基于等距螺线的板凳龙路径优化

摘要

“板凳龙”作为浙闽地区的传统地方民俗文化，是一项非常珍贵的国家级非遗活动。因此，通过数学建模对盘龙时螺线的螺距和行进速度等参数进行优化设计，使得盘龙时的观赏性达到最大具有重大的现实意义。

对于问题一，首先引入极坐标系，建立**等距螺线运动方程**，再通过螺线的弧微分方程确定龙头把手在螺旋线上一定时间内经过的弧长，由于该微分方程不存在解析解，因此利用**数值积分**可以得出龙把手的位置，由**等时间步下递推方程**递推求解每个时刻每个把手的位置。最后利用差分近似积分计算得到每个时刻每个把手的速度。结果见表 1、表 2 及 result1.xlsx。

对于问题二，首先建立**状态延续方程**证明龙身相邻的板凳状态是可延续的，从而将碰撞检测简化为只考虑龙头或第一节龙身是否会与其他板凳发生碰撞。然后利用几何计算得到板凳边界点，采用**叉积法**进行板凳间碰撞检验。运动过程中依旧沿用问题一当中的等距螺线运动方程和计算方法，由此得出“板凳龙”盘入的终止时刻为 **412.473837s**，其余结果见表 3 及 result2.xlsx。

对于问题三，满足“板凳龙”各板凳间不发生碰撞地盘入调头空间进行掉头，考虑碰撞约束和调头空间约束建立起螺距最小化优化模型，利用二**分时间搜索**和**等时间步下递推方程**求解得到最小螺距为 **0.450273m**。同时在进一步的误差分析中，通过多次调整搜索精度，有效避免了碰撞发生的时间区间的离散性。

对于问题四，本问在考虑相切约束、碰撞约束、调头空间约束的情况下建立最优化调头圆弧路径长度的优化模型，使得龙头能够在调头空间内进行 S 形的调头。在实际的生活当中盘龙在舞动的过程当中必定不可能有后退的情况产生，因此优化模型还需要考虑加入后退约束。通过**变步长搜索算法**和时间步递推方程求解该优化模型，可得到最优圆弧长度为 **11.564990m**，其余结果见表 4 及 result4.xlsx。

对于问题五，龙头保持匀速运动，且后续龙身的速度不会超过 2m/s。对于该问题，依旧利用二分搜索算法龙头速度去求得最终的龙头最大速率为 **1.631786m/s**。

关键字：等距螺线　等时间步下递推方程　状态延续方程　变步长搜索算法

一、 问题重述

1.1 问题背景

“板凳龙”，又称“盘龙”，是一种传统民俗表演形式，由 223 节板凳首尾相连，形成蜿蜒的龙形，象征着力量、团结和吉祥。龙头板凳长 341 厘米，宽 30 厘米，装饰精美，位于最前方引导龙体；221 节龙身板凳长 220 厘米，宽 30 厘米，设计简洁统一，确保整体美观。最后一节为龙尾，象征威严。

每节板凳设有两个孔，用于连接相邻板凳，孔径 5.5 厘米，孔中心距板头 27.5 厘米，通过特殊把手连接，确保稳固与灵活性。板凳龙的表演需表演者高度配合，以展示盘龙的生动与灵动，传承和创新传统文化。

1.2 问题要求

问题 1：舞龙队沿螺距为 55 厘米的等距螺线顺时针前进，龙头初始位于螺线第 16 圈的 A 点，速度为 1 米每秒。记录并计算从初始时刻到 300 秒内，每秒钟龙队各把手的位置和速度，结果保存到文件 result1.xlsx 中。同时，在论文中给出龙头、部分龙身前把手和龙尾后把手在不同时刻的位置和速度。

问题 2：基于问题 1 的螺线，确定舞龙队盘入的终止时刻，以避免板凳直线发生碰撞，并记录此时龙队的位置和速度，保存结果到 result2.xlsx 中。在论文中描述龙头、部分龙身前把手和龙尾后把手在不同时刻的位置和速度。

问题 3：舞龙队在螺线盘入时，需要一个以螺线中心为圆心、直径为 9 米的圆形调头空间。确定盘入螺线的最小螺距，使龙头前把手能够盘入调头空间边界。

问题 4：将螺线螺距改为 1.7 米，调头空间仍为直径 9 米的圆形区域，舞龙队通过由两段圆弧组成的 S 型曲线调头。曲线前半段的半径是后半段的两倍，并且曲线与螺线相切。计算并记录从调头时刻前后各 100 秒内，舞龙队每秒的位置和速度，保存结果到 result4.xlsx，并在论文中描述龙头、部分龙身前把手和龙尾后把手在不同时刻的位置和速度。

问题 5：基于问题 4 的调头路径，确定龙头最大速度，使每个把手的速度不超过 2 米/秒。

二、 问题分析

2.1 问题一分析

对于问题一，首先，我们需要引入极坐标模拟并解析一条由 223 节板凳组成的“板凳龙”沿着螺距为 0.55 米的等距螺旋线顺时针盘入的运动情况，龙头以 1 米每秒的速度起始于螺线第 16 圈的 A 点。这个问题的核心是记录和计算每秒钟各个把手的位置和速度。这涉及到了等距螺线运动的数值积分和时间步递推方程的运用，目的是得到从 0 秒到 300 秒的完整运动轨迹。

2.2 问题二分析

对于问题二，基于问题 1 的运动模型，此问题要求确定舞龙队的盘入螺线的终止时刻，以避免板凳之间发生碰撞。此时需要记录龙把手的各个位置和速度。并且我们需要证明出来龙身以及龙尾的整个运动状态是延续的，从而我们能够将问题简化成只有龙头和后面第一个龙身会产生碰撞。对于边界点之间的碰撞检验，我们利用叉积法去进行矩形的点的碰撞检验，从而得到停止的时间节点。

2.3 问题三分析

对于问题三，要求在考虑碰撞和调头空间的和约束之下建立最小螺距优化模型，确保最小螺距满足所有条件。以使舞龙能够进入以螺线中心为圆心，直径为 9 米的圆形区域，并完成调头，同时避免碰撞。在就求解时我们利用数值方法进行求解解决最小螺距问题，并考虑精度问题。最终利用二分时间搜索和等时间步下递推方程求解。

2.4 问题四分析

对于问题四，螺距改为 1.7 米，调头空间为直径 9 米的圆形区域。此时舞龙通过 S 型曲线调头，要求曲线前段半径是后段的两倍，并且与螺线相切。此问题要求计算并记录调头前后各 100 秒内的运动数据。在本问当中我们需要考虑相切约束、碰撞约束、调头空间约束、限制后退约束，建立起最小盘入，盘出曲线间距最优化模型。

2.5 问题五分析

对于问题五，此问题要求在问题 4 的调头路径下，确定龙头的最大速度，同时确保龙身各把手的速度不会超过 2 米/秒。通过二分速度搜索，找到龙头的最大速度，同时满足龙身速度的限制。

下面我们给出解决问题较为关键点的思维导图：

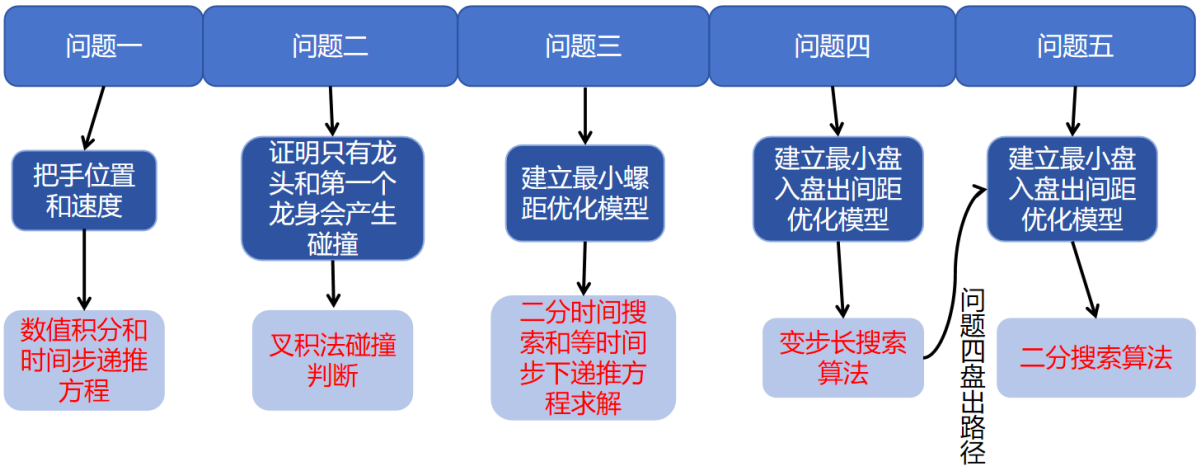


图 1　思路导图

三、 模型假设

为简化问题，本文做出以下假设：

- 假设 1 板凳龙在运动过程中板凳与把手间连接良好，在圆弧曲线上角度变化均匀。
- 假设 2 板凳龙中所有的板凳均为刚性的，板凳间的相对位置不变，板凳本身不产生形变。
- 假设 3 板凳龙的龙身运动速率连续变化，不存在中途停止或突然加速。

四、 符号说明

符号	说明
r	极坐标内极轴
θ	极坐标内极角
p	螺距
v_h	龙头前把手的速度
v_i	第 i 个龙身前把手的速度
L_h	龙头板凳的长度
L_i	第 i 个龙身前把手的长度
…	…

五、 问题一模型的建立和求解

5.1 阿基米德螺线运动方程建立

5.1.1 等距螺线方程

首先我们给出整个模型的示意图，我们可以知道板凳的两个把手都在螺线之上，LT 表示龙头所在的位置，A_n 表示后续的龙身或者龙尾的前后把手。

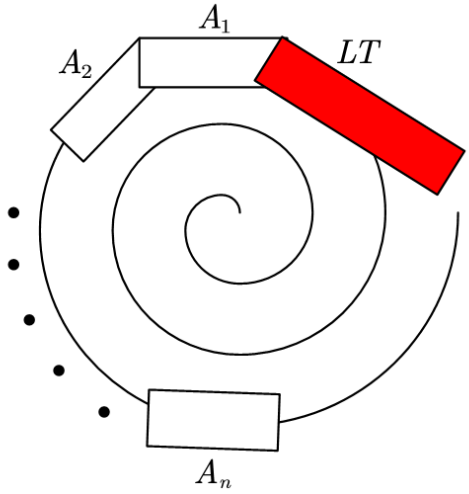


图 2　刚体螺线运动示意图

在题目给出的螺线图形当中我们能够得知其为等距螺线，即阿基米德螺线。我们考虑整体模型建立在阿基米德螺旋线的基础上，引入极坐标 (r, θ) 进行表示，对于“板凳龙”上的第 i 个龙身的前把手所处的位置，我们有：

$$\begin{cases} x_i = r_i cos\theta_i \\ y_i = r_i sin\theta_i \end{cases}$$

其中：

· θ_i 为第 i 个龙身前把手所处位置的极角，单位为弧度。

· r_i 为第 i 个龙身前把手所处位置的极径，单位为米。

对于龙头的前把手，它沿着等距螺线顺时针盘入，在极坐标中阿基米德等距螺旋线的方程可以描述为：

$$r(\theta) = r_0 + k\theta$$

其中：

· r_0 为初始时刻的螺线半径。在本题中，板凳龙从 A 点顺时针盘入至原心的运动轨迹可以等效看作从原心逆时针盘出，因此初始时刻的螺线半径为 0 米。

· p 是等距螺旋线的螺距，等距螺线的增长率 $k = \dfrac{p}{2\pi}$，即螺线每旋转一圈，半径变化 k 米。

5.1.2 弧微分方程

下面，我们需要求解把手在螺旋线上运动时经历时间 t 后转过的弧长，引入极坐标系中的弧微分方程：

$$ds = \sqrt{[r(\theta)]^2 + [r'(\theta)]^2}d\theta$$

又有在弧线上的运动微分可知：

$$ds = v \cdot dt$$

带入上式可得：

$$\frac{d\theta}{dt} = \frac{v}{\sqrt{[r(\theta)]^2 + [r'(\theta)]^2}}$$

该微分方程没有一个简单的原始函数形式，因此在模型求解时需要引入数值积分的方法进行计算，即可得出龙头前把手在经历时间 t 后转过的弧长。

由此，对于等距螺线中的龙头前把手，当其转过 θ 时，考虑板凳龙沿着等距螺线顺时针的盘入，我们有：

$$\int_{\theta}^{32\pi} \sqrt{[r(\theta)]^2 + [r'(\theta)]^2}d\theta = v_h t$$

5.2 模型求解

5.2.1 位置求解

对于求解龙头以及后续指定龙身的前后把手的极坐标位置，首先我们需要求解龙头前把手在时刻 t 所处的极坐标位置，再根据龙头的位置递推即可得到所需龙身前后把手的位置。

对于龙头的前把手经历时间 t 后运动的螺线弧长，可以表示为：

$$\begin{cases} s = v_h t \\ s = \int_t^3 2\pi ds = \int_0^t \sqrt{[r(\theta)]^2 + [r'(\theta)]^2}d\theta \end{cases}$$

