基于等距螺线的“盘龙”运动规划分析

摘　要

“盘龙”作为传统地方民俗文化，研究舞龙队盘龙过程中盘入盘出不同轨迹下各把手的位置和速度对于了解其观赏性具有重大意义。本文从等距螺线原理出发，建立龙头运动轨迹模型，运用平面几何方法、四阶龙格库塔算法、单目标规划模型、智能优化算法等方法研究各把手在不同行进时间下，盘入终止时刻的位置和速度，以及调头曲径与最优化调头空间等问题。

针对问题一：首先介绍了舞龙队沿螺线盘入的**等距螺线模型原理**，建立以螺线中心为坐标原点的直角坐标系。其次结合由极坐标转化而得的**笛卡尔坐标**初始化龙头前把手的位置，并通过等距螺线的公式推导螺线前进的弧长，进而写出**极角微分方程**，用以拟合龙头前把手在运动轨迹中位置和速度的变化，从而模拟龙头运动轨迹。接着应用**四阶龙格库塔算法**求解上述极角变化方程，通过得出的极角递推前 300s 龙头龙尾及各把手的位置与速度。

针对问题二：在龙头运动轨迹的基础上，基于我们对舞龙队盘入终止时刻的理解，建立**板凳碰撞的几何物理模型**。其次根据板凳之间的相对距离，在**分类讨论**后着重考虑后把手与外圈木板发生碰撞的情况，然后将问题转化为**计算螺线直径小于或等于龙头前后把手的距离的临界时刻**。沿用第一问的方程与算法，可以得到第 431 秒时龙头前把手的位置坐标数据异常，不符合实际情况，**430 秒**时后把手的**位置坐标为(1.193865, 1.238682)**，此时龙头后把手位于等距螺线**第四圈**上，可认为舞龙队在 **430 秒时终止盘入，在 431 秒时发生碰撞**。

针对问题三：首先构建了基于固定调头空间的**单目标优化模型**。我们以**舞龙队能够盘入调头空间的螺距的最小值**为目标函数，以龙头极径小于或等于调头空间的半径，龙头前后把手的距离大于或等于螺线直径，螺距应满足的上下界限制等要求确定了多个约束条件。紧接着通过构建**近似多边形拟合可能的碰撞边界，以坐标原点到近似多边形的距离与到龙头板凳边缘的距离的大小比较作为优化决策**，最终采用**变步长遍历算法**单目标优化模型进行求解，最终求得了约束条件下的**最小螺距为 0.4770m**。

针对问题四：首先基于 Matlab 对盘入盘出螺线以及调头区域的仿真测绘情况，我们构建了**舞龙队盘出模型**，由于调头曲线的路径由圆弧构成与半径与线速度的关系可知：**舞龙队在调头曲线上的移动速度保持不变，**结合几何证明，通过比较调头区域内的调头曲线总弧长与两端圆弧的弧长和，**弧长和会始终等于调头空间的一半周长，为 4.5π**。得出**结论：不能通过调整圆弧，使得调头曲线变短。**紧接着应用问题一的微分方程及公式带入参数求解，**递归迭代**相邻把手之间的位置和速度，最后得到题目所要求调头前后每秒舞龙队的位置和速度。

针对问题五：首先在问题四计算出板凳龙盘出路径的前提下，我们构建了**龙头最大行进速度模型**，通过不断递增龙头前把手的行进速度，利用**智能优化算法**递归计算出盘出调头区域的把手速度，通过缩短步长精细遍历，最终得到了龙头前把手的**最大速度为** 1.2480m/s。

关键词：龙头运动轨迹模型；四阶龙格库塔算法；单目标规划模型；智能优化

一、 问题重述

“盘龙”，是人们将多条板凳首尾相连形成的蜿蜒曲折的板凳龙。在盘龙的盘入和旋出过程中，会形成阿基米德螺线。某板凳龙龙头的板长为 341 cm，龙身和龙尾的板长均为 220 cm，所有板凳的板宽均为 30 cm。每节板凳上均有两个孔，孔径（孔的直径）为 5.5 cm，孔的中心距离最近的板头 27.5 cm。相邻两条板凳通过把手连接,其运动轨迹可构成阿基米德螺线。

阿基米德螺线，亦称“等距螺线”。 当一点 P 沿动射线 OP 按相同速度运动的同时， 该射线又以等角速度绕点 O 旋转，点 P 的轨迹称为阿基米德螺线[1]。多应用于螺线盘，螺线圈等领域研究[2]。

结合板凳龙的特征与运动轨迹，基于等距螺线原理，建立数学模型解决以下问题：

问题一：题目给出了螺距为 55 cm 的等距螺线，且方向为顺时针。龙头前把手的行进速度始终保持 1 m/s。龙头的初始位置。要求建立模型求出从初始时刻到 300 s 为止，每秒整个舞龙队的位置和速度，并将结果保存到文件 result1.xlsx 中。 同时在论文中给出 0 s、60 s、120 s、180 s、240 s、300 s 时，龙头前把手、龙头后面第 1、51、101、151、201 节龙身前把手和龙尾后把手的位置和速度。

问题二：确定舞龙队盘入使得板凳之间不发生碰撞的最终时刻，并给出此时舞龙队的位置和速度，将结果存放到文件 result2.xlsx 中。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。

问题三：此时螺线的方向为逆时针旋出，需要一定的调头空间。当调头空间是以螺线中心为圆心、直径为 9 m 的圆形区域，要求确定最小螺距，满足龙头前把手能够沿着相应的螺线盘入到调头空间的边界的限制要求。

问题四：在问题 3 完成调头的基础上，且已知调头路径是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，它与盘入、盘出螺线均相切。给定螺线的螺距为 1.7 m，盘出螺线与盘入螺线关于螺线中心呈中心对称。调整圆弧，仍保持各部分相切，使得调头曲线变短；同时以调头开始时间为零时刻，给出从−100 s 开始到 100 s 为止，每秒舞龙队的位置和速度，将结果存放到文件 result4.xlsx 中。

问题五：按问题四设定的行进路线前进，龙头行进速度保持不变，要求计算龙头的最大行进速度，并满足舞龙队各把手的速度均不超过 2 m/s 的约束条件。

二、 问题分析

2.1. 问题 1 的分析

问题一给定龙头初始位置点 A，盘龙队沿螺距为 55 cm 的等距螺线顺时针盘入，且龙头前把手的行进速度始终保持 1 m/s。题目给定各把手行进组成的运动轨迹为螺线，通过建立螺线方程，基于螺线方程对龙头以及各把手处的位置进行定位并作出数学描述。首先基于极坐标系，以 O 点为坐标原点，结合极坐标与螺线方程的基本知识表示出螺线的极坐标，进而表示出整个舞龙队在螺线上的极坐标公式。至此，模型建立完毕，将龙头初始坐标等参数代入模型，可得到不同时间下，指定节数把手的位置和速度。

2.2. 问题 2 的分析

问题二中，前 300s 内各把手点位的位置和速度均可用极坐标表示，要求我们建立刚性物体盘入等距螺线时，物体不再发生碰撞的数学矢量关系式。首先我们基于问题一模型的应用，将时间参数延长，测绘出盘龙的具体路径，然后由平面几何知识分类讨论螺线内相邻情况下板凳间的碰撞情况以及螺线内圈龙头溢出碰撞的情况。至此，将碰撞

时刻的参数带入，得到碰撞时螺线上各把手的位置和速度，既舞龙队的位置和速度。

2.3. 问题 3 的分析

在问题 3 中，给定以螺线中心为圆心、直径为 9 m 的圆形区域作为调头空间区域。要求我们在满足龙头前把手能够沿着相应的螺线盘入到调头空间的边界的约束条件下，算出最小螺距。首先在螺线直径等约束条件下，加入更严谨的碰撞条件分析，确定螺距最小的目标函数。至此，模型建立完毕，采用递归的方法，得到满足约束的螺距最小值。

2.4. 问题 4 的分析

在问题 4 中，给定盘入螺线的螺距为 1.7 m，盘出螺线与盘入螺线关于螺线中心呈中心对称，要求在保证调头路径与盘入盘出曲线相切的条件下，能否调整圆弧使得调头曲线最短。首先需要在调头区域内设计两个相切的调头圆弧，使得舞龙队在调头区域内的路径最短；至此，盘出模型建立完毕，我们采用一二问的算法和公式得到舞龙队每一个把手盘入调头区域的时间和速度，最终求解出舞龙队的位置和速度。

2.5. 问题 5 的分析

在问题 5 中，要求舞龙队沿问题 4 设定的路径行进，并且龙头行进速度保持不变，在舞龙队各把手的速度均不超过 2 m/s 的约束条件下，求解龙头的最大行进速度。首先需要通过龙头前把手的行进速度，计算出盘出调头区域的把手速度，然后根据给定的参数计算出各把手的速度，并遍历检验是否满足约束条件，直到不满足约束，输出结果。

三、 模型假设

1. 本题假设只考虑离散的时间参数，即：$t=0，1，2・・・，t∈Z$。
2. 本题假设螺线 16 圈以内及以外，舞龙队各把手点位的排列按螺线圈轨迹排列。
3. 本题假设各个板凳把手人员运动所组成的轨迹近似为点在螺线上的集合。
4. 在模型中螺旋线运动和板凳的位置更新基于理想化的物理模型，如忽略了空气阻力和孔径与把手间的摩擦力等。

四、 定义与符号说明

符号定义	符号说明
θ	极角，单位为弧度
$r(\theta)$	等距螺线的半径随极角θ的变化
r_0	螺线的起始半径
p	螺距
s	龙头前把手沿螺线前进的弧长
$d_{handle1\_2}$	龙头前后把手的距离
r_{turn}	调头半径的距离
d_{tp}	龙头顶点的距离
w	板凳宽度
r_{crush}	龙头碰撞时的缩小半径

五、 模型的建立与求解

5.1. 问题 1 的模型建立与求解

5.1.1. 螺旋线模型的说明

由题目所给的情境可知，舞龙队沿着等距螺线盘入，可将等距螺线的极坐标方程被定义为：

$$r(\theta) = r_0 - \frac{p}{2\pi}\theta \tag{1}$$

其中:

- θ为极角，单位为弧度
- $r(\theta)$ 为等距螺线的半径随极角θ的变化
- r_0 为螺线的起始半径
- 螺距 $p = 55\ cm$，可得出螺线参数 $b = \frac{p}{2\pi} = \frac{55}{2\pi} \approx 8.75\ cm$

螺线在极坐标系中的坐标转化为笛卡尔坐标为：

$$\begin{cases} x(\theta) = r(\theta) \cdot cos(\theta) \\ y(\theta) = r(\theta) \cdot sin(\theta) \end{cases} \tag{2}$$

由龙头前把手的初始位置对应于螺线第 16 圈，即初始坐标为(880, 0) cm，初始角度 $\theta_0 = 16 \times 2\pi = 32\pi$。

