“板凳龙”闹元宵

摘要

“板凳龙”起源于汉代，是浙闽地区由来已久的传统民俗，深受老百姓喜爱。本文从优化角度出发，对舞龙队运动时的轨迹进行研究，分析舞龙队沿轨迹自由盘入、盘出的条件。综合考虑板凳的宽度、在不同位置的长度，孔径和初始条件，在舞龙队正常进行运动的假设下，建立“板凳龙”的阿基米德螺线模型 [1]，碰撞检测算法 [2] 等，可用于计算在特定情况下龙的运动速度、位置和运动路径，希望能够帮助舞龙队得到更好的观赏效果。

对于问题一：假设舞龙队正常运动，本文建立了二维平面上计算龙的运动的数学模型。首先建立极坐标系和平面直角坐标系，引入阿基米德螺线的公式：

$$r = 2\pi \cdot (t/p)$$

考虑到舞龙队龙头前把手匀速前进，确定龙头前把手的初始运动参数，得到描述龙头前把手的运动方程。我们知道龙身和龙尾的前后孔径通过把手连接且与龙头的距离固定，故其位置和速度可通过龙头的位置和速度及各节板凳之间的相对位置关系来确定。最后求解，具体结果见附录。

对于问题二：假设舞龙队正常运动，本文运用了碰撞检测模型，舞龙队在问题一设定的路径盘入，计算盘入终止时板凳不发生碰撞的时间点 t_2。首先，我们对板凳发生碰撞的因素 d——相邻两个板凳中一个的末端和另一个的起始点之间的距离进行分析。接着求解碰撞检测函数。最后，计算出舞龙队不能再盘入的时刻，并求出此时舞龙队的位置和速度。具体结果见附录。

对于问题三：假设舞龙队正常运动，运用遗传算法，舞龙队调为逆时针盘出，为求出让龙头沿相应螺线盘入到调头空间边界的最小螺距 p_{min}，通过初始化种群，锦标赛选择，单点交叉和变异操作，选择最好个体，创造新种群，得到最小螺距为 0.965235 米。

对于问题四：舞龙队调头路径为相切于盘入，盘出螺线且前一段圆弧半径是后一段的 2 倍的由两段圆弧相切而成的 S 形曲线，在问题三设定的调头空间内掉头，为求可否调整圆弧，保持各部分相切使调头曲线变短。首先，采用粒子群优化寻找合适的圆弧参数，用模拟退火算法在粒子群优化的基础上进行局部优化，将两种算法结合，确保圆弧与螺旋线相切并最小化路径总长度。结果为能调整圆弧。最后建立阿基米德螺旋线模型和指数递减的速度模型求一定时间范围内每秒舞龙队位置和速度，具体结果见附录。

对于问题五：舞龙队沿问题四的路径行进，龙头行进速度保持不变。在限定舞龙队各把手的速度均不超过 2 m/s 的条件下，基于问题四的模型进行改进并计算结果。我们采用人工蜂群 ABC 优化算法，对计算得到的结果进行对应，更新并记忆最好的结果，从而得到龙头的最大行进速度。

关键字：阿基米德螺线　碰撞检测模型　遗传算法　模拟退火算法　ABC 优化算法

一、 问题重述

1.1 问题背景

“板凳龙” 起源于汉代，是浙闽地区由来已久的传统民俗文化活动，它又被称为 “盘龙”，由龙头、龙身和龙尾三部分组成。一条龙从头到尾，由几十节、几百节甚至上千节板凳串联而成，长度从数百米到几千米不等。该项目用 223 节宽为 30cm 的板凳，一节长为 341cm 的板凳做龙头，一节做龙尾，221 节做龙身，龙身和龙尾的板长均为 220cm。相邻两条板凳通过把手连接。每节板凳上均有两个孔径为 5.5cm 的孔，孔中心距最近的板头 27.5cm。舞龙队沿着螺线形的轨迹行进，行进速度越快，且形成的盘龙面积越小，表演的观赏性就越高。

1.2 问题要求

问题 1 当舞龙队沿螺距为 55cm 的等距螺线顺时针盘入，各把手中心均位于螺线上，龙头前把手的行进速度始终保持为 1 m/s 且龙头位于螺线第 16 圈 A 点处时：(1) 请给出从初始时刻到 300s 为止，每秒整个舞龙队的位置和速度（指龙头、龙身和龙尾各前把手及龙尾后把手中心的位置和速度，下同) (2) 同时在论文中给出 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.

问题 2 当舞龙队沿问题 1 设定的螺线盘入时：(1) 请确定舞龙队盘入的终止时刻，使得板凳之间不发生碰撞（即舞龙队不能再继续盘入的时间），并给出此时舞龙队的位置和速度 (2) 同时在论文中给出此时龙头前把手、龙头后面第 1、51、101、151、201 条龙身前把手和龙尾后把手的位置和速度。

问题 3 从盘入到盘出，舞龙队将由顺时针盘入调头切换为逆时针盘出，这需要一定的调头空间。(1) 若调头空间是以螺线中心为圆心、直径为 9m 的圆形区域，请确定最小螺距，使得龙头前把手能够沿着相应的螺线盘入到调头空间的边界。

问题 4 当舞龙队盘入的螺距固定为 1.7m，盘出与盘入螺线关于螺线中心呈中心对称，由两段圆弧相切连接而成的 S 形曲线，前一段圆弧的半径是后一段的 2 倍的调头路径与盘入、盘出螺线均相切时，舞龙队在问题 3 设定的调头空间内完成调头：（1）能否调整圆弧，使各部分仍相切，但调头曲线变短?（2）以调头开始时间为零时刻，计算从 −100s 开始到 100s 结束，每秒舞龙队的位置和速度。

问题 5 当龙头行进速度不变，舞龙队沿问题 4 的路径行进时：（1）确定龙头的最大行进速度，使得舞龙队各把手的速度均不超过 2m/s。

二、 问题分析

2.1 问题一分析

对于问题一，本题要求我们在龙头前把手的运动参数确定时，计算从初始时刻到300 s 为止，每秒整个舞龙队的位置和速度。题中告知舞龙队沿等距螺线行进，首先我们可以根据等距螺线的定义写出舞龙队的运动轨迹公式，建立螺旋轨迹模型，然后通过龙头前把手的运动参数构建指数加速模型。其次考虑到龙身和龙尾与龙头前把手的运动轨迹相同，我们可以推导出其他前后把手的数学模型，最后利用这些公式和模型计算从初始时刻到 300 s 为止，每秒整个舞龙队的位置和速度。

2.2 问题二分析

对于问题二，我们需要沿用第一问的模型并对此做出相应改进。首先，基于问题一的模型，我们可以将问题二看做相邻两个板凳中一个的末端和另一个的起始点之间的距离与板凳自身长度间的关系。然后，运用碰撞检测算法。由于板凳长度固定，所以我们着重考虑相邻两个板凳中一个的末端和另一个的起始点之间的距离，以此来计算舞龙队不能再盘入的时刻和此时舞龙队的位置和速度。

2.3 问题三分析

对于问题三，在舞龙队将调为逆时针盘出，具有圆心为螺线中心，直径是 9m 的圆形区域的调头空间的情况下，为了求出让龙头前把手沿相应螺线盘入到调头空间边界的最小螺距 p_{min}，该问题可用遗传算法，转换为通过初始化种群，锦标赛选择，单点交叉和变异操作，寻找最好个体生成种群的问题，并运用螺旋路径函数计算出龙头前把手到调头空间边界的距离，最终求出满足要求的最小螺距 p_{min}。

2.4 问题四分析

对于问题四，在问题三设定的调头空间内掉头, 调头路径为相切于盘入，盘出螺线并且前一段圆弧的半径是后一段的 2 倍的由两段圆弧相切而成的 S 形曲线，已知盘入螺线螺距是 1.7m, 为了得知是否可以调整圆弧到保持各部分相切并且使调头曲线变短，我们采用粒子群优化算法进行全局搜索，寻找最佳的圆弧半径和中心位置。PSO 在广泛的搜索空间中寻找潜在的全局最优解，确定圆弧的参数以满足相切条件和最小化路径总长度. 接着，使用模拟退火算法优化粒子群搜索算法的局部搜索能力。具体来说，模拟退火用于细化粒子群优化的结果，尤其是在局部最优解附近进行更精细的调整，帮助使得结果更加精确。对于从 −100 s 开始到 100 s 为止，每秒舞龙队的位置和速度，我们已知以调头开始时间为零时刻，龙头前把手的行进速度始终保持 1 m/s。从而我们建立阿基

米德螺线模型，计算每节板凳的位置，使用

$$x(\theta) = r(\theta)\cos(\theta)$$

$$y(\theta) = r(\theta)\sin(\theta)$$

公式计算，为了计算每节板凳的速度，使用了一个指数递减模型：

$$v(i) = v_0 \cdot e^{-ki}$$

2.5 问题五分析

对于问题五，舞龙队沿问题四设定的路径行进，龙头行进速度保持不变。本文将舞龙队各把手的速度限制不得超过 2m/s。为计算出在此限定条件下龙头的最大行进速度 v_{max}，我们根据螺旋参数，在位置一和位置二的螺旋半径以及最大行进速度使用 ABC 算法进行计算。沿用问题四的模型并进行改进，人工蜂群 ABC 优化算法可以在公式计算得到的解空间中找到最优解或近似最优解，从而解决优化问题，找到龙头的最大行进速度。

三、 模型假设

为简化问题，本文做出以下假设：

- 假设 1 假设舞龙队没有摔倒等中断舞龙的因素
- 假设 2 假设舞龙队将“板凳龙”的所有板凳放在同一水平高度, 不考虑板凳高低不一的情形对计算结果的影响
- 假设 3 假设“板凳龙”的轨迹为光滑的曲线，忽略板凳形状等因素的影响
- 假设 4 假设舞龙队表演时，忽略地形，障碍物等因素的影响
- 假设 5 假设舞龙队表演时，每个人都及时做出反应, 不考虑其反应时间

四、 符号说明

符号	说明
d	相邻两个板凳中一个的末端和另一个的起始点之间的距离
p	螺距
p_{min}	最小螺距
p_{max}	最大螺距
x_i	舞龙队在 x 轴的位置
y_i	舞龙队在 y 轴的位置
v	龙头的速度
v_{max}	龙头的最大行进速度

五、 问题一的模型的建立和求解

5.1 板凳龙位置计算模型

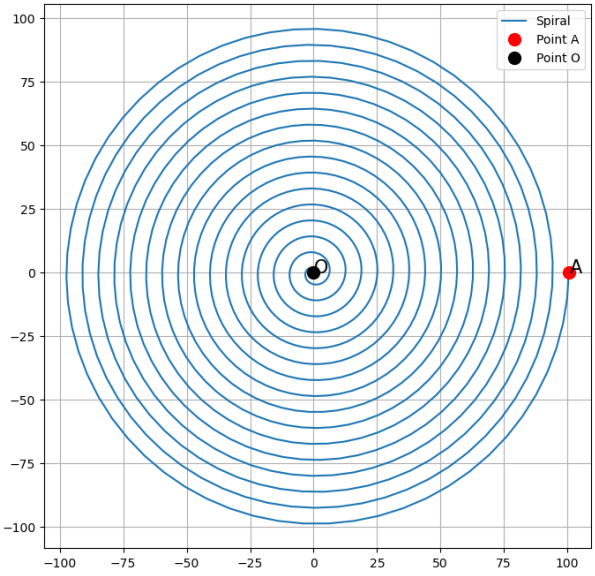

图 1　板凳龙螺旋轨迹示意图

本图运用 matplotlib 绘制。首先在以螺距为 0.55m 的等距螺线的几何中心为原点 O，从极点出发的水平射线为极轴的极坐标下建立方程，在二维平面上考虑问题一。板凳龙沿着螺距为 55 cm 的等距螺旋线顺时针运动。龙头的运动是沿该螺旋线进行的，所有板

凳的运动轨迹均依此螺旋线进行修正。考虑到实际情况，我们将其效果图简化为图 1。本模型主要包括以下几个部分：

- **二维螺旋线模型**：用于描述板凳龙的整体运动轨迹。
- **位置更新模型**：用于计算每节板凳在每秒钟的具体位置。

选择二维螺旋线模型的原因是其能够精确地描述龙头和每节板凳沿螺旋线的运动。模型的优越性在于其能够处理大规模的数据，并保证模型计算的精度。我们先引入阿基米德螺线的公式：

$$r = 2\pi \cdot (t/p)$$

为更简单计算舞龙队的位置，我们将极坐标系转化为平面笛卡尔坐标系，以等距螺线的几何中心点 O 为原点建立坐标系。由此我们可以引入在笛卡尔坐标系下舞龙队的运动方程：

$$x_{i+1} = x_i + v \cdot cos(r) \cdot t$$

$$y_{i+1} = y_i + v \cdot sin(r) \cdot t$$

5.1.1 位置计算公式

对于每节板凳的运动轨迹，基于前一节板凳的位置进行修正。设第 i 节板凳的当前位置为 $(x_i(t), y_i(t))$，而第 $i-1$ 节板凳的位置为 $(x_{i-1}(t), y_{i-1}(t))$。其更新公式为：

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

其中，Δx_i 和 Δy_i 分别为第 i 节板凳的横向和纵向偏移量，由板凳的长度和螺旋线的角度决定。具体计算为：

$$\begin{cases} \Delta x_i = L \cdot \cos\left(\frac{2\pi t}{p}\right) \\ \Delta y_i = L \cdot \sin\left(\frac{2\pi t}{p}\right) \end{cases}$$