由此联立解得龙头前把手在时刻 t 所处的 $\theta_{LT}(t)$，根据 $r = r_0 + k\theta$ 可以得到龙头前把手在时刻 t 所处的完整位置信息为：$(k\theta_{LT}(t), \theta_{LT}(t))$。

对于后续龙身前把手的极坐标位置 $(k\theta_i(t), \theta_i(t))$，我们可以给出等时间步下的递推方程：

$$\begin{cases} s = L_i \\ s = \sqrt{[\theta_{i+1}(t) - \theta_i(t)]^2 + [k\theta_{i+1}(t) - k\theta_i(t)]^2} \end{cases}$$

其中：

· L_i 为龙身板凳的长度，$i = \{LT, 1, 2, ..., 222\}$，$L_{LT}$ 表示龙头板凳的长度。

· 特别地，龙尾后把手的位置可以看做再接入一个龙身板凳后其前把手的位置。

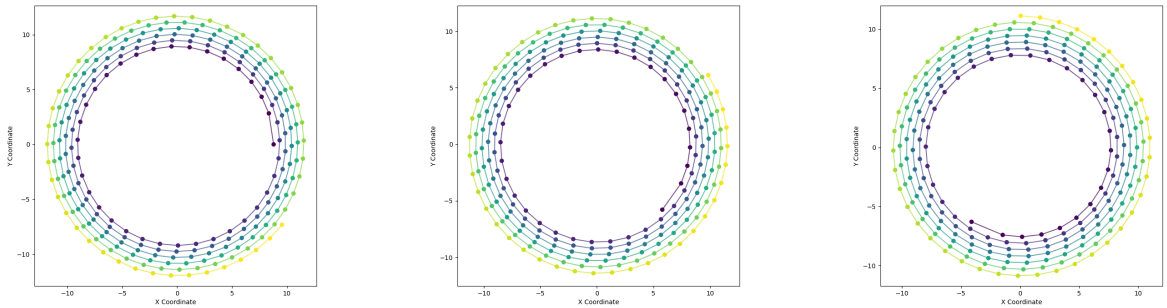
5.2.2 速度求解

对于龙头，我们可以知道其板凳前把手的速度恒定为 1m/s。而各节龙身板凳的前把手的速度可以根据得到的弧长微分方程由差分近似积分得到，定义 Δt 为龙身前把手在阿基米德螺旋线上运动的一小段时间，我们有：

$$v = \frac{ds}{dt} = \frac{\int_{\theta_i(t)}^{\theta_i(t+\Delta t)} \sqrt{[r(\theta)]^2 + [r'(\theta)]^2} d\theta}{\Delta t}$$

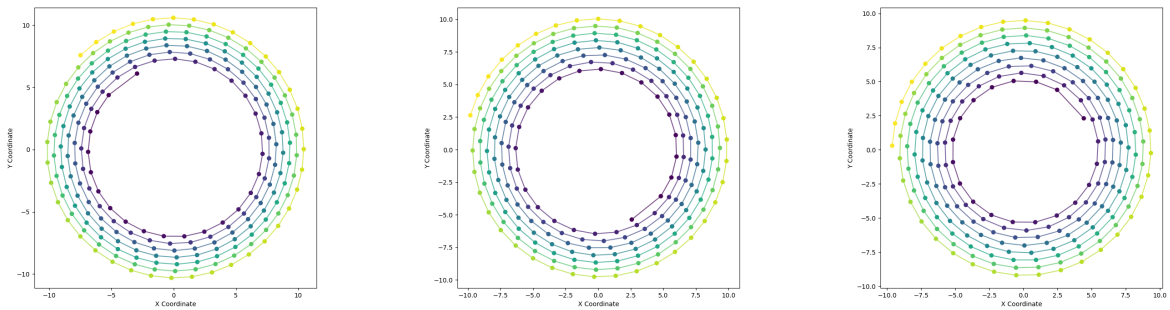
5.3 求解结果

我们可以得到在 0s、60s、120s、180s、240s、300s 整个盘龙位于的位置：



(a) 0 秒时盘龙的位置 **(b) 60 秒时盘龙的位置** **(c) 120 秒时盘龙的位置**

图 3　不同时间点盘龙的不同位置



(a) 180 秒时盘龙的位置 **(b) 240 秒时盘龙的位置** **(c) 300 秒时盘龙的位置**

图 4　不同时间点盘龙的不同位置

得出的结果下所示：

表 1　位置结果

	0s	60s	120s	180s	240s	300s
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)	1.828753	8.134544	9.727411	9.424751	8.399721	4.393013
第 201 节龙身 x(m)	4.555102	-6.619664	-10.627211	-9.287720	-7.457151	-7.458662
第 201 节龙身 y(m)	10.725118	9.025570	1.359847	-4.246673	-6.180726	-5.263384
龙尾（后）x(m)	-5.305444	7.364557	10.974348	7.383896	3.241051	1.785033
龙尾（后）y(m)	-10.676584	-8.797992	0.843473	7.492370	9.469336	9.301164

表 2　速度结果

	0s	60s	120s	180s	240s	300s
龙头 (m/s)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身 (m/s)	0.999971	0.999961	0.999945	0.999917	0.999859	0.999709
第 51 节龙身 (m/s)	0.999742	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	0.997817	0.996478

六、 问题二模型的建立和求解

6.1 模型建立

对于问题二，我们沿用问题一当中所建立的等距螺线模型，考虑板凳龙在运动过程中由于螺线曲率半径的减小，各板凳区块之间的距离不断缩小而产生碰撞，此时“板凳龙”需终止盘入。

6.1.1 碰撞检验证明

在问题二当中我们需要考虑碰撞的检测，由于龙头和后续龙身的规格是不同的，因此我们想到将龙头和龙身的碰撞检测分开来考虑。

首先考虑到“板凳龙”除龙头以外的把手都是在阿基米德等距螺线之上，且规格相同，因此我们首先考虑除掉龙头的情况，猜想所有板凳的状态是可以延续的，即下一个板凳会经历上一个板凳的状态。为了证明这一个猜想，首先我们利用极坐标表示前一个板凳的前后把手、后一个板凳的前后把手：

$$\begin{cases} x_i = r_i cos\theta_i, x_{i-1} = r_{i-1} cos\theta_{i-1} \\ y_i = r_i sin\theta_i, y_{i-1} = r_{i-1} sin\theta_{i-1} \end{cases}$$

由于阿基米德螺线为上面的 r 和 θ 在每一个状态都是唯一的，可得：

$$\begin{cases} r_i(\theta_i) = r_0 + k\theta_i \\ r_{i-1}(\theta_{i-1}) = r_0 + k\theta_{i-1} \end{cases}$$

又由于板凳的两个把手都在阿基米德螺线之上，因此我们在 t 时刻的板凳把手在螺线上的位置 (r_{i-1}, θ_{i-1})，必定会在后续的 t+n 时刻到达 (r_i, θ_i)，因此我们可以将这个状态延续方程表达成：

$$\begin{cases} x_{it} = x_{(i-1)(t+1)} \\ y_{it} = y_{(i-1)(t+1)} \end{cases}$$

由此我们可以证明得到龙身的相邻的板凳的状态是延续的，同时我们还需要证明龙身在不同圈数之上的相邻多个板凳之间是不可能发生碰撞的。下面我们给出不同圈数板凳的状态示意图：

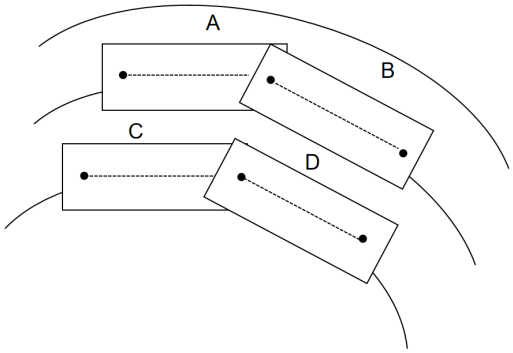


图 5　多圈板凳状态示意

我们分别给予 A、C 板凳一个极坐标：

$$\begin{cases} x_m = r_m cos\theta_m, x_n = r_n cos\theta_n \\ y_m = r_m sin\theta_m, y_n = r_n sin\theta_n \end{cases}$$

其中 m 表示 A 板凳 t 时刻的状态，n 表示 B 板凳在 t 时刻的状态。由于相邻的板凳之间我们已经可以得证状态是延续的，且整个盘龙的运动状态应该是一个整体，根据连续介质力学和状态延续方程不断的向后续的板凳进行递推，我们能够得到 t 时刻 m 状态的处于外圈的 A 板凳在经过 t_1 时间过后一定会处于 n 状态的处于内圈的 B 板凳所在的位置上，从而我们能够得证整个“板凳龙”的状态连续方程成立。

由于龙身状态延续方程的成立，因此我们在问题二当中的碰撞检测即可以只考虑龙头或者龙头后的第一节龙身会不会产生碰撞。现在我们先证明龙头后的第一节龙身是可能产生碰撞的，下面我们给出碰撞示意图：

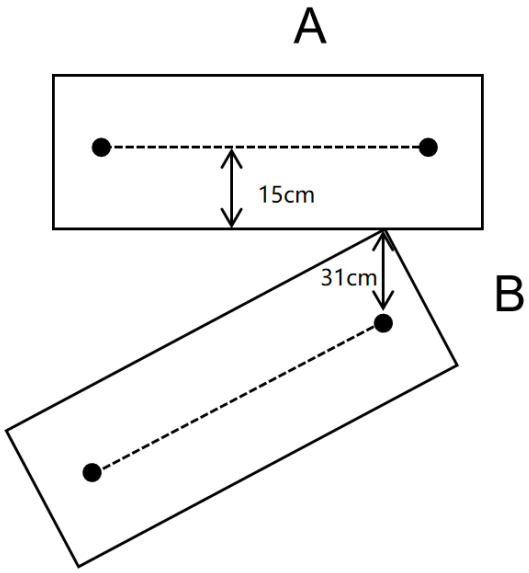


图 6　龙身碰撞示意图

由于龙身的两个把手都在阿基米德螺线之上，因此我们最小可以接受的距离为

0.55m，由于板凳 A 的两个把手之间的连线距离边界的距离为 0.15m，又因为板凳 B 的一个把手距离板凳的角的距离约为 0.31，求得距离之和为 0.56m，因此判断可得龙头后的第一个龙身确实会与其他的龙身产生碰撞。

对于龙头碰撞的判断而言，在龙头不断向内进行螺线运动的过程当中由于内部空间越来越小，龙头长度肯定会达到一个临界的碰撞点。由于龙身状态延续方程的成立，以及龙头在运动过程当中由于内部空间的缩小一定碰撞，因此我们在进行碰撞考虑的时候只需要龙头和龙头后面第一节的龙身是否碰撞即可。

