用相邻把手之间的距离关系，建立各把手位置的递推方程组。最后沿板凳方向分解速度，建立各把手速度的递推关系。通过数值求解，得到把手位置和速度的时间变化规律。

2.2 问题二的分析

针对问题二，为了预测“板凳龙”运动中碰撞的发生时间，并记录碰撞时刻的龙身位置和速度数据，我们首先建立了同心圆碰撞近似模型。该模型将板凳龙的螺线轨迹近似为两个同心圆，当内圈板凳最远点极径大于等于外圈板凳最内点极径时，认为

发生碰撞。

利用几何关系，估计碰撞时刻，并在其附近进行精细搜索。随后，我们对板凳龙的行进过程进行仿真模拟，利用叉乘法建立板凳龙的碰撞判断模型。最终，对估计区间进行变步长搜索，得到精确的碰撞时间。

2.3 问题三的分析

针对问题三，由于龙身位置出现的对称性，为简化问题，只考虑龙头与第一节板凳的碰撞情况。最后我们采用二分法求解满足最大速度条件的最小螺距，通过迭代调整螺距区间，最终得到最小螺距，确保龙头能够安全进入调头空间。

2.4 问题四的分析

针对问题四，首先我们分析了“板凳龙”调头轨迹，将其划分为四个阶段：盘入，调头第一段圆弧，调头第二段圆弧，盘出，然后讨论调头曲线的长度。针对每个阶段，我们分别建立了相应的运动方程，并基于问题一中位置和速度的递推模型，通过引入单值函数求解“板凳龙”龙头和龙身的准确位置和速度。

2.5 问题五的分析

针对问题五，在探究“板凳龙”最大速度分布规律时，龙头前把手离开第二段圆弧时最大速度出现明显峰值，之后呈现周期性变化。通过分析曲率半径和速度与板凳方向夹角的关系，我们得出结论：最大速度出现位置不随龙头行进速度变化，出现在第二段小圆弧末端附近。因此，我们确定最大速度出现在龙头前把手离开第二段圆弧附近的一段时间区间内，并采用二分法求解龙头行进的最大速度。

三、 模型假设

1. 假设板凳不可伸长，即沿板凳方向各点速度相同。
2. 假设各把手中心始终位于螺线上，忽略其他因素影响。
3. 假设“板凳龙”在行进过程中不会倒退。
4. 假设板凳始终平行于地面，不会发生倾斜，简化为二维运动模型。

四、 符号说明

表1 符号说明

符号	符号说明	单位
(r_i,θ_i)	第i节龙身前把手中心在极坐标系中的坐标	(m,rad)
(x_i,y_i)	第i节龙身前把手中心在平面直角坐标系中的坐标	(m,m)
d	等距螺线的间距	m
l	龙身前后把手的距离	m
L	龙头前后把手的距离	m
v_i	第i节龙身前把手的速度	m/s
D_l	孔的中心距离最近的板头的距离	m
D_h	龙身板宽的二分之一	m

五、 问题一模型的建立与求解

5.1 模型建立

5.1.1 龙头运动模型

根据题意，龙头前把手沿着等距螺线顺时针盘入。以螺线中心为极点，极轴指向出发点方向，建立极坐标系，则等距螺线方程为：

$$r(\theta)=\alpha\theta \tag{1}$$

其中α为常数，θ为螺线的极角，$r(\theta)$为螺线的径向距离。

当极角相差2π时，对应极径相差为螺距d，于是有：

$$r(\theta+2\pi)-r(\theta)=\alpha\cdot 2\pi=d \tag{2}$$

解得$\alpha=\frac{d}{2\pi}$。

因此，螺线的极坐标方程为：

$$r(\theta)=\frac{2\pi}{d}\theta \tag{3}$$

将其转化为平面直角坐标系下的参数方程，整理得：

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

利用微元法推理极角θ与行进时间t的关系。考虑t时刻 A 点运动到$t+dt$时刻 B 点的过程，dt充分小。

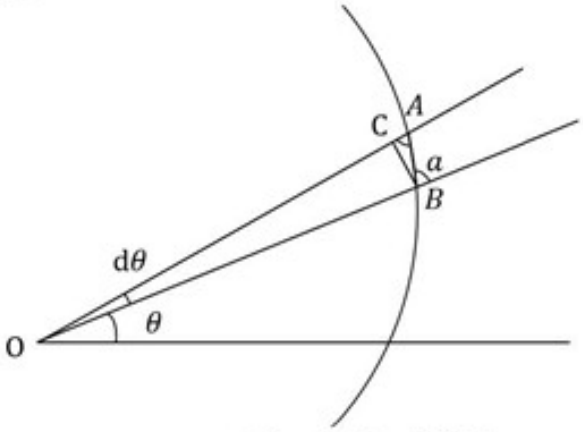


图1 龙头运动模型

根据图（1），有：

$$OC=OB=\alpha\theta \tag{5}$$

$$OA=\alpha(\theta+d\theta) \tag{6}$$

$$AC=OA-OC=\alpha d\theta \tag{7}$$

由于$d\theta$极小，故可以认为$OA\parallel OB$，$OA\perp BC$，且$BC=\overset{\frown}{BC}$，则有：

$$\tan\angle a=\tan\angle BAC=\frac{BC}{AC}=\frac{\alpha\theta d\theta}{\alpha d\theta}=\theta \tag{8}$$

$$BC=OB\cdot d\theta=\alpha\theta d\theta \tag{9}$$

根据$v=\frac{s}{t}$得：

$$v=\frac{AB}{dt}=\frac{AC}{\cos\angle a\cdot dt}=\frac{\alpha d\theta}{\cos\angle a\cdot dt} \tag{10}$$

将式（8）代入式（10），整理得微分方程：

$$\frac{d\theta}{dt}=\frac{v}{\alpha\sqrt{1+\theta^2}} \tag{11}$$

解得解析解：

$$\theta\sqrt{\theta^2+1}+\ln\left(\theta+\sqrt{\theta^2+1}\right)=\frac{2v}{\alpha}t+C \tag{12}$$

又根据题意，在初始时刻时，$\theta=32\pi$，计算得：

$$C=32\pi\sqrt{(32\pi)^2+1}+\ln\left(32\pi+\sqrt{(32\pi)^2+1}\right) \tag{13}$$

5.1.2 把手位置模型

记上文中的螺线方程（4）为$F(x,y)=0$，根据假设，各把手中心均位于螺线上，则各节板凳前把手中心(x_i,y_i)均满足螺线方程：

$$F(x_i,y_i)=0 \tag{14}$$

$$F(x_{i+1},y_{i+1})=0 \tag{15}$$

又根据题意，第i节龙身、第$i+1$节龙身前把手之间的距离即为第i节龙身前后把手的距离，记为l，于是有：

$$\sqrt{(x_i-x_{i+1})^2+(y_i-y_{i+1})^2}=l \tag{16}$$

此处将龙头看作第 0 节龙身，则：

$$\sqrt{(x_0-x_1)^2+(y_0-y_1)^2}=L \tag{17}$$

此外，考虑第$i+1$节龙身前把手运动的极角的范围：

$$0\leqslant\theta_{i+1}-\theta_i\leqslant\pi \tag{18}$$

综上，整理得各把手中心的位置方程组：

$$\begin{cases}F(x_i,y_i)=0\\F(x_{i+1},y_{i+1})=0\\\sqrt{(x_i-x_{i+1})^2+(y_i-y_{i+1})^2}=l\\0\leqslant\theta_{i+1}-\theta_i\leqslant\pi\end{cases} \tag{19}$$

根据方程组（19）整理得：

$$\theta_i^2+\theta_{i+1}^2-2\theta_i\theta_{i+1}\cos(\theta_i-\theta_{i+1})=\frac{l^2}{\alpha^2},\ 0\leqslant\theta_{i+1}-\theta_i\leqslant\pi \tag{20}$$

5.1.3 把手速度模型

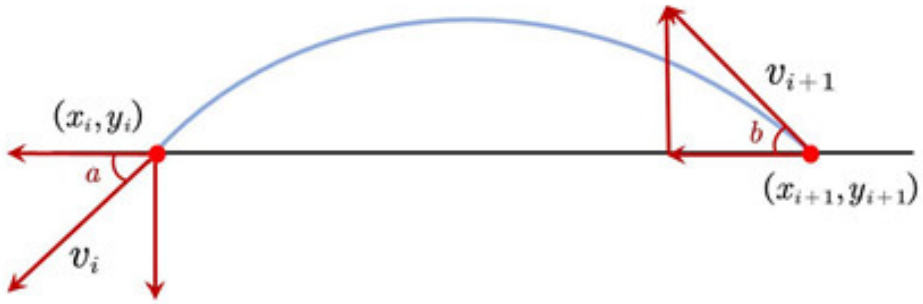


图2 把手速度模型

根据螺线的参数方程（4）计算得螺线的切线方向$\boldsymbol{e_i}=\frac{(x'(\theta_i),y'(\theta_i))}{\sqrt{(x'(\theta_i))^2+(y'(\theta_i))^2}}$，其中，

$$x'(\theta_i)=\alpha(\cos\theta_i-\theta\sin\theta_i) \tag{21}$$

$$y'(\theta_i)=\alpha(\sin\theta_i+\theta_i\cos\theta_i) \tag{22}$$

于是第i节龙身前把手的速度为：

$$v_i=|v_i|e_i \tag{23}$$

记第i节龙身后把手指向第i节板凳前把手的方向为第i节龙身的板凳方向：

$$e_{i,i+1}=(x_i-x_{i+1},y_i-y_{i+1}) \tag{24}$$

由于假设板凳不可伸缩，因此第i节板凳前把手中心沿板凳方向的分速度与第$i+1$节板凳前把手中心沿板凳方向的分速度相同，于是有：

$$v_i\cdot e_{i,i+1}=v_{i+1}\cdot e_{i,i+1} \tag{25}$$

5.2 模型求解结果

我们将参数$d=0.55\text{m}$，$v=1\text{m/s}$代入模型进行求解，以 1s 的时间间隔将龙身位置和速度数据记录在文件 result1.xlsx 中。

以 300s 时刻为例，我们绘制如图（3）所示的板凳龙位置图，此刻把手均位于螺线上，符合题目要求。

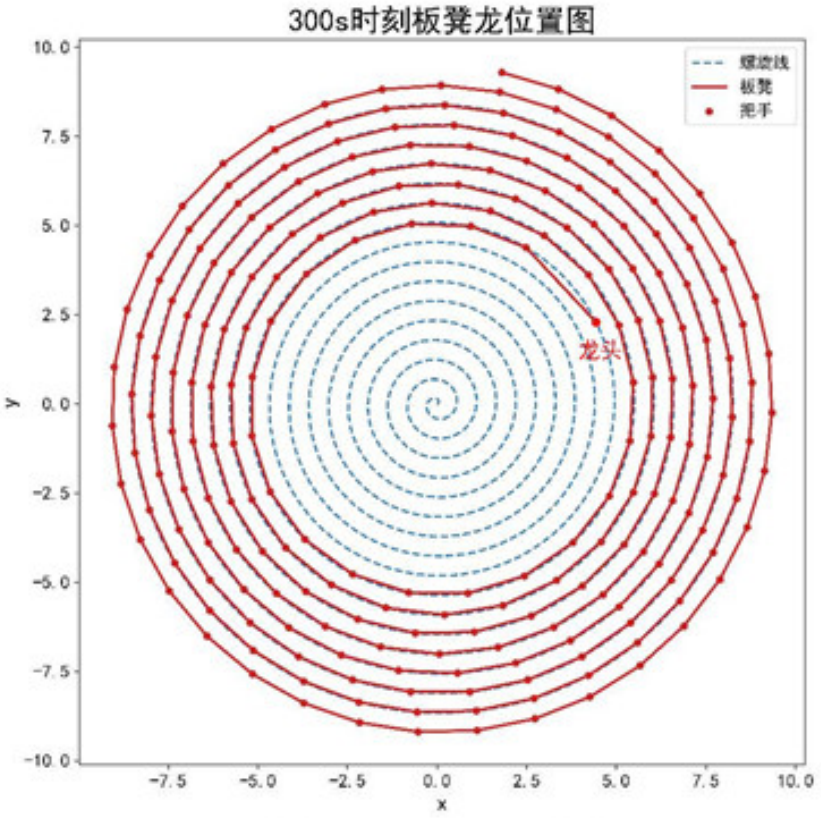


图3 300s“板凳龙”位置

根据题目要求，我们给出 0s、60s、120s、180s、240s、300s 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数据，结果如表（2），表（3）所示：

