

基于计算几何学求解“板凳龙”穿梭行进问题

摘要

浙闽地区的“板凳龙”是传统舞龙习俗的一种代表形式，舞龙队能够控制板凳龙，灵活地盘旋运动。本文通过建立计算几何模型，利用数值方法分析板凳龙盘入、调头、盘出的过程，对板凳龙行进轨迹、行进速度、板凳碰撞等物理过程建立了严谨细致的数学模型。该问题的研究能为“板凳龙”民俗活动提供指导性意见，取得更好的观赏性。

针对问题一，本文建模描述了 t 时刻板凳龙各个把手的位置及速度。本文首先建立极坐标系，利用平面解析几何、微元法、刚体运动学等方法，列出各个把手位置矢量的约束方程，并采用 **Halley** 方法迭代求解。本文还提出了一种精较为确的近似估计相邻把手位置的方法，以避免约束方程多解导致的若干问题。在此基础上应用平面矢量分析得出相邻把手速度的递推关系。本文还可视化了板凳龙的运动状态，总结出了一些对后续分析有意义的规律。

针对问题二，本文建模并导出了碰撞判定算法、首次碰撞时刻搜索算法。本文用 Python 代码实现了静态碰撞判定，并使用二分查找解得首次碰撞时刻是 412.47383 s。随后复用问题一的模型，求解此时各个把手的位置和速度。

针对问题三，本文利用问题一位置模型与问题二碰撞判断模型，将“求解满足条件的最小螺距”的问题转化为二维变量空间上的优化问题。本文定义了一个性质较好的目标函数，并综合使用模拟退火算法和自适应网格搜索算法，做到了快速准确求解最小螺距。本文解得的最小螺距为 $45.0337 \text{ cm} = 0.450337 \text{ m}$ 。

针对问题四，本文推广了在盘入路径上建立的位置与速度模型，用于建模分析板凳龙在“盘入—调头—盘出”复合路径上的运动状态。首先通过平面解析几何方法证明了无法通过优化题目所述的圆弧获得更短的调头路径。然后，本文引入一个与极角密切相关的参数 ξ ，用于统一构建复合路径的参数方程模型，并类比问题一建立了板凳龙各个把手的位置与速度模型。重要的是，本文构造的参数方程 $\mathbf{r}(\xi) \in C^1(\mathbb{R})$ ，因此能应用 Halley 方法快速数值求解刚体运动学方程，以导出板凳龙的运动状态。

针对问题五，本文首先定义一个最大速度比例函数 $\max_i \frac{v_i(t)}{v_0}$ ，可以根据问题四的复合路径参数方程模型求解该函数在 t 时刻的值。本文利用已有结果，结合定性分析，将出现上界的时间范围锁定至 $t \in [13 \text{ s}, 16 \text{ s}]$ ，并采用三分查找法求解其最大值。最后解出满足题目限速的龙头的最大行进速度 $v_{0\ max} = 1.246267 \text{ m/s}$ 。

关键词：平面解析几何 微元法 刚体运动学 Halley 方法 三分查找 模拟退火
自适应网格搜索

一、问题重述

1.1 问题背景

舞龙习俗起源于古时的祈雨祭祀，蕴含着古老的自然崇拜、祖先崇拜、图腾崇拜等信仰内核，被称为“民间重要的文化基石和源泉”。浙闽地区的“板凳龙”，也称“盘龙”，则是传统舞龙习俗的一个代表形式。人们会将上百条的板凳首尾相接，制成百转千回的板凳龙。舞龙时，龙身和龙尾在龙头的带领下呈圆盘状相随盘旋。为了达到更好的观赏性，在舞龙队能无碰撞地盘入与盘出的条件之下，如何使盘动所需要的面积尽可能小、行进的速度更快，是我们需要通过数学建模解决的问题。

1.2 需要解决的问题

现已知某板凳龙中每一节板凳的尺寸与把手中心在板凳上的位置运用数学思想建立对应模型，解决以下问题：

1. 板凳龙盘动时，龙头前把手以 1 m/s 的速度沿螺距 55 cm 的等距螺旋线顺时针盘入，求解从初始时刻到第 300 s 之间，舞龙队中所有把手中心的位置—时间关系、速度—时间关系。
2. 仍沿上述螺线盘入，在板凳不发生碰撞的条件下求解舞龙队终止盘入的时刻，以及此时舞龙队的位置与速度。
3. 舞龙队将在一个位于螺线中心、直径为 9 m 的“调头空间”内实现盘入盘出的切换。在龙头前把手能沿着螺线一直盘入至调头空间边界的前提下，确定最小螺距。
4. 已知
 - (a) 盘入螺线的螺距为 1.7 m ；
 - (b) 盘出螺线与盘入螺线关于螺线中心呈中心对称；
 - (c) 调头路径是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。

证明更短的两弧相切而成的调头路径是否存在。再规定龙头前把手的行进速率恒等于 1 m/s ，需解出调头开始时刻的前 100 s 至后 100 s 期间舞龙队的位置和速度。

5. 舞龙队沿前述的“盘入—调头—盘出”路径行进，龙头行进速率保持不变。需确定龙头的最大行进速率，使得舞龙队各把手的速度均不超过 2 m/s 。

二、问题分析

2.1 问题一：已知时间求解舞龙队的位置与速度

在问题一中，要求求解 0 s 至 300 s 内，每秒各个把手中心的位置与速率。

首先我们需要建立描述板凳龙把手位置的数学模型。以螺线中心为极点，用极坐标描述每个把手的位置。由等距螺线方程 $r = \frac{D\theta}{2\pi}$ ，容易获得某个把手极径与极角的关系。又已知等距螺线的螺距，以及每条板凳的把手间距，因此只要获得龙头前把手的位置，就可以综合使用微元法、平面解析几何、Halley 算法，一一求解每个把手的位置。综上，我们可以用龙头前把手的极角唯一确定整条板凳龙的位置。

然后，使用微元法分析等距螺线的形状，积分获得从极点开始计算的弧长与极角的函数 $s(\theta)$ ，再由弧长 s 反解出极角 θ 的数值解。至此就解出了 t 时刻整条板凳龙的位置。

至于速度的求解，需要应用平面向量和运动学分析。因为板凳的运动是理想刚体的平面运动，所以其前、后把手的速度矢量 v_i 、 v_{i+1} 在前、后把手连线上的分量必须相等。据此能够解出瞬时速率 v_i 与 v_{i+1} 的递推关系式。由龙头前把手的速率，可以由此递推得出各个把手的速率。

2.2 问题二：求解首次碰撞时刻

在问题二中，舞龙队仍沿问题一设定的螺线盘入，要求求解能使板凳间不发生碰撞的盘入终止时刻，即首次碰撞的时刻 T_c 以及此时舞龙队的位置与速度。

我们利用几何法分析和 Python 中的 Matplotlib 库，将每一条板凳绘制成分布在等距螺线上的矩形，再利用 `intersection` 函数判断矩形之间是否存在交集即板凳间是否碰撞。接着我们使用二分搜索法求出一个碰撞临界时间，再向前回溯检查该时刻前是否存在发生碰撞的时间，多次迭代更新二分搜索法的右边界，即可得出首次碰撞时刻。再根据第一问中由时间求解位置与速度的模型即可得出 T_c 时舞龙队的位置与速度。

2.3 问题三：在调头空间外不发生碰撞的最小螺距

在问题三中，我们要求解能使龙头在盘入到调头空间边界的过程中，不产生碰撞的最小螺距。基于前两问建立的模型，只要提供螺距 D 、龙头前把手的极角 θ_0 ，就能判定此状态下舞龙队是否发生了碰撞。这是一个优化类问题，求解方式有 2 种：

1. 设计一个能够综合反映碰撞与否、当前螺距的目标函数 $check(D, \theta_0)$ ，采用模拟退火算法寻找其最小值点，从中获得满足条件的 $\min D$ ；
2. 在有意义的搜索空间内，均匀采集若干 (D, θ_0) 样本点，根据碰撞与否绘制出散点图，再根据图像缩小查找范围，反复迭代查找满足条件的 $\min D$ 。

前者属于启发式算法，往往运行较快但精度不足；后者属于二分法暴力搜索，计算很慢但精度可控。因此，我们选择先用模拟退火算法获得 4 位有效数字的近似解，然后用自适应网格搜索法改进的二分法获得精确解，使精度提高到 6 位有效数字。

2.4 问题四：求解舞龙队调头前后的运动状态

在问题四中，我们首先需建立调头路径的平面几何模型，以证明更短的两弧相切而成的调头路径不存在（定理 1）。

然后根据几何分析、平面向量等方法，确定“盘入—调头—盘出”复合路径的结构。然后选取一个全局统一的参数 ξ 建立复合路径的参数方程 $\mathbf{r} = \mathbf{r}(\xi)$ ，以便用统一的变量描述路径上一点的位置。同时我们希望构造出的 $\mathbf{r}(\xi) \in C^1(\mathbb{R})$ ，以便在计算机上部署 Halley 法求解程序^[2]。最后使用问题一中的舞龙队位置与速度模型，求解出给定时刻下整个舞龙队各个把手中心的位置与速度。

2.5 问题五：已知限速求解龙头最大行进速度

在问题五中，我们要求解使得舞龙队速率不超过 2m/s 的最大龙头速率。根据问题四建立的复合路径参数方程模型， t 时刻下我们可以遍历全部把手速度求出速率最大比值 $\max_i \frac{v_i(t)}{v_0}$ ，它是时刻 t 的函数。要求解该函数在一定时间范围内的上界，可以先由问题四的结果找到速度最大值出现的大致范围，作为三分查找算法的左右边界获得 $\sup_t \max_i \frac{v_i(t)}{v_0}$ ，最后反解出龙头的最大行进速度 $v_{0\ max}$ 。

三、模型假设

假设 1 忽略板凳的高低错落、左右倾斜，将板凳与把手中心点的运动视为二维平面运动；

假设 2 忽略板凳的形变，将每个板凳视为矩形理想刚体；

假设 3 每个把手都能够随时任意改变运动速度，以响应龙头的运动。

假设 4 板凳与板凳的连接处足能够随时任意改变夹角，以响应前、后把手中心点的运动。

四、符号说明

表1 符号表

符号	说明	单位
i	板凳或把手的标号 ¹	/
\mathbf{r}_i	第 i 个把手的位矢	/
r_i, θ_i	第 i 个把手的极径、极角	cm, rad
x_i, y_i	第 i 个把手的横、纵坐标	cm
\mathbf{v}_i	第 i 个把手的瞬时速度矢量	/
v_i	第 i 个把手的瞬时速率	cm/s
θ_{init}	盘入过程的起点对应的极角	rad
\mathbf{l}_i	第 i 条板凳上，前把手相对后把手的位矢	/
l_i	第 i 条板凳前、后把手的间距	cm
$L_{\text{head}}, L_{\text{body}}, L_{\text{tail}}$	龙头、龙身、龙尾三类板凳的前、后把手的间距	cm
D	螺距	cm
s	螺线的弧长，从极点开始计算	cm
t	时间	s
T_c	首次发生碰撞的时刻	s
R	掉头空间边界半径	cm
ξ	复合路径的参数方程中的参数	(rad)

五、模型的建立与求解

5.1 问题一模型的建立及求解

5.1.1 问题一：舞龙队位置模型的建立

舞龙队的盘入轨迹为等距螺线（也称为阿基米德螺线）其上任一点的极径 r_i 与该点的极角 θ_i 之间满足一定的关系。具体来说，螺距为 D 的等距螺线的极坐标方程可以表示为

$$r(\theta) = \frac{D\theta}{2\pi} \quad (1)$$

由此我们获得了把手极径与极角的关系。求解 t 时刻舞龙队每一个把手的位置 $\{\mathbf{r}_i\}_{i=0}^{223}$ 的问题也就转化为了求 t 时刻下每一个把手的极角 $\{\theta_i\}_{i=0}^{223}$ 的问题。首先计算出 θ_0 ，再

¹注： $i = 0$ 表示龙头板凳、龙头前把手， $1 \leq i \leq 221$ 表示第 i 个龙身板凳、龙身前把手， $i = 222$ 表示龙尾板凳、龙尾前把手， $i = 223$ 表示龙尾后把手。

向后递推。如图 1 所示，使用微元法分析等距螺线，可求得等距螺线的弧长公式为

$$\begin{aligned}s(\theta) &= \int_0^\theta \sqrt{(r(\theta))^2 + (r'(\theta))^2} d\theta \\&= \frac{D}{2\pi} \int_0^\theta \sqrt{\theta^2 + 1} d\theta \\&= \frac{D}{4\pi} \left(\theta \sqrt{1 + \theta^2} + \sinh^{-1} \theta \right)\end{aligned}\quad (2)$$

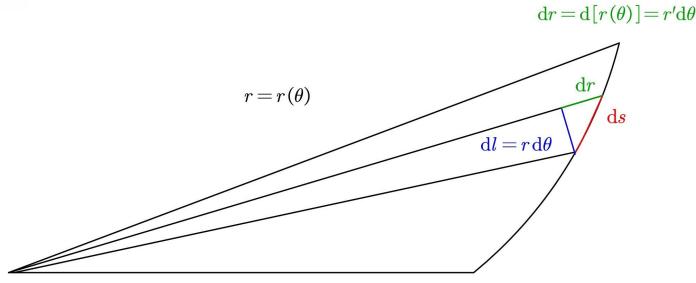


图 1 微元法积分计算 $s(\theta)$

已知龙头前把手的行进速度始终保持 $v_0 = 100 \text{ cm/s}$ ，则该时刻下龙头前把手处的弧长

$$s(\theta_0) = s(\theta_{\text{init}}) - v_0 t \quad (3)$$

其中， θ_{init} 是盘入起点的极角，本题中 $\theta_{\text{init}} = 32\pi$ 。

在本研究中，我们联立式 (2) 与式 (3)，采用 Halley 方法来求解式 (3) 的根 θ_0 。Halley 方法是一种用于寻找非线性方程的根的技术，由于其在迭代时采用了二阶近似，故收敛速度通常比 Newton 迭代求解法更快。详细推导过程见附录 A。

下面由 θ_0 计算 $\{\theta_i\}_{i=1}^{223}$ 。根据等距螺线的方程 (式 (1))，将第 i 条板凳的前后把手的极角 θ_i 和 θ_{i+1} 代入，我们可由平面解析几何写出这两个把手的位矢

$$\mathbf{r}_i = r(\theta_i) \begin{bmatrix} \cos \theta_i \\ \sin \theta_i \end{bmatrix} = \begin{bmatrix} \frac{D}{2\pi} \theta_i \cos \theta_i \\ \frac{D}{2\pi} \theta_i \sin \theta_i \end{bmatrix} \quad (4)$$

$$\mathbf{r}_{i+1} = r(\theta_{i+1}) \begin{bmatrix} \cos \theta_{i+1} \\ \sin \theta_{i+1} \end{bmatrix} = \begin{bmatrix} \frac{D}{2\pi} \theta_{i+1} \cos \theta_{i+1} \\ \frac{D}{2\pi} \theta_{i+1} \sin \theta_{i+1} \end{bmatrix} \quad (5)$$

又已知龙头板凳前后把手的间距

$$l_i = \begin{cases} L_{\text{head}} & = 286 \text{ cm}, \quad i = 0 \\ L_{\text{body}} & = 165 \text{ cm}, \quad 1 \leq i \leq 221 \\ L_{\text{tail}} & = 165 \text{ cm}, \quad i = 222 \end{cases} \quad (6)$$

且有刚体运动学方程

$$\|\mathbf{r}_i - \mathbf{r}_{i+1}\| = l_i \quad (7)$$

联立式(4)至(7), 可以列出方程

$$\theta_i^2 + \theta_{i+1}^2 - 2\theta_i\theta_{i+1} \cos(\theta_i - \theta_{i+1}) - \frac{4\pi^2 l_i^2}{D^2} = 0 \quad (8)$$

理论上, 可以据此获得 θ_i 到 θ_{i+1} 的递推关系, 进而确定所有 $\{\theta_i\}_{i=1}^{223}$ 的值。然而式(8)没有解析解, 故采用 Halley 方法求其数值解。

现尝试导出 Halley 方法在本问题中的具体迭代公式。将式(8)的等号左边记为 θ_{i+1} 的函数 $f(\theta_{i+1})$, 即

$$f(\theta_{i+1}) = \theta_i^2 + \theta_{i+1}^2 - 2\theta_i\theta_{i+1} \cos(\theta_i - \theta_{i+1}) - \frac{4\pi^2 l_i^2}{D^2} \quad (9)$$

则有迭代公式

$$\theta_{i+1}^{(k+1)} = \theta_{i+1}^{(k)} - \frac{f(\theta_{i+1}^{(k)})}{f'(\theta_{i+1}^{(k)})} \left(1 - \frac{f(\theta_{i+1}^{(k)}) f''(\theta_{i+1}^{(k)})}{2(f'(\theta_{i+1}^{(k)}))^2} \right)^{-1} \quad (10)$$

其中 $f(\theta_{i+1})$ 的一阶导数和二阶导数如下

$$f'(\theta) = 2\theta - 2\theta_i \cos(\theta_i - \theta) - 2\theta_i \theta \sin(\theta_i - \theta) \quad (11)$$

$$f''(\theta) = 2 - 4\theta_i \sin(\theta_i - \theta) + 2\theta_i \theta \cos(\theta_i - \theta) \quad (12)$$

迭代直到 $|\theta_{i+1}^{(k+1)} - \theta_{i+1}^{(k)}|$ 小于预设的收敛阈值 (本研究中设定为 10^{-6}) 时停止。当达到此条件时, 序列 $\{\theta_{i+1}^{(k)}\}$ 被认为收敛至真实 θ_{i+1} 的近似值。

然而进一步研究发现, 式(8)一般存在多个解。故迭代初值的选取对于结果的收敛性和正确性具有至关重要的作用。

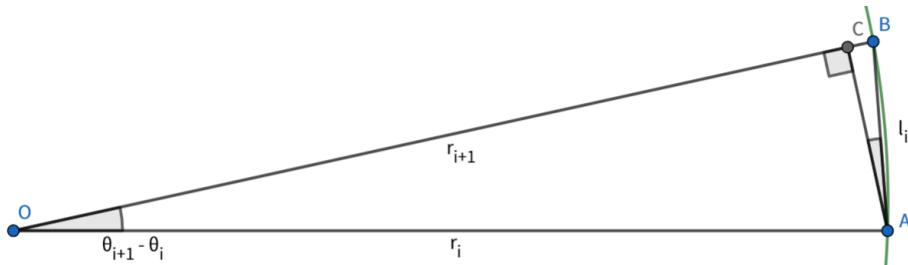


图 2 微元法近似处理

为了选取合适的迭代初值, 我们在此提出一种估计 $|\theta_i - \theta_{i+1}|$ 的方法。即利用 l_i 远小于 r_i 处曲率半径的特点, 将 r_i 附近的一段等距螺线以及长为 l_i 的板凳都近似处理为

小线段，如图 2。使用微元法分析，得到如下关系

$$\left\{ \begin{array}{l} AB \approx l_i \\ AO = r_i \\ \tan \angle BAC = \frac{D}{2\pi} \\ AC = AB \cos \angle BAC \\ |\theta_i - \theta_{i+1}| \approx \frac{AC}{AO} \end{array} \right. \quad (13)$$

联立得

$$|\theta_i - \theta_{i+1}| \approx \frac{2\pi l_i}{D\sqrt{\theta_i^2 + 1}} \quad (14)$$

因此可以得到 Halley 方法用 θ_i 求解 θ_{i+1} 的迭代初值

$$\theta_{i+1}^{(0)} = \theta_i \pm \frac{2\pi l_i}{D\sqrt{\theta_i^2 + 1}} \quad (15)$$

盘入时取“+”，在盘出时取“-”。在本题情景下，该估计的误差在 10^{-3} rad 以下。

至此，我们获得了从 θ_i 到 θ_{i+1} 的映射关系。再结合由式 (3) 解出的 θ_0 ，即可递推得到全部的 $\{\theta_i\}_{i=0}^{223}$ ，代入式 (1) 和 (4) 即可计算出板凳龙上全部把手的位置矢量 $\{\mathbf{r}_i\}_{i=0}^{223}$ 。

5.1.2 问题一：舞龙队位置的求解

上文中，我们建立了在不同的时间 t 下计算舞龙队每一个把手中心的位置的模型，将求解的结果以保留小数点后六位的形式保存在文件 result1.xlsx 中。表 2 为龙头前把手、第 1、51、101、151、201 节龙身前把手和龙尾后把手的位置表。