6.1.2 板凳边界点推导

根据观察整个板凳的实际运动情况，我们能够得到产生碰撞的是龙头或是第一节龙身板凳的四个角和其他板凳产生碰撞，而在问题一当中我们得到的坐标都为板凳的两个把手在螺线上的坐标，因此我们需要去计算推理得到这两个把手所确定的板凳其边界四个点坐标。通过对于板凳的几何观察我们可以通过相似计算可以得到板凳的边界四个点坐标，下面给出一个板凳的推导示意图：

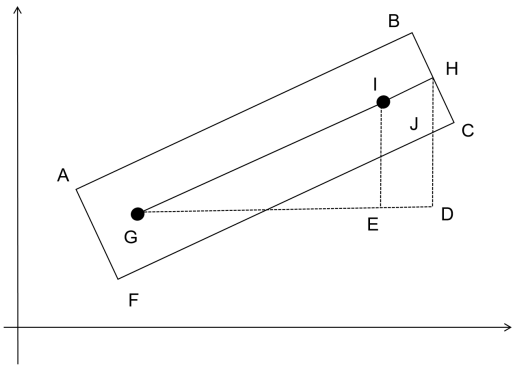


图 7　板凳边界点推导示意图

由于 $\triangle GIE \sim \triangle GHD \sim \triangle HJC$，因此我们能够得到

$$\frac{HG}{GD} = \frac{GI}{GE}$$

由于我们知道板凳的两个把手 G、I 的坐标 $(x_1, y_1), (x_2, y_2)$，因此我们能够得到 H 点的坐标 (x_3, y_3), 又由于我们能够得到：

$$tan\alpha = \frac{GD}{GH} = \frac{x_3 - x_1}{GH}$$

因此我们能够通过平面几何得到 C 点的坐标，同理我们也能够推得 A、B、F 的坐标。任意的龙头或者龙身、龙尾都是能够得到这样子的平面几何相似模型得到板凳的边界四个坐标。

6.1.3 碰撞区块叉积检验

根据得到的板凳区块边界的坐标，我们将龙头板凳或第 1 个龙身板凳与其余所有的龙身板凳进行叉积检验，以此来判断两板凳在运动过程中是否产生碰撞。

对于两个连续的把手 (x_i, y_i) 和 (x_{i+1}, y_{i+1})，我们可以根据上面给出它们对应的板凳边界的四个点的坐标，在此为方便起见我们直接将第 i 个龙身板凳把手和第 i+1 个龙身板凳把手确定的板凳边界的四个点分别表示为 $A_i(ma_i, na_i)$，$B_i(mb_i, nb_i)$，$C_i(mc_i, nc_i)$，$D_i(md_i, nd_i)$。

下面我们采用叉积法判断另一个板凳边界上的某个点是否落在一个板凳的边界内，有第 i 个龙身板凳把手和第 i+1 个龙身板凳把手确定的板凳边界的四个点 A_i，B_i，C_i，D_i 和第 k 个龙身板凳把手和第 k+1 个龙身板凳把手确定的板凳边界上的其中一个点 A_k，根据叉积给出碰撞区块检验：

$$\begin{cases} (\overrightarrow{A_iA_k} \times \overrightarrow{A_iB_i}) * (\overrightarrow{C_iA_k} \times \overrightarrow{C_iD_i}) \geq 0 \\ (\overrightarrow{B_iA_k} \times \overrightarrow{B_iC_i}) * (\overrightarrow{D_iA_k} \times \overrightarrow{D_iA_i}) \geq 0 \end{cases}$$

其中，对于第 k 个龙身板凳把手和第 k+1 个龙身板凳把手确定的板凳，需要将其四个点均带入上述叉积检验中，遍历整个运动过程即可说明这两个板凳区块在运动过程中的位置关系。

特别地，当两个板凳完全相交呈“X”型时，上述碰撞约束检验不出但实际两板凳已经产生碰撞。对于这种情况，由于板凳龙运动过程中板凳位置的变化具有连续性，因此在出现该情况前一定存在能被叉积检验出的时刻，上述约束依旧成立。

6.2 模型求解

通过考虑板凳的四个边界碰撞点才会产生碰撞，并且通过碰撞检验证明我们考虑碰撞的情况只会发生在龙头、第一个龙身之上。碰撞的区块我们利用龙头板凳或第 1 个龙身板凳与其余所有的龙身板凳进行叉积检验，构建起考虑碰撞检验证明的数学模型进行求解。利用 python 代码进行求解，给出算法的流程：

Step 1: 初始化参数与几何模型

首先我们设定板凳龙的螺线运动模型为等距螺线运动模型，其参数为：

- 螺距参数 $a = \frac{0.55}{2\pi}$，决定螺线的螺旋程度。
- 运动的角度从 0 变化到 32π，即龙的头部围绕螺线旋转。

Step 2: 路径长度与角度计算

我们使用路径积分公式计算龙头在时间 t 时的路径长度 $s(t)$：

$$s = \int_{x_{\text{prev}}}^{x_{\text{now}}} \sqrt{r_0^2 + r_1^2}\, dx$$

其中，$r_0(\theta)$ 是半径关于角度 θ 的函数，r_1 是恒定值。

通过二分法搜索，我们找到给定时间 t 时的角度 $\theta(t)$，满足路径长度约束。

Step 3: 位置计算

对于给定的角度 θ，计算龙头的 x 和 y 坐标：

$$x(\theta) = r_0(\theta)\cos(\theta), \quad y(\theta) = r_0(\theta)\sin(\theta)$$

同理，龙身每个把手的位置通过递推计算。

Step 4: 速度计算

我们根据龙头的当前位置和前一个位置之间的路径长度比例，计算速度 v。速度的更新公式为：

$$v_{\text{next}} = v_{\text{prev}} \times \frac{s_{\text{next}}}{s_{\text{prev}}}$$

Step 5: 碰撞检测

为了确保板凳龙的各个把手在运动过程中不会碰撞，我们使用叉积法检测矩形是否发生重叠：

$$C(x_1, y_1, x_2, y_2, x, y) = (x_2 - x_1)(y - y_1) - (x - x_1)(y_2 - y_1)$$

如果 $C(x_1, y_1, x_2, y_2, x, y)$ 与 $C(x_3, y_3, x_4, y_4, x, y)$ 同号，则表示没有碰撞。

通过这个方法，我们检查在运动的每一时刻，板凳龙的矩形部分是否发生了碰撞，并确定发生碰撞的时间节点。

通过算法我们能够得到最终的舞龙队终止盘入的时间是 412.473837s 过后，我们将结果保存在附件 resul2.xlsx 中，下面我们给出在此时刻整个盘龙队伍的状态可视化：

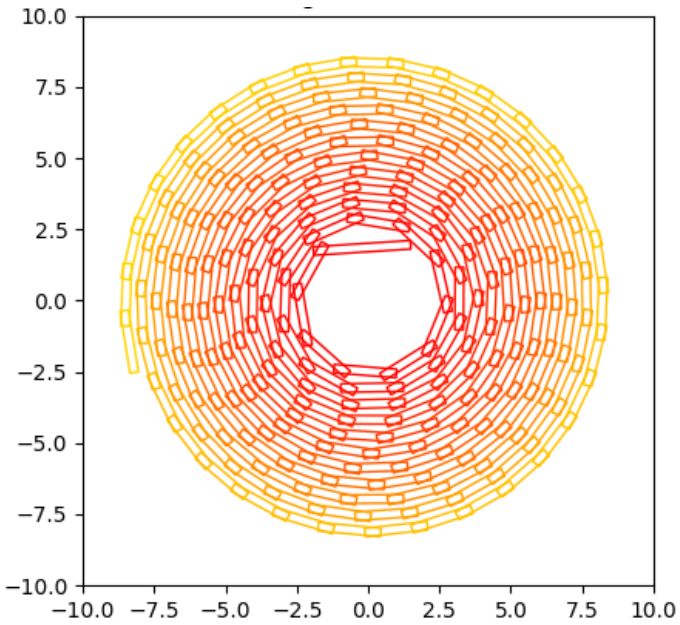


图 8　舞龙队停止时刻位置状态可视化

在此我们给出此时龙头前把手、龙头后面第 1、51、101、151、201 条龙身前把手和龙尾后把手的位置和速度表格。

表 3　碰撞时的位置和速度

	横坐标 x(m)	纵坐标 y(m)	速度 (m/s)
龙头	1.209931	1.942784	1.000000
第 1 节龙身	-1.643792	1.753399	0.991551
第 51 节龙身	1.281201	4.326588	0.976858
第 101 节龙身	-0.536246	-5.880138	0.974550
第 151 节龙身	0.968840	-6.957479	0.973608
第 201 节龙身	-7.893161	-1.230764	0.973096
龙尾（后）	0.956217	8.322736	0.972938

七、 问题三模型的建立和求解

7.1 模型建立

考虑板凳龙要能够在以螺线中心为圆心、直径为 9m 的圆形区域内调头，当板凳龙沿着相应的螺旋线盘入到调头空间前龙身间不能产生碰撞，且运动螺旋线的螺距要尽可能小，我们可以建立如下优化模型：

目标函数：

$$\min \quad p$$

约束条件：

（1）碰撞约束：我们需要满足在整个运动过程中，龙头跟龙身以及龙身间的各板凳不会产生碰撞，对此我们沿用问题二中的碰撞检验方式：

$$\begin{cases} (\overrightarrow{A_iA_k} \times \overrightarrow{A_iB_i}) * (\overrightarrow{C_iA_k} \times \overrightarrow{C_iD_i}) \geq 0 \\ (\overrightarrow{B_iA_k} \times \overrightarrow{B_iC_i}) * (\overrightarrow{D_iA_k} \times \overrightarrow{D_iA_i}) \geq 0 \end{cases}$$

（2）调头空间约束：我们需要满足该板凳龙最终能够沿螺线盘入调头空间，在板凳龙龙头前把手的运动过程中存在有时刻 t_0，其坐标 $(k\theta_{LT}(t), \theta_{LT}(t))$ 需满足：

$$[k\theta_{LT}(t_0)]^2 + [\theta_{LT}(t_0)]^2 \leq 4.5^2$$

综上所述，我们可以得出优化模型为：

$$\min p$$

$$\text{s.t.} \begin{cases} \begin{cases} (\overrightarrow{A_iA_k} \times \overrightarrow{A_iB_i}) * (\overrightarrow{C_iA_k} \times \overrightarrow{C_iD_i}) \geq 0 \\ (\overrightarrow{B_iA_k} \times \overrightarrow{B_iC_i}) * (\overrightarrow{D_iA_k} \times \overrightarrow{D_iA_i}) \geq 0 \end{cases} \\ [k\theta_{LT}(t_0)]^2 + [\theta_{LT}(t_0)]^2 \leq 4.5^2 \end{cases}$$

7.2 模型求解

下面我们给出代码的求解思维流程：

Step 1: 计算螺旋轨迹的半径、路径长度等基本几何信息。

Step 2: 计算路径长度为 t 时的对应角度; 计算在特定角度下螺旋的 x 和 y 坐标。

Step 3: 计算两个坐标点之间的欧几里得距离; 给定当前角度和路径长度，计算下一个角度。

Step 4: 根据前后的路径长度调整速度; 从时间 t 开始，计算螺旋轨迹的点集 pos 和速度 v。

Step 5: 计算点与线段的相对位置，用于判断点在线段哪一侧; 判断两个点集之间是否发生碰撞, 将四个点按逆时针排序，用于矩形碰撞检测。

Step 6: 判断在时间 t 时是否发生了碰撞; 计算并返回发生碰撞的最小时间 t。

Step 7: 通过二分法查找一个合适的 a，使得螺旋的最大半径不超过 4.5。最后在找到的 a 下调用 check 函数进行终极碰撞检测和可视化。

最终我们能够得到最小的螺距为 0.450278m

7.3 误差分析

在求解上述模型的过程中，我们考虑到在对龙头板凳或龙身板凳碰撞与否的检测时，我们是通过加权二分的方式将离散的时刻带入约束条件进行检验，此时可能存在一种情况为检验的时刻错过了碰撞的“时机”。对此，我们对第三问的求解结果进行误差分析：