中国大学生在线 dxs.moe.gov.cn

表2 特殊节点位置结果

	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3 特殊节点速度结果

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

六、 问题二模型的建立与求解

6.1 模型建立

由等距螺线曲率半径公式[1]，

$$\frac{\rho}{r}=\frac{(\theta^2+1)^{\frac{3}{2}}}{(\theta^2+2)\theta}\to 1\ (\theta\to\infty) \tag{26}$$

因此，我们认为当θ较大时，ρ与r近似相等。于是建立如下同心圆碰撞近似模型。

6.1.1 同心圆碰撞近似模型

我们将板凳龙运动的螺线轨迹近似为两个同心圆，其中内圆半径为R，外圆半径为$R+d$，其中R可任意取值，如图（4）所示。

我们认为，当内圈板凳最远点极径$R^{(1)}$大于等于外圈板凳最内点极径$R^{(2)}$时，发生碰撞。根据同心圆的对称性，第一次碰撞可确定发生在龙头处。

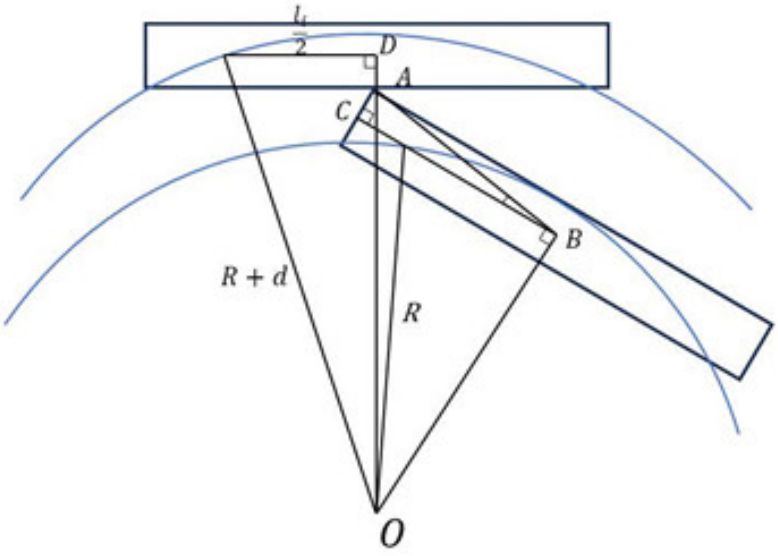


图4　同心圆轨迹碰撞示意图

记h_1为圆心到外圆板中轴线距离，h_2为圆心到内圆板中轴线距离，则根据勾股定理：

$$h_1 = OD = \sqrt{(R+d)^2 - \left(\frac{l}{2}\right)^2} \tag{27}$$

$$h_2 = OB = \sqrt{R^2 + \left(\frac{L}{2}\right)^2} \tag{28}$$

由于$R^{(1)}$为板凳移动时距离圆心最近的距离，因此我们过圆心向板凳下边缘作垂线，有：

$$R^{(1)} = h_1 - 0.15 \tag{29}$$

由于$R^{(2)}$为板凳移动时距离圆心最远的距离，则$R^{(2)}$即为圆心与板凳上边缘顶点之间的距离：

$$OA = R^{(2)} \tag{30}$$

当$OA = h_1 - 0.15$时，判定龙头和龙身相撞，下计算OA的长度：

根据几何关系，我们有：

$$\tan\angle ABC = \frac{AC}{BC} \tag{31}$$

$$AB = \sqrt{AC^2 + BC^2} \tag{32}$$

则$\angle ABC = \arctan\frac{AC}{BC}$，$\angle ABO = \frac{\pi}{2} + \arctan\frac{AC}{BC}$

根据余弦定理：$OA^2 = AB^2 + BO^2 - 2AB \cdot BO \cdot \cos\angle ABO$即可计算$OA$。

6.1.2 “板凳龙”碰撞仿真模拟

计算板凳四个顶点坐标

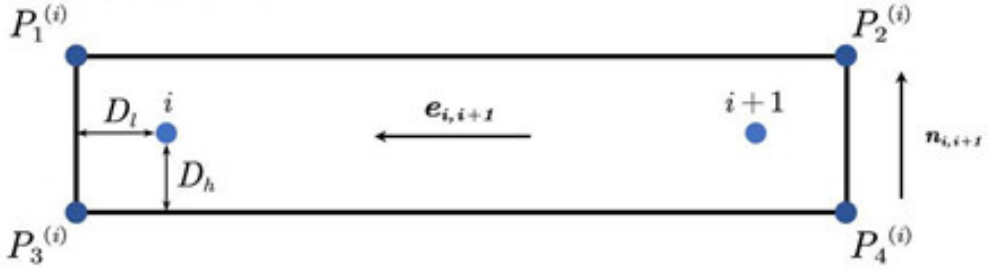


图5 板凳顶点示意图

由图（5），第i节龙身后把手指向第i节龙身前把手的方向为：

$$e_{i,i+1}=\frac{(x_i-x_{i+1},y_i-y_{i+1})}{l} \tag{33}$$

对应的法向量为：

$$n_{i,i+1}=\frac{(y_{i+1}-y_i,x_{i+1}-x_i)}{l} \tag{34}$$

计算顶点$P_1^{(i)}$、$P_2^{(i)}$、$P_3^{(i)}$、$P_4^{(i)}$的坐标：

$$\begin{cases} P_1^{(i)}=(x_i,y_i)-(l+D_l)\cdot e_{i,i+1}+D_h\cdot n_{i,i+1} \\ P_2^{(i)}=(x_i,y_i)+D_l\cdot e_{i,i+1}+D_h\cdot n_{i,i+1} \\ P_3^{(i)}=(x_i,y_i)+D_l\cdot e_{i,i+1}-D_h\cdot n_{i,i+1} \\ P_4^{(i)}=(x_i,y_i)-(l+D_l)\cdot e_{i,i+1}-D_h\cdot n_{i,i+1} \end{cases} \tag{35}$$

当$i=0$时，l替换为L。

碰撞判断模型

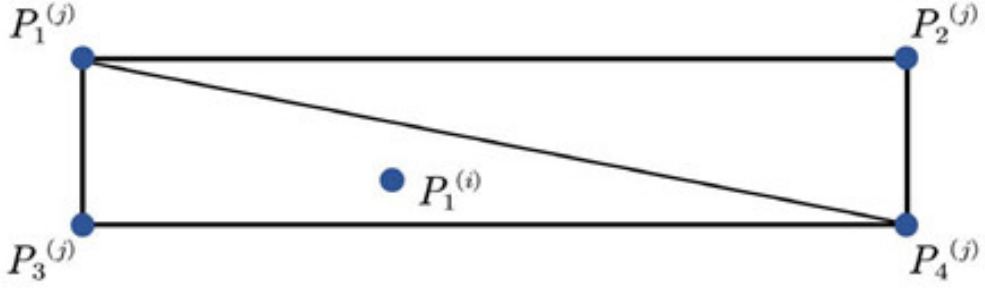


图6 碰撞判断模型

当第i节龙身的顶点$P_1^{(i)}$、$P_2^{(i)}$在$\square P_1P_2P_3P_4^{(j)}$内部时，我们判定第$i$节龙身与第$j$节龙身碰撞。如图（6）将$\square P_1P_2P_3P_4^{(j)}$沿对角线划分为两个三角形，分别判断$P_1^{(i)}$、$P_2^{(i)}$是否在两个三角形内部，以判断$P_1^{(i)}$是否在$\triangle P_1P_3P_4^{(j)}$内部为例进行说明。

判断一个点是否位于三角形内部，可以通过叉乘法[2]来实现。构造从三角形顶点到点$P_1^{(i)}$的向量，并计算三个向量之间的叉乘，如果叉乘结果都为正，或者都为负，则点$P_1^{(i)}$位于三角形内部。

$$\begin{cases} \left(\overrightarrow{P_1^{(j)}P_3^{(j)}}\times\overrightarrow{P_1^{(j)}P_1^{(i)}}\right)\cdot\left(\overrightarrow{P_3^{(j)}P_4^{(j)}}\times\overrightarrow{P_3^{(j)}P_1^{(i)}}\right)>0 \\ \left(\overrightarrow{P_3^{(j)}P_4^{(j)}}\times\overrightarrow{P_3^{(j)}P_1^{(i)}}\right)\cdot\left(\overrightarrow{P_4^{(j)}P_1^{(j)}}\times\overrightarrow{P_4^{(j)}P_1^{(i)}}\right)>0 \\ \left(\overrightarrow{P_4^{(j)}P_1^{(j)}}\times\overrightarrow{P_4^{(j)}P_1^{(i)}}\right)\cdot\left(\overrightarrow{P_1^{(j)}P_3^{(j)}}\times\overrightarrow{P_1^{(j)}P_1^{(i)}}\right)>0 \end{cases} \tag{36}$$

中国大学生在线 dxs.moe.gov.cn

6.2 模型求解方法

根据同心圆碰撞近似模型，我们计算出发生碰撞的上界时间 420s。我们在时间区间 [400s, 420s] 之间变步长遍历搜索[3]，计算碰撞时间，具体遍历步骤如下图（7）所示：

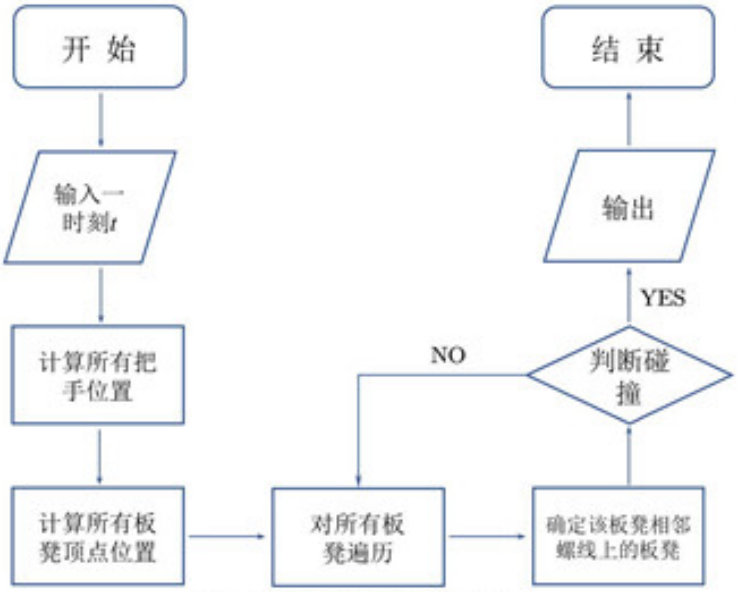


图7 判断碰撞算法流程图

6.3 模型求解结果

我们首先以 $\Delta t = 1\text{s}$ 的步长进行遍历，缩小碰撞时间范围至 [412s, 413s]；最后以 $\Delta t = 0.0001\text{s}$ 变步长进行精细搜索，计算板凳发生碰撞时间为 412.4739s。按照题目要求，我们将此刻龙身位置和速度数据记录在文件 result2.xlsx 中，并给出此刻龙头前把手、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数据，结果如表（4），表（5）所示：

表4 特殊节点位置结果

龙头 x(m)	1.209983	第 101 节龙身 y(m)	-5.880132
龙头 y(m)	1.942749	第 151 节龙身 x(m)	0.968780
第 1 节龙身 x(m)	-1.643745	第 151 节龙身 y(m)	-6.957487
第 1 节龙身 y(m)	1.753440	第 201 节龙身 x(m)	-7.893170
第 51 节龙身 x(m)	1.281259	第 201 节龙身 y(m)	-1.230704
第 51 节龙身 y(m)	4.326570	龙尾（后）x(m)	0.956277
第 101 节龙身 x(m)	-0.536307	龙尾（后）y(m)	8.322728

表5 特殊节点速度结果

龙头 (m/s)	第 1 节龙身(m/s)	第 51 节龙身(m/s)	第 101 节龙身(m/s)	第 151 节龙身(m/s)	第 201 节龙身(m/s)	龙尾（后）(m/s)
1.000000	0.991551	0.976858	0.974550	0.973608	0.973096	0.972938