表 2 龙头前把手、多节龙身前把手和龙尾后把手的位置表

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 x (m)	8.800000	5.799209	-4.084887	-2.963609	2.594494	4.420274
龙头 y (m)	0.000000	-5.771092	-6.304479	6.094780	-5.356743	2.320429
第 1 节龙身 x (m)	8.363824	7.456758	-1.445473	-5.237118	4.821221	2.459489
第 1 节龙身 y (m)	2.826544	-3.440399	-7.405883	4.359627	-3.561949	4.402476
第 51 节龙身 x (m)	-9.518732	-8.686317	-5.543150	2.890455	5.980011	-6.301346
第 51 节龙身 y (m)	1.341137	2.540108	6.377946	7.249289	-3.827758	0.465829
第 101 节龙身 x (m)	2.913983	5.687116	5.361939	1.898794	-4.917371	-6.237722
第 101 节龙身 y (m)	-9.918311	-8.001384	-7.557638	-8.471614	-6.379874	3.936008
第 151 节龙身 x (m)	10.861726	6.682311	2.388757	1.005154	2.965378	7.040740

接下页

续上页表 2

	0 s	60 s	120 s	180 s	240 s	300 s
第 151 节龙身 y (m)	1.828753	8.134544	9.727411	9.424751	8.399721	4.393013
第 201 节龙身 x (m)	4.555102	-6.619664	-10.627211	-9.287720	-7.457151	-7.458662
第 201 节龙身 y (m)	10.725118	9.025570	1.359847	-4.246673	-6.180726	-5.263384
龙尾 (后) x (m)	-5.305444	7.364557	10.974348	7.383896	3.241051	1.785033
龙尾 (后) y (m)	-10.676584	-8.797992	0.843473	7.492370	9.469336	9.301164

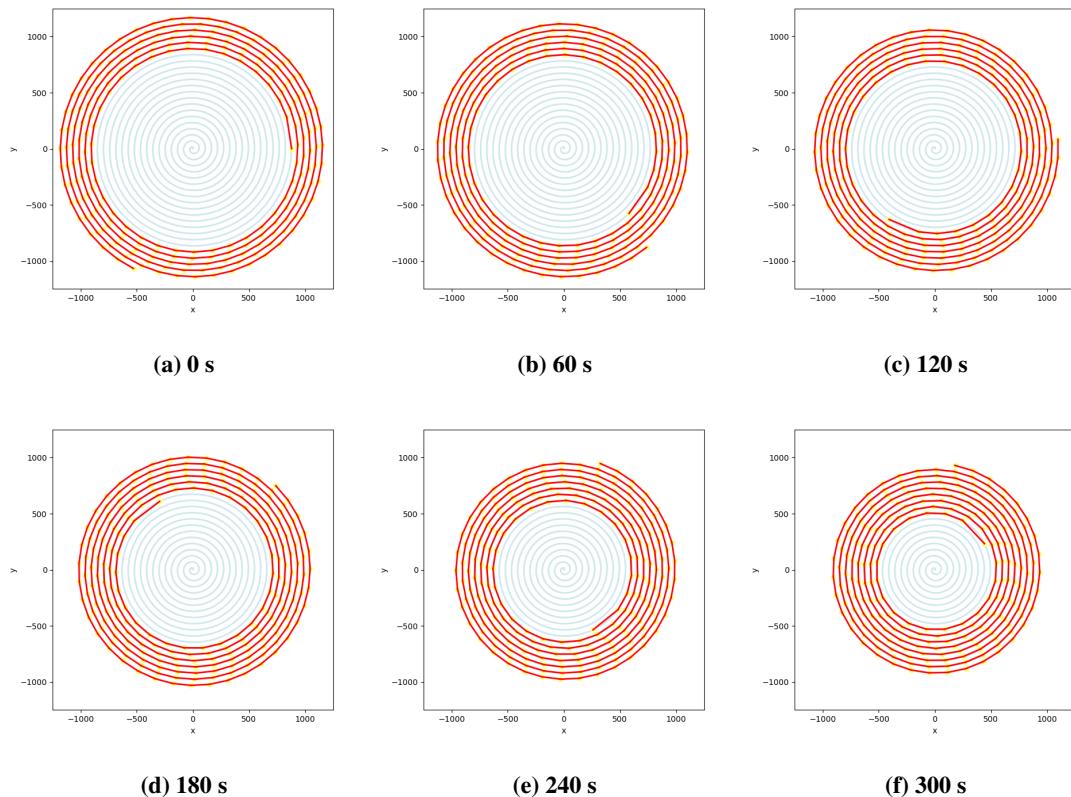


图 3 多时刻下舞龙队的位置示意图

5.1.3 问题一：舞龙队速度模型的建立

在舞龙队速度模型中，我们将每个板凳视为矩形理想刚体，利用刚体运动学，平面几何学我们可以很容易地用 θ_k 和 θ_{k+1} 表示出 v_k 与 v_{k+1} 的方向及 l_k ，表达式如下，具体推导见附录 B

$$\frac{v_k}{\|v_k\|} = \frac{\begin{bmatrix} \theta_k \sin \theta_k - \cos \theta_k \\ -\theta_k \cos \theta_k - \sin \theta_k \end{bmatrix}}{\sqrt{\theta_k^2 + 1}} \quad (16)$$

$$l_k = l_k \begin{bmatrix} \theta_{k+1} \cos \theta_{k+1} - \theta_k \cos \theta_k \\ \theta_{k+1} \sin \theta_{k+1} - \theta_k \sin \theta_k \end{bmatrix} \quad (17)$$

联立式(6), (16), (33)和(36), 且已知 $v_0 = 100 \text{ cm/s}$ 与 t 时刻下的 $\{\theta_i\}_{i=0}^{223}$, 即可求解出 t 时刻下每个把手的瞬时速率 $\{v_i\}_{i=0}^{223}$ 。

5.1.4 问题一：舞龙队速度的求解

上文中, 我们建立了在不同的时间 t 下计算舞龙队每一个把手中心的速度大小的模型, 将求解的结果以保留小数点后六位的形式保存在文件result1.xlsx中。表3为龙头前把手、龙头后面第1、51、101、151、201节龙身前把手和龙尾后把手的速度, 速度方向均沿着等距螺线的切线方向。

表3 龙头前把手、多节龙身前把手和龙尾后把手的速率表

	0 s	60 s	120 s	180 s	240 s	300 s
龙头 (m/s)	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
第1节龙身 (m/s)	0.999971	0.999961	0.999945	0.999917	0.999859	0.999709
第51节龙身 (m/s)	0.999742	0.999662	0.999538	0.999331	0.998941	0.998065
第101节龙身 (m/s)	0.999575	0.999453	0.999269	0.998971	0.998435	0.997302
第151节龙身 (m/s)	0.999448	0.999299	0.999078	0.998727	0.998115	0.996861
第201节龙身 (m/s)	0.999348	0.999180	0.998935	0.998551	0.997894	0.996574
龙尾 (后) (m/s)	0.999311	0.999136	0.998883	0.998489	0.997816	0.996478

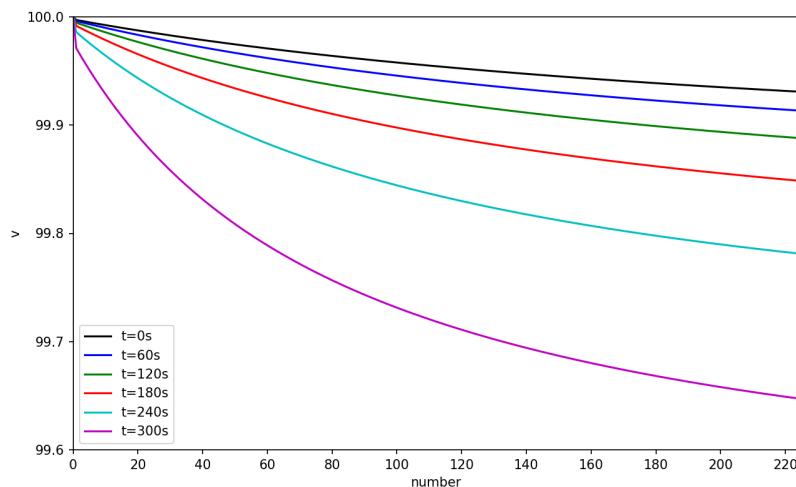


图4 不同时刻下舞龙队各位置的速率图

我们将0 s、60 s、120 s、180 s、240 s、300 s时整个舞龙队的速度大小呈现在图4中。

可以发现，在盘入过程中把手的速率由内圈向外圈速率逐渐减慢，并且随着时间的增加，龙头不断向更内圈行进，速率下降的趋势越发明显。

5.2 问题二模型的建立及求解

5.2.1 问题二：碰撞判定模型

如图 5，若矩形 $A_1B_1C_1D_1$ 与矩形 $A_2B_2C_2D_2$ 发生“碰撞”，则至少有一个矩形的顶点位于另一矩形的内部（图 5(a)）或边界（图 5(b)）。我们可以用向量叉乘判定点与直线的位置关系。设 \mathbf{k} 是垂直于纸面向外的 z 方向基矢，如图有

$$\overrightarrow{D_1C_1} \times \overrightarrow{D_1A_2} \cdot \mathbf{k} \sim \begin{cases} > 0 & , \text{若 } A_2 \text{ 在 } \overrightarrow{D_1C_1} \text{ 左侧} \\ = 0 & , \text{若 } A_2 \text{ 在 } \overrightarrow{D_1C_1} \text{ 上或其延长线上} \\ < 0 & , \text{若 } A_2 \text{ 在 } \overrightarrow{D_1C_1} \text{ 右侧} \end{cases} \quad (18)$$

由此只要循环遍历依次判定一个矩形的 4 个顶点是否位于另一矩形的四条边的内侧即可判定两矩形是否碰撞。相同的相交判定算法已被非常多的代码库实现，例如 Python 中的 Matplotlib 库就提供了 `intersection` 函数，用于判定矩形与矩形是否有交集（相撞），且精度很高。因此，本研究将主要使用该函数来判定矩形之间的碰撞。

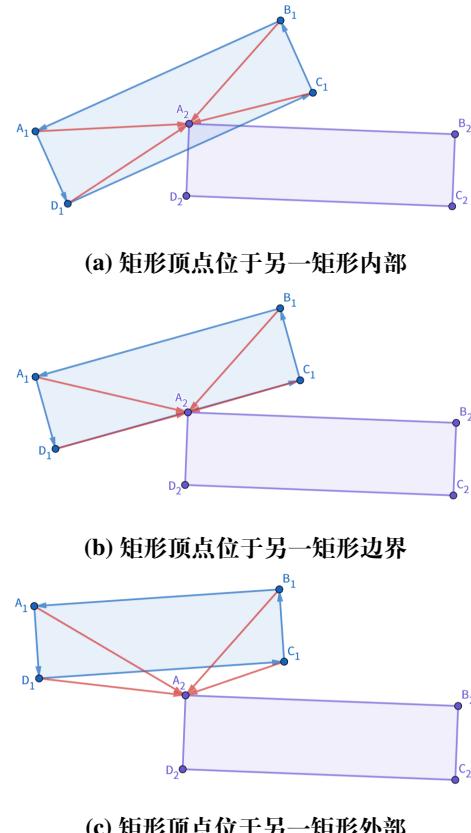


图 5 碰撞判定的 3 种情形

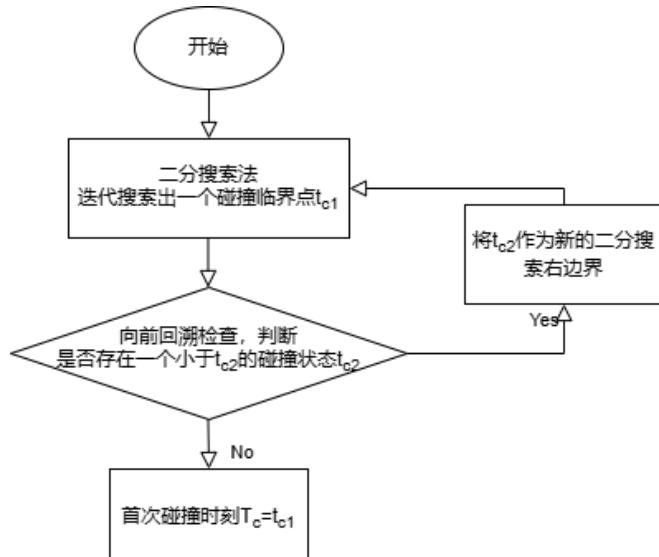


图 6 首次碰撞时刻 T_c 求解流程图

5.2.2 问题二：首次碰撞时刻的搜索模型

由位置和速度模型，当前任务聚焦于首次碰撞时刻 T_c 的求解策略。假设舞龙队首次碰撞后一直保持碰撞状态，则可采用二分搜索法在预设的时间区间内迭代搜索出一个碰撞临界点 t_{c1} 。同时考虑可能存在的“碰撞-脱离-碰撞”情况，我们加入一个检验过程以确保最后找到的碰撞临界值 t_{c1} 确为首次碰撞时间，时间步长为 0.05s。

5.2.3 问题二：首次碰撞时刻及此刻运动状态的求解

前文中，我们利用 Matplotlib 库与 intersection 函数建立了碰撞判定模型和求解首次碰撞时刻 T_c 的模型，在 10^{-5} s 的时间分辨率下，得出 412.47383 s 时恰好不碰撞，412.47384 s 时发生第一次碰撞，即终止时刻 $T_c = 412.47383$ s。

再根据问题一的舞龙队的位置与速度模型即可求得 T_c 时舞龙队的位置与速度，将结果以保留六位小数的形式保存在文件 result2.xlsx 中。表 4 为此时龙头前把手、龙头后面第 1、51、101、151、201 条龙身前把手和龙尾后把手的位置和速度。

表 4 首次碰撞时龙头前把手、多条龙身前把手和龙尾后把手的位置和速度

	x (m)	y (m)	v (m/s)
龙头	1.209925	1.753394	1.000000
第 1 节龙身	-1.643797	0.310577	0.991551
第 51 节龙身	1.281194	4.530388	0.976858
第 101 节龙身	-0.536239	-5.823663	0.974550
第 151 节龙身	0.968848	-6.559643	0.973608
第 201 节龙身	-7.893160	-2.830649	0.973096
龙尾 (后)	0.956210	0.000000	0.972938

5.3 问题三模型的建立及求解

5.3.1 问题三：最小螺距优化模型的建立

问题三需求解能使舞龙队盘入至调头空间边界处，期间不发生碰撞的最小螺距，即

$$\min D \quad \text{s.t. 不碰撞, } \forall \theta_0 > \frac{2\pi R}{D} \quad (19)$$

其中 D 表示螺距， R 表示调头空间半径 (4.5 m)， $\frac{2\pi R}{D}$ 即为调头空间边界对应的极角。

对于一个确定的 (D, θ_0) 数组，我们可以使用问题一中的舞龙队位置模型求得此时整个舞龙队的位置，接着使用问题二中的碰撞判断模型检验该位置下是否产生碰撞。因此问题就转化为：在 $\left\{ (D, \theta_0) \left| \begin{array}{l} D > 0 \\ \theta_0 > \frac{2\pi R}{D} \end{array} \right. \right\}$ 的空间内搜索最优 (D, θ_0) 的问题。

然而这个搜索空间开销过大，并不实用。基于这一想法，我们关注到以下若干事实：

- 龙头最容易与倒数第二圈的龙身碰撞；
- 通过问题二的求解过程得知，当螺距 $D = 55 \text{ cm}$ 时，碰撞发生在 $r(\theta) = 2.288743 \text{ m}$ 的位置，显然在调头空间以外没有发生碰撞，故 D 的上界可确定为 55 cm ；
- 在 $\theta_0 = \frac{2\pi R}{D}$ 的位置上，用二分法搜索恰好碰撞时的螺距²，得到 $D = 42.021 \text{ cm}$ ，由此获得 D 的下界；
- 随着 D 增大，发生碰撞时的 θ_0 大致呈减小趋势，故我们可以只关心 $\frac{2\pi R}{D}$ 附近的一段 θ 区间，例如 $(\frac{2\pi R}{D}, \frac{2\pi R}{D} + 3\pi)$ 。

至此，我们将搜索空间精确到了 $\left\{ (D, \theta_0) \middle| \begin{cases} 42 \text{ cm} < D < 55 \text{ cm} \\ \frac{2\pi R}{D} < \theta_0 < \frac{2\pi R}{D} + 3\pi \end{cases} \right\}$ 。

再定义目标函数 $\text{check}(D, \theta_0)$

$$\text{check}(D, \theta_0) = \begin{cases} D & , (D, \theta_0) \text{ 处不发生碰撞} \\ 1 \times 10^6 \text{ cm} & , (D, \theta_0) \text{ 处发生碰撞} \end{cases} \quad (20)$$

那么式 (19) 描述的优化问题可以转化为

$$\min (\text{check}(D, \theta_0)) \quad \text{s.t.} \quad \begin{cases} 42 \text{ cm} < D < 55 \text{ cm} \\ \frac{2\pi R}{D} < \theta_0 < \frac{2\pi R}{D} + 3\pi \end{cases} \quad (21)$$

可以用模拟退火算法 [3] 搜索最佳的 (D, θ_0) ，求解 $\min D$ 。

另外，我们可以对 check 进行一定的变换作为目标函数，检验该算法的稳定性。分析表 5 中的数据发现该算法只能精确到 10^{-3} ，结果稳定性达不到精度要求。因此我们在模拟退火算法的基础上采取自适应网格搜索法 [4]，进一步提高结果的精度。

首先使用模拟退火算法求解出 100 个可能解，以这一百个点的最大螺距 D_{max} 与最小螺距 D_{min} 作为 D 的搜索上下界。由于 θ_0 对目标函数 check 的影响很不灵敏，因此由 3σ 原则取得 θ_0 的搜索上下界，作为自适应搜索算法的首次搜索空间。如图 7 所示，每一次搜索都可以得到一个估计解 (d_{min}, θ_{0min}) ，若该点左右两侧的第一个不发生碰撞的点有 θ_1 和 θ_2 ，该次搜索在 D 方向上的精度为 d ，则进行下一次搜索的搜索空间为

$$\left\{ (D, \theta_0) \middle| \begin{cases} (d_{min} - d) < D < (d_{min} + d) \\ \theta_1 < \theta_0 < \theta_2 \end{cases} \right\}$$

在 D 方向上的搜索精度为更新 $\frac{d}{10}$ ， θ_0 方向上的搜索精度均取 50 等分。如此不断迭代直至 D 的精确度达到 10^{-6} ，可以得到一个更为精确的最紧螺距解。

² 代码见附录章节 5.2 `check_3.py`

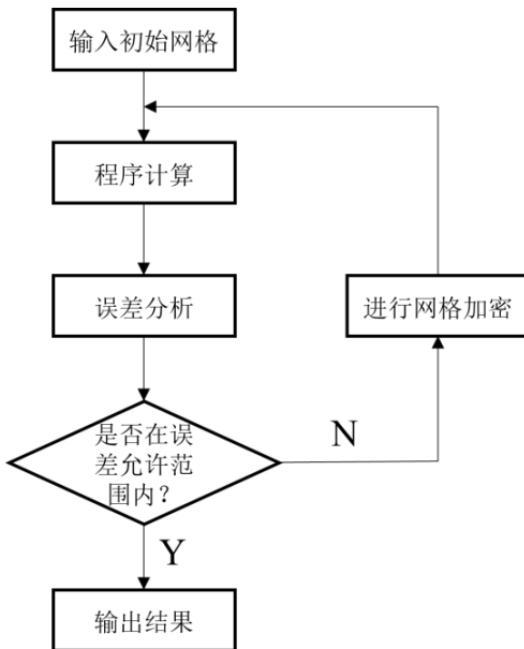


图 7 自适应网格技术的实现流程

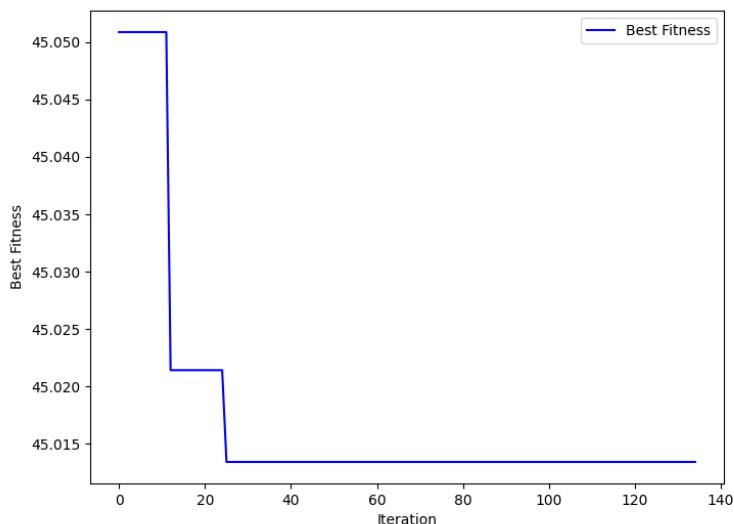


图 8 模拟退火算法的学习曲线

5.3.2 问题三：最小螺距的求解

根据前文建立的最小螺距优化模型，最终求得 $\min D = 0.450337 \text{ cm}$ 。图 8 为模拟退火算法的学习曲线，图 9 展示出了自适应网格搜索法的部分扫描流程。