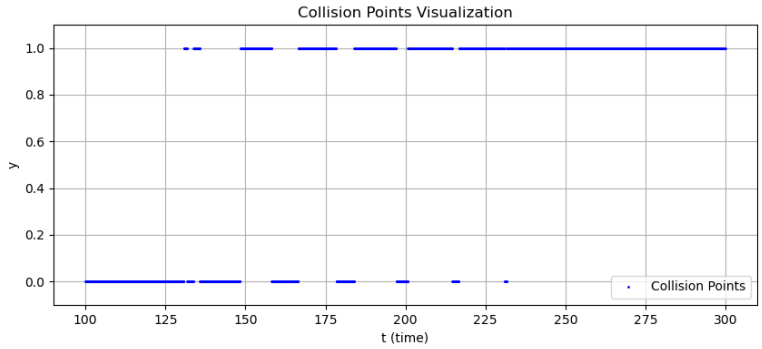


图 9　0.44m 螺距碰撞分析

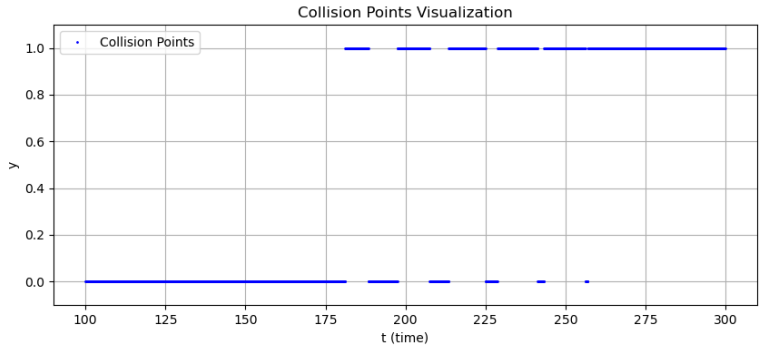


图 10　0.43m 螺距碰撞分析

上图中，我们取略小于得出的最小螺距的 0.44m 和 0.43m 螺距进行碰撞“时机”检测，纵坐标为 1 表示在该时刻发生碰撞。

通过观察两张碰撞分析图，我们可以发现碰撞发生的时间区间具有离散性，即在我们算法实现的过程中，对时间步长的二分可能会影响到我们不能确定地得出最早碰撞的时刻，因此我们需要多次更改时间步长进行对比更正。在本题中，我们分别采用了、、的精度进行计算，均得出最早碰撞的时刻对应的最小螺距为 0.45m，结果准确性较高。

八、 问题四模型的建立和求解

8.1 模型建立

在问题四当中，要求我们在问题三设立的调头空间之内进行调头，调头路径为两段圆弧相切的链接而成的 S 形曲线，且前一段的曲线半径是后一段曲线半径的 2 倍，这两段圆弧均与盘入、盘出曲线相切，要求我们的调头曲线尽可能的短，即要求我们盘入、盘出曲线之间的直线距离最短。设置盘入盘出曲线之间的直线距离为 s，我们基于此建立如下优化模型：

目标函数:

$$\min \quad s$$

约束条件:

(1) 相切约束：在问题当中我们需要满足在进行掉头的那一时刻所处的螺线的切线与整个调头的圆弧是相切的，即我们的切线和盘入盘出曲线之间的直线是垂直的，由于我们在螺线曲线之上能够得到盘入曲线和圆弧的相切点为 A(x_1, y_1), 中心对称的盘出曲线和圆弧的相切点为 C(x_3, y_3), 两个圆弧之间的交点为 B(x_2, y_2)，示意图如下:

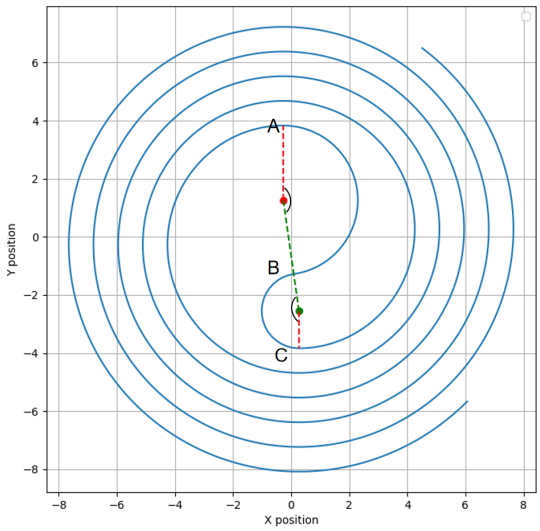


图 11　S 型调头曲线说明

由于这个 A、B、C 三个点都处于笛卡尔坐标系上，斜率 k 值可能得到 0 或者其他正常的数值，因此我们可以得到相切约束的的表达式为:

$$\begin{cases} -(y_1 - y_3) = \frac{x_1 - x_3}{k}, & k \neq 0 \\ x_1 = x_3, & k = 0 \end{cases}$$

(2) 碰撞约束：由于板凳的调头过程当中依旧不能够产生碰撞，因此我们沿用问题三中的碰撞约束。

(3) 调头空间约束：我们整个 S 型的调头曲线都应该是在问题所给定的调头空间之间，因此我们同样沿用问题三中的调头空间约束。

(4) 实际后退约束：在问题当中我们考虑了舞龙队在运动的过程当中肯定不能够产生向后的速度，因为在实际的舞龙过程当中肯定不能够产生整个舞龙在一个时间突然向后移动的情况。因此我们能够将后退约束写成:

$$V \geq 0$$

综上考虑相切约束、碰撞约束、调头空间约束、实际后退约束，我们能够得到优化模型为：

$$\min \quad s$$

$$\text{s.t.} \begin{cases} \begin{cases} -(y_1 - y_3) = \frac{x_1 - x_3}{k}, & k \neq 0 \\ x_1 = x_3, & k = 0 \end{cases} \\ \begin{cases} (\overrightarrow{A_iA_k} \times \overrightarrow{A_iB_i}) * (\overrightarrow{C_iA_k} \times \overrightarrow{C_iD_i}) \geq 0 \\ (\overrightarrow{B_iA_k} \times \overrightarrow{B_iC_i}) * (\overrightarrow{D_iA_k} \times \overrightarrow{D_iA_i}) \geq 0 \end{cases} \\ [k\theta_{LT}(t_0)]^2 + [\theta_{LT}(t_0)]^2 \leq 4.5^2 \\ V = \frac{ds}{dt} \geq 0 \end{cases}$$

8.2 模型求解

在问题当中我们约束只说明了 $V > 0$，因此我们需要考虑将 V 与其他的约束或变量产生联系。在整个运动过程当中我们能够假设两个圆弧的长度为 S_1, S_2, 假设盘出曲线的长度为 S，盘入曲线为

$$\int_{\theta_1}^{\theta_2} \sqrt{r_1(\theta)^2 + r_1^2}\, d\theta$$

因此我们能够得到整个在运动过程当中的方程为：

$$Vt = S_1 + S_2 + S + \int_{\theta_1}^{\theta_2} \sqrt{r_0(\theta)^2 + r_0^2}\, d\theta$$

其中 r_0 为盘入曲线方程

通过该方程我们能够求解得到盘入曲线与圆弧的交点 A 的 θ_2，即可得到盘入坐标，由于盘出坐标关于坐标原点中心对称，因此我们能够得到盘出坐标的 θ_3

通过上述的求解推到我们能够得到龙头的运动方程，后面所有把手轨迹方程与龙头把手的运动轨迹方程一致，从而能够用龙头的运动方程通过时间的平移来得到后某个位置的运动方程，通过约束两把手距离得到所有把手的 xy 坐标。

在得到把手 xy 坐标过后我们可以通过一个非常微小的常数 t 到上一个虚拟的坐标，从而能够利用差分近似微分得到把手的 V 速度，从而我们能够把禁止后退约束的 V 和整个运动过程建立联系。

下面我们给出在代码中的主要逻辑：核心目的是模拟螺线运动中的板凳龙，并解决板凳龙沿螺线运动过程中产生的碰撞检测、路径为 S 形调头曲线以及速度更新问题。

Step 1: 初始化与螺线函数的定义

首先，定义螺线运动的基本函数：

- 螺线的半径函数 $r_0(\theta) = a \cdot \theta$，其中 a 为螺旋参数。
- 确定龙头在不同角度下的 x 坐标和 y 坐标：$x(\theta)$ 和 $y(\theta)$。
- 路径长度计算函数 $s(x)$，用于计算螺线运动时的路径长度。

Step 2: 二分法搜索角度与路径

为了得到给定时间 t 对应的运动角度，代码通过二分法搜索路径长度：

$$\int_{\theta_{\text{pre}}}^{\theta_{\text{now}}} \sqrt{r_0(\theta)^2 + r_1^2}\, d\theta$$

这个积分表示从起点到当前角度的路径长度。

Step 3: 计算下一步角度与速度

根据当前角度和速度，代码通过二分法计算下一步角度，保证移动的距离不超过给定的长度约束。速度的计算通过路径长度比例进行更新：

$$v_{\text{next}} = v_{\text{prev}} \times \frac{s_{\text{next}}}{s_{\text{prev}}}$$

Step 4: 碰撞检测

对于板凳龙的每个把手，代码通过几何计算检测是否发生碰撞。碰撞检测的核心逻辑基于叉积方法来判断点是否在矩形内部：

$$C(x_1, y_1, x_2, y_2, x, y) = (x_2 - x_1)(y - y_1) - (x - x_1)(y_2 - y_1)$$

Step 5: 终止条件与碰撞时刻计算

通过不断缩小时间步长，代码确定发生碰撞的确切时刻。对于每个时刻，代码会计算对应的速度、位置，并检测碰撞。如果检测到碰撞，则记录碰撞时刻。

8.3 求解结果

通过代码求解我们能够得到最优的圆弧长度为 11.56499m，拐入圆弧调头的时间点为 1340.177307s。下面我们给出题目当中要求的不同时间节点不同板凳位置的（x，y）坐标和速度

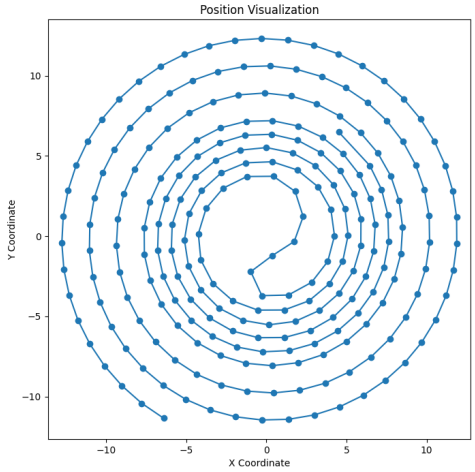
表 4　位置结果

	-100s	-50s	0s	50s	100s
龙头 x(m)	-0.271553	3.042200	-1.260353	5.956132	4.514075
龙头 y(m)	3.834525	-5.751635	3.705219	0.260636	6.496322
第 1 节龙身 x(m)	-2.932247	5.292138	-3.576062	5.259771	6.456619
第 1 节龙身 y(m)	2.785571	-3.986024	2.026798	-2.513292	4.397246
第 51 节龙身 x(m)	-6.376881	-9.163491	-5.780144	-5.508342	2.821624
第 51 节龙身 y(m)	-4.520915	-2.150083	-5.313189	1.988475	-2.883757
第 101 节龙身 x(m)	9.777702	2.917533	9.449541	1.198933	0.387404
第 101 节龙身 y(m)	3.191424	-11.169373	4.126900	8.807161	7.200041
第 151 节龙身 x(m)	2.646376	7.604215	1.674795	-10.738344	2.080373
第 151 节龙身 y(m)	11.977290	-10.959382	12.173340	-2.892929	-9.610081
第 201 节龙身 x(m)	11.879822	-8.744966	1.918007	2.444384	2.890784
第 201 节龙身 y(m)	-0.472973	-9.585060	13.855660	-14.717925	13.666204
龙尾（后）x(m)	-10.642093	8.718904	-9.949786	12.567824	-12.366105
龙尾（后）y(m)	-10.075610	12.892729	-10.784716	5.481698	2.843633