中国大学生在线 dxs.moe.gov.cn

七、 问题三模型的建立与求解

模型建立

经过分析，螺距与碰撞时龙头前把手的极径呈现明显的单调递减关系。因此我们只需保证，在碰撞时龙头前把手的极径小于等于调头空间的半径的条件下，求解最小螺距。故建立如下优化模型：

$$obj:\min\ d \tag{37}$$

$$s.t.\ d\in S \tag{38}$$

其中，S 表示龙头前把手进入调头空间前，未发生碰撞时螺距 d 组成的集合。

模型求解

在此基础上，采用二分法[4] 来求解最小螺距，具体步骤如下：

1. **参数初始化：**首先，设定最小螺距的初始搜索区间，设定最小螺距的下界 $l=0.3\text{m}$ 和上界 $r=0.55\text{m}$，精度设为 $\delta=10^{-4}\text{m}$。
2. **最大速度条件验证：**计算中间值 $mid=(l+r)/2$，并以 0.01s 为时间步长，判断时间区间 $(t_{in}-10,t_{in})$ 内是否发生碰撞。
3. **搜索区间调整：**如果发生碰撞，更新下界 $l=mid$；否则，更新上界 $r=mid$。
4. **迭代逼近：**重复上述操作，直到 $r-l$ 满足精度要求，此时，根据最终的搜索区间得到最小螺距 $d=(l+r)/2$。

通过二分法求解得最小螺距为 **0.4000m**，误差控制在 0.0001m 之内。

八、 问题四模型的建立与求解

8.1 调头路径

8.1.1 调头路径几何特性

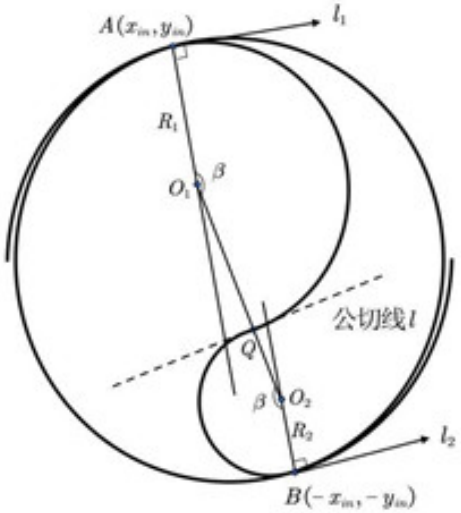


图8 调头空间示意图

记盘入螺线与调头空间边缘交点为 (x_{in},y_{in})，由题意，盘出螺线与盘入螺线关于螺线中心呈中心对称，则盘出螺线与调头空间边缘交点为 $(-x_{in},-y_{in})$，且切线 $l_1\parallel l_2$。由于 (x_{in},y_{in}) 处的切线方向为 $e=(x_{in}',y_{in}')$，法线方向为 $n=(-y_{in}',x_{in}')$，记前一段圆弧的圆心 O_1 坐标为：

$$O_1=(x_{in},y_{in})+2\lambda\cdot n \tag{39}$$

其中，λ 为参数。由于前一段圆弧的半径是后一段圆弧半径的 2 倍，则后一段圆弧

的圆心坐标O_2为：

$$O_2=(-x_{in},-y_{in})-\lambda\cdot\boldsymbol{n} \tag{40}$$

此外，由于$AO_1\perp l_1$，$BO_2\perp l_2$，$l_1\parallel l_2$，则$AO_1\parallel BO_2$。

记R_1为前一段圆弧的半径，R_2为后一段圆弧的半径，于是有$R_1=2\lambda|\boldsymbol{n}|$，$R_2=\lambda|\boldsymbol{n}|$。

由于调头路径中的两段圆弧相切，则$QO_1\perp l$，$QO_2\perp l$，O_1、Q、O_2三点共线，两圆弧圆心O_1、O_2之间的距离等于两圆弧半径之和：

$$|O_1O_2|=3\lambda|\boldsymbol{n}| \tag{41}$$

记β为圆弧圆心角，根据几何关系和三角函数关系，我们有：

$$\sin\frac{\beta}{2}=\frac{1}{6}\cdot\frac{|AB|}{R_2} \tag{42}$$

解得：

$$\beta=2\arcsin\frac{1}{6}\frac{|AB|}{R_2} \tag{43}$$

其中，$\beta\in\left(\frac{\pi}{2},\pi\right)$。

8.1.2 调头曲线长度

从图（8）中截取圆弧部分，如图（9）所示，进行调头曲线长度分析。

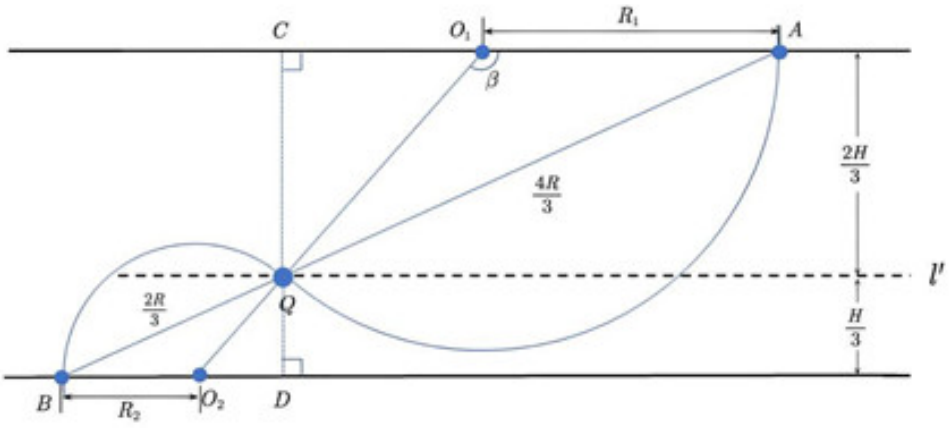


图9 调头空间内几何关系示意图

记AO_1与BO_2之间的距离为H，根据上文，我们有$AO_1\parallel BO_2$，$AO_1=2BO_2$。过O点做AO_1、BO_2的垂线OC、OD，于是有$OC=2H/3$，$OD=H/3$，则此时$\sin\angle O_1AO=H/2R$为一定值，因此$\beta=\pi-2\angle O_1AO$恒不变。弧长$C=\beta(R_1+R_2)$，其中$R_1+R_2=H/\sin\beta$为一定值。

因此，**若进入调头空间后立刻开始沿圆弧调头，则无法通过调整圆弧，仍保持各部分相切，使得调头曲线变短，此时圆弧长度为 13.621m。**

若进入调头空间后仍继续盘入，则可通过调整圆弧半径缩短调头曲线。具体分析如下：

定义龙头开始调头的径向距离为调头半径，记为r；又弧长$C=H\beta/\sin\beta$，其中$\beta=\pi-2\arcsin(H/2r)$，可知弧长与调头半径正相关。根据假设，“板凳龙”在行进过程中不会倒退，小圆弧的直径大于等于龙头前后把手之间的距离，则$2\min\{r_1,r_2\}\geq L$。针对不同约束，可对最小弧长进行求解：

若仅调整调头半径，而不改变圆弧半径比例，调头半径最短为 4.281m，此时圆弧长度为 12.936m；

若同时调整调头半径和圆弧半径比例，当比例为 1:1 时，调头半径最短为 2.847m，此时圆弧长度为 8.443m。

8.2 “板凳龙”位置速度模型的建立与求解

8.2.1 模型建立

我们将“板凳龙”调头过程分为四个部分，分别为龙头进入盘入，调头第一段圆弧，调头第二段圆弧，盘出。记$F(t)=(x,y)$为“板凳龙”运动轨迹的分段函数。

在龙头进入调头空间前，即$-100\ \mathrm{s}<t<0\ \mathrm{s}$时，“板凳龙”运动方程满足螺线参数方程（4）：

$$F(t)=(\alpha\theta(t)\cos(\theta(t)),\alpha\theta(t)\sin(\theta(t))) \tag{44}$$

根据式（11）的解析解，修改初始条件为$\theta\big|_{t=0}=\theta_{in}$，其中$\theta^{*}$表示龙头进入调头空间时的极角。此时，$\theta$满足：

$$\begin{aligned}&\theta\sqrt{\theta^2+1}+\ln\left(\theta+\sqrt{\theta^2+1}\right)=\frac{2v}{\alpha}t+C_1\\&C_1=\theta_{in}\sqrt{{\theta_{in}}^2+1}+\ln\left(\theta_{in}+\sqrt{{\theta_{in}}^2+1}\right)\end{aligned} \tag{45}$$

当“板凳龙”进入第一段圆弧，即$0<t<\frac{\beta}{\omega_1}$时，由于此时“板凳龙”沿第一段圆弧以角速度$\omega_1=\frac{v}{R_1}$做顺时针运动，则：

$$F(t)=\begin{pmatrix}x_{in}-2\lambda {y_{in}}'+R_1\cos(\gamma_1-\omega_1 t)\\y_{in}-2\lambda {x_{in}}'+R_1\sin(\gamma_1-\omega_1 t)\end{pmatrix}^{T} \tag{46}$$

其中，γ_1满足$\cos\gamma_1=-\frac{{y_{in}}'}{\sqrt{{x_{in}}'^2+{y_{in}}'^2}}$，$\sin\gamma_1=\frac{{x_{in}}'}{\sqrt{{x_{in}}'^2+{y_{in}}'^2}}$。

当“板凳龙”进入第二段圆弧，即$\frac{\beta}{\omega_1}<t<\beta\left(\frac{1}{\omega_1}+\frac{1}{\omega_2}\right)$时，由于此时“板凳龙”沿第二段圆弧以角速度$\omega_2=\frac{v}{R_2}$做逆时针运动，于是有：

$$F(t)=\begin{pmatrix}-x_{in}+\lambda {y_{in}}'+R_2\cos\left(\gamma_2+\omega_2\left(t-\frac{\beta}{\omega_1}\right)\right)\\-y_{in}-\lambda {x_{in}}'+R_2\sin\left(\gamma_2+\omega_2\left(t-\frac{\beta}{\omega_1}\right)\right)\end{pmatrix}^{T} \tag{47}$$

其中，$\gamma_2=\gamma_1-\beta+\pi$。

在“板凳龙”离开调头空间后，即$\beta\left(\frac{1}{\omega_1}+\frac{1}{\omega_2}\right)<t<100\ \mathrm{s}$时，由于盘出螺线与盘入螺线关于螺线中心呈中心对称，有：

$$F(t)=(\alpha\theta(t)\cos(\theta(t)+\pi),\alpha\theta(t)\sin(\theta(t)+\pi)) \tag{48}$$

与第一种情况一致，修改初始条件为$\theta\Big|_{t=\beta\left(\frac{1}{\omega_1}+\frac{1}{\omega_2}\right)}=\theta_{in}$，其中$\theta_{in}$表示龙头进入调头空间时的极角。此时，$\theta$满足：

$$\theta\sqrt{\theta^2+1}+\ln\left(\theta+\sqrt{\theta^2+1}\right)=\frac{2v}{\alpha}t+C_2$$
$$C_1=\theta_{in}\sqrt{\theta_{in}^2+1}+\ln\left(\theta_{in}+\sqrt{\theta_{in}^2+1}\right)-\frac{2v\beta}{\alpha}\left(\frac{1}{\omega_1}+\frac{1}{\omega_2}\right) \tag{49}$$

把手位置模型

由于“板凳龙”的运动轨迹是关于θ的多值函数，因为我们考虑将$F(t)$看作其余龙身的单值参数方程，将其代入各把手中心的位置方程组（19）：

$$\begin{cases}F(t_i)=0\\F(t_{i+1})=0\\D(t_i,\ t_{i+1})=l_i\\t_i-t_{i+1}>0\end{cases} \tag{50}$$

其中，$F(t_i)=0$表示第i节龙身把手在螺线轨迹上。

把手速度模型

在问题一的基础上，根据“板凳龙”位置分段函数$F(t)$调整龙头运动的切线方程$F'(t)$得：

$$F'(t)=(x',y')=\begin{cases}\begin{pmatrix}\alpha(\cos\theta(t)-\theta(t)\sin\theta(t))\\\alpha(\sin\theta(t)+\theta(t)\cos\theta(t))\end{pmatrix}^T,\ -100<t\leqslant 0\\(y(t)-y_{in}-2\lambda x_{in}',\ -x(t)+x_{in}-2\lambda y_{in}'),\ 0<t<\dfrac{\beta}{\omega_1}\\(-y(t)-y_{in}-2x_{in}',x(t)+x_{in}-2y_{in}'),\ \dfrac{\beta}{\omega_1}<t<\beta\left(\dfrac{1}{\omega_1}+\dfrac{1}{\omega_2}\right)\\\begin{pmatrix}\alpha(\cos(\theta(t)+\pi)-\theta(t)\sin(\theta(t)+\pi))\\\alpha(\sin(\theta(t)+\pi)+\theta(t)\cos(\theta(t)+\pi))\end{pmatrix}^T,\ \beta\left(\dfrac{1}{\omega_1}+\dfrac{1}{\omega_2}\right)<t<100\end{cases} \tag{51}$$