5.1.2. 龙头运动轨迹模型的说明

在等距螺线中，s 为龙头前把手沿螺线前进的弧长，等距螺线上的弧长可表示为：

$$s = \int_{\theta 1}^{\theta_2} \sqrt{r^2(\theta) - \left(\frac{dr}{d\theta}\right)^2}\, d\theta \tag{3}$$

其中：

- $r(\theta)$为等距螺线的半径随极角θ的变化，即式(1)
- $\frac{dr}{d\theta} = b$为螺线半径对角度的变化率

简化后，表示龙头在每个微小角度变化时对应的弧长变化的等距螺线的弧长的公式为：

$$ds = \sqrt{(r_0 - b{\cdot}\theta)^2 + b^2}\, d\theta \tag{4}$$

由题可知，龙头前把手的线速度恒定在 1 m/s，且龙头线速度方向始终与螺线相切，可得出：

$$v = \frac{ds}{dt} = 1\ m/s \tag{5}$$

结合式(4),我们可以得到极角θ(t)对时间的变化关系：

$$v = \sqrt{(r_0 - b{\cdot}\theta)^2 + b^2}\,\frac{d\theta}{dt} = 1\ m/s \tag{6}$$

经过整理，我们得到极角的微分方程：

$$\frac{d\theta}{dt} = \frac{1}{\sqrt{(r_0 - b\cdot\theta)^2 + b^2}} \tag{7}$$

5.1.3. 龙头运动轨迹模型的求解

为了求解上述的微分方程获得每一秒极角$\theta(t)$的变化，我们可以使用 MATLAB 求解。MATLAB 求解微分方程的常用方法有欧拉法，龙格库塔法，线性多步法等等。由于需要求 223 个板凳，224 个把手每一秒的位置和速度，本文直接使用龙格库塔法进行改进并求解。

(1) 四阶龙格库塔算法分析

本文使用精度较高的四阶龙格库塔法(Runge-Kutta)。在解决形如 $\frac{dy}{dt} = f(t,y)$的微分方程时，四阶龙格库塔法通过计算几个中间点的斜率，来对解的估计进行加权平均，基本步骤和公式如下：

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\ k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\ k_4 &= f(t_n + h, y_n + h\cdot k_3) \\ y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{8}$$

通过这四个斜率的加权平均，得到 y 在下一个时间点t_{n+1}的估计值。

(2) 四阶龙格库塔算法在极角变化方程中的应用

由上述模型可知，对于我们的微分方程，运用龙格库塔法有以下步骤：

$$\begin{aligned} k_1 &= \frac{1}{\sqrt{(r_0 - b\cdot\theta_n)^2 + b^2}} \\ k_2 &= \frac{1}{\sqrt{(r_0 - b\cdot(\theta_n + \frac{h}{2}\cdot k_1))^2 + b^2}} \\ k_3 &= \frac{1}{\sqrt{(r_0 - b\cdot(\theta_n + \frac{h}{2}\cdot k_1))^2 + b^2}} \\ k_4 &= \frac{1}{\sqrt{(r_0 - b\cdot(\theta_n + h\cdot k_3))^2 + b^2}} \\ \theta_{n+1} &= \theta_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{9}$$

通过设置步长为 1，遍历 300 个周期，每个周期为 1 秒，按照递推公式依次求出上面四个一阶微分方程所对应的k_1，k_2，k_3，k_4，然后求出对应的下一个离散值θ_{n+1}；

(3) 通过得出的极角递推龙头和龙身每个把手的位置

根据上述龙格库塔算法得到的极角，结合式(1)式(2)可计算出龙头前把手的位置，再根据前一个把手的位置和固定距离，逐步递归得到龙头后把手与龙身的把手在每秒的位置坐标。

(4) 通过得出的位置坐标递推每个把手的速度

由于龙头前把手的速度是固定的，意味着在每个时间段内，龙头前把手都会沿等距螺线移动相同的距离，而由于龙身和龙尾的每个把手保持固定的物理距离，所以龙身和龙尾的速度会逐渐减慢，以保持距离约束。所以我们可以通过以下公式计算速度：

$$v(t) = \frac{\Delta s}{\Delta t} = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{\Delta t} \tag{10}$$

在本题的情况下，$\Delta t = 1s$，所以速度公式可简化为：$v(t) = \sqrt{\Delta x^2 + \Delta y^2}$。即在每个时间步 t 和 t+1 之间，通过两个连续时刻的 x 和 y 坐标变化来计算速度。

5.1.4. 求解过程及结果分析

根据前文中所述的计算方法，我们可以计算出所需的相关数据信息并建立部分二维平面模型，图 1 展示了龙头前把手在等距螺线上的盘入运动轨迹，图 2 展示了龙头前把手每一秒的极角变化，易知极角的变化范围一直在 0～2π之间变化，如下所示：

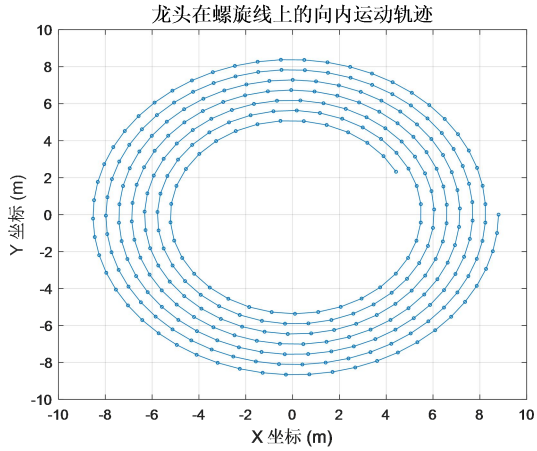

图 1 龙头前把手在等距螺线上的盘入运动轨迹

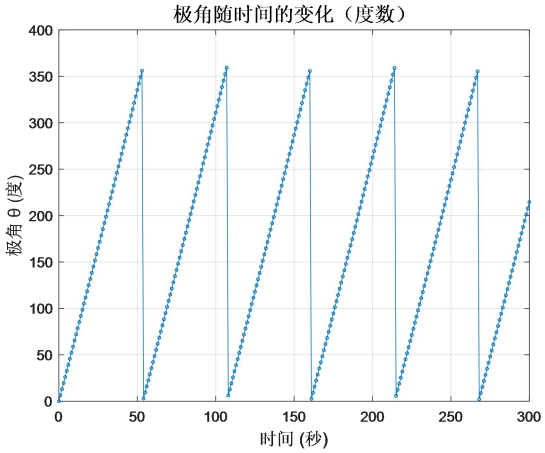

图 2 极角随时间的变化

通过递推相邻把手之间的位置和速度，我们可以得到题目所求结果如表 2 表 3 所示，从初始状态到第 300 秒，整个舞龙队的位置和速度已写入附件 result1.xlsx。

表 2 问题 1 的部分把手位置坐标展示

	0s	60s	120s	180s	240s	300s
龙头 x(m)	8.800000	5.799211	-4.084889	-2.963603	2.594510	4.420392
龙头 y(m)	0.000000	-5.771091	-6.304478	6.094783	-5.356735	2.320193
第 1 节龙身 x(m)	8.363824	7.456759	-1.445474	-5.237114	4.821232	2.459716
第 1 节龙身 y(m)	2.826544	-3.440397	-7.405882	4.359632	-3.561934	4.402343
第 51 节龙身 x(m)	-9.518732	-8.686318	-5.543148	2.890461	5.980021	-6.301323
第 51 节龙身 y(m)	1.341137	2.540105	6.377947	7.249286	-3.827743	0.466091
第 101 节龙身 x(m)	2.913984	5.687118	5.361937	1.898788	-4.917357	-6.237580
第 101 节龙身 y(m)	-9.918311	-8.001382	-7.557639	-8.471615	-6.379886	3.936228
第 151 节龙身 x(m)	10.861726	6.682309	2.388759	1.005160	2.965361	7.040877
第 151 节龙身 y(m)	1.828754	8.134546	9.727411	9.424750	8.399727	4.392789
第 201 节龙身 x(m)	4.555102	-6.619666	-10.627210	-9.287722	-7.457140	-7.458811
第 201 节龙身 y(m)	10.725118	9.025569	1.359849	-4.246667	-6.180740	-5.263168
龙尾(后)x(m)	-5.305444	7.364559	10.974348	7.383900	3.241034	1.785290
龙尾(后)y(m)	-10.676584	-8.797990	0.843472	7.492366	9.469343	9.301112

表 3 问题 1 的部分把手速度展示

	0s	60s	120s	180s	240s	300s
龙头(m/s)	1.000566	1.000001	1.000002	0.999999	1.000007	1.000015
第 1 节龙身(m/s)	1.000533	0.999962	0.999947	0.999916	0.999865	0.999725
第 51 节龙身(m/s)	1.000215	0.999662	0.999540	0.999329	0.998946	0.998076
第 101 节龙身(m/s)	0.999984	0.999453	0.999271	0.998969	0.998440	0.997312
第 151 节龙身(m/s)	0.999808	0.999299	0.999079	0.998725	0.998119	0.996871
第 201 节龙身(m/s)	0.999669	0.999180	0.998936	0.998549	0.997898	0.996584
龙尾（后）(m/s)	0.999617	0.999136	0.998884	0.998487	0.997820	0.996487

2.1. 问题 2 的模型建立与求解

2.1.1. 板凳碰撞模型的建立

在舞龙队盘入的过程中，相邻板凳保持通过把手相连，但随着螺线半径的逐渐减小，板凳之间的相对距离逐渐变短，从而两板凳之间发生碰撞。即**当龙头前后把手的距离大于或等于螺线直径时($d_{handle1\_2} \geq 2r(t)$)，龙头前后把手与外圈木板发生碰撞。**

对于龙头前把手或后把手与外圈木板发生碰撞这两种情况，根据实际，当龙头前把手无法前进时，龙头前把手停止运动，此时后把手仍在运动，由于木板固定的物理距离，后把手会脱离螺线轨迹，与外圈木板碰撞，所以本问我们重点考虑后把手与外圈木板发生碰撞的情形。如图 3 所示：

