“板凳长龙”机理建模与设计优化

摘要

“板凳龙”，是起源于我国河洛地区的一种传统民俗活动，是我国南方许多省市的元宵庆祝活动之一，有丰富的文化内涵。本文主要分析了舞龙过程，研究舞龙路径、龙头速度等参数对舞龙过程的影响，建立了**基于刚体限制的舞龙队模型**，并利用**变步长搜索、二分法、模拟退火**等方法进行求解。

针对问题一：我们建立了**舞龙队位置速度模型**。首先，引入阿基米德螺线，以螺线起点为原点建立极坐标系，得到螺线方程为 $r(\theta) = b\theta$。以盘入瞬间为时间原点，建立时间 t 与龙头前把手极角 θ_0 之间的函数关系，以此求出龙头前把手的位置，并利用刚体限制，**将变步长搜索与二分法结合**，计算出所有把手的位置。然后，计算各把手所在位置的切线方向和各板凳方向，由刚体限制推导出速度计算公式，最终得到所有把手的位置和速度结果。结果详见表 2 、表 3 和文件 **result1.xlsx**。

针对问题二：证明了只需考虑第一、二块板凳是否与其他板凳发生碰撞，我们建立了**舞龙队碰撞检测模型**和两种**最小螺距计算模型**，结合变步长搜索和二分法，求解舞龙队盘入的终止时刻。在问题一的基础上，先确定所有板凳的位置信息，然后基于舞龙队碰撞检测模型，用变步长搜索对时间进行两轮迭代，将第一次碰撞时间锁定在较精确的小区间内，以确保没有漏过某次碰撞，且此区间不会出现其它碰撞情况。最后，借助二分法，求出终止时刻的精确值。得到碰撞时间为 **412.473838 s**，其它计算结果详见文件 **result2.xlsx**。

针对问题三：在问题二的基础上，建立了**螺距参数可调的舞龙队碰撞检测模型**。问题三的本质与问题二相同，只是螺距参数可调，以便于对不同螺距下的舞龙队进行碰撞检测。只需要改变螺距的值，求解出碰撞时刻的极径长度，与调头空间半径作对比，即可得到最终结果。考虑到时间坐标离散带来的误差影响，在实际解题过程中，我们采用了两种不同的做法：**二分法**、**模拟退火法**进行求解，两种方法得到的最小螺距分别为 **0.450337 m** 和 **0.450297 m**。

针对问题四：建立**调头位置可变的全路径舞龙队位置速度模型**，并求解出满足刚体限制的**最小调头半径为 4.254674 m** 。为了保持精度的同时简便计算，规定实际调头路径起始点和终止点呈中心对称，由此可证**调头路线形状唯一确定**，随后根据调头路线形状，确定了全路径在直角坐标系下的参数方程，并在编程过程中对不同把手所处不同曲线进行区分，最终得出所需时间段内各把手的位置速度信息，计算结果详见文件 **result4.xlsx**。

针对问题五：在问题四中模型的基础上，考虑速度大小限制条件，对龙头行进速度进行递减迭代，最终求得满足速度限制的**最大龙头行进速度为 0.409122 m · s^{-1}**。

关键词: 阿基米德螺线，刚体限制，变步长搜索法，二分法，模拟退火

1. 问题重述

板凳龙，是起源于我国河洛地区的传统民俗活动，是百姓迎神祈福的重要活动之一。板凳龙表演过程中有“盘龙”的舞蹈动作，我们对盘龙路线与相关参数建立了**基于刚体限制**的高精度数学模型，并以此通过计算解决以下问题。

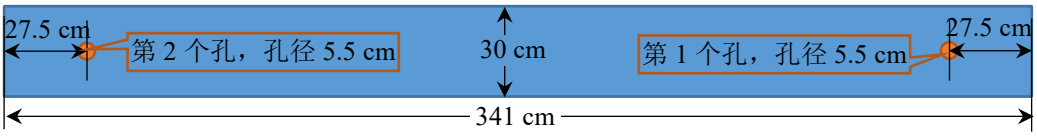


图 1: 龙头俯视图

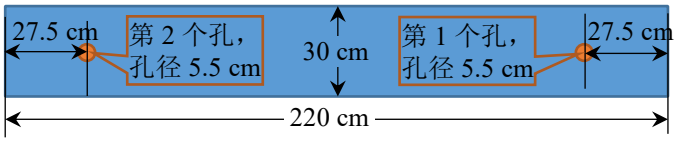


图 2: 龙身俯视图

问题一：“板凳龙”的行进路线是一条螺距为 55 cm 的阿基米德螺线，龙头为特殊的板凳，其它所有板凳有完全一致的结构，且相邻把手是等间距的（龙头前后把手除外），龙头从特定起点开始，沿阿基米德螺线匀速前进。以开始盘入为时间起点，求出 0 s ～ 300 s 时间段内，每秒整个舞龙队的的位置和速度。将最终结果保留 6 位小数，写入到文件 result1.xlsx 中，在论文中以表格形式给出部分结果。

问题二：舞龙队沿问题 1 设定的螺线路径盘入，以板凳之间不发生碰撞为限制，计算盘入的终止时刻，即发生第一次碰撞的前一瞬间。记录此时舞龙队的具体位置与速度，将结果写入文件 result2.xlsx。

问题三：舞龙队从盘入到盘出需要一定的调头空间，调头空间是一个直径 9 m、圆心位于螺线中心的圆，求解最小螺距，使得龙头前把手能够进入调头空间。

问题四：舞龙队需要在调头空间内完成调头，调头路径由两段半径比为 2:1 且相连为 S 形的圆弧构成，且与盘入盘出螺线相切。确定调头路径，并调整圆弧，使得调头曲线变短。然后设定龙头前把手速度为 1 $\mathrm{m \cdot s^{-1}}$，给出调头前后时间段内，舞龙队的位置和速度。将结果保存到文件 result4.xlsx。

问题五：队伍沿问题 4 设定路径前行，龙头行进速度保持不变，队伍各把手速度均不超过 2 $\mathrm{m \cdot s^{-1}}$，求解龙头的最大行进速度。

2. 问题分析

2.1 问题一分析

对于问题一，题目已经给出“板凳龙”的行进路线是一条**阿基米德螺线**，考虑到阿基米德螺线的特性，我们考虑以曲线出发点为原点建立极坐标系,得到路径曲线方程为 $r = b\theta$，其中 b 由螺距确定。根据题目描述可知，除龙头上的两个把手外，其它所有相邻把手间都是等距的，所以只需要先求出第一、二块板凳的把手的位置，即沿用同样的思路，确定后续每个把手任意时刻的位置。由**刚体限制**推导出速度计算公式，并代入位置结果，就能求出所需的速度信息。

2.2 问题二分析

对于问题二，我们需要在问题一的基础上进行扩展。在问题一中，我们仅考虑了不同时刻下每个把手在阿基米德螺线上的位置和速度信息。在问题二中，我们需要根据每一对相邻把手的方向（板凳方向），**定位出其矩形位置**（板凳位置），这样就能模拟出板凳的全部信息，以便**建立碰撞检测模型**。又因为板凳龙中自第三个板凳开始，每一个板凳都一定会重复第二块板凳的运动，再结合盘入螺线的对称性，只需考虑的龙头和第二块板凳的外边缘棱角是否会同外圈板凳发生碰撞即可。因为题目要确定碰撞的时刻，我们考虑采用**变步长搜索法**，对行进时间 t 进行变步长迭代，并判断每步迭代后是否会发生碰撞。确定第一次碰撞时间后，回退多段时间防止漏测，再缩小步长进行二次迭代，以确定碰撞时间所在的小区间，最后利用**二分法**，二分碰撞时间，并根据是否碰撞的条件缩小时间范围，以达成优化精度的目的。

2.3 问题三分析

问题三的本质和问题二是完全一致的。对于问题三，我们只需改变螺距这一参数，重复问题二的计算过程以得到碰撞时龙头距原点的距离，与调头空间半径作比较，若在进入调头区域前不发生碰撞则符合题意。

在确定螺距的具体实现过程中，考虑到时间离散所带来的检测误差，我们采用**二分法和模拟退火两种不同方法**进行解答。对于二分法，作出图像后可以发现，在调头空间半径附近，碰撞时的半径随螺距大小的变化是**严格单调**的，所以很容易想到利用二分法快速解题。但为了防止遗漏某些情况，我们还可以采用模拟退火法，在一定范围内对螺距进行迭代，以搜寻最小螺距，其优点是可以自动寻找全局最优解，降低了陷入局部最优解的概率。

2.4 问题四分析

问题四分为两个部分。对于第一部分，需要考虑到进入调头空间后，板凳龙仍可以**继续沿螺线行进一段距离再开始调头**，也即在调头空间完成调头即可，并不是进入调头空间瞬间便开始调头。虽然实际调头路径的起始点和中止点可以不中心对称，但是这样会导致全路径曲线的参数方程求解困难。为了保持计算精度的同时，尽可能地简便计算，在后续问题中，我们**规定调头路径的起始点和中止点中心对称**。在此基础上，通过平面几何知识，可以证明，板凳龙的调头路径长度可以由调头起始点和原点连线的长度唯一确定，且**严格单调递增**。于是只需要考虑最晚何时调头可以保证板凳龙“合法”地走完全程（不破坏刚体限制）。由此可确定最短的调头路径，这便回归到了问题二，后续利用变步长搜索法即可计算出最小调头半径，得到板凳龙的行进路径。

确定路径之后，**求解全路径的参数方程**，以此建立板凳龙全路径位置速度模型，并计算所需结果。只需要注意确定每一秒所要求的把手所处的是哪一段曲线，对于所处不同的曲线段，应采用不同的速度计算方式。

2.5 问题五分析

问题五回归到第一问，只需考虑每个时刻各把手的位置和速度情况。因为要求是所有把手的最大速度均不大于 $2\ \mathrm{m\cdot s^{-1}}$，所以可以考虑利用**变步长搜索法**，将龙头速度设定为 $2\ \mathrm{m\cdot s^{-1}}$ 并设定变化的速度步长（龙头速度随迭代次数增大而减小），使板凳龙开始运动。每一步检查是否有把手速度超过 $2\ \mathrm{m\cdot s^{-1}}$，如果有则直接改变龙头速度使得当前违规把手的速度变为 $2\ \mathrm{m\cdot s^{-1}}$，最后如果所有把手都能正常通过调头区域则此时龙头的速度就是最大的行进速度。

3. 模型假设

(1) 假设龙头前把手以恒定速度沿阿基米德螺线运动，且过程中速度不变。
(2) 假设每节板凳可以视为刚体，在运动过程中其长宽保持不变，且不会发生形变。
(3) 假设板凳之间通过把手连接时不发生相对滑动，即整个板凳龙作为一个整体运动。
(4) 假设每节板凳的密度均匀，即每节板凳的质量分布是均匀的。
(5) 假设所有把手理想连接，连接处无摩擦，且连接足够坚固，不会发生断裂。
(6) 假设舞龙场地绝对平坦且板凳始终保持绝对水平。

4. 符号说明

表 1: 符号含义与约定

符号	说明	单位
r	当前点到螺线起点的距离	m
θ	极角	rad
θ_0	龙头（前把手）极角	rad
b	极坐标系下阿基米德曲线方程参量	m
t	板凳龙行进时间	s
α_i	速度公式推导中两点连线和速度向量的夹角	rad
t_{crushed}	碰撞时间	s
r_t	调头半径（调头起始点和原点连线的长度）	m
θ_t	调头时的极角	rad
$d_{\min_b}$	最小螺距	m
v_0	龙头行进速度	$\text{m}\cdot\text{s}^{-1}$
$v_{0,\max}$	龙头最大行进速度	$\text{m}\cdot\text{s}^{-1}$

5. 模型建立与求解

5.1 问题一

5.1.1 板凳龙位置速度模型

根据题目对于“板凳龙”的描述，我们可以很轻易地知道，龙的行进路线是一条**阿基米德螺线**。阿基米德螺线是一个点匀速离开一个固定点的同时又以固定的角速度绕该固定点转动而产生的轨迹。我们考虑以螺线起点为原点**建立极坐标**，则螺线方程 $r = r(\theta)$ 和弧长 s 分别为：

$$r = r(\theta) = b\theta, \quad s(\theta) = \int_0^\theta \sqrt{r^2 + r'^2}\,\mathrm{d}\theta = \frac{b}{2}\left(\theta\sqrt{1+\theta^2} + \ln(\theta + \sqrt{1+\theta^2})\right) \tag{1}$$

其中 b 的值等于螺距除以 2π，例如螺距为 0.55 m 时，$b = 0.087535$ m.