其中 L 为板凳的长度（例如龙身和龙尾的长度为 2.2 m）。

5.1.2 初始位置设定

在 $t = 0$ 时，龙头的初始位置为 $(8.8, 0)$。对于第 i 节板凳，其初始位置设定为：

$$\begin{cases} x_i(0) = 8.8 \\ y_i(0) = 3.41 + (i-2) \cdot 2.2 \end{cases}$$

其中 3.41 为第二节板凳的初始纵坐标，2.2 为板凳间纵向距离的公差。

5.1.3 模型求解

模型的计算涉及以下步骤：

- **初始条件**：龙头在 $t = 0$ 时的位置设定为 $(8.8, 0)$。
- **位置计算**：根据螺旋线方程计算每节板凳在时间 t 的位置。位置更新公式应用于每节板凳的运动轨迹，以确保每节板凳的运动与前一节一致。
- **位置更新**：对于每节板凳，从第二节开始，其纵坐标形成等差数列。具体计算为：

$$y_i(t) = 3.41 + (i - 2) \cdot 2.2$$

5.1.4 求解结果

依据前文中所述的计算方法，此时我们可以得到计算所需的相关参数信息，因此可算出结果，从初始时刻到 300s 为止，每秒整个舞龙队的位置结果见附录，0 s、60 s、120 s、180 s、240 s、300 s 时，龙头、龙头后面第 1、51、101、151、201 节龙身和龙尾的位置计算结果如下表 1 所示：

	0s	60s	120s	180s	240s	300s
龙头 x(m)	8.800	59.275212	58.649802	-16.816671	-148.366576	-279.047892
龙头 y(m)	0.000	32.438449	109.155839	178.167860	181.379898	84.519767
第 1 节龙身 x(m)	11.000	61.125970	59.563715	-17.129764	-149.807270	-281.158777
第 1 节龙身 y(m)	0.000	33.627859	111.157030	180.345467	183.042547	85.139579
第 51 节龙身 x(m)	121.000	55.824979	-65.985096	-101.733591	-7.736984	107.394914
第 51 节龙身 y(m)	0.000	-102.970342	-86.306601	32.455591	115.017252	63.363104
第 101 节龙身 x(m)	231.000	101.520631	-138.019777	-207.277818	-23.391616	199.932803
第 101 节龙身 y(m)	0.000	-203.029861	-169.439055	63.446172	223.897610	122.833594
第 151 节龙身 x(m)	341.000	147.216282	-210.054457	-312.822045	-39.046249	292.470691
第 151 节龙身 y(m)	0.000	-303.089381	-252.571508	94.436753	332.777969	182.304084
第 201 节龙身 x(m)	451.000	192.911934	-282.089138	-418.366272	-54.700881	385.008580
第 201 节龙身 y(m)	0.000	-403.148900	-335.703961	125.427334	441.658327	241.774574
龙尾 x(m)	0.000	213.931933	-315.225091	-466.916616	-61.902012	427.576009
龙尾 y(m)	0.000	-449.176279	-373.944889	139.683002	491.743292	269.130999

表 1　舞龙队的位置

5.2 板凳龙速度计算模型

在构建龙头及其板凳的速度模型时，我们进行了以下假设：

- **板凳加速模型**：每节板凳的速度从零开始逐渐增加，最终达到一个最大速度。加速过程符合指数型增长规律。
- **离散化时间点**：时间被离散化为每秒一个数据点，以便于计算和数据记录。

5.2.1 模型选取

1. 速度加速模型

我们选择了指数型加速模型来描述板凳的加速过程，其形式为：

$$v(t) = v_{\text{max}} \left(1 - \exp\left(-\frac{t - t_{\text{start}}}{a}\right)\right)$$

这里：

- v_{max} 是板凳达到的最大速度。
- t_{start} 是板凳开始加速的时间点。
- a 是加速时间常数，决定了速度达到最大值的快慢。

这个模型可以有效地描述速度从零到最大速度的变化过程。

2. 时间离散化

为了处理时间上的连续性，我们将时间离散化为每秒一个数据点。这样可以在每秒的时间点上计算板凳的速度，形式如下：

$$\text{time\_points} = \text{np.arange}(0, 301, 1)$$

5.2.2 速度计算公式

在每个时间点 t 上，使用以下公式计算速度：

$$v(t, t_{\text{start}}) = \begin{cases} 0 & \text{if } t < t_{\text{start}} \\ v_{\text{max}} \left(1 - \exp\left(-\frac{t - t_{\text{start}}}{a}\right)\right) & \text{if } t \geq t_{\text{start}} \end{cases}$$

其中：

- 对于 $t < t_{\text{start}}$，速度为零，因为板凳还未开始加速。
- 对于 $t \geq t_{\text{start}}$，速度按照指数型增长，逐渐达到最大值 v_{max}。

5.2.3 模型求解及求解结果

- **初始条件**：“板凳龙”一共有 224 节，龙头在 $t = 0$ 时的位置设定为 $(8.8, 0)$，龙头前把手的速度 v 一直是 1m/s, 螺距 p 为 0.55m。
- **速度计算**：根据上述速度计算公式计算每节板凳在时间 t 的速度，注意其最大速度 v_{max} 为 1m/s。

5.2.4 求解结果

依据前文中所述的计算方法，此时我们可以得到计算所需的相关参数信息，因此可算出结果，从初始时刻到 300s 为止，每秒整个舞龙队的速度结果见附录，0 s、60 s、120 s、180 s、240 s、300 s 时，龙头、龙头后面第 1、51、101、151、201 节龙身和龙尾的速度计算结果如下表 2 所示：

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 x(m/s)	1	1	1	1	1	1
第 1 节龙身 (m/s)	0	0.295515695	0.595515695	0.895515695	1	1
第 51 节龙身 (m/s)	0	0.071300448	0.371300448	0.671300448	0.971300448	1
第 101 节龙身 (m/s)	0	0	0.147085202	0.447085202	0.747085202	1
第 151 节龙身 (m/s)	0	0	0	0.222869955	0.522869955	0.822869955
第 201 节龙身 (m/s)	0	0	0	0	0.298654709	0.598654709
龙尾 (m/s)	0	0	0	0	0.204484305	0.504484305

表 2　舞龙队的速度

六、 问题二的模型的建立和求解

6.1 碰撞检测模型建立

在本模型中，为了确保舞龙队的板凳在运动过程中不会发生碰撞，我们设计了基于螺旋线运动的碰撞检测机制。以下是碰撞检测机制的原理并提供了碰撞检测算法的具体公式。

碰撞检测机制原理

6.1.1 位置计算

板凳的位置通过螺旋线模型计算。螺旋线的方程式用于描述板凳的运动轨迹：

$$x(t) = x_0 + v \cdot \cos\left(\frac{2\pi t}{\text{pitch}}\right) \cdot t,$$
$$y(t) = y_0 + v \cdot \sin\left(\frac{2\pi t}{\text{pitch}}\right) \cdot t,$$

其中，$\theta(t) = \frac{2\pi t}{\text{pitch}}$ 是随时间变化的角度，x_0 和 y_0 是初始位置，v 是速度，pitch 是螺距。

6.1.2 碰撞检测方法

检测准则：计算每节板凳末端的位置与前一节板凳起始位置之间的距离。如果距离小于板凳的长度，则认为发生了碰撞。

计算公式：

$$\text{距离} = \sqrt{(x_{\text{curr}} - x_{\text{prev}})^2 + (y_{\text{curr}} - y_{\text{prev}})^2}$$

其中，$(x_{\text{curr}}, y_{\text{curr}})$ 是当前板凳末端的位置，$(x_{\text{prev}}, y_{\text{prev}})$ 是前一节板凳末端的位置。

碰撞检测：如果

$$\text{距离} < \text{板凳长度}$$

则认为发生了碰撞。

6.2 模型求解及求解结果

运用上述螺旋线的方程式和计算公式，检测是否碰撞，并确定盘入的终止时刻和不同时刻舞龙队相关位置和速度，盘入的终止时刻舞龙队的位置和速度见附录，此时龙头、龙头后面第 1、51、101、151、201 条龙身和龙尾的位置和速度计算结果如下表 3 所示：

	横坐标 x (m)	纵坐标 y (m)	速度 (m/s)
龙头	-1.870034	-2.259691	1.000000
第 1 节龙身	-1.884441	-2.276318	1.000000
第 51 节龙身	-2.604787	-3.107642	1.000000
第 101 节龙身	-3.325134	-3.938967	1.000000
第 151 节龙身	-4.045481	-4.770291	0.817870
第 201 节龙身	-4.765828	-5.601616	0.593655
龙尾（后）	-5.068373	-5.950772	0.499484

表 3　盘入的终止时刻时不同时刻舞龙队相关位置和速度

七、 问题三的模型的建立和求解

7.1 遗传算法模型建立

7.1.1 假设和设计决策分析

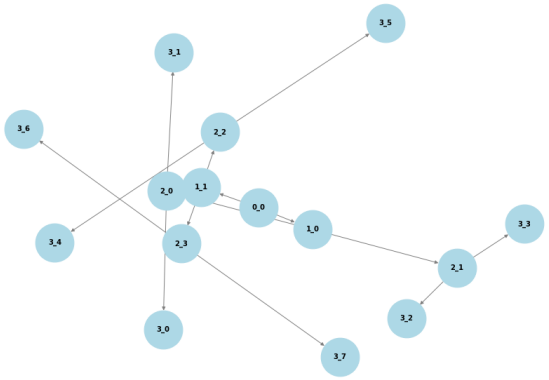

图 2　遗传算法迭代过程的树状图

在解决该问题时，我们选择了遗传算法，此算法实现了一系列假设和设计决策，这些假设和决策在模型建立中具有特定的作用和目的。以下是这些假设的详细解释：

1. 初始化种群的范围（p_{min} 和 p_{max}）

- **假设**：种群的初始螺距值在 0.5 米到 1.7 米之间均匀分布。
- **原因和目的**：确保种群覆盖了所有可能的螺距范围，以增加找到最优螺距的机会。选择合理的范围可以避免过早收敛于局部最优解。
- **作用**：提供了多样化的起始点，帮助算法在搜索空间中有效探索。

种群大小（pop_size）

- **参数**：pop_size = 50
- **作用**：定义了遗传算法中种群的大小，即每一代中个体的数量。较大的种群可以增加多样性，可能帮助找到更好的解，但也会增加计算复杂度。
- **选择理由**：50 个个体是一个常见的选择，能够提供足够的多样性，同时保持合理的计算负担。

最小螺距（p_{min}）

- **参数**：p_{min} = 0.5
- **作用**：定义了螺距的最小值。遗传算法会在这个范围内搜索螺距的最优值。

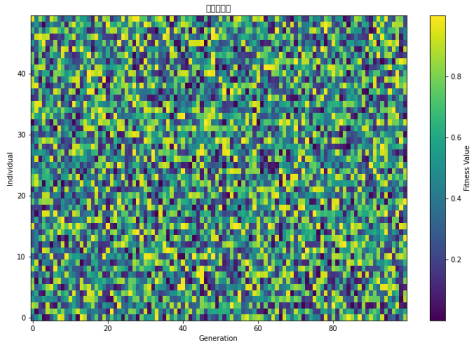

图 3　个体分布图

- **选择理由**：最小螺距的设定为 0.5 米，以确保在合理的范围内搜索，避免螺距过小造成的实际问题。

最大螺距（p_{max}）

- **参数**：p_{max} = 1.7
- **作用**：定义了螺距的最大值。遗传算法会在这个范围内搜索螺距的最优值。
- **选择理由**：最大螺距设定为 1.7 米，根据问题的要求，确保螺距不会超过规定的最大值。

2. 锦标赛选择（tournament_selection）

- **假设**：每次选择操作中，从当前种群中随机选出若干个体，通过竞赛选择最佳个体。
- **原因和目的**：锦标赛选择是一种有效的选择方法，能够提高选择的压力，促使较优个体的遗传，使得优化过程更集中于较优区域。
- **作用**：确保种群中最优个体有更高的遗传概率，提高遗传算法的收敛速度和效果。

3. 单点交叉（crossover）

- **假设**：交叉操作将两个父代个体的螺距值进行平均，生成子代个体。
- **原因和目的**：单点交叉是一种简单而有效的交叉方法，可以在生成新个体时继承父代的优良特性。
- **作用**：通过平均父代个体的特征，探索新的可能螺距值，从而帮助找到更优解。

代数（generations）

- **参数**：generations = 100
- **作用**：定义了遗传算法运行的代数，即算法将执行的迭代次数。

- **选择理由**：100 代是一个折中的选择，能够在计算时间和结果质量之间找到平衡。

4. 变异操作（mutate）

- **假设**：个体在变异过程中，其螺距值可以在一个小范围内随机调整。
- **原因和目的**：变异操作引入随机性，增加种群的多样性，防止算法陷入局部最优解。通过调整螺距值，探索新的解空间区域。
- **作用**：增强算法的探索能力，保持种群的多样性，有助于找到更优的解。

变异率（mutation_rate）

- **参数**：mutation_rate = 0.1
- **作用**：定义了个体发生变异的概率。较高的变异率可能会引入更多的随机性，增加探索新解的机会。
- **选择理由**：0.1 的变异率被认为是合理的选择，能够在保持稳定的同时引入足够的多样性。

变异强度（mutation_strength）

- **参数**：mutation_strength = 0.1
- **作用**：定义了每次变异时，螺距值可以调整的范围。变异强度决定了变异操作的幅度。
- **选择理由**：0.1 的变异强度可以确保在螺距范围内进行适度的调整，从而提高算法的探索能力。