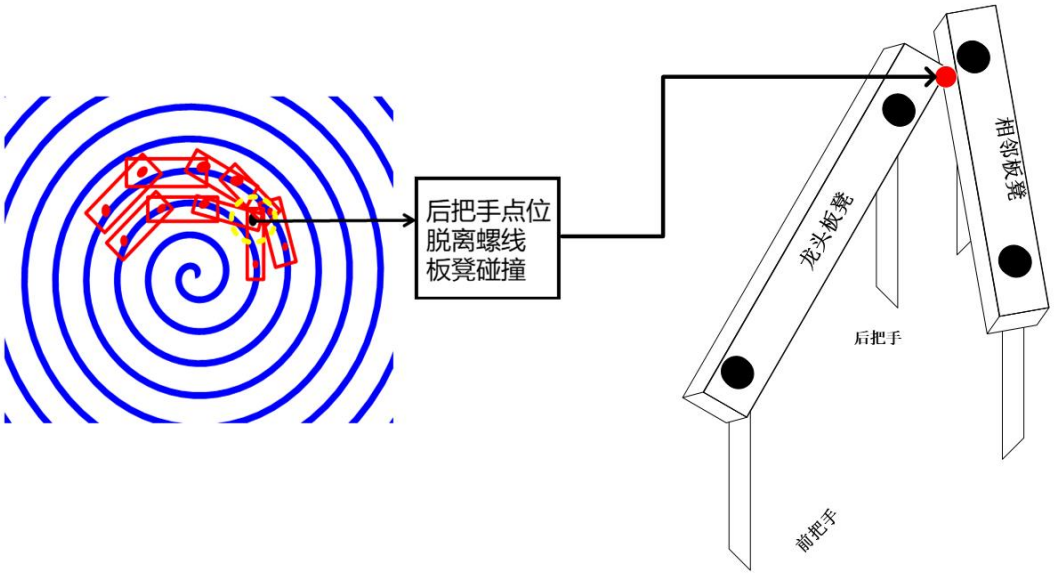

图 3 板凳碰撞模型示意图

由上述条件可得，求解舞龙队盘入的终止时刻问题可以转化为**计算螺线直径小于或等于龙头前后把手的距离的临界时刻。**在求得舞龙队盘入的终止时刻后，我们可以根据问题 1 的各公式和条件求出临界时刻舞龙队各把手的位置和速度。

2.1.2. 板凳碰撞模型的求解

根据分析，我们可以使用 MATLAB 来计算临界时刻并绘画出碰撞前的运动轨迹，计算流程与龙头前把手的运动轨迹如下图 4、图 5 所示。

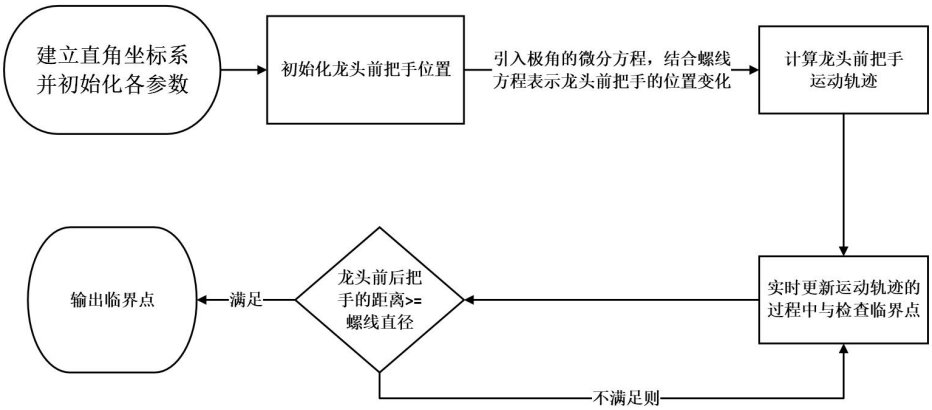

图 4 临界时刻计算流程图

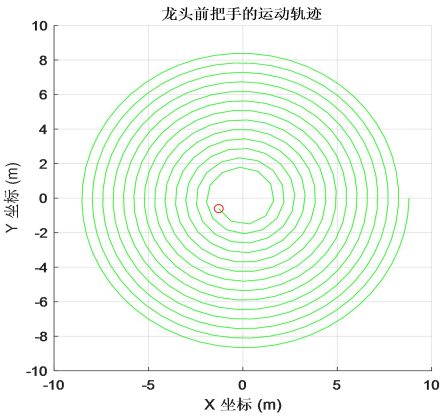

图 5 龙头前把手的运动轨迹

根据计算结果，我们得出临界点时间为 431 秒，即舞龙队在行进 431 秒时发生碰撞，临界点龙头前把手坐标(-1.277560，0.605381)，临界点龙头后把手坐标(-3.146032, -1.559893)。

为验证碰撞的发生，基于上文龙头后把手向外突出导致与外圈板凳发生碰撞的情况，通过龙头后把手的位置和板凳的宽度，可以计算龙头后把手在临界点前后的位置坐标。具体求解步骤如下：

(1) **计算龙头后把手在临界时刻的位置半径与极角**

根据龙头后把手的坐标可计算出当前半径与极角：

$$\begin{aligned} r_{back} &= \sqrt{x^2 + y^2} \approx 3.656\ m \\ \theta_{back} &= tan^{-1}(\frac{y}{x}) \approx 0.463\ rad \end{aligned} \tag{11}$$

由题干可知螺距为 55cm，此时可计算出龙头后把手位于等距螺线第六圈上。

(2) **计算龙头后把手在临界时刻前的位置半径与极角**

通过上文的公式，易得出 430 秒时也就是发生碰撞前一秒时后把手的位置坐标为(-2.505811, -0.817974)，可计算出当前半径与极角：

$$\begin{aligned} r_{pre} &= \sqrt{x^2 + y^2} \approx 2.636\ m \\ \theta_{pre} &= tan^{-1}(\frac{y}{x}) \approx 0.315\ rad \end{aligned} \tag{12}$$

可计算出此时龙头后把手位于等距螺线第四圈上。

(3) **将两组数据进行比较**

根据临界条件前后龙头后把手位置坐标的分析，第 431 秒时龙头前把手的位置坐标数据异常，不符合实际情况，则认为是与外圈发生碰撞。

综上，我们可验证结论：舞龙队在第 431 秒时发生碰撞，因此舞龙队在 430 秒时终止盘入。

2.1.3. 结果分析

根据问题 1 的计算方法，易得出舞龙队在 430 秒的位置坐标和速度，通过递推相邻把手之间的位置和速度，我们可以得到题目所求部分结果如表 4 所示，第 430 秒整个舞龙队的位置和速度已写入附件 result2.xlsx。

表 4 问题 2 的部分把手位置坐标展示

	横坐标 x (m)	纵坐标 y (m)	速度 (m/s)
龙头	0.346236	-1.492824	0.981049
第 1 节龙身	1.193865	1.238682	0.893906
第 51 节龙身	-2.893953	-3.038161	0.871631
第 101 节龙身	-1.875507	5.347678	0.869883
第 151 节龙身	-5.800386	3.599312	0.869234
第 201 节龙身	1.995637	7.555795	0.868896
龙尾	6.599798	-4.857149	0.868797

3.1. 问题 3 的模型建立与求解

3.1.1 舞龙队盘入调头空间最小螺距模型的建立

在问题三中，我们要确定舞龙队能够盘入调头空间的最小螺距。而调头空间是一个直径为 9 米的圆形区域，可得出调头空间的半径是$r_{turn} = 4.5m$。

对于本题的研究，我们可以明确我们需要优化舞龙队盘入调头空间的最小螺距，即：

优化目标：

$$min\, p \tag{13}$$

约束条件：

(1) 首先为了使龙头能够顺利进入调头空间，龙头的极径$r(\theta)$必须小于或等于调头空间的半径，如下所示：

$$r(\theta) = r_0 - \frac{p}{2\pi}\theta \le r_{turn} \tag{14}$$

(2) 其次龙头不与龙身发生碰撞，条件如同问题二：

$$d_{handle1\_2} \ge 2r(\theta) \tag{15}$$

(3) 对于本题，由于题目条件的不同，我们优化了我们对龙头的碰撞检测，通过距离比较来判断龙头顶点是否发生碰撞，即龙头靠近外圈的两端顶点与螺线中心的距离与缩小后的螺线半径进行比较，板凳运动过程中的碰撞问题也可以使用到碰撞优化算法[3]与遍历算法[4]进行求解，具体板凳之间碰撞方法和流程如下：

1)计算顶点与螺线中心的距离：

从螺线中心(0, 0)开始，设龙头靠近外圈顶点 1 与龙头靠近外圈顶点 2 的坐标为(x_1, y_1)，(x_2, y_2)，龙头顶点的坐标可以通过龙头前把手的坐标以及方向角θ_{head}计算得出，设宽度为 w，前把手的坐标为(x_{fh}, y_{fh})，则顶点 1 和顶点 2 的坐标分别为：

$$\begin{aligned} x_1 &= x_{fh} + \frac{w}{2}\cdot\cos\left(\theta_{head} + \frac{\pi}{2}\right) \\ y_1 &= y_{fh} + \frac{w}{2}\cdot\sin\left(\theta_{head} + \frac{\pi}{2}\right) \\ x_2 &= x_{fh} - \frac{w}{2}\cdot\cos\left(\theta_{head} + \frac{\pi}{2}\right) \\ y_2 &= y_{fh} - \frac{w}{2}\cdot\sin\left(\theta_{head} + \frac{\pi}{2}\right) \end{aligned} \tag{16}$$

两顶点到螺线中心的距离为：

$$\begin{aligned} d_{tp1} &= \sqrt{x_1^2 + y_1^2} \\ d_{tp2} &= \sqrt{x_2^2 + y_2^2} \end{aligned} \tag{17}$$

2) 计算外圈龙身螺线的顶点函数：

如果龙头当前在等距螺线的第 n 圈上，那么螺线的第 n+1 圈的半径可以表示为：

$$r_{n+1} = r_0 + p(n+1) \tag{18}$$

对于龙头所在螺线外圈的龙身来说，龙身板凳的靠近内圈的顶点之间的连线可以近似被认定为一条螺线，当龙头靠近外圈两顶点到圆心的距离大于龙身板凳靠近内圈顶点的近似螺线的半径时，我们认定龙头发生碰撞。具体的示意图如下图 6 所示。为简便计算，我们将这个近似螺线的半径设置为龙身所在螺线减去板凳宽度，即15cm，所以我们可以得到检测碰撞时的缩小半径为：

$$r_{crush} = r_0 + p(n+1) - 0.15 \tag{19}$$

3) 不发生碰撞条件：

$$\begin{cases} d_{tp1} \geq r_{crush} \\ d_{tp2} \geq r_{crush} \end{cases} \tag{20}$$