给定时间 t 后，由于龙头前把手速率恒定，由下面的方程，可以解得龙头前把手的极角 θ_0。

$$v_0 t = s(\theta_0) = \frac{b}{2}\left(\theta_0\sqrt{1+\theta_0^2} + \ln(\theta_0 + \sqrt{1+\theta_0^2})\right) \tag{2}$$

在确定龙头的位置后，我们以龙头的极角 θ_0 为初始值，以与龙头距离为 $d_0 = 2.86$ m 为判断界限，逐渐增大 θ，利用变步长搜索和二分法，得到第一节龙身的极角。除了

第一块板凳（龙头）外，所有板凳的物理结构完全一致，因此后续沿用同样的思路，以与龙头距离为 $d = 1.65$ m 为判断界限，即可推出曲线上每个把手的位置，也即获得了板凳龙的全部位置。改变时间 t，我们可以算出每一个时刻龙的每个把手所处的位置。

关于位置的计算：根据极坐标公式：$r = b\theta$，我们考虑当前点的角度 θ_i，由 θ_i 可以轻易地计算得 r_i，转为直角坐标系，可以得到当前点的平面直角坐标：

$$(x_i, y_i) = (r_i \cos\theta_i, r_i \sin\theta_i) = (b\theta_i \cos\theta_i, b\theta_i \sin\theta_i) \tag{3}$$

则第 θ_i 个点与第 $\theta i - 1$ 个之间的距离为：

$$\begin{aligned}\text{distance} &= \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \\ &= \sqrt[2]{(b\theta_i \cos\theta_i - b\theta_{i-1}\cos\theta_{i-1})^2 + (b\theta_i \sin\theta_i - b\theta_{i-1}\sin\theta_{i-1})^2}\end{aligned}$$

关于速度的计算：已知的是龙头的速度为 1 m · s^{-1}，我们考虑采用递推的方式得到后续每一个点的速度。

如图 3 所示，考虑由第 i 个点的速度计算第 $i+1$ 个点的速度。其中 $\vec{x}$ 为从第 $i+1$ 个把手位置指向第 i 个把手位置的向量。和 $\vec{v}_i$、$\vec{v}_{i+1}$ 的夹角分别为 α_i，α_{i+1}。此外，记 $\vec{v}_{i+1}$ 的切线方向为 $\vec{\tau}_{i+1}$，$\vec{v}_i$ 的切线方向为 $\vec{\tau}_i$。

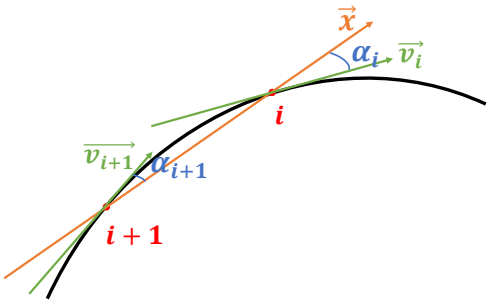


图 3: 螺线上两点间关系示意图

刚体限制方程、直角坐标系中的螺线切线方向分别为，

$$|\vec{v}_i| \cos\alpha_i = |\vec{v}_{i+1}| \cos\alpha_{i+1} \mathrm{d}\alpha \tag{4}$$

$$\vec{\tau} = -(\cos\alpha - \alpha\sin\alpha, \sin\alpha + \alpha\cos\alpha) \tag{5}$$

又 $\cos\alpha = \frac{|\vec{\tau}\cdot\vec{x}|}{|\vec{\tau}||\vec{x}|}$，代回递推式中得到

$$|\vec{v}_{i+1}| = \frac{|\vec{\tau}_{i+1}|\,|\vec{\tau}_i \cdot \vec{x}|}{|\vec{\tau}_i|\,|\vec{\tau}_{i+1}\cdot\vec{x}|} \cdot |\vec{v}_i| \tag{6}$$

题目已经给出了初始条件（龙头的行进速度），即 $|\vec{v}_0|$，由此只需要计算出每个点的 $\vec{\tau}$ 即可通过递推的方法求得每个点的速度。

5.1.2 模型求解结果

计算结果已经保存到 **result1.xlsx** 中，这里给出 0 s、60 s、120 s、180 s、240 s、300 s 时，龙头前把手、龙头后面第 1、51、101、151、201 节、龙身前把手和龙尾后把手的位置和速度，详见表 2 和表 3。

表 2: 问题 1 中各时刻板凳龙不同部位所处位置

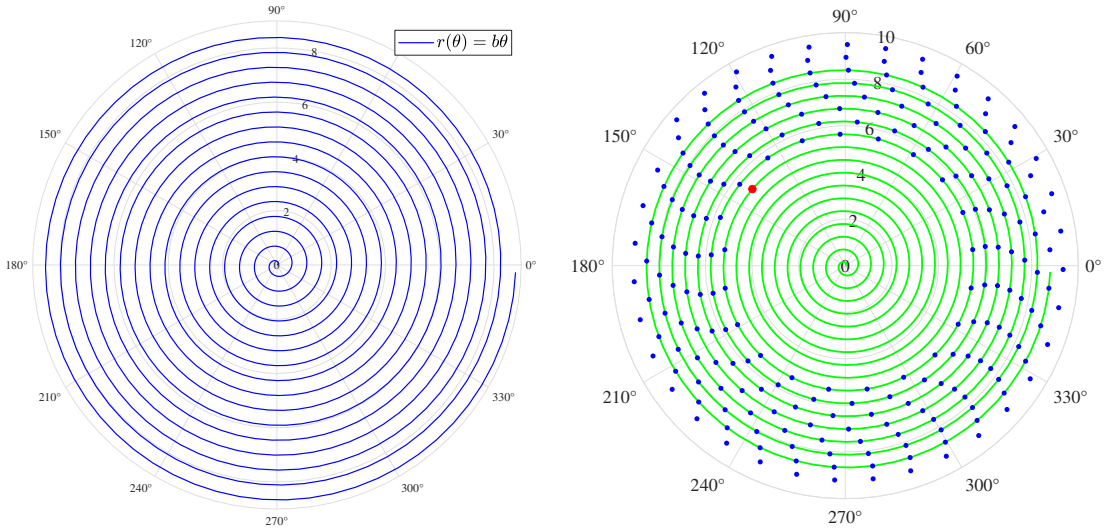
	0 s	60 s	120 s	180 s	240 s	300 s
龙头 x (m)	8.800000	5.799209	-4.084887	-2.963609	2.594494	4.420274
龙头 y (m)	0.000000	-5.771092	-6.304479	6.094780	-5.356743	2.320429
第 1 节龙身 x (m)	8.363824	7.456758	-1.445473	-5.237118	4.821221	2.459489
第 1 节龙身 y (m)	2.826544	-3.440399	-7.405883	4.359627	-3.561949	4.402476
第 51 节龙身 x (m)	-9.518732	-8.686317	-5.543150	2.890455	5.980011	-6.301346
第 51 节龙身 y (m)	1.341137	2.540108	6.377946	7.249289	-3.827758	0.465829
第 101 节龙身 x (m)	2.913983	5.687116	5.361939	1.898794	-4.917371	-6.237722
第 101 节龙身 y (m)	-9.918311	-8.001384	-7.557638	-8.471614	-6.379874	3.936008
第 151 节龙身 x (m)	10.861726	6.682311	2.388757	1.005154	2.965378	7.040740
第 151 节龙身 y (m)	1.828753	8.134544	9.727411	9.424751	8.399721	4.393013
第 201 节龙身 x (m)	4.555102	-6.619664	-10.627211	-9.287720	-7.457151	-7.458662
第 201 节龙身 y (m)	10.725118	9.025570	1.359847	-4.246673	-6.180726	-5.263384
龙尾（后）x (m)	-5.305444	7.364557	10.974348	7.383896	3.241051	1.785033
龙尾（后）y (m)	-10.676584	-8.797992	0.843473	7.492370	9.469336	9.301164

表 3: 问题 1 中各时刻板凳龙不同部位的速度

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

5.1.3 部分结果可视化

盘入螺线和 t=290 s 时板凳龙位置示意图如图 4 所示。



(a) 阿基米德螺线示意图 **(b) t = 290 s 时板凳龙全把手位置示意图**

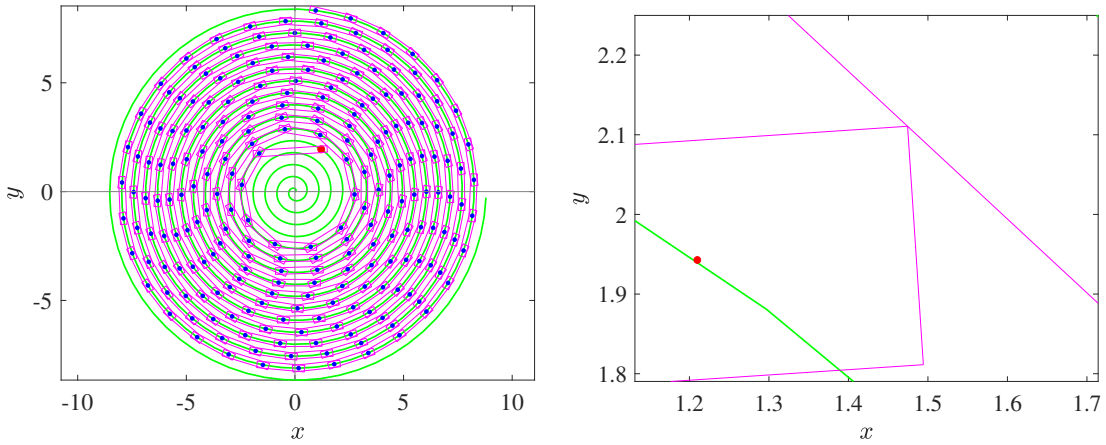
图 4: 部分结果可视化

5.2 问题二

5.2.1 舞龙队碰撞检测模型模型

对于问题二，我们可以根据问题一的做法，得到每一对把手的位置信息，而每一对相邻把手可以定位一张板凳。所以，考虑每一对相邻把手，我们向外扩展出一个矩形，最终的扩展结果即为板凳龙的俯视效果。通过矩形顶点的位置关系，可以判断板凳间是否存在重叠情况，进而得知是否发生碰撞。

撞击过程如图所示：



(a) 碰撞瞬间舞龙队位置 **(b) 碰撞点局部放大**

图 5: 撞击过程

我们可以进一步优化对于碰撞过程的判定。因为除龙头外所有的板凳完全一致，所以第三块和后续所有板凳一定会**重复第二块板凳的运动**，因此我们只需要考虑第一、二块板凳是否会与其他板凳发生碰撞。又因为第一、二块板凳的内圈不存在其他板凳，不需要考虑。所以最终我们只需要定位第一、二块板凳**外侧的两个顶点**，共计四个点是否在其他矩形内即可判断是否发生碰撞。

判断是否发生碰撞的具体方法推导如下：如图所示，下方的板凳为龙头或第二块板凳（余下的无需考虑），以 i 号板凳（对应 i 号把手和 $i+1$ 号把手）的 1 号点为新坐标系的原点，以 $\vec{v}_{i,1}$ 和 $\vec{v}_{i,2}$ 为基底，建立新直角坐标系 O'。记 $\vec{p}_{i,k} = (p_{i,k,x},\ p_{i,k,y})$ 为第 i 号板凳的 k 号顶点坐标，则有

$$\vec{v}_{i,1} = \vec{p}_{i,2} - \vec{p}_{i,1} = (p_{i,2,x} - p_{i,1,x}\ ,\ p_{i,2,y} - p_{i,1,y}) \tag{7}$$

$$\vec{v}_{i,2} = \vec{p}_{i,4} - \vec{p}_{i,1} = (p_{i,4,x} - p_{i,1,x}\ ,\ p_{i,4,y} - p_{i,1,y}) \tag{8}$$

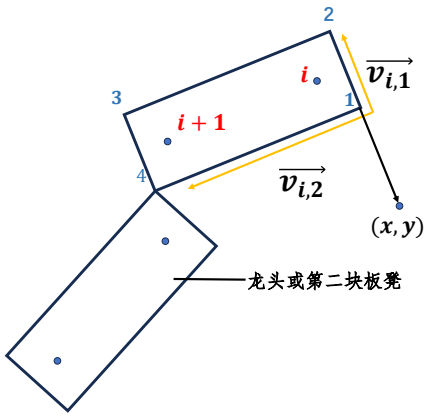


图 6: 发生碰撞的情形示意图

接下来进行坐标系的转换。在线性空间 V 中，设 $\vec{x} = (x, y) \in \mathbb{R}^2$，$v = \vec{x} \cdot \vec{v}_0$，其中 $\vec{v}_0$ 为 V 的一组基，则对于任意的线性映射 φ，我们有：

$$\varphi(v) = \vec{x} A_\varphi \vec{v}_0 \tag{9}$$

代入本题，得到坐标系变换公式：

$$\vec{x}' = (a, b) = (\vec{x} - \vec{x}_O) A_\varphi^{-1} \tag{10}$$

其中 $\vec{x}'$ 表示在新坐标系 O 下的位置向量，转换矩阵 A_φ 为

$$A_\varphi = \begin{bmatrix} p_{i,2,x} - p_{i,1,x} & p_{i,2,y} - p_{i,1,y} \\ p_{i,4,x} - p_{i,1,x} & p_{i,4,y} - p_{i,1,y} \end{bmatrix} \tag{11}$$