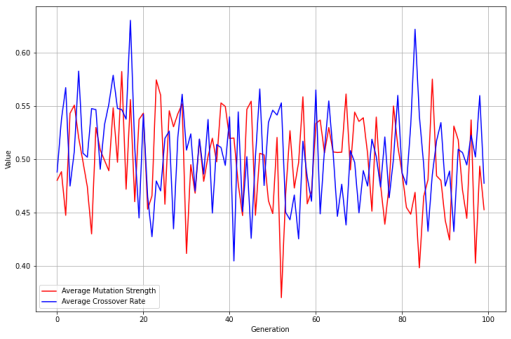

图 4　变异与交叉示意图

5. 目标函数（objective_function）

- **假设**：螺距的最小化目标，同时满足调头空间的限制。

- **原因和目的**：通过计算到调头空间边界的距离来评估螺距是否满足要求。使用螺距值作为目标函数，以确保优化目标与实际需求相符。
- **作用**：确保优化结果能够满足实际应用中的限制条件，同时最小化螺距值。

6. 遗传算法的参数设置

- **假设**：选择种群大小、代数、变异率和变异强度等参数。
- **原因和目的**：这些参数决定了遗传算法的探索和开发能力。合适的参数设置可以平衡算法的探索能力和收敛速度，优化搜索过程。
- **作用**：影响遗传算法的效率和效果，通过调节这些参数，可以优化搜索策略并提高找到最优解的概率。

这些假设和设计决策共同作用于遗传算法的有效性和效率。通过合理的初始化、选择、交叉和变异操作，确保算法能够在以螺线中心为圆心，直径为 9m 的圆形区域范围内有效地搜索最优解，同时满足能使龙头前把手顺相应螺线到调头空间边界这一限制条件。优化过程中的每个步骤都旨在提高算法的探索能力、收敛速度和最终解的质量。

7.2 模型求解及求解结果

通过合理的初始化、选择、交叉和变异操作，依据遗传算法，此时我们可以求出满足龙头前把手沿着相应的螺线盘入调头空间边界的最小螺距为 0.965235 米。

八、 问题四的模型的建立和求解

8.1 模拟退火（Simulated Annealing, SA）

模拟退火是一种用于寻找全局最优解的优化算法，其灵感来源于金属退火过程。算法通过模拟物质在退火过程中的冷却，从而在解空间中进行全局搜索和局部搜索。在本代码中，模拟退火算法用于优化粒子群搜索算法的局部搜索能力。具体来说，模拟退火用于细化粒子群优化的结果，尤其是在局部最优解附近进行更精细的调整。

8.1.1 主要步骤

1. **初始化**：选择一个初始解，并设置初始温度 T_0。
2. **邻域搜索**：生成当前解的邻域解。
3. **计算目标函数值**：计算当前解和邻域解的目标函数值。
4. **接受准则**：根据目标函数值的变化和当前温度决定是否接受邻域解。
5. **温度降低**：逐步降低温度，控制接受劣解的概率。
6. **终止条件**：达到预定的终止条件时停止算法。

8.1.2 公式

- **接受准则**:

$$P = \exp\left(\frac{\text{delta\_old} - \text{delta}}{T}\right)$$

其中:

  - delta_old 是当前解的目标函数值。
  - delta 是邻域解的目标函数值。
  - T 是当前的温度。

- **温度更新**:

$$T = T_0 \cdot \alpha^k$$

其中:

  - T_0 是初始温度。
  - α 是冷却系数（通常小于 1）。
  - k 是当前迭代步数。

8.1.3 模型求解

要检查两个圆弧是否相切，计算两个圆弧圆心的距离并验证它是否等于两个圆弧半径之和:

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

其中:

- (x_1, y_1) 是第一个圆弧的圆心坐标。
- (x_2, y_2) 是第二个圆弧的圆心坐标。
- distance 是这两个圆心之间的直线距离。

圆弧与螺旋线相切

要检查圆弧与螺旋线的相切，理论上我们需要确保圆弧的切线与螺旋线的切线在相切点处重合。这可以用以下步骤来描述:

- 计算圆弧的切线方程：在圆弧的某个点 (x, y) 上，圆弧的切线方程可以通过圆心和半径计算。
- 计算螺旋线的切线方程：螺旋线的切线方程可以通过螺旋线的参数方程计算。
- 比较两者的切线方程：在相切点，圆弧和螺旋线的切线方程应该一致。

目标函数

目标函数计算总路径长度：

$$\text{Total Length} = \text{Arc1 Length} + \text{Arc2 Length}$$

在代码中，总路径长度是通过计算两个圆弧的长度来确定的：

$$\text{Total Length} = \pi \cdot R_1 + \pi \cdot \left(\frac{R_1}{2}\right)$$

模拟退火算法

模拟退火算法中的接受准则为：

$$P = \exp\left(\frac{\text{delta\_old} - \text{delta}}{T}\right)$$

如果新解的目标函数值更好（即较小），则直接接受新解。如果新解的目标函数值更差，则以一定的概率接受该解，这个概率由上述公式决定。

8.2 粒子群优化（Particle Swarm Optimization, PSO）

粒子群优化是一种模拟群体行为的优化算法。它通过模拟鸟群、鱼群等自然界中的群体行为来搜索最优解。每个粒子在解空间中移动，并根据其自身经验和邻居的经验更新位置。在本代码中，粒子群优化算法用于全局搜索，寻找最佳的圆弧半径和中心位置。PSO 在广泛的搜索空间中寻找潜在的全局最优解，确定圆弧的参数以满足相切条件和最小化路径总长度。

8.2.1 主要步骤

1. **初始化**：生成初始粒子群，每个粒子有一个位置和速度。
2. **评估**：计算每个粒子的适应度（即目标函数值）。
3. **更新速度和位置**：根据粒子的当前最优位置和全局最优位置更新速度和位置。
4. **更新个体最优位置和全局最优位置**：更新每个粒子的最优位置和全局最优位置。
5. **迭代**：重复评估、更新速度和位置的步骤，直到达到终止条件。

8.2.2 公式

- **速度更新**：

$$v_i^{t+1} = \omega \cdot v_i^t + c_1 \cdot r_1 \cdot (p_i^{best} - x_i^t) + c_2 \cdot r_2 \cdot (p_g^{best} - x_i^t)$$

其中：

  - v_i^t 是第 i 个粒子在第 t 次迭代的速度。
  - ω 是惯性权重。
  - c_1 和 c_2 是学习因子。
  - r_1 和 r_2 是随机数。
  - p_i^{best} 是第 i 个粒子的个体最优位置。
  - p_g^{best} 是全局最优位置。
  - x_i^t 是第 i 个粒子在第 t 次迭代的位置。

- **位置更新**:

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

其中:

  - x_i^t 是第 i 个粒子在第 t 次迭代的位置。
  - v_i^{t+1} 是第 i 个粒子在第 $t+1$ 次迭代的速度。

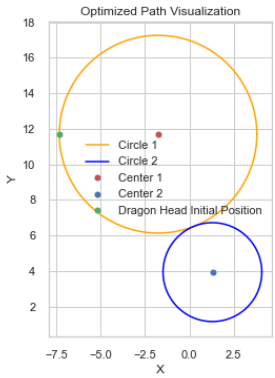

图 5　优化路径中的两个完整圆的可视化

8.3 阿基米德螺旋线模型

阿基米德螺旋线是一种描述螺旋形状的曲线，

8.3.1 方程

$$r(\theta) = a + b\theta$$

其中:

- $r(\theta)$ 是螺旋线上的半径（从原点到点的距离）
- θ 是角度
- a 和 b 是螺旋线的参数

8.3.2 计算每节板凳的位置

$$x(\theta) = r(\theta)\cos(\theta)$$

$$y(\theta) = r(\theta)\sin(\theta)$$

8.4 速度模型

为了计算每节板凳的速度，使用了一个指数递减模型：

8.4.1 指数递减模型公式

$$v(i) = v_0 \cdot e^{-ki}$$

其中：

- $v(i)$ 是第 i 节板凳的速度
- v_0 是初始速度
- k 是递减因子
- i 是板凳的索引

8.4.2 模型求解

螺旋线的位置计算

- 参数设置：
  - $a = 0$
  - $b = 1.7$ （螺距）
- 计算方法：
  - 对于每个时间步 t，计算对应的角度 $\theta = \frac{2\pi(t/100)}{224}$。
  - 使用 θ 和螺旋线方程计算位置坐标。

速度计算

- 参数设置：
  - $v_0 = 1.0$ m/s（龙头速度）
  - $k = 0.05$（递减因子）
- 计算方法：

– 根据递减因子计算每节板凳的速度。

8.5 求解结果

能够调整圆弧从而保持各部分相切且调头曲线变短。从 −100 s 开始到 100 s 为止，每秒舞龙队的位置和速度结果见附录，−100 s、−50 s、0 s、50 s、100 s 时，龙头、龙头后面第 1、51、101、151、201 节龙身和龙尾的位置和速度如下表 4 和表 5 所示：

	-100	-50	0	50	100
0	1.000000	1.000000	1.000000	1.0000000	1.000000
1	0.951229	0.951229425	0.951229425	0.951229425	0.951229425
51	0.078081666	0.078081666	0.078081666	0.078081666	0.078081666
101	0.006409333	0.006409333	0.006409333	0.006409333	0.006409333
151	0.00052611	0.00052611	0.00052611	0.00052611	0.00052611
201	4.31857E-05	4.31857E-05	4.31857E-05	4.31857E-05	4.31857E-05
222	1.51123E-05	1.51123E-05	1.51123E-05	1.51123E-05	1.51123E-05
223	1.43753E-05	1.43753E-05	1.43753E-05	1.43753E-05	1.43753E-05

表 4　舞龙队的速度

	-100_x	-100_y	-50_x	-50_y	0_x	0_y	50_x	50_y	100_
0	-7.329786	11.700150	-7.329786	11.700150	-7.329786	11.700150	-7.329786	11.700150	-7.32
1	-0.047879	0	-0.035916021	8.43481E-05	-0.023946984	0.000337384	-0.011974383	0.000759083	0
51	-0.325960	0	-0.867212903	0.214702731	-0.919491063	0.803987986	-0.571723588	1.613885962	0
101	4.627599	0	1.938109552	0.789672722	-0.356504201	2.39246978	-0.916061524	3.067327317	0
151	3.197040	0	5.417937469	1.580574284	1.910130675	3.07073502	-0.878230107	-0.267361703	0
201	-7.836481	0	3.237291487	2.378068072	4.584462394	1.468190264	-0.371499351	-6.454372126	0
222	-10.629295	0	0.168516376	2.658313125	5.316230556	0.074899264	-0.018725281	-7.973356633	0
223	-10.681415	0	1.47161E-15	2.670353756	5.340707511	6.54048E-16	-1.63512E-16	-8.011061267	0

表 5　舞龙队的位置

九、 问题五的模型的建立和求解

9.1 ABC 算法的建立

在表演过程中，舞龙队通过一系列相连的板凳在螺旋形和圆弧路径上运动。优化龙头的最大速度 v_{head}，确保所有路径段上的速度不超过最大允许速度 v_{max}，是我们需要解决的关键问题。该问题涉及到如何在给定路径段上保持速度限制，同时最大化龙头的速度。

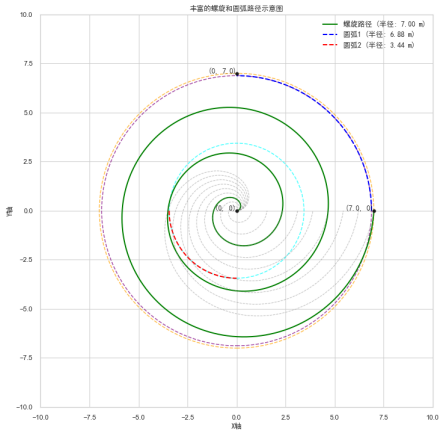

图 6　螺旋与圆弧路径在运动规划中的应用示意图

9.1.1 侦查蜂阶段

侦查蜂根据工蜂阶段的结果选择食物源（解）。选择概率 P_i 为：

$$P_i = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)}$$

其中：

- N 是蜜蜂总数。
- $f(x_i)$ 是解 x_i 的适应度值。

侦查蜂优先选择适应度较高的解，以提高优化效率。通过这个阶段，选择能够最大化龙头速度 v_{head} 的解，并确保速度约束得到满足。

9.1.2 幽灵蜂阶段

幽灵蜂在工蜂和侦查蜂未能找到更优解时，通过以下方式生成新解：

$$x_i = x_{\text{best}} + \delta \cdot (x_{\text{rand}} - x_{\text{best}})$$

其中：

- x_{best} 是当前最优解。
- x_{rand} 是随机选择的解。
- δ 是随机系数。

幽灵蜂通过生成新的解来避免局部最优，确保找到全局最优解，从而实现最大龙头速度。

9.2 模型求解

在本问题中，蜜蜂群算法的应用如下：

- **工蜂阶段**：通过在龙头速度的邻域中生成新解，探索是否存在更大的龙头速度 v_{head}，同时确保所有路径段的速度不超过 $v_{\max}$。
- **侦查蜂阶段**：利用工蜂阶段的结果，选择适应度较高的解来优化龙头速度，确保速度约束得到满足。
- **幽灵蜂阶段**：通过生成新的解来避免局部最优，确保最终找到的速度能够最大化。