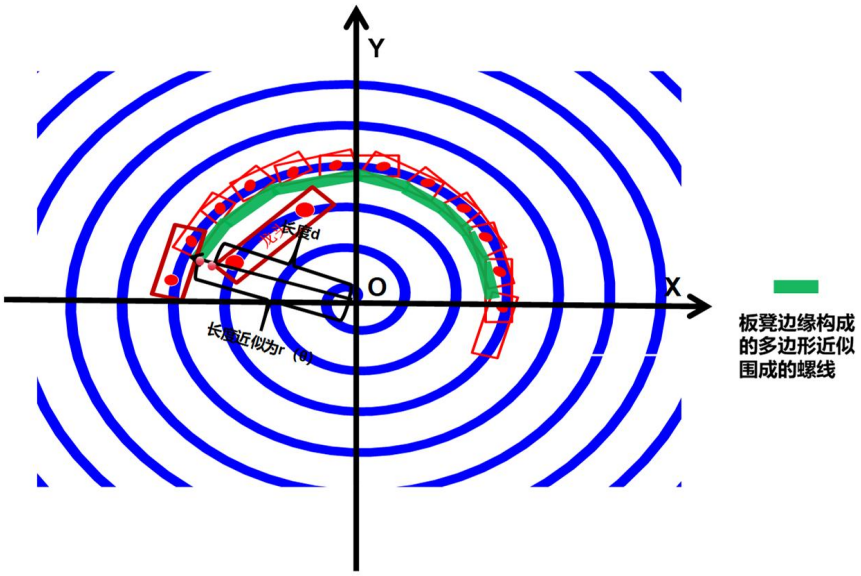

图 6 碰撞检测的示意图

另外我们从问题 1 问题 2 中已知当螺距为 55cm 时，舞龙队发生碰撞时的螺径小于调头空间半径 $r_{turn} = 4.5$m，所以为减少计算，我们可以设置螺距上界为 55cm，即能盘入调头空间的最小螺距必定小于 55cm：

$$p_{min} < 55\ cm \tag{21}$$

由题干可得板凳的宽度为 30cm，则在不发生碰撞的前提下，最小螺距应为 $2 \times \frac{30}{2} =$ 30cm。即为了减少计算，螺距的下界为 30cm：

$$p_{min} > 30\ cm \tag{22}$$

综上所述，舞龙队盘入调头空间的最小螺距的单目标优化模型如下：

$$\min p$$

$$\text{s.t} \begin{cases} r(\theta) = r_0 - \frac{p}{2\pi}\theta \leq r_{turn} \\ d_{handle1\_2} \geq 2r(\theta) \\ d_{tp1} \geq r_{crush} \\ d_{tp2} \geq r_{crush} \\ 30\ cm < p < 55\ cm \end{cases} \tag{23}$$

3.1.2 舞龙队盘入调头空间最小螺距模型的求解

根据上述优化条件，我们可以得到以下的智能优化算法步骤：

(1) **确定部分参数初始值：**我们给定龙头长度，龙身龙尾长度，螺距初值，迭代步长，调头空间半径，龙头龙身龙尾个数。

(2) **通过变步长遍历算法检验龙头是否碰撞以及检验龙头是否盘入调头空间边界：**

根据第一个步骤所设置的参数，我们将螺距 p 作为迭代目标进行遍历，若未发生碰撞且龙头盘入调头空间边界，则通过设定的迭代步长减小螺距 p 直到检测出碰撞，最后输出碰撞时上一次迭代的极限螺距。

(3) **缩短步长精细遍历：**将步长逐步缩小遍历，重复步骤一与步骤二操作，最终确定满足约束条件下的最小盘入螺距。

具体的流程步骤如下图 7 所示：

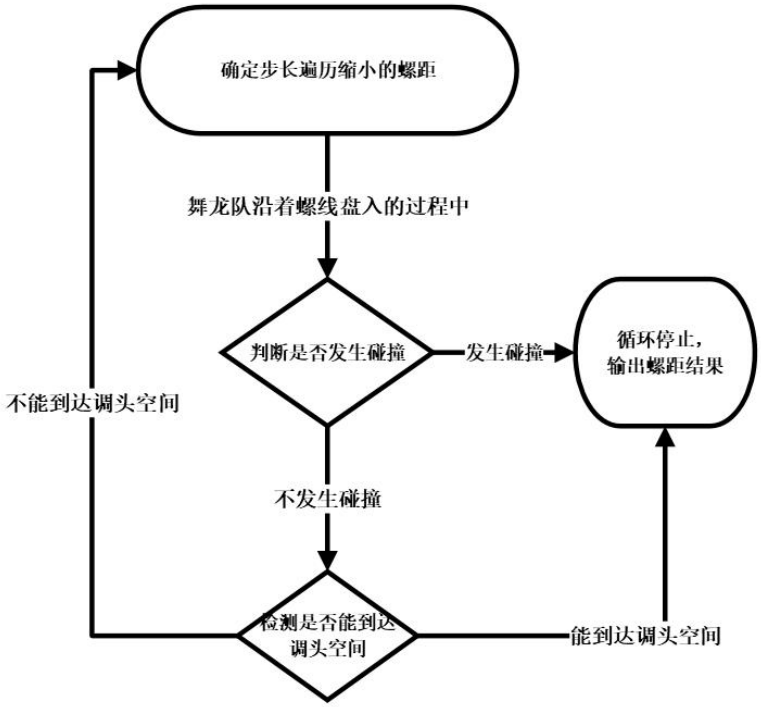

图 7 问题 3 智能优化算法流程图

3.1.3 结果分析

根据题给条件，我们自定义参数进行了智能优化算法的迭代，得到了最小的螺距为 0.4770m。

4.1. 问题 4 的模型建立与求解

4.1.1 舞龙队盘出模型的建立和说明

在问题四中，首先需要在调头区域内设计两个相切的调头圆弧，使得舞龙队在调头区域内调头的路径曲线最短；其次，要计算出调头前后舞龙队每秒的位置和速度。

由于盘入螺线与盘出螺线成中心对称，所以盘出螺线的螺距也为 1.7m，所以两螺线与调头区域的交点也关于中心对称，也就是盘入调头曲线的点(之后省略为“入点”)和盘出调头曲线的点(之后省略为“出点”)成中心对称，因此入点和出点连线过圆心(0, 0)。由于调头路径为两段圆弧相切且两圆弧的半径比为 2:1，因此两段调头圆弧为两个半圆首尾相连组成，具体的盘入盘出螺线和调头位置与调头曲线图像如图 8、图 9 所示：

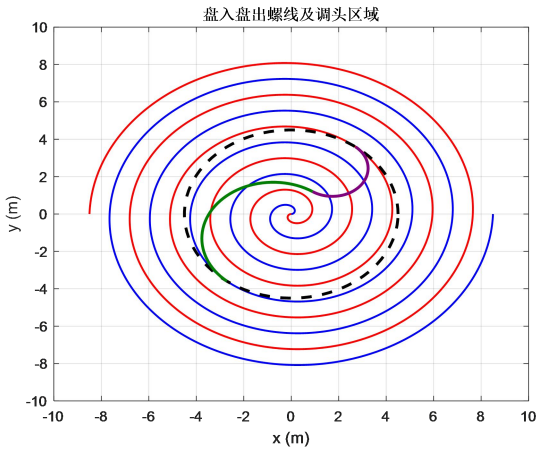

图 8 盘入盘出螺线及调头区域

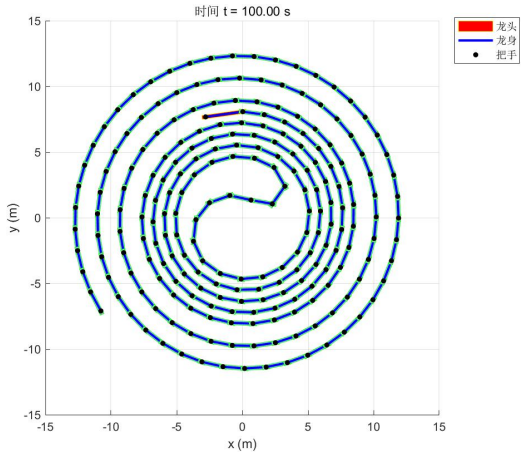

图 9 t=100s 时刻舞龙队位置示意图

若在保持各部分相切的条件下调整圆弧，即调整两圆弧的半径比，通过几何证明，易得出结论调头曲线长度不变，即不能通过调整圆弧，使得调头曲线变短。证明过程如下：

(1) 假设前一段圆弧和后一段圆弧的半径比为 $m:n$，我们可得出两段弧长各个的半径为：

$$r_m = \frac{m}{m+n} r_{turn}$$
$$r_n = \frac{n}{m+n} r_{turn} \tag{24}$$

(2) 根据弧长公式，可计算出两段弧长和为：

$$s = \pi r_m + \pi r_n = \pi r_{turn} \approx 14.13m \tag{24}$$

由弧长和易知，由于$\mathrm{r_{turn}} = 4.5\mathrm{m}$ 为定值，在调头区域内的调头曲线总弧长不会随着两弧长半径比的变化而变化，引出得证：不能通过调整圆弧，使得调头曲线变短。

4.1.2 舞龙队盘出模型的求解

为了求解调头前后舞龙队每秒的位置和速度，根据上文问题一问题二中的公式与算法，我们可求出舞龙队每一个把手盘入调头区域的时间和速度，由于调头曲线为两段圆弧，所以在调头空间内运动的速度不会发生改变，直到盘出调头区域。而在盘出调头区域后，由于每秒极角的变化量$\Delta\theta$不断减小，根据式(9)(10)，在龙头的前把手的行进速度保持不变的情况下，之后每个把手的运动速度都会不断增大。

4.1.3 结果分析

根据前文中所述的计算方法，我们可以计算出所需的相关数据信息，通过递推相邻把手之间的位置和速度，我们可以得到题目所求结果如表 5、表 6 所示，以调头开始时间为零时刻，从−100 s 开始到 100 s 为止，每秒舞龙队的位置和速度已写入附件 result4.xlsx。

表 5 问题 4 的部分把手位置坐标展示

	-100s	-50s	0s	50s	100s
龙头 x(m)	3.717164	1.898865	-3.591078	6.068711	7.674782
龙头 y(m)	7.778034	6.608301	-2.711856	1.696077	-2.800178
第 1 节龙身 x(m)	6.108521	4.475404	-4.670888	4.583227	8.073870
第 1 节龙身 y(m)	6.209273	5.366911	-0.063534	4.140036	0.031841
第 51 节龙身 x(m)	2.831491	-8.963800	-7.778145	-6.174520	3.825309
第 51 节龙身 y(m)	-10.608038	-3.629944	2.459962	-1.335770	2.412925
第 101 节龙身 x(m)	-4.802378	-5.972246	10.108539	4.889133	1.691516
第 101 节龙身 y(m)	-11.922761	10.125787	3.008493	-7.791439	-7.398277
第 151 节龙身 x(m)	-1.980993	-3.810357	10.337482	-10.531097	-3.178180
第 151 节龙身 y(m)	-14.351032	12.974784	-7.002789	-4.287404	9.601159
第 201 节龙身 x(m)	10.566998	-10.807424	12.382609	-13.171235	8.881479
第 201 节龙身 y(m)	-11.952942	10.522509	-6.872842	0.685730	8.250655
龙尾(后)x(m)	-16.527573	15.720588	-14.713128	12.762031	-7.101180
龙尾(后)y(m)	-1.011059	0.189808	-1.933627	5.543770	-10.778673

表 6 问题 4 的部分把手速度展示

	-100s	-50s	0s	50s	100s
龙头(m/s)	0.999437	0.999112	0.995393	0.998956	0.999373
第 1 节龙身(m/s)	0.999352	0.998900	0.981077	0.999273	0.999483
第 51 节龙身(m/s)	0.998996	0.998184	0.978773	0.999847	1.010323
第 101 节龙身(m/s)	0.998834	0.997933	0.978332	0.998874	0.964641
第 151 节龙身(m/s)	0.998741	0.997805	0.978144	0.998577	0.964093
第 201 节龙身(m/s)	0.998680	0.997727	0.978041	0.998433	0.963879
龙尾（后）(m/s)	0.998660	0.997702	0.978009	0.998391	0.963821