表 5 模拟退火稳定性检验表

变化方式	D (cm)
原函数	45.00729
开根号	45.03367
平方	45.00756
缩小 1000 倍	45.02214

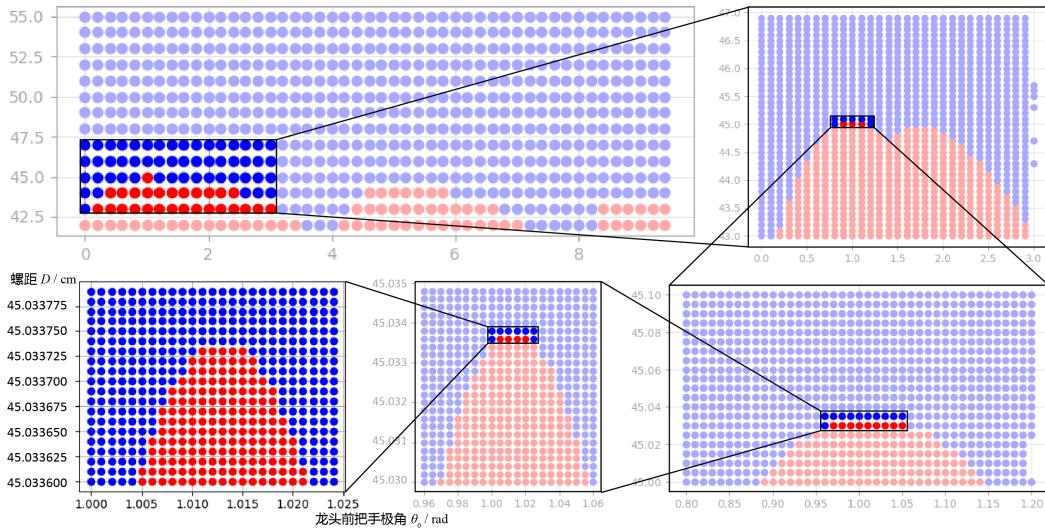


图 9 问题三自适应网格搜索法的部分扫描流程

5.4 问题四模型的建立及求解

5.4.1 问题四：复合路径参数方程模型的建立

题目所述的调头路径是由两段圆弧构成的 S 形复合路径，且要求前一段圆弧的半径是后一段的 2 倍，路径与盘入、盘出螺线均相切。我们构建了该路径的平面几何模型，以证明不能通过优化圆弧获得更短的调头路径（定理 1）。

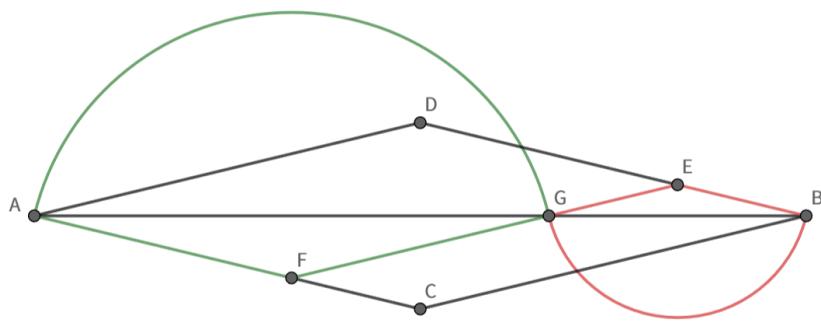


图 10 调头路径示意图

定理 1 不能通过优化圆弧获得更短的调头路径。

证明 1 如图 10, 假设调头空间的入点为 A 、出点为 B , $CD \perp AB$, AC 垂直于 A 点速度矢量, BD 垂直于 B 点速度矢量, $\widehat{AG'}$ 、 $\widehat{BG'}$ 是两段符合要求的圆弧路径, 相切于 G' 点 (图中未画出), 半径分别为 R_1 、 R_2 。

由中心对称的性质易知 $AC \parallel BD$, 亦可知 $ACBD$ 构成菱形, $\angle BAC = \angle ABD$ 。由于圆弧必须与 A 、 B 点速度相切, 故 $\widehat{AG'}$ 、 $\widehat{BG'}$ 的圆心 F 、 E 应分别位于 AC 、 BD 上。在

AB 上寻找一点 G , 使得 $\frac{AG}{BG} = \frac{R_1}{R_2}$ 。由于 $\begin{cases} \frac{AF}{BE} = \frac{AG}{BG} = \frac{R_1}{R_2} \\ \angle FAG = \angle EBG \end{cases}$, 可得 $\triangle FAG \sim \triangle EBG$ 。

由此可知, E 、 G 、 F 三点共线, 且 $AD \parallel EF \parallel BC$, $\frac{FG}{EF} = \frac{R_1}{R_1+R_2}$, $\frac{EG}{EF} = \frac{R_2}{R_1+R_2}$ 。又由于两圆弧相切, 所以圆心距 $EF = R_1 + R_2$, 即 $FG = R_1$, $EG = R_2$ 。由此可知 G 与 G' 为同一点, 由相似三角形可知圆心角 $\angle AFG = \angle BEG$, 又由平行四边形的性质可知 $\angle AFG = \angle BEG = \angle ACB$ 。最后计算出总弧长 $s = R_1 \cdot \angle AFG + R_2 \cdot \angle BEG = AC \cdot \angle ACB$, 它与 R_1 、 R_2 的选取无关。因此, 不能通过优化圆弧获得更短的调头路径。

为了描述舞龙队从盘入到调头再到盘出的完整运动过程, 我们建立了盘入、盘出螺线与调头路径的复合路径的参数方程 $\mathbf{r} = \mathbf{r}(\xi)$, ξ 为参数, 取值范围是全体实数。下面根据 4 段不同的路径, 分段构建该参数方程。

1. 盘入螺线, $\xi \leq 0$, 有

$$\mathbf{r}(\xi) = \frac{D}{2\pi} \theta_{in} \begin{bmatrix} \cos \theta_{in} \\ \sin \theta_{in} \end{bmatrix} \quad (22)$$

其中 $\theta_{in} = -\xi + \frac{2\pi R}{D}$, 它表示盘入螺线上的一点的极角。 R 是调头空间的半径 (4.5m)。

2. 第一段圆弧, $0 \leq \xi \leq \frac{4}{3}\xi_0$, 其中定义了新常量 $\xi_0 = \arctan \frac{2\pi R}{D}$ 。此时有

$$\mathbf{r}(\xi) = \begin{bmatrix} \cos\left(\frac{2\pi R}{D} - \frac{3\pi}{2}\right) & -\sin\left(\frac{2\pi R}{D} - \frac{3\pi}{2}\right) \\ \sin\left(\frac{2\pi R}{D} - \frac{3\pi}{2}\right) & \cos\left(\frac{2\pi R}{D} - \frac{3\pi}{2}\right) \end{bmatrix} (\mathbf{r}_{c1} + \mathbf{r}_{r1}) \quad (23)$$

其中 \mathbf{r}_{c1} 是第一段圆弧的弧心的位矢, \mathbf{r}_{r1} 是该弧上一点相对弧心的位矢。两者分别定义为

$$\mathbf{r}_{c1} = \begin{bmatrix} \frac{D}{3\pi} \\ -\frac{R}{3} \end{bmatrix} \quad \mathbf{r}_{r2} = \begin{bmatrix} \cos(-\alpha_1) & -\sin(-\alpha_1) \\ \sin(-\alpha_1) & \cos(-\alpha_1) \end{bmatrix} \begin{bmatrix} -\frac{D}{3\pi} \\ -\frac{2R}{3} \end{bmatrix}$$

其中 α_1 是弧上一点对应的圆心角, 定义为 $\alpha_1 = \frac{3}{2}\xi$ 。

3. 第二段圆弧, $\frac{4}{3}\xi_0 \leq \xi \leq 2\xi_0$, 有

$$\mathbf{r}(\xi) = \begin{bmatrix} \cos\left(\frac{2\pi R}{D} - \frac{3\pi}{2}\right) & -\sin\left(\frac{2\pi R}{D} - \frac{3\pi}{2}\right) \\ \sin\left(\frac{2\pi R}{D} - \frac{3\pi}{2}\right) & \cos\left(\frac{2\pi R}{D} - \frac{3\pi}{2}\right) \end{bmatrix} (\mathbf{r}_{c2} + \mathbf{r}_{r2}) \quad (24)$$

其中 \mathbf{r}_{c2} 是第二段圆弧的弧心的位矢, \mathbf{r}_{r2} 是该弧上一点相对弧心的位矢。两者分别定义为

$$\mathbf{r}_{c2} = \begin{bmatrix} -\frac{D}{6\pi} \\ \frac{2R}{3} \end{bmatrix} \quad \mathbf{r}_{r2} = \begin{bmatrix} \cos(\alpha_2 - 2\xi_0) & -\sin(\alpha_2 - 2\xi_0) \\ \sin(\alpha_2 - 2\xi_0) & \cos(\alpha_2 - 2\xi_0) \end{bmatrix} \begin{bmatrix} \frac{D}{6\pi} \\ \frac{R}{3} \end{bmatrix}$$

其中 α_2 是弧上一点对应的圆心角, 定义为 $\alpha_2 = 3\xi - 4\xi_0$ 。

4. 盘出螺线, $\xi \geq 2\xi_0$, 有

$$\mathbf{r}(\xi) = \frac{D}{2\pi} (\theta_{\text{out}} - \pi) \begin{bmatrix} \cos \theta_{\text{out}} \\ \sin \theta_{\text{out}} \end{bmatrix} \quad (25)$$

其中 $\theta_{\text{out}} = \xi - 2\xi_0 + \frac{2\pi R}{D} + \pi$, 它表示盘出螺线上的一点的极角。

可以证明, 该参数方程在分段点上的导数连续, 即 $\mathbf{r}(\xi) \in C^1(\mathbb{R})$ 的, 故我们可以部署 Halley 算法。只要得知龙头的位矢 \mathbf{r}_0 和各个板凳长度 $\{l_i\}_{i=0}^{222}$, 代入刚体运动学方程 (式(7)), 依次求出各个把手所对应的参数 $\{\xi_i\}_{i=1}^{223}$ 的数值解, 进而确定各个把手的位置 $\{\mathbf{r}_i\}_{i=1}^{223}$ 。

5.4.2 问题四：复合路径速度模型的建立

由问题一建立的舞龙队速度模型的式 (16), 可以变形得到盘入、盘出螺线上 $\frac{\mathbf{v}}{\|\mathbf{v}\|}$ 与 $\theta_{\text{in, out}}$ 的关系。再由复合路径参数方程模型中引入的 $\mathbf{r}_{r1}, \mathbf{r}_{r2}$, 可以导出圆周运动阶段的速度方向角, 进而得到 $\frac{\mathbf{v}}{\|\mathbf{v}\|}$ 与 $\alpha_{1,2}$ 的关系。综合得出

$$\frac{\mathbf{v}}{\|\mathbf{v}\|} = \begin{cases} \frac{1}{\sqrt{\theta_{\text{in}}^2 + 1}} \begin{bmatrix} \theta_{\text{in}} \sin \theta_{\text{in}} - \cos \theta_{\text{in}} \\ -\theta_{\text{in}} \cos \theta_{\text{in}} - \sin \theta_{\text{in}} \end{bmatrix}, & \xi \leq 0 \\ \begin{bmatrix} \cos \left(\frac{2\pi R}{D} - \pi + \xi_0 - \alpha_1 \right) \\ \sin \left(\frac{2\pi R}{D} - \pi + \xi_0 - \alpha_1 \right) \end{bmatrix}, & 0 \leq \xi \leq \frac{4}{3}\xi_0 \\ \begin{bmatrix} \cos \left(\frac{2\pi R}{D} - \pi - \xi_0 + \alpha_2 \right) \\ \sin \left(\frac{2\pi R}{D} - \pi - \xi_0 + \alpha_2 \right) \end{bmatrix}, & \frac{4}{3}\xi_0 \leq \xi \leq 2\xi_0 \\ \frac{1}{\sqrt{(\theta_{\text{out}} - \pi)^2 + 1}} \begin{bmatrix} (\theta_{\text{out}} - \pi) \sin \theta_{\text{out}} - \cos \theta_{\text{out}} \\ -(\theta_{\text{out}} - \pi) \cos \theta_{\text{out}} - \sin \theta_{\text{out}} \end{bmatrix}, & \xi \geq 2\xi_0 \end{cases} \quad (26)$$

最后利用问题一中的速度模型迭代解出各把手的速度 $\{\mathbf{v}_i\}_{i=0}^{223}$ 。

5.4.3 问题四模型的求解

我们根据参数方程模型使用用参数 ξ 唯一确定路径上一点的位置。再使用问题一中的舞龙队位置与速度模型, 以调头开始时间为零时刻, 求解出了从 -100 s 至 100 s 内每个时刻下舞龙队的位置与速度。结果保存在文件result4.xlsx中, 精度 10^{-6} 。图 11展示了掉头过程中 0 s、 50 s、 100 s 时整个舞龙队的速度矢量图。表 6 和 7 为 -100 s、 -50 s、 0 s、 50 s、 100 s 时, 龙头前把手、龙头后面第 $1, 51, 101, 151, 201$ 节龙身前把手和龙尾后把手的位置和速度。

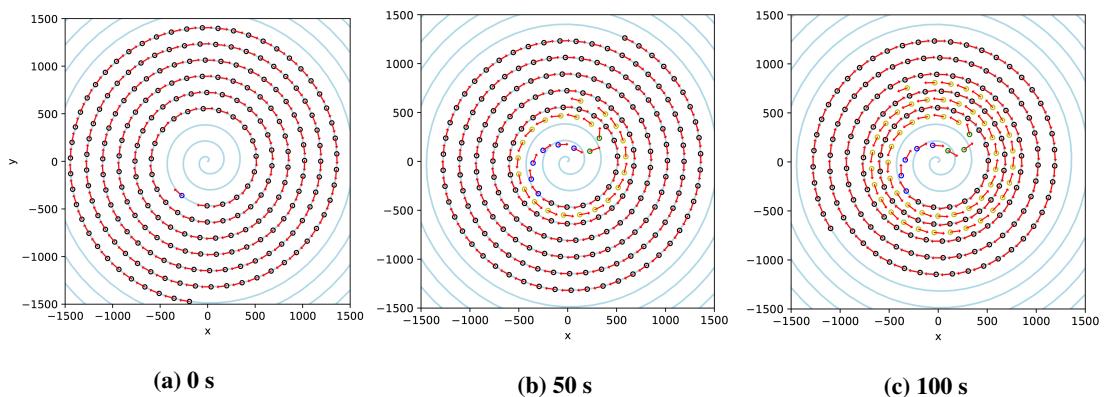


图 11 调头过程不同时刻位置和速度矢量图

表 6 调头过程中龙头前把手、多节龙身前把手与龙尾后把手的位置表

	-100 s	-50 s	0 s	50 s	100 s
龙头 x (m)	7.778034	6.608301	-2.711856	1.332696	-3.157229
龙头 y (m)	3.717164	1.898865	-3.591078	6.175324	7.548511
第 1 节龙身 x (m)	6.209269	5.366911	-0.063534	3.862265	-0.346887
第 1 节龙身 y (m)	6.108526	4.475404	-4.670888	4.840827	8.079166
第 51 节龙身 x (m)	-10.608982	-3.629353	2.461174	-1.670204	2.095838
第 51 节龙身 y (m)	2.828273	-8.964059	-7.777806	-6.077093	4.033306
第 101 节龙身 x (m)	-11.918906	10.130039	3.005363	-7.593129	-7.289467
第 101 节龙身 y (m)	-4.812549	-5.965397	10.109558	5.173677	2.061724
第 151 节龙身 x (m)	-14.349690	12.978094	-7.012883	-4.598596	9.463798
第 151 节龙身 y (m)	-1.992233	-3.799850	10.330952	-10.390087	-3.537182
第 201 节龙身 x (m)	-11.961633	10.531092	-6.883745	0.349759	8.515379
第 201 节龙身 y (m)	10.557489	-10.799357	12.376820	-13.177539	8.616238
龙尾 (后) x (m)	-0.996849	0.177279	-1.921049	5.847439	-10.973564
龙尾 (后) y (m)	-16.528669	15.720950	-14.715010	12.618580	-6.781206

表 7 调头过程中龙头前把手、多节龙身前把手与龙尾后把手的速度表

	-100 s	-50 s	0 s	50 s	100 s
龙头 (m/s)	1.000000	1.000000	1.000000	1.000000	1.000000
第 1 节龙身 (m/s)	0.999904	0.999762	0.998687	1.000363	1.000124
第 51 节龙身 (m/s)	0.999346	0.998641	0.995134	0.949795	1.003966

接下页

续上页表 7

	-100 s	-50 s	0 s	50 s	100 s
第 101 节龙身 (m/s)	0.999091	0.998641	0.994447	0.948342	1.096145
第 151 节龙身 (m/s)	0.998944	0.998047	0.994155	0.947898	1.095188
第 201 节龙身 (m/s)	0.998849	0.997925	0.993994	0.947683	1.094815
龙尾 (后) (m/s)	99.881721	99.788517	99.394342	94.762021	109.471504

算法 1 三分查找法

输入: 目标函数 $f(\cdot) = \max_i \frac{v_i(\cdot)}{v_0}$ 、初始左边界 t_{left} 、初始右边界 t_{right} 、容许误差 ε

输出: 目标函数的最大值点 t_{end} 和最大值 $f(t_{\text{end}})$

```

1: while  $t_{\text{right}} - t_{\text{left}} > \varepsilon$  do
2:    $t_{\text{mid},1} \leftarrow (2t_{\text{left}} + t_{\text{right}})/3$ 
3:    $t_{\text{mid},2} \leftarrow (t_{\text{left}} + 2t_{\text{right}})/3$ 
4:    $f_1 \leftarrow f(t_{\text{mid},1})$ 
5:    $f_2 \leftarrow f(t_{\text{mid},2})$ 
6:   if  $f_1 < f_2$  then
7:      $t_{\text{left}} \leftarrow t_{\text{mid},1}$ 
8:   else
9:      $t_{\text{right}} \leftarrow t_{\text{mid},2}$ 
10:  end if
11: end while
12:  $t_{\text{end}} \leftarrow t_{\text{left}}$ 
13: return  $\{t_{\text{end}}, f(t_{\text{end}})\}$ 

```

5.5 问题五模型的建立及求解

5.5.1 问题五：三分查找模型的建立

问题五中，要使行进过程中舞龙队的全部把手中心的最大速度不超过 $v_{\text{limit}} = 2 \text{ m/s}$ ，求解龙头的行进速度。根据问题四建立的复合路径参数方程模型，我们可以求出 t 时刻第 i 个把手的速度 v_i 与龙头速度 v_0 的比值，遍历全部 224 个把手可以求出最大比值 $\max_i \frac{v_i}{v_0} = \max_i \frac{v_i(t)}{v_0}$ ，它是时刻 t 的函数。

用 $\sup_t \max_i \frac{v_i(t)}{v_0}$ 表示行进过程中，该函数在 $t \in [-100 \text{ s}, 100 \text{ s}]$ 范围内的上界。可以使用三分查找法来估计此上界。

值得关注的是初始左、右边界的选取。三分查找法要求目标函数在搜索范围内

具有唯一极大值，否则不保证结果的正确性。因此需要利用问题四中获得的速度表（result4.xlsx），先找到最大值出现的大致范围，再调用三分查找法。

获得 $\sup_t \max_i \frac{v_i(t)}{v_0}$ 后，可以算出龙头的最大行进速度

$$v_{0\ max} = \frac{v_{\text{limit}}}{\sup_t \max_i \frac{v_i(t)}{v_0}} \quad (27)$$

5.5.2 问题五：三分查找模型的求解与检验

前文提到，在三分查找之前，要确定 t 的查找范围，以满足目标函数单极值条件。我们对图 4 所示的盘入过程中舞龙队各部分速率的变化趋势进行分析，发现盘入时龙头的速率始终大于其他任意把手，因此 $\sup_t \max_i \frac{v_i(t)}{v_0}$ 一定不会落在盘入阶段。对问题四中获得的表 7 做进一步数据处理，我们发现 $\max_i \frac{v_i(t)}{v_0}$ 在 $t \in [0 \text{ s}, 20 \text{ s}]$ 范围内明显有最高峰值，如图 12，因此在此范围内应用三分查找法。查找历程如图 13，最终得到 $t_{\text{end}} = 14.4799708 \text{ s}$, $\sup_t \max_i \frac{v_i(t)}{v_0} = 1.604793$, $v_{0\ max} = 1.246267 \text{ m/s}$ 。