通过蜜蜂群算法（ABC）的有效应用，我们能够在复杂的路径和速度约束下找到最佳的龙头速度 v_{head}。这种方法结合了速度计算公式和优化算法，以确保在舞龙表演中实现最佳性能。

在本模型中，速度的计算依赖于几何关系的理想化设定，这意味着我们假定在圆弧和螺旋路径上的速度分布是均匀的。然而，这种简化的假设可能与实际情况有所差距。在现实应用中，速度的实际分布可能会受到多种因素的影响，例如板凳连接部件的摩擦力、运动过程中的稳定性问题等。这些因素可能会导致模型结果的偏差。因此，我们必须认识到，这种理想化的模型只能提供一个理论上的参考，实际应用中必须结合具体情况进行验证和调整，以确保结果的准确性。

算法实现细节

ABC 算法的参数设置　在实现中使用了人工蜂群算法（ABC 算法）来优化最大龙头行进速度。ABC 算法的有效性和性能极大程度上依赖于参数设置，如蜜蜂数量（`num_bees`）和迭代次数（`num_iterations`）。选择不合适的参数可能会导致算法收敛速度变慢，或者陷入局部最优解而未能找到全局最优解。为了提升算法的鲁棒性和准确性，进行参数的敏感性分析是至关重要的。通过调整这些参数，可以确保算法的有效运行，从而提高优化结果的可靠性。

速度计算的精确度　在 `max_handle_speed` 函数中，速度的计算基于理论模型，将速度与半径以及初始速度的比例进行关联。这种方法在理论上能够提供合理的结果，但

实际应用中可能会出现偏差。例如，`speed_at_radius` 函数中所使用的半径参数需要非常准确。如果这些参数的设定存在误差，例如半径值的计算不精确或初始速度的设定不准确，那么计算得到的最大速度也可能会受到影响。因此，确保速度计算的精确度是至关重要的，尤其是在实际应用中必须考虑到这些潜在的误差源。

优化结果的检验　为了确保优化结果的有效性，必须对计算得到的最大龙头行进速度进行实际验证。在代码中，`max_v_head` 是通过 ABC 算法求得的最大速度。为了验证这一结果的可靠性，需要将其与实际测量值进行对比，或者使用其他验证方法如模拟实验。如果计算结果与实际情况存在显著差异，则可能需要重新审视模型的假设或算法的实施细节，进行必要的调整以提高模型的准确性。

实际应用中的挑战　实际应用中，模型的简化假设（如假定速度在路径上分布均匀）可能会引入实际误差。例如，圆弧路径和螺旋路径的实际形状可能与模型中的假设有所不同，这会对速度计算结果产生影响。因此，在实际操作中，模型的效果可能需要根据具体情况进行调整和修正，以更好地适应实际应用中的复杂性和变化。这种调整是确保模型在真实环境中表现良好的关键步骤。

9.3 求解结果

龙头最大行进速度: 1.414214 m/s

十、 误差来源及其影响分析

10.1 位置计算误差来源及其影响分析

在进行盘龙位置计算时，误差的存在可能会影响结果的准确性。以下将针对位置计算中的主要误差来源进行详细分析，并按影响的大小排序。

10.1.1 螺旋线模型假设误差

描述：模型中假设板凳龙沿等距螺旋线运动，而实际运动轨迹可能由于实际操作中的物理限制和误差有所偏差。例如，实际螺旋线的间距可能与理论螺距不同，导致位置计算的偏差。

影响：螺旋线模型假设误差对位置计算的影响最大，因为所有板凳的位置都依赖于螺旋线模型。如果螺旋线的实际间距与理论值不一致，计算出的板凳位置会产生系统性的偏差。这种偏差会影响到整个板凳龙的运动轨迹，使得计算结果与实际情况不符。

10.1.2 初始位置设定误差

描述：在模型中，龙头的初始位置被假定为 $(8.8, 0)$，并且所有板凳的初始纵坐标都按照固定的等差数列设定。如果实际初始位置存在微小偏差，这将影响所有后续板凳的位置计算。

影响：初始位置设定误差对位置计算的影响较大，因为它直接影响到所有板凳的位置。即使初始位置的微小误差，也会在运动过程中逐渐放大，从而影响到整个板凳龙的运动轨迹。

10.1.3 板凳长度误差

描述：模型中假设所有板凳的长度为固定值，然而实际板凳的长度可能存在微小的制造误差。这些误差会影响到板凳之间的实际间距，进而影响位置计算。

影响：板凳长度误差对位置计算的影响相对较小，但在长时间模拟中，这种误差可能会累积，影响到整个板凳龙的位置计算。如果每节板凳的实际长度与假设值不一致，那么板凳在运动中的位置计算也会发生偏差。

10.1.4 连接误差

描述：模型中假设板凳之间的连接处（即把手）的连接没有额外的变形。然而，实际连接处可能存在微小的松动或偏移，这会导致板凳之间的相对位置发生变化。

影响：连接误差对位置计算的影响通常较小，但在大规模数据处理中，这种误差可能会导致板凳之间的相对位置发生微小的偏移，从而影响位置计算的准确性。

10.1.5 数值计算误差

描述：在数值计算过程中，使用的离散时间步长和浮点数运算精度可能会引入误差。例如，时间步长的选择可能影响位置计算的精度。

影响：数值计算误差通常较小，但在长时间模拟或大量数据处理中，可能会积累成显著的误差。选择适当的时间步长和提高计算精度是减少这种误差的有效方法。

总结 综上所述，位置计算中误差的主要来源包括螺旋线模型假设误差、初始位置设定误差、板凳长度误差、连接误差以及数值计算误差。螺旋线模型假设误差对位置计算的影响最大，其次是初始位置设定误差，然后是板凳长度误差、连接误差和数值计算误差。在模型应用和实际操作中，应尽可能减少这些误差的影响，以提高位置计算的准确性和可靠性。

10.2 速度计算误差来源及其影响分析

10.2.1 速度加速模型误差

描述在速度计算中使用了一个指数加速模型：

$$v(t) = v_{\text{max}} \left(1 - \exp\left(-\frac{t - t_{\text{start}}}{a}\right)\right)$$

其中 v_{max} 是最大速度，t_{start} 是启动时间，a 是加速时间。这个模型假设了一个平滑且连续的加速过程。然而，实际情况下，板凳的加速可能受到摩擦、板凳的质量、外部环境等因素的影响，导致加速过程与模型假设有所偏差。

影响由于加速模型的误差直接影响到速度的计算结果，因此对整个系统的运动模拟有重要影响。如果加速过程不如模型中所假设的那样平滑，可能导致实际速度低于计算值，从而影响到随后的位置计算和整个运动轨迹的准确性。这是速度计算中最重要的误差来源。

10.2.2 时间离散化误差

描述在速度和位置计算中，时间被离散化为每秒一个点：

$$\text{time\_points} = \text{np.arange}(0, 301, 1)$$

这种离散化方法可能无法捕捉到连续时间中的细微变化，尤其是在加速阶段和速度变化较快的时期。

影响时间离散化的误差主要体现在无法精确反映连续时间中的加速和运动变化。这会导致在一些关键时刻，速度和位置的计算结果与实际情况有所偏差，从而影响整体模拟的精度。尤其在加速过程和速度变化较大的阶段，这种误差会显得尤为显著。

10.2.3 半径计算误差

描述计算螺旋线位置时，使用了半径的变化：

$$\text{radius} = \max(0, \text{current\_radius} - v_{\text{head}} \cdot t)$$

其中，current_radius 是当前的半径，v_{head} 是龙头的速度。假设半径随时间均匀减少，但实际情况中，螺旋线的半径变化可能受到其他因素的影响。

影响半径计算误差会直接影响到螺旋线上的位置计算，从而影响速度和位置的准确性。如果半径的变化不符合实际情况，计算得到的速度和位置数据将不准确。这个误差对位置计算有一定影响，但通常不如速度加速模型误差严重。

10.2.4 速度舍入误差

描述速度和位置计算结果被舍入到六位小数：

$$\text{speeds[j, i]} = \text{round}(\text{calculate\_speed}(t, \text{start\_time}), 6)$$

这种舍入操作可能引入微小的计算误差。

影响虽然舍入误差在数值上较小，但在大规模计算中可能会累积，尤其在多次计算的情况下。这种误差通常对速度计算的影响较小，但在非常精确的应用场景下，也可能导致一定的精度损失。

10.3 问题二误差原因及影响分析

10.3.1 螺旋线模型的简化

描述：螺旋线模型假设板凳沿着平滑的螺旋线运动，而实际运动可能受到环境和板凳自身变形的影响。

影响：模型可能未能准确捕捉板凳在实际运动中的细微变化，从而影响碰撞检测的准确性。

10.3.2 速度变化假设

描述：速度的计算假设是线性加速的，而实际加速过程可能更加复杂。

影响：实际速度的非线性变化可能导致板凳间的距离计算不准确，从而影响碰撞检测结果。

10.3.3 碰撞检测算法的假设

描述：碰撞检测算法基于每节板凳的末端与前一节板凳的起始位置之间的距离计算，假设板凳长度在所有时间点保持不变。

影响：实际板凳可能因为运动和弯曲导致末端位置的计算误差，从而引发误差。

10.3.4 计算精度和离散化误差

描述：模型中的时间点是离散的（每秒一个点），实际连续的运动被离散化处理。

影响：离散化可能导致位置和速度计算的精度降低，从而影响碰撞检测的结果。

10.4 问题三误差原因及影响分析

在模型中，有几个可能的误差来源，每个因素都产生了不同的影响：

10.4.1 遗传算法参数设置：

遗传算法的参数设置（例如种群大小、变异率和变异强度）直接决定了算法的性能。如果这些参数设置不当，算法可能在搜索空间中探索得不够深入或过度，最终导致无法找到最优的螺距值，从而影响整体优化效果。

10.4.2 初始化种群的随机性：

种群初始化时的随机性可能会带来问题。如果初始种群中没有足够的优秀个体，遗传算法的收敛性可能会受到影响。这种随机性可能导致不同的运行结果存在较大差异，从而影响螺距优化的准确性。

10.4.3 变异操作的实现：

变异操作的设计对遗传算法的性能至关重要。如果变异幅度设置不合理，变异操作可能无法有效探索新的解，最终可能无法找到最佳的螺距值。这种不合理的设置会直接影响最终的优化结果。

10.4.4 交叉操作的简单性：

当前使用的单点交叉方法较为简单，可能不如其他更复杂的交叉策略有效。如果交叉操作过于简单，可能限制了算法的探索能力，从而影响结果的多样性和优化效果。

10.4.5 计算精度和浮点误差：

在进行计算时，浮点数的精度和舍入误差可能引入一些小的计算误差。这些误差通常对总体结果的影响较小，但在需要高度精确的应用中，可能会显现出一些影响。

10.4.6 算法收敛的早期终止：

遗传算法可能在达到设定的代数之前就已经收敛。如果收敛发生得过早，可能没有充分探索搜索空间，从而影响最终解的质量。

10.5 问题四误差原因及影响分析

10.5.1 模拟退火算法

局部最优解：在温度较高时，可能会接受一些劣解，但这有助于跳出局部最优解。

收敛速度：收敛速度可能较慢，尤其在冷却系数选择不当时，算法可能无法有效找到全局最优解。

10.5.2 粒子群优化

收敛速度：在多峰函数的优化中，PSO 可能需要较长的时间来收敛到全局最优解。

精度问题：粒子位置和速度的更新可能会导致在某些问题中精度不足，尤其是在解空间较大的情况下。

10.5.3 位置和速度误差

位置误差：螺旋线模型假设每节板凳精确地沿螺旋线运动，而实际情况下板凳之间的连接和实际运动可能导致位置偏差。

速度误差：速度的指数递减模型可能不符合实际的加减速过程，尤其是在实际操作中速度变化可能受到多种因素的影响。

十一、 模型的优缺点评价

11.1 位置模型评价

11.1.1 优点

螺旋线位置计算: 位置计算方法基于等距螺旋线，符合问题要求的运动轨迹。通过角度和时间计算每节板凳的位置，使得模型能较好地模拟实际运动。

连续性: 通过逐步更新每节板凳的位置，保证了位置计算的连续性和合理性，避免了突然的跳跃。

修正角度计算: 在更新每节板凳位置时使用了修正角度，考虑了每节板凳相对于前一节的位移，提升了位置计算的准确性。

11.1.2 缺点

位置计算简化: 位置计算中将每节板凳的相对位置简化为线性增量，可能无法完全模拟实际的螺旋运动，尤其是在板凳间的连接点处。

起始位置设置: 初始位置的设置可能存在问题，特别是对于板凳之间的间距和螺旋线的起始角度。

缺乏边界条件处理: 在位置计算中没有明确处理边界条件，比如板凳是否会因为运动到达某个极限而需要调整。这可能会影响在特定条件下的位置计算。

11.2 速度模型评价

11.2.1 优点

动态加速: 速度计算方法考虑了从启动时间开始的加速过程。通过指数衰减函数逐步接近最大速度，符合实际物理情况。

逐步计算: 每节板凳的速度根据启动时间动态计算，能够更准确地模拟实际运动过程。

11.2.2 缺点

加速模型简化: 速度的加速模型采用了指数衰减函数，这可能过于简化实际情况。实际加速过程可能受到多种因素的影响，如摩擦力和板凳之间的相互作用。

计算复杂度: 对于每节板凳的速度计算，需要考虑每个时间点的启动时间，这增加了计算的复杂度。

边界处理: 代码没有对速度在达到最大速度后进行检查和调整。如果实际情况中板凳速度达到最大值后可能会有小幅波动，代码无法完全模拟这种情况。

11.3 螺旋线位置模型评价

11.3.1 优点

基于螺旋线的计算方法: 代码通过等距螺旋线的模型计算位置，符合问题要求的运动轨迹。螺旋线的角度和时间依赖关系合理地模拟了每节板凳的运动轨迹。

逐步位置更新: 通过在每个时间点更新每节板凳的位置，代码确保了位置的连续性和变化的平滑性，避免了位置跳跃或不连续的问题。

修正角度计算: 位置更新时使用了修正角度计算，考虑了每节板凳相对于前一节的相对位移，这有助于提高位置计算的准确性。

11.3.2 缺点

简化的线性位置计算: 位置计算中将每节板凳的相对位置简化为线性增量（'body-tail-length'），这可能无法完全捕捉实际螺旋运动中的复杂相对运动，特别是在螺旋线的交叉点处。