至此上式，可以解出转坐标系后的新一组坐标 (a, b)。若 $a \in (0, 1)$ 且 $b \in (0, 1)$ 则发生碰撞，否则未发生碰撞。

当判断完成后我们还需要进一步考虑如何得到精确的碰撞时间，考虑使用**变步长迭代法**，对时间 t 进行迭代。

变步长迭代法实现流程如下：第一次迭代为粗测，迭代步长较长。将时间步长设置的稍大一些，在第一次发现碰撞情况后停止。由于步长较大，可能出现漏测情况，所以将时间向前回调十个步长，并可认为这一时间内范围内发生了第一次碰撞。

接下来，调小时间步长，在缩小的时间范围上进行二次迭代，将第一次碰撞的时间锁定在较精确的区间内，以确保没有漏过某次碰撞，且此区间不会出现其它碰撞情况。

最后利用**二分法**，下界 $\text{bound}_{\text{low}}$ 为最早的未碰撞时间，上界 bound_{up} 为最晚的已碰撞时间，不断二分时间，并判断当前时刻是否发生碰撞，以此提高精度。

5.2.2 模型求解结果

求解上述模型，得到：

$$\textbf{碰撞时间 } t_{crushed} = \textbf{412.473838s} \tag{12}$$

其它计算结果详见文件 **result2.xlsx**。

5.3 问题三：

问题三和问题二的**本质是相同的**。只需要改变螺距，再通过问题二中的模型计算出碰撞时对应的半径（后称为碰撞半径）。若碰撞半径小于调头半径（龙头在调头区内），则符合题目条件，可以将螺距进一步缩小，否则不符合题目条件，需要调大螺距。

考虑到所有板凳的宽度为 0.3 m，螺距的大小不应小于 0.35 m。在实际计算螺距时，我们有**两种不同的思路**，下面依次说明。

5.3.1 思路一：基于二分法的最小螺距求解模型

作出螺距大小（横坐标）与碰撞半径（纵坐标）之间的关系图，如图 7 和图 8 所示。由图可知，在调头半径附近，碰撞半径和螺距大小是**严格单调**的，因此很自然地考虑到，利用**二分法**对制定螺距范围进行二分，并得到精度优秀的结果。

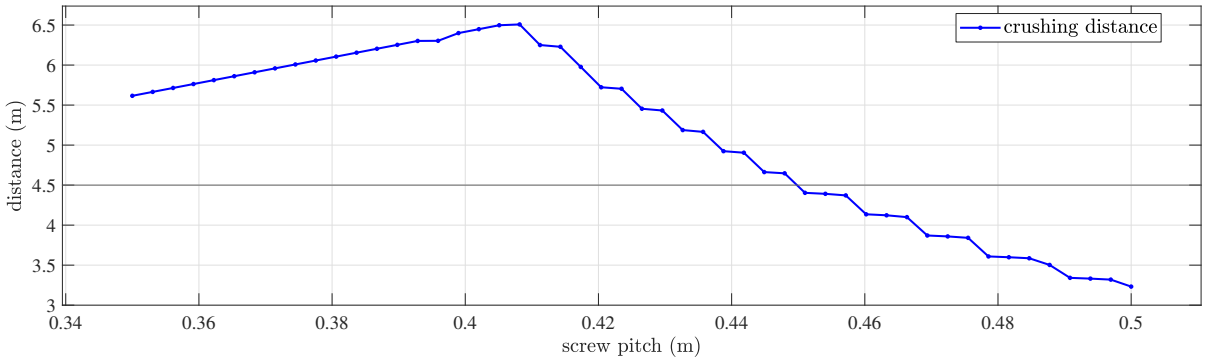


图 7: 螺距范围 [0.35 m, 0.55 m]

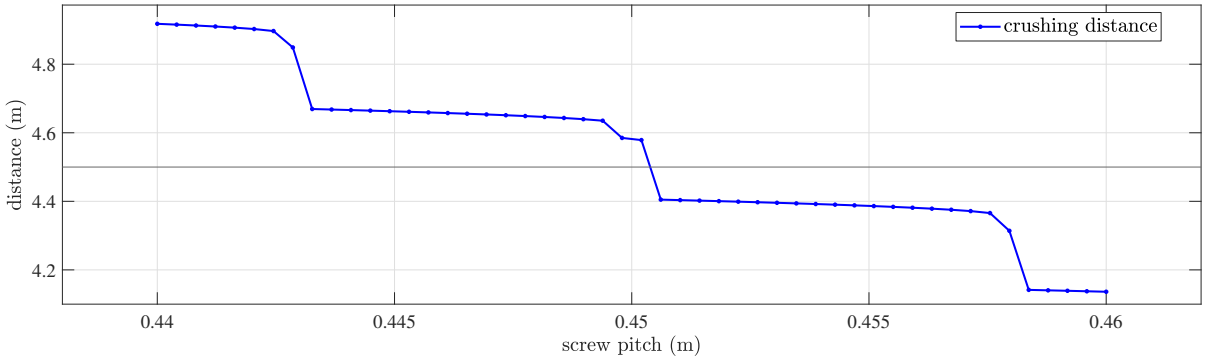


图 8: 螺距范围 [0.44 m, 0.46 m]

5.3.2 思路二：基于模拟退火的最小螺距求解模型

模拟退火算法是基于蒙特卡洛迭代求解策略的一种**随机寻优算法**，该算法在搜索过程中引入随机变量，并以一定概率接受一个比当前解差的解，因此可以有效避免陷入局部最优解，找到**全局最优解**。

在本题中，我们的具体求解步骤如下：

(1) 初始化参数：设置退火初始温度 $T_0 = 50\ ^\circ\mathrm{C}$，温度下降系数 $\alpha = 0.98$，结束温度 $T_{end} = 0.1\ ^\circ\mathrm{C}$，马尔科夫链长度 $L_{\mathrm{mkv}} = 6$。仅有一个参数（螺距），参数范围为 $[0.35\ \mathrm{m}, 0.5\ \mathrm{m}]$。

(2) 生成新解：根据当前最优解，在合理范围下生成一定的扰动，并生成新解 $\vec{x}$。

(3) 检验新解：根据生成的新解，计算目标函数增量 ΔE，若满足 $\Delta E < 0$，则接受新解，否则计算接受新解的概率，判断是否接受新解，并进行降温。

(4) 结束退火：若当前温度达到终止温度 T_{end}，输出最终结果，结束退火过程。

模拟退火流程图如下：

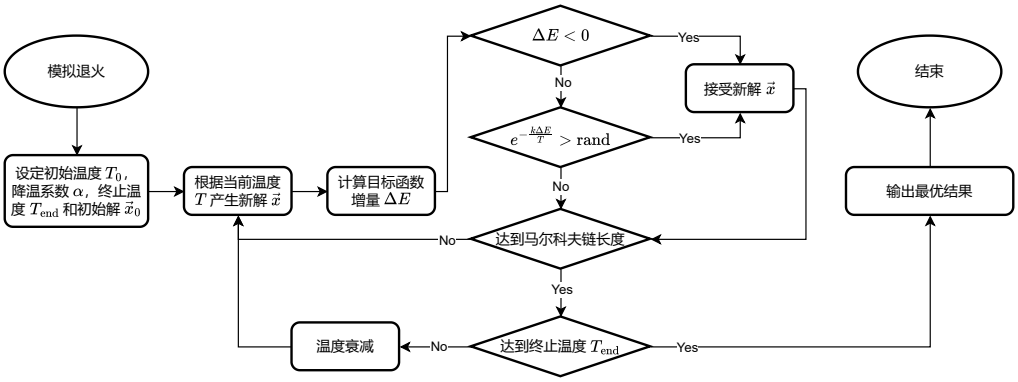


图 9: 模拟退火流程图

此外，我们还可以进行多次模拟，且尝试选择偏离正常螺距较大的数值作为起点进行模拟后，将得到的数据进行对比，分析结果的稳定性和收敛性。

5.3.3 模型求解结果

两种方法得到的最小螺距结果为：

二分法最小螺距: 0.450337 m，模拟退火最小螺距: 0.450297 m (13)

结果之间的**绝对误差为 0.008882 %**。两相对照之下可以认为两种做法均可行，得到的**结果合理且有较高精度**。

实际运行时的模拟退火过程如下，由退火过程图可知，模拟退火结果收敛性较好，且有较高精度。

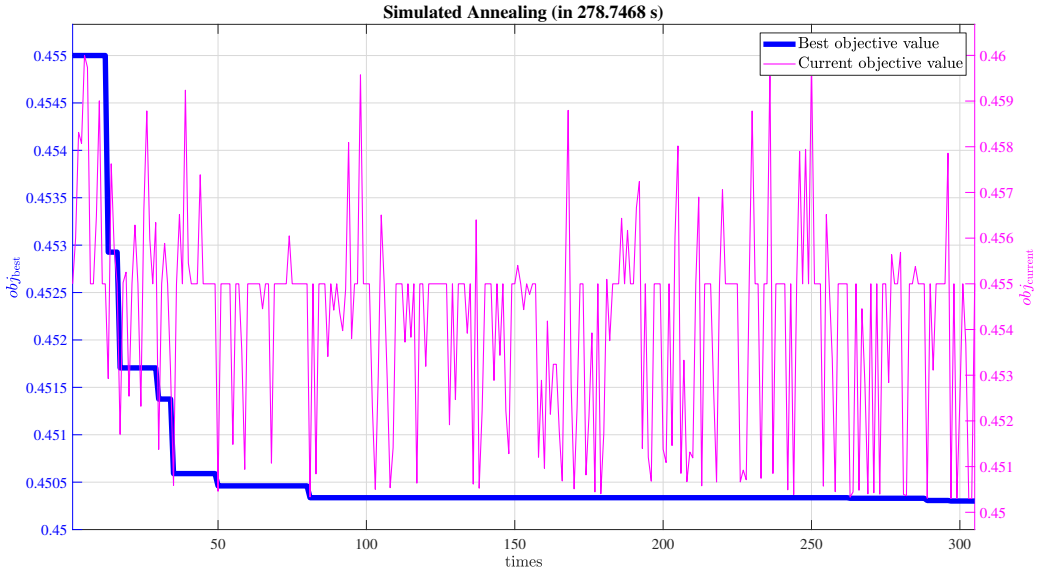


图 10: 模拟退火过程

5.4 问题四：

5.4.1 实际调头路径的几何性质

问题四可分为两部分，一部分是求解最优调头位置，另一部分是根据实际调头路径，求解位置和速度结果。

对于前半部分，考虑到板凳龙进入调头空间后**不一定马上开始调头**，而是可以继续沿着螺线前进。为了确定这一情况下的最短调头曲线，我们需要研究调头曲线的几何性质。我们注意到，实际调头路径的起始点和中止点可以不中心对称，这样会导致全路径曲线的参数方程求解困难。为了保持计算精度的同时，尽可能地简便计算，我们在后续问题中，**规定调头路径的起始点和中止点中心对称**。

在规定了中心对称后，可以得到如下结论：板凳龙进入调头区域后，**沿螺线走的越远（越晚开始调头），调头曲线长度就越短**。为了证明这一结论，首先要证明：调头曲线**形状唯一**，并且，其长度只与实际调头起始点的半径长（与原点距离）有关，只会随开始调头时间等比缩放。

具体证明如下：

如图 11 所示，点 M 为盘入螺线与实际调头空间的交点，又因为盘出螺线与盘入螺线**中心对称**，所以点 H 为盘出点。作出螺线上点 H 处的切线，与圆交于点 E，点 M 处的切线，与圆交于点 F，由中心对称可知**两直线平行**，且四边形 $HDMF$ 构成矩形。

因为调头路径与盘入、盘出螺线**均相切**，所以调头路径前后两段圆弧所在的圆一定和 HD、MF **分别相切**。进而得到两段圆弧所在圆的圆心（设前半段为圆弧圆心为 C、后半段圆弧圆心为 A）一定分别位于直线 MD、HF 上。

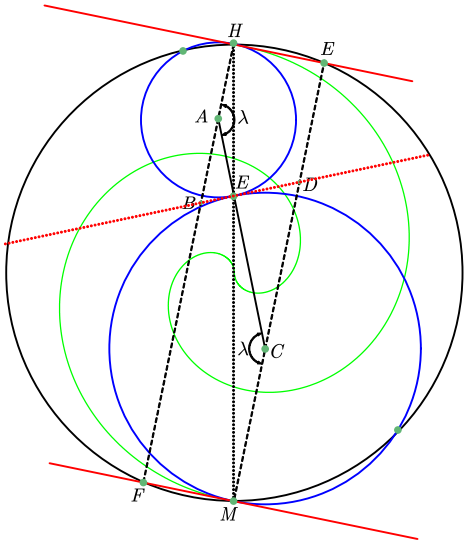


图 11: 调头区域示意图

由题目条件可知，两段圆弧所在的圆是相切的，设切点为 E，则有 AE、CE 均垂直于图中红色虚线 BD，所以 A、E、C **三点共线**。因为 MD 平行于 HF，所以 $\angle HAC = \angle MCA$。由题目条件可知两圆的半径比为 2:1。又有两个圆心角 $\angle HAC = \angle MCA$ 相等，所以可知，$ME : EH = 2 : 1$，因此，点 E 可以被唯一确定。由此两个圆**是唯一确定的**，所以整条调头路径**是唯一确定的**, 所以整条调头曲线的长度只会守两个圆的半径影响，而两圆的半径又能被调头区域的直径**唯一确定**，因此调头曲线的长度**仅取决于调头起始点与圆心连线的长度**。