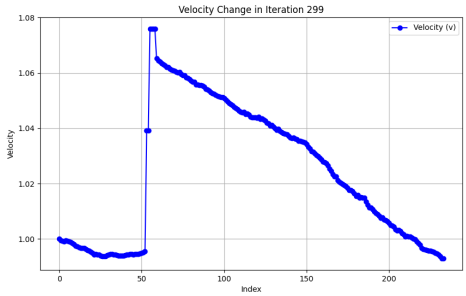
表 5　速度结果

	-100s	-50s	0s	50s	100s
龙头 (m/s)	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身 (m/s)	0.997198	0.999405	0.997521	1.000445	0.999986
第 51 节龙身 (m/s)	0.981276	0.981981	0.983815	0.960074	0.996560
第 101 节龙身 (m/s)	0.964904	0.966030	0.96520	0.947695	0.890233
第 151 节龙身 (m/s)	0.943430	0.946732	0.944281	0.931896	0.876105
第 201 节龙身 (m/s)	0.920483	0.922191	0.923814	0.907681	0.860959
龙尾（后）(m/s)	0.912535	0.913095	0.911225	0.900409	0.853656

下面我们给出“板凳龙”开始拐弯 100s 过后的图像，即已经开始盘出的图像和在开始拐弯 100s 后的时刻的整个“板凳龙”的速度可视化曲线



(a) 开始拐弯 100s 过后的图像



(b) 开始拐弯 100s 过后的速度可视化曲线

九、 问题五模型的建立和求解

9.1 模型建立

为确定“板凳龙”在问题四设定的调头曲线下运动时，满足龙身各把手速度始终不超过 2m/s 的情况下，给出最大的龙头把手速度。我们设第 i 个龙身把手的速度为 v_i，龙头把手的速度为 v_h，建立以下模型：

目标函数：

$$\max v_h$$

约束条件：

对于“板凳龙”上的各个把手，龙身上各把手的速度不能超过 2m/s，即

$$v_i \leq 2 \quad i = 1, 2, ..., 222$$

其他的约束都与问题四当中建立的相切约束、碰撞约束、调头空间约束、实际后退约束一致，因此我们能够得到问题五的模型

9.2 模型求解

在模型五中我们需要得到固定的龙头速度使得后面的整个“板凳龙”各把手速度均不超过 2m/s，我们在本文当中依旧利用变步长搜索编写 python 代码去进行求解，下面我们给出代码的流程，这段代码的核心是通过二分法变步长搜索来确定板凳龙的龙头最佳速度，同时确保龙身的速度不会超过指定的最大值。接下来我们通过分步骤来说明代码的逻辑。

首先，定义路径函数和角度函数，通过计算路径长度来确定龙头运动的角度与问题四当中都是一致的，下面给出核心的求解代码流程

Step 1: 龙头步长搜索

接下来，通过二分法逐步调整龙头的步长速度，确保速度合理：

- 初始步长设定为 $v = 1$，目标是找到使得速度 v 满足龙身的速度不会超过 2 m/s 的条件。
- 搜索区间设置为 $[1, 2]$，通过二分法来调整步长：

$$\text{mid} = \frac{l + r}{2}$$

每次搜索之后，检查龙头和龙身的最大速度是否超过阈值。

Step 2: 检查龙身速度

为了确保龙身的速度不会超过 2 m/s，在每次计算龙头的路径后，通过递推方式计算每个把手的速度：

- 通过函数 $solve4(t, t_0)$ 计算每个时间点龙头和龙身的坐标位置和速度，逐步更新：

$$v_{\text{next}} = \frac{\text{路径长度}_{\text{next}}}{\text{路径长度}_{\text{prev}}} \times v_{\text{prev}}$$

- 如果任意一个把手的速度超过 2 m/s，则调整龙头的速度继续搜索。

Step 3: 终止条件与优化结果

二分搜索的终止条件为搜索精度达到设定的阈值 ‘eps2‘，即满足：

$$r - l \leq \epsilon_2$$

搜索结束时，得到龙头的最佳速度，并计算出龙身所有把手的最终速度和位置。通过算法的求解我们能够得到龙头的速度为 1.631786m/s。

十、 模型的评价

10.1 模型的优点

• 优点 1 采用了严谨的数学证明说明了碰撞情况的检验，在求解过程中极大优化了搜索的时间复杂度。

• 优点 2 问题四五采用了加权二分法进行搜索，可以更快地定位到目标值，从而提升搜索效率。

• 优点 3 在结果的误差分析中考虑到碰撞发生时间区间的离散性，并通过多次更改时间步长精度进行计算确保结果具有较高的准确性。

10.2 模型的缺点

- 缺点 1 求解过程多采用暴力枚举的方法，时间复杂度较高。
- 缺点 2 问题求解的结果精度依赖于算法枚举的步长，在有限时间内无法得出非常精确的结果。

参考文献

[1] 司守奎, 孙玺菁. 数学建模算法与应用[M]. 北京: 国防工业出版社, 2011.

[2] 李诗琬. 咸丰地区传统舞蹈“板凳龙”传承及动作实践研究[D]. [出版地不详]: 武汉音乐学院, 2023.

[3] 史明娜, 孙亮亮. 文化生态学视野下村落传统体育文化研究——以四川省非物质文化遗产安仁板凳龙为例[J]. 重庆交通大学学报 (社会科学版), 2015, 15(02):105-107.

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

附录 A　文件列表

文件名	功能描述
problem1.python	问题一程序代码
problem2.python	问题二程序代码
problem3.python	问题三程序代码
problem4final.python	问题四程序代码
problem5.python	问题五程序代码

附录 B　代码

Problem1.py

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.integrate as integrate

eps = 1e-12
epsangle = 0.00001 * np.pi
maxangle = 32 * np.pi
cnt = 224
length0 = 2.86
length1 = 1.65
a = 0.55 / (2 * np.pi)

def r0(angle):
    return a * angle
def r1():
    return a
def s(x):
    return np.sqrt(r0(x) ** 2 + r1() ** 2)
def get_s(pre, now):
    ss = integrate.quad(s, pre, now)
    return ss[0]
def get_angle(t):
```

```
    l = 0.0
    r = 32.0 * np.pi
    while r - l > eps:
        mid = (l + r) / 2.0
        if get_s(mid, maxangle) <= t:r = mid
        else:l = mid
    return l
def get_x(angle):
    ans = r0(angle) * np.cos(angle)
    return ans
def get_y(angle):
    ans = r0(angle) * np.sin(angle)
    return ans
def get_xy(angle):
    return [get_x(angle), get_y(angle)]
def get_dis(x1, y1, x2, y2):
    return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
def check3(pre, now, len):
    pre_x = get_x(pre)
    pre_y = get_y(pre)
    now_x = get_x(now)
    now_y = get_y(now)
    return get_dis(pre_x, pre_y, now_x, now_y) <= len
def get_next_angle(x, len):
    l = x
    r = x + np.pi / 2
    while r - l > eps:
        mid = (l + r) / 2.0
        if check3(x, mid, len):l = mid
        else:r = mid
    return l
def get_next_v(pre, now, prev, len):
    pre2 = pre - epsangle
    now2 = get_next_angle(pre2, len)
    s_pre = get_s(pre2, pre)
```

```
    s_now = get_s(now2, now)
    return prev * s_now / s_pre
def visualization(pos, v):
    print(len(pos))
    print(pos)
    x_coords = [p[0] for p in pos]
    y_coords = [p[1] for p in pos]

    plt.figure(figsize=(8, 8))
    plt.plot(x_coords, y_coords, marker='o')
    plt.title('Position Visualization')
    plt.xlabel('X Coordinate')
    plt.ylabel('Y Coordinate')
    plt.axis('equal')
    plt.show()

    print(len(v))
    print(v)
    plt.figure(figsize=(10, 6))
    plt.plot(range(len(v)), v, marker='o', linestyle='-',
   color='b', label='Velocity (v)')
    plt.xlabel('Index')
    plt.ylabel('Velocity')
    plt.title('Velocity Change in Iteration 299')
    plt.legend()
    plt.grid(True)
    plt.show()
def solve(t0, t1):
    for t in range(t0, t1):
        print(t)
        pos = []
        pos2 = []
        v = []
        angle = get_angle(t)
        prev = 1.0
```

```
        pos.append(get_x(angle))
        pos.append(get_y(angle))
        pos2.append(get_xy(angle))
        v.append(1.0)
        for i in range(cnt):
            angle2 = 0
            v2 = 0
            if i == 0:
                angle2 = get_next_angle(angle, length0)
                #if(angle2 > maxangle):break
                v2 = get_next_v(angle, angle2, prev, length0)
                v.append(v2)
            else:
                angle2 = get_next_angle(angle, length1)
                #if(angle2 > maxangle):break
                v2 = get_next_v(angle, angle2, prev, length1)
                v.append(v2)
            pos.append(get_x(angle2))
            pos.append(get_y(angle2))
            pos2.append(get_xy(angle2))
            angle = angle2
            prev = v2

        print(pos)

        print(v)

        # 将 pos 列表转换为 DataFrame，其中每个元素占一行

        df_speed = pd.DataFrame(v, columns=[t])

        print(df_speed)

        # 读取 Excel 文件
        file_path = 'result1.xlsx'
```

```
        df = pd.read_excel(file_path)

        print(df)
        df[f"{t} s"] = df_speed[t]

        # 保存更新后的 DataFrame 回到 Excel
        df.to_excel(file_path, index=False)

        print("数据已成功写入 Excel 文件。")

        if(t == t1 - 1):
            visualization(pos2, v)