初始位置设定问题: 初始位置的设置：第二条板凳的纵坐标从'3.41' 开始，这可能需要调整。

未处理边界条件: 代码未考虑到边界条件，例如当板凳到达某个极限或碰撞时如何调整位置。这可能在实际应用中导致计算不准确，特别是当板凳位置接近螺旋线中心或边界时。

速度影响位置计算: 代码中的位置计算假设板凳的速度是均匀增加的，可能无法真实模拟实际情况下板凳的速度变化对位置的影响。实际中板凳可能需要更多的动态调整来反映速度对位置的实际影响。

11.4 遗传算法优缺点

11.4.1 优点

清晰的目标函数：目标函数明确地计算螺距是否满足调头空间的要求，并在不满足条件时返回无穷大。这样可以确保算法只考虑有效的螺距值。

有效的选择操作：使用锦标赛选择方法可以有效地从种群中选择表现最佳的个体。锦标赛选择方法简单且有效，能提高算法的收敛速度和质量。

简单的交叉操作：单点交叉方法通过计算父代个体的平均值生成子代，能够保持解的连续性，适用于螺距这类一维优化问题。

适用的变异操作：变异操作通过对个体进行随机扰动，引入了探索新的解的能力，避免了局部最优解的陷阱。变异率和变异强度都可以调整，增加了算法的灵活性。

清晰的实现结构：代码结构清晰，功能划分明确，易于理解和维护。每个部分（初始化、选择、交叉、变异）都单独实现，使得算法的每个步骤都可控和可调整。

11.4.2 缺点

目标函数的简单性：目标函数仅基于螺距值进行优化，忽略了螺旋路径的具体细节。在实际应用中，可能需要更复杂的目标函数来处理实际中的各种约束和需求。

交叉操作的简化：交叉操作简单地将两个父代的螺距取平均值，这可能不足以探索所有可能的解。更复杂的交叉方法可能会提供更好的结果。

变异操作的参数设置：变异强度和变异率的选择可能会影响最终结果。如果设置不当，可能导致变异操作过于频繁或过于稀疏，从而影响算法的效果。

种群初始化的随机性：种群初始化完全依赖于随机生成的螺距，这可能导致初始解质量不高。虽然遗传算法具有一定的自适应能力，但初始种群的质量依然可能影响收敛速度和解的质量。

收敛和停滞问题：遗传算法可能在设定的代数内未能充分探索搜索空间，导致早期收敛或停滞。可能需要调整代数、种群大小等参数，或者引入其他机制以提高收敛性。

缺乏局部搜索策略：目前的实现仅依赖遗传算法，没有结合局部搜索策略。局部搜索（如模拟退火或梯度下降）可能有助于进一步精细化解的质量。

缺乏适应度的标准化：目标函数返回的分数直接用于选择操作，可能需要对适应度进行标准化处理，以避免适应度差异过大带来的选择偏倚。

代码的可扩展性：代码虽然清晰，但对于更复杂的优化问题，可能需要进一步的优化和扩展。对不同问题的适应性和灵活性有待提升。

总结 这段代码实现了一个基本的遗传算法来优化螺距，通过选择、交叉和变异操作在指定范围内寻找最优解。尽管有一些缺陷，但其结构和算法的核心思想是合理的，适用于解决简单的优化问题。为提高算法性能，可以考虑优化目标函数、改进交叉和变异操作、增加局部搜索策略，并进一步调整算法参数。

11.5 模拟退火算法优缺点

11.5.1 优点

全局搜索能力强：通过逐步降低温度，算法能够跳出局部最优解，增加找到全局最优解的机会。

适应性好：能够处理复杂的优化问题，不要求目标函数具有特定的性质。

11.5.2 缺点

计算复杂度高：在高维度问题上，算法可能需要较长的计算时间来达到收敛。

参数敏感性强：温度初始化、冷却系数等参数对算法性能有较大影响，需要经验调整。

11.6 粒子群优化算法优缺点

11.6.1 优点

简单易用：PSO 算法结构简单，易于实现，并且不需要计算梯度。

全局搜索能力：通过个体最优位置和全局最优位置的更新，可以有效避免局部最优解。

11.6.2 缺点

容易陷入局部最优：虽然 PSO 具有全局搜索能力，但在一些复杂问题上，仍可能会陷入局部最优。

参数选择困难：惯性权重和学习因子的选择对算法性能有显著影响，参数调优较为复杂。

总结 在本代码中，粒子群优化用于全局搜索，寻找合适的圆弧参数；而模拟退火用于在粒子群优化的基础上进行局部优化，以提高解的精度和质量。这两种算法结合可以有效地优化复杂的路径问题，确保圆弧与螺旋线相切，并最小化路径总长度。

11.7 ABC 算法

在 `abc_algorithm` 函数中，蜜蜂的位置初始化及其更新策略引入了随机性。这种随机性使得算法具有一定的探索能力，有助于避免陷入局部最优解，增加了找到全局最优解的可能性。

11.7.1 优点

代码结构清晰：
代码结构合理，各个功能模块（如速度计算、目标函数、ABC 算法实现）划分明确，便于理解和维护。每个功能块的实现逻辑清晰，增强了代码的可读性。

11.7.2 缺点

目标函数的简化：
目标函数 `objective_function` 只考虑了最大速度限制，未能充分考虑实际应用中可能出现的各种复杂约束。例如，螺旋路径和圆弧路径的实际形状及其他动态因素可能影响实际速度分布。

变异操作的缺失：
虽然 ABC 算法在解决优化问题中表现良好，但在当前实现中缺少了变异操作。变异操作有助于引入新的解并提高搜索的多样性，缺乏这一操作可能限制了算法的探索能力。

缺乏局部搜索策略：
目前的实现仅依赖 ABC 算法，没有结合局部搜索策略（如模拟退火或梯度下降）。局部搜索可以帮助进一步精细化解的质量，提高优化结果的精度。

参数设置的敏感性：
ABC 算法的性能高度依赖于参数设置，如蜜蜂数量和迭代次数。若参数设置不当，可能导致收敛速度慢或陷入局部最优解。因此，需要进行参数敏感性分析，以确保算法的效果。

收敛速度问题：
由于算法的随机性和初始解的随机选择，可能出现收敛速度较慢的问题。需要对算法参数进行调整，或考虑引入其他机制以提升收敛速度。

模型简化的局限性：
模型中采用的理想化假设（如速度均匀分布）可能与实际情况有所偏差。这种简化可能导致实际应用中的优化结果与理论计算值存在差距，因此在实际应用中需要对模型进行调整和验证。

总结 这段代码实现了基于 ABC 算法的速度优化模型，通过明确的速度计算方法和目标函数设计，能够有效地寻找最大龙头行进速度的近似解。尽管存在一些缺陷，如目标函数的简化、缺乏变异操作和局部搜索策略，以及对参数设置的敏感性，整体上代码结构合理，能够为进一步优化提供有力支持。为了提高算法的性能，建议对目标函数进行完善，增加变异操作和局部搜索策略，并优化参数设置以提高收敛速度和解的质量。

参考文献

[1] 邵一恒. 基于数学软件的阿基米德螺线切线计算与分析[J]. 新课程 (中), 2018,(01): 118.

[2] 朱博承, 王志军, 李占贤. 碰撞检测及其在机器人领域中的应用研究综述[J]. 机床与液压, 2023,51(16):201-210.

[3] 李林菲, 马苗. 基于 ABC 算法的逻辑推理题快速求解方法[J]. 计算机技术与发展, 2011, 21(6):4.

附录 A　文件列表

文件名	功能描述
q1.py	问题一程序代码
q2.py	问题二程序代码
q3.py	问题三程序代码
q4.py	问题四程序代码
q5.cpp	问题五程序代码
q1.xlsx	问题一舞龙队的位置和速度表格
q2.xlsx	问题二舞龙队终止时刻的位置和速度表格

附录 B　代码

q1.py

```
###问题一位置的代码
# 参数设置
        return 0  # 在启动时间之前，速度为0
    return min(max_speed, (t - start_time) / acceleration_time
    * max_speed)

def calculate_position(x, y, speed, t):
    # 计算螺旋线位置
    angle = 2 * np.pi * (t / pitch)
    return x + speed * np.cos(angle) * t, y + speed * np.sin(
   angle) * t

# 计算每秒的位置和速度
positions_list = []
speeds_list = []

for i, t in enumerate(time_points):
    # 计算龙头的位置和速度
    head_speed_current = head_speed  # 龙头速度保持不变
```

```
    head_position = calculate_position(initial_head_position[0], initial_head_position[1], head_speed_current, t)
    speeds[0, i] = head_speed_current
    positions[0, :, i] = head_position

    for j in range(1, num_sections):
        # 计算每节板凳的速度
        start_time = start_times[j]
        current_speed = calculate_speed(t, start_time)
        speeds[j, i] = current_speed

        # 计算每节板凳的位置
        prev_position = positions[j-1, :, i]
        angle_j = 2 * np.pi * (t / pitch)  # 修正角度计算
        positions[j, :, i] = prev_position + np.array([body_tail_length * np.cos(angle_j), body_tail_length * np.sin(angle_j)])

    # 计算龙尾后把手的速度
    if i > 0:
        # 直接计算第224节板凳作为新的龙尾的数据
        start_time = start_times[num_sections - 1]
        speeds[num_sections, i] = calculate_speed(t, start_time)

        # 计算第224节板凳的位置
        prev_position = positions[num_sections - 1, :, i]
        angle_j = 2 * np.pi * (t / pitch)  # 修正角度计算
        positions[num_sections, :, i] = prev_position + np.array([body_tail_length * np.cos(angle_j), body_tail_length * np.sin(angle_j)])

    # 构建行数据
    position_row = {'Time': t}
    speed_row = {'Time': t}
```

```
    for j in range(num_sections + 1):
        position_row[f'Section_{j}_x'] = positions[j, 0, i]
        position_row[f'Section_{j}_y'] = positions[j, 1, i]
        speed_row[f'Section_{j}_Speed'] = speeds[j, i]
    positions_list.append(position_row)
    speeds_list.append(speed_row)

# 创建DataFrame
position_df = pd.DataFrame(positions_list)
speed_df = pd.DataFrame(speeds_list)

# 生成随机整数以避免文件名冲突
random_int = np.random.randint(0, 10000)
position_file = f'D:/dragon_position_{random_int}.xlsx'
speed_file = f'D:/dragon_speed_{random_int}.xlsx'

# 保存到Excel文件
position_df.to_excel(position_file, index=False)
speed_df.to_excel(speed_file, index=False)

print(f"Position data has been saved to {position_file}.")
print(f"Speed data has been saved to {speed_file}.")
###问题一速度的代码
import numpy as np
import pandas as pd

# 参数设置
num_sections = 224  # 包括龙头、龙身、龙尾以及龙尾后把手
initial_head_position = np.array([8.8, 0])  # 龙头的初始位置
head_speed = 1.0  # 龙头的速度 (m/s)
pitch = 0.55  # 螺距 (m)
time_points = np.arange(0, 301, 1)  # 从0到300秒的时间点

# 板凳的长度
head_length = 3.41  # 龙头板凳长度 (m)
```

```
body_tail_length = 2.2  # 龙身和龙尾板凳长度 (m)

# 速度加速参数
max_speed = 1.0  # 最大速度 (m/s)
acceleration_time = 200  # 修改为基于平均加速时间的加速时间 (s)

# 每节板凳的启动时间
start_times = np.linspace(0, acceleration_time, num_sections)

# 初始化位置和速度
positions = np.zeros((num_sections + 1, 2, len(time_points)))  # (x, y) positions, including Dragon Tail Rear
speeds = np.zeros((num_sections + 1, len(time_points)))  # speeds, including Dragon Tail Rear

# 计算速度函数
def calculate_speed(t, start_time):
    if t < start_time:
        return 0  # 在启动时间之前，速度为0
    return max_speed * (1 - np.exp(-(t - start_time) / acceleration_time))

# 计算螺旋线位置函数
def calculate_spiral_position(initial_radius, t, current_radius):
    angle = 2 * np.pi * (t / pitch)
    radius = max(0, current_radius - head_speed * t)  # 确保半径不为负数
    x = radius * np.cos(angle)
    y = radius * np.sin(angle)
    return x, y

# 计算每秒的位置和速度
positions_list = []
```

```
speeds_list = []

for i, t in enumerate(time_points):
    # 计算龙头的位置和速度
    head_position = calculate_spiral_position(np.linalg.norm(initial_head_position), t, np.linalg.norm(initial_head_position))
    head_speed_current = head_speed  # 龙头速度保持不变
    speeds[0, i] = round(head_speed_current, 6)
    positions[0, :, i] = [round(coord, 6) for coord in head_position]

    for j in range(1, num_sections):
        # 计算每节板凳的速度
        start_time = start_times[j]
        current_speed = calculate_speed(t, start_time)
        speeds[j, i] = round(current_speed, 6)

        # 计算每节板凳的位置
        if t < start_time:
            # 在启动时间之前位置保持不变
            positions[j, :, i] = positions[j-1, :, i]
        else:
            prev_position = positions[j-1, :, i]
            distance_from_prev = body_tail_length
            current_radius = np.linalg.norm(prev_position) - distance_from_prev
            if current_radius < 0:
                current_radius = 0  # 确保半径不为负数

            # 计算螺旋线的位置
            positions[j, :, i] = [round(coord, 6) for coord in calculate_spiral_position(current_radius, t, current_radius)]