又由题目可知两段圆弧的长度比为 $\overset{\frown}{ME} : \overset{\frown}{EH} = 2 : 1$，而两端弧的圆心角相等，所以可以推出两个圆的**半径长度比也为** $2 : 1$。

考虑到半径长为 $2 : 1$，圆心角相同，所以 $ME : EH = 2 : 1$，因此 E 点可以**被唯一确定**，由此两个圆**是唯一确定的**，所以整条调头路径**是唯一确定的**, 所以整条调头曲线的长度只会守两个圆的半径影响，而两圆的半径又能被调头区域的直径**唯一确定**，因此调头曲线的长度**仅取决于调头起始点与圆心连线的长度**。**证毕**。

通过上述证明，我们知道：想要得到最短的调头路径，需要在**“不违法”**的情况下尽可能晚的开始调头。这里的“违法”有两种情况：

(1) **发生碰撞**。即在最新路径上板凳龙在行进过程中会发生板凳间的碰撞。

(2) **违背刚体限制**。当某一个板凳在运动过程中，出现了板凳头尾速度向量点乘该板凳本身的速度向量的两个结果异号，这意味着下一瞬间，两节点间的距离会被拉长，也即板凳上的节点长度发生变化，违背了刚体性质。

5.4.2 求解全路径参数方程

但在开始迭代前，需要先解出两个圆的方程，以得到调头路径，才能进行进一步编写代码。所以，这里给出**曲线中两个圆的参数方程**的推导过程。

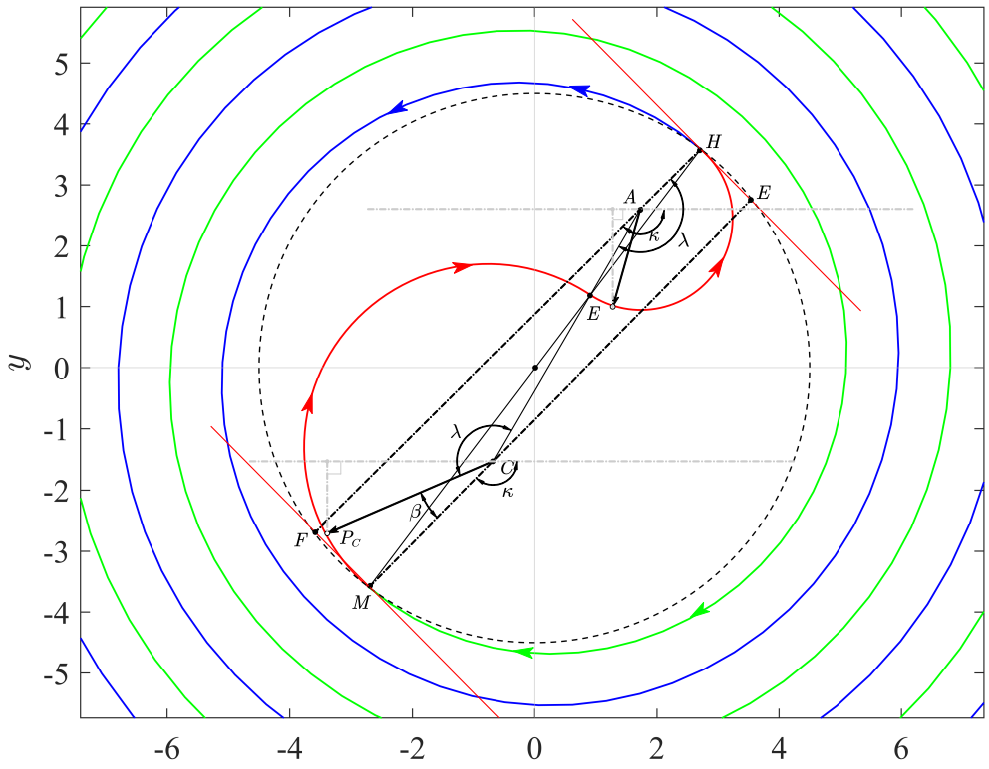


图 12: 调头路径参数方程求解

如图 12 所示，对于给定的 r_t，可根据阿基米德螺线方程 $r = b\theta$ 得到 $\theta_t = \frac{r_t}{b}$。点 M 为调头起始点，则有 $M = (b\theta_t \cos\theta_t, b\theta_t \sin\theta_t)$，又盘入盘出螺线中心对称，所以点 H 的坐标为 $H = (-b\theta_t \cos\theta_t, -b\theta_t \sin\theta_t)$。由先前证明可知，$ME : EH = 2 : 1$，于是 $HE = \frac{1}{3}r_t$，所以有 $OE = \frac{1}{2}EH = \frac{1}{3}HE$，即点 E 的坐标为 $E = (-\frac{1}{3}x_M, -\frac{1}{3}y_M)$。分别记点 M 处、点 H 处的（螺线）切线方向为 $\vec{l}_M$, $\vec{l}_H$（沿前进方向），则有：

$$\vec{l}_M = -[\cos(\theta_t) - \theta_t \sin(\theta_t),\ \ \sin(\theta_t) + \theta_t \cos(\theta_t)] \tag{14}$$

$$\vec{l}_H = [\cos(\theta_t) - \theta_t \sin(\theta_t),\ \ \sin(\theta_t) + \theta_t \cos(\theta_t)] \tag{15}$$

可求得 $\overrightarrow{HA}$ 的方向为：

$$\vec{l}_{HA} = [0, 0, 1] \times [\vec{l}_H, 0] = (x_{HA}, y_{HA}, 0) \tag{16}$$

进而得角 κ 的大小 $\kappa = \pi - \arctan\frac{y_{HA}}{x_{HA}}$。实际计算过程中因为 arctan 的值域范围是 $[-\frac{\pi}{2}, \frac{\pi}{2}]$，而 κ 的范围是 $[0, 2\pi]$，两者含义不符，因此除了需要减去 π，还需要判断 x_{HA} 的正负，并对 $x_{HA} < 0$ 时的 κ 进行修正，才能得到计算所需要的正确 κ 的值。具体而言，我们有：

$$\kappa = \begin{cases} \pi - \arctan\frac{y_{HA}}{x_{HA}}, & x_{HA} \leqslant 0 \\ -\arctan\frac{y_{HA}}{x_{HA}}, & x_{HA} > 0 \end{cases} \tag{17}$$

考虑用 $A = (x_A, y_A)$ 表示点 A 的坐标，$H = (x_H, y_H)$ 表示点 E 的坐标，以此类推。设点 A 为：

$$A = H + n \cdot \vec{l}_{HA} \Longrightarrow \begin{cases} x_A = x_H + n \cdot x_{HA} \\ y_A = y_H + n \cdot y_{HA} \end{cases} \tag{18}$$

其中 n 为待定未知量。根据半径相等，又有方程：

$$|\overrightarrow{AE}| = |\overrightarrow{AH}| \Longrightarrow f(n) = |E - A| - |H - A| = 0$$

解此方程，得到 n 的值，也即求出了点 A 的坐标。又 $\vec{OC} = \vec{OA} + 3\vec{AE}$，所以点 C 的坐标为 $C = 3E - 2A$，至此，两个圆心的坐标都已确定。

对于角度 λ，在小圆中由余弦定理可得：

$$\lambda = \arccos\left(1 - \frac{|\overrightarrow{HE}|^2}{2|\overrightarrow{AH}|^2}\right) \tag{19}$$

设原坐标系 O 的基向量为 $\vec{e}_1 = (1, 0)$、$\vec{e}_2 = (0, 1)$，大圆的半径为 R_1，小圆的半径为 R_2，可以求得圆 C 的参数方程：

$$P_C = C + \overrightarrow{CP} = C + R\cos(\beta + \kappa)\vec{e}_1 - R_1\sin(\beta + \kappa)\vec{e}_2 \tag{20}$$

$$\Longleftrightarrow \begin{cases} x = x_C + R_1 \cdot \cos(\beta + \kappa) \\ y = y_C - R_1 \cdot \sin(\beta + \kappa) \end{cases} \tag{21}$$

在圆 C 中 β 的正方向是顺时针，而在圆 A 中 β 的正方向是逆时针，因此结合角度关系后，作映射 $\beta \longmapsto -\beta$，得到圆 A 的参数方程为：

$$P_A = A + \overrightarrow{AP} = A - R_2\cos(-\beta + \kappa + \lambda)\vec{e}_1 + R_2\sin(-\beta + \kappa + \lambda)\vec{e}_2 \tag{22}$$

$$\Longleftrightarrow \begin{cases} x = x_A - R_2 \cdot \cos(-\beta + \kappa + \lambda) \\ y = y_A + R_2 \cdot \sin(-\beta + \kappa + \lambda) \end{cases} \tag{23}$$

至此我们就计算出了两个圆的参数方程，后续每一步迭代过程中都需要求出最新的调头路径，以判断是否发生碰撞，或是否存在某一时刻出现速度“违法”，即违背了刚体限制的情况（同一板凳的头尾速度向量分别点乘该板凳本身的速度向量，若两个结果异号，则破坏刚体限制）。

5.4.3 最小实际调头半径模型建立

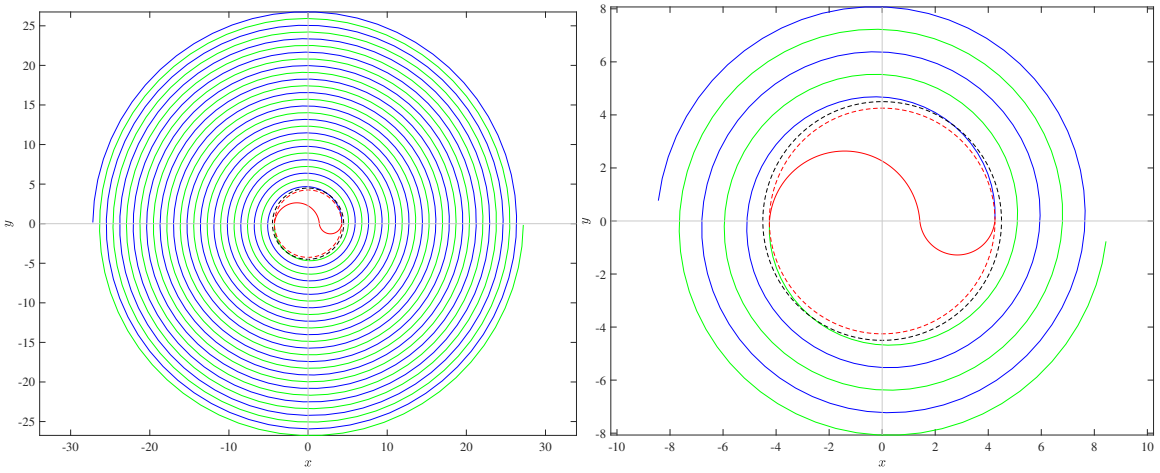
基于上述理论，我们考虑最晚在什么时刻开始调头是合法的，这一问题与问题 2 不谋而合，我们仍用变步长搜索法结合二分法来解决。考虑对实际调头半径 r_t 进行迭代，对于每一个 r_t，计算新的行进路线上是否出现“违法情况”，并逐步调整实际调头半径，经过变步长搜索后，得到较精确调头半径范围，再利用二分法得到高精度结果。

5.4.4 最小实际调头半径求解结果

求解上述模型，得到：

$$\textbf{最小实际调头半径为 } r_t = \textbf{4.254674 m} \tag{24}$$

由此可以得到板凳龙从**盘入、调头到盘出**的全过程路径图，如图 13 所示。其中，图 13 (b) 中的黑色虚线表示题目中给出的调头区域范围，红色虚线表示实际调头区域范围。



(a) 全路径曲线（盘入、调头、盘出） **(b) 调头空间与实际调头位置**

图 13: 板凳龙路径示意图

5.4.5 全路径位置速度模型建立

对于要求计算的数据，其中 [-100 s, 0 s) 部分，板凳龙还在盘入螺线上，这部分计算方式和问题一是完全相同的。而对于 [0 s, 100 s] 的部分，需要根据全路径参数方程计算出各把手的位置和速度。

在实际实现过程中，还需要注意判断当前把手所处的曲线为哪一段（盘入、调头、盘出）。所处位置不同，计算速度时需要的切线方向（速度方向）也不同。

盘入和盘出螺线上的节点的切线方向仍由式 5 给出，圆弧上的节点的切线方向由

下式给出：

$$\vec{\tau} = (x_\tau, y_\tau, 0) = \begin{cases} -[0,0,1] \times \overrightarrow{CP}, & \text{点 P 位于大圆弧} \\ [0,0,1] \times \overrightarrow{AP}, & \text{点 P 位于小圆弧} \end{cases} \tag{25}$$