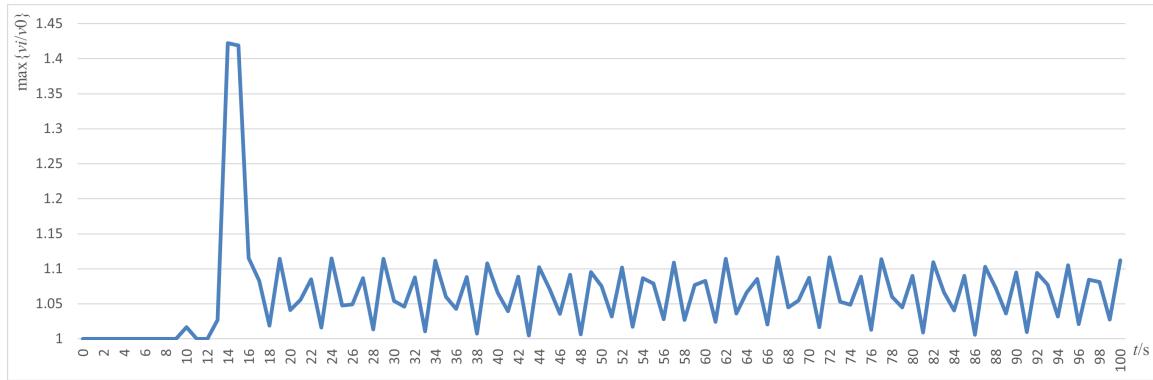


图 12 问题四中获得的 $\max_i \frac{v_i(t)}{v_0}$

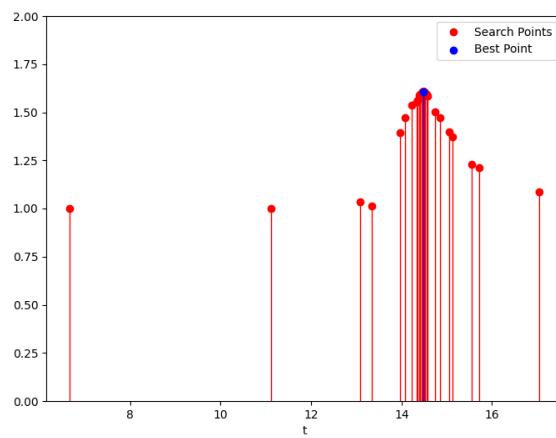


图 13 [0 s, 20 s] 的三分查找历程

六、模型的评价与改进

6.1 模型的优点

1. 本模型对于非线性方程的数值解采用 Halley 算法，提高了迭代求解的收敛速度；
2. 本模型改进了模拟退火算法，解决了不同维度自变量变化灵敏度不同导致的难以收敛的现象；
3. 本模型结合了模拟退火算法和自适应网格搜索算法的优点，提高了求解速度和求解精度；
4. 本模型对模拟退火算法进行了稳定性和灵敏度检验，证实了模型的不适用性；
5. 本模型在分析复合路径时，引入了全程统一的参数来构建参数方程模型，使得模型的表达更为简洁通用；
6. 本模型构建的参数方程是一阶连续可导的，使得相关方程的求解更简便，例如可以部署 Halley 方法迭代求解；
7. 本模型自行设计了目标函数，并应用智能优化算法，规避了二分查找或遍历查找的一部分工作量。

6.2 模型的缺点

1. 问题三的二维搜索空间、问题五的一维变量空间的维数太小，导致主流的智能优化算法并不是特别高效适用；
2. 问题二的仍然部分采用了二分查找“暴力搜索”的算法，运算量巨大，仍有改进空间。

6.3 已知的改进方向

1. 确定问题二的自变量搜索空间时，还可以通过分析板凳龙运动的包络线，来构造碰撞的必要条件，以建立更强的约束。

参考文献

- [1] On the United Theory of the Family of Euler-Halley Type Methods with Cubical Convergence in Banach Spaces [J]. *Journal of Computational Mathematics*, 2003, (02): 195-200.
- [2] 刘希普. 关于 Halley 迭代方法 [J]. 菏泽师专学报, 1997, (02): 48-51. DOI: 10.16393/j.cnki.37-1436/z.1997.02.010.
- [3] 卢宇婷, 林禹攸, 彭乔姿, 等. 模拟退火算法改进综述及参数探究 [J]. 大学数学, 2015, 31(06): 96-103.
- [4] Babuška I, Rheinboldt W C. Aposteriori error estimates for the finite element method[J]. International Journal for Numerical Methods in Engineering, 1978, 12(10).

附录 A Halley 算法具体公式推导

在迭代过程开始前，我们需要将式 (3) 改写为关于 θ_0 的目标函数 $f(\theta_0)$ ，即

$$f(\theta_0) = s(\theta_0) - s(\theta_{\text{init}}) + v_0 t \quad (28)$$

迭代过程从选取迭代初值 $\theta_0^{(0)}$ 开始，由于 $s(\theta_0)$ 与 $f(\theta_0)$ 是严格单调递增的函数，故 $\theta_0^{(0)}$ 可以任意选取。 $\theta_0^{(0)}$ 确定后，按照以下公式进行迭代

$$\theta_0^{(k+1)} = \theta_0^{(k)} - \frac{f(\theta_0^{(k)})}{f'(\theta_0^{(k)})} \left(1 - \frac{f(\theta_0^{(k)}) f''(\theta_0^{(k)})}{2(f'(\theta_0^{(k)}))^2} \right)^{-1} \quad (29)$$

其中， $f(\theta_0)$ 的一阶导数 $f'(\theta_0)$ 和二阶导数 $f''(\theta_0)$ 分别为

$$f'(\theta_0) = \frac{D}{4\pi} \frac{2\theta_0^2 + 1}{\sqrt{\theta_0^2 + 1}} \quad (30)$$

$$f''(\theta_0) = \frac{D}{4\pi} \frac{2\theta_0^3 + 3\theta_0}{(\theta_0^2 + 1)^{\frac{3}{2}}} \quad (31)$$

迭代直到 $|\theta_0^{k+1} - \theta_0^k|$ 小于预设的收敛阈值（本研究中设定为 10^{-10} ）时停止。当达到此条件时，序列 $\{\theta_0^{(k)}\}$ 可认为收敛至为式 (3) 的根的近似值，即可采用该值为 θ_0 。

附录 B 舞龙队盘入速度模型求解过程

在舞龙队速度模型中，我们将每个板凳视为矩形理想刚体，因此其前、后把手的速度矢量在板凳上的分量应当相等，即对于第 i 条板凳应有递推关系 $\mathbf{v}_i \cdot \mathbf{l}_i = \mathbf{v}_{i+1} \cdot \mathbf{l}_{i+1}$ ，其中 $\mathbf{l}_i = \mathbf{r}_i - \mathbf{r}_{i+1}$ ，为板凳前后把手之间的位矢。移项得

$$\frac{\mathbf{v}_{i+1}}{\mathbf{v}_i} = \frac{\frac{\mathbf{v}_i}{\|\mathbf{v}_i\|} \cdot \mathbf{l}_i}{\frac{\mathbf{v}_{i+1}}{\|\mathbf{v}_{i+1}\|} \cdot \mathbf{l}_{i+1}} \quad (32)$$

其中粗体的 \mathbf{v}_i 表示速度矢量，正常粗细的 v_i 表示速率，即 $v_i = \|\mathbf{v}_i\|$ 。而 $\frac{\mathbf{v}_i}{\|\mathbf{v}_i\|} (\mathbf{v}_i \neq \mathbf{0})$ 表示与 \mathbf{v}_i 平行的单位矢量。由于龙头前把手的速率是已知的，故式 (32) 累积后即可由龙头前把手的速率 v_0 的表示第 i 个把手的速率 v_i ，即

$$v_i = v_0 \prod_{k=0}^{i-1} \frac{\frac{\mathbf{v}_k}{\|\mathbf{v}_k\|} \cdot \mathbf{l}_k}{\frac{\mathbf{v}_{k+1}}{\|\mathbf{v}_{k+1}\|} \cdot \mathbf{l}_k} \quad (33)$$

并且由等距螺线的几何性质，速度方向就是螺线上一点处的切线方向。由于我们研究的是是盘入运动过程，所以速度方向可以通过对式(4)矢量求负导数获得，即

$$\begin{aligned}
 \mathbf{v}_k // -\frac{d\mathbf{r}(\theta_k)}{d\theta} &= -\frac{d}{d\theta}\left(\frac{D\theta_k}{2\pi}\begin{bmatrix} \cos\theta_k \\ \sin\theta_k \end{bmatrix}\right) \\
 &= -\frac{D}{2\pi}\left(\begin{bmatrix} \cos\theta_k \\ \sin\theta_k \end{bmatrix} + \theta_k \begin{bmatrix} -\sin\theta_k \\ \cos\theta_k \end{bmatrix}\right) \\
 &= \begin{bmatrix} \theta_k \sin\theta_k - \cos\theta_k \\ -\theta_k \cos\theta_k - \sin\theta_k \end{bmatrix}
 \end{aligned} \tag{34}$$

至此，我们可以很容易地用 θ_k 和 θ_{k+1} 表示出 \mathbf{v}_k 与 \mathbf{v}_{k+1} 的方向及 \mathbf{l}_k ，表达式如下

$$\frac{\mathbf{v}_k}{\|\mathbf{v}_k\|} = \frac{\begin{bmatrix} \theta_k \sin\theta_k - \cos\theta_k \\ -\theta_k \cos\theta_k - \sin\theta_k \end{bmatrix}}{\sqrt{\theta_k^2 + 1}} \tag{35}$$

$$\mathbf{l}_k = l_k \begin{bmatrix} \theta_{k+1} \cos\theta_{k+1} - \theta_k \cos\theta_k \\ \theta_{k+1} \sin\theta_{k+1} - \theta_k \sin\theta_k \end{bmatrix} \tag{36}$$

附录 C 问题一代码

3.1 save_1.py

```

1 import numpy as np
2 from scipy.optimize import newton
3 import pandas as pd
4 import math
5
6 # 求导
7 def f_0(y, x):
8     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (55/(2*np.pi))**2
9
10 def f_1(y, x):
11     return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
12
13 def f_2(y, x):
14     return 2 - 2*x*np.sin(x-y) - 2*x*y*np.sin(x-y) - 2*x*y*np.cos(x-y)
15
16
17
18 # Halley方法
19 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
20     y = y_initial

```

```

21     for _ in range(max_iter):
22         f0 = f_0(y, x)
23         f1 = f_1(y, x)
24         f2 = f_2(y, x)
25
26         # 第一个板凳长度不一样
27         if i == 0:
28             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 /
29             (55/(2*np.pi))**2
30
31         delta_y = f0/f1*(1-f0*f2/(2*f1**2))
32         y -= delta_y
33
34         if abs(delta_y) < tol:
35             break
36
37     return y
38
39
40 # 创建两个空的 DataFrame 用于存储数据
41 x_results_df = pd.DataFrame()
42 y_results_df = pd.DataFrame()
43 v_results_df = pd.DataFrame()
44
45 # 遍历 s 的不同值
46 for j in range(0, 301):
47     print(j)
48     s_given = j * 100
49
50     # 计算 x 的初始值
51     A = 0.5 * 55 / (2 * np.pi)
52     x = 2 * np.pi * 16
53     all_value = A * (x * np.sqrt(1 + x**2) + np.arcsinh(x))
54
55     def f(x):
56         return A * (x * np.sqrt(1 + x**2) + np.log(x + np.sqrt(1 + x**2))) -
57             (all_value - s_given)
58
59     try:
60         x_solution = newton(f, 1, tol=1e-10)
61     except Exception as e:
62         print(f"Error for s={s_given}: {e}")
63         x_solution = np.nan
64
65     x_constant = x_solution

```

```

66     # 迭代与记录坐标
67     x_coords = [55 * x_constant / (2 * np.pi) * np.cos(x_constant)]
68     y_coords = [55 * x_constant / (2 * np.pi) * np.sin(x_constant)]
69     theta_coords = [x_constant]
70     v_coords=[100]
71     v=100
72
73     for i in range(223):
74         y_initial = x_constant + 159 / (55 / (2 * np.pi) * np.sqrt(x_constant**2 -
75             1))
76         y_solution = halley_method(y_initial, x_constant, i)
77         x_constant = y_solution
78         L=165
79         if i == 0 :
80             L=341-2*27.5
81             a=55/2/np.pi
82             # 记录坐标
83             theta_coords.append(x_constant)
84             x_1=theta_coords[i]
85             x_2=theta_coords[i+1]
86             v = v * np.sqrt((1 + x_2**2) / (1 + x_1**2)) * (
87                 ((x_1 * np.sin(x_1) - np.cos(x_1)) * (x_2 * np.cos(x_2) - x_1 *
88                     np.cos(x_1)) -
89                     (x_1 * np.cos(x_1) + np.sin(x_1)) * (x_2 * np.sin(x_2) - x_1 *
90                         np.sin(x_1))) /
91                 ((x_2 * np.sin(x_2) - np.cos(x_2)) * (x_2 * np.cos(x_2) - x_1 *
92                     np.cos(x_1)) -
93                     (x_2 * np.cos(x_2) + np.sin(x_2)) * (x_2 * np.sin(x_2) - x_1 *
94                         np.sin(x_1))))
95
96             v_coords.append(v)
97             # 记录坐标
98             x_coord = 55 * y_solution / (2 * np.pi) * np.cos(y_solution)
99             y_coord = 55 * y_solution / (2 * np.pi) * np.sin(y_solution)
100            x_coords.append(x_coord)
101            y_coords.append(y_coord)
102
103        # 将结果保存到 DataFrame
104        x_results_df[f'{math.ceil((s_given-1)/100)}s'] = x_coords
105        y_results_df[f'{math.ceil((s_given-1)/100)}s'] = y_coords
106        v_results_df[f'{math.ceil((s_given-1)/100)}s'] = v_coords
107
108    # 保存到 Excel 文件
109    x_results_df.to_excel('x_1.xlsx', index=True)
110    y_results_df.to_excel('y_1.xlsx', index=True)
111    v_results_df.to_excel('v_1.xlsx', index=True)

```

3.2 Halley_1.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.optimize import newton
4
5 # 求导
6 def f_0(y, x):
7     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (55/(2*np.pi))**2
8
9 def f_1(y, x):
10    return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
11
12 def f_2(y, x):
13    return 2 - 2*x*np.sin(x-y) - 2*x*y*np.sin(x-y) - 2*x*y*np.cos(x-y)
14
15
16
17 # Halley方法
18 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
19     y = y_initial
20     for _ in range(max_iter):
21         f0 = f_0(y, x)
22         f1 = f_1(y, x)
23         f2 = f_2(y, x)
24
25         if i == 0:
26             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 /
27                 (55/(2*np.pi))**2
28
29         delta_y = f0/f1*(1-f0*f2/(2*f1**2))
30         y -= delta_y
31
32         if abs(delta_y) < tol:
33             break
34
35     return y
36
37 # 第一个点走了多远
38 A = 0.5 * 55 / (2 * np.pi)
39 x = 2 * np.pi * 16
40 all_value = A * (x * np.sqrt(1 + x**2) + np.arcsinh(x))
41 s_given = 300 * 100 # 几秒
42
43 def f(x):
44     return A * (x * np.sqrt(1 + x**2) + np.log(x + np.sqrt(1 + x**2))) -
45     (all_value - s_given)
```

```

44
45 x_solution = newton(f, 1, tol=1e-10)
46 x_constant = x_solution
47 print(x_constant)
48
49 # 迭代与可视化
50 points = []
51 point = (55 * x_constant / (2 * np.pi) * np.cos(x_constant), 55 * x_constant / (2
    * np.pi) * np.sin(x_constant))
52 points.append(point)
53
54 for i in range(0, 223):
55     if i == 0:
56         y_initial = x_constant + (341 - 2 * 27.5) / (55 / (2 * np.pi) *
57             np.sqrt(x_constant**2 - 1))
58         y_initial = x_constant + 159 / (55 / (2 * np.pi) * np.sqrt(x_constant**2 - 1))
59         y_solution = halley_method(y_initial, x_constant, i)
60         x_constant = y_solution
61         print(y_solution)
62
63         point = (55 * y_solution / (2 * np.pi) * np.cos(y_solution), 55 * y_solution /
64             (2 * np.pi) * np.sin(y_solution))
65         points.append(point)
66
67 # 可视化
68 plt.figure(figsize=(8, 8))
69
70 # 绘制等距螺线
71 theta = np.linspace(0, 32 * np.pi, 1000)
72 max_radius = 55 / (2 * np.pi) * 32 * np.pi
73 r = np.linspace(0, max_radius, 1000)
74 x_helix = r * np.cos(theta)
75 y_helix = r * np.sin(theta)
76 plt.plot(x_helix, y_helix, 'lightblue', linestyle='-', linewidth=2, alpha=0.6)
77
78 # 绘制把手位置点
79 for i, point in enumerate(points):
80     plt.plot(point[0], point[1], 'o', color="#FFFF80")
81
82 for i in range(1, len(points)):
83     plt.plot([points[i-1][0], points[i][0]], [points[i-1][1], points[i][1]], 'r-',
84         linewidth=2)
85
86 # 设置坐标轴边界
87 plt.xlim(-1250, 1250)
88 plt.ylim(-1250, 1250)

```

```

87 plt.xlabel('x')
88 plt.ylabel('y')
89 plt.gca().set_aspect('equal', adjustable='box')
90 plt.show()

```

3.3 draw_v.py

```

1      import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.optimize import newton
4
5 # 求导
6 def f_0(y, x):
7     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (55/(2*np.pi))**2
8
9 def f_1(y, x):
10    return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
11
12 def f_2(y, x):
13    return 2 + 2*x*np.sin(x-y) + 2*x*np.sin(x-y) + 2*x*y*np.cos(x-y)
14
15
16 # Halley方法
17 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
18     y = y_initial
19     for _ in range(max_iter):
20         f0 = f_0(y, x)
21         f1 = f_1(y, x)
22         f2 = f_2(y, x)
23
24         if i == 0:
25             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 /
26                 (55/(2*np.pi))**2
27
28         delta_y = f0/f1*(1-f0*f2/(2*f1**2))
29         y -= delta_y
30
31         if abs(delta_y) < tol:
32             break
33
34     return y
35
36 # 指定秒
37 j_values = [0, 60, 120, 180, 240, 300]
38 colors = ['k', 'b', 'g', 'r', 'c', 'm']

```

```

39 plt.figure(figsize=(10, 6))
40
41 for j, color in zip(j_values, colors):
42     s_given = j * 100
43     A = 0.5 * 55 / (2 * np.pi)
44     x = 2 * np.pi * 16
45     all_value = A * (x * np.sqrt(1 + x**2) + np.arcsinh(x))
46
47     def f(x):
48         return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x)) - (all_value - s_given)
49
50     try:
51         x_solution = newton(f, 1, tol=1e-10)
52     except Exception as e:
53         print(f"Error for s={s_given}: {e}")
54     x_solution = np.nan
55
56     x_constant = x_solution
57     theta_coords = [x_constant]
58     v_coords = [100]
59     v = 100
60
61     for i in range(223):
62         y_initial = x_constant + 159 / (55 / (2 * np.pi) * np.sqrt(x_constant**2 -
63             1))
64         y_solution = halley_method(y_initial, x_constant, i)
65         x_constant = y_solution
66
67         L = 165
68         if i == 0:
69             L = 341 - 2 * 27.5
70
71         theta_coords.append(x_constant)
72         x_1 = theta_coords[i]
73         x_2 = theta_coords[i + 1]
74         v = v * np.sqrt((1 + x_2**2) / (1 + x_1**2)) * (
75             ((x_1 * np.sin(x_1) - np.cos(x_1)) * (x_2 * np.cos(x_2) - x_1 *
76                 np.cos(x_1)) -
77                 (x_1 * np.cos(x_1) + np.sin(x_1)) * (x_2 * np.sin(x_2) - x_1 *
78                     np.sin(x_1))) /
79             ((x_2 * np.sin(x_2) - np.cos(x_2)) * (x_2 * np.cos(x_2) - x_1 *
80                 np.cos(x_1)) -
81                 (x_2 * np.cos(x_2) + np.sin(x_2)) * (x_2 * np.sin(x_2) - x_1 *
82                     np.sin(x_1)))
83     )
84
85     v_coords.append(v)

```

```

81     plt.plot(range(0, len(v_coords)), v_coords, linestyle='--', color=color,
82               label=f't={j}s')
83
84 # 调整坐标轴
85 plt.xlim([0, len(v_coords)])
86 plt.ylim([99.6, 100])
87
88 plt.xlabel('number')
89 plt.ylabel('v')
90 plt.xticks(range(0, len(v_coords), 20))
91 plt.yticks(np.arange(99.6, 100.02, 0.1))
92 plt.legend()
93 plt.show()

```

附录 D 问题二代码

4.1 check_2.py

```

1 import numpy as np
2 from scipy.optimize import newton
3 from matplotlib.patches import Polygon
4 from matplotlib.path import Path
5
6 #求导
7 def f_0(y, x):
8     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (55/(2*np.pi))**2
9
10 def f_1(y, x):
11     return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
12
13 def f_2(y, x):
14     return 2 - 2*x*np.sin(x-y) - 2*x*np.sin(x-y) - 2*x*y*np.cos(x-y)
15
16
17 # Halley方法
18 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
19     y = y_initial
20     for _ in range(max_iter):
21         f0 = f_0(y, x)
22         f1 = f_1(y, x)
23         f2 = f_2(y, x)
24
25         if i == 0:
26             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 /

```