5.1. 问题 5 的模型建立与求解

5.1.1 龙头最大行进速度模型的建立

在问题 5 中，需要在确保舞龙队各把手在问题 4 中的路径行进时的速度均不超过 $2\ m/s$ 的前提下，确定龙头前把手的最大行进速度。根据上文所提到的速度变化概念，易得当把手盘出调头区域后，速度将会逐渐增大，而在 100s 时刻，我们可以得到盘出调头区域的把手并计算他们的速度，为简化计算，所以本题的解决思路转变为在 100s 时刻，通过不断递增龙头前把手的行进速度，递归迭代出盘出调头区域的把手速度，直到舞龙队有某一把手检验出速度超过 $2\ m/s$ 时停止。

5.1.2 龙头最大行进速度模型的求解

根据上述模型的建立，我们可以得到以下的智能优化算法步骤：

(1) **确定部分参数初始值**：我们给定龙头长度，龙身龙尾长度，龙头前把手速度初值，迭代步长，盘出调头区域把手个数。

(2) **通过智能优化算法检验龙头是否碰撞以及检验龙头是否盘入调头空间边界**：根据第一个步骤所设置的参数，我们将龙头前把手速度v_1作为迭代目标进行遍历并计算盘出调头区域的各个把手的速度，检测把手速度是否超过 $2\ m/s$，若未超出，根据迭代步长逐步增加龙头前把手的速度并重复迭代，直到检测出有把手速度超出 $2\ m/s$，最后输出上一次迭代的极限龙头前把手速度。

(3) **缩短步长精细遍历**：将步长逐步缩小遍历，重复步骤一与步骤二操作，最终确定满足约束条件下的最小盘入螺距。

具体的流程步骤如下图 10 所示：

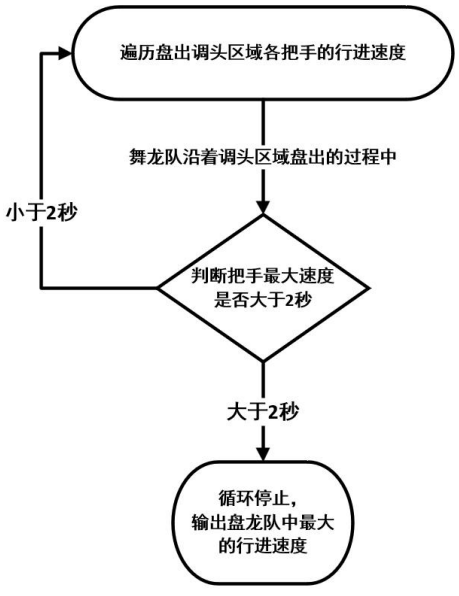

图 10 问题 5 智能优化算法流程图

5.1.3 结果分析

根据题给条件，我们自定义参数进行了**智能优化算法**的迭代，得到了最大的**龙头前把手速度为 1. 2480 m/s**。

六、 模型的评价及优化

6.1. 误差分析

1. 在求解速度和位置时，递增和迭代会逐步引入小的积累误差。
2. 在模型中螺旋线运动和板凳的位置更新基于理想化的物理模型，如忽略了空气阻力和摩擦等等。

6.2. 模型的优点

1．准确地公式：对于覆盖宽度等等信息给出了确切的公式，计算精准。
2．灵活的优化方法：采用学习类模型来处理数据，方便了后续的计算的进行，同时增加了精度。
3．融合并改进了相关算法，并分类解决，更加贴近问题，并能给出较好的解。

6.3. 模型的缺点

累计误差问题：在时间步长较大的情况下，数值方法会产生积累误差，特别是长时间模拟中，位置和速度的计算误差可能扩大。

6.4. 模型的推广

该模型可以在类似的运动轨迹模拟，生产流程优化以及非线性优化问题中，可以进一步推广和应用，例如多关节机器人或机械臂的运动规划，尤其在路径的速度和加速度控制，以及类似的螺旋线运动模型可以用于无人机或其他飞行器在复杂环境下的路径规划。

七、 参考文献

[1] 刘崇军.等距螺旋的原理与计算[J].数学的实践与认识,2018,48(11):165-174.
[2] 白亚红.阿基米德螺线在盘类机械零件中的应用实例分析[J].内燃机与配件,2018,(16):49-50.
[3] 吴晓鹏.物体碰撞优化算法及应用研究[D].广西民族大学,2018.
[4] 许熠.基于混合包围盒的碰撞检测算法的优化研究[D].南京理工大学,2013.

附录

附录清单

介绍：

result1

result2

result4

阿基米德螺线在盘类机械零件中的应用实...

表2 表3 result1 答案Python代码.ipynb

表4 result2结果 Matlab代码

等距螺旋的原理与计算_刘崇军

图1 图2 Matlab代码

图5 Matlab代码

问题3： Matlab变步长遍历算法代码

问题4： Matlab代码

问题5：确定龙头的最大行进速度并调整...

附录 1

问题 1：图 1 图 2 Matlab 代码

```
% 基本参数
pitch = 0.55;  % 螺距 55 cm = 0.55 m
b = pitch / (2 * pi);  % 螺线的参数 (b = 螺距/2π)
r0 = 16 * pitch;  % 初始半径（16 圈的半径）

% 速度参数
v = 1;  % 龙头速度 1 m/s

% 时间设置
time_step = 1;  % 每秒计算一次
max_time = 300;  % 最大模拟时间为 300 秒
time_full = 0:time_step:max_time;  % 全时间段

% 初始化极角和位置数组
theta = zeros(1, length(time_full));  % 极角随时间的变化
r = zeros(1, length(time_full));  % 半径随时间变化

% 初始条件
theta(1) = 0;  % 初始极角为 0
r(1) = r0;  % 初始半径

% 微分方程中的导数函数 dtheta/dt
```

```
dtheta_dt = @(theta) v / sqrt((r0 - b * theta)^2 + b^2);  % 修正：r(θ) = r0 - b*θ

% 使用 Runge-Kutta 4th-order 方法
for t = 2:length(time_full)
    k1 = dtheta_dt(theta(t-1));
    k2 = dtheta_dt(theta(t-1) + 0.5 * k1 * time_step);
    k3 = dtheta_dt(theta(t-1) + 0.5 * k2 * time_step);
    k4 = dtheta_dt(theta(t-1) + k3 * time_step);

    % RK4 公式更新 theta
    theta(t) = theta(t-1) + (1/6) * (k1 + 2*k2 + 2*k3 + k4) * time_step;

    % 更新半径 r(t)，修正为向内盘入
    r(t) = r0 - b * theta(t);

end

% 计算笛卡尔坐标
x = r .* cos(theta);
y = r .* sin(theta);

% 输出龙头位置随时间变化的图
figure;
plot(x, -y, '-o', 'MarkerSize', 2);
xlabel('X 坐标 (m)', 'FontSize', 12);
ylabel('Y 坐标 (m)', 'FontSize', 12);
title('龙头在螺旋线上的向内运动轨迹', 'FontSize', 14);
grid on;

% 将极角归一化到 [0, 2π] 之间
for t = 2:length(time_full)
    theta(t) = mod(theta(t), 2 * pi);
end
% 将极角转换为度数，以便更好地观察
theta_degrees = rad2deg(theta);

% 输出极角随时间变化的图（角度单位：度）
figure;
plot(time_full, theta_degrees, '-o', 'MarkerSize', 2);
xlabel('时间 (秒)', 'FontSize', 12);
ylabel('极角 θ (度)', 'FontSize', 12);
title('极角随时间的变化（度数）', 'FontSize', 14);
grid on;
```

附录 2

问题 1：表 2 表 3 result1 答案 Python 代码

```
import numpy as np
import pandas as pd
```

```
# 基本参数
pitch = 0.55   # 螺距  55 cm = 0.55 m
b = pitch / (2 * np.pi)   # 螺线的参数
r0 = 16 * pitch   # 初始半径（16 圈）
L_head = 2.86   # 扣除 27.5 cm 后龙头前后把手的实际距离
L_body = 1.65   # 扣除 27.5 cm 后龙身前后把手的实际距离

# 速度参数
v_head = 1   # 龙头前把手的固定速度  1 m/s

# 时间设置
time_step = 1   # 每秒计算一次
max_time = 301   # 模拟总时长
time_full = np.arange(0, max_time + time_step, time_step)

# 初始化坐标数组，龙头前把手的初始位置为  (8.8, 0)
x_coords = np.zeros((224, len(time_full)))
y_coords = np.zeros((224, len(time_full)))

# 第 0 秒，初始化龙头和龙身的静态坐标
x_coords[0, 0] = 8.8   # 龙头前把手的初始 X 坐标
y_coords[0, 0] = 0   # 龙头前把手的初始 Y 坐标

# 初始化龙身各个把手在 Y 轴上的排列
for i in range(1, 224):
    if i == 1:   # 龙头后把手，距离龙头前把手 2.86 米
        x_coords[i, 0] = 8.8
        y_coords[i, 0] = y_coords[i - 1, 0] + L_head
    else:   # 龙身把手，距离为 1.65 米
        x_coords[i, 0] = 8.8
        y_coords[i, 0] = y_coords[i - 1, 0] + L_body

# 初始化极角和半径数组，龙头从第 1 秒开始运动
theta = np.zeros(len(time_full))   # 龙头前把手的极角变化
r = np.zeros(len(time_full))   # 龙头前把手的半径变化

# 初始条件
theta[0] = 0   # 第 0 秒时的极角
r[0] = 8.8   # 第 0 秒时的半径


# 微分方程中的导数函数  dtheta/dt
def dtheta_dt(theta):
    return v_head / np.sqrt((r0 - b * theta) ** 2 + b ** 2)


# 从第 1 秒开始，龙头沿着螺旋线运动
for t in range(1, len(time_full)):
    # Runge-Kutta 4 阶方法求解极角
    k1 = dtheta_dt(theta[t - 1])
    k2 = dtheta_dt(theta[t - 1] + 0.5 * k1 * time_step)
    k3 = dtheta_dt(theta[t - 1] + 0.5 * k2 * time_step)
    k4 = dtheta_dt(theta[t - 1] + k3 * time_step)
```

```
        theta[t] = theta[t - 1] + (1 / 6) * (k1 + 2 * k2 + 2 * k3 + k4) * time_step
        r[t] = r0 - b * theta[t]  # 更新半径

        # 更新龙头前把手的位置
        x_coords[0, t] = r[t] * np.cos(theta[t])
        y_coords[0, t] = r[t] * np.sin(theta[t])

        # 逐个更新龙头后把手和龙身的把手位置
        for i in range(1, 224):
            # 计算当前把手与前一个把手的距离约束
            if i == 1:
                distance = L_head
            else:
                distance = L_body

            # 计算与前一个把手的相对角度方向
            dx = x_coords[i - 1, t] - x_coords[i, t - 1]
            dy = y_coords[i - 1, t] - y_coords[i, t - 1]
            angle = np.arctan2(dy, dx)

            # 根据前一个把手的位置和固定距离，计算当前把手的位置
            x_coords[i, t] = x_coords[i - 1, t] - distance * np.cos(angle)
            y_coords[i, t] = y_coords[i - 1, t] - distance * np.sin(angle)

# 计算速度
speeds = np.zeros((224, len(time_full) - 1))  # 因为速度需要两点间计算
for i in range(224):
    for t in range(len(time_full) - 1):
        dx = x_coords[i, t + 1] - x_coords[i, t]
        dy = y_coords[i, t + 1] - y_coords[i, t]
        speeds[i, t] = np.sqrt(dx ** 2 + dy ** 2) / time_step  # 计算速度

# 输出数据到 excel
output_data = {
    '时间 (s)': time_full
}

# 保存位置数据
for i in range(224):
    output_data[f'节点 {i + 1} x (m)'] = x_coords[i, :]
    output_data[f'节点 {i + 1} y (m)'] = y_coords[i, :]

# 保存速度数据
for i in range(224):
    output_data[f'节点 {i + 1} 速度 (m/s)'] = np.append(speeds[i, :], np.nan)  # 最后一个速度没有

# 转换为数据框
df = pd.DataFrame(output_data)

# 保存为 Excel 文件
df.to_excel('result_224_segments_corrected_speed.xlsx', index=False)
print("数据已保存到 result_224_segments_corrected_speed.xlsx")
```