solve(0, 301)
```

Problem2.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
import math
import scipy.integrate as integrate

eps = 1e-12
epsangle = 0.00001 * np.pi
maxangle = 32 * np.pi
cnt = 233
length0 = 2.86
length1 = 1.65
a = 0.55 / (2 * np.pi)

def r0(angle):
    return a * angle
def r1():
    return a
```

```
def s(x):
    return np.sqrt(r0(x) ** 2 + r1() ** 2)
def get_s(pre, now):
    ss = integrate.quad(s, pre, now)
    return ss[0]
def get_angle(t):
    l = 0.0
    r = 32.0 * np.pi
    while r - l > eps:
        mid = (l + r) / 2.0
        if get_s(mid, maxangle) <= t:r = mid
        else:l = mid
    return l
def get_x(angle):
    ans = r0(angle) * np.cos(angle)
    return ans
def get_y(angle):
    ans = r0(angle) * np.sin(angle)
    return ans
def get_xy(angle):
    return [get_x(angle), get_y(angle)]
def get_dis(x1, y1, x2, y2):
    return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
def check3(pre, now, len):
    pre_x = get_x(pre)
    pre_y = get_y(pre)
    now_x = get_x(now)
    now_y = get_y(now)
    return get_dis(pre_x, pre_y, now_x, now_y) <= len
def get_next_angle(x, len):
    l = x
    r = x + np.pi / 2
    while r - l > eps:
        mid = (l + r) / 2.0
        if check3(x, mid, len):l = mid
```

```
55          else:r = mid
56      return l
57  def get_next_v(pre, now, prev, len):
58      pre2 = pre - epsangle
59      now2 = get_next_angle(pre2, len)
60      s_pre = get_s(pre2, pre)
61      s_now = get_s(now2, now)
62      return prev * s_now / s_pre
63  def visualization(pos, v):
64      print(len(pos))
65      print(pos)
66      x_coords = [p[0] for p in pos]
67      y_coords = [p[1] for p in pos]
68
69      plt.figure(figsize=(8, 8))
70      plt.plot(x_coords, y_coords, marker='o')
71      plt.title('Position Visualization')
72      plt.xlabel('X Coordinate')
73      plt.ylabel('Y Coordinate')
74      plt.axis('equal')
75      plt.show()
76
77      print(len(v))
78      print(v)
79      plt.figure(figsize=(10, 6))
80      plt.plot(range(len(v)), v, marker='o', linestyle='-',
    color='b', label='Velocity (v)')
81      plt.xlabel('Index')
82      plt.ylabel('Velocity')
83      plt.title('Velocity Change in Iteration 299')
84      plt.legend()
85      plt.grid(True)
86      plt.show()
87  def solve(t0, t1, step = 1, flag = 0):
88      for t in np.arange(t0, t1, step):
```

```
        pos = []
        pos2 = []
        v = []
        angle = get_angle(t)
        prev = 1.0
        pos.append(get_x(angle))
        pos.append(get_y(angle))
        pos2.append(get_xy(angle))
        if flag:v.append(1.0)
        for i in range(cnt):
            angle2 = 0
            v2 = 0
            if i == 0:
                angle2 = get_next_angle(angle, length0)
                #if(angle2 > maxangle):break
                if flag:
                    v2 = get_next_v(angle, angle2, prev,
    length0)
                    v.append(v2)
            else:
                angle2 = get_next_angle(angle, length1)
                #if(angle2 > maxangle):break
                if flag:
                    v2 = get_next_v(angle, angle2, prev,
    length1)
                    v.append(v2)
            pos.append(get_x(angle2))
            pos.append(get_y(angle2))
            pos2.append(get_xy(angle2))
            angle = angle2
            prev = v2
        if(t == t1 - 1):
            #visualization(pos2, v)
            return pos2, v
def GetCross(x1, y1, x2, y2, x, y):
```

```
122     return (x2 - x1) * (y - y1) - (x - x1) * (y2 - y1)
123 def is_collision(x1, y1, x2, y2, x3, y3, x4, y4, x, y):
124     return (GetCross(x1, y1, x2, y2, x, y) * GetCross(x3, y3,
    x4, y4, x, y) >= 0 and
125             GetCross(x2, y2, x3, y3, x, y) * GetCross(x4, y4,
    x1, y1, x, y) >= 0)
126 def sort_counterclockwise(x1, y1, x2, y2, x3, y3, x4, y4):
127     points = [(x1, y1), (x2, y2), (x3, y3), (x4, y4)]
128     center_x = (x1 + x2 + x3 + x4) / 4.0
129     center_y = (y1 + y2 + y3 + y4) / 4.0
130     def angle_from_center(point):
131         x, y = point
132         return math.atan2(y - center_y, x - center_x)
133     points_sorted = sorted(points, key=angle_from_center)
134     return points_sorted[0][0], points_sorted[0][1],
    points_sorted[1][0], points_sorted[1][1], \
135         points_sorted[2][0], points_sorted[2][1],
    points_sorted[3][0], points_sorted[3][1]
136 def is_rect_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5,
    x6, y6, x7, y7, x8, y8):
137     x1, y1, x2, y2, x3, y3, x4, y4 = sort_counterclockwise(x1,
     y1, x2, y2, x3, y3, x4, y4)
138     x5, y5, x6, y6, x7, y7, x8, y8 = sort_counterclockwise(x5,
     y5, x6, y6, x7, y7, x8, y8)
139     return (is_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5,
    y5) or
140                 is_collision(x1, y1, x2, y2, x3, y3, x4, y4,
    x6, y6) or
141                 is_collision(x1, y1, x2, y2, x3, y3, x4, y4,
    x7, y7) or
142                 is_collision(x1, y1, x2, y2, x3, y3, x4, y4,
    x8, y8) or
143                 is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x1, y1) or
144                 is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
```

```
    x2, y2) or
145                 is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x3, y3) or
146                 is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x4, y4))
147 def get_xxyy(pos, i):
148     x1 = pos[i][0]
149     y1 = pos[i][1]
150     x2 = pos[i + 1][0]
151     y2 = pos[i + 1][1]
152     if x1 > x2: x1, y1, x2, y2 = x2, y2, x1, y1
153     return x1, y1, x2, y2
154 def get_rectangle(xx1, yy1, xx2, yy2, len):
155     lenx = 0.15 * (yy2 - yy1) / len
156     leny = 0.15 * (xx2 - xx1) / len
157     xx3 = xx1 + (xx2 - xx1) * (len + 0.275) / len
158     yy3 = yy1 + (yy2 - yy1) * (len + 0.275) / len
159     x1 = xx3 + lenx
160     x2 = xx3 - lenx
161     y1 = yy3 - leny
162     y2 = yy3 + leny
163     xx4 = xx2 + (xx1 - xx2) * (len + 0.275) / len
164     yy4 = yy2 + (yy1 - yy2) * (len + 0.275) / len
165     x3 = xx4 - lenx
166     x4 = xx4 + lenx
167     y3 = yy4 + leny
168     y4 = yy4 - leny
169     return x1, y1, x2, y2, x3, y3, x4, y4
170 def plot_rectangles(rects):
171     fig, ax = plt.subplots()
172     for rect in rects:
173         x1, y1, x2, y2, x3, y3, x4, y4 = rect
174         polygon = Polygon([[x1, y1], [x2, y2], [x3, y3], [x4,
    y4]], closed=True, edgecolor='r', fill=False)
175         ax.add_patch(polygon)
```

```
176
177     ax.set_aspect('equal')
178     plt.xlim([-10, 10])  # Adjust limits based on expected
    coordinates
179     plt.ylim([-10, 10])
180     plt.grid(True)
181     plt.title('Rectangles Visualization')
182     plt.show()
183 def get_term_t1(start, end, step):
184     for t in np.arange(start, end, step):
185         pos, v = solve(t, t + 1, step)
186         xx1, yy1, xx2, yy2 = get_xxyy(pos, 0)
187         x1, y1, x2, y2, x3, y3, x4, y4 = get_rectangle(xx1,
    yy1, xx2, yy2, length0)
188         xx11, yy11, xx21, yy21 = get_xxyy(pos, 1)
189         x11, y11, x21, y21, x31, y31, x41, y41 = get_rectangle
    (xx11, yy11, xx21, yy21, length0)
190
191         for i in range(2, 60):
192             xx3, yy3, xx4, yy4 = get_xxyy(pos, i)
193             x5, y5, x6, y6, x7, y7, x8, y8 = get_rectangle(xx3
    , yy3, xx4, yy4, length1)
194             if i == 2:
195                 if (is_rect_collision(x1, y1, x2, y2, x3, y3,
    x4, y4, x5, y5, x6, y6, x7, y7, x8, y8)):
196                     return t
197             else:
198                 if (is_rect_collision(x1, y1, x2, y2, x3, y3,
    x4, y4, x5, y5, x6, y6, x7, y7, x8, y8) or
199                         is_rect_collision(x11, y11, x21, y21,
    x31, y31, x41, y41, x5, y5, x6, y6, x7, y7, x8, y8)):
200                     return t
201 def get_term_t():
202     t = get_term_t1(400, 1000,  1)
203     step = 1.0
```

```
    for i in range(9):
        t = get_term_t1(t - step, t + step, step / 10)
        step = step / 10
    print(t)
    rects = []
    pos, v = solve(t, t + 1, 1, 1)
    print(pos)
    print(v)
    xx1, yy1, xx2, yy2 = get_xxyy(pos, 0)
    rects.append(get_rectangle(xx1, yy1, xx2, yy2, length0))
    for i in range(1, cnt):
        xx3, yy3, xx4, yy4 = get_xxyy(pos, i)
        rects.append(get_rectangle(xx3, yy3, xx4, yy4, length1
  ))
    plot_rectangles(rects)
get_term_t()
```

Problem3.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
import math
import scipy.integrate as integrate

eps = 1e-8
epsangle = 0.00001 * np.pi
maxangle = 32 * np.pi
cnt = 233
length0 = 2.86
length1 = 1.65
a = 0.55 / (2 * np.pi)

def r0(angle):
    return a * angle
```

```
def r1():
    return a
def s(x):
    return np.sqrt(r0(x) ** 2 + r1() ** 2)
def get_s(pre, now):
    ss = integrate.quad(s, pre, now)
    return ss[0]
def get_angle(t):
    l = 0.0
    r = 32.0 * np.pi
    while r - l > eps:
        mid = (l + r) / 2.0
        if get_s(mid, maxangle) <= t:r = mid
        else:l = mid
    return l
def get_x(angle):
    ans = r0(angle) * np.cos(angle)
    return ans
def get_y(angle):
    ans = r0(angle) * np.sin(angle)
    return ans
def get_xy(angle):
    return [get_x(angle), get_y(angle)]
def get_dis(x1, y1, x2, y2):
    return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
def get_next_angle(x, len):
    def check(pre, now, len):
        pre_x = get_x(pre)
        pre_y = get_y(pre)
        now_x = get_x(now)
        now_y = get_y(now)
        return get_dis(pre_x, pre_y, now_x, now_y) <= len
    l = x
    r = x + np.pi / 2
    while r - l > eps:
```

```
        mid = (l + r) / 2.0
        if check(x, mid, len):l = mid
        else:r = mid
    return l
def get_next_v(pre, now, prev, len):
    pre2 = pre - epsangle
    now2 = get_next_angle(pre2, len)
    s_pre = get_s(pre2, pre)
    s_now = get_s(now2, now)
    return prev * s_now / s_pre

def visualization(pos, v):
    print(len(pos))
    print(pos)
    x_coords = [p[0] for p in pos]
    y_coords = [p[1] for p in pos]

    plt.figure(figsize=(8, 8))
    plt.plot(x_coords, y_coords, marker='o')
    plt.title('Position Visualization')
    plt.xlabel('X Coordinate')
    plt.ylabel('Y Coordinate')
    plt.axis('equal')
    plt.show()

    print(len(v))
    print(v)
    plt.figure(figsize=(10, 6))
    plt.plot(range(len(v)), v, marker='o', linestyle='-',
   color='b', label='Velocity (v)')
    plt.xlabel('Index')
    plt.ylabel('Velocity')
    plt.title('Velocity Change in Iteration 299')
    plt.legend()
    plt.grid(True)
```

```
    plt.show()
def solve(t, t1, step = 1, flag = 0):
    pos = []
    pos2 = []
    v = []
    angle = get_angle(t)
    prev = 1.0
    pos.append(get_x(angle))
    pos.append(get_y(angle))
    pos2.append(get_xy(angle))
    if flag: v.append(1.0)
    for i in range(cnt):
        angle2 = 0
        v2 = 0
        if i == 0:
            angle2 = get_next_angle(angle, length0)
            # if(angle2 > maxangle):break
            if flag:
                v2 = get_next_v(angle, angle2, prev, length0)
                v.append(v2)
        else:
            angle2 = get_next_angle(angle, length1)
            # if(angle2 > maxangle):break
            if flag:
                v2 = get_next_v(angle, angle2, prev, length1)
                v.append(v2)
        pos.append(get_x(angle2))
        pos.append(get_y(angle2))
        pos2.append(get_xy(angle2))
        angle = angle2
        prev = v2
    # visualization(pos2, v)
    return pos2, v
def GetCross(x1, y1, x2, y2, x, y):
    return (x2 - x1) * (y - y1) - (x - x1) * (y2 - y1)
```

```
122 def is_collision(x1, y1, x2, y2, x3, y3, x4, y4, x, y):
123     return (GetCross(x1, y1, x2, y2, x, y) * GetCross(x3, y3,
    x4, y4, x, y) >= 0 and
124             GetCross(x2, y2, x3, y3, x, y) * GetCross(x4, y4,
    x1, y1, x, y) >= 0)
125 def sort_counterclockwise(x1, y1, x2, y2, x3, y3, x4, y4):
126     points = [(x1, y1), (x2, y2), (x3, y3), (x4, y4)]
127     center_x = (x1 + x2 + x3 + x4) / 4.0
128     center_y = (y1 + y2 + y3 + y4) / 4.0
129     def angle_from_center(point):
130         x, y = point
131         return math.atan2(y - center_y, x - center_x)
132     points_sorted = sorted(points, key=angle_from_center)
133     return points_sorted[0][0], points_sorted[0][1],
    points_sorted[1][0], points_sorted[1][1], \
134         points_sorted[2][0], points_sorted[2][1],
    points_sorted[3][0], points_sorted[3][1]
135 def is_rect_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5,
    x6, y6, x7, y7, x8, y8):
136     x1, y1, x2, y2, x3, y3, x4, y4 = sort_counterclockwise(x1,
     y1, x2, y2, x3, y3, x4, y4)
137     x5, y5, x6, y6, x7, y7, x8, y8 = sort_counterclockwise(x5,
     y5, x6, y6, x7, y7, x8, y8)
138     return (is_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5,
    y5) or
139                 is_collision(x1, y1, x2, y2, x3, y3, x4, y4,
    x6, y6) or
140                 is_collision(x1, y1, x2, y2, x3, y3, x4, y4,
    x7, y7) or
141                 is_collision(x1, y1, x2, y2, x3, y3, x4, y4,
    x8, y8) or
142                 is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x1, y1) or
143                 is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x2, y2) or
```