```

                (55/(2*np.pi))**2
27
28     delta_y = f0/f1*1/(1-f0*f2/(2*f1**2))
29     y -= delta_y
30
31     if abs(delta_y) < tol:
32         break
33
34 return y
35
36 # 初始化参数
37 A = 0.5 * 55 / (2 * np.pi)
38 x = 2 * np.pi * 16
39 all = A * (x * np.sqrt(1 + x**2) + np.log(x + np.sqrt(1 + x**2)))
40
41 #开始检查, j为时间
42 for j in np.arange(100, 500, 0.01):
43     s_given = j * 100
44     print(j)
45
46 def f(x):
47     return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x)) - (all - s_given)
48
49 x_solution = newton(f, 1, tol=1e-10)
50 x_constant = x_solution
51
52
53 points = []
54 point = (55 * x_constant / (2 * np.pi) * np.cos(x_constant), 55 * x_constant /
55     (2 * np.pi) * np.sin(x_constant))
55 points.append(point)
56 y_threshold = 2 * 16 * np.pi
57
58 for i in range(0, 223):
59     if i == 0:
60         y_initial = x_constant + (341-2*27.5) / (55 / (2 * np.pi) *
61             np.sqrt(x_constant**2 - 1))
62
62     y_initial = x_constant + 159 / (55 / (2 * np.pi) * np.sqrt(x_constant**2 -
63         1))
63     y_solution = halley_method(y_initial, x_constant, i)
64
65     # 是否进入螺旋
66     if np.isclose(y_solution, y_threshold, atol=1e-10):
67         print(f"Stopping iteration at i={i} because y_solution reached the
68             threshold {y_threshold}")
68         break

```

```

69
70     x_constant = y_solution
71
72     point = (55 * y_solution / (2 * np.pi) * np.cos(y_solution), 55 *
73             y_solution / (2 * np.pi) * np.sin(y_solution))
74     points.append(point)
75
76     patches = []
77     rectangles = []
78     #根据比例尺画矩形
79     for i in range(1, len(points)):
80         x1, y1 = points[i-1]
81         x2, y2 = points[i]
82
83
84         length = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
85         width = length * 30 / (341-55) if i == 1 else length * 30 / 165
86
87
88         extension = length * 27.5 / (341-55) if i == 1 else length * 27.5 / 165
89
90
91         dx = x2 - x1
92         dy = y2 - y1
93         norm = np.sqrt(dx**2 + dy**2)
94         dx /= norm
95         dy /= norm
96
97
98         perp_dx = -dy
99         perp_dy = dx
100
101
102         corner1 = (x1 - dx * extension + perp_dx * width / 2, y1 - dy * extension
103             + perp_dy * width / 2)
104         corner2 = (x1 - dx * extension - perp_dx * width / 2, y1 - dy * extension
105             - perp_dy * width / 2)
106         corner3 = (x2 + dx * extension + perp_dx * width / 2, y2 + dy * extension
107             + perp_dy * width / 2)
108         corner4 = (x2 + dx * extension - perp_dx * width / 2, y2 + dy * extension
109             - perp_dy * width / 2)
110
111
112         rectangle = Polygon([corner1, corner2, corner4, corner3], closed=True)
113         patches.append(rectangle)
114         rectangles.append(rectangle)

```

```

111
112
113     intersected_patches = []
114     intersection_found = False
115     #判断矩形相交
116     for i in range(len(rectangles)):
117         for j_rect in range(len(rectangles)):
118             if j_rect == i or j_rect == i-1 or j_rect == i+1:
119                 continue
120             path_i = Path(rectangles[i].get_xy())
121             path_j = Path(rectangles[j_rect].get_xy())
122             if path_i.intersects_path(path_j):
123                 print(f"Collision detected at j = {j}, between rectangles {i} and
124                     {j_rect}")
125                 intersection_found = True
126                 intersected_patches.append(rectangles[i])
127                 intersected_patches.append(rectangles[j_rect])
128
129             if intersection_found:
130                 break
131
132     # 一旦矩形相交结束
133     if intersection_found:
134         break

```

4.2 Halley_2.py

```

1      import numpy as np
2      import matplotlib.pyplot as plt
3      from scipy.optimize import newton
4      from matplotlib.patches import Polygon
5      from matplotlib.collections import PatchCollection
6      from matplotlib.path import Path
7
8      # 定义函数
9      def f_0(y, x):
10         return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (55/(2*np.pi))**2
11
12      def f_1(y, x):
13         return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
14
15      def f_2(y, x):
16         return 2 - 2*x*np.sin(x-y) - 2*x*np.sin(x-y) - 2*x*y*np.cos(x-y)
17
18

```

```

19 # Halley方法
20 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
21     y = y_initial
22     for _ in range(max_iter):
23         f0 = f_0(y, x)
24         f1 = f_1(y, x)
25         f2 = f_2(y, x)
26
27         if i == 0:
28             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 /
29                 (55/(2*np.pi))**2
30
31         delta_y = f0/f1**1/(1-f0*f2/(2*f1**2))
32         y -= delta_y
33
34         if abs(delta_y) < tol:
35             break
36
37     return y
38
39 # 初始化
40 A = 0.5 * 55 / (2 * np.pi)
41 x = 2 * np.pi * 16
42 all = A * (x * np.sqrt(1 + x**2) + np.log(x + np.sqrt(1 + x**2)))
43 s_given = 412.47383 * 100
44
45 def f(x):
46     return A * (x * np.sqrt(1 + x**2) + np.log(x + np.sqrt(1 + x**2))) - (all -
47         s_given)
48
49 x_solution = newton(f, 1, tol=1e-10)
50 x_constant = x_solution
51 points = []
52 point = (55 * x_constant / (2 * np.pi) * np.cos(x_constant), 55 * x_constant / (2
53     * np.pi) * np.sin(x_constant))
54 points.append(point)
55 y_threshold = 2 * 16 * np.pi
56
57 for i in range(0, 223):
58     if i == 0:
59         y_initial = x_constant + (341-2*27.5) / (55 / (2 * np.pi) *
60             np.sqrt(x_constant**2 - 1))
61
62         y_initial = x_constant + 159 / (55 / (2 * np.pi) * np.sqrt(x_constant**2 - 1))
63         y_solution = halley_method(y_initial, x_constant, i)
64         x_constant = y_solution

```

```

62     point = (55 * y_solution / (2 * np.pi) * np.cos(y_solution), 55 * y_solution /
63         (2 * np.pi) * np.sin(y_solution))
64     points.append(point)
65
66 # 可视化
67 plt.figure(figsize=(8, 8))
68
69 # 绘制等距螺线 (最底层)
70 theta = np.linspace(0, 32 * np.pi, 1000)
71 max_radius = 55 / (2 * np.pi) * 32 * np.pi
72 r = np.linspace(0, max_radius, 1000)
73 x_helix = r * np.cos(theta)
74 y_helix = r * np.sin(theta)
75 plt.plot(x_helix, y_helix, 'lightblue', linestyle='-', linewidth=2, alpha=0.6,
76           zorder=0) # 设置zorder为0, 使其在底层
77
78 patches = []
79 rectangles = []
80
81 # 绘制点
82 for i, point in enumerate(points):
83     plt.plot(point[0], point[1], 'b', zorder=1)
84
85 # 根据比例尺定义矩形
86 for i in range(1, len(points)):
87     x1, y1 = points[i-1]
88     x2, y2 = points[i]
89
90     length = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
91     width = length * 30 / (341-55) if i == 1 else length * 30 / 165
92     extension = length * 27.5 / (341-55) if i == 1 else length * 27.5 / 165
93     dx = x2 - x1
94     dy = y2 - y1
95     norm = np.sqrt(dx**2 + dy**2)
96     dx /= norm
97     dy /= norm
98     perp_dx = -dy
99     perp_dy = dx
100    corner1 = (x1 - dx * extension + perp_dx * width / 2, y1 - dy * extension +
101                perp_dy * width / 2)
102    corner2 = (x1 - dx * extension - perp_dx * width / 2, y1 - dy * extension -
103                perp_dy * width / 2)
104    corner3 = (x2 + dx * extension + perp_dx * width / 2, y2 + dy * extension +
105                perp_dy * width / 2)
106    corner4 = (x2 + dx * extension - perp_dx * width / 2, y2 + dy * extension -
107                perp_dy * width / 2)
108    rectangle = Polygon([corner1, corner2, corner4, corner3], closed=True)

```

```

103     patches.append(rectangle)
104     rectangles.append(rectangle)
105
106 # 检查矩形相交
107 intersected_patches = []
108 intersection_found = False
109
110 for i in range(len(rectangles)):
111     for j in range(len(rectangles)):
112         if j == i or j == i-1 or j == i+1:
113             continue
114         path_i = Path(rectangles[i].get_xy())
115         path_j = Path(rectangles[j].get_xy())
116         if path_i.intersects_path(path_j):
117             print(f"Intersection found between rectangles {i} and {j}")
118             intersection_found = True
119             intersected_patches.append((rectangles[i], i))
120             intersected_patches.append((rectangles[j], j))
121
122 # 绘制矩形
123 p = PatchCollection(patches, facecolor='lightcoral', edgecolor='red', zorder=2)
124 plt.gca().add_collection(p)
125 if intersected_patches:
126     p_intersected = PatchCollection([patch for patch, _ in intersected_patches],
127                                     facecolor='darkred', edgecolor='red', alpha=0.8, zorder=3) #
128     # 设置zorder为3，使相交矩形在其他矩形之上
129     plt.gca().add_collection(p_intersected)
130
131 plt.xlabel('x')
132 plt.ylabel('y')
133 plt.gca().set_aspect('equal', adjustable='box')
134 plt.show()

```

4.3 save_2.py

```

1 import numpy as np
2 from scipy.optimize import newton
3 import pandas as pd
4 import math
5
6 # 求导
7 def f_0(y, x):
8     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (55/(2*np.pi))**2
9
10 def f_1(y, x):
11     return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)

```

```

12
13 def f_2(y, x):
14     return 2 - 2*x*np.sin(x-y) - 2*x*np.sin(x-y) - 2*x*y*np.cos(x-y)
15
16
17 # Halley方法
18 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
19     y = y_initial
20     for _ in range(max_iter):
21         f0 = f_0(y, x)
22         f1 = f_1(y, x)
23         f2 = f_2(y, x)
24
25
26         # 第一个板凳长度不一样
27         if i == 0:
28             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 /
29             (55/(2*np.pi))**2
30
31         delta_y = f0/f1**1/(1-f0*f2/(2*f1**2))
32         y -= delta_y
33
34         if abs(delta_y) < tol:
35             break
36
37
38
39 # 创建两个空的 DataFrame 用于存储数据
40 x_results_df = pd.DataFrame()
41 y_results_df = pd.DataFrame()
42 v_results_df = pd.DataFrame()
43
44 # 遍历 s 的不同值
45 s_given = 412.47383 * 100
46
47 # 计算 x 的初始值
48 A = 0.5 * 55 / (2 * np.pi)
49 x = 2 * np.pi * 16
50 all_value = A * (x * np.sqrt(1 + x**2) + np.arcsinh(x))
51
52 def f(x):
53     return A * (x * np.sqrt(1 + x**2) + np.log(x + np.sqrt(1 + x**2))) -
54             (all_value - s_given)
55
56 try:
57     x_solution = newton(f, 1, tol=1e-10)

```

```

57 except Exception as e:
58     print(f"Error for s={s_given}: {e}")
59     x_solution = np.nan
60
61 x_constant = x_solution
62 x_coords = [55 * x_constant / (2 * np.pi) * np.cos(x_constant)]
63 y_coords = [55 * x_constant / (2 * np.pi) * np.sin(x_constant)]
64 theta_coords = [x_constant]
65 v_coords=[100]
66 v=100
67
68 for i in range(223):
69     y_initial = x_constant + 159 / (55 / (2 * np.pi) * np.sqrt(x_constant**2 - 1))
70     y_solution = halley_method(y_initial, x_constant, i)
71     x_constant = y_solution
72     L=165
73     if i == 0 :
74         L=341-2*27.5
75     a=55/2/np.pi
76     # 记录坐标
77     theta_coords.append(x_constant)
78     x_1=theta_coords[i]
79     x_2=theta_coords[i+1]
80     v = v * np.sqrt((1 + x_2**2) / (1 + x_1**2)) * (
81         ((x_1 * np.sin(x_1) - np.cos(x_1)) * (x_2 * np.cos(x_2) - x_1 * np.cos(x_1)) -
82         (x_1 * np.cos(x_1) + np.sin(x_1)) * (x_2 * np.sin(x_2) - x_1 * np.sin(x_1))) /
83         ((x_2 * np.sin(x_2) - np.cos(x_2)) * (x_2 * np.cos(x_2) - x_1 * np.cos(x_1)) -
84         (x_2 * np.cos(x_2) + np.sin(x_2)) * (x_2 * np.sin(x_2) - x_1 * np.sin(x_1))))
85
86     v_coords.append(v)
87     # 记录坐标
88     x_coord = 55 * y_solution / (2 * np.pi) * np.cos(y_solution)
89     y_coord = 55 * y_solution / (2 * np.pi) * np.sin(y_solution)
90     x_coords.append(x_coord)
91     y_coords.append(y_coord)
92
93 # 将结果保存到 DataFrame
94 x_results_df[f'{math.ceil((s_given-1)/100)}s'] = x_coords
95 y_results_df[f'{math.ceil((s_given-1)/100)}s'] = y_coords
96 v_results_df[f'{math.ceil((s_given-1)/100)}s'] = v_coords
97
98 # 保存到 Excel 文件
99 x_results_df.to_excel('x_2.xlsx', index=True)
100 y_results_df.to_excel('y_2.xlsx', index=True)
101 v_results_df.to_excel('v_2.xlsx', index=True)

```

附录 E 问题三代码

5.1 fire.py

```
1 import numpy as np
2 from matplotlib.patches import Polygon
3 from matplotlib.path import Path
4 import matplotlib.pyplot as plt
5
6
7 # 求导
8 def f_0(y, x, a):
9     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (a)**2
10
11 def f_1(y, x):
12     return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
13
14 def f_2(y, x):
15     return 2 - 2*x*np.sin(x-y) - 2*x*np.sin(x-y) + 2*x*y*np.cos(x-y)
16
17
18 # Halley方法
19 def halley_method(y_initial, x, i, a, tol=1e-15, max_iter=100):
20     y = y_initial
21     for _ in range(max_iter):
22         f0 = f_0(y, x, a)
23         f1 = f_1(y, x)
24         f2 = f_2(y, x)
25
26
27         if i == 0:
28             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 / (a)**2
29
30         delta_y = f0/f1*(1-f0*f2/(2*f1**2))
31         y -= delta_y
32
33         if abs(delta_y) < tol:
34             break
35
36     return y
37
38 # 定义优化目标函数
39 def check(k, j):
40     print(k, j)
41     no_intersection_found = True
42     a_0 = k / (2 * np.pi)
```

```

43     x_constant = j
44     points = []
45     point = (a_0 * x_constant * np.cos(x_constant), a_0 * x_constant *
46               np.sin(x_constant))
47     points.append(point)
48     for i in range(0, 223):
49         if i == 0:
50             y_initial = x_constant + (341 - 2 * 27.5) / (a_0 *
51                         np.sqrt(x_constant**2 - 1))
52
53             y_initial = x_constant + 159 / (a_0 * np.sqrt(x_constant**2 - 1))
54             y_solution = halley_method(y_initial, x_constant, i, a_0)
55             x_constant = y_solution
56             point = (a_0 * y_solution * np.cos(y_solution), a_0 * y_solution *
57                       np.sin(y_solution))
58             points.append(point)
59
60
61
62
63     # 根据坐标比例画矩形
64     for i in range(1, len(points)):
65         x1, y1 = points[i-1]
66         x2, y2 = points[i]
67
68         length = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
69         width = length * 30 / (341-55) if i == 1 else length * 30 / 165
70         extension = length * 27.5 / (341-55) if i == 1 else length * 27.5 / 165
71
72         dx = x2 - x1
73         dy = y2 - y1
74         norm = np.sqrt(dx**2 + dy**2)
75         dx /= norm
76         dy /= norm
77
78         perp_dx = -dy
79         perp_dy = dx
80
81         corner1 = (x1 - dx * extension + perp_dx * width / 2, y1 - dy *
82                     extension + perp_dy * width / 2)
83         corner2 = (x1 - dx * extension - perp_dx * width / 2, y1 - dy *
84                     extension - perp_dy * width / 2)
85         corner3 = (x2 + dx * extension + perp_dx * width / 2, y2 + dy *
86                     extension + perp_dy * width / 2)

```

```

84     corner4 = (x2 + dx * extension - perp_dx * width / 2, y2 + dy *
85             extension - perp_dy * width / 2)
86
86     rectangle = Polygon([corner1, corner2, corner4, corner3], closed=True)
87     patches.append(rectangle)
88     rectangles.append(rectangle)
89
90     # 检查相交，相邻不检查
91     intersection_found = False
92
93     for i in range(len(rectangles)):
94         for j in range(len(rectangles)):
95             if j == i or j == i-1 or j == i+1:
96                 continue
97             path_i = Path(rectangles[i].get_xy())
98             path_j = Path(rectangles[j].get_xy())
99             if path_i.intersects_path(path_j):
100                 intersection_found = True
101                 no_intersection_found = False
102                 break
103             if intersection_found:
104                 break
105
106
107     if no_intersection_found:
108         return np.sqrt(k)
109     else:
110         return 1000000
111
112
113 # 模拟退火参数
114 initial_temperature = 1000
115 final_temperature = 1
116 alpha = 0.95 # 温度衰减率
117 max_iterations = 300 # 最大迭代轮数
118 k_bounds = (45, 46)
119
120 # 初始化种群
121 def initialize_individual():
122     k = np.random.uniform(*k_bounds)
123     j_lower_bound = 450 * 2 * np.pi / k
124     j_upper_bound = 450 * 2 * np.pi / k + 3 * np.pi
125     j = np.random.uniform(j_lower_bound, j_upper_bound)
126     return k, j
127
128 # 评估适应度
129 def evaluate_individual(individual):

```

```

130     k, j = individual
131     return check(k, j)
132
133 # 生成邻域解
134 def generate_neighbor(current_solution):
135     k_new = np.random.uniform(*k_bounds)
136     j_lower_bound = 450 * 2 * np.pi / k_new
137     j_upper_bound = 450 * 2 * np.pi / k_new + 3 * np.pi
138     j_new = np.random.uniform(j_lower_bound, j_upper_bound)
139     return k_new, j_new
140
141 # 模拟退火算法
142 def simulated_annealing(initial_temp, final_temp, alpha, max_iter):
143     current_solution = initialize_individual()
144     current_fitness = evaluate_individual(current_solution)
145     best_solution = current_solution
146     best_fitness = current_fitness
147     temperature = initial_temp
148     fitness_history = []
149
150     for _ in range(max_iter):
151         neighbor = generate_neighbor(current_solution)
152         neighbor_fitness = evaluate_individual(neighbor)
153         delta_fitness = neighbor_fitness - current_fitness
154
155         if delta_fitness < 0 or np.random.rand() < np.exp(-delta_fitness) /
156             temperature:
157             current_solution = neighbor
158             current_fitness = neighbor_fitness
159
160             if current_fitness < best_fitness:
161                 best_solution = current_solution
162                 best_fitness = current_fitness
163
164             temperature *= alpha
165             fitness_history.append(best_fitness)
166
167             if temperature < final_temp:
168                 break
169
170     return best_solution, best_fitness, fitness_history
171
172 # 运行模拟退火算法
173 best_solution, best_value, fitness_history = simulated_annealing(
174     initial_temperature, final_temperature, alpha, max_iterations
175 )

```

```

176 print("最佳解决方案:", best_solution)
177 print("最佳函数值:", best_value)
178 k_best, j_best = best_solution
179 print("最佳解决方案 (k, j):", (k_best, j_best))
180 # 可视化学习曲线
181 plt.figure(figsize=(8, 6))
182 plt.plot(range(len(fitness_history)), fitness_history, 'b-', label='Best Fitness')
183 plt.title('Simulated Annealing Learning Curve')
184 plt.xlabel('Iteration')
185 plt.ylabel('Best Fitness')
186 plt.legend()
187 plt.show()

```

5.2 check_3.py