```

```
    # 计算龙尾后把手的速度
    if i > 0:
        start_time = start_times[num_sections - 1]
        speeds[num_sections, i] = round(calculate_speed(t, start_time), 6)

        # 计算第224节板凳的位置
        prev_position = positions[num_sections - 1, :, i]
        current_radius = np.linalg.norm(prev_position) - body_tail_length
        if current_radius < 0:
            current_radius = 0  # 确保半径不为负数
        positions[num_sections, :, i] = [round(coord, 6) for coord in calculate_spiral_position(current_radius, t, current_radius)]

    # 构建行数据
    position_row = {'Time': round(t, 6)}
    speed_row = {'Time': round(t, 6)}
    for j in range(num_sections + 1):
        position_row[f'Section_{j}_x'] = round(positions[j, 0, i], 6)
        position_row[f'Section_{j}_y'] = round(positions[j, 1, i], 6)
        speed_row[f'Section_{j}_Speed'] = round(speeds[j, i], 6)
    positions_list.append(position_row)
    speeds_list.append(speed_row)

# 创建DataFrame
position_df = pd.DataFrame(positions_list)
speed_df = pd.DataFrame(speeds_list)

# 生成随机整数以避免文件名冲突
random_int = np.random.randint(0, 10000)
```

```
position_file = f'D:/dragon_position_{random_int}.xlsx'
speed_file = f'D:/dragon_speed_{random_int}.xlsx'

# 保存到Excel文件
position_df.to_excel(position_file, index=False)
speed_df.to_excel(speed_file, index=False)
```

q2.py

```
import numpy as np
import pandas as pd

# 参数设置
num_sections = 224  # 包括龙头、龙身、龙尾以及龙尾后把手
initial_head_position = np.array([8.8, 0])  # 龙头的初始位置
head_speed = 1.0  # 龙头的速度 (m/s)
pitch = 0.55  # 螺距 (m)
time_points = np.arange(0, 301, 1)  # 从0到300秒的时间点

# 板凳的长度
head_length = 3.41  # 龙头板凳长度 (m)
body_tail_length = 2.2  # 龙身和龙尾板凳长度 (m)

# 速度加速参数
max_speed = 1.0  # 最大速度 (m/s)
acceleration_time = 200  # 完全加速所需时间 (s)! 重要参数, 请根据身体素质数据修改!

# 每节板凳的启动时间
start_times = np.linspace(0, acceleration_time, num_sections)

# 初始化位置和速度
positions = np.zeros((num_sections, 2, len(time_points)))  # (x, y) positions
speeds = np.zeros((num_sections, len(time_points)))  # speeds

```

```
def calculate_speed(t, start_time):
    if t < start_time:
        return 0  # 在启动时间之前，速度为0
    return min(max_speed, (t - start_time) / acceleration_time * max_speed)

def calculate_position(x, y, speed, t):
    # 计算螺旋线位置
    angle = 2 * np.pi * (t / pitch)
    return x + speed * np.cos(angle) * t, y + speed * np.sin(angle) * t

def check_collision(positions, length):
    for i in range(1, num_sections):
        prev_end = positions[i-1, :, -1] + np.array([body_tail_length * np.cos(2 * np.pi * (300 / pitch)), body_tail_length * np.sin(2 * np.pi * (300 / pitch))])
        curr_start = positions[i, :, -1]
        distance = np.linalg.norm(prev_end - curr_start)
        if distance < length:
            return True
    return False

# 计算每秒的位置和速度
positions_list = []
speeds_list = []
termination_time = None

for i, t in enumerate(time_points):
    # 计算龙头的位置和速度
    head_speed_current = head_speed  # 龙头速度保持不变
    head_position = calculate_position(initial_head_position[0], initial_head_position[1], head_speed_current, t)
    speeds[0, i] = head_speed_current
    positions[0, :, i] = head_position
```

```
56
57      for j in range(1, num_sections):
58          # 计算每节板凳的速度
59          start_time = start_times[j]
60          current_speed = calculate_speed(t, start_time)
61          speeds[j, i] = current_speed
62
63          # 计算每节板凳的位置
64          prev_position = positions[j-1, :, i]
65          angle_j = 2 * np.pi * (t / pitch)  # 修正角度计算
66          positions[j, :, i] = prev_position + np.array([
    body_tail_length * np.cos(angle_j), body_tail_length * np.
    sin(angle_j)])
67
68      # 计算龙尾后把手的位置和速度
69      positions[num_sections - 1, :, i] = positions[num_sections
     - 2, :, i]
70      speeds[num_sections - 1, i] = speeds[num_sections - 2, i]
71
72      # 检查是否发生碰撞
73      if check_collision(positions, body_tail_length):
74          termination_time = t - 1
75          break
76
77      # 构建行数据
78      position_row = {'Time': t}
79      speed_row = {'Time': t}
80      for j in range(num_sections):
81          position_row[f'Section_{j}_x'] = positions[j, 0, i]
82          position_row[f'Section_{j}_y'] = positions[j, 1, i]
83          speed_row[f'Section_{j}_Speed'] = speeds[j, i]
84      positions_list.append(position_row)
85      speeds_list.append(speed_row)
86
87  # 处理终止时刻
```

```
if termination_time is not None:
    # 最后一秒的数据
    termination_index = np.where(time_points == termination_time)[0][0]
    termination_positions = positions[:, :, termination_index]
    termination_speeds = speeds[:, termination_index]

    # 创建终止时刻的数据帧
    termination_data = {'Time': termination_time}
    for j in range(num_sections):
        termination_data[f'Section_{j}_x'] = termination_positions[j, 0]
        termination_data[f'Section_{j}_y'] = termination_positions[j, 1]
        termination_data[f'Section_{j}_Speed'] = termination_speeds[j]

    termination_position_df = pd.DataFrame([termination_data])
    termination_speed_df = pd.DataFrame([termination_data])

    # 保存终止时刻的数据到Excel
    random_int = np.random.randint(0, 10000)
    termination_position_file = f'D:/dragon_termination_position_{random_int}.xlsx'
    termination_speed_file = f'D:/dragon_termination_speed_{random_int}.xlsx'

    termination_position_df.to_excel(termination_position_file, index=False)
    termination_speed_df.to_excel(termination_speed_file, index=False)

    print(f"Termination position data has been saved to {termination_position_file}.")
    print(f"Termination speed data has been saved to {
```

```
    termination_speed_file}.")
else:
    print("No collision detected within the specified time.")

# 保存所有时间的数据到Excel
random_int = np.random.randint(0, 10000)
position_file = f'D:/dragon_position_{random_int}.xlsx'
speed_file = f'D:/dragon_speed_{random_int}.xlsx'

position_df = pd.DataFrame(positions_list)
speed_df = pd.DataFrame(speeds_list)

position_df.to_excel(position_file, index=False)
speed_df.to_excel(speed_file, index=False)

print(f"Position data has been saved to {position_file}.")
print(f"Speed data has been saved to {speed_file}.")
```

q3.py

```
#最终：最小螺距：0.965235 米
def mutate(individual, mutation_rate, pitch_min, pitch_max,
    mutation_strength=0.05):
    if np.random.rand() < mutation_rate:
        return np.clip(individual + np.random.uniform(-
    mutation_strength, mutation_strength), pitch_min, pitch_max)
    return individual

# 遗传算法主程序
def genetic_algorithm(pop_size, pitch_min, pitch_max,
    generations, mutation_rate, mutation_strength=0.05):
    population = initialize_population(pop_size, pitch_min,
    pitch_max)

    for generation in range(generations):
        scores = np.array([objective_function(ind[0]) for ind
```

```
    in population])
        best_score_index = np.argmin(scores)
        best_pitch = population[best_score_index][0]

        # 选择最好的个体
        selected_population = tournament_selection(population,
     scores)

        # 生成新种群
        new_population = []
        while len(new_population) < pop_size:
            parent1, parent2 = selected_population[np.random.
    choice(len(selected_population), 2, replace=False)]
            child = crossover(parent1, parent2)
            child = mutate(child, mutation_rate, pitch_min,
    pitch_max, mutation_strength)
            new_population.append(child)

        population = np.array(new_population).reshape(-1, 1)

    # 最终的最优解
    scores = np.array([objective_function(ind[0]) for ind in
    population])
    best_score_index = np.argmin(scores)
    best_pitch = population[best_score_index][0]

    return best_pitch

# 主程序
if __name__ == "__main__":
    pop_size = 50  # 种群大小
    pitch_min = 0.5  # 最小螺距
    pitch_max = 1.7  # 最大螺距
    generations = 100  # 代数
    mutation_rate = 0.1  # 变异率
```

```
    mutation_strength = 0.1  # 变异强度

    optimal_pitch = genetic_algorithm(pop_size, pitch_min, pitch_max, generations, mutation_rate, mutation_strength)

    print(f"最小螺距：{optimal_pitch:.6f} 米")
```

q4.py

```
import numpy as np
import random
from scipy.optimize import minimize

# 目标函数
def objective_function(params):
    R1, x1, y1, x2, y2 = params
    arc1_length = np.pi * R1
    arc2_length = np.pi * (R1 / 2)
    total_length = arc1_length + arc2_length
    if not check_arc_tangent(x1, y1, R1, x2, y2, R1 / 2):
        return np.inf
    spiral_params = {}  # 使用实际螺旋线参数
    if not (check_arc_spiral_tangent(x1, y1, R1, spiral_params) and
            check_arc_spiral_tangent(x2, y2, R1 / 2, spiral_params)):
        return np.inf
    return total_length

def check_arc_tangent(x1, y1, R1, x2, y2, R2):
    distance = np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    return np.isclose(distance, R1 + R2, atol=1e-2)

def check_arc_spiral_tangent(x, y, R, spiral_params):
    # 这里应实现具体的检查逻辑
    return True
```

```
26
27  # 模拟退火算法
28  def simulated_annealing(particle, temp, cooling_rate):
29      new_particle = particle + np.random.uniform(-1, 1, size=
    particle.shape)
30      new_particle = np.clip(new_particle, bounds[:, 0], bounds
    [:, 1])
31      delta = objective_function(new_particle)
32      delta_old = objective_function(particle)
33      if delta < delta_old or np.random.rand() < np.exp((
    delta_old - delta) / temp):
34          return new_particle
35      return particle
36
37  # 粒子群优化算法
38  def pso_algorithm(num_particles, num_iterations):
39      global best_particle
40      particles = np.random.uniform(bounds[:, 0], bounds[:, 1],
    size=(num_particles, len(bounds)))
41      velocities = np.random.uniform(-1, 1, size=(num_particles,
     len(bounds)))
42      personal_best_particles = np.copy(particles)
43      personal_best_scores = np.array([objective_function(p) for
     p in personal_best_particles])
44      global_best_particle = personal_best_particles[np.argmin(
    personal_best_scores)]
45      global_best_score = min(personal_best_scores)
46
47      for iteration in range(num_iterations):
48          for i in range(num_particles):
49              r1, r2 = np.random.rand(), np.random.rand()
50              velocities[i] = (0.5 * velocities[i] +
51                               2 * r1 * (personal_best_particles
    [i] - particles[i]) +
52                               2 * r2 * (global_best_particle -
```

```
    particles[i]))
            particles[i] = particles[i] + velocities[i]
            particles[i] = np.clip(particles[i], bounds[:, 0], bounds[:, 1])
            temp = 1.0 / (iteration + 1)  # 模拟退火温度逐渐降低
            particles[i] = simulated_annealing(particles[i], temp, 0.99)
            score = objective_function(particles[i])
            if score < personal_best_scores[i]:
                personal_best_particles[i] = particles[i]
                personal_best_scores[i] = score
            if score < global_best_score:
                global_best_particle = particles[i]
                global_best_score = score

    return global_best_particle, global_best_score

# 初始化和优化
def optimize_path():
    global bounds
    bounds = np.array([
        [4.5, 15],  # R1
        [-15, 15],  # x1
        [-15, 15],  # y1
        [-15, 15],  # x2
        [-15, 15]   # y2
    ])

    num_particles = 30
    num_iterations = 100

    best_particle, best_score = pso_algorithm(num_particles, num_iterations)