将$F'(t)$代入把手速度模型求解即可。

8.2.2 模型求解结果

根据**题目所给的调整路径**，我们对龙身位置和速度进行求解，并将其记录在文件result4.xlsx中，同时给出-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，结果如表（6），表（7）所示：

表6 特殊节点位置结果

	−100 s	−50 s	0 s	50 s	100 s
龙头 x(m)	7.778034	6.608301	-2.711856	1.332696	-3.157229
龙头 y(m)	3.717164	1.898865	-3.591078	6.175324	7.548511
第 1 节龙身 x(m)	6.209273	5.366911	-0.063534	3.862265	-0.346890
第 1 节龙身 y(m)	6.108521	4.475403	-4.670888	4.840828	8.079166
第 51 节龙身 x(m)	-10.608038	-3.629945	2.459962	-1.671385	2.095033
第 51 节龙身 y(m)	2.831491	-8.963800	-7.778145	-6.076713	4.033787
第 101 节龙身 x(m)	-11.922761	10.125787	3.008493	-7.591816	-7.288774
第 101 节龙身 y(m)	-4.802378	-5.972247	10.108539	5.175487	2.063875
第 151 节龙身 x(m)	-14.351032	12.974784	-7.002789	-4.605165	9.462513
第 151 节龙身 y(m)	-1.980993	-3.810357	10.337482	-10.386988	-3.540357
第 201 节龙身 x(m)	-11.952942	10.522509	-6.872842	0.336952	8.524374

第 201 节龙身 y(m)	10.566998	-10.807425	12.382609	-13.177610	8.606933
龙尾（后）x(m)	-1.011059	0.189809	-1.933627	5.859094	-10.980157
龙尾（后）y(m)	-16.527573	15.720588	-14.713128	12.612894	-6.770006

表7 特殊节点速度结果

	−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	0.998687	1.000363	1.000124
第 51 节龙身(m/s)	0.999346	0.998642	0.995134	0.949935	1.003966
第 101 节龙身(m/s)	0.999091	0.998248	0.994448	0.948482	1.096263
第 151 节龙身(m/s)	0.998944	0.998047	0.994156	0.948038	1.095306
第 201 节龙身(m/s)	0.998849	0.997925	0.993994	0.947823	1.094933
龙尾（后）(m/s)	0.998817	0.997885	0.993944	0.947760	1.094833

九、 问题五模型的建立与求解

9.1 最大速度分布规律探究

当龙头速度为 1m/s 时，我们探索最大速度随时间变化的情况，如图（10）所示：

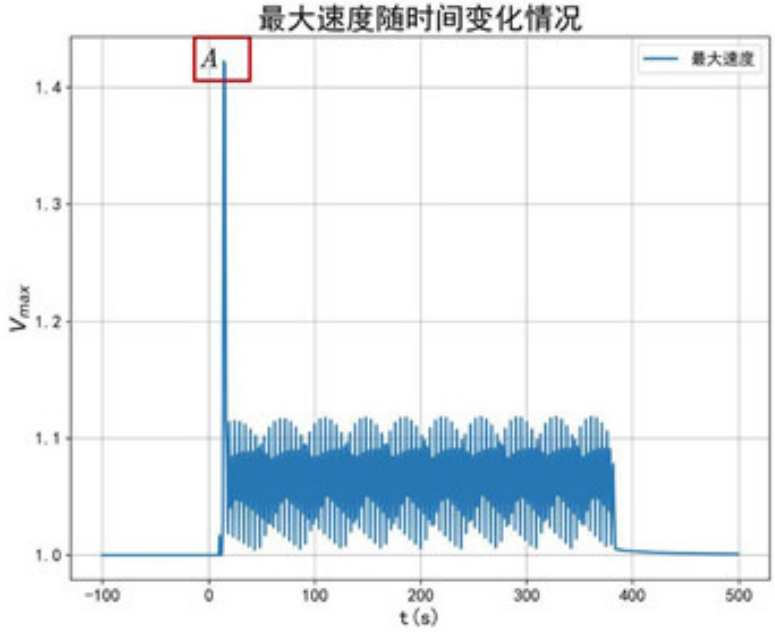


图10 最大速度随时间变化情况

我们发现图（10）中出现明显峰值 A 点，且 A 点之后最大速度的变化呈现周期性。经过计算，A 点时刻为“板凳龙”龙头前把手离开第二段圆弧时的时刻。下面说明出现明显峰值 A 点的原因。

首先我们分析曲率半径ρ和速度与板凳方向夹角φ之间的关系：

中国大学生在线 dxs.moe.gov.cn

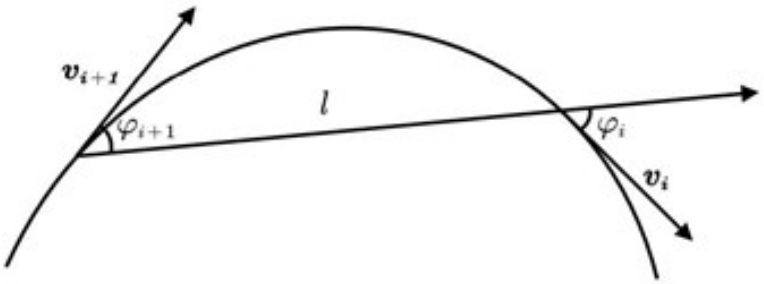


图11 探究曲率半径与速度变化之间的关系

根据几何关系得：

$$\sin\varphi = \frac{l}{2\rho} \tag{52}$$

由于第i节龙身前把手中心沿板凳方向的分速度与第$i+1$节龙身前把手中心沿板凳方向的分速度相同，则：

$$v_i \cdot \cos\varphi_i = v_{i+1} \cdot \cos\varphi_{i+1} \tag{53}$$

整理得：

$$v_{i+1} = \frac{\cos\varphi_i}{\cos\varphi_{i+1}} v_i = \sqrt{\frac{1-\frac{l^2}{4\rho_i^2}}{1-\frac{l^2}{4\rho_{i+1}^2}}} \cdot v_i \tag{54}$$

$$\Delta v_i = v_{i+1} - v_i \approx \mu \cdot (\rho_i{}^2 - \rho_{i+1}{}^2) \tag{55}$$

其中，μ可近似为常数。

离开第二段小圆弧位置，第二段圆弧曲率半径为ρ_{i+1}，盘出螺线曲率半径为ρ_i，相差最为明显，于是在此处附近的相邻节点速度变化较大，所以在此处最大速度出现明显峰值。

根据上述分析，我们猜想，最大速度出现位置不随龙头行进速度变化，出现在龙头前把手离开第二段小圆弧的时刻附近。于是我们针对不同的龙头行进速度对最大速度出现位置的影响进行分析，记龙头前把手离开第二段小圆弧的时刻为t_{out}，以 0.1s 为步长，绘制$(t_{out}-10, t_{out}+10)$这一时间段中最大速度的图像如图（12）所示：

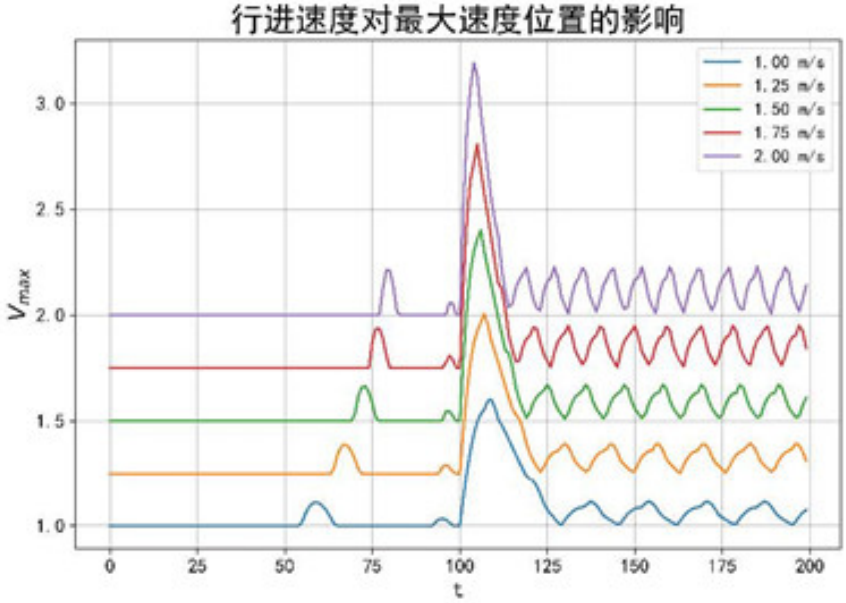


图12　行进速度对最大速度位置的影响

由图（12）可知，最大速度峰值出现后的最大速度变化呈时间周期性。此外，我们发现最大速度均在在 100～125 时刻的附近的位置范围内出现，该时间段对应龙头前把手离开第二段小圆弧的时间区间。因此，我们印证猜想，接下来我们对该区域进行精细搜索。

9.2 模型建立

由上述分析，建立优化模型：

$$obj:\max\ v \tag{56}$$

$$s.t.\ v_{\max} \leqslant 2 \tag{57}$$

其中，$v_{\max}$ 表示龙头前把手速度为 v 是“板凳龙”行进过程中，把手出现的最大速度。

9.3 模型求解结果

根据**最大速度分布规律探究**的分析，我们发现当“板凳龙”龙头前把手离开第二段圆弧时，出现最大速度。根据计算，调头空间内的圆弧长度为 $C=13.621\text{m}$；由 $t=s/v$ 估算最大速度时间区间，以 $\Delta t=0.001\text{s}$ 为步长，计算 $(t-0.5,t+2.5)$ 内的 $v_{\max}$。

我们采用二分法进行求解。具体步骤如下：

1. **参数初始化**：首先，设定最大速度的初始搜索区间，设定最大速度的下界 $l=1\text{m/s}$ 和上界 $r=2\text{m/s}$，精度要求为 $\delta=10^{-4}\text{m/s}$。
2. **最大速度条件验证**：计算中间值 $mid=(l+r)/2$，并计算 $v_{\max}$ 与 2m/s 进行比较。
3. **搜索区间调整**：如果 $v_{\max}>2\text{m/s}$，更新上界 $r=mid$；否则，更新下界 $l=mid$。
4. **迭代逼近**：重复上述操作，直到 $r-l$ 满足精度要求，此时，根据最终的搜索区间得到龙头的最大行进速度 $v=(l+r)/2$。