```
144                 is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x3, y3) or
145                 is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x4, y4))
146 def get_xxyy(pos, i):
147     x1 = pos[i][0]
148     y1 = pos[i][1]
149     x2 = pos[i + 1][0]
150     y2 = pos[i + 1][1]
151     if x1 > x2: x1, y1, x2, y2 = x2, y2, x1, y1
152     return x1, y1, x2, y2
153 def get_rectangle(xx1, yy1, xx2, yy2, len):
154     lenx = 0.15 * (yy2 - yy1) / len
155     leny = 0.15 * (xx2 - xx1) / len
156     xx3 = xx1 + (xx2 - xx1) * (len + 0.275) / len
157     yy3 = yy1 + (yy2 - yy1) * (len + 0.275) / len
158     x1 = xx3 + lenx
159     x2 = xx3 - lenx
160     y1 = yy3 - leny
161     y2 = yy3 + leny
162     xx4 = xx2 + (xx1 - xx2) * (len + 0.275) / len
163     yy4 = yy2 + (yy1 - yy2) * (len + 0.275) / len
164     x3 = xx4 - lenx
165     x4 = xx4 + lenx
166     y3 = yy4 + leny
167     y4 = yy4 - leny
168     return x1, y1, x2, y2, x3, y3, x4, y4
169 def plot_rectangles(rects):
170     fig, ax = plt.subplots()
171     for rect in rects:
172         x1, y1, x2, y2, x3, y3, x4, y4 = rect
173         polygon = Polygon([[x1, y1], [x2, y2], [x3, y3], [x4,
    y4]], closed=True, edgecolor='r', fill=False)
174         ax.add_patch(polygon)
175
```

```
    ax.set_aspect('equal')
    plt.xlim([-10, 10])  # Adjust limits based on expected coordinates
    plt.ylim([-10, 10])
    plt.grid(False)
    plt.title('Rectangles Visualization')
    plt.show()
def is_t_collision(t):
    pos, v = solve(t, t + 1)
    xx1, yy1, xx2, yy2 = get_xxyy(pos, 0)
    x1, y1, x2, y2, x3, y3, x4, y4 = get_rectangle(xx1, yy1, xx2, yy2, length0)
    xx11, yy11, xx21, yy21 = get_xxyy(pos, 1)
    x11, y11, x21, y21, x31, y31, x41, y41 = get_rectangle(xx11, yy11, xx21, yy21, length0)

    for i in range(2, 60):
        xx3, yy3, xx4, yy4 = get_xxyy(pos, i)
        x5, y5, x6, y6, x7, y7, x8, y8 = get_rectangle(xx3, yy3, xx4, yy4, length1)
        if i == 2:
            if (is_rect_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7, x8, y8)):
                return True
        else:
            if (is_rect_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7, x8, y8) or
                    is_rect_collision(x11, y11, x21, y21, x31, y31, x41, y41, x5, y5, x6, y6, x7, y7, x8, y8)):
                return True

def get_term_t(flag = 0):
    def get_term_t1(start, end, step):
        for t in np.arange(start, end, step):
            if is_t_collision(t): return t
```

```
    t = get_term_t1(0, 500,  1.0)
    step = 1.0
    for i in range(9):
        t = get_term_t1(t - step, t + step, step / 10)
        step = step / 10

    rects = []
    pos, v = solve(t, t + 1, 1, 1)
    xx1, yy1, xx2, yy2 = get_xxyy(pos, 0)
    rects.append(get_rectangle(xx1, yy1, xx2, yy2, length0))
    for i in range(1, cnt):
        xx3, yy3, xx4, yy4 = get_xxyy(pos, i)
        rects.append(get_rectangle(xx3, yy3, xx4, yy4, length1
))
    if flag:
        plot_rectangles(rects)
    return t

def geta(x):
    global a
    a = x / (2 * np.pi)
def get_pitch():
    def check(x, flag = 0):
        geta(x)
        t = get_term_t(flag)
        return r0(get_angle(t)) <= 4.5
    l = 0.3
    r = 0.55
    while r - l > eps:
        mid = (l + r) / 2.0
        print(mid)
        if check(mid):r = mid
        else:l = mid
    print(l)
    check(l, 1)
```

```
238
239 get_pitch()
```

Problem4final.py

```
1  import os
2  import numpy as np
3  import pandas as pd
4  import matplotlib.pyplot as plt
5  from matplotlib.patches import Polygon
6  import math
7  import scipy.integrate as integrate
8
9  eps = 1e-12
10 eps2 = 1e-9
11 epsangle = 0.00001 * np.pi
12 maxangle = 32 * np.pi
13 cnt = 233
14 length0 = 2.86
15 length1 = 1.65
16 a = 0.55 / (2 * np.pi)
17
18 def r0(angle):
19     return a * angle
20 def r1():
21     return a
22 def s(x):
23     return np.sqrt(r0(x) ** 2 + r1() ** 2)
24 def get_s(pre, now):
25     ss = integrate.quad(s, pre, now)
26     return ss[0]
27 def get_angle(t):
28     l = 0.0
29     r = 32.0 * np.pi
30     while r - l > eps:
31         mid = (l + r) / 2.0
```

```
        if get_s(mid, maxangle) <= t:r = mid
        else:l = mid
    return l
def get_x(angle):
    ans = r0(angle) * np.cos(angle)
    return ans
def get_y(angle):
    ans = r0(angle) * np.sin(angle)
    return ans
def get_xy(angle):
    return [get_x(angle), get_y(angle)]
def get_dis(x1, y1, x2 = 0, y2 = 0):
    return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
def get_s2(x1, y1, x2, y2, xo, yo, r):
    return r * math.acos(((x1 - xo) * (x2 - xo) + (y1 - yo) *
  (y2 - yo)) / (r * r))
def rotate(x1, y1, x2, y2, angle):
    x = (x1 - x2) * np.cos(angle) - (y1 - y2) * np.sin(angle)
  + x2
    y = (y1 - y2) * np.cos(angle) + (x1 - x2) * np.sin(angle)
  + y2
    return x, y
def get_next_angle(x, len):
    def check(pre, now, len):
        pre_x = get_x(pre)
        pre_y = get_y(pre)
        now_x = get_x(now)
        now_y = get_y(now)
        return get_dis(pre_x, pre_y, now_x, now_y) <= len
    l = x
    r = x + np.pi / 2
    while r - l > eps:
        mid = (l + r) / 2.0
        if check(x, mid, len):l = mid
        else:r = mid
```

```
64      return l
65  def get_next_v(pre, now, prev, len):
66      pre2 = pre - epsangle
67      now2 = get_next_angle(pre2, len)
68      s_pre = get_s(pre2, pre)
69      s_now = get_s(now2, now)
70      return prev * s_now / s_pre
71  def visualization(pos, v):
72      print(len(pos))
73      print(pos)
74      x_coords = [p[0] for p in pos]
75      y_coords = [p[1] for p in pos]
76
77      plt.figure(figsize=(8, 8))
78      plt.plot(x_coords, y_coords, marker='o')
79      plt.title('Position Visualization')
80      plt.xlabel('X Coordinate')
81      plt.ylabel('Y Coordinate')
82      plt.axis('equal')
83      plt.show()
84
85      print(len(v))
86      print(v)
87      plt.figure(figsize=(10, 6))
88      plt.plot(range(len(v)), v, marker='o', linestyle='-',
    color='b', label='Velocity (v)')
89      plt.xlabel('Index')
90      plt.ylabel('Velocity')
91      plt.title('Velocity Change in Iteration 299')
92      plt.legend()
93      plt.grid(True)
94      plt.show()
95  def solve(t, t1, step = 1, flag = 0):
96      pos = []
97      pos2 = []
```

```
    v = []
    angle = get_angle(t)
    prev = 1.0
    pos.append(get_x(angle))
    pos.append(get_y(angle))
    pos2.append(get_xy(angle))
    if flag: v.append(1.0)
    for i in range(cnt):
        angle2 = 0
        v2 = 0
        if i == 0:
            angle2 = get_next_angle(angle, length0)
            # if(angle2 > maxangle):break
            if flag:
                v2 = get_next_v(angle, angle2, prev, length0)
                v.append(v2)
        else:
            angle2 = get_next_angle(angle, length1)
            # if(angle2 > maxangle):break
            if flag:
                v2 = get_next_v(angle, angle2, prev, length1)
                v.append(v2)
        pos.append(get_x(angle2))
        pos.append(get_y(angle2))
        pos2.append(get_xy(angle2))
        angle = angle2
        prev = v2
    # visualization(pos2, v)
    return pos2, v
def GetCross(x1, y1, x2, y2, x, y):
    return (x2 - x1) * (y - y1) - (x - x1) * (y2 - y1)
def is_collision(x1, y1, x2, y2, x3, y3, x4, y4, x, y):
    return (GetCross(x1, y1, x2, y2, x, y) * GetCross(x3, y3,
   x4, y4, x, y) >= 0 and
            GetCross(x2, y2, x3, y3, x, y) * GetCross(x4, y4,
```

```
    x1, y1, x, y) >= 0)
def sort_counterclockwise(x1, y1, x2, y2, x3, y3, x4, y4):
    points = [(x1, y1), (x2, y2), (x3, y3), (x4, y4)]
    center_x = (x1 + x2 + x3 + x4) / 4.0
    center_y = (y1 + y2 + y3 + y4) / 4.0
    def angle_from_center(point):
        x, y = point
        return math.atan2(y - center_y, x - center_x)
    points_sorted = sorted(points, key=angle_from_center)
    return points_sorted[0][0], points_sorted[0][1],
    points_sorted[1][0], points_sorted[1][1], \
        points_sorted[2][0], points_sorted[2][1],
    points_sorted[3][0], points_sorted[3][1]
def is_rect_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5,
    x6, y6, x7, y7, x8, y8):
    x1, y1, x2, y2, x3, y3, x4, y4 = sort_counterclockwise(x1,
    y1, x2, y2, x3, y3, x4, y4)
    x5, y5, x6, y6, x7, y7, x8, y8 = sort_counterclockwise(x5,
    y5, x6, y6, x7, y7, x8, y8)
    return (is_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5,
    y5) or
                is_collision(x1, y1, x2, y2, x3, y3, x4, y4,
    x6, y6) or
                is_collision(x1, y1, x2, y2, x3, y3, x4, y4,
    x7, y7) or
                is_collision(x1, y1, x2, y2, x3, y3, x4, y4,
    x8, y8) or
                is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x1, y1) or
                is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x2, y2) or
                is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x3, y3) or
                is_collision(x5, y5, x6, y6, x7, y7, x8, y8,
    x4, y4))
```

```
def get_xxyy(pos, i):
    x1 = pos[i][0]
    y1 = pos[i][1]
    x2 = pos[i + 1][0]
    y2 = pos[i + 1][1]
    if x1 > x2: x1, y1, x2, y2 = x2, y2, x1, y1
    return x1, y1, x2, y2
def get_rectangle(xx1, yy1, xx2, yy2, len):
    lenx = 0.15 * (yy2 - yy1) / len
    leny = 0.15 * (xx2 - xx1) / len
    xx3 = xx1 + (xx2 - xx1) * (len + 0.275) / len
    yy3 = yy1 + (yy2 - yy1) * (len + 0.275) / len
    x1 = xx3 + lenx
    x2 = xx3 - lenx
    y1 = yy3 - leny
    y2 = yy3 + leny
    xx4 = xx2 + (xx1 - xx2) * (len + 0.275) / len
    yy4 = yy2 + (yy1 - yy2) * (len + 0.275) / len
    x3 = xx4 - lenx
    x4 = xx4 + lenx
    y3 = yy4 + leny
    y4 = yy4 - leny
    return x1, y1, x2, y2, x3, y3, x4, y4
def plot_rectangles(rects):
    fig, ax = plt.subplots()
    for rect in rects:
        x1, y1, x2, y2, x3, y3, x4, y4 = rect
        polygon = Polygon([[x1, y1], [x2, y2], [x3, y3], [x4,
   y4]], closed=True, edgecolor='r', fill=False)
        ax.add_patch(polygon)

    ax.set_aspect('equal')
    plt.xlim([-10, 10])  # Adjust limits based on expected
   coordinates
    plt.ylim([-10, 10])
```