```

1 import numpy as np
2 from matplotlib.patches import Polygon
3 from matplotlib.path import Path
4 import matplotlib.pyplot as plt
5
6
7 # 求导
8 def f_0(y, x, a):
9     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (a)**2
10
11 def f_1(y, x):
12     return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
13
14 def f_2(y, x):
15     return 2 - 2*x*np.sin(x-y) - 2*x*np.sin(x-y) + 2*x*y*np.cos(x-y)
16
17
18 # Halley方法
19 def halley_method(y_initial, x, i, a, tol=1e-15, max_iter=100):
20     y = y_initial
21     for _ in range(max_iter):
22         f0 = f_0(y, x, a)
23         f1 = f_1(y, x)
24         f2 = f_2(y, x)
25
26         if i == 0:
27             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 / (a)**2
28
29         delta_y = f0/f1*(1-f0*f2/(2*f1**2))
30         y -= delta_y

```

```

32         if abs(delta_y) < tol:
33             break
34
35
36     return y
37
38 # 定义优化目标函数
39 def check(k,j):
40     print(k,j)
41     no_intersection_found = True
42     a_0 = k / (2 * np.pi)
43     x_constant = j
44     points = []
45     point = (a_0 * x_constant * np.cos(x_constant), a_0 * x_constant *
46               np.sin(x_constant))
47     points.append(point)
48     for i in range(0, 223):
49         if i == 0:
50             y_initial = x_constant + (341 - 2 * 27.5) / (a_0 *
51                                         np.sqrt(x_constant**2 - 1))
52
53             y_initial = x_constant + 159 / (a_0 * np.sqrt(x_constant**2 - 1))
54             y_solution = halley_method(y_initial, x_constant, i, a_0)
55             x_constant = y_solution
56             point = (a_0 * y_solution * np.cos(y_solution), a_0 * y_solution *
57                       np.sin(y_solution))
58             points.append(point)
59
60
61
62
63     # 根据坐标比例画矩形
64     for i in range(1, len(points)):
65         x1, y1 = points[i-1]
66         x2, y2 = points[i]
67
68         length = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
69         width = length * 30 / (341-55) if i == 1 else length * 30 / 165
70         extension = length * 27.5 / (341-55) if i == 1 else length * 27.5 / 165
71
72         dx = x2 - x1
73         dy = y2 - y1
74         norm = np.sqrt(dx**2 + dy**2)
75         dx /= norm

```

```

76     dy /= norm
77
78     perp_dx = -dy
79     perp_dy = dx
80
81     corner1 = (x1 - dx * extension + perp_dx * width / 2, y1 - dy *
82         extension + perp_dy * width / 2)
83     corner2 = (x1 - dx * extension - perp_dx * width / 2, y1 - dy *
84         extension - perp_dy * width / 2)
85     corner3 = (x2 + dx * extension + perp_dx * width / 2, y2 + dy *
86         extension + perp_dy * width / 2)
87     corner4 = (x2 + dx * extension - perp_dx * width / 2, y2 + dy *
88         extension - perp_dy * width / 2)
89
90     rectangle = Polygon([corner1, corner2, corner4, corner3], closed=True)
91     patches.append(rectangle)
92     rectangles.append(rectangle)
93
94     # 检查相交，相邻不检查
95     intersection_found = False
96
97     for i in range(len(rectangles)):
98         for j in range(len(rectangles)):
99             if j == i or j == i-1 or j == i+1:
100                 continue
101             path_i = Path(rectangles[i].get_xy())
102             path_j = Path(rectangles[j].get_xy())
103             if path_i.intersects_path(path_j):
104                 intersection_found = True
105                 no_intersection_found = False
106                 break
107             if intersection_found:
108                 break
109
110     if no_intersection_found:
111         return np.sqrt(k)
112     else:
113         return 1000000
114
115
116     # 模拟退火参数
117     initial_temperature = 1000
118     final_temperature = 1
119     alpha = 0.95 # 温度衰减率
120     max_iterations = 300 # 最大迭代轮数
121     k_bounds = (45, 46)

```

```

119
120 # 初始化种群
121 def initialize_individual():
122     k = np.random.uniform(*k_bounds)
123     j_lower_bound = 450 * 2 * np.pi / k
124     j_upper_bound = 450 * 2 * np.pi / k + 3 * np.pi
125     j = np.random.uniform(j_lower_bound, j_upper_bound)
126     return k, j
127
128 # 评估适应度
129 def evaluate_individual(individual):
130     k, j = individual
131     return check(k, j)
132
133 # 生成邻域解
134 def generate_neighbor(current_solution):
135     k_new = np.random.uniform(*k_bounds)
136     j_lower_bound = 450 * 2 * np.pi / k_new
137     j_upper_bound = 450 * 2 * np.pi / k_new + 3 * np.pi
138     j_new = np.random.uniform(j_lower_bound, j_upper_bound)
139     return k_new, j_new
140
141 # 模拟退火算法
142 def simulated_annealing(initial_temp, final_temp, alpha, max_iter):
143     current_solution = initialize_individual()
144     current_fitness = evaluate_individual(current_solution)
145     best_solution = current_solution
146     best_fitness = current_fitness
147     temperature = initial_temp
148     fitness_history = []
149
150     for _ in range(max_iter):
151         neighbor = generate_neighbor(current_solution)
152         neighbor_fitness = evaluate_individual(neighbor)
153         delta_fitness = neighbor_fitness - current_fitness
154
155         if delta_fitness < 0 or np.random.rand() < np.exp(-delta_fitness /
156             temperature):
157             current_solution = neighbor
158             current_fitness = neighbor_fitness
159
160             if current_fitness < best_fitness:
161                 best_solution = current_solution
162                 best_fitness = current_fitness
163
164             temperature *= alpha
165             fitness_history.append(best_fitness)

```

```

165     if temperature < final_temp:
166         break
167
168     return best_solution, best_fitness, fitness_history
169
170
171 # 运行模拟退火算法
172 best_solution, best_value, fitness_history = simulated_annealing(
173     initial_temperature, final_temperature, alpha, max_iterations
174 )
175
176 print("最佳解决方案:", best_solution)
177 print("最佳函数值:", best_value)
178 k_best, j_best = best_solution
179 print("最佳解决方案 (k, j):", (k_best, j_best))
180 # 可视化学习曲线
181 plt.figure(figsize=(8, 6))
182 plt.plot(range(len(fitness_history)), fitness_history, 'b-', label='Best Fitness')
183 plt.title('Simulated Annealing Learning Curve')
184 plt.xlabel('Iteration')
185 plt.ylabel('Best Fitness')
186 plt.legend()
187 plt.show()

```

5.3 Halley_3.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib.patches import Polygon
4 from matplotlib.collections import PatchCollection
5 from matplotlib.path import Path
6 #具体指定参数
7 a_0=45.034/(2*np.pi)
8 # 求导
9 def f_0(y, x):
10     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (a_0)**2
11
12 def f_1(y, x):
13     return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
14
15 def f_2(y, x):
16     return 2 + 2*x*np.sin(x-y) + 2*x*y*np.sin(x-y) + 2*x*y*np.cos(x-y)
17
18
19 # Halley方法
20 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):

```

```

21     y = y_initial
22     for _ in range(max_iter):
23         f0 = f_0(y, x)
24         f1 = f_1(y, x)
25         f2 = f_2(y, x)
26
27
28         if i == 0:
29             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 / (a_0)**2
30
31         delta_y = f0/f1*(1-f0*f2/(2*f1**2))
32         y -= delta_y
33
34         if abs(delta_y) < tol:
35             break
36
37
38     return y
39
40 # 初始化参数
41 x_constant = 450/a_0
42 points = []
43 point = (a_0 * x_constant * np.cos(x_constant), a_0 * x_constant *
44           np.sin(x_constant))
44 points.append(point)
45 y_threshold = 2 * 16 * np.pi
46
47 for i in range(0, 223):
48     if i == 0:
49         y_initial = x_constant + (341-2*27.5) / (a_0 * np.sqrt(x_constant**2 - 1))
50
51         y_initial = x_constant + 159 / (a_0 * np.sqrt(x_constant**2 - 1))
52         y_solution = halley_method(y_initial, x_constant, i)
53         x_constant = y_solution
54
55         point = (a_0 * y_solution * np.cos(y_solution), a_0 * y_solution *
56                   np.sin(y_solution))
56 points.append(point)
57
58
59 # 可视化
60 plt.figure(figsize=(8, 8))
61
62 # 绘制等距螺线 (最底层)
63 theta = np.linspace(0, 32 * np.pi, 1000)
64 max_radius = 55 / (2 * np.pi) * 32 * np.pi
65 r = np.linspace(0, max_radius, 1000)

```

```

66 x_helix = r * np.cos(theta)
67 y_helix = r * np.sin(theta)
68 plt.plot(x_helix, y_helix, 'lightblue', linestyle='--', linewidth=2, alpha=0.6,
69      zorder=0) # 设置zorder为0, 使其在底层
70
71 patches = []
72 rectangles = []
73
74 # 绘制点
75 for i, point in enumerate(points):
76     plt.plot(point[0], point[1], 'b', zorder=1)
77
78 # 根据比例定义矩形
79 for i in range(1, len(points)):
80     x1, y1 = points[i-1]
81     x2, y2 = points[i]
82
83     length = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
84     width = length * 30 / (341-55) if i == 1 else length * 30 / 165
85     extension = length * 27.5 / (341-55) if i == 1 else length * 27.5 / 165
86     dx = x2 - x1
87     dy = y2 - y1
88     norm = np.sqrt(dx**2 + dy**2)
89     dx /= norm
90     dy /= norm
91     perp_dx = -dy
92     perp_dy = dx
93     corner1 = (x1 - dx * extension + perp_dx * width / 2, y1 - dy * extension +
94                 perp_dy * width / 2)
95     corner2 = (x1 - dx * extension - perp_dx * width / 2, y1 - dy * extension -
96                 perp_dy * width / 2)
97     corner3 = (x2 + dx * extension + perp_dx * width / 2, y2 + dy * extension +
98                 perp_dy * width / 2)
99     corner4 = (x2 + dx * extension - perp_dx * width / 2, y2 + dy * extension -
100                  perp_dy * width / 2)
101    rectangle = Polygon([corner1, corner2, corner4, corner3], closed=True)
102    patches.append(rectangle)
103    rectangles.append(rectangle)
104
105    # 检查矩形相交
106    intersected_patches = []
107    intersection_found = False
108
109    for i in range(len(rectangles)):
110        for j in range(len(rectangles)):
111            if j == i or j == i-1 or j == i+1:
112                continue
113
114            # 判断两个矩形是否相交
115            if rectangles[i].intersects(rectangles[j]):
116                intersected_patches.append(rectangles[i])
117
118    if intersection_found:
119        print("Intersection found between rectangles at index", i, "and", j)
120
121    # 将相交的矩形从列表中移除
122    for patch in intersected_patches:
123        rectangles.remove(patch)

```

```

108     path_i = Path(rectangles[i].get_xy())
109     path_j = Path(rectangles[j].get_xy())
110     if path_i.intersects_path(path_j):
111         print(f"Intersection found between rectangles {i} and {j}")
112         intersection_found = True
113         intersected_patches.append((rectangles[i], i))
114         intersected_patches.append((rectangles[j], j))
115
116 # 绘制矩形
117 p = PatchCollection(patches, facecolor='lightcoral', edgecolor='red', zorder=2) #
    设置zorder为2, 使非相交矩形在点之上
118 plt.gca().add_collection(p)
119 if intersected_patches:
120     p_intersected = PatchCollection([patch for patch, _ in intersected_patches],
121                                     facecolor='darkred', edgecolor='red', alpha=0.8, zorder=3) #
    设置zorder为3, 使相交矩形在其他矩形之上
122     plt.gca().add_collection(p_intersected)
123
124 plt.xlabel('x')
125 plt.ylabel('y')
126 plt.show()

```

5.4 auto_grid.py

```

1 import numpy as np
2 from matplotlib.patches import Polygon
3 from matplotlib.path import Path
4
5 # 求导
6 def f_0(y, x, a):
7     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (a)**2
8
9 def f_1(y, x):
10    return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
11
12 def f_2(y, x):
13    return 2 + 2*x*np.sin(x-y) + 2*x*np.sin(x-y) + 2*x*y*np.cos(x-y)
14
15
16 # Halley数值解
17 def halley_method(y_initial, x, i, a, tol=1e-15, max_iter=100):
18     y = y_initial
19     for _ in range(max_iter):
20         f0 = f_0(y, x, a)
21         f1 = f_1(y, x)

```

```

22     f2 = f_2(y, x)
23
24
25     if i == 0:
26         f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 / (a)**2
27
28
29     delta_y = f0/f1*1/(1-f0*f2/(2*f1**2))
30     y -= delta_y
31
32     if abs(delta_y) < tol:
33         break
34
35     return y
36
37 def adaptive_grid_search():
38     # 由模拟退火算法筛选数据后得到的初始查询范围
39     initial_k_range = (45.03, 45.04)
40     d1 = 0.001
41     k_step = d1
42     tol_k = 1e-4
43     k_values = np.arange(initial_k_range[0], initial_k_range[1], k_step)
44     count = 1
45
46     # 当螺距的精度达到6位小数时停止划分网格
47     while d1 >= tol_k:
48         results = []
49         print("第", count, "轮自适应网格")
50         count += 1
51
52         for k in k_values:
53             print(f"Testing k = {k}")
54
55             # 使用当前的 k 值初始化 j_values
56             j_values = np.arange(450 / (k / (2 * np.pi)) + 0.96, 450 / (k / (2 *
57                                         np.pi)) + 1.01, 0.001)
58
59             for j in j_values:
60                 print(f"Testing j = {j}")
61                 # 开始查询, 判断给定 k, j 下是否发生碰撞
62
63                 a_0 = k / (2 * np.pi)
64                 x_constant = j
65                 points = []
66                 point = (a_0 * x_constant * np.cos(x_constant), a_0 * x_constant *
67                           np.sin(x_constant))
68                 points.append(point)

```

```

67
68     # 求解此时刻队伍位置
69     for i in range(0, 223):
70         if i == 0:
71             y_initial = x_constant + (341 - 2 * 27.5) / (a_0 *
72                                         np.sqrt(x_constant ** 2 - 1))
73
74             y_initial = x_constant + 159 / (a_0 * np.sqrt(x_constant ** 2
75                                         - 1))
76             y_solution = halley_method(y_initial, x_constant, i, a_0)
77             x_constant = y_solution
78             point = (a_0 * y_solution * np.cos(y_solution), a_0 *
79                         y_solution * np.sin(y_solution))
80             points.append(point)
81
82             patches = []
83             rectangles = []
84             # 根据坐标比例画矩形
85             for i in range(1, len(points)):
86                 x1, y1 = points[i - 1]
87                 x2, y2 = points[i]
88
89                 length = np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
90                 width = length * 30 / (341 - 55) if i == 1 else length * 30 /
91                                         165
92                 extension = length * 27.5 / (341 - 55) if i == 1 else length *
93                                         27.5 / 165
94
95                 dx = x2 - x1
96                 dy = y2 - y1
97                 norm = np.sqrt(dx ** 2 + dy ** 2)
98                 dx /= norm
99                 dy /= norm
100
101                 perp_dx = -dy
102                 perp_dy = dx
103
104                 corner1 = (x1 - dx * extension + perp_dx * width / 2, y1 - dy
105                             * extension + perp_dy * width / 2)
106                 corner2 = (x1 - dx * extension - perp_dx * width / 2, y1 - dy
107                             * extension - perp_dy * width / 2)
108                 corner3 = (x2 + dx * extension + perp_dx * width / 2, y2 + dy
109                             * extension + perp_dy * width / 2)
110                 corner4 = (x2 + dx * extension - perp_dx * width / 2, y2 + dy
111                             * extension - perp_dy * width / 2)
112
113                 rectangle = Polygon([corner1, corner2, corner4, corner3],

```

```

                closed=True)
105             patches.append(rectangle)
106             rectangles.append(rectangle)
107
108     # 矩形相交检验
109     intersection_found = False
110     for i in range(len(rectangles)):
111         for t in range(len(rectangles)):
112             if t == i or t == i - 1 or t == i + 1:
113                 continue
114             path_i = Path(rectangles[i].get_xy())
115             path_t = Path(rectangles[t].get_xy())
116             if path_i.intersects_path(path_t):
117                 print(f"Intersection found between rectangles {i} and
118                     {t}")
119                 intersection_found = True
120                 break
121             if intersection_found:
122                 break
123
124         flag = 0 if intersection_found else 1
125         results.append((k, j - 450 / (k / (2 * np.pi)), flag))
126
127     # 检验完毕后，找到红色点中螺距最大的点，更新新一轮的网格
128     max_k_with_intersection = max((k for k, j, flag in results if flag == 0),
129                                     default=None)
130
131     # 螺距上下界更新为上下变换一格，十等分
132     red_points = [(k, j) for k, j, flag in results if flag == 0]
133     max_red_point_j = max(red_points, key=lambda x: x[0])[1]
134     d1 /= 2
135
136     # 时间上界更新为右侧第一个蓝点的时间，下界更新为左侧第一个蓝点的时间，50等分
137     left_limit = next((j for k, j, flag in results if flag == 1 and j <
138                         max_red_point_j), max_red_point_j - 0.05)
139     right_limit = next((j for k, j, flag in results if flag == 1 and j >
140                         max_red_point_j), max_red_point_j + 0.05)
141
142     j_step = (right_limit - left_limit) / 50
143     j_values = np.arange(left_limit, right_limit, j_step)
144     k_values = np.arange(max_k_with_intersection - d1, max_k_with_intersection
145                         + d1, d1)
146
147     print("最大螺距为：", max_k_with_intersection)
148
149     adaptive_grid_search()

```

5.5 draw_3.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib.patches import Polygon
4 from matplotlib.path import Path
5
6 # 求导
7 def f_0(y, x, a):
8     return x**2 + y**2 - 2*x*y*np.cos(x-y) - (165)**2 / (a)**2
9
10 def f_1(y, x):
11     return 2*y - 2*x*np.cos(x-y) - 2*x*y*np.sin(x-y)
12
13 def f_2(y, x):
14     return 2 + 2*x*np.sin(x-y) + 2*x*np.sin(x-y) + 2*x*y*np.cos(x-y)
15
16
17 # Halley数值解
18 def halley_method(y_initial, x, i, a, tol=1e-15, max_iter=100):
19     y = y_initial
20     for _ in range(max_iter):
21         f0 = f_0(y, x, a)
22         f1 = f_1(y, x)
23         f2 = f_2(y, x)
24
25
26         if i == 0:
27             f0 = x**2 + y**2 - 2*x*y*np.cos(x-y) - (341-2*27.5)**2 / (a)**2
28
29
30         delta_y = f0/f1*(1-f0*f2/(2*f1**2))
31         y -= delta_y
32
33         if abs(delta_y) < tol:
34             break
35
36     return y
37
38 # 存储 k 和 j 的值以及对应的 flag
39 results = []
40
41
42 for k in np.arange(45.0329, 45.0340, 0.0001):
43     print(f "Testing k = {k}")
44
45     for j in np.arange(450 / (k / (2 * np.pi)) + 0.96, 450 / (k / (2 *
```

```

        np.pi))+1.05,0.001):
46    print(f "Testing j = {j}")
47
48    no_intersection_found = True # This will track if all iterations for
49    # this k have no intersections
50
51    a_0 = k / (2 * np.pi)
52    x_constant = j
53    points = []
54    point = (a_0 * x_constant * np.cos(x_constant), a_0 * x_constant *
55    np.sin(x_constant))
56    points.append(point)
57    y_threshold = 2 * 16 * np.pi
58
59    for i in range(0, 223):
60        if i == 0:
61            y_initial = x_constant + (341 - 2 * 27.5) / (a_0 *
62            np.sqrt(x_constant**2 - 1))
63
64            y_initial = x_constant + 159 / (a_0 * np.sqrt(x_constant**2 - 1))
65            y_solution = halley_method(y_initial, x_constant, i, a_0)
66            x_constant = y_solution
67            point = (a_0 * y_solution * np.cos(y_solution), a_0 * y_solution *
68            np.sin(y_solution))
69            points.append(point)
70
71
72    # 根据坐标比例画矩形
73    for i in range(1, len(points)):
74        x1, y1 = points[i-1]
75        x2, y2 = points[i]
76
77        length = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
78        width = length * 30 / (341-55) if i == 1 else length * 30 / 165
79        extension = length * 27.5 / (341-55) if i == 1 else length * 27.5
80        # 165
81
82        dx = x2 - x1
83        dy = y2 - y1
84        norm = np.sqrt(dx**2 + dy**2)
85        dx /= norm
86        dy /= norm

```