解得龙头的最大行进速度为 **1.2462m/s**，其中误差控制在 0.0001 m/s 之内。

十、 模型检验

为检验运动模型的正确性，本节针对问题一对位置和速度模型分别进行验证。

10.1 位置模型检验

我们将螺线的极径近似为曲率半径，估计螺线中心为圆心，建立如下近似圆估计模型：

$$r \cdot \frac{d\theta}{dt} = v \tag{58}$$

其中，r为螺线的极径。代入问题一的初始条件$\theta|_{t=0} = 32\pi$，解得：

$$\theta = \sqrt{(32\pi)^2 - \frac{4\pi v}{r}t} \tag{59}$$

计算其结果，与问题一的求解结果进行比较，计算误差：

$$\varepsilon_j = \sum_i dis_{ij} \tag{60}$$

其中，dis_{ij}表示j时刻两个求解结果中第i节龙身前把手位置之间的距离。误差结果如下图（13）所示：

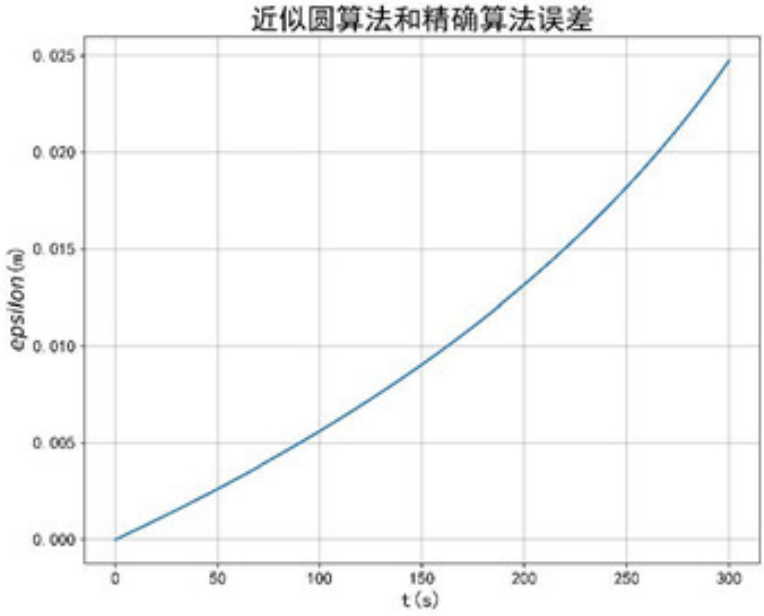


图13 求解结果位置误差

由图（13）可知，误差ε随时间t增大，这是由于随着极径变短，曲率半径估计误差偏大引起的。

10.2 速度模型检验

在问题一中，我们利用板凳的不可伸缩性，通过切线方程求解速度的理论值；此处我们利用差分算法进行速度近似值求解：

利用差商近似微商，即$v = \frac{ds}{dt} \approx \frac{\Delta s}{\Delta t}$，取$\Delta t = 10^{-4}$s，计算结果与问题一的求解结果进行比较，计算误差，最大误差数量级为10^{-8}m/s。

中国大学生在线 dss.moe.gov.cn

十一、 模型优缺点

11.1 模型优点

1. 利用微元法获得微分方程，并求得解析解，避免了数值求解时带来的误差积累，提高求解效率和精度。
2. 问题二采取同心圆近似螺线估计碰撞时间，极大地减少了后续对时间遍历的搜索空间，减小求解成本；同时估计结果与真实结果较为接近，误差在1%以内。
3. 本文针对问题的最优子结构进行分析，分别采取二分法、变步长搜索等方法进行求解，保证结果的正确性。

11.2 模型缺点

1. 对于“板凳龙”运动这一连续过程，本文仅对离散时刻进行碰撞判定。
2. 考虑到实际问题中的随机因素存在误差。

中国大学生在线 dxs.moe.gov.cn

参考文献

[1] https://blog.csdn.net/BUAAer_xuyang/article/details/135187865

[2] https://blog.csdn.net/adolph_yang/article/details/79123304

[3] 贾宝新,李峰,潘一山,等.基于变步长加速搜索的微震源定位方法[J].岩土力学,2022,43(03):843-856.DOI:10.16285/j.rsm.2021.0872.

[4] 刘春燕,闫广峰,林成,等.基于二分法的 KPCA 核参数优选[J].内江师范学院学报,2024,39(02):71-76.DOI:10.13603/j.cnki.51-1621/z.2024.02.012.

中国大学生在线 dxs.moe.gov.cn

附录

附录一 支撑文件列表

Problem1_1.py	# 圆半径近似计算位置和速度
Problem1_2.py	# 精确计算位置和速度
Problem1_3.py	# 差分算法计算速度
Problem1_4.py	# 误差对比
Problem2_1.py	# 同心圆估计碰撞时间
Problem2_2.py	# 计算碰撞时刻，保存数据，绘制示意图
Problem3_1.py	# 二分法计算最小螺距
Problem4_1.py	# 精确计算问题四
Problem4_2.py	# 差分算法计算速度
Problem4_3.py	# 计算最短圆弧
Problem5_1.py	# 二分法求最大速度
Plot1_1.py	# 绘制 300s 时刻的精确版本的板凳龙位置图
Plot4_1.py	# 绘制调整路径示意图
Plot5_1.py	# 绘制最大速度随时间变化分布图
Plot5_2.py	# 绘制最大速度随速度变化分布图
Table.py	# 导出论文结果
result1.xlsx	# 问题一结果
result2.xlsx	# 问题二结果
result4.xlsx	# 问题四结果

中国大学生在线 dxs.moe.gov.cn

附录二 程序代码

本文采用 Python 进行编程。

第一问精确计算位置和速度（Problem1_2.py）

```
# 问题一：求解各把手位置和速度
import numpy as np
import pandas as pd
from numpy import pi, sin, cos, sqrt, log
from scipy.optimize import root

# 参数
d = 0.55                                                    # 螺距(m)
v = 1                                                       # 龙头行进速度(m/s)

C = (32 * pi) * sqrt((32*pi)**2 + 1) + log((32 * pi) +
sqrt((32*pi)**2 + 1))
alpha = d / (2 * pi)

# 给定时刻 t,计算龙头的位置
def head(t):
    theta = root(lambda x: x * sqrt(x**2 + 1) + log(x + sqrt(x**2 +
1)) - C + 2*v*t/alpha, 10).x[0]
    return theta * alpha * cos(theta), theta * alpha * sin(theta),
theta

# 给定龙头的位置,计算各把手的位置
def postition(theta):
    x_post, y_post, theta_post = [theta * alpha * cos(theta)], [theta
* alpha * sin(theta)], [theta]

    # 计算第一节龙身前把手
    def func(t1):
        t0, l = theta, 2.86
        return t1**2 + t0**2 - 2*t1*t0*cos(t1 - t0) - (l / alpha) ** 2

    t1 = root(func, theta+1).x[0]
    x_post.append(t1 * alpha * cos(t1)); y_post.append(t1 * alpha *
sin(t1)); theta_post.append(t1)

    # 计算后续的把手
    for _ in range(222):
        def func(t):
            l = 1.65
            return t**2 + t1**2 - 2*t*t1*cos(t - t1) - (l / alpha) **
2
```

```
        t1 = root(func, t1+1).x[0]
        x_post.append(t1 * alpha * cos(t1)); y_post.append(t1 * alpha
* sin(t1)); theta_post.append(t1)
    return x_post, y_post, theta_post

# 给定龙头的速度和各把手的位置,计算各把手的速度
def speed(x, y, theta):
    v1 = 1
    speed = [1]

    for i in range(223):
        d1 = np.array([cos(theta[i]) - theta[i]*sin(theta[i]),
sin(theta[i]) +
theta[i]*cos(theta[i])])                                    # 上一把手速
度方向
        d1 = d1 / np.linalg.norm(d1)
        d =  np.array([cos(theta[i+1]) - theta[i+1]*sin(theta[i+1]),
sin(theta[i+1]) + theta[i+1]*cos(theta[i+1])])                          #
这一把手速度方向
        d = d / np.linalg.norm(d)
        vector = np.array([x[i+1] - x[i], y[i+1] - y[i]])    # 板凳方向
        v1 = root(lambda v: np.dot(v1*d1, vector) - np.dot(v*d,
vector), v1).x[0]
        speed.append(v1)
    return speed

# 主函数
if __name__ == '__main__':
    # 划分时间
    t = np.arange(0, 301, step=1)

    post_result = pd.DataFrame(columns = t, index = range(224*2))
    speed_result = pd.DataFrame(columns = t, index = range(224))

    # 计算龙头前把手的位置和速度
    head_vec = np.vectorize(head)
    head_post_x, head_post_y, head_theta = head_vec(t)

    # 计算各时刻把手的位置和速度
    for i, t in enumerate(head_theta):
        # 计算把手位置
        handle_post_x, handle_post_y, handle_theta =
postition(t)
        handle_post = []
```

```
        for j in range(224):
            handle_post.append(handle_post_x[j]);
handle_post.append(handle_post_y[j])
        post_result[i] = handle_post

        # 计算把手速度
        handle_speed = speed(handle_post_x, handle_post_y,
handle_theta)
        speed_result[i] = handle_speed

    # 保存答案
    post_result.round(6).to_excel('问题一_位置.xlsx')
    speed_result.round(6).to_excel('问题一_速度.xlsx')
```