由此可以计算任意时刻下的舞龙队位置和速度。

5.4.6 问题四后半部分求解结果

设定极角求解精度为 10^{-16} 进行求解，最终计算结果存放在 **result4.xlsx** 中。

图 14 为 $t = 100$ s 时，板凳龙各把手的位置结果可视化，其中粉色圆点表示龙头。

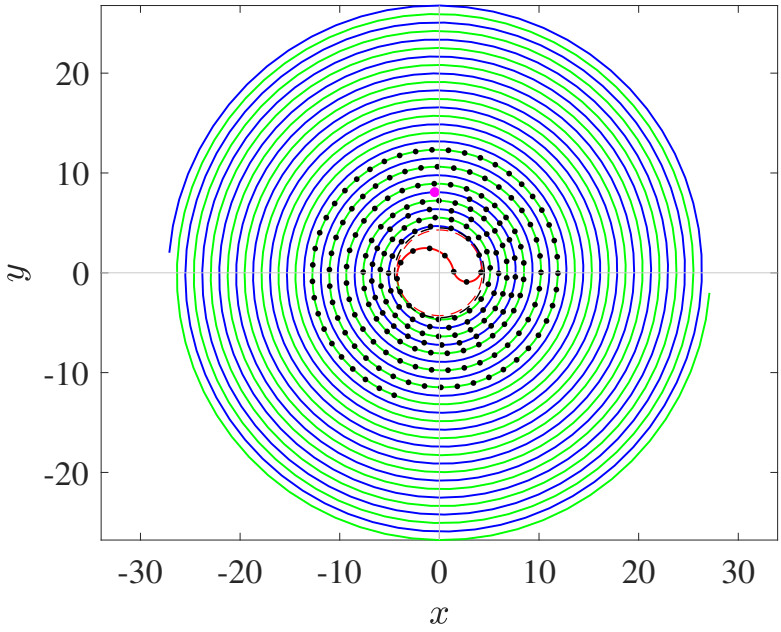


图 14: 100 s 时板凳龙各把手位置示意图

5.5 问题五

5.5.1 龙头最大速度迭代模型

问题五回归到第一问，仅需考虑每个把手的位置和速度。

易见龙头的速度范围是 $[0\ \mathrm{m \cdot s^{-1}}, 2\ \mathrm{m \cdot s^{-1}}]$，考虑将龙头的行进速度直接设定为最大值 $2\ \mathrm{m \cdot s^{-1}}$，根据问题一中的递推结论，由龙头速度可以递推得到每个把手的速度。设定合适的时间步长，模拟舞龙队的前行过程，若某一时刻发现存在一个把手的速度大于 $2\ \mathrm{m \cdot s^{-1}}$，则考虑将龙头的速度降低，使当前出现错误的把手此刻的速度正好等于 $2\ \mathrm{m \cdot s^{-1}}$。这与从一开始龙头的速度就被降低是一致的，因为行进路径不变，速度满足刚体限制（递推关系），各节点间的速度应是等比缩放的。

对于板凳龙的模拟行进长度，由于调头区域的两个圆的**曲率**明显大于阿基米德螺线上的点的曲率，且在进入盘出螺线后，根据问题一对速率的计算结果，越向螺线外圈

行进，速度越趋近龙头速度，所以最终需要模拟的前行范围，是从板凳龙进入调头区域开始，直到板凳龙完整通过调头路径曲线为止。

后续不断重复上述过程，直到所有把手在任何时刻的最大速度都不超过 2 $\mathrm{m \cdot s^{-1}}$，即可得到龙头的最大行进速度。

5.5.2 问题五模型求解结果

设定求解时间步长为 10^{-7} s，经计算，**龙头的最大行进速度为：**

$$v_{max} = \mathbf{0.409122\ m \cdot s^{-1}} \tag{26}$$

6. 模型评价与推广

6.1 模型优点

6.1.1 优点一：

本文的建模始终基于数学理性思维，利用刚体物理、平面几何、线性代数等数学理论建立模型，结合现实经验进行优化，对多个结论进行了可靠性分析或结果可视化。模型的构建基于清晰的物理意义和数学表达，便于其他研究者理解和实现，有利于学术交流和技术传播。模型建立、结果分析较为严谨、全面。

6.1.2 优点二：

本文在对问题三的求解过程中应用了模拟退火启发式算法和二分法，同时推进，互相验证，保证了计算结果的精度和准确性双高。在其他问题中将变步长搜索法与二分法结合，尽可能优化了算法和代码结构，在保证精度的同时，显著提高了计算效率，同时可以支撑更大数据量的计算。

6.2 模型缺点

6.2.1 缺点一：

模型建立过程中忽略了现实情况下舞龙过程中可能会出现的特殊情况，以及客观环境条件带来的潜在误差，距实际应用还存在一定距离。

6.2.2 缺点二：

问题三中模拟退火算法用时较长，进行 2000 次迭代耗时约 700 s。

6.2.3 缺点三：

问题四中直接规定了“实际调头起始点和终止点中心对称”，对于不中心对称的情况，仍有讨论和优化的空间。

6.3 模型推广

1. 本文工作成果实际可视作对于连续矩形链在阿基米德螺线上的运动的研究。可用与本文中类似的方法，求解实际生活中研究阿基米德螺线上运动的问题。

2. 在未来的研究中，可以考虑板凳龙运动中的非线性因素，如摩擦力和空气阻力，以提高模型的现实适应性。

参考文献

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

[2] 张晓勇，王仲君. 二分法和牛顿迭代法求解非线性方程的比较及应用. 教育教学论坛, (25):139, 2013.

[3] 云磊. 牛顿迭代法的 matlab 实现. 信息通信, (06):20,22, 2011.

[4] 高尚. 模拟退火算法中的退火策略研究. 航空计算技术, (04):20–22,26, 2002.

[5] 上官文斌，黄虎，刘德清. 多刚体系统动力学在转向系和悬架运动学分析中的应用. 汽车工程, (01):19–31, 1992.

[6] 王栋，周可璞. 基于阿基米德螺线走法的全区域覆盖路径规划. 工业控制计算机, 31(05):83–84,87, 2018.