```

87         perp_dx = -dy
88         perp_dy = dx
89
90         corner1 = (x1 - dx * extension + perp_dx * width / 2, y1 - dy *
91             extension + perp_dy * width / 2)
92         corner2 = (x1 - dx * extension - perp_dx * width / 2, y1 - dy *
93             extension - perp_dy * width / 2)
94         corner3 = (x2 + dx * extension + perp_dx * width / 2, y2 + dy *
95             extension + perp_dy * width / 2)
96         corner4 = (x2 + dx * extension - perp_dx * width / 2, y2 + dy *
97             extension - perp_dy * width / 2)
98
99         rectangle = Polygon([corner1, corner2, corner4, corner3],
100                         closed=True)
101     patches.append(rectangle)
102     rectangles.append(rectangle)
103
104     # 检查相交，相邻不检查
105     intersection_found = False
106
107     for i in range(len(rectangles)):
108         for t in range(len(rectangles)):
109             if t == i or t == i-1 or t == i+1:
110                 continue
111             path_i = Path(rectangles[i].get_xy())
112             path_t = Path(rectangles[t].get_xy())
113             if path_i.intersects_path(path_t):
114                 print(f"Intersection found between rectangles {i} and {t}")
115                 intersection_found = True
116                 no_intersection_found = False # Set this to False since
117                 we found an intersection
118                 break
119             if intersection_found:
120                 break
121
122     # 设置 flag
123     flag = 0 if intersection_found else 1
124     results.append((k, j-450 / (k / (2 * np.pi)), flag))
125
126     # 绘制 k 和 j 的图
127     plt.figure(figsize=(8, 6))
128     for k, j, flag in results:
129         if flag == 0:
130             plt.plot(j, k, 'ro') # 红色点表示相交
131         else:
132             plt.plot(j, k, 'bo') # 蓝色点表示不相交

```

```

128 plt.xlabel('j')
129 plt.ylabel('k')
130 plt.grid()
131 plt.show()

```

附录 F 问题四代码

6.1 v.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.optimize import newton
4
5 # 具体指定参数
6 D = 170
7 a_0 = D / (2 * np.pi)
8
9 # 求导函数定义
10 def f_0(y, x):
11     return x**2 + y**2 - 2 * x * y * np.cos(x - y) - (165)**2 / (a_0)**2
12
13 def f_1(y, x):
14     return 2 * y - 2 * x * np.cos(x - y) - 2 * x * y * np.sin(x - y)
15
16 def f_2(y, x):
17     return 2 - 2 * x * np.sin(x - y) - 2 * x * np.sin(x - y) + 2 * x * y *
18         np.cos(x - y)
19
20 # Halley数值解
21 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
22     y = y_initial
23     for _ in range(max_iter):
24         f0 = f_0(y, x)
25         f1 = f_1(y, x)
26         f2 = f_2(y, x)
27
28         if i == 0:
29             f0 = x**2 + y**2 - 2 * x * y * np.cos(x - y) - (341 - 2 * 27.5)**2 /
30                 (a_0)**2
31
32         delta_y = f0 / f1 * 1 / (1 - f0 * f2 / (2 * f1**2))
33         y -= delta_y
34

```

```

35     if abs(delta_y) < tol:
36         break
37
38     return y
39
40 # 两个圆心
41 O_1x = -2 / 3 * 170/(2*np.pi)
42 O_1y = 150
43 O_2x = 170/(2*np.pi) / 3
44 O_2y = -300
45 # 掉头角度
46 x_constant = 450 / a_0
47 A = 0.5 * D / (2 * np.pi)
48 x_0 = 2 * np.pi * 16
49 all_value = A * (x_0 * np.sqrt(1 + x_0**2) + np.arcsinh(x_0))
50 # 螺线长度公式
51 def f(x):
52     return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x))-(A*(x_constant * np.sqrt(1
53             + x_constant**2) + np.arcsinh(x_constant))+100*100)
54
55 # 起点角度与长度
56 x_begin = newton(f, 1, tol=1e-10)
57 begin_S = A * (x_constant * np.sqrt(1 + x_constant**2) + np.arcsinh(x_constant)) +
58             100*100
59
60 # 两段弧长
61 alpha = np.arctan(450 / D * (2*np.pi))
62 beta = np.arctan(450 / D * (2*np.pi))
63 kexi_0 = np.arctan(450 / D * (2*np.pi))
64 R = 1 / 3 * np.sqrt(450**2 + (D/(2*np.pi))**2)
65 S_2 = 2 * np.pi * 2 * R * 2 * alpha / (2 * np.pi)
66 S_3 = 2 * np.pi * R * 2 * beta / (2 * np.pi)
67
68 # 出去角度
69 x_out = np.pi + 2*np.pi*450/D
70 # 由距离解kexi
71 def judge_1(S):
72     if S <= 100 * 100:
73         def f(x):
74             return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x)) - (begin_S - S)
75         x = newton(f, -20, tol=1e-10)
76         kexi = -x + 450*2*np.pi/D
77     elif S >= 100 * 100 and S <= 100*100 + S_2:
78         alpha_0 = (S - 100 * 100)/(2*R)
79         kexi = 2*alpha_0/3
80     elif S >= 100*100 + S_2 and S <= 100*100 + S_2 + S_3:
81         beta_0 = (S - 100 * 100 - S_2)/(R)

```

```

80     kexi = (beta_0 + 4*kexi_0)/3
81 else:
82     S_out = A * ((x_out-np.pi) * np.sqrt(1 + (x_out-np.pi)**2) +
83                   np.arcsinh(x_out-np.pi))
84 def f(x):
85     return A * ((x-np.pi) * np.sqrt(1 + (x-np.pi)**2) +
86                   np.arcsinh(x-np.pi)) - S_out - (S-S_2-S_3-100*100)
87 x = newton(f, 1, tol=1e-10)
88 kexi = x + 2*kexi_0 - np.pi - (2*np.pi*450)/D
89     return kexi
90 #由kexi解位置
91 def distance(kexi):
92     if kexi <= 0:
93         x = -kexi + 450/(D/(2*np.pi))
94         x_coord = a_0 * x * np.cos(x)
95         y_coord = a_0 * x * np.sin(x)
96     elif kexi <= 4/3*kexi_0:
97         theta = x_constant - 2*2*np.pi
98         rotation_angle = theta - 0.5*np.pi
99         alpha_0 = 3/2*kexi
100        delta = alpha - alpha_0
101        point = (2 * R * np.cos(delta) + 0_1x, 2 * R * np.sin(delta) + 0_1y)
102        rotation_matrix = np.array([[np.cos(rotation_angle),
103                                     -np.sin(rotation_angle)],
104                                     [np.sin(rotation_angle), np.cos(rotation_angle)]])
105        (x_coord, y_coord) = np.dot(rotation_matrix, point)
106    elif kexi <= 2*kexi_0:
107        theta = x_constant - 2*2*np.pi
108        rotation_angle = theta - 0.5*np.pi
109        beta_0 = 3*kexi - 4*kexi_0
110        delta = beta - beta_0
111        point = (-R * np.cos(delta) + 0_2x, R * np.sin(delta) + 0_2y)
112        rotation_matrix = np.array([[np.cos(rotation_angle),
113                                     -np.sin(rotation_angle)],
114                                     [np.sin(rotation_angle), np.cos(rotation_angle)]])
115        (x_coord, y_coord) = np.dot(rotation_matrix, point)
116    else:
117        x = kexi - 2*kexi_0 + np.pi + 2*np.pi*450/D
118        x_coord = D/(2*np.pi)*(x-np.pi)*np.cos(x)
119        y_coord = D/(2*np.pi)*(x-np.pi)*np.sin(x)
120
121        return (x_coord, y_coord)
122 #由kexi解速度方向
123 def direction(kexi):
124     if kexi <= 0:
125         x = -kexi + 450/(D/(2*np.pi))
126         v_x = 1/np.sqrt(x**2+1)*(x*np.sin(x)-np.cos(x))

```

```

123     v_y = 1/np.sqrt(x**2+1)*(-x*np.cos(x)-np.sin(x))
124 elif kexi <= 4/3*kexi_0:
125     alpha_0 = 3/2*kexi
126     v_x = np.cos(2*np.pi*450/D-np.pi+kexi_0-alpha_0)
127     v_y = np.sin(2*np.pi*450/D-np.pi+kexi_0-alpha_0)
128 elif kexi <= 2*kexi_0:
129     beta_0 = 3*kexi - 4*kexi_0
130     v_x = np.cos(2*np.pi*450/D-np.pi-kexi_0+beta_0)
131     v_y = np.sin(2*np.pi*450/D-np.pi-kexi_0+beta_0)
132 else:
133     x = kexi - 2*kexi_0 + np.pi + 2*np.pi*450/D
134     v_x = 1/np.sqrt(1+(x-np.pi)**2)*((x-np.pi)*np.sin(x)-np.cos(x))
135     v_y = 1/np.sqrt(1+(x-np.pi)**2)*(-(x-np.pi)*np.cos(x)-np.sin(x))
136
137     return (-v_x, -v_y)
138 #某时刻画图
139 for _ in np.arange(200, 201):
140     t=200
141     S = 100 * t
142     kexi = judge_1(S)
143     point = distance(kexi)
144     points = [(point, kexi)]
145     kexi_old = kexi
146     V = 100
147     (v_xold, v_yold) = direction(kexi)
148     V_xs = [v_xold * V / np.sqrt(v_xold**2 + v_yold**2)]
149     V_ys = [v_yold * V / np.sqrt(v_xold**2 + v_yold**2)]
150     for i in np.arange(0, 223):
151         if i == 0:
152             L = 341 - 55
153         else:
154             L = 165
155         # 上一个点
156         (x_old, y_old) = point
157
158         def f(xi):
159             x_new, y_new = distance(xi)
160             return (x_new - x_old)**2 + (y_new - y_old)**2 - L**2
161
162     try:
163         # 自适应调整迭代初值
164         if kexi_old > 2 * kexi_0:
165             solve_kexi = newton(f, kexi_old - 0.5, tol=1e-3, maxiter=500)
166             if -solve_kexi + kexi_old < np.pi and abs(solve_kexi - kexi_old) >
167                 1e-6:
168                 point = distance(solve_kexi)
169                 kexi_old = solve_kexi

```

```

169             points.append((point, solve_kexi))
170     else:
171         solve_kexi = newton(f, kexi_old - 1.2, tol=1e-3, maxiter=500)
172         if -solve_kexi + kexi_old < np.pi and abs(solve_kexi - kexi_old) >
173             1e-6:
174             point = distance(solve_kexi)
175             kexi_old = solve_kexi
176             points.append((point, solve_kexi))
177
177     # 计算速度
178     (v_x, v_y) = direction(solve_kexi)
179     # 上一个点和现在点
180     rx_old = x_old
181     ry_old = y_old
182     (rx_new, ry_new) = point
183     l_x = rx_old - rx_new
184     l_y = ry_old - ry_new
185     # 速度大小递推
186     V = V * (v_xold*l_x + v_yold*l_y) / (v_x*l_x + v_y*l_y)
187     v_xold = v_x
188     v_yold = v_y
189     V_xs.append(v_x * V / np.sqrt(v_x**2 + v_y**2))
190     V_ys.append(v_y * V / np.sqrt(v_x**2 + v_y**2))
191
192     except RuntimeError:
193         print("no")
194
195 # 绘制等距螺线
196 theta = np.linspace(0, 32 * np.pi, 1000)
197 r = D / (2 * np.pi) * theta
198 x_spiral = r * np.cos(theta)
199 y_spiral = r * np.sin(theta)
200 plt.plot(x_spiral, y_spiral, color='lightblue', zorder=0)
201
202 # 绘制所有点及其速度矢量
203 for (point, kexi), v_x, v_y in zip(points, V_xs, V_ys):
204     if kexi < 0:
205         plt.plot(point[0], point[1], 'ko', markersize=4, markerfacecolor='none')
# 黑色空心点
206     elif 0 <= kexi < 4 / 3 * kexi_0:
207         plt.plot(point[0], point[1], 'bo', markersize=4, markerfacecolor='none')
# 蓝色空心点
208     elif 4 / 3 * kexi_0 <= kexi < 2 * kexi_0:
209         plt.plot(point[0], point[1], 'go', markersize=4, markerfacecolor='none')
# 绿色空心点
210     else:
211         plt.plot(point[0], point[1], 'yo', markersize=4, markerfacecolor='none')

```

```

# 黄色空心点
212
213     # 绘制速度矢量
214     plt.arrow(point[0], point[1], v_x, v_y,
215                 head_width=12, head_length=7, fc='red', ec='red',
216                 length_includes_head=True, zorder=3)
217
218 # 设置图形属性
219 plt.xlabel('x')
220 plt.ylabel('y')
221 plt.xlim(-1500,1500)
222 plt.ylim(-1500,1500)
223 plt.gca().set_aspect('equal', adjustable='box')
224 plt.show()

```

6.2 save_v.py

```

1 import numpy as np
2 from scipy.optimize import newton
3 import pandas as pd
4
5 # 具体指定参数
6 D = 170
7 a_0 = D / (2 * np.pi)
8
9 # 求导函数定义
10 def f_0(y, x):
11     return x**2 + y**2 - 2 * x * y * np.cos(x - y) - (165)**2 / (a_0)**2
12
13 def f_1(y, x):
14     return 2 * y - 2 * x * np.cos(x - y) - 2 * x * y * np.sin(x - y)
15
16 def f_2(y, x):
17     return 2 - 2 * x * np.sin(x - y) - 2 * x * np.sin(x - y) + 2 * x * y *
18             np.cos(x - y)
19
20 # Halley方法
21 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
22     y = y_initial
23     for _ in range(max_iter):
24         f0 = f_0(y, x)
25         f1 = f_1(y, x)
26         f2 = f_2(y, x)
27         if i == 0:
28             f0 = x**2 + y**2 - 2 * x * y * np.cos(x - y) - (341 - 2 * 27.5)**2 /

```

```

        (a_0)**2

29
30     delta_y = f0 / f1 * 1 / (1 - f0 * f2 / (2 * f1**2))
31     y -= delta_y
32
33     if abs(delta_y) < tol:
34         break
35
36     return y
37
38 # 两个圆心
39 O_1x = -2 / 3 * 170/(2*np.pi)
40 O_1y = 150
41 O_2x = 170/(2*np.pi) / 3
42 O_2y = -300
43 # 掉头角度
44 x_constant = 450 / a_0
45 A = 0.5 * D / (2 * np.pi)
46 x_0 = 2 * np.pi * 16
47 all_value = A * (x_0 * np.sqrt(1 + x_0**2) + np.arcsinh(x_0))
48 # 螺线长度公式
49 def f(x):
50     return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x))-(A*(x_constant * np.sqrt(1
51             + x_constant**2) + np.arcsinh(x_constant))+100*100)
52
53 # 起点角度与长度
54 x_begin = newton(f, 1, tol=1e-10)
55 begin_S = A * (x_constant * np.sqrt(1 + x_constant**2) + np.arcsinh(x_constant)) +
56             100*100
57
58 # 两段弧长
59 alpha = np.arctan(450 / D * (2*np.pi))
60 beta = np.arctan(450 / D * (2*np.pi))
61 kexi_0 = np.arctan(450 / D * (2*np.pi))
62 R = 1 / 3 * np.sqrt(450**2 + (D/(2*np.pi))**2)
63 S_2 = 2 * np.pi * 2 * R * 2 * alpha / (2 * np.pi)
64 S_3 = 2 * np.pi * R * 2 * beta / (2 * np.pi)
65
66 # 出去角度
67 x_out = np.pi + 2*np.pi*450/D
68
69 # 由S解kexi
70 def judge_1(S):
71     if S <= 100 * 100:
72         def f(x):
73             return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x)) - (begin_S - S)
74         x = newton(f, -20, tol=1e-10)

```

```

73     kexi = -x + 450*2*np.pi/D
74 elif S >= 100 * 100 and S <= 100*100 + S_2:
75     alpha_0 = (S - 100 * 100)/(2*R)
76     kexi = 2*alpha_0/3
77 elif S >= 100*100 + S_2 and S <= 100*100 + S_2 + S_3:
78     beta_0 = (S - 100 * 100 - S_2)/(R)
79     kexi = (beta_0 + 4*kexi_0)/3
80 else:
81     S_out = A * ((x_out-np.pi) * np.sqrt(1 + (x_out-np.pi)**2) +
82                   np.arcsinh(x_out-np.pi))
83     def f(x):
84         return A * ((x-np.pi) * np.sqrt(1 + (x-np.pi)**2) +
85                     np.arcsinh(x-np.pi)) - S_out - (S-S_2-S_3-100*100)
86     x = newton(f, 1, tol=1e-10)
87     kexi = x + 2*kexi_0 - np.pi - (2*np.pi*450)/D
88     return kexi
89
90 #由kexi解位置
91 def distance(kexi):
92     if kexi <= 0:
93         x = -kexi + 450/(D/(2*np.pi))
94         x_coord = a_0 * x * np.cos(x)
95         y_coord = a_0 * x * np.sin(x)
96     elif kexi <= 4/3*kexi_0:
97         theta = x_constant - 2*2*np.pi
98         rotation_angle = theta - 0.5*np.pi
99         alpha_0 = 3/2*kexi
100        delta = alpha - alpha_0
101        point = (2 * R * np.cos(delta) + 0_1x, 2 * R * np.sin(delta) + 0_1y)
102        rotation_matrix = np.array([[np.cos(rotation_angle),
103                                     -np.sin(rotation_angle)],
104                                     [np.sin(rotation_angle), np.cos(rotation_angle)]])
105        (x_coord, y_coord) = np.dot(rotation_matrix, point)
106    elif kexi <= 2*kexi_0:
107        theta = x_constant - 2*2*np.pi
108        rotation_angle = theta - 0.5*np.pi
109        beta_0 = 3*kexi - 4*kexi_0
110        delta = beta - beta_0
111        point = (-R * np.cos(delta) + 0_2x, R * np.sin(delta) + 0_2y)
112        rotation_matrix = np.array([[np.cos(rotation_angle),
113                                     -np.sin(rotation_angle)],
114                                     [np.sin(rotation_angle), np.cos(rotation_angle)]])
115        (x_coord, y_coord) = np.dot(rotation_matrix, point)
116    else:
117        x = kexi - 2*kexi_0 + np.pi + 2*np.pi*450/D
118        x_coord = D/(2*np.pi)*(x-np.pi)*np.cos(x)
119        y_coord = D/(2*np.pi)*(x-np.pi)*np.sin(x)

```

```

116
117     return (x_coord, y_coord)
118 #由kexi解速度方向
119 def direction(kexi):
120     if kexi <= 0:
121         x = -kexi + 450/(D/(2*np.pi))
122         v_x = 1/np.sqrt(x**2+1)*(x*np.sin(x)-np.cos(x))
123         v_y = 1/np.sqrt(x**2+1)*(-x*np.cos(x)-np.sin(x))
124     elif kexi <= 4/3*kexi_0:
125         alpha_0 = 3/2*kexi
126         v_x = np.cos(2*np.pi*450/D-np.pi+kexi_0-alpha_0)
127         v_y = np.sin(2*np.pi*450/D-np.pi+kexi_0-alpha_0)
128     elif kexi <= 2*kexi_0:
129         beta_0 = 3*kexi - 4*kexi_0
130         v_x = np.cos(2*np.pi*450/D-np.pi-kexi_0+beta_0)
131         v_y = np.sin(2*np.pi*450/D-np.pi-kexi_0+beta_0)
132     else:
133         x = kexi - 2*kexi_0 + np.pi + 2*np.pi*450/D
134         v_x = 1/np.sqrt(1+(x-np.pi)**2)*((x-np.pi)*np.sin(x)-np.cos(x))
135         v_y = 1/np.sqrt(1+(x-np.pi)**2)*(-(x-np.pi)*np.cos(x)-np.sin(x))
136
137     return (-v_x, -v_y)
138
139 # 存储速度数据
140 all_Vs = []
141
142 for t in np.arange(0, 201):
143     print(t)
144     S = 100 * t
145     kexi = judge_1(S)
146     point = distance(kexi)
147     points = [(point, kexi)]
148     kexi_old = kexi
149     V = 100
150     (v_xold, v_yold) = direction(kexi)
151     V_xs = [v_xold * V / np.sqrt(v_xold**2 + v_yold**2)]
152     V_ys = [v_yold * V / np.sqrt(v_xold**2 + v_yold**2)]
153     V_s=[np.sqrt((v_xold * V / np.sqrt(v_xold**2 + v_yold**2)))**2+(v_yold * V /
154                                         np.sqrt(v_xold**2 + v_yold**2))**2]
155
156     for i in np.arange(0, 223):
157         if i == 0:
158             L = 341 - 55
159         else:
160             L = 165
161
162         # 上一个点