```

```
    R1, x1, y1, x2, y2 = best_particle
    print(f"Optimized Radius 1: {R1}")
    print(f"Optimized Center 1: ({x1}, {y1})")
    print(f"Optimized Radius 2: {R1 / 2}")
    print(f"Optimized Center 2: ({x2}, {y2})")
    print(f"Arc 1 Equation: (x - {x1})^2 + (y - {y1})^2 = {R1**2}")
    print(f"Arc 2 Equation: (x - {x2})^2 + (y - {y2})^2 = {(R1 / 2)**2}")
    print(f"Optimized Path Length: {best_score}")

    # 打印龙头的初始位置
    initial_position_x = x1 - R1
    initial_position_y = y1
    print(f"Dragon Head Initial Position: ({initial_position_x}, {initial_position_y})")

if __name__ == "__main__":
    optimize_path()
```

q5.py

```
import numpy as np
from scipy.optimize import minimize

# 计算在给定半径处的把手速度
def speed_at_radius(v_head, radius, r0):
    return v_head * np.sqrt(radius / r0)

# 计算所有把手的最大速度
def max_handle_speed(v_head, spiral_radius, arc1_radius, arc2_radius, spiral_params):
    # 计算在螺旋路径上的最大速度
    max_speed_in_spiral = v_head * (spiral_radius / spiral_params['initial_radius'])

```

```
    # 计算在圆弧上的最大速度
    max_speed_in_arc1 = speed_at_radius(v_head, arc1_radius, arc1_radius)
    max_speed_in_arc2 = speed_at_radius(v_head, arc2_radius, arc2_radius / 2)

    return max(max_speed_in_spiral, max_speed_in_arc1, max_speed_in_arc2)

# 目标函数，确保把手的速度不超过2 m/s
def objective_function(v_head, spiral_params, arc1_radius, arc2_radius, v_max):
    max_speed = max_handle_speed(v_head, spiral_params['radius'], arc1_radius, arc2_radius, spiral_params)
    return (max_speed - v_max)**2

# ABC算法实现
def abc_algorithm(spiral_params, arc1_radius, arc2_radius, v_max, num_bees=20, num_iterations=100):
    np.random.seed(0)
    colony = np.random.uniform(0, v_max, num_bees)  # 初始化蜜蜂位置（速度）
    best_v_head = colony[0]
    best_score = objective_function(best_v_head, spiral_params, arc1_radius, arc2_radius, v_max)

    for _ in range(num_iterations):
        for i in range(num_bees):
            food_source = colony[i]

            phi = np.random.uniform(-1, 1)
            k = np.random.randint(num_bees)
            new_solution = colony[i] + phi * (colony[i] - colony[k])
            new_solution = np.clip(new_solution, 0, v_max)
```

```
39
40              new_score = objective_function(new_solution,
    spiral_params, arc1_radius, arc2_radius, v_max)
41              if new_score < best_score:
42                  best_v_head = new_solution
43                  best_score = new_score
44
45              if new_score < objective_function(food_source,
    spiral_params, arc1_radius, arc2_radius, v_max):
46                  colony[i] = new_solution
47
48      return best_v_head
49
50  # 初始化螺旋参数和圆弧半径
51  spiral_params = {
52      'radius': 7.0,  # 螺旋半径的初始值，需要在实际应用中计算或
    优化
53      'initial_radius': 7.0
54  }
55
56  arc1_radius = 6.879451574886364  # 从优化结果中得到的半径
57  arc2_radius = 3.439725787443182  # 从优化结果中得到的半径
58
59  v_max = 2.0  # m/s 最大把手速度
60
61  # 运行ABC算法求解最大龙头行进速度
62  max_v_head = abc_algorithm(spiral_params, arc1_radius,
    arc2_radius, v_max)
63
64  print(f"最大龙头行进速度: {max_v_head:.6f} m/s")
```

附录 C　表格

	0 s	1 s	2 s	3 s	4 s	5 s	6 s	7 s	8 s	9 s	10 s	11 s	12 s	13 s	14 s
龙头 (m/s)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
第1节龙身 (m/s)	0	0.0005	0.0055	0.0105	0.0155	0.0205	0.0255	0.0305	0.0355	0.0405	0.0455	0.0505	0.0555	0.0605	0.0655
第2节龙身 (m/s)	0	0	0.001	0.006	0.011	0.016	0.021	0.026	0.031	0.036	0.041	0.046	0.051	0.056	0.061
第3节龙身 (m/s)	0	0	0	0.0015	0.0065	0.0115	0.0165	0.0215	0.0265	0.0315	0.0365	0.0415	0.0465	0.0515	0.0565
第4节龙身 (m/s)	0	0	0	0	0.0021	0.0071	0.0121	0.0171	0.0221	0.0271	0.0321	0.0371	0.0421	0.0471	0.0521
第5节龙身 (m/s)	0	0	0	0	0	0.0026	0.0076	0.0126	0.0176	0.0226	0.0276	0.0326	0.0376	0.0426	0.0476
第6节龙身 (m/s)	0	0	0	0	0	0	0.0031	0.0081	0.0131	0.0181	0.0231	0.0281	0.0331	0.0381	0.0431
第7节龙身 (m/s)	0	0	0	0	0	0	0	0.0036	0.0086	0.0136	0.0186	0.0236	0.0286	0.0336	0.0386
第8节龙身 (m/s)	0	0	0	0	0	0	0	0	0.0041	0.0091	0.0141	0.0191	0.0241	0.0291	0.0341
第9节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0.0046	0.0096	0.0146	0.0196	0.0246	0.0296
第10节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0.0002	0.0052	0.0102	0.0152	0.0202	0.0252
第11节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0.0007	0.0057	0.0107	0.0157	0.0207
第12节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0.0012	0.0062	0.0112	0.0162
第13节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0.0017	0.0067	0.0117
第14节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0022	0.0072
第15节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0027
第16节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
第17节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
第18节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
第19节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
第20节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
第21节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
第22节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
第23节龙身 (m/s)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 7　问题一舞龙队速度部分表格

	0 s	1 s	2 s	3 s	4 s	5 s	6 s	7 s	8 s	9 s	10 s	11 s	12 s
龙头x (m)	8.8	9.2154	7.4903	5.9215	8.2307	13.006	13.848	7.8038	1.1241	2.9063	12.954	19.8	13.785
龙头y (m)	0	-0.91	-1.511	0.8452	3.9593	2.7032	-3.244	-6.929	-2.254	6.8017	9.0963	-5E-14	-10.92
第1节龙身x (m)	11	10.129	6.0496	3.8106	7.9176	14.857	15.698	7.4907	-0.987	1.4656	13.868	22	14.699
第1节龙身y (m)	0	-2.911	-3.174	1.465	6.1369	3.8926	-4.433	-9.106	-2.874	8.4644	11.098	-6E-14	-12.92
第2节龙身x (m)	13.2	11.043	4.6089	1.6998	7.6046	16.708	17.549	7.1776	-3.098	0.0249	14.782	24.2	15.613
第2节龙身y (m)	0	-4.912	-4.837	2.0848	8.3145	5.082	-5.623	-11.28	-3.493	10.127	13.099	-8E-14	-14.92
第3节龙身x (m)	15.4	11.957	3.1682	-0.411	7.2915	18.559	19.4	6.8645	-5.209	-1.416	15.696	26.4	16.527
第3节龙身y (m)	0	-6.913	-6.499	2.7046	10.492	6.2714	-6.812	-13.46	-4.113	11.79	15.1	-9E-14	-16.92
第4节龙身x (m)	17.6	12.871	1.7275	-2.522	6.9784	20.409	21.251	6.5514	-7.319	-2.857	16.61	28.6	17.441
第4节龙身y (m)	0	-8.914	-8.162	3.3244	12.67	7.4608	-8.001	-15.64	-4.733	13.452	17.101	-1E-13	-18.92
第5节龙身x (m)	19.8	13.785	0.2868	-4.633	6.6653	22.26	23.101	6.2383	-9.43	-4.297	17.524	30.8	18.355
第5节龙身y (m)	0	-10.92	-9.825	3.9443	14.847	8.6503	-9.191	-17.82	-5.353	15.115	19.102	-1E-13	-20.92
第6节龙身x (m)	22	14.699	-1.154	-6.744	6.3522	24.111	24.952	5.9252	-11.54	-5.738	18.438	33	19.268
第6节龙身y (m)	0	-12.92	-11.49	4.5641	17.025	9.8397	-10.38	-19.99	-5.973	16.778	21.103	-1E-13	-22.92
第7节龙身x (m)	24.2	15.613	-2.595	-8.855	6.0391	25.962	26.803	5.6121	-13.65	-7.179	19.352	35.2	20.182
第7节龙身y (m)	0	-14.92	-13.15	5.1839	19.203	11.029	-11.57	-22.17	-6.593	18.44	23.105	-1E-13	-24.92
第8节龙身x (m)	26.4	16.527	-4.035	-10.97	5.726	27.812	28.654	5.2991	-15.76	-8.619	20.265	37.4	21.096
第8节龙身y (m)	0	-16.92	-14.81	5.8037	21.38	12.218	-12.76	-24.35	-7.212	20.103	25.106	-1E-13	-26.93
第9节龙身x (m)	28.6	17.441	-5.476	-13.08	5.4129	29.663	30.504	4.986	-17.87	-10.06	21.179	39.6	22.01
第9节龙身y (m)	0	-18.92	-16.48	6.4235	23.558	13.408	-13.95	-26.53	-7.832	21.766	27.107	-2E-13	-28.93
第10节龙身x (m)	30.8	18.355	-6.917	-15.19	5.0998	31.514	32.355	4.6729	-19.98	-11.5	22.093	41.8	22.924
第10节龙身y (m)	0	-20.92	-18.14	7.0433	25.735	14.597	-15.14	-28.7	-8.452	23.428	29.108	-2E-13	-30.93
第11节龙身x (m)	33	19.268	-8.357	-17.3	4.7867	33.365	34.206	4.3598	-22.1	-12.94	23.007	44	23.838
第11节龙身y (m)	0	-22.92	-19.8	7.6631	27.913	15.787	-16.33	-30.88	-9.072	25.091	31.109	-2E-13	-32.93
第12节龙身x (m)	35.2	20.182	-9.798	-19.41	4.4736	35.215	36.057	4.0467	-24.21	-14.38	23.921	46.2	24.752
第12节龙身y (m)	0	-24.92	-21.46	8.2829	30.091	16.976	-17.52	-33.06	-9.692	26.754	33.111	-2E-13	-34.93

图 8　问题一舞龙队位置部分表格

	横坐标x (m)	纵坐标y (m)	速度 (m/s)
龙头	-1.870034	-2.259691	1.000000
第1节龙身	-1.884441	-2.276318	1.000000
第2节龙身	-1.898847	-2.292944	1.000000
第3节龙身	-1.913254	-2.309571	1.000000
第4节龙身	-1.927661	-2.326197	1.000000
第5节龙身	-1.942068	-2.342824	1.000000
第6节龙身	-1.956475	-2.359450	1.000000
第7节龙身	-1.970882	-2.376077	1.000000
第8节龙身	-1.985289	-2.392703	1.000000
第9节龙身	-1.999696	-2.409330	1.000000
第10节龙身	-2.014103	-2.425956	1.000000
第11节龙身	-2.028510	-2.442583	1.000000
第12节龙身	-2.042917	-2.459209	1.000000
第13节龙身	-2.057324	-2.475836	1.000000
第14节龙身	-2.071731	-2.492462	1.000000
第15节龙身	-2.086138	-2.509089	1.000000
第16节龙身	-2.100545	-2.525715	1.000000
第17节龙身	-2.114952	-2.542342	1.000000
第18节龙身	-2.129358	-2.558968	1.000000
第19节龙身	-2.143765	-2.575595	1.000000
第20节龙身	-2.158172	-2.592221	1.000000
第21节龙身	-2.172579	-2.608848	1.000000
第22节龙身	-2.186986	-2.625474	1.000000
第23节龙身	-2.201393	-2.642101	1.000000
第24节龙身	-2.215800	-2.658727	1.000000
第25节龙身	-2.230207	-2.675353	1.000000
第26节龙身	-2.244614	-2.691980	1.000000
第27节龙身	-2.259021	-2.708606	1.000000
第28节龙身	-2.273428	-2.725233	1.000000
第29节龙身	-2.287835	-2.741859	1.000000
第30节龙身	-2.302242	-2.758486	1.000000