附录 3

问题 2：图 5 Matlab 代码

```
% 基本参数
pitch = 0.55;  % 螺距 55 cm = 0.55 m
b = pitch / (2 * pi);  % 螺线的参数
r0 = 8.8;  % 初始半径，设定为 8.8 m，即龙头前把手初始位置的半径
L_head = 2.86;  % 龙头前后把手的固定距离 2.86 m
v_head = 1;  % 龙头前把手的固定速度 1 m/s

% 时间设置
time_step = 1;  % 每秒计算一次
max_time = 500;  % 模拟时间 300 秒
time_full = 0:time_step:max_time;

% 初始化坐标数组，龙头前把手的初始位置为 (8.8, 0)
x_coords = zeros(1, length(time_full));  % 只计算龙头前把手的 X 坐标
y_coords = zeros(1, length(time_full));  % 只计算龙头前把手的 Y 坐标

% 初始化极角和半径数组
theta = zeros(1, length(time_full));  % 龙头前把手的极角变化
r = zeros(1, length(time_full));  % 龙头前把手的半径变化

% 初始条件
theta(1) = 0;  % 初始角度为 0 度，意味着从 x 轴正方向出发
r(1) = 8.8;  % 初始半径为 8.8 米

% 微分方程中的导数函数 dtheta/dt
dtheta_dt = @(theta) v_head / sqrt((r0 - b * theta)^2 + b^2);

% 计算运动轨迹并检查临界点
critical_time = NaN;  % 用于记录临界点的时间
critical_x = NaN;  % 用于记录临界点的 X 坐标
critical_y = NaN;  % 用于记录临界点的 Y 坐标
for t = 2:length(time_full)
    k1 = dtheta_dt(theta(t-1));
    k2 = dtheta_dt(theta(t-1) + 0.5 * k1 * time_step);
    k3 = dtheta_dt(theta(t-1) + 0.5 * k2 * time_step);
    k4 = dtheta_dt(theta(t-1) + k3 * time_step);

    theta(t) = theta(t-1) + (1/6) * (k1 + 2*k2 + 2*k3 + k4) * time_step;
    r(t) = r0 - b * theta(t);  % 更新半径

    % 更新龙头前把手的位置
    x_coords(t) = r(t) * cos(theta(t));  % 顺时针旋转
    y_coords(t) = r(t) * sin(theta(t));  % y 轴负方向

    % 计算螺线的直径
    current_diameter = 2 * r(t);
```

```
    % 检查临界条件：当龙头前后把手距离大于或等于螺线直径
    if L_head >= current_diameter
        critical_time = time_full(t);  % 记录临界时间
        critical_x = x_coords(t);  % 记录临界点的 X 坐标
        critical_y = y_coords(t);  % 记录临界点的 Y 坐标
        disp(['临界点时间: ', num2str(critical_time), ' 秒']);
        disp(['临界点位置: (', num2str(critical_x), ', ',
num2str(critical_y), ')']);
        break;  % 停止运动
    end
end

x_coords(1) = 8.8;
% 如果找到了临界点，输出结果
if ~isnan(critical_time)
    disp(['龙头前后把手距离 >= 螺线直径，运动在 ', num2str(critical_time), '
秒停止。']);
else
    disp('未找到临界点，运动完成。');
end

% 动画绘制并保存
figure;
axis equal;
hold on;
xlim([-10 10]);
ylim([-10 10]);
xlabel('X 坐标 (m)');
ylabel('Y 坐标 (m)');
title('龙头前把手的运动轨迹');
plot_handle = plot(0, 0, 'g');  % 绿色线表示轨迹
point_handle = plot(0, 0, 'ro');  % 红点表示当前位置
grid on;

for t = 1:length(time_full)
    if isnan(critical_time) || time_full(t) <= critical_time
        set(plot_handle, 'XData', x_coords(1:t), 'YData',
-y_coords(1:t));  % 更新轨迹
        set(point_handle, 'XData', x_coords(t), 'YData',
-y_coords(t));  % 更新红点
        drawnow;
    else
        break;
    end
end
```

附录 4

问题 2：表 4 result2 结果 Matlab 代码

```
% 基本参数
pitch = 0.55;  % 螺距 55 cm = 0.55 m
b = pitch / (2 * pi);  % 螺线的参数
r0 = 8.8;  % 初始半径，设定为 8.8 m，即龙头前把手初始位置的半径
L_head = 2.86;  % 龙头前后把手的固定距离 2.86 m
v_head = 1;  % 龙头前把手的固定速度 1 m/s
board_width = 0.15;  % 板宽 15 cm

% 时间设置
time_step = 1;  % 每秒计算一次
max_time = 500;  % 模拟时间 500 秒
time_full = 0:time_step:max_time;

% 初始化坐标数组
x_coords_front = zeros(1, length(time_full));  % 计算龙头前把手的 X 坐标
y_coords_front = zeros(1, length(time_full));  % 计算龙头前把手的 Y 坐标
x_coords_back = zeros(1, length(time_full));  % 计算龙头后把手的 X 坐标
y_coords_back = zeros(1, length(time_full));  % 计算龙头后把手的 Y 坐标

% 初始化极角和半径数组
theta = zeros(1, length(time_full));  % 龙头前把手的极角变化
r = zeros(1, length(time_full));  % 龙头前把手的半径变化

% 初始条件
theta(1) = 0;  % 初始角度为 0 度，意味着从 x 轴正方向出发
r(1) = 8.8;  % 初始半径为 8.8 米

% 微分方程中的导数函数 dtheta/dt
dtheta_dt = @(theta) v_head / sqrt((r0 - b * theta)^2 + b^2);

% 计算运动轨迹
collision_detected = false;  % 标志是否发生碰撞
for t = 2:length(time_full)
    k1 = dtheta_dt(theta(t-1));
    k2 = dtheta_dt(theta(t-1) + 0.5 * k1 * time_step);
    k3 = dtheta_dt(theta(t-1) + 0.5 * k2 * time_step);
    k4 = dtheta_dt(theta(t-1) + k3 * time_step);

    theta(t) = theta(t-1) + (1/6) * (k1 + 2*k2 + 2*k3 + k4) * time_step;
    r(t) = r0 - b * theta(t);  % 更新半径

    % 更新龙头前把手的位置
    x_coords_front(t) = r(t) * cos(theta(t));  % 顺时针旋转
    y_coords_front(t) = r(t) * sin(theta(t));  % y 轴负方向

    % 更新龙头后把手的位置，距离龙头前把手固定 2.86 米
    x_coords_back(t) = (r(t) - L_head) * cos(theta(t));  % 后把手 x 坐标
    y_coords_back(t) = (r(t) - L_head) * sin(theta(t));  % 后把手 y 坐标

    % 计算当前螺线的外圈半径并扣除 15cm
    r_outer = r(t) - board_width;

    % 检查后把手是否位于圆上
```

```
        y_circle = sqrt(r_outer^2 - x_coords_back(t)^2);   % 根据 x 值计算 y_circle
        if abs(y_circle - y_coords_back(t)) < 1e-3    % 如果 y 值接近，判定为碰撞
            collision_detected = true;
            disp(['发生碰撞，时间: ', num2str(time_full(t)), ' 秒']);
            disp(['后把手坐标: (', num2str(x_coords_back(t)), ', ', num2str(y_coords_back(t)), ')']);
            break;
        end
    end

    % 如果未检测到碰撞
    if ~collision_detected
        disp('未发生碰撞');
    end

    % 绘制轨迹图像
    figure;
    plot(x_coords_front, y_coords_front, 'b', 'LineWidth', 1.5);   % 前把手的轨迹（蓝色线）
    hold on;
    plot(x_coords_back, y_coords_back, 'r', 'LineWidth', 1.5);   % 后把手的轨迹（红色线）
    axis equal;   % 保持坐标系比例一致
    xlabel('X 坐标 (m)');
    ylabel('Y 坐标 (m)');
    title('龙头前后把手的运动轨迹（顺时针）');
    legend('龙头前把手', '龙头后把手');
    grid on;

    % 保存图像
    saveas(gcf, 'dragon_head_trajectory_collision_check.png');
    disp('图像已保存为 dragon_head_trajectory_collision_check.png');
```