第二问变步长搜索碰撞时间（Problem2_2.py）

```
# 问题二:变步长求解碰撞时刻
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import root
from numpy import pi, sin, cos, sqrt, arcsin, log

# 参数
d = 0.55                                                # 螺距(m)
v = 1                                                   # 龙头行进速度(m/s)

C = (32 * pi) * sqrt((32*pi)**2 + 1) + log((32 * pi) +
sqrt((32*pi)**2 + 1))
alpha = d / (2 * pi)

# 给出螺旋线方程
def helix(theta):
    return np.array([alpha * theta * cos(theta), alpha * theta *
sin(theta)])

# 判断点在矩形内部
def Point_in_Polygon(point, polygon):
    A, B, C, D = polygon[0], polygon[1], polygon[2], polygon[3]
    P = np.array([point[0], point[1]])

    flag1, flag2 = 0, 0

    AB, BC, CA = B-A, C-B, A-C
    AP, BP, CP = P-A, P-B, P-C
```

```
    if np.sign(AB[0] * AP[1] - AP[0] * AB[1]) == np.sign(BC[0] *
BP[1] - BP[0] * BC[1]) and  np.sign(BC[0] * BP[1] - BP[0] * BC[1]) ==
np.sign(CA[0] * CP[1] - CP[0] * CA[1]):
        flag1 = 1

    AD, DC, CA = D-A, C-D, A-C
    AP, DP, CP = P-A, P-D, P-C
    if np.sign(AD[0] * AP[1] - AP[0] * AD[1]) == np.sign(DC[0] *
DP[1] - DP[0] * DC[1]) and np.sign(DC[0] * DP[1] - DP[0] * DC[1]) ==
np.sign(CA[0] * CP[1] - CP[0] * CA[1]):
        flag2 = 1

    if flag2 or flag1:
        return True
    else:
        return False

# 根据前把手计算四个点:
def vertices(front, back, L):
    Dl = 0.275; Dh = 0.15
    handle1 = helix(front)
    handle2 = helix(back)
    e = (handle1 - handle2) / (L - 0.55)
    n = np.array([-e[1], e[0]])
    points = np.array([handle1 - (L - Dl) * e + Dh * n,
                       handle1 + Dl * e + Dh * n,
                       handle1 + Dl * e - Dh * n,
                       handle1 - (L - Dl) * e - Dh * n,])
    return points

# 判断前把手是否碰撞:所有点的位置信息,判断是否会发生碰撞
def crash(theta):
    for i, t in enumerate(theta[:2]):
        delta = 2 * arcsin(2.86 / (2 * alpha * t))

        # 寻找外圈的板凳
        neighbor = [j+i for j, k in enumerate(theta[i:-1]) if t + 2*pi
- delta < k and k < t + 2*pi + delta]        # 效率版本
        # neighbor = [j for j, k in enumerate(theta[:-1]) if t  < k
and k < t + 4*pi ]                                # 准确版本

        # 碰撞点
        if i == 0:
            point1 = vertices(theta[i], theta[i+1], 3.41)[1]
```

```
            point2 = vertices(theta[i], theta[i+1], 3.41)[0]
        else:
            point1 = vertices(theta[i], theta[i+1], 2.2)[1]
            point2 = vertices(theta[i], theta[i+1], 2.2)[0]

        # 判断是否碰撞
        for nei in neighbor:
            if Point_in_Polygon(point1, vertices(theta[nei], theta[nei
+ 1], 2.2)) or Point_in_Polygon(point2, vertices(theta[nei],
theta[nei + 1], 2.2)):
                print(f'第{i+1}块板凳与第{nei+1}块板凳发生碰撞')
                return True, (i, t, nei)

    print('未发生碰撞')
    return False, -1

# 给定时刻 t,计算龙头的位置
def head(t):
    theta = root(lambda x: x * sqrt(x**2 + 1) + log(x + sqrt(x**2 +
1)) - C + 2*v*t/alpha, 10).x[0]
    return theta * alpha * cos(theta), theta * alpha * sin(theta),
theta

# 给定龙头的位置,计算各把手的位置
def postition(theta):
    x_post, y_post, theta_post = [theta * alpha * cos(theta)], [theta
* alpha * sin(theta)], [theta]

    # 计算第一节龙身前把手
    def func(t1):
        t0, l = theta, 2.86
        return t1**2 + t0**2 - 2*t1*t0*cos(t1 - t0) - (l / alpha) ** 2

    t1 = root(func, theta+1).x[0]
    x_post.append(t1 * alpha * cos(t1)); y_post.append(t1 * alpha *
sin(t1)); theta_post.append(t1)

    # 计算后续的把手
    for _ in range(222):
        def func(t):
            l = 1.65
            return t**2 + t1**2 - 2*t*t1*cos(t - t1) - (l / alpha) **
2
        t1 = root(func, t1+1).x[0]
```

```
        x_post.append(t1 * alpha * cos(t1)); y_post.append(t1 * alpha
* sin(t1)); theta_post.append(t1)
    return x_post, y_post, theta_post

# 给定龙头的速度和各把手的位置,计算各把手的速度
def speed(x, y, theta):
    v1 = 1
    speed = [1]

    for i in range(223):
        d1 = np.array([cos(theta[i]) - theta[i]*sin(theta[i]),
sin(theta[i]) +
theta[i]*cos(theta[i])])                                    # 上一把手速
度方向
        d1 = d1 / np.linalg.norm(d1)
        d =  np.array([cos(theta[i+1]) - theta[i+1]*sin(theta[i+1]),
sin(theta[i+1]) + theta[i+1]*cos(theta[i+1])])                          #
这一把手速度方向
        d = d / np.linalg.norm(d)
        vector = np.array([x[i+1] - x[i], y[i+1] - y[i]])    # 板凳方向
        v1 = root(lambda v: np.dot(v1*d1, vector) - np.dot(v*d,
vector), v1).x[0]
        speed.append(v1)
    return speed

# 绘图检查
def draw(x, y, theta, t):
    the = np.linspace(0, 32*pi, 10000)
    x1, y1 = helix(the)

    fig = plt.figure(figsize=(9, 9))
    axes3 = fig.add_subplot(1, 1, 1)
    # 绘制板凳
    for i, tk in enumerate(theta[:-1]):
        if i == 0:
            l = 3.41
        else:
            l = 2.2
        p = plt.Polygon(xy=vertices(theta[i], theta[i+1], l),
color='#C82423', alpha=0.8)
        axes3.add_patch(p)

    plt.plot(x1, y1, linestyle='--', color='#2878B5', label='螺旋线')
    plt.scatter(x, y, s=5, c='k', label='把手')
```

```
    plt.legend(fontsize='large')


    plt.title(f'{round(t, 4)}s 时刻板凳龙位置图', fontsize=22)
    plt.xlabel('x', fontsize=16)
    plt.ylabel('y', fontsize=16)

    # # 坐标轴调整
    plt.tick_params(labelsize=13)
    plt.subplots_adjust(top=0.9, bottom=0.1, left=0.1, right=0.9)  #
调整页边距

    plt.show()
    plt.close()

# 主函数
if __name__ == '__main__':
    plt.rcParams['font.family'] = ['SimHei']      # 显示中文
    plt.rcParams['axes.unicode_minus'] = False    # 显示负号
    plt.axis('equal')                             # 等比例

    # 划分时间
    time = np.arange(412.47, 413, 0.0001)

    # 计算龙头前把手的位置和速度
    head_vec = np.vectorize(head)
    head_post_x, head_post_y, head_theta = head_vec(time)


    # 计算各时刻把手的位置和速度
    for i, t in enumerate(head_theta):
        # 计算把手位置
        handle_post_x, handle_post_y, handle_theta = postition(t)

        # 绘制图片
        draw(handle_post_x, handle_post_y, handle_theta, time[i])

        flag, mess = crash(handle_theta)

        if flag:
            break

    # 保存结果
```

```
    handle_speed = speed(handle_post_x, handle_post_y,
handle_theta)

    result = pd.DataFrame({'x':handle_post_x,
                           'y':handle_post_y,
                           'v':handle_speed})

    result.round(6).to_excel('问题二.xlsx')

    # 绘制结果
    the = np.linspace(0, 32*pi, 10000)
    x1, y1 = helix(the)

    fig = plt.figure(figsize=(9, 9))
    axes3 = fig.add_subplot(1, 1, 1)
    # 绘制板凳
    for j in [mess[0], mess[2]]:
        if j == 0:
            l = 3.41
        else:
            l = 2.2
        p = plt.Polygon(xy=vertices(handle_theta[j],
handle_theta[j+1], l), color='r', alpha=0.8)
        axes3.add_patch(p)

    plt.plot(x1, y1, linestyle='--', color='#2878B5', label='螺旋线')
    plt.plot(handle_post_x, handle_post_y, c='#C82423', label='板凳',
marker='o', markersize=0, linewidth=2)
    plt.scatter(handle_post_x, handle_post_y, s=20, c='#934B43',
label='把手')
    plt.legend(fontsize='large')

    plt.title(f'{round(time[i], 4)}s 时刻板凳龙位置图', fontsize=22)
    plt.xlabel('x', fontsize=16)
    plt.ylabel('y', fontsize=16)

    # # 坐标轴调整
    plt.tick_params(labelsize=13)
    plt.subplots_adjust(top=0.9, bottom=0.1, left=0.1, right=0.9)  #
调整页边距

    plt.savefig('碰撞结果.png', format='png', dpi=800)
    plt.show()
    plt.close()
```

第三问二分法计算最小螺距（Problem3_1.py）

```
# 二分法求解最小螺距
import numpy as np
import matplotlib.pyplot as plt
from numpy import pi, sin, cos, arcsin, sqrt, log
from scipy.optimize import root

plt.rcParams['font.family'] = ['SimHei']      # 显示中文
plt.rcParams['axes.unicode_minus'] = False    # 显示负号
plt.axis('equal')                             # 等比例

# 参数
v = 1                                          # 龙头行进速度(m/s)
C = (32 * pi) * sqrt((32*pi)**2 + 1) + log((32 * pi) +
sqrt((32*pi)**2 + 1))

# 上下界
dl = 0.4 ; dr = 0.5
while dr - dl > 0.0001:
    d = (dl + dr) / 2
    alpha = d / (2 * pi)

    # 给出螺旋线方程
    def helix(theta):
        return np.array([d/(2*pi) * theta * cos(theta), d/(2*pi) *
theta * sin(theta)])

    # 判断前把手是否碰撞:所有点的位置信息,判断是否会发生碰撞
    def crash(theta):
        for i, t in enumerate(theta[:2]):
            delta = 2 * arcsin(2.86 / (2 * alpha * t))

            # 寻找外圈的板凳
            neighbor = [j+i for j, k in enumerate(theta[i:-1]) if t +
2*pi - delta < k and k < t + 2*pi + delta]        # 效率版本
            # neighbor = [j for j, k in enumerate(theta[:-1]) if t  <
k and k < t + 4*pi ]                              # 准确版本

            # 碰撞点
            if i == 0:
                point1 = vertices(theta[i], theta[i+1], 3.41)[1]
                point2 = vertices(theta[i], theta[i+1], 3.41)[0]
            else:
```

```
                point1 = vertices(theta[i], theta[i+1], 2.2)[1]
                point2 = vertices(theta[i], theta[i+1], 2.2)[0]

            # 判断是否碰撞
            for nei in neighbor:
                if Point_in_Polygon(point1, vertices(theta[nei],
theta[nei + 1], 2.2)) or Point_in_Polygon(point2,
vertices(theta[nei], theta[nei + 1], 2.2)):
                    return True, (i, t, nei)
        return False, -1


    # 判断点在矩形内部
    def Point_in_Polygon(point, polygon):
        A, B, C, D = polygon[0], polygon[1], polygon[2], polygon[3]
        P = np.array([point[0], point[1]])

        flag1, flag2 = 0, 0

        AB, BC, CA = B-A, C-B, A-C
        AP, BP, CP = P-A, P-B, P-C
        if np.sign(AB[0] * AP[1] - AP[0] * AB[1]) == np.sign(BC[0] *
BP[1] - BP[0] * BC[1]) and  np.sign(BC[0] * BP[1] - BP[0] * BC[1]) ==
np.sign(CA[0] * CP[1] - CP[0] * CA[1]):
            flag1 = 1

        AD, DC, CA = D-A, C-D, A-C
        AP, DP, CP = P-A, P-D, P-C
        if np.sign(AD[0] * AP[1] - AP[0] * AD[1]) == np.sign(DC[0] *
DP[1] - DP[0] * DC[1]) and np.sign(DC[0] * DP[1] - DP[0] * DC[1]) ==
np.sign(CA[0] * CP[1] - CP[0] * CA[1]):
            flag2 = 1

        if flag2 or flag1:
            return True
        else:
            return False

    # 根据前把手计算四个点:
    def vertices(front, back, L):
        Dl = 0.275; Dh = 0.15
        handle1 = helix(front)
        handle2 = helix(back)
        e = (handle1 - handle2) / (L - 0.55)
```

```
        n = np.array([-e[1], e[0]])
        points = np.array([handle1 - (L - Dl) * e + Dh * n,
                           handle1 + Dl * e + Dh * n,
                           handle1 + Dl * e - Dh * n,
                           handle1 - (L - Dl) * e - Dh * n,])
        return points



    # 给定龙头的速度和各把手的位置,计算各把手的速度
    def speed(x, y, theta):
        v1 = 1
        speed = [1]

        for i in range(223):
            d1 = np.array([cos(theta[i]) - theta[i]*sin(theta[i]),
sin(theta[i]) +
theta[i]*cos(theta[i])])                                    # 上一把手速
度方向
            d1 = d1 / np.linalg.norm(d1)
            d =  np.array([cos(theta[i+1]) -
theta[i+1]*sin(theta[i+1]), sin(theta[i+1]) +
theta[i+1]*cos(theta[i+1])])                          # 这一把手速度方向
            d = d / np.linalg.norm(d)
            vector = np.array([x[i+1] - x[i], y[i+1] - y[i]])    # 板
凳方向
            v1 = root(lambda v: np.dot(v1*d1, vector) - np.dot(v*d,
vector), v1).x[0]
            speed.append(v1)
        return speed

    # 计算 theta
    def ode(t, y):
        return - v/alpha * (y**2 + 2) / (y**2 + 1) ** (3/2)

    # 给定时刻 t,计算龙头的位置
    def head(t):
        theta = root(lambda x: x * sqrt(x**2 + 1) + log(x + sqrt(x**2
+ 1)) - C + 2*v*t/alpha, 10).x[0]
        return theta * alpha * cos(theta), theta * alpha * sin(theta),
theta

    # 给定龙头的位置,计算各把手的位置
    def postition(theta):
```

```
        x_post, y_post, theta_post = [theta * alpha * cos(theta)],
[theta * alpha * sin(theta)], [theta]

        # 计算第一节龙身前把手
        def func(t1):
            t0, l = theta, 2.86
            return t1**2 + t0**2 - 2*t1*t0*cos(t1 - t0) - (l / alpha)
** 2

        t1 = root(func, theta+1).x[0]
        x_post.append(t1 * alpha * cos(t1)); y_post.append(t1 * alpha
* sin(t1)); theta_post.append(t1)

        # 计算后续的把手
        for _ in range(222):
            def func(t):
                l = 1.65
                return t**2 + t1**2 - 2*t*t1*cos(t - t1) - (l / alpha)
** 2
            t1 = root(func, t1+1).x[0]
            x_post.append(t1 * alpha * cos(t1)); y_post.append(t1 *
alpha * sin(t1)); theta_post.append(t1)
        return x_post, y_post, theta_post

    # 求进入圆弧的位置和时间
    in_theta = 4.5 / alpha
    in_time = (in_theta * sqrt(in_theta**2 + 1) + log(in_theta +
sqrt(in_theta**2 + 1)) - C) * alpha / 2 / v

    # 划分时间
    time = np.arange(in_time - 5, in_time, 0.01)

    # 计算龙头前把手的位置和速度
    head_vec = np.vectorize(head)
    head_post_x, head_post_y, head_theta = head_vec(time)

    # 判断是否发生碰撞
    for i, t in enumerate(head_theta):
        # 计算把手位置
        handle_post_x, handle_post_y, handle_theta = postition(t)

        flag, mess = crash(handle_theta)

        if flag:
```

```
            break

    if flag:
        dl = (dl + dr) / 2
        print(f'{d}, d 偏小')
    else:
        dr = (dl + dr) / 2
        print(f'{d}, d 偏大')

print(f'二分搜索完成,最小螺距为:{(dl+dr)/2}')
```