```
    plt.grid(False)
    plt.title('Rectangles Visualization')
    plt.show()
def is_t_collision(t):
    pos, v = solve(t, t + 1)
    xx1, yy1, xx2, yy2 = get_xxyy(pos, 0)
    x1, y1, x2, y2, x3, y3, x4, y4 = get_rectangle(xx1, yy1, xx2, yy2, length0)
    xx11, yy11, xx21, yy21 = get_xxyy(pos, 1)
    x11, y11, x21, y21, x31, y31, x41, y41 = get_rectangle(xx11, yy11, xx21, yy21, length1)

    for i in range(2, 60):
        xx3, yy3, xx4, yy4 = get_xxyy(pos, i)
        x5, y5, x6, y6, x7, y7, x8, y8 = get_rectangle(xx3, yy3, xx4, yy4, length1)
        if i == 2:
            if (is_rect_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7, x8, y8)):
                return True
        else:
            if (is_rect_collision(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7, x8, y8) or
                    is_rect_collision(x11, y11, x21, y21, x31, y31, x41, y41, x5, y5, x6, y6, x7, y7, x8, y8)):
                return True
def get_term_t(flag = 0):
    def get_term_t1(start, end, step):
        for t in np.arange(start, end, step):
            if is_t_collision(t): return t
    t = get_term_t1(0, 500,  1.0)
    step = 1.0
    for i in range(9):
        t = get_term_t1(t - step, t + step, step / 10)
        step = step / 10
```

```
215
216     rects = []
217     pos, v = solve(t, t + 1, 1, 1)
218     xx1, yy1, xx2, yy2 = get_xxyy(pos, 0)
219     rects.append(get_rectangle(xx1, yy1, xx2, yy2, length0))
220     for i in range(1, cnt):
221         xx3, yy3, xx4, yy4 = get_xxyy(pos, i)
222         rects.append(get_rectangle(xx3, yy3, xx4, yy4, length1
    ))
223     if flag:
224         plot_rectangles(rects)
225     return t
226 def geta(x):
227     global a
228     a = x / (2 * np.pi)
229 def get_pitch():
230     def check(x, flag = 0):
231         geta(x)
232         t = get_term_t(flag)
233         return r0(get_angle(t)) <= 4.5
234     l = 0.3
235     r = 0.55
236     while r - l > eps:
237         mid = (l + r) / 2.0
238         print(mid)
239         if check(mid):r = mid
240         else:l = mid
241     print(l)
242     check(l, 1)
243
244 geta(1.7)
245 angle = 4.5 / a
246 t0 = get_s(angle, maxangle)
247 def solve4(t, t_0):#t秒开始圆周运动，求t_0秒的位置
248     angle2 = get_angle(t)
```

```
    x1 = get_x(angle2)
    y1 = get_y(angle2)
    x2 = -x1
    y2 = -y1
    x3 = (x1 + 2 * x2) / 3
    y3 = (y1 + 2 * y2) / 3
    d = 2 * r0(angle2)

    def geto():
        dy = np.sin(angle2) + angle2 * np.cos(angle2)
        dx = np.cos(angle2) - angle2 * np.sin(angle2)
        def f(x):
            return -dx / dy * (x - x1) + y1
        l = 0
        r = d
        while r - l > eps:
            mid = (l + r) / 2
            if abs(dy) <= eps:
                if get_dis(x1, y1 + mid, x1, y1) <= get_dis(x1, y1 + mid, x3, y3):l = mid
                else:r = mid
            else:
                if get_dis(x1 + mid, f(x1 + mid), x1, y1) <= get_dis(x1 + mid, f(x1 + mid), x3, y3) :l = mid
                else:r = mid
        if abs(dy) <= eps:
            if abs(get_dis(x1, y1 + l, x1, y1) - get_dis(x1, y1 + l, x3, y3)) <= eps2:
                return x1, y1 + l, -x1, -(y1 + l + y1) / 2
        else:
            if abs(get_dis(x1 + l, f(x1 + l), x1, y1) - get_dis(x1 + l, f(x1 + l), x3, y3)) <= eps2:
                return x1 + l, f(x1 + l), -(x1 + l + x1) / 2, -(f(x1 + l) + y1) / 2
        l = 0
```

```
279         r = -d
280         while r - l > eps:
281             mid = (l + r) / 2
282             if abs(dy) <= eps:
283                 if get_dis(x1, y1 + mid, x1, y1) <= get_dis(x1
    , y1 + mid, x3, y3):l = mid
284                 else:r = mid
285             else:
286                 if get_dis(x1 + mid, f(x1 + mid), x1, y1) <=
    get_dis(x1 + mid, f(x1 + mid), x3, y3):l = mid
287                 else:r = mid
288         if abs(dy) <= eps:
289             if abs(get_dis(x1, y1 + l, x1, y1) - get_dis(x1,
    y1 + l, x3, y3)) <= eps2:
290                 return x1, y1 + l, -x1, -(y1 + l + y1) / 2
291         else:
292             if abs(get_dis(x1 + l, f(x1 + l), x1, y1) -
    get_dis(x1 + l, f(x1 + l), x3, y3)) <= eps2:
293                 return x1 + l, f(x1 + l), -(x1 + l + x1) / 2,
    -(f(x1 + l) + y1) / 2
294         return 10000,10000,10000,10000
295
296     xo1, yo1, xo2, yo2 = geto()
297     if(xo1==yo1==xo2==yo2==10000):return [],[],1
298
299     ro1 = get_dis(xo1, yo1, x1, y1)
300     ro2 = get_dis(xo2, yo2, x2, y2)
301     so1 = get_s2(x1, y1, x3, y3, xo1, yo1, ro1)
302     so2 = get_s2(x2, y2, x3, y3, xo2, yo2, ro2)
303     def get_pos(tt, v):
304         if tt <= t:
305             return get_x(get_angle(tt)), get_y(get_angle(tt))
306         tt -= t
307         s = tt * v
308         if s <= so1:
```

```
309             return rotate(x1, y1, xo1, yo1, -s / ro1)
310         elif s <= so1 + so2:
311             s -= so1
312             return rotate(x3, y3, xo2, yo2, s / ro2)
313         else:
314             s -= so1 + so2
315             ttt = s / v
316             angle3 = get_angle(t - ttt)
317             return -get_x(angle3), -get_y(angle3)
318 
319     x_pre, y_pre = get_pos(t_0, 1.0)
320     t_pre = t_0
321     v_pre = 1.0
322     pos = []
323     v = []
324     pos.append([x_pre, y_pre])
325     v.append(v_pre)
326     def get_nowpos(xx, yy, tt, len):
327         def check(xx, yy, tt, len):
328             xx2, yy2 = get_pos(tt, 1.0)
329             return (get_dis(xx, yy, xx2, yy2) <= len)
330         l = 0
331         r = 3
332         while r - l > eps:
333             mid = (l + r) / 2.0
334             if check(xx, yy, tt - mid, len):l = mid
335             else:r = mid
336         xx2, yy2 = get_pos(tt - l, 1.0)
337         return xx2, yy2, tt - l
338 
339     def get_nowv(x_pre, y_pre, t_pre, v_pre, x_now, y_now,
    t_now, len):
340         epslen = 0.00000001
341         x2_pre, y2_pre, t2_pre = get_nowpos(x_pre, y_pre,
    t_pre, epslen)
```

```
342         x2_now, y2_now, t2_now = get_nowpos(x2_pre, y2_pre,
    t2_pre, len)
343         if t2_now > t_now: return -1
344         return get_dis(x2_now, y2_now, x_now, y_now) / epslen
    * v_pre
345 
346     for i in range(cnt):
347         if i == 0:
348             x_now, y_now, t_now = get_nowpos(x_pre, y_pre,
    t_pre, length0)
349             v_now = get_nowv(x_pre, y_pre, t_pre, v_pre, x_now
    , y_now, t_now, length0)
350             if v_now < -eps:return pos, v, 1
351             v.append(v_now)
352             v_pre = v_now
353         else:
354             x_now, y_now, t_now = get_nowpos(x_pre, y_pre,
    t_pre, length1)
355             v_now = get_nowv(x_pre, y_pre, t_pre, v_pre, x_now
    , y_now, t_now, length1)
356             if v_now < -eps:return pos, v, 1
357             v.append(v_now)
358             v_pre = v_now
359         x_pre, y_pre, t_pre = x_now, y_now, t_now
360         pos.append([x_pre, y_pre])
361 
362     #visualization(pos, [])
363     return pos, v, 0
364 
365 def collision(pos):
366     xx1, yy1, xx2, yy2 = get_xxyy(pos, 0)
367     x1, y1, x2, y2, x3, y3, x4, y4 = get_rectangle(xx1, yy1,
    xx2, yy2, length0)
368     xx11, yy11, xx21, yy21 = get_xxyy(pos, 1)
369     x11, y11, x21, y21, x31, y31, x41, y41 = get_rectangle(
```

```
    xx11, yy11, xx21, yy21, length1)

    for i in range(2, 40):
        xx3, yy3, xx4, yy4 = get_xxyy(pos, i)
        x5, y5, x6, y6, x7, y7, x8, y8 = get_rectangle(xx3,
    yy3, xx4, yy4, length1)
        if i == 2:
            if (is_rect_collision(x1, y1, x2, y2, x3, y3, x4,
    y4, x5, y5, x6, y6, x7, y7, x8, y8)):
                return True
        else:
            if (is_rect_collision(x1, y1, x2, y2, x3, y3, x4,
    y4, x5, y5, x6, y6, x7, y7, x8, y8) or
                    is_rect_collision(x11, y11, x21, y21, x31,
     y31, x41, y41, x5, y5, x6, y6, x7, y7, x8, y8)):
                return True

def check(x):
    angle2 = get_angle(x)
    d = 2 * r0(angle2)
    for t in np.arange(x, x + d, 1):
        pos, v, flag = solve4(x, t)
        if flag == 1 or collision(pos):
            return False
    return True

def getx():
    l = 9 + t0
    r = 11 + t0
    while r - l > eps2:
        mid = (l + r) / 2
        if check(mid):l = mid
        else:r = mid
    return l

```

```
x = 1340.177307524383

time = -100

for t in np.arange(x + 100, x + 101):
    pos, v, flag = solve4(x, t)
    visualization(pos, v)
    pos2 = []
    for i in range(len(pos)):
        pos2.append(pos[i][0])
        pos2.append(pos[i][1])
    print(pos2)

    print(v)


    # df_pos = pd.DataFrame(pos2,columns=[time+100])
    # df_speed = pd.DataFrame(v, columns=[time+100])


    # file_path1 = 'p4_pos.csv'
    # file_path2 = 'p4_speed.csv'

    # # Check if files exist, if not create them with the initial columns
    # if not os.path.exists(file_path1):
    #     df_pos = pd.DataFrame(pos2, columns=[time + 100])
    #     df_pos.to_csv(file_path1, index=False)

    # if not os.path.exists(file_path2):
    #     df_speed = pd.DataFrame(v, columns=[time + 100])
    #     df_speed.to_csv(file_path2, index=False)


    # df1 = pd.read_csv(file_path1)
```

```
    # df2 = pd.read_csv(file_path2)

    # df1[f"{time}"] = df_pos[time+100]
    # df1.to_csv(file_path1, index=False)

    # df2[f"{time}"] = df_speed[time+100]
    # df2.to_csv(file_path2, index=False)

    time = time+1

    print(time)
```