```

```

162     (x_old, y_old) = point
163
164     def f(xi):
165         x_new, y_new = distance(xi)
166         return (x_new - x_old)**2 + (y_new - y_old)**2 - L**2
167
168     try:#自适应调整迭代初值
169         if kexi_old > 2 * kexi_0:
170             solve_kexi = newton(f, kexi_old - 0.5, tol=1e-3, maxiter=500)
171             if -solve_kexi + kexi_old < np.pi and abs(solve_kexi - kexi_old) >
172                 1e-6:
173                 point = distance(solve_kexi)
174                 kexi_old = solve_kexi
175                 points.append((point, solve_kexi))
176             else:
177                 solve_kexi = newton(f, kexi_old - 1.2, tol=1e-3, maxiter=500)
178                 if -solve_kexi + kexi_old < np.pi and abs(solve_kexi - kexi_old) >
179                     1e-6:
180                     point = distance(solve_kexi)
181                     kexi_old = solve_kexi
182                     points.append((point, solve_kexi))
183
184         # 计算速度
185         (v_x, v_y) = direction(solve_kexi)
186         # 上一个点和现在点
187         rx_old = x_old
188         ry_old = y_old
189         (rx_new, ry_new) = point
190         l_x = rx_old - rx_new
191         l_y = ry_old - ry_new
192         # 速度大小递推
193         V = V * (v_xold * l_x + v_yold * l_y) / (v_x * l_x + v_y * l_y)
194         v_xold = v_x
195         v_yold = v_y
196         V_xs.append(v_x * V / np.sqrt(v_x**2 + v_y**2))
197         V_ys.append(v_y * V / np.sqrt(v_x**2 + v_y**2))
198         V_s.append(abs(V))
199
200
201
202     all_Vs.append(V_s)
203
204
205     # 将速度数据保存到Excel文件
206     df_x = pd.DataFrame(all_Vs).T

```

207 | df_x.to_excel('V.xlsx', index=True, header=False)

6.3 save_xy.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.optimize import newton
4 import pandas as pd # 导入pandas库
5
6 # 具体指定参数
7 D=170
8 a_0 = D / (2 * np.pi)
9
10 # 求导函数定义
11 def f_0(y, x):
12     return x**2 + y**2 - 2 * x * y * np.cos(x - y) - (165)**2 / (a_0)**2
13
14 def f_1(y, x):
15     return 2 * y - 2 * x * np.cos(x - y) - 2 * x * y * np.sin(x - y)
16
17 def f_2(y, x):
18     return 2 - 2*x*np.sin(x-y) - 2*x*np.sin(x-y) - 2*x*y*np.cos(x-y)
19
20 # Halley数值解
21 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
22     y = y_initial
23     for _ in range(max_iter):
24         f0 = f_0(y, x)
25         f1 = f_1(y, x)
26         f2 = f_2(y, x)
27         if i == 0:
28             f0 = x**2 + y**2 - 2 * x * y * np.cos(x - y) - (341 - 2 * 27.5)**2 /
29                         (a_0)**2
30
31         delta_y = f0 / f1 * 1 / (1 - f0 * f2 / (2 * f1**2))
32         y -= delta_y
33
34         if abs(delta_y) < tol:
35             break
36
37     return y
38
39 # 两个圆心
40 O_1x = -2 / 3 * 170/(2*np.pi)
41 O_1y = 150
```

```

42 0_2x = 170/(2*np.pi) / 3
43 0_2y = -300
44 # 捉头角度
45 x_constant = 450 / a_0
46 A = 0.5 * D / (2 * np.pi)
47 x_0 = 2 * np.pi * 16
48 all_value = A * (x_0 * np.sqrt(1 + x_0**2) + np.arcsinh(x_0))
49 #螺线长度公式
50 def f(x):
51     return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x))-(A*(x_constant * np.sqrt(1
52         + x_constant**2) + np.arcsinh(x_constant))+100*100)
53
54 # 起点角度与长度
55 x_begin = newton(f, 1, tol=1e-10)
56 begin_S=A*(x_constant * np.sqrt(1 + x_constant**2) +
57             np.arcsinh(x_constant))+100*100
58
59 #两段弧长
60 alpha = np.arctan(450 / D*(2*np.pi))
61 beta = np.arctan(450 / D*(2*np.pi))
62 kexi_0 = np.arctan(450 / D*(2*np.pi))
63 R = 1 / 3 * np.sqrt(450**2 + (D/(2*np.pi))**2)
64 S_2 = 2 * np.pi * 2 * R * 2 * alpha / (2 * np.pi)
65 S_3 = 2 * np.pi * R * 2 * beta / (2 * np.pi)
66
67 #出去角度
68 x_out=np.pi+2*np.pi*450/D
69 #由距离解kexi
70 def judge_1(S):
71     if S <= 100 * 100:
72         def f(x):
73             return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x)) - (begin_S - S)
74         x = newton(f, -20, tol=1e-10)
75         kexi=-x+450*2*np.pi/D
76     elif S >= 100 * 100 and S <= 100*100 +S_2:
77         alpha_0=(S - 100 * 100)/(2*R)
78         kexi=2*alpha_0/3
79     elif S >= 100*100 +S_2 and S <= 100*100 + S_2+S_3:
80         beta_0=(S - 100 * 100-S_2)/(R)
81         kexi=(beta_0+4*kexi_0)/3
82     else:
83
84         S_out=A * ((x_out-np.pi) * np.sqrt(1 + (x_out-np.pi)**2) +
85                     np.arcsinh(x_out-np.pi))
86         def f(x):
87             return A * ((x-np.pi) * np.sqrt(1 + (x-np.pi)**2) +
88                         np.arcsinh(x-np.pi))-S_out-(S-S_2-S_3-100*100)

```

```

85     x= newton(f, 1, tol=1e-10)
86     kexi=x+2*kexi_0-np.pi-(2*np.pi*450)/D
87     return kexi
88 #由kexi解位置
89 def distance(kexi):
90     if kexi <= 0 :
91         x = -kexi + 450/(D/(2*np.pi))
92         x_coord= a_0 * x * np.cos(x)
93         y_coord= a_0 * x * np.sin(x)
94     elif kexi <= 4/3*kexi_0:
95         theta = x_constant -2*2*np.pi
96         rotation_angle = theta - 0.5*np.pi
97         alpha_0=3/2*kexi
98         delta = alpha - alpha_0
99         point = (2 * R * np.cos(delta) + 0_1x, 2 * R * np.sin(delta) + 0_1y)
100        rotation_matrix = np.array([[np.cos(rotation_angle),
101                                    -np.sin(rotation_angle)],
102                                    [np.sin(rotation_angle), np.cos(rotation_angle)]])
103        (x_coord,y_coord) = np.dot(rotation_matrix, point)
104    elif kexi <= 2*kexi_0:
105        theta = x_constant -2*2*np.pi
106        rotation_angle = theta - 0.5*np.pi
107        beta_0=3*kexi-4*kexi_0
108        delta = beta - beta_0
109        point = (-R * np.cos(delta) + 0_2x, R * np.sin(delta) + 0_2y)
110        rotation_matrix = np.array([[np.cos(rotation_angle),
111                                    -np.sin(rotation_angle)],
112                                    [np.sin(rotation_angle), np.cos(rotation_angle)]])
113        (x_coord,y_coord) = np.dot(rotation_matrix, point)
114    else :
115        x=kexi-2*kexi_0+np.pi+2*np.pi*450/D
116        x_coord=D/(2*np.pi)*(x-np.pi)*np.cos(x)
117        y_coord=D/(2*np.pi)*(x-np.pi)*np.sin(x)
118
119
120 # 初始化变量
121 all_x = []
122 all_y = []
123 #遍历时间-100s认为是0s
124 for t in np.arange(0, 201):
125     S = 100 * t
126     kexi = judge_1(S)
127     point = distance(kexi)
128     points = [(point, kexi)]
129     kexi_old = kexi

```

```

130
131     for i in np.arange(0, 223):
132         if i == 0:
133             L = 341 - 55
134         else:
135             L = 165
136         (x_old, y_old) = point
137
138     def f(xi):
139         x_new, y_new = distance(xi)
140         return (x_new - x_old)**2 + (y_new - y_old)**2 - L**2
141
142     try:
143         # 自适应调整初值
144         if kexi_old > 2 * kexi_0:
145             solve_kexi = newton(f, kexi_old - 0.5, tol=1e-3, maxiter=500)
146
147         # 检查解是否比旧解小且不同
148         if -solve_kexi + kexi_old < np.pi and abs(solve_kexi - kexi_old) >
149             1e-6:
150             point = distance(solve_kexi)
151             kexi_old = solve_kexi
152             points.append((point, solve_kexi))
153         else:
154             solve_kexi = newton(f, kexi_old - 1.2, tol=1e-3, maxiter=500)
155
156         # 检查解是否比旧解小且不同
157         if -solve_kexi + kexi_old < np.pi and abs(solve_kexi - kexi_old) >
158             1e-6:
159             point = distance(solve_kexi)
160             kexi_old = solve_kexi
161             points.append((point, solve_kexi))
162     except RuntimeError:
163         print("no")
164
165     # 从 points 中提取所有的 x 和 y 坐标
166     x_coords = [p[0][0] for p in points]
167     y_coords = [p[0][1] for p in points]
168     all_x.append(x_coords)
169     all_y.append(y_coords)
170
171 # 将 x 和 y 坐标分别保存到两个 DataFrame 中，并写入 Excel 文件
172 df_x = pd.DataFrame(all_x).transpose()
173 df_y = pd.DataFrame(all_y).transpose()
174 with pd.ExcelWriter('points_coordinates.xlsx') as writer:
175     df_x.to_excel(writer, sheet_name='X_Coordinates', index=False)
176     df_y.to_excel(writer, sheet_name='Y_Coordinates', index=False)

```

```

175
176 print("x 和 y 坐标已保存到 points_coordinates.xlsx 文件中")
177
178 # 绘制所有点
179 for point, kexi in points:
180     if kexi < 0:
181         plt.plot(point[0], point[1], 'ko') # 黑色点
182     elif 0 <= kexi < 4 / 3 * kexi_0:
183         plt.plot(point[0], point[1], 'bo') # 蓝色点
184     elif 4 / 3 * kexi_0 <= kexi < 2 * kexi_0:
185         plt.plot(point[0], point[1], 'go') # 绿色点
186     else:
187         plt.plot(point[0], point[1], 'yo') # 黄色点
188
189 # 设置图形属性
190 plt.xlabel('x')
191 plt.ylabel('y')
192 plt.gca().set_aspect('equal', adjustable='box')
193 plt.show()

```

附录 G 问题五代码：divided.py

```

1 import numpy as np
2 from scipy.optimize import newton
3 import matplotlib.pyplot as plt
4
5 # 具体指定参数
6 D = 170
7 a_0 = D / (2 * np.pi)
8
9 # 求导函数定义
10 def f_0(y, x):
11     return x**2 + y**2 - 2 * x * y * np.cos(x - y) - (165)**2 / (a_0)**2
12
13 def f_1(y, x):
14     return 2 * y - 2 * x * np.cos(x - y) - 2 * x * y * np.sin(x - y)
15
16 def f_2(y, x):
17     return 2 - 2*x*np.sin(x-y) - 2*x*np.sin(x-y) - 2*x*y*np.cos(x-y)
18
19 # Halley方法
20 def halley_method(y_initial, x, i, tol=1e-15, max_iter=100):
21     y = y_initial
22     for _ in range(max_iter):
23         f0 = f_0(y, x)

```

```

24     f1 = f_1(y, x)
25     f2 = f_2(y, x)
26
27     if i == 0:
28         f0 = x**2 + y**2 - 2 * x * y * np.cos(x - y) - (341 - 2 * 27.5)**2 /
29             (a_0)**2
30
31     delta_y = f0 / f1 * 1 / (1 - f0 * f2 / (2 * f1**2))
32     y -= delta_y
33
34     if abs(delta_y) < tol:
35         break
36
37
38 #速度
39 v = 100
40
41 # 两个圆心
42 O_1x = -2 / 3 * 170/(2*np.pi)
43 O_1y = 150
44 O_2x = 170/(2*np.pi) / 3
45 O_2y = -300
46 # 掉头角度
47 x_constant = 450 / a_0
48 A = 0.5 * D / (2 * np.pi)
49 x_0 = 2 * np.pi * 16
50 all_value = A * (x_0 * np.sqrt(1 + x_0**2) + np.arcsinh(x_0))
51 #螺线长度公式
52 def f(x):
53     return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x)) - (A * (x_constant *
54         np.sqrt(1 + x_constant**2) + np.arcsinh(x_constant)) + v * 100)
55
56 # 起点角度与长度
57 x_begin = newton(f, 1, tol=1e-10)
58 begin_S = A * (x_constant * np.sqrt(1 + x_constant**2) + np.arcsinh(x_constant)) +
59     v * 100
60
61 #两段弧长
62 alpha = np.arctan(450 / D * (2 * np.pi))
63 beta = np.arctan(450 / D * (2 * np.pi))
64 kexi_0 = np.arctan(450 / D * (2 * np.pi))
65 R = 1 / 3 * np.sqrt(450**2 + (D / (2 * np.pi))**2)
66 S_2 = 2 * np.pi * 2 * R * 2 * alpha / (2 * np.pi)
67 S_3 = 2 * np.pi * R * 2 * beta / (2 * np.pi)
68
69 #出去角度

```

```

68 x_out = np.pi + 2 * np.pi * 450 / D
69 #由距离解kexi
70 def judge_1(S):
71     if S <= v * 100:
72         def f(x):
73             return A * (x * np.sqrt(1 + x**2) + np.arcsinh(x)) - (begin_S - S)
74             x = newton(f, -20, tol=1e-10)
75             kexi = -x + 450 * 2 * np.pi / D
76     elif S >= v * 100 and S <= v * 100 + S_2:
77         alpha_0 = (S - 100 * 100) / (2 * R)
78         kexi = 2 * alpha_0 / 3
79     elif S >= 100 * 100 + S_2 and S <= 100 * 100 + S_2 + S_3:
80         beta_0 = (S - 100 * 100 - S_2) / (R)
81         kexi = (beta_0 + 4 * kexi_0) / 3
82     else:
83         S_out = A * ((x_out - np.pi) * np.sqrt(1 + (x_out - np.pi)**2) +
84             np.arcsinh(x_out - np.pi))
85         def f(x):
86             return A * ((x - np.pi) * np.sqrt(1 + (x - np.pi)**2) + np.arcsinh(x -
87                 np.pi)) - S_out - (S - S_2 - S_3 - 100 * 100)
88         x = newton(f, 1, tol=1e-10)
89         kexi = x + 2 * kexi_0 - np.pi - (2 * np.pi * 450) / D
90     return kexi
91 #由kexi解位置
92 def distance(kexi):
93     if kexi <= 0:
94         x = -kexi + 450 / (D / (2 * np.pi))
95         x_coord = a_0 * x * np.cos(x)
96         y_coord = a_0 * x * np.sin(x)
97     elif kexi <= 4 / 3 * kexi_0:
98         theta = x_constant - 2 * 2 * np.pi
99         rotation_angle = theta - 0.5 * np.pi
100        alpha_0 = 3 / 2 * kexi
101        delta = alpha - alpha_0
102        point = (2 * R * np.cos(delta) + 0_1x, 2 * R * np.sin(delta) + 0_1y)
103        rotation_matrix = np.array([[np.cos(rotation_angle),
104            -np.sin(rotation_angle)],
105                            [np.sin(rotation_angle),
106                                np.cos(rotation_angle)]])
107        (x_coord, y_coord) = np.dot(rotation_matrix, point)
108    elif kexi <= 2 * kexi_0:
109        theta = x_constant - 2 * 2 * np.pi
110        rotation_angle = theta - 0.5 * np.pi
111        beta_0 = 3 * kexi - 4 * kexi_0
112        delta = beta - beta_0
113        point = (-R * np.cos(delta) + 0_2x, R * np.sin(delta) + 0_2y)
114        rotation_matrix = np.array([[np.cos(rotation_angle),

```

```

        -np.sin(rotation_angle)],
111                         [np.sin(rotation_angle),
112                          np.cos(rotation_angle)])
113
114     (x_coord, y_coord) = np.dot(rotation_matrix, point)
115
116 else:
117
118     x = kexi - 2 * kexi_0 + np.pi + 2 * np.pi * 450 / D
119     x_coord = D / (2 * np.pi) * (x - np.pi) * np.cos(x)
120     y_coord = D / (2 * np.pi) * (x - np.pi) * np.sin(x)
121
122
123 return (x_coord, y_coord)
#由kexi解速度方向
124 def direction(kexi):
125
126     if kexi <= 0:
127
128         x = -kexi + 450 / (D / (2 * np.pi))
129         v_x = 1 / np.sqrt(x**2 + 1) * (x * np.sin(x) - np.cos(x))
130         v_y = 1 / np.sqrt(x**2 + 1) * (-x * np.cos(x) - np.sin(x))
131
132     elif kexi <= 4 / 3 * kexi_0:
133
134         alpha_0 = 3 / 2 * kexi
135         v_x = np.cos(2 * np.pi * 450 / D - np.pi + kexi_0 - alpha_0)
136         v_y = np.sin(2 * np.pi * 450 / D - np.pi + kexi_0 - alpha_0)
137
138     elif kexi <= 2 * kexi_0:
139
140         beta_0 = 3*kexi - 4*kexi_0
141         v_x = np.cos(2*np.pi*450/D-np.pi-kexi_0+beta_0)
142         v_y = np.sin(2*np.pi*450/D-np.pi-kexi_0+beta_0)
143
144     else:
145
146         x = kexi - 2 * kexi_0 + np.pi + 2 * np.pi * 450 / D
147         v_x = 1 / np.sqrt(1 + (x - np.pi)**2) * ((x - np.pi) * np.sin(x) -
148                                         np.cos(x))
149         v_y = 1 / np.sqrt(1 + (x - np.pi)**2) * (- (x - np.pi) * np.cos(x) -
150                                         np.sin(x))
151
152
153     return (-v_x, -v_y)
#自定义优化目标函数
154 def Max(t):
155
156     S = 100 * t
157
158     kexi = judge_1(S)
159
160     point = distance(kexi)
161
162     points = [(point, kexi)]
163
164     kexi_old = kexi
165
166     V = 100
167
168     (v_xold, v_yold) = direction(kexi)
169
170     ratio = [1]
171
172     for i in np.arange(0, 223):
173
174         if i == 0:
175
176             L = 341 - 55
177
178         else:
179
180             L = 165

```

```

154     # 上一个点
155     (x_old, y_old) = point
156
157     def f(xi):
158         x_new, y_new = distance(xi)
159         return (x_new - x_old)**2 + (y_new - y_old)**2 - L**2
160
161     try:
162         # 自适应调整迭代初值
163         if kexi_old > 2 * kexi_0:
164             solve_kexi = newton(f, kexi_old - 0.5, tol=1e-3, maxiter=500)
165             if -solve_kexi + kexi_old < np.pi and abs(solve_kexi - kexi_old) >
166                 1e-6:
167                 point = distance(solve_kexi)
168                 kexi_old = solve_kexi
169                 points.append((point, solve_kexi))
170             else:
171                 solve_kexi = newton(f, kexi_old - 1.2, tol=1e-3, maxiter=500)
172                 if -solve_kexi + kexi_old < np.pi and abs(solve_kexi - kexi_old) >
173                     1e-6:
174                     point = distance(solve_kexi)
175                     kexi_old = solve_kexi
176                     points.append((point, solve_kexi))
177
178         # 计算速度
179         (v_x, v_y) = direction(solve_kexi)
180         # 上一个点和现在点
181         rx_old = x_old
182         ry_old = y_old
183         (rx_new, ry_new) = point
184         l_x = rx_old - rx_new
185         l_y = ry_old - ry_new
186         # 速度大小递推
187         V = V * (v_xold * l_x + v_yold * l_y) / (v_x * l_x + v_y * l_y)
188         ratio.append(abs(V / v))
189         v_xold = v_x
190         v_yold = v_y
191
192     except RuntimeError:
193         print("no")
194
195     return max(ratio)
196
197     def ternary_search(l, r, eps=1e-10):
198         points = []
199         while r - l > eps:

```

```

199     m1 = l + (r - l) / 3
200     m2 = r - (r - l) / 3
201     f1 = Max(m1)
202     f2 = Max(m2)
203     print(m1, m2)
204     points.append((m1, f1))
205     points.append((m2, f2))
206
207     if f1 < f2:
208         l = m1
209     else:
210         r = m2
211
212     return (l + r) / 2, points
213
214 # 三分搜索范围
215 l = 180
216 r = 200
217 best_t, points = ternary_search(l, r)
218 best_max_value = Max(best_t)
219 print(f"Best t: {best_t}")
220 print(f"Max(Max(t)): {best_max_value}")
221 print(f"龙头速度: {2 / best_max_value}")
222
223 # 提取所有的搜索点和对应的Max值
224 x_vals, y_vals = zip(*points)
225
226 # 绘制图像
227 plt.figure(figsize=(4, 6))
228 plt.plot([x - 100 for x in x_vals], y_vals, 'ro', label='Search Points')
229
230 # 为每个点作一条垂线到横轴
231 for x, y in points:
232     plt.plot([x - 100, x - 100], [0, y], 'r-', linewidth=1)
233
234 plt.scatter([best_t - 100], [best_max_value], color='blue', label='Best Point',
235             zorder=5) # 标记最佳点
236 plt.plot([best_t - 100, best_t - 100], [0, best_max_value], 'b-', linewidth=1)
237 plt.xlabel('t')
238 plt.ylim(0, 2)
239 plt.legend()
240 plt.show()

```