问题四精确计算问题四（Problem4_1.py）

```
# 问题四:计算位置和速度
import numpy as np
import pandas as pd
from numpy import pi, sin, cos, sqrt, arcsin, arctan, log
from scipy.optimize import root

# 参数
d = 1.7                                                    # 螺距(m)
v = 1                                                      # 龙头行进速度(m/s)
R = 4.5                                                    # 调整区域半径(m)

alpha = d / (2 * pi)

# 求入圆弧的位置和时间
in_theta = R/ alpha
C1 = in_theta * sqrt(in_theta**2 + 1) + log(in_theta +
sqrt(in_theta**2 + 1))
in_time = (in_theta * sqrt(in_theta**2 + 1) + log(in_theta +
sqrt(in_theta**2 + 1)) - C1) * alpha / 2 / v

# 求圆弧半径和圆心角
k = 2
x0, y0 = alpha * in_theta * cos(in_theta), alpha * in_theta *
sin(in_theta)
rn = np.array([cos(in_theta) - in_theta*sin(in_theta), sin(in_theta)
+ in_theta*cos(in_theta)])
rt = np.array([-rn[1], rn[0]])

l = root(lambda x: x0**2 - (k+1)*x*x0*rn[1] + y0**2 +
(k+1)*x*rn[0]*y0, -10).x[0]

g1 = k * np.linalg.norm(rt) * l
```

```
g2 = np.linalg.norm(rt) * l

w1 = v / g1
w2 = v / g2

beta = 2 * arcsin(sqrt(x0**2 + y0**2)/ (k+1) / g2)

t1 = arctan(- rn[0] / rn[1]) + pi
t2 = t1 - beta + pi

# 求出圆弧的位置和时间
out_theta = in_theta
out_time = beta * (1 / w1 + 1 / w2)
C2 = out_theta * sqrt(out_theta**2 + 1) + log(out_theta +
sqrt(out_theta**2 + 1)) - 2 * v * out_time / alpha

# 给定时间 t,计算时间 t 时的角度
def head(t):
    theta = root(lambda x: x * sqrt(x**2 + 1) + log(x + sqrt(x**2 +
1)) - C1 + 2*v*t/alpha, 10).x[0]
    return theta

def tail(t):
    theta = root(lambda x: x * sqrt(x**2 + 1) + log(x + sqrt(x**2 +
1)) - C2 - 2*v*t/alpha, 10).x[0]
    return theta

def post(t):
    if t <= 0:
        theta = head(t)
        return alpha * theta * cos(theta), alpha * theta * sin(theta),
theta, 0
    elif t <= beta / w1:
        x, y = x0 - 2*l*rn[1] + g1 * cos(t1 - t * w1), y0 + 2*l*rn[0]
+ g1 * sin(t1 - t * w1),
        return x, y, sqrt(x**2 + y**2)/alpha, 1
    elif t <= beta * (1/w1 + 1/w2):
        x, y = -x0 + l*rn[1] + g2*cos(t2 + (t - beta / w1) * w2), -y0
- l*rn[0] + g2*sin(t2 + (t - beta / w1) * w2)
        return x, y, sqrt(x**2 + y**2)/alpha, 2
    elif t > beta * (1/w1 + 1/w2):
        theta = tail(t)
        return alpha * theta * cos(theta + pi), alpha * theta *
sin(theta + pi), theta, 3
```

```
# 给定时间,计算把手位置
def handle_post(t):
    x1, y1, theta1, flag1 = post(t)
    x_lst, y_lst, theta_lst, flag_lst = [x1], [y1], [theta1], [flag1]

    # 计算第一节龙身前把手
    def func(t):
        l = 2.86
        x, y, theta, flag = post(t)
        return (x1 - x) ** 2 + (y1 - y) ** 2 - l ** 2

    t = root(func, t - 1).x[0]
    x1, y1, theta1, flag1 = post(t)
    x_lst.append(x1); y_lst.append(y1); theta_lst.append(theta1);
flag_lst.append(flag1)

    # 计算后续的把手
    for _ in range(222):
        def func(t):
            l = 1.65
            x, y, theta, flag = post(t)
            return (x1 - x) ** 2 + (y1 - y) ** 2 - l ** 2
        t = root(func, t - 1).x[0]
        x1, y1, theta1, flag1 = post(t)
        x_lst.append(x1); y_lst.append(y1); theta_lst.append(theta1);
flag_lst.append(flag1)
    return x_lst, y_lst, theta_lst, flag_lst

# 给定时间,求单位切线方向:
def direction(x, y, theta, flag):
    if flag == 0:
        rt = np.array([cos(theta) - theta*sin(theta), sin(theta) +
theta*cos(theta)])
        return -rt / np.linalg.norm(rt)
    elif flag == 1:
        rt = np.array([y - y0 - 2*l*rn[0], x0 - 2*l*rn[1] - x])
        return rt / np.linalg.norm(rt)
    elif flag == 2:
        rt = np.array([- y0 - l*rn[0] - y, x + x0 - l*rn[1]])
        return rt / np.linalg.norm(rt)
    else:
        rt = np.array([cos(theta + pi) - theta*sin(theta + pi),
sin(theta + pi) + theta*cos(theta + pi)])
```

```
        return rt / np.linalg.norm(rt)

# 给定龙头的速度和各把手的位置,计算各把手的速度
def speed(x, y, theta, flag):
    v1 = v
    speed = [v]

    for i in range(223):
        d1 = direction(x[i], y[i], theta[i],
flag[i])                    # 上一把手速度方向
        d =  direction(x[i+1], y[i+1], theta[i+1],
flag[i+1])                # 这一把手速度方向
        vector = np.array([x[i+1] - x[i], y[i+1] - y[i]])    # 板凳方向
        v1 = root(lambda v: v1*(np.dot(d1, vector)) - v*(np.dot(d,
vector)), v1).x[0]
        speed.append(v1)
    return speed

time = np.arange(-100, 101, 1)

post_result = pd.DataFrame(columns = time, index = range(224*2))
speed_result = pd.DataFrame(columns = time, index = range(224))


# 计算各时刻把手的位置和速度
for i in time:
    # 计算把手位置
    handle_x, handle_y, handle_theta, handle_flag = handle_post(i)

    handle_posts = []
    for j in range(224):
        handle_posts.append(handle_x[j]);
handle_posts.append(handle_y[j])
    post_result[i] = handle_posts

    # 计算把手速度
    handle_speed = speed(handle_x, handle_y, handle_theta,
handle_flag)
    speed_result[i] = handle_speed

post_result.round(6).to_excel('问题四_位置.xlsx')
    speed_result.round(6).to_excel('问题四_速度.xlsx')
```

问题四绘制调整路径示意图

```
# 绘制问题四调整路径图
import numpy as np
import matplotlib.pyplot as plt
from numpy import pi, sin, cos, arcsin, arctan, sqrt, log
from scipy.optimize import root

plt.rcParams['font.family'] = ['SimHei']      # 显示中文
plt.rcParams['axes.unicode_minus'] = False    # 显示负号
plt.axis('equal')                             # 等比例

# 参数
d = 1.7                                       # 螺距(m)
v = 1                                         # 龙头行进速度(m/s)
R = 4.5                                       # 调整区域半径(m)

alpha = d / (2 * pi)
k = 2

# 求入圆弧的位置和时间
in_theta = R / alpha
C1 = in_theta * sqrt(in_theta**2 + 1) + log(in_theta +
sqrt(in_theta**2 + 1))
in_time = (in_theta * sqrt(in_theta**2 + 1) + log(in_theta +
sqrt(in_theta**2 + 1)) - C1) * alpha / 2 / v

# 求圆弧半径和圆心角
k = 2
x0, y0 = alpha * in_theta * cos(in_theta), alpha * in_theta *
sin(in_theta)
rn = np.array([cos(in_theta) - in_theta*sin(in_theta), sin(in_theta)
+ in_theta*cos(in_theta)])
rt = np.array([-rn[1], rn[0]])

l = root(lambda x: x0**2 - (k+1)*x*x0*rn[1] + y0**2 +
(k+1)*x*rn[0]*y0, -10).x[0]

g1 = k * np.linalg.norm(rt) * l
g2 = np.linalg.norm(rt) * l

w1 = v / g1
w2 = v / g2
```

```
beta = 2 * arcsin(sqrt(x0**2 + y0**2)/ (k+1) / g2)

t1 = arctan(- rn[0] / rn[1]) + pi
t2 = t1 - beta + pi

# 求出圆弧的位置和时间
out_theta = in_theta
out_time = beta * (1 / w1 + 1 / w2)
C2 = out_theta * sqrt(out_theta**2 + 1) + log(out_theta +
sqrt(out_theta**2 + 1)) - 2 * v * out_time / alpha

# 给出螺旋线方程
def helix(theta):
    return d/(2*pi) * theta * cos(theta), d/(2*pi) * theta *
sin(theta)

plt.figure(figsize=(9, 9))
plt.legend(fontsize='large')

# 绘制盘入轨迹
t1 = np.linspace(in_theta, 24*pi, num=500)
x, y = helix(t1)
plt.plot(x, y, color='#F8AC8C', label='盘入轨迹', linewidth=2.5)

# 绘制圆弧
t1 = arctan(- rn[0] / rn[1]) + pi
t2 = t1 - beta + pi

t = np.linspace(0, g1 * beta )
x, y = x0 - k*l*rn[1] + g1 * cos(t1 - t/g1), y0 + k * l * rn[0] + g1
* sin(t1 - t/g1)
plt.plot(x, y, color='#05B9E2', label='前一段圆弧', linewidth=2.5)

t = np.linspace(g1 * beta, beta * (g1 + g2))
x, y = -x0 + l*rn[1] + g2*cos(t2 + (t - beta*g1)/g2), -y0 - l*rn[0] +
g2*sin(t2 + (t - beta*g1)/g2)
plt.plot(x, y, color='#F27970', label='后一段圆弧', linewidth=2.5)

# 绘制盘出轨迹
t = np.linspace(in_theta, 24*pi, num=500)
x, y = d/(2*pi) * t * cos(t + pi), d/(2*pi) * t * sin(t + pi)
plt.plot(x, y, color='#32B897', label='盘出路径', linewidth=2.5,
linestyle='--')
```

```
plt.title('调整路径示意图', fontsize=22)
plt.xlabel('x', fontsize=16)
plt.ylabel('y', fontsize=16)
plt.legend(fontsize='large')

# 坐标轴调整
plt.tick_params(labelsize=13)
plt.subplots_adjust(top=0.9, bottom=0.1, left=0.1, right=0.9)  # 调整
页边距

plt.savefig('调整路径示意图.png', format='png', dpi=800)
plt.show()
plt.close()
```