附录 A.　支撑材料列表

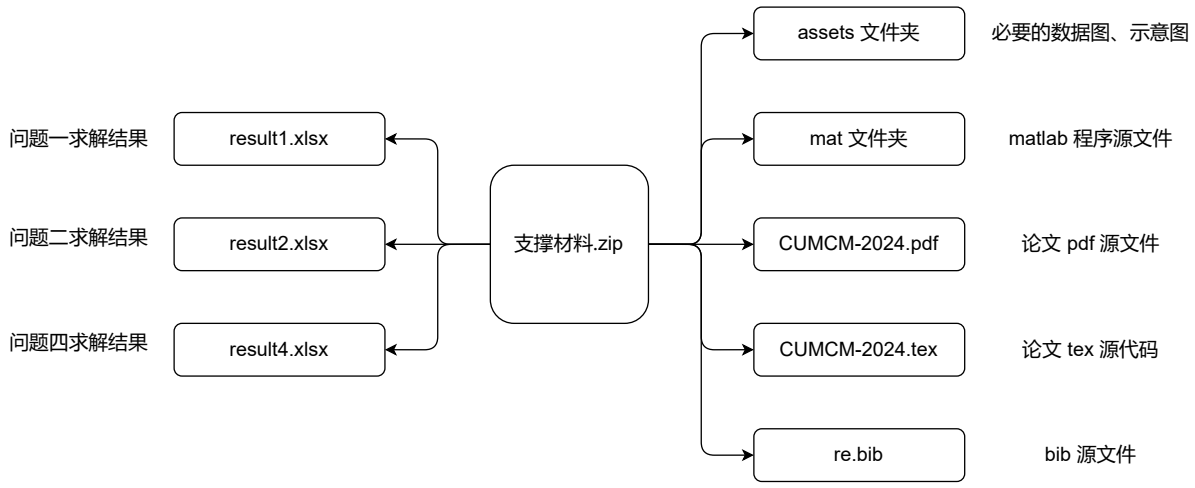


图 15: 支撑材料列表

附录 B.　matlab 源代码

B.1　问题 1 代码

```
clear, clc, close all
%% 程序主代码 %%
%%%%%%%%%%%%%%%%%

% 数据准备
    d = 1.65;              % 板上节点距离 (m)
    d_0 = 2.86;            % 龙头板上节点距离 (m)
    luoju = 0.55;          % 螺距 (m)
    v_0 = 1;               % 龙头速度 (m/s)
    b = luoju/(2*pi);      % 计算 r = a + b*theta 的参数 b

% 等距螺线可视化
    figure
    stc.line = polarplot(0:0.1:(32*pi), b*(0:0.1:(32*pi)));
    hold on
    % 设置样式
        % 坐标轴
            stc.fig = gcf;
            stc.axes = gca;
            stc.axes.FontName = "Times New Roman"; % 全局 FontName
            stc.axes.Box = 'on';
        % 图例
```

```
            stc.leg = legend(stc.axes, 'Location', 'northeast');
            stc.leg.FontSize = 15;
            stc.leg.Interpreter = "latex";
            stc.leg.String = '$r(\theta) = b\theta$';

        % 标题
            stc.axes.Title.String = '';
            stc.axes.Title.FontSize = 17;
            stc.axes.Title.FontWeight = 'bold';
        % 线的样式
            stc.line.LineWidth = 1;
            stc.line.Color = [0 0 1];   % 蓝色
        % 收尾
            hold(stc.axes,'off')
        %MyExport_pdf_docked

% 计算所有数据
    theta_array = zeros(224, 301);
    speed_array = zeros(224, 301);
    for t = 0:300
        theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*t);
        theta_array(:, t+1) = GetAllPoints(theta_0, b, d, 10^(-16));
        speed_array(:, t+1) = GetAllSpeed(theta_array(:, t+1), b, v_0);
    end

% 输出结果
    PrintResult_Q1(theta_array, speed_array, b)

% 部分结果可视化
    for t = [10, 290]
        theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*t);
        theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
        stc_drawpoints = DrawPoints(theta_all, b);
        stc_drawpoints.axes.Title.String = '';
        MyExport_pdf
        stc_rectangles = DrawPointsAndRectangles(theta_all, b);
        stc_rectangles.axes.Title.String = '';
        MyExport_pdf
    end

%% 问题一函数区 %%
%%%%%%%%%%%%%%%%%%
```

```
65
66  function dis = GetDistance(theta, theta_0, b)
67      dis = sqrt( b^2*(theta.^2 + theta_0.^2) -2*b^2.*theta.*theta_0.*cos
        (theta-theta_0) );
68  end
69
70  % 此函数已废弃，之后使用牛顿迭代法
71  function [theta, rho] = GetNextPoint( theta_0 , b, d , error_level)
72      % 第 1 层
73      theta_range = theta_0:10^(-1):(theta_0 + 10*pi);
74      for i = 2:length(theta_range)
75          if GetDistance(theta_range(i), theta_0, b) > d
76              break
77          end
78      end
79
80      % 第 2 ~ error_level 层
81      for j = 2:error_level
82          theta_range = theta_range(i-1):10^(-j):theta_range(i);
83          for i = 2:length(theta_range)
84              if GetDistance(theta_range(i), theta_0, b) > d
85                  break
86              end
87          end
88      end
89
90      % 输出结果
91      theta = 0.5*( theta_range(i-1) + theta_range(i) );
92      rho = b*theta;
93  end
94
95  function theta = GetNextPoint_newton( theta_0 , b, d , error_max_abs)
96      % 第 1 层
97      theta_range = linspace(theta_0, theta_0 + 10*pi, 300);
98      for i = 2:length(theta_range)
99          if GetDistance(theta_range(i), theta_0, b) > d
100             break
101         end
102     end
103
104     % 进行牛顿迭代
105     min = theta_range(i-1);
```

```
106     max = theta_range(i);
107
108     for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使
    得 4*pi/2^n < 10^(-20)
109         error_max = max - min;
110         temp = 0.5 * (max + min);
111         temp_dis = GetDistance(temp, theta_0, b);
112         if temp_dis < d
113             min = temp;
114         elseif temp_dis > d
115             max = temp;
116         elseif temp_dis == d
117             theta = temp;
118             return
119         end
120         if error_max <= error_max_abs
121             break
122         end
123     end
124
125     % 输出结果
126     theta = 0.5*(min + max);
127 end
128
129 function theta_all = GetAllPoints( theta_0 , b, d , error_max_abs)
130     theta_all = zeros(224, 1);
131     theta_all(1) = theta_0;
132     % 龙头板凳长不同
133     theta_all(2) = GetNextPoint_newton(theta_all(1), b, 2.86,
    error_max_abs);
134     for i = 3:224
135         theta_all(i) = GetNextPoint_newton(theta_all(i-1), b, d,
    error_max_abs);
136         %polarscatter(theta, r, 100, 'r.');
137     end
138 end
139
140
141 function stc = DrawPoints(theta_all, b)
142
143     X = 0:0.1:(32*pi);
144     Rho = b*X;
```

```
145
146     % 作图
147     figure
148     stc.line = polarplot(X, Rho);
149     hold on
150     stc.point_0 = polarscatter(theta_all(1), b*theta_all(1), 200, 'r.')
        ;
151     stc.points_rest = polarscatter(theta_all(2:end), b*theta_all(2:end)
        , 70, 'b.');
152
153 % 设置样式
154     % 坐标轴
155         stc.fig = gcf;
156         stc.axes = gca;
157         stc.axes.FontName = "Times New Roman"; % 全局 FontName
158         stc.axes.Box = 'on';
159
160     % 标题
161         stc.axes.Title.String = 'Figure: Draw Points';
162         stc.axes.Title.FontSize = 17;
163         stc.axes.Title.FontWeight = 'bold';
164
165     % 线的样式
166         stc.line.LineWidth = 1;
167         stc.line.Color = [0 1 0];   % 绿色
168
169     % 收尾
170         hold(stc.axes,'off')
171 end
172
173
174 function Rectangle_Points = GetRectangles(theta_all, b)
175     Coordinates = b*theta_all.*[cos(theta_all), sin(theta_all)];
176     Vec_X = diff(Coordinates);
177     Vec_X = Vec_X ./ sqrt(sum(Vec_X.^2, 2));      % 单位化
178     Vec_N = [ -Vec_X(: ,2), Vec_X(: ,1) ];  % 法向量
179
180
181     % 计算矩形坐标
182     Rectangle_P1 = Coordinates(1:223, :) - Vec_X*0.275 + Vec_N*0.15;
        % 注意是 1:223
183     Rectangle_P2 = Coordinates(1:223, :) - Vec_X*0.275 - Vec_N*0.15;
```

```
    Rectangle_P3 = Coordinates(2:224, :) + Vec_X*0.275 - Vec_N*0.15;
    % 注意是 2:224
    Rectangle_P4 = Coordinates(2:224, :) + Vec_X*0.275 + Vec_N*0.15;

    Rectangle_Points = zeros(4, 223, 2);
    Rectangle_Points(1, :, :) = Rectangle_P1;
    Rectangle_Points(2, :, :) = Rectangle_P2;
    Rectangle_Points(3, :, :) = Rectangle_P3;
    Rectangle_Points(4, :, :) = Rectangle_P4;
end


function stc = DrawPointsAndRectangles(theta_all, b)
% 作线和点
    x = 0:0.1:(32*pi);
    X = x';

    % 转为直角坐标
        coor_line = [
            b*X.*cos(X), b*X.*sin(X)
        ];
        coor_all = [
            b*theta_all.*cos(theta_all), b*theta_all.*sin(theta_all)
        ];

    % 作图
        figure
        stc.line = plot(coor_line(:, 1), coor_line(:, 2));
        hold on
        stc.point_0 = scatter(coor_all(1, 1), coor_all(1, 2), 200, 'r.');
        stc.points_rest = scatter(coor_all(2:end, 1), coor_all(2:end, 2), 70, 'b.');

    % 设置样式
        % 坐标轴
            stc.fig = gcf;
            axis equal
            stc.axes = gca;
            stc.axes.FontName = "Times New Roman"; % 全局 FontName
            stc.axes.Box = 'on';
            stc.axes.FontSize = 11;
```

```
223             xline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
224             yline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
225             stc.label.x = xlabel(stc.axes, '$x\ (\mathrm{m})$', '
    Interpreter', 'latex', 'FontSize', 15);
226             stc.label.y = ylabel(stc.axes, '$y\ (\mathrm{m})$', '
    Interpreter', 'latex', 'FontSize', 15);
227
228
229         % 标题
230             stc.axes.Title.String = 'Figure: Draw Rectangles';
231             stc.axes.Title.FontSize = 17;
232             stc.axes.Title.FontWeight = 'bold';
233
234         % 线的样式
235             stc.line.LineWidth = 1;
236             stc.line.Color = [0 1 0];   % 绿色
237
238 % 作方框
239     Rectangle_Points = GetRectangles(theta_all, b);
240     Rectangle_Points(5, :, :) = Rectangle_Points(1, :, :);    % plot 围
    成闭合曲线
241     hold on
242     for i = 1:223
243         plot(Rectangle_Points(:, i, 1), Rectangle_Points(:, i, 2), '
    LineWidth', 0.3, 'Color', [1 0 1]);
244         %scatter(Rectangle_Points(1, i, 1), Rectangle_Points(1, i, 2),
    45, '.red');
245         %scatter(Rectangle_Points(2, i, 1), Rectangle_Points(2, i, 2),
    45, '.black');
246         %scatter(Rectangle_Points(3, i, 1), Rectangle_Points(3, i, 2),
    45, '.black');
247     end
248
249
250 % 收尾
251     hold off
252 end
253
254
255 function Speed_all = GetAllSpeed(theta_all, b, v_0)
256     Speed_all = zeros(224 ,1);
257     Speed_all(1) = v_0;
```

```
258
259     % 转为直角坐标
260     Coordinates = b*theta_all.*[cos(theta_all), sin(theta_all)];
261     Vec_X = diff(Coordinates);
262     % 下面注意要有负号
263     Vec_Tao = -[cos(theta_all) - theta_all.*sin(theta_all),  sin(
        theta_all) + theta_all.*cos(theta_all)];
264     for i = 1:223
265         Speed_all(i+1) = Speed_all(i) * ( norm(Vec_Tao(i+1, :))*abs(sum
        (Vec_Tao(i, :).*Vec_X(i, :)))  ) / ( norm(Vec_Tao(i, :))*abs(sum(
        Vec_Tao(i+1, :).*Vec_X(i, :))) );
266     end
267 end
268
269
270 % 此函数已废弃，因为 .xlsx 格式不同
271 function printLocation(theta_all, Speed_all, b, column_initial)
272     %logic_matrix = (theta_all - 16*2*pi) > 0;
273     theta_all((theta_all - 16*2*pi) > 0) = nan;    % 将不合法 (未进圈)
        的数据设为 nan
274     Speed_all((theta_all - 16*2*pi) > 0) = nan;
275     % 计算直角坐标位置
276     Coordinates = b*theta_all.*[cos(theta_all), sin(theta_all)];
277     rowHeaders = arrayfun(@(i) sprintf('第%d节点', i), 1:length(
        theta_all), 'UniformOutput', false);
278
279     Output = array2table([Coordinates, Speed_all], "RowNames",
        rowHeaders, "VariableNames", {'位置 x', '位置 y', '速率 v'});
280     writetable(Output, 'Q1_Result.xlsx', 'WriteRowNames', true, 'Range'
        , column_initial);
281 end
282
283 function PrintResult_Q1(theta, Speed, b)
284     % theta: 224*301 矩阵
285     % Speed: 224*301 矩阵
286     % b: 用于转直角坐标
287     % 注：未进圈的数据本应是 nan，为方便考虑
288     %     这里未做处理，theta_all((theta_all - 16*2*pi) > 0) = nan 即可
        筛选
289
290     t_length = size(theta, 2);
291
```

```
    % 计算直角坐标位置
    Coordinates = zeros(2, 224, t_length);
    Coordinates(1, :, :) = b*theta.*cos(theta);
    Coordinates(2, :, :) = b*theta.*sin(theta);
    Location = zeros(2*224, t_length);
    for i = 1:2*224
        if mod(i, 2) == 1   % i 奇数, 对应 x
            Location(i, :) = Coordinates(1, ceil(i/2), :);
        else                % i 偶数, 对应 y
            Location(i, :) = Coordinates(2, i/2, :);
        end
    end
    % 遍历矩阵, 将每个元素格式化为保留 6 位小数的字符串
    for i = 1:size(Location, 1)
        for j = 1:size(Location, 2)
            Location_str{i, j} = num2str(Location(i, j), '%.6f');
        end
    end
    for i = 1:size(Speed, 1)
        for j = 1:size(Speed, 2)
            Speed_str{i, j} = num2str(Speed(i, j), '%.6f');
        end
    end

    writecell(Location_str, 'Q1_Result.xlsx', 'Sheet', 'Sheet1'); % 输出位置
    writecell(Speed_str, 'Q1_Result.xlsx', 'Sheet', 'Sheet2'); % 输出速度
end

function theta = GetTheta(b, v, t)
    % 弧长公式
    S = @(theta) b/2 * (  theta.*sqrt(1+theta.^2) + log(theta + sqrt(1+theta.^2))  );
    S_real = @(theta) S(16*2*pi) - S(theta);

    % 第 1 层
    theta_range = 16*2*pi:(-10^(-1)):5*pi;
    for i = 2:length(theta_range)
        if GetDistance(theta_range(i), theta_0, b) > d
            break
        end
```

```
    end

    % 进行牛顿迭代
    min = theta_range(i-1);
    max = theta_range(i);

    for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使得 4*pi/2^n < 10^(-20)
        error_max = max - min;
        temp = 0.5 * (max + min);
        temp_dis = GetDistance(temp, theta_0, b);
        if temp_dis < d
            min = temp;
        elseif temp_dis > d
            max = temp;
        elseif temp_dis == d
            theta = temp;
            return
        end
        if error_max <= error_max_abs
            break
        end
    end
    % 输入参数
    % b = 55 / (2 * pi);     % 螺线递增参数 (b)
    % theta0 = 16 * 2 * pi; % 初始点的极坐标角度 (θ_0)
    % L = 500;               % 沿螺线走的距离 (L)
end

function theta_final = func_find_start(b, theta0, L)
    % 初始化参数
    % b = 55 / (2 * pi);     % 螺线递增参数 (b)
    % theta0 = 16 * 2 * pi; % 初始点的极坐标角度 (θ_0)
    % L = 500;               % 沿螺线走的距离 (L)

    % 定义螺旋线方程函数
    r = @(theta) b * (theta0-theta);        % 极径方程：r(θ) = b(θ_0-θ)
    ds = @(theta) sqrt(b^2 + r(theta).^2); % 曲线长度微元
    r_real = @(theta) b * theta;        % 极径方程：r(θ) = b(θ_0-θ)
    % 使用Numerical Integration计算θ值
    theta_final = fzero(@(theta) integral(ds, 0, theta) - L, theta0 + 2*pi);
```

```
end
```