图 9　问题二舞龙队速度与位置部分表格

	-100	-99	-98	-97	-96	-95	-94	-93	-92	-91	-90	-89	-88
龙头	1	1	1	1	1	1	1	1	1	1	1	1	1
第1节龙身	0.951229	0.951229	0.951229	0.951229	0.951229	0.951229	0.951229	0.951229	0.951229	0.951229	0.951229	0.951229	0.951229
第2节龙身	0.904837	0.904837	0.904837	0.904837	0.904837	0.904837	0.904837	0.904837	0.904837	0.904837	0.904837	0.904837	0.904837
第3节龙身	0.860708	0.860708	0.860708	0.860708	0.860708	0.860708	0.860708	0.860708	0.860708	0.860708	0.860708	0.860708	0.860708
第4节龙身	0.818731	0.818731	0.818731	0.818731	0.818731	0.818731	0.818731	0.818731	0.818731	0.818731	0.818731	0.818731	0.818731
第5节龙身	0.778801	0.778801	0.778801	0.778801	0.778801	0.778801	0.778801	0.778801	0.778801	0.778801	0.778801	0.778801	0.778801
第6节龙身	0.740818	0.740818	0.740818	0.740818	0.740818	0.740818	0.740818	0.740818	0.740818	0.740818	0.740818	0.740818	0.740818
第7节龙身	0.704688	0.704688	0.704688	0.704688	0.704688	0.704688	0.704688	0.704688	0.704688	0.704688	0.704688	0.704688	0.704688
第8节龙身	0.67032	0.67032	0.67032	0.67032	0.67032	0.67032	0.67032	0.67032	0.67032	0.67032	0.67032	0.67032	0.67032
第9节龙身	0.637628	0.637628	0.637628	0.637628	0.637628	0.637628	0.637628	0.637628	0.637628	0.637628	0.637628	0.637628	0.637628
第10节龙身	0.606531	0.606531	0.606531	0.606531	0.606531	0.606531	0.606531	0.606531	0.606531	0.606531	0.606531	0.606531	0.606531
第11节龙身	0.57695	0.57695	0.57695	0.57695	0.57695	0.57695	0.57695	0.57695	0.57695	0.57695	0.57695	0.57695	0.57695
第12节龙身	0.548812	0.548812	0.548812	0.548812	0.548812	0.548812	0.548812	0.548812	0.548812	0.548812	0.548812	0.548812	0.548812
第13节龙身	0.522046	0.522046	0.522046	0.522046	0.522046	0.522046	0.522046	0.522046	0.522046	0.522046	0.522046	0.522046	0.522046
第14节龙身	0.496585	0.496585	0.496585	0.496585	0.496585	0.496585	0.496585	0.496585	0.496585	0.496585	0.496585	0.496585	0.496585
第15节龙身	0.472367	0.472367	0.472367	0.472367	0.472367	0.472367	0.472367	0.472367	0.472367	0.472367	0.472367	0.472367	0.472367
第16节龙身	0.449329	0.449329	0.449329	0.449329	0.449329	0.449329	0.449329	0.449329	0.449329	0.449329	0.449329	0.449329	0.449329
第17节龙身	0.427415	0.427415	0.427415	0.427415	0.427415	0.427415	0.427415	0.427415	0.427415	0.427415	0.427415	0.427415	0.427415
第18节龙身	0.40657	0.40657	0.40657	0.40657	0.40657	0.40657	0.40657	0.40657	0.40657	0.40657	0.40657	0.40657	0.40657
第19节龙身	0.386741	0.386741	0.386741	0.386741	0.386741	0.386741	0.386741	0.386741	0.386741	0.386741	0.386741	0.386741	0.386741
第20节龙身	0.367879	0.367879	0.367879	0.367879	0.367879	0.367879	0.367879	0.367879	0.367879	0.367879	0.367879	0.367879	0.367879
第21节龙身	0.349938	0.349938	0.349938	0.349938	0.349938	0.349938	0.349938	0.349938	0.349938	0.349938	0.349938	0.349938	0.349938
第22节龙身	0.332871	0.332871	0.332871	0.332871	0.332871	0.332871	0.332871	0.332871	0.332871	0.332871	0.332871	0.332871	0.332871
第23节龙身	0.316637	0.316637	0.316637	0.316637	0.316637	0.316637	0.316637	0.316637	0.316637	0.316637	0.316637	0.316637	0.316637
第24节龙身	0.301194	0.301194	0.301194	0.301194	0.301194	0.301194	0.301194	0.301194	0.301194	0.301194	0.301194	0.301194	0.301194
第25节龙身	0.286505	0.286505	0.286505	0.286505	0.286505	0.286505	0.286505	0.286505	0.286505	0.286505	0.286505	0.286505	0.286505
第26节龙身	0.272532	0.272532	0.272532	0.272532	0.272532	0.272532	0.272532	0.272532	0.272532	0.272532	0.272532	0.272532	0.272532
第27节龙身	0.25924	0.25924	0.25924	0.25924	0.25924	0.25924	0.25924	0.25924	0.25924	0.25924	0.25924	0.25924	0.25924
第28节龙身	0.246597	0.246597	0.246597	0.246597	0.246597	0.246597	0.246597	0.246597	0.246597	0.246597	0.246597	0.246597	0.246597
第29节龙身	0.23457	0.23457	0.23457	0.23457	0.23457	0.23457	0.23457	0.23457	0.23457	0.23457	0.23457	0.23457	0.23457
第30节龙身	0.22313	0.22313	0.22313	0.22313	0.22313	0.22313	0.22313	0.22313	0.22313	0.22313	0.22313	0.22313	0.22313
第31节龙身	0.212248	0.212248	0.212248	0.212248	0.212248	0.212248	0.212248	0.212248	0.212248	0.212248	0.212248	0.212248	0.212248
第32节龙身	0.201897	0.201897	0.201897	0.201897	0.201897	0.201897	0.201897	0.201897	0.201897	0.201897	0.201897	0.201897	0.201897
第33节龙身	0.19205	0.19205	0.19205	0.19205	0.19205	0.19205	0.19205	0.19205	0.19205	0.19205	0.19205	0.19205	0.19205
第34节龙身	0.182684	0.182684	0.182684	0.182684	0.182684	0.182684	0.182684	0.182684	0.182684	0.182684	0.182684	0.182684	0.182684
第35节龙身	0.173774	0.173774	0.173774	0.173774	0.173774	0.173774	0.173774	0.173774	0.173774	0.173774	0.173774	0.173774	0.173774
第36节龙身	0.165299	0.165299	0.165299	0.165299	0.165299	0.165299	0.165299	0.165299	0.165299	0.165299	0.165299	0.165299	0.165299
第37节龙身	0.157237	0.157237	0.157237	0.157237	0.157237	0.157237	0.157237	0.157237	0.157237	0.157237	0.157237	0.157237	0.157237
第38节龙身	0.149569	0.149569	0.149569	0.149569	0.149569	0.149569	0.149569	0.149569	0.149569	0.149569	0.149569	0.149569	0.149569
第39节龙身	0.142274	0.142274	0.142274	0.142274	0.142274	0.142274	0.142274	0.142274	0.142274	0.142274	0.142274	0.142274	0.142274

图 10　问题四舞龙队速度部分表格

	-100_x	-99_x	-98_x	-97_x	-96_x	-95_x	-94_x	-93_x	-92_x	-91_x	-90_x	-89_x	-88_x
龙头	-7.32979	11.70015	-7.32979	11.70015	-7.32979	11.70015	-7.32979	11.70015	-7.32979	11.70015	-7.32979	11.70015	-7.32979
第1节龙身	-0.04788	0.001349	-0.0474	0.001323	-0.04692	0.001296	-0.04644	0.00127	-0.04597	0.001244	-0.04549	0.001218	-0.04501
第2节龙身	-0.09565	0.005395	-0.09469	0.005288	-0.09374	0.005182	-0.09278	0.005077	-0.09183	0.004973	-0.09088	0.00487	-0.08992
第3节龙身	-0.14318	0.012132	-0.14176	0.011891	-0.14034	0.011652	-0.13892	0.011416	-0.13749	0.011182	-0.13607	0.01095	-0.13465
第4节龙身	-0.19038	0.021548	-0.1885	0.02112	-0.18662	0.020696	-0.18474	0.020277	-0.18286	0.019862	-0.18097	0.019451	-0.17909
第5节龙身	-0.23712	0.033628	-0.2348	0.032961	-0.23247	0.032301	-0.23014	0.031647	-0.22781	0.031	-0.22548	0.030359	-0.22315
第6节龙身	-0.2833	0.048354	-0.28054	0.047396	-0.27779	0.046448	-0.27503	0.045509	-0.27227	0.04458	-0.26951	0.04366	-0.26674
第7节龙身	-0.32879	0.065702	-0.32563	0.064402	-0.32247	0.063116	-0.3193	0.061842	-0.31613	0.060581	-0.31295	0.059333	-0.30977
第8节龙身	-0.3735	0.085644	-0.36995	0.083954	-0.3664	0.08228	-0.36284	0.080623	-0.35928	0.078982	-0.35572	0.077357	-0.35214
第9节龙身	-0.4173	0.108148	-0.4134	0.106019	-0.40949	0.10391	-0.40557	0.101821	-0.40164	0.099754	-0.39771	0.097706	-0.39377
第10节龙身	-0.4601	0.13318	-0.45587	0.130564	-0.45163	0.127973	-0.44737	0.125407	-0.44311	0.122866	-0.43883	0.12035	-0.43455
第11节龙身	-0.50178	0.160697	-0.49725	0.15755	-0.49271	0.154432	-0.48816	0.151344	-0.48359	0.148286	-0.479	0.145257	-0.4744
第12节龙身	-0.54224	0.190658	-0.53746	0.186935	-0.53265	0.183247	-0.52782	0.179594	-0.52298	0.175975	-0.51812	0.172391	-0.51324
第13节龙身	-0.58138	0.223013	-0.57637	0.218673	-0.57134	0.214373	-0.56628	0.210113	-0.5612	0.205893	-0.5561	0.201713	-0.55097
第14节龙身	-0.61908	0.257711	-0.6139	0.252715	-0.60868	0.247763	-0.60343	0.242856	-0.59815	0.237995	-0.59284	0.233179	-0.58751
第15节龙身	-0.65526	0.294696	-0.64994	0.289005	-0.64458	0.283364	-0.63918	0.277774	-0.63374	0.272234	-0.62828	0.266745	-0.62277
第16节龙身	-0.68981	0.333909	-0.6844	0.327487	-0.67894	0.321122	-0.67344	0.314813	-0.66789	0.308559	-0.66231	0.302362	-0.65668
第17节龙身	-0.72264	0.375285	-0.71719	0.368101	-0.71168	0.360978	-0.70612	0.353917	-0.70051	0.346917	-0.69485	0.339979	-0.68914
第18节龙身	-0.75365	0.418759	-0.74821	0.410782	-0.7427	0.402871	-0.73714	0.395027	-0.73151	0.387249	-0.72583	0.379539	-0.72008
第19节龙身	-0.78275	0.464259	-0.77737	0.455461	-0.77192	0.446735	-0.7664	0.43808	-0.76081	0.429497	-0.75515	0.420986	-0.74942
第20节龙身	-0.80986	0.511712	-0.8046	0.502069	-0.79926	0.492501	-0.79384	0.48301	-0.78834	0.473596	-0.78275	0.464259	-0.77709
第21节龙身	-0.83487	0.56104	-0.8298	0.55053	-0.82463	0.540099	-0.81936	0.529749	-0.814	0.519481	-0.80855	0.509294	-0.80301
第22节龙身	-0.85772	0.612163	-0.85289	0.600767	-0.84794	0.589454	-0.84289	0.578225	-0.83773	0.567082	-0.83247	0.556025	-0.8271
第23节龙身	-0.87833	0.664997	-0.87379	0.652698	-0.86913	0.640487	-0.86436	0.628363	-0.85946	0.616328	-0.85444	0.604384	-0.8493
第24节龙身	-0.8966	0.719455	-0.89243	0.706242	-0.88812	0.693118	-0.88368	0.680085	-0.8791	0.667145	-0.87439	0.654298	-0.86954
第25节龙身	-0.91248	0.775446	-0.90874	0.761309	-0.90484	0.747264	-0.9008	0.733312	-0.8966	0.719455	-0.89226	0.705693	-0.88776
第26节龙身	-0.92588	0.832878	-0.92264	0.817813	-0.91922	0.80284	-0.91564	0.787961	-0.91189	0.773178	-0.90797	0.758493	-0.90389
第27节龙身	-0.93674	0.891656	-0.93406	0.875659	-0.9312	0.859755	-0.92814	0.843946	-0.9249	0.828233	-0.92147	0.812619	-0.91786
第28节龙身	-0.945	0.95168	-0.94295	0.934754	-0.9407	0.917919	-0.93824	0.901179	-0.93557	0.884535	-0.93271	0.86799	-0.92963
第29节龙身	-0.95059	1.012851	-0.94925	0.995	-0.94768	0.977238	-0.94588	0.95957	-0.94386	0.941997	-0.94161	0.924521	-0.93914
第30节龙身	-0.95346	1.075065	-0.95289	1.056297	-0.95207	1.037617	-0.951	1.019027	-0.94969	1.00053	-0.94813	0.982129	-0.94633
第31节龙身	-0.95356	1.138216	-0.95383	1.118545	-0.95383	1.098958	-0.95356	1.079456	-0.95303	1.060044	-0.95223	1.040725	-0.95116
第32节龙身	-0.95083	1.202196	-0.95201	1.181639	-0.9529	1.161159	-0.9535	1.14076	-0.95382	1.120445	-0.95384	1.100219	-0.95357
第33节龙身	-0.94523	1.266897	-0.94739	1.245473	-0.94925	1.224119	-0.95079	1.20284	-0.95201	1.181639	-0.95293	1.16052	-0.95353
第34节龙身	-0.93671	1.332205	-0.93993	1.309939	-0.94282	1.287735	-0.94537	1.265596	-0.94758	1.243529	-0.94945	1.221536	-0.95099
第35节龙身	-0.92524	1.398008	-0.92959	1.374928	-0.93358	1.351899	-0.93721	1.328927	-0.94047	1.306016	-0.94337	1.283171	-0.94592
第36节龙身	-0.91079	1.464192	-0.91634	1.440329	-0.9215	1.416506	-0.92627	1.392729	-0.93065	1.369001	-0.93465	1.34533	-0.93827
第37节龙身	-0.89332	1.530638	-0.90014	1.506029	-0.90654	1.481447	-0.91253	1.456896	-0.9181	1.432384	-0.92327	1.407915	-0.92802
第38节龙身	-0.87282	1.597229	-0.88097	1.571915	-0.88868	1.546611	-0.89595	1.521324	-0.90278	1.49606	-0.90918	1.470826	-0.91514
第39节龙身	-0.84925	1.663847	-0.85881	1.637871	-0.8679	1.611887	-0.87652	1.585904	-0.88468	1.559927	-0.89237	1.533965	-0.8996

图 11　问题四舞龙队位置部分表格