第五问二分法求最大速度（Problem5_1.py）

```
# 二分法求最大速度
import numpy as np
from numpy import pi, sin, cos, sqrt, arcsin, arctan, log
from scipy.optimize import root

# 参数
d = 1.7                                                    # 螺距(m)
R = 4.5                                                    # 调整区域半径(m)

alpha = d / (2 * pi)

ll, rr  =  1.24, 1.25
delta = 0.0001
while rr - ll > delta:
    mid = (rr + ll) / 2
    v = mid
    max_v = 0

    # 求入圆弧的位置和时间
    in_theta = R/ alpha
    C1 = in_theta * sqrt(in_theta**2 + 1) + log(in_theta +
sqrt(in_theta**2 + 1))
    in_time = (in_theta * sqrt(in_theta**2 + 1) + log(in_theta +
sqrt(in_theta**2 + 1)) - C1) * alpha / 2 / v

    # 求圆弧半径和圆心角
    k = 2
    x0, y0 = alpha * in_theta * cos(in_theta), alpha * in_theta *
sin(in_theta)
```

```
    rn = np.array([cos(in_theta) - in_theta*sin(in_theta),
sin(in_theta) + in_theta*cos(in_theta)])
    rt = np.array([-rn[1], rn[0]])

    l = root(lambda x: x0**2 - (k+1)*x*x0*rn[1] + y0**2 +
(k+1)*x*rn[0]*y0, -10).x[0]

    g1 = k * np.linalg.norm(rt) * l
    g2 = np.linalg.norm(rt) * l

    w1 = v / g1
    w2 = v / g2

    beta = 2 * arcsin(sqrt(x0**2 + y0**2)/ (k+1) / g2)

    t1 = arctan(- rn[0] / rn[1]) + pi
    t2 = t1 - beta + pi

    # 求出圆弧的位置和时间
    out_theta = in_theta
    out_time = beta * (1 / w1 + 1 / w2)
    C2 = out_theta * sqrt(out_theta**2 + 1) + log(out_theta +
sqrt(out_theta**2 + 1)) - 2 * v * out_time / alpha

    # 给定时间 t,计算时间 t 时的角度
    def head(t):
        theta = root(lambda x: x * sqrt(x**2 + 1) + log(x + sqrt(x**2
+ 1)) - C1 + 2*v*t/alpha, 10).x[0]
        return theta

    def tail(t):
        theta = root(lambda x: x * sqrt(x**2 + 1) + log(x + sqrt(x**2
+ 1)) - C2 - 2*v*t/alpha, 10).x[0]
        return theta

    def post(t):
        if t <= 0:
            theta = head(t)
            return alpha * theta * cos(theta), alpha * theta *
sin(theta), theta, 0
        elif t <= beta / w1:
            x, y = x0 - 2*l*rn[1] + g1 * cos(t1 - t * w1), y0 +
2*l*rn[0] + g1 * sin(t1 - t * w1),
            return x, y, sqrt(x**2 + y**2)/alpha, 1
```

```
        elif t <= beta * (1/w1 + 1/w2):
            x, y = -x0 + l*rn[1] + g2*cos(t2 + (t - beta / w1) * w2),
-y0 - l*rn[0] + g2*sin(t2 + (t - beta / w1) * w2)
            return x, y, sqrt(x**2 + y**2)/alpha, 2
        elif t > beta * (1/w1 + 1/w2):
            theta = tail(t)
            return alpha * theta * cos(theta + pi), alpha * theta *
sin(theta + pi), theta, 3

    # 给定时间,计算把手位置
    def handle_post(t):
        x1, y1, theta1, flag1 = post(t)
        x_lst, y_lst, theta_lst, flag_lst = [x1], [y1], [theta1],
[flag1]

        # 计算第一节龙身前把手
        def func(t):
            l = 2.86
            x, y, theta, flag = post(t)
            return (x1 - x) ** 2 + (y1 - y) ** 2 - l ** 2

        t = root(func, t - 1).x[0]
        x1, y1, theta1, flag1 = post(t)
        x_lst.append(x1); y_lst.append(y1); theta_lst.append(theta1);
flag_lst.append(flag1)

        # 计算后续的把手
        for _ in range(222):
            def func(t):
                l = 1.65
                x, y, theta, flag = post(t)
                return (x1 - x) ** 2 + (y1 - y) ** 2 - l ** 2
            t = root(func, t - 1).x[0]
            x1, y1, theta1, flag1 = post(t)
            x_lst.append(x1); y_lst.append(y1);
theta_lst.append(theta1); flag_lst.append(flag1)
        return x_lst, y_lst, theta_lst, flag_lst

    # 给定时间,求单位切线方向:
    def direction(x, y, theta, flag):
        if flag == 0:
            rt = np.array([cos(theta) - theta*sin(theta), sin(theta) +
theta*cos(theta)])
            return -rt / np.linalg.norm(rt)
```

```
        elif flag == 1:
            rt = np.array([y - y0 - 2*l*rn[0], x0 - 2*l*rn[1] - x])
            return rt / np.linalg.norm(rt)
        elif flag == 2:
            rt = np.array([- y0 - l*rn[0] - y, x + x0 - l*rn[1]])
            return rt / np.linalg.norm(rt)
        else:
            rt = np.array([cos(theta + pi) - theta*sin(theta + pi),
sin(theta + pi) + theta*cos(theta + pi)])
            return rt / np.linalg.norm(rt)

    # 给定龙头的速度和各把手的位置,计算各把手的速度
    def speed(x, y, theta, flag):
        v1 = v
        speed = [v]

        for i in range(223):
            d1 = direction(x[i], y[i], theta[i],
flag[i])                    # 上一把手速度方向
            d =  direction(x[i+1], y[i+1], theta[i+1],
flag[i+1])                # 这一把手速度方向
            vector = np.array([x[i+1] - x[i], y[i+1] - y[i]])    # 板
凳方向
            v1 = root(lambda v: v1*(np.dot(d1, vector)) - v*(np.dot(d,
vector)), v1).x[0]
            speed.append(v1)
        return speed

    s_time = (g1 + g2) * beta / v

    time = np.arange(s_time - 0.5, s_time + 2.5, 0.0001)

    # 计算各时刻把手的位置和速度
    for i in time:
        # 计算把手位置
        handle_x, handle_y, handle_theta, handle_flag = handle_post(i)

        # 计算把手速度
        handle_speed = speed(handle_x, handle_y, handle_theta,
handle_flag)
        max_v = max(max_v, max(handle_speed))

    if max_v <= 2:
        print(f'{v}, {max_v}, 速度过小, {rr-ll}')
```

```
        ll = mid
    else:
        print(f"{v}, {max_v}, 速度过大, {rr-ll}")
        rr = mid

print(f'二分搜索完毕,最大速度为{(ll+rr)/2}')
```

第五问绘制最大速度随速度变化分布图（Plot5_2.py）

```
# 绘制最大速度随速度的变化情况
import numpy as np
import matplotlib.pyplot as plt
from numpy import pi, sin, cos, sqrt, arcsin, arctan, log
from scipy.optimize import root
from scipy.integrate import solve_ivp

plt.rcParams['font.family'] = ['SimHei']      # 显示中文
plt.rcParams['axes.unicode_minus'] = False    # 显示负号
plt.axis('equal')                             # 等比例

# 参数
d = 1.7                                       # 螺距(m)
R = 4.5                                       # 调整区域半径(m)

alpha = d / (2 * pi)

result = []
plt.figure(figsize=(9, 6))

for v in np.linspace(1, 2, 2):
    # 求入圆弧的位置和时间
    in_theta = R/ alpha
    C1 = in_theta * sqrt(in_theta**2 + 1) + log(in_theta +
sqrt(in_theta**2 + 1))
    in_time = (in_theta * sqrt(in_theta**2 + 1) + log(in_theta +
sqrt(in_theta**2 + 1)) - C1) * alpha / 2 / v

    # 求圆弧半径和圆心角
    k = 2
    x0, y0 = alpha * in_theta * cos(in_theta), alpha * in_theta *
sin(in_theta)
    rn = np.array([cos(in_theta) - in_theta*sin(in_theta),
sin(in_theta) + in_theta*cos(in_theta)])
    rt = np.array([-rn[1], rn[0]])
```

```
    l = root(lambda x: x0**2 - (k+1)*x*x0*rn[1] + y0**2 +
(k+1)*x*rn[0]*y0, -10).x[0]

    g1 = k * np.linalg.norm(rt) * l
    g2 = np.linalg.norm(rt) * l

    w1 = v / g1
    w2 = v / g2

    beta = 2 * arcsin(sqrt(x0**2 + y0**2)/ (k+1) / g2)

    t1 = arctan(- rn[0] / rn[1]) + pi
    t2 = t1 - beta + pi

    # 求出圆弧的位置和时间
    out_theta = in_theta
    out_time = beta * (1 / w1 + 1 / w2)
    C2 = out_theta * sqrt(out_theta**2 + 1) + log(out_theta +
sqrt(out_theta**2 + 1)) - 2 * v * out_time / alpha

    # 给定时间 t,计算时间 t 时的角度
    def head(t):
        theta = root(lambda x: x * sqrt(x**2 + 1) + log(x + sqrt(x**2
+ 1)) - C1 + 2*v*t/alpha, 10).x[0]
        return theta

    def tail(t):
        theta = root(lambda x: x * sqrt(x**2 + 1) + log(x + sqrt(x**2
+ 1)) - C2 - 2*v*t/alpha, 10).x[0]
        return theta

    def post(t):
        if t <= 0:
            theta = head(t)
            return alpha * theta * cos(theta), alpha * theta *
sin(theta), theta, 0
        elif t <= beta / w1:
            x, y = x0 - 2*l*rn[1] + g1 * cos(t1 - t * w1), y0 +
2*l*rn[0] + g1 * sin(t1 - t * w1),
            return x, y, sqrt(x**2 + y**2)/alpha, 1
        elif t <= beta * (1/w1 + 1/w2):
            x, y = -x0 + l*rn[1] + g2*cos(t2 + (t - beta / w1) * w2),
-y0 - l*rn[0] + g2*sin(t2 + (t - beta / w1) * w2)
```

```
            return x, y, sqrt(x**2 + y**2)/alpha, 2
        elif t > beta * (1/w1 + 1/w2):
            theta = tail(t)
            return alpha * theta * cos(theta + pi), alpha * theta *
sin(theta + pi), theta, 3

    # 给定时间,计算把手位置
    def handle_post(t):
        x1, y1, theta1, flag1 = post(t)
        x_lst, y_lst, theta_lst, flag_lst = [x1], [y1], [theta1],
[flag1]

        # 计算第一节龙身前把手
        def func(t):
            l = 2.86
            x, y, theta, flag = post(t)
            return (x1 - x) ** 2 + (y1 - y) ** 2 - l ** 2

        t = root(func, t - 1).x[0]
        x1, y1, theta1, flag1 = post(t)
        x_lst.append(x1); y_lst.append(y1); theta_lst.append(theta1);
flag_lst.append(flag1)

        # 计算后续的把手
        for _ in range(222):
            def func(t):
                l = 1.65
                x, y, theta, flag = post(t)
                return (x1 - x) ** 2 + (y1 - y) ** 2 - l ** 2
            t = root(func, t - 1).x[0]
            x1, y1, theta1, flag1 = post(t)
            x_lst.append(x1); y_lst.append(y1);
theta_lst.append(theta1); flag_lst.append(flag1)
        return x_lst, y_lst, theta_lst, flag_lst

    # 给定时间,求单位切线方向:
    def direction(x, y, theta, flag):
        if flag == 0:
            rt = np.array([cos(theta) - theta*sin(theta), sin(theta) +
theta*cos(theta)])
            return -rt / np.linalg.norm(rt)
        elif flag == 1:
            rt = np.array([y - y0 - 2*l*rn[0], x0 - 2*l*rn[1] - x])
            return rt / np.linalg.norm(rt)
```

```
        elif flag == 2:
            rt = np.array([- y0 - l*rn[0] - y, x + x0 - l*rn[1]])
            return rt / np.linalg.norm(rt)
        else:
            rt = np.array([cos(theta + pi) - theta*sin(theta + pi),
sin(theta + pi) + theta*cos(theta + pi)])
            return rt / np.linalg.norm(rt)

    # 给定龙头的速度和各把手的位置,计算各把手的速度
    def speed(x, y, theta, flag):
        v1 = v
        speed = [v]

        for i in range(223):
            d1 = direction(x[i], y[i], theta[i],
flag[i])                        # 上一把手速度方向
            d =  direction(x[i+1], y[i+1], theta[i+1],
flag[i+1])                  # 这一把手速度方向
            vector = np.array([x[i+1] - x[i], y[i+1] - y[i]])    # 板
凳方向
            v1 = root(lambda v: v1*(np.dot(d1, vector)) - v*(np.dot(d,
vector)), v1).x[0]
            speed.append(v1)
        return speed

    s_time = (g1 + g2) * beta / v

    # 计算龙头的位置
    t = np.arange(s_time - 10, s_time + 10, 0.1)

    max_list = []

    # 计算各时刻把手的位置和速度
    for i in t:
        # 计算把手位置
        handle_x, handle_y, handle_theta, handle_flag = handle_post(i)

        # 计算把手速度
        handle_speed = speed(handle_x, handle_y, handle_theta,
handle_flag)
        max_list.append(max(handle_speed))


    result.append(max_list)
```

```
    plt.plot(max_list, label=f'{v:.2f} m/s')


plt.legend(fontsize='large')

plt.title('行进速度对最大速度位置的影响', fontsize=22)
plt.xlabel('t', fontsize=16)
plt.ylabel(r'$V_{max}$', fontsize=16)

# 坐标轴调整
plt.tick_params(labelsize=13)
plt.grid(linestyle='-', linewidth=0.7, color='black', alpha=0.3)
plt.subplots_adjust(top=0.9, bottom=0.1, left=0.1, right=0.9)  # 调整
页边距

# plt.savefig('行进速度对最大速度位置的影响.png', format='png', dpi=800)

plt.show()
plt.close()
```


中国大学生在线 dxs.moe.gov.cn