B.2 问题 2 代码

```
clear, clc, close all
%% 程序主代码 %%
%%%%%%%%%%%%%%%%%%%%
% 数据准备
    d = 1.65;           % 板上节点距离 (m)
    d_0 = 2.86;         % 龙头板上节点距离 (m)
    luoju = 0.55;       % 螺距 (m)
    v_0 = 1;            % 龙头速度 (m/s)
    b = luoju/(2*pi);   % 计算 r = a + b*theta 的参数 b


% 搜索 crush 时间
    t_range = [350 450];
    t_step1 = 0.5;
    t_step2 = 0.1;
    error_max_abs = 10^(-16);
    t_crushed = GetCrushedTime(b, d, v_0, t_range, t_step1, t_step2,
    error_max_abs);

% 输出结果
    theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*(t_crushed-
    error_max_abs));
    theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
    speed_all = GetAllSpeed(theta_all, b, v_0);
    PrintResult_Q2(theta_all, speed_all, b);
    disp(['t_crushing = ', num2str(t_crushed, '%.12f')])
% 可靠性检验 (结果可视化)
    theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*(t_crushed-
    t_step2));
    theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
    stc = DrawPointsAndRectangles(theta_all, b);
    stc.axes.Title.String = '';
    %MyExport_pdf_docked

    theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*(t_crushed));
    theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
    stc = DrawPointsAndRectangles(theta_all, b);
    stc.axes.Title.String = '';
```

```
    %MyExport_pdf_docked


%% 问题二函数区 %%
%%%%%%%%%%%%%%%%%%

function t_crushed = GetCrushedTime(b, d, v_0, t_range, t_step1,
    t_step2, error_max_abs)
    % 第一层粗搜
    for t = t_range(1):t_step1:t_range(2)
        theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*t);
        theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
        logic = IsCrushed(theta_all, b);
        if logic == true
            break
        end
    end

    % 第二层细搜
    t_range = [t - 3*t_step1, t];
    for t = t_range(1):t_step2:t_range(2)
        theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*t);
        theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
        logic = IsCrushed(theta_all, b);
        if logic == true
            break
        end
    end

    % 进行牛顿迭代
    min = t - t_step2;  max = t;
    for i = 1:67     % ceil( (log(1)+20*log(10))/log(2) ) = 67, 这使得
    1/2^n < 10^(-20)
        error_max = max - min;
        temp_t = 0.5 * (max + min);
        theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*temp_t);
        theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
        logic = IsCrushed(theta_all, b);
        if logic ~= true
            min = temp_t;
        elseif logic == true
            max = temp_t;
```

```
76          end
77          if error_max <= error_max_abs
78              break
79          end
80      end
81
82      % 输出结果
83      t_crushed = 0.5*(min+max);
84  end
85
86  function logic = IsCrushed(theta_all, b)
87      % 数据准备
88      d_special = sqrt(3.135^2 + 0.15^2) + sqrt(0.275^2 + 0.15^2);  % 龙
    头 3 号点最大判断间距
89      d_common = sqrt(1.925^2 + 0.15^2) + sqrt(0.275^2 + 0.15^2);  % 2, 3
     号点最大判断间距
90      Rectangle_Points = GetRectangles(theta_all, b);
91
92      % 只需判断第 2, 3 号点是否 legal
93
94      % 判断龙头 (第一个板凳)
95          point_1_2(1, 1:2) = Rectangle_Points(2, 1, :);
96          point_1_3(1, 1:2) = Rectangle_Points(3, 1, :);
97
98          % 挑选可能的节点
99          distance = sqrt( b^2*(theta_all(1).^2 + theta_all.^2) -2*b^2.*
    theta_all(1).*theta_all.*cos(theta_all(1)-theta_all) );
100         Logic = (distance < d_special) ;
101         Logic(1:2) = 0; % 忽略第 1(本身),2(之后)
102         for i = find(Logic)'    % 这里必须转置使得右值为行向量
103             % 新的坐标原点
104             new_Origin(1, 1:2) = Rectangle_Points(1, i, :);
105
106             % 检查 2 号点
107             new_cor = (point_1_2 - new_Origin ) / [
108                 Rectangle_Points(2, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(2, i, 2) - Rectangle_Points(1, i, 2);
109                 Rectangle_Points(4, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(4, i, 2) - Rectangle_Points(1, i, 2);
110                 ];   % 坐标系转换
111             %disp(['龙头2号点 new_cor:', num2str(new_cor)])
112             if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&new_cor
```

```
(2)<1)
113                 logic = true;
114                 return
115             end
116             % 检查 3 号点
117             new_cor = (point_1_3 - new_Origin ) / [
118                 Rectangle_Points(2, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(2, i, 2) - Rectangle_Points(1, i, 2);
119                 Rectangle_Points(4, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(4, i, 2) - Rectangle_Points(1, i, 2);
120                 ];   % 坐标系转换
121             %disp(['龙头3号点 new_cor:', num2str(new_cor)])
122             if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&new_cor
    (2)<1)
123                 logic = true;
124                 return
125             end
126         end
127
128
129     % 判断第二个板凳
130         point_2_2(1, 1:2) = Rectangle_Points(2, 2, :);
131         point_2_3(1, 1:2) = Rectangle_Points(3, 2, :);
132
133         % 挑选可能的节点
134         distance = sqrt( b^2*(theta_all(2).^2 + theta_all.^2) -2*b^2.*
    theta_all(2).*theta_all.*cos(theta_all(2)-theta_all) );
135         Logic = (distance < d_common) ;
136         Logic(1:3) = 0; % 忽略第 1(龙头),2(本身),3(其后) 节点
137         for i = find(Logic)'    % 这里必须转置使得右值为行向量
138             % 新的坐标原点
139             new_Origin(1, 1:2) = Rectangle_Points(1, i, :);
140
141             % 检查 2 号点
142             new_cor = (point_2_2 - new_Origin ) / [
143                 Rectangle_Points(2, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(2, i, 2) - Rectangle_Points(1, i, 2);
144                 Rectangle_Points(4, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(4, i, 2) - Rectangle_Points(1, i, 2);
145                 ];   % 坐标系转换
146             %disp(['板凳2号点 new_cor:', num2str(new_cor)])
147             if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&new_cor
```

```
    (2)<1)
148                 logic = true;
149                 return
150             end
151             % 检查 3 号点
152             new_cor = (point_2_3 - new_Origin) / [
153                 Rectangle_Points(2, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(2, i, 2) - Rectangle_Points(1, i, 2);
154                 Rectangle_Points(4, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(4, i, 2) - Rectangle_Points(1, i, 2);
155                 ];  % 坐标系转换
156             %disp(['板凳3号点 new_cor:', num2str(new_cor)])
157             if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&new_cor
    (2)<1)
158                 logic = true;
159                 return
160             end
161         end
162
163     % 检查通过
164     logic = false;
165     return
166 end
167
168
169 function PrintResult_Q2(theta, Speed, b)
170     % theta: 224*1 矩阵
171     % Speed: 224*1 矩阵
172     % b: 用于转直角坐标
173     % 注：未进圈的数据本应是 nan，为方便考虑
174     %     这里未做处理，theta_all((theta_all - 16*2*pi) > 0) = nan 即可
    筛选
175
176     % 计算直角坐标位置
177     Coordinates = b*theta.*[cos(theta), sin(theta)];
178
179     % 遍历矩阵，将每个元素格式化为保留 6 位小数的字符串
180     Output = num2cell(zeros(224,3));
181     for i = 1:size(Coordinates, 1)
182             Output{i, 1} = num2str(Coordinates(i, 1), '%.6f');
183             Output{i, 2} = num2str(Coordinates(i, 2), '%.6f');
184     end
```

```
    for i = 1:size(Speed, 1)
        Output{i, 3} = num2str(Speed(i), '%.6f');
    end

    writecell(Output, 'Q2_Result.xlsx', 'Sheet', 'Sheet1'); % 输出结果
end


%% 问题一函数区 %%
%%%%%%%%%%%%%%%%%%

function dis = GetDistance(theta, theta_0, b)
    dis = sqrt( b^2*(theta.^2 + theta_0.^2) -2*b^2.*theta.*theta_0.*cos
    (theta-theta_0) );
end

% 此函数已废弃，之后使用牛顿迭代法
function [theta, rho] = GetNextPoint( theta_0 , b, d , error_level)
    % 第 1 层
    theta_range = theta_0:10^(-1):(theta_0 + 10*pi);
    for i = 2:length(theta_range)
        if GetDistance(theta_range(i), theta_0, b) > d
            break
        end
    end

    % 第 2 ~ error_level 层
    for j = 2:error_level
        theta_range = theta_range(i-1):10^(-j):theta_range(i);
        for i = 2:length(theta_range)
            if GetDistance(theta_range(i), theta_0, b) > d
                break
            end
        end
    end

    % 输出结果
    theta = 0.5*( theta_range(i-1) + theta_range(i) );
    rho = b*theta;
end
```

```
function theta = GetNextPoint_newton( theta_0 , b, d , error_max_abs)
    % 第 1 层
    theta_range = linspace(theta_0, theta_0 + 10*pi, 300);
    for i = 2:length(theta_range)
        if GetDistance(theta_range(i), theta_0, b) > d
            break
        end
    end

    % 进行牛顿迭代
    min = theta_range(i-1);
    max = theta_range(i);

    for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使得 4*pi/2^n < 10^(-20)
        error_max = max - min;
        temp = 0.5 * (max + min);
        temp_dis = GetDistance(temp, theta_0, b);
        if temp_dis < d
            min = temp;
        elseif temp_dis > d
            max = temp;
        elseif temp_dis == d
            theta = temp;
            return
        end
        if error_max <= error_max_abs
            break
        end
    end

    % 输出结果
    theta = 0.5*(min + max);
end

function theta_all = GetAllPoints( theta_0 , b, d , error_max_abs)
    theta_all = zeros(224, 1);
    theta_all(1) = theta_0;
    % 龙头板凳长不同
    theta_all(2) = GetNextPoint_newton(theta_all(1), b, 2.86, error_max_abs);
    for i = 3:224
```

```
266         theta_all(i) = GetNextPoint_newton(theta_all(i-1), b, d,
    error_max_abs);
267         %polarscatter(theta, r, 100, 'r.');
268     end
269 end
270
271
272 function stc = DrawPoints(theta_all, b)
273
274     X = 0:0.1:(32*pi);
275     Rho = b*X;
276
277     % 作图
278     figure
279     stc.line = polarplot(X, Rho);
280     hold on
281     stc.point_0 = polarscatter(theta_all(1), b*theta_all(1), 200, 'r.')
    ;
282     stc.points_rest = polarscatter(theta_all(2:end), b*theta_all(2:end)
    , 70, 'b.');
283
284 % 设置样式
285     % 坐标轴
286         stc.fig = gcf;
287         stc.axes = gca;
288         stc.axes.FontName = "Times New Roman"; % 全局 FontName
289         stc.axes.Box = 'on';
290
291     % 标题
292         stc.axes.Title.String = 'Figure: Draw Points';
293         stc.axes.Title.FontSize = 17;
294         stc.axes.Title.FontWeight = 'bold';
295
296     % 线的样式
297         stc.line.LineWidth = 1;
298         stc.line.Color = [0 1 0];   % 绿色
299
300     % 收尾
301         hold(stc.axes,'off')
302 end
303
304
```

```
function Rectangle_Points = GetRectangles(theta_all, b)
    Coordinates = b*theta_all.*[cos(theta_all), sin(theta_all)];
    Vec_X = diff(Coordinates);
    Vec_X = Vec_X ./ sqrt(sum(Vec_X.^2, 2));      % 单位化
    Vec_N = [ -Vec_X(:,2), Vec_X(:,1) ];  % 法向量


    % 计算矩形坐标
    Rectangle_P1 = Coordinates(1:223, :) - Vec_X*0.275 + Vec_N*0.15;
    % 注意是 1:223
    Rectangle_P2 = Coordinates(1:223, :) - Vec_X*0.275 - Vec_N*0.15;
    Rectangle_P3 = Coordinates(2:224, :) + Vec_X*0.275 - Vec_N*0.15;
    % 注意是 2:224
    Rectangle_P4 = Coordinates(2:224, :) + Vec_X*0.275 + Vec_N*0.15;

    Rectangle_Points = zeros(4, 223, 2);
    Rectangle_Points(1, :, :) = Rectangle_P1;
    Rectangle_Points(2, :, :) = Rectangle_P2;
    Rectangle_Points(3, :, :) = Rectangle_P3;
    Rectangle_Points(4, :, :) = Rectangle_P4;
end


function stc = DrawPointsAndRectangles(theta_all, b)
% 作线和点
    x = 0:0.1:(32*pi);
    X = x';

    % 转为直角坐标
        coor_line = [
            b*X.*cos(X), b*X.*sin(X)
        ];
        coor_all = [
            b*theta_all.*cos(theta_all), b*theta_all.*sin(theta_all)
        ];

    % 作图
        figure('Color', [1 1 1])
        stc.line = plot(coor_line(:, 1), coor_line(:, 2));
        hold on
        stc.point_0 = scatter(coor_all(1, 1), coor_all(1, 2), 200, 'r.'
    );
```

```
344         stc.points_rest = scatter(coor_all(2:end, 1), coor_all(2:end,
    2), 70, 'b.');
345
346     % 设置样式
347         % 坐标轴
348             stc.fig = gcf;
349             axis equal
350             stc.axes = gca;
351             stc.axes.FontName = "Times New Roman"; % 全局 FontName
352             stc.axes.Box = 'on';
353             stc.axes.FontSize = 14;
354             xline(0, 'LineWidth', 0.3, 'Color', [0.3, 0.3, 0.3]);
355             yline(0, 'LineWidth', 0.3, 'Color', [0.3, 0.3, 0.3]);
356             stc.label.x = xlabel(stc.axes, '$x$', 'Interpreter', 'latex
    ', 'FontSize', 15);
357             stc.label.y = ylabel(stc.axes, '$y$', 'Interpreter', 'latex
    ', 'FontSize', 15);
358
359
360         % 标题
361             stc.axes.Title.String = 'Figure: Draw Rectangles';
362             stc.axes.Title.FontSize = 17;
363             stc.axes.Title.FontWeight = 'bold';
364
365         % 线的样式
366             stc.line.LineWidth = 1;
367             stc.line.Color = [0 1 0];   % 绿色
368
369 % 作方框
370     Rectangle_Points = GetRectangles(theta_all, b);
371     Rectangle_Points(5, :, :) = Rectangle_Points(1, :, :);    % plot 围
    成闭合曲线
372     hold on
373     for i = 1:223
374         plot(Rectangle_Points(:, i, 1), Rectangle_Points(:, i, 2), '
    LineWidth', 0.3, 'Color', [1 0 1]);
375     end
376
377 % 收尾
378     hold off
379 end
380
```

```
381
382 function Speed_all = GetAllSpeed(theta_all, b, v_0)
383     Speed_all = zeros(224 ,1);
384     Speed_all(1) = v_0;
385
386     % 转为直角坐标
387     Coordinates = b*theta_all.*[cos(theta_all), sin(theta_all)];
388     Vec_X = diff(Coordinates);
389     % 下面注意要有负号
390     Vec_Tao = -[cos(theta_all) - theta_all.*sin(theta_all),  sin(
    theta_all) + theta_all.*cos(theta_all)];
391     for i = 1:223
392         Speed_all(i+1) = Speed_all(i) * ( norm(Vec_Tao(i+1, :))*abs(sum
    (Vec_Tao(i, :).*Vec_X(i, :)))  ) / ( norm(Vec_Tao(i, :))*abs(sum(
    Vec_Tao(i+1, :).*Vec_X(i, :))) );
393     end
394 end
395
396
397 % 此函数已废弃，因为 .xlsx 格式不同
398 function printLocation(theta_all, Speed_all, b, column_initial)
399     %logic_matrix = (theta_all - 16*2*pi) > 0;
400     theta_all((theta_all - 16*2*pi) > 0) = nan;     % 将不合法（未进圈）
    的数据设为 nan
401     Speed_all((theta_all - 16*2*pi) > 0) = nan;
402     % 计算直角坐标位置
403     Coordinates = b*theta_all.*[cos(theta_all), sin(theta_all)];
404     rowHeaders = arrayfun(@(i) sprintf('第%d节点', i), 1:length(
    theta_all), 'UniformOutput', false);
405
406     Output = array2table([Coordinates, Speed_all], "RowNames",
    rowHeaders, "VariableNames", {'位置 x', '位置 y', '速率 v'});
407     writetable(Output, 'Q1_Result.xlsx', 'WriteRowNames', true, 'Range'
    , column_initial);
408 end
409
410 function PrintResult_Q1(theta, Speed, b)
411     % theta: 224*301 矩阵
412     % Speed: 224*301 矩阵
413     % b: 用于转直角坐标
414     % 注：未进圈的数据本应是 nan，为方便考虑
415     %     这里未做处理，theta_all((theta_all - 16*2*pi) > 0) = nan 即可
```

```
    筛选
416
417     t_length = size(theta, 2);
418
419     % 计算直角坐标位置
420     Coordinates = zeros(2, 224, t_length);
421     Coordinates(1, :, :) = b*theta.*cos(theta);
422     Coordinates(2, :, :) = b*theta.*sin(theta);
423     Location = zeros(2*224, t_length);
424     for i = 1:2*224
425         if mod(i, 2) == 1   % i 奇数, 对应 x
426             Location(i, :) = Coordinates(1, ceil(i/2), :);
427         else                % i 偶数, 对应 y
428             Location(i, :) = Coordinates(2, i/2, :);
429         end
430     end
431     % 遍历矩阵, 将每个元素格式化为保留 6 位小数的字符串
432     for i = 1:size(Location, 1)
433         for j = 1:size(Location, 2)
434             Location_str{i, j} = num2str(Location(i, j), '%.6f');
435         end
436     end
437     for i = 1:size(Speed, 1)
438         for j = 1:size(Speed, 2)
439             Speed_str{i, j} = num2str(Speed(i, j), '%.6f');
440         end
441     end
442
443     writecell(Location_str, 'Q1_Result.xlsx', 'Sheet', 'Sheet1'); % 输
    出位置
444     writecell(Speed_str, 'Q1_Result.xlsx', 'Sheet', 'Sheet2'); % 输出速
    度
445 end
446
447 function theta = GetTheta(b, v, t)
448     % 弧长公式
449     S = @(theta) b/2 * (  theta.*sqrt(1+theta.^2) + log(theta + sqrt(1+
    theta.^2))  );
450     S_real = @(theta) S(16*2*pi) - S(theta);
451
452     % 第 1 层
453     theta_range = 16*2*pi:(-10^(-1)):5*pi;
```

```
    for i = 2:length(theta_range)
        if GetDistance(theta_range(i), theta_0, b) > d
            break
        end
    end

    % 进行牛顿迭代
    min = theta_range(i-1);
    max = theta_range(i);

    for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使得 4*pi/2^n < 10^(-20)
        error_max = max - min;
        temp = 0.5 * (max + min);
        temp_dis = GetDistance(temp, theta_0, b);
        if temp_dis < d
            min = temp;
        elseif temp_dis > d
            max = temp;
        elseif temp_dis == d
            theta = temp;
            return
        end
        if error_max <= error_max_abs
            break
        end
    end
    % 输入参数
    % b = 55 / (2 * pi);     % 螺线递增参数 (b)
    % theta0 = 16 * 2 * pi; % 初始点的极坐标角度 (θ_0)
    % L = 500;              % 沿螺线走的距离 (L)
end

function theta_final = func_find_start(b, theta0, L)
    % 初始化参数
    % b = 55 / (2 * pi);     % 螺线递增参数 (b)
    % theta0 = 16 * 2 * pi; % 初始点的极坐标角度 (θ_0)
    % L = 500;              % 沿螺线走的距离 (L)

    % 定义螺旋线方程函数
    r = @(theta) b * (theta0-theta);        % 极径方程: r(θ) = b(θ_0-θ)
    ds = @(theta) sqrt(b^2 + r(theta).^2); % 曲线长度微元
```

```
    r_real = @(theta) b * theta;        % 极径方程：r(θ) = b(θ_0-θ)
    % 使用Numerical Integration计算θ值
    theta_final = fzero(@(theta) integral(ds, 0, theta) - L, theta0 + 2*pi);

end
```