附录 5

问题 3： Matlab 变步长遍历算法代码

```
%% 初始化参数
clear; close all; clc;
warning off
time_step = 5; % 时间步长（秒）(这里修改，越大算的越快，但是精度也较低)
spiral_pitch_range = [50:-0.5:40] * 1e-2; % 螺距取值范围（单位：米）
min_angle = zeros(size(spiral_pitch_range)); % 每个螺距对应的最小角度


%% 主循环
for pitch_index = 1:length(spiral_pitch_range)
current_pitch = spiral_pitch_range(pitch_index);
spiral_coefficient = current_pitch / (2*pi); % 螺线方程系数 r = k*theta
head_length = 341e-2; % 龙头长度（米）
head_handle_distance = head_length - 27.5e-2*2; % 龙头把手两个孔之间的距离
body_length = 220e-2; % 龙身和龙尾长度（米）
body_handle_distance = body_length - 27.5e-2*2; % 其他凳子把手两个孔之间的距离
% 确定初始位置
initial_turns = ceil(4.5/current_pitch) + 3; % 从 4.5 米圆的外面的第三条螺线开始出发
initial_angle = 2*pi*initial_turns; % 初始角度
collision_flag = 0;
iteration_count = 0;
```

```
total_segments = 223; % 龙头+龙身+龙尾总的个数
x_positions = nan(total_segments+1, 3); % 记录每个把手点在一个时间区间内的 x 坐标
y_positions = nan(total_segments+1, 3); % 记录每个把手点在一个时间区间内的 y 坐标
angle_data = nan(total_segments+1, 3); % 记录每个孔在时间区间的位置对应的角度 theta
angle_data(1,3) = initial_angle; % 头把手初始角度
%% 盘入模拟
while collision_flag == 0
iteration_count = iteration_count + 1;
x_positions(:,1) = x_positions(:,3);
y_positions(:,1) = y_positions(:,3);
angle_data(:,1) = angle_data(:,3);
% 求解龙头运动
time_span = [0, time_step/2, time_step];
[~, new_angles] = ode45(@(t,theta) -1/(spiral_coefficient*sqrt(1+theta^2)), time_span,
angle_data(1,1));
new_x = spiral_coefficient * new_angles .* cos(new_angles);
new_y = spiral_coefficient * new_angles .* sin(new_angles);
% 更新位置和角度
x_positions(1,:) = new_x;
y_positions(1,:) = new_y;
angle_data(1,:) = new_angles;
% 计算龙身和龙尾位置
for time_index = 2:length(time_span)
for segment_index = 2:total_segments+1
handle_distance = head_handle_distance*(segment_index<=2) +
body_handle_distance*(segment_index>2);
new_angle = solve_angle(current_pitch, x_positions(segment_index-1,time_index),
y_positions(segment_index-1,time_index), angle_data(segment_index-1,time_index),
handle_distance);
angle_data(segment_index,time_index) = new_angle;
x_positions(segment_index,time_index) = spiral_coefficient * new_angle * cos(new_angle);
y_positions(segment_index,time_index) = spiral_coefficient * new_angle * sin(new_angle);
end
end
% 碰撞检测 - 优化后的版本
for segment_index = 1:round(total_segments/2)
x1 = x_positions(segment_index,2); x2 = x_positions(segment_index+1,2);
y1 = y_positions(segment_index,2); y2 = y_positions(segment_index+1,2);
angle1 = angle_data(segment_index,2);
angle2 = angle_data(segment_index+1,2);
outer_indices = find((angle1+2*pi-angle_data(:,2))>0);
outer_indices = outer_indices(max(1,end-2):end);
inner_indices = find(angle_data(:,2)-(angle2+2*pi)>0);
if isempty(inner_indices)
break;
else
inner_indices = inner_indices(1:min(3,length(inner_indices)));
end
check_indices = outer_indices(1):inner_indices(end);
for check_index = 1:length(check_indices)-1
point1 = [x_positions(check_indices(check_index),2); y_positions(check_indices(check_index),2)];
point2 = [x_positions(check_indices(check_index+1),2);
y_positions(check_indices(check_index+1),2)];
collision =
optimized_check_intersection(head_length*(segment_index<=1)+body_length*(segment_index>1),
[x1;y1], [x2;y2], body_length, point1, point2, 10, 20);
if ~isempty(collision)
collision_flag = 1;
break;
end
end
if collision_flag == 1
break;
end
end
% 检查是否到达调头空间边界
```

```
    if sqrt(x_positions(1,end)^2 + y_positions(1,end)^2) <= 4.5
    collision_flag = 1;
    end
    fprintf('当前螺距: %.2f m, 迭代步数: %d\n', current_pitch, iteration_count);
    end
    min_angle(pitch_index) = angle_data(1,end);
    end
    save problem3_tmp_data
    %% 结果可视化
    load problem3_tmp_data
    figure('Position', [100, 100, 800, 600]);
    plot(spiral_pitch_range, min_angle, 'b-', 'LineWidth', 2);
    hold on;
    theoretical_min_angle = 9*pi./spiral_pitch_range;
    plot(spiral_pitch_range, theoretical_min_angle, 'r--', 'LineWidth', 2);
    xlabel('螺距 (m)', 'FontSize', 14);
    ylabel('最小角度 (rad)', 'FontSize', 14);
    title('螺距与最小角度的关系', 'FontSize', 16);
    legend('实际最小角度', '理论最小角度', 'FontSize', 12);
    grid on;
    saveas(gcf, '问题 3_螺距与最小角度关系.png');
    saveas(gcf, '问题 3_螺距与最小角度关系.fig');
    print('问题 3_螺距与最小角度关系_高清', '-dpng', '-r300');


    %% 找到最优螺距
    [~, optimal_index] = min(abs(min_angle - theoretical_min_angle));
    optimal_pitch = spiral_pitch_range(optimal_index);
    fprintf('最优螺距: %.4f m\n', optimal_pitch);


    %% 保存结果
    results = table(spiral_pitch_range', min_angle', theoretical_min_angle', 'VariableNames', {'螺距
    ', '实际最小角度', '理论最小角度'});
    writetable(results, '问题 3_结果.xlsx');


    %% 辅助函数


    function new_angle = solve_angle(pitch, x1, y1, angle1, distance)
    spiral_coeff = pitch / (2*pi);
    angle_func = @(angle) (spiral_coeff*angle.*cos(angle)-x1).^2 +
    (spiral_coeff*angle.*sin(angle)-y1).^2 - distance^2;
    options = optimoptions('fsolve', 'Display', 'off');
    new_angle = fsolve(angle_func, angle1+0.1, options);
    while new_angle <= angle1 || abs(spiral_coeff*new_angle-spiral_coeff*angle1) > pitch/2
    new_angle = fsolve(angle_func, new_angle+0.1, options);
    end
    end


    function intersection = optimized_check_intersection(L1, X1_1, X1_2, L2, X2_1, X2_2, n, m)
    % 简化计算交点之间的矢量差值
    vec_diff = X2_1 - X1_1;
    dist_squared = sum(vec_diff.^2);
    % 如果距离较远，可以直接跳过不进行进一步检测
    if dist_squared > (L1 + L2)^2
    intersection = [];
    return;
    end


    % 如果距离较近，再进行精确的碰撞检查
    k1 = (X1_1(2)-X1_2(2))/(X1_1(1)-X1_2(1));
    k1_perp = -1/k1;
    k2 = (X2_1(2)-X2_2(2))/(X2_1(1)-X2_2(1));
    k2_perp = -1/k2;
```

```
X1_center = (X1_1+X1_2)/2;
X2_center = (X2_1+X2_2)/2;
A = [k1_perp -1; k2_perp -1];
P = A\[k1_perp*X1_center(1)-X1_center(2); k2_perp*X2_center(1)-X2_center(2)];
vec1 = X1_center-P;
vec2 = X2_center-P;
theta1 = angle(vec1(1)+1i*vec1(2));
theta2 = angle(vec2(1)+1i*vec2(2));
delta_theta = theta1-theta2;
d1 = norm(vec1);
d2 = norm(vec2);
[X,Y] = meshgrid(linspace(d1-30e-2/2,d1+30e-2/2,n), linspace(0-L1/2,0+L1/2,m));
T = [cos(delta_theta) -sin(delta_theta); sin(delta_theta) cos(delta_theta)];
XY_new = T*[X(:) Y(:)]';
intersection = find(abs(XY_new(1,:)-d2)<30e-2/2 & abs(XY_new(2,:)-0)<L2/2);
end
```