B.3 问题 3 代码

B.3.1 主程序代码

```
clc, clear, close all
%% 主程序代码区 %%
%%%%%%%%%%%%%%%%%%

% 模拟退火获取最小螺距

    % 数据准备
        d = 1.65;           % 板上节点距离 (m)
        d_0 = 2.86;         % 龙头板上节点距离 (m)
        v_0 = 1;            % 龙头速度 (m/s)
        t_range = [0, 350];
        t_step1 = 0.5;
        t_step2 = 0.1;
        t_error_max_abs = 10^(-12);

    % 构建退火结构体
        Struct_SA.Var.num = 1; % 一个参数
        Struct_SA.Var.range = [0.44 0.46 0.455];    % 螺距参数范围和初始值
        Struct_SA.Annealing.T_0 = 10;          % 初始温度
        Struct_SA.Annealing.alpha = 0.97;      % 降温系数
        Struct_SA.Annealing.T_end = 1;         % 停止温度
        Struct_SA.Annealing.mkvlength = 4;     % 马尔科夫链长度

% 进行模拟退火
    Struct_SA = MySimulatedAnnealing(Struct_SA, @Objective_Annealing, d, v_0, t_range, t_step1, t_step2, t_error_max_abs);
    vpa(Struct_SA.X_best)   % 显示结果


% 可靠性检验
```

```
    % 图 1
    luoju_range = [0.35 0.5];
    luoju_array = linspace(luoju_range(1), luoju_range(2), 50);
    dist_array = zeros(size(luoju_array));
    for i = 1:length(luoju_array)
        dist_array(i) = GetCurushedDistance(luoju_array(i), d, v_0,
    t_range, t_step1, t_step2, t_error_max_abs);
    end
    stc = MyPlot(luoju_array, dist_array);
    yline(4.5);
    stc.label.x.String = 'screw pitch (m)';
    stc.label.y.String = 'distance (m)';
    stc.axes.Title.String = '';
    stc.leg.String = ['crushing distance'; ''; ''; ''; ''; ''];

    % 图 2
    luoju_range = [0.44 0.46];
    luoju_array = linspace(luoju_range(1), luoju_range(2), 50);
    dist_array = zeros(size(luoju_array));
    for i = 1:length(luoju_array)
        dist_array(i) = GetCurushedDistance(luoju_array(i), d, v_0,
    t_range, t_step1, t_step2, t_error_max_abs);
    end
    stc = MyPlot(luoju_array, dist_array);
    yline(4.5);
    stc.label.x.String = 'screw pitch (m)';
    stc.label.y.String = 'distance (m)';
    stc.axes.Title.String = '';
    stc.leg.String = ['crushing distance'; ''; ''; ''; ''; ''];

%% 问题三函数区 %%
%%%%%%%%%%%%%%%%%%

function obj = Objective_Annealing(luoju, d, v_0, t_range, t_step1,
    t_step2, t_error_max_abs)
    dist = GetCurushedDistance(luoju, d, v_0, t_range, t_step1, t_step2
    , t_error_max_abs);
    if dist < 4.5   % 能进入掉头区
        obj = luoju;
    else    % 不能进入调头区
        obj = 0.455;
    end
```

```
68  end
69
70  function dist = GetCurushedDistance(luoju, d, v_0, t_range, t_step1,
        t_step2, t_error_max_abs)
71  % 函数：获取最小螺距
72      b = luoju/(2*pi);   % 计算 r = a + b*theta 的参数 b
73      t_crushed = GetCrushedTime(b, d, v_0, t_range, t_step1, t_step2,
        t_error_max_abs);
74      theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*t_crushed);
75      coordinates = b*theta_0.*[cos(theta_0), sin(theta_0)];
76      dist = sqrt(sum(coordinates.^2));
77      %disp(['luoju = ', num2str(luoju), ', t_crushed = ', num2str(
        t_crushed), ', theta_0 = ', num2str(theta_0) ', distance = ',
        num2str(dist) ])
78      %DrawPointsAndRectangles(GetAllPoints(theta_0 , b, d , 10^(-12)), b
        );
79  end
80
81  %% 问题二函数区 %%
82  %%%%%%%%%%%%%%%%%%
83
84  function t_crushed = GetCrushedTime(b, d, v_0, t_range, t_step1,
        t_step2, error_max_abs)
85      % 第一层粗搜
86      for t = t_range(1):t_step1:t_range(2)
87          theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*t);
88          theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
89          logic = IsCrushed(theta_all, b);
90          if logic == true
91              break
92          end
93      end
94
95      % 第二层细搜
96      t_range = [t - 3*t_step1, t];
97      for t = t_range(1):t_step2:t_range(2)
98          theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*t);
99          theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
100         logic = IsCrushed(theta_all, b);
101         if logic == true
102             break
103         end
```

```
104     end
105
106     % 牛顿迭代法获取高精度解
107     min = t - t_step2;  max = t;
108     for i = 1:67      % ceil( (log(1)+20*log(10))/log(2) ) = 67, 这使得
    1/2^n < 10^(-20)
109         error_max = max - min;
110         temp_t = 0.5 * (max + min);
111         theta_0 = 16*2*pi - func_find_start(b, 16*2*pi, v_0*temp_t);
112         theta_all = GetAllPoints(theta_0, b, d, 10^(-16));
113         logic = IsCrushed(theta_all, b);
114         if logic ~= true
115             min = temp_t;
116         elseif logic == true
117             max = temp_t;
118         end
119         if error_max <= error_max_abs
120             break
121         end
122     end
123
124     % 输出结果
125     t_crushed = 0.5*(min+max);
126 end
127
128 function logic = IsCrushed(theta_all, b)
129     % 数据准备
130     d_special = sqrt(3.135^2 + 0.15^2) + sqrt(0.275^2 + 0.15^2);  % 龙
    头 3 号点最大判断间距
131     d_common = sqrt(1.925^2 + 0.15^2) + sqrt(0.275^2 + 0.15^2);  % 2, 3
     号点最大判断间距
132     Rectangle_Points = GetRectangles(theta_all, b);
133
134     % 只需判断第 2, 3 号点是否 legal
135
136     % 判断龙头 (第一个板凳)
137         point_1_2(1, 1:2) = Rectangle_Points(2, 1, :);
138         point_1_3(1, 1:2) = Rectangle_Points(3, 1, :);
139
140         % 挑选可能的节点
141         distance = sqrt( b^2*(theta_all(1).^2 + theta_all.^2) -2*b^2.*
    theta_all(1).*theta_all.*cos(theta_all(1)-theta_all) );
```

```
        Logic = (distance < d_special) ;
        Logic(1:2) = 0; % 忽略第 1(本身),2(之后)
        Logic(224) = 0; % 忽略最后一个节点 (无板凳)
        for i = find(Logic)'    % 这里必须转置使得右值为行向量
            % 新的坐标原点
            new_Origin(1, 1:2) = Rectangle_Points(1, i, :);

            % 检查 2 号点
            new_cor = (point_1_2 - new_Origin ) / [
                Rectangle_Points(2, i, 1) - Rectangle_Points(1, i, 1), Rectangle_Points(2, i, 2) - Rectangle_Points(1, i, 2);
                Rectangle_Points(4, i, 1) - Rectangle_Points(1, i, 1), Rectangle_Points(4, i, 2) - Rectangle_Points(1, i, 2);
                ];   % 坐标系转换
            %disp(['龙头2号点 new_cor:', num2str(new_cor)])
            if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&new_cor(2)<1)
                logic = true;
                return
            end
            % 检查 3 号点
            new_cor = (point_1_3 - new_Origin ) / [
                Rectangle_Points(2, i, 1) - Rectangle_Points(1, i, 1), Rectangle_Points(2, i, 2) - Rectangle_Points(1, i, 2);
                Rectangle_Points(4, i, 1) - Rectangle_Points(1, i, 1), Rectangle_Points(4, i, 2) - Rectangle_Points(1, i, 2);
                ];   % 坐标系转换
            %disp(['龙头3号点 new_cor:', num2str(new_cor)])
            if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&new_cor(2)<1)
                logic = true;
                return
            end
        end


    % 判断第二个板凳
        point_2_2(1, 1:2) = Rectangle_Points(2, 2, :);
        point_2_3(1, 1:2) = Rectangle_Points(3, 2, :);

        % 挑选可能的节点
        distance = sqrt( b^2*(theta_all(2).^2 + theta_all.^2) -2*b^2.*
```

```
    theta_all(2).*theta_all.*cos(theta_all(2)-theta_all) );
178         Logic = (distance < d_common) ;
179         Logic(1:3) = 0; % 忽略第 1(龙头),2(本身),3(其后) 节点
180         Logic(224) = 0; % 忽略最后一个节点 (无板凳)
181         for i = find(Logic)'    % 这里必须转置使得右值为行向量
182             % 新的坐标原点
183             new_Origin(1, 1:2) = Rectangle_Points(1, i, :);
184 
185             % 检查 2 号点
186             new_cor = (point_2_2 - new_Origin ) / [
187                 Rectangle_Points(2, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(2, i, 2) - Rectangle_Points(1, i, 2);
188                 Rectangle_Points(4, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(4, i, 2) - Rectangle_Points(1, i, 2);
189                 ];   % 坐标系转换
190             %disp(['板凳2号点 new_cor:', num2str(