附录 6

问题 4： Matlab 代码

```
%% 初始化参数
clear; close all; clc; % 清除工作区、关闭所有图窗、清除命令窗口
warning off % 关闭警告信息

spiral_pitch = 1.7; % 螺距（米）
spiral_coefficient = spiral_pitch / (2*pi); % 螺线方程的系数 r = k * theta
head_length = 341e-2; % 龙头长度（米）
head_handle_distance = head_length - 27.5e-2*2; % 龙头把手两个孔之间的距离
body_length = 220e-2; % 龙身和龙尾长度（米）
body_handle_distance = body_length - 27.5e-2*2; % 其他凳子把手两个孔之间的距离
head_velocity = 1; % 头节点速度（米/秒）

%% 绘制盘入和盘出螺线
theta = 5*2*pi:-0.01:0; % 定义角度范围
r = spiral_coefficient * theta; % 计算螺线半径
x_in = r .* cos(theta); % 计算盘入螺线 x 坐标
y_in = r .* sin(theta); % 计算盘入螺线 y 坐标

figure('Name', '盘入盘出螺线及调头区域'); % 创建新图窗
plot(x_in, y_in, 'b-', 'LineWidth', 1.3); % 绘制盘入螺线
hold on; % 保持当前图形

theta_out = theta - pi; % 计算盘出螺线的角度
r_out = spiral_coefficient * (theta_out + pi); % 计算盘出螺线半径
x_out = r_out .* cos(theta_out); % 计算盘出螺线 x 坐标
y_out = r_out .* sin(theta_out); % 计算盘出螺线 y 坐标
plot(x_out, y_out, 'r-', 'LineWidth', 1.3); % 绘制盘出螺线

% 绘制调头区域
turning_radius = 4.5; % 调头区域半径（米）
theta_circle = linspace(0, 2*pi, 100); % 创建圆的角度数组
```

```
x_circle = turning_radius * cos(theta_circle); % 计算圆的 x 坐标
y_circle = turning_radius * sin(theta_circle); % 计算圆的 y 坐标
plot(x_circle, y_circle, 'k--', 'LineWidth', 2); % 绘制调头区域边界

xlabel('x (m)'); % 设置 x 轴标签
ylabel('y (m)'); % 设置 y 轴标签
title('盘入盘出螺线及调头区域'); % 设置图标题
xlim([-10, 10]); ylim([-10, 10]);
grid on; % 显示网格

saveas(gcf, '问题 4_盘入盘出螺线及调头区域.png'); % 保存图形

%% 计算关键几何参数
theta_in_end = turning_radius / spiral_coefficient; % 计算盘入螺线终点角度
theta_out_start = turning_radius / spiral_coefficient - pi; % 计算盘出螺线起点角度

% 计算盘入螺线终点处的斜率
slope_in_end = (spiral_coefficient*sin(theta_in_end) +
turning_radius*cos(theta_in_end)) / ...
   (spiral_coefficient*cos(theta_in_end) - turning_radius*sin(theta_in_end));

theta_max_1 = atan(-1/slope_in_end) + pi; % 计算第一段圆弧的最大角度
theta_delta = atan(tan(theta_in_end)) + pi - theta_max_1; % 计算角度差

r_c1_c2 = turning_radius / cos(theta_delta); % 计算两个圆心之间的距离
r_c2 = r_c1_c2 / 3; % 计算第二段圆弧半径
r_c1 = r_c2 * 2; % 计算第一段圆弧半径

phi = 2 * theta_delta; % 计算圆弧角度
arc_length_c1 = r_c1 * (pi - phi); % 计算第一段圆弧长度
arc_length_c2 = r_c2 * (pi - phi); % 计算第二段圆弧长度

theta_min_1 = theta_max_1 - arc_length_c1 / r_c1; % 计算第一段圆弧的最小角度
theta_min_2 = theta_min_1 - pi; % 计算第二段圆弧的最小角度
theta_max_2 = theta_min_2 + arc_length_c2 / r_c2; % 计算第二段圆弧的最大角度

% 计算第一段圆弧的圆心坐标
x_c1 = turning_radius * cos(theta_in_end) + r_c1 * cos(theta_max_1 - pi);
y_c1 = turning_radius * sin(theta_in_end) + r_c1 * sin(theta_max_1 - pi);

% 计算第二段圆弧的圆心坐标
x_c2 = turning_radius * cos(theta_out_start) - r_c2 * cos(theta_max_2);
y_c2 = turning_radius * sin(theta_out_start) - r_c2 * sin(theta_max_2);

% 绘制调头曲线
t1 = linspace(theta_min_1, theta_max_1, 50); % 创建第一段圆弧的角度数组
x_arc1 = x_c1 + r_c1 * cos(t1); % 计算第一段圆弧的 x 坐标
y_arc1 = y_c1 + r_c1 * sin(t1); % 计算第一段圆弧的 y 坐标
plot(x_arc1, y_arc1, 'g-', 'LineWidth', 2, 'Color', [0 0.5 0]); % 绘制第一段圆弧
```

```
t2 = linspace(theta_min_2, theta_max_2, 50); % 创建第二段圆弧的角度数组
x_arc2 = x_c2 + r_c2 * cos(t2); % 计算第二段圆弧的 x 坐标
y_arc2 = y_c2 + r_c2 * sin(t2); % 计算第二段圆弧的 y 坐标
plot(x_arc2, y_arc2, 'm-', 'LineWidth', 2, 'Color', [0.5 0 0.5]); % 绘制第二段圆弧
saveas(gcf, '问题 4_完整路径.png'); % 保存完整路径图

%% 计算龙头运动轨迹
time_step = 1; % 时间步长（秒）
total_segments = 223; % 总段数

% 盘入阶段
t_in = 0:time_step:100; % 创建盘入阶段的时间数组
dtheta_dt = @(t, theta) 1 ./ (spiral_coefficient * sqrt(1 + theta.^2)); % 定义角速度函数
[t_in, theta_in] = ode45(dtheta_dt, t_in, theta_in_end); % 求解微分方程

x_head_in = spiral_coefficient * theta_in .* cos(theta_in); % 计算盘入阶段龙头 x 坐标
y_head_in = spiral_coefficient * theta_in .* sin(theta_in); % 计算盘入阶段龙头 y 坐标

% 调头阶段
t_turn = 0:time_step:(arc_length_c1 + arc_length_c2); % 创建调头阶段的时间数组
theta_c1 = theta_max_1 - t_turn(t_turn <= arc_length_c1) / r_c1; % 计算第一段圆弧的角度
theta_c2 = theta_min_2 + (t_turn(t_turn > arc_length_c1) - arc_length_c1) / r_c2; % 计算第二段圆弧的角度

x_head_turn = [r_c1 * cos(theta_c1) + x_c1, r_c2 * cos(theta_c2) + x_c2]; % 计算调头阶段龙头 x 坐标
y_head_turn = [r_c1 * sin(theta_c1) + y_c1, r_c2 * sin(theta_c2) + y_c2]; % 计算调头阶段龙头 y 坐标

% 盘出阶段
t_out = 0:time_step:(100 - length(t_turn)*time_step); % 创建盘出阶段的时间数组
dtheta_dt_out = @(t, theta) 1 ./ (spiral_coefficient * sqrt(1 + (theta + pi).^2)); % 定义盘出阶段角速度函数
[t_out, theta_out] = ode45(dtheta_dt_out, t_out, theta_out_start); % 求解微分方程

x_head_out = spiral_coefficient * (theta_out + pi) .* cos(theta_out); % 计算盘出阶段龙头 x 坐标
y_head_out = spiral_coefficient * (theta_out + pi) .* sin(theta_out); % 计算盘出阶段龙头 y 坐标

% 合并所有阶段的轨迹
t_total = [-flip(t_in); t_turn'; t_out + t_turn(end)] - 100; % 合并时间数组并调整为 -100 到 100 秒
x_head_total = [flip(x_head_in); x_head_turn'; x_head_out]; % 合并所有阶段的 x 坐标
y_head_total = [flip(y_head_in); y_head_turn'; y_head_out]; % 合并所有阶段的 y 坐标
```

```
% 计算速度
v_x = diff(x_head_total) / time_step; % 计算 x 方向速度
v_y = diff(y_head_total) / time_step; % 计算 y 方向速度
v_total = sqrt(v_x.^2 + v_y.^2); % 计算总速度

%% 辅助函数定义
function [x, y] = calculate_next_point_spiral_in(spiral_pitch, x1, y1, theta1, d)
 % 计算盘入螺线上的下一个点
 k = spiral_pitch / (2*pi); % 计算螺线系数
 fun = @(theta) (k*theta.*cos(theta)-x1).^2 + (k*theta.*sin(theta)-y1).^2 - d^2; % 定
义距离方程
 options = optimset('Display', 'off'); % 设置 fsolve 选项
 theta = fsolve(fun, theta1 + 0.1, options); % 求解方程
 while theta <= theta1 || abs(k*theta - k*theta1) > spiral_pitch/2 % 检查解是否有效
  theta = fsolve(fun, theta + 0.1, options); % 如果无效，重新求解
 end
 x = k * theta * cos(theta); % 计算 x 坐标
 y = k * theta * sin(theta); % 计算 y 坐标
end

function [x, y] = calculate_next_point_spiral_out(spiral_pitch, x1, y1, theta1, d)
 % 计算盘出螺线上的下一个点
 k = spiral_pitch / (2*pi); % 计算螺线系数
 fun = @(theta) (k*(theta+pi).*cos(theta)-x1).^2 + (k*(theta+pi).*sin(theta)-y1).^2
- d^2; % 定义距离方程
 options = optimset('Display', 'off'); % 设置 fsolve 选项
 theta = fsolve(fun, theta1 - 0.1, options); % 求解方程
 while theta >= theta1 || abs(k*theta - k*theta1) > spiral_pitch/2 % 检查解是否有效
  theta = fsolve(fun, theta - 0.1, options); % 如果无效，重新求解
 end
 x = k * (theta + pi) * cos(theta); % 计算 x 坐标
 y = k * (theta + pi) * sin(theta); % 计算 y 坐标
end

function [x, y] = calculate_next_point_arc(x1, y1, theta1, d, r, xc, yc)
 % 计算圆弧上的下一个点
 delta_theta = 2 * asin(d / (2 * r)); % 计算角度增量
 theta = theta1 + delta_theta; % 计算新的角度
 x = xc + r * cos(theta); % 计算 x 坐标
 y = yc + r * sin(theta); % 计算 y 坐标
end
```

附录 7

问题 5：确定龙头的最大行进速度并调整的 Matlab 代码

```
clear;

% 加载数据
load problem4_save_data;  % 假设 x_positions, y_positions 已经加载

% 初始化参数
max_speed_limit = 2;  % 板凳的最大允许速度 2 m/s
head_speed_increment = 0.01;  % 每次增加龙头速度的增量
current_head_speed = head_speed;  % 初始龙头速度
max_iterations = 1000;  % 限制最大循环次数，防止无限循环

% 循环增加龙头速度，直到找到最大允许速度
for iter = 1:max_iterations
    % 计算新的速度
    velocity_x = diff(x_positions, 1, 2) / time_step;
    velocity_y = diff(y_positions, 1, 2) / time_step;
    velocity = sqrt(velocity_x.^2 + velocity_y.^2) * (current_head_speed / head_speed);

    % 计算每个板凳的最大速度
    max_velocities = max(velocity, [], 2);

    % 诊断输出当前最大速度
    fprintf('当前龙头速度: %.4f m/s，最大板凳速度: %.4f m/s\n', current_head_speed,
max(max_velocities));

    % 检查是否所有板凳的速度都小于等于 2 m/s
    if all(max_velocities <= max_speed_limit)
        % 如果所有板凳速度都在限制范围内，则继续增加龙头速度
        current_head_speed = current_head_speed + head_speed_increment;
    else
        % 一旦发现某个板凳的速度超过限制，则停止并使用上一次的速度
        current_head_speed = current_head_speed - head_speed_increment;
        fprintf('板凳速度超过限制，调整后的最大允许龙头速度: %.4f m/s\n',
current_head_speed);
        break;
    end

    % 添加限制防止过多循环
    if iter == max_iterations
        warning('达到最大迭代次数，可能需要进一步优化！');
    end
end

% 输出最终结果
fprintf('问题 5 结果：\n');
fprintf('允许的最大龙头速度: %.4f m/s\n', current_head_speed);

% 计算并保存最终的速度数据
velocity_x = diff(x_positions, 1, 2) / time_step;
velocity_y = diff(y_positions, 1, 2) / time_step;
velocity = sqrt(velocity_x.^2 + velocity_y.^2) * (current_head_speed / head_speed);

% 计算每个板凳的最大速度
max_velocities = max(velocity, [], 2);
```

```
% 可视化各板凳的最大速度分布
figure('Name', '问题 5：各板凳最大速度分布');
bar(max_velocities);
hold on;
plot([1, length(max_velocities)], [max_speed_limit, max_speed_limit], 'r--', 'LineWidth', 2);
title('各板凳最大速度分布');
xlabel('板凳编号');
ylabel('最大速度 (m/s)');
legend('最大速度', '速度限制');
saveas(gcf, '问题 5_各板凳最大速度分布.png');

% 保存最终结果到 Excel 文件
T = table((1:length(max_velocities))', max_velocities, ...
    'VariableNames', {'板凳编号', '最大速度'});
writetable(T, '问题 5_各板凳最大速度.xlsx');

% 计算新的速度矩阵
new_velocity = velocity;

% 保存新速度矩阵到 Excel
T_new_velocity = table((1:size(new_velocity, 1))', new_velocity);
writetable(T_new_velocity, '问题 5_各板凳速度.xlsx');

% 验证新的速度矩阵是否所有元素都不超过 2 m/s
if all(new_velocity(:) <= max_speed_limit)
    disp('验证通过：新的速度方案中，所有板凳的速度均不超过 2 m/s');
else
    warning('新的速度方案中，仍有板凳速度超过 2 m/s，可能需要进一步优化');
end

% 可视化新旧速度对比
figure('Name', '问题 5：新旧速度对比');
plot(max_velocities, 'b-', 'LineWidth', 2);
hold on;
plot(max(new_velocity, [], 2), 'r-', 'LineWidth', 2);
plot([1, length(max_velocities)], [max_speed_limit, max_speed_limit], 'k--', 'LineWidth', 2);
title('新旧速度方案对比');
xlabel('板凳编号');
ylabel('最大速度 (m/s)');
legend('原速度方案', '新速度方案', '速度限制');
saveas(gcf, '问题 5_新旧速度对比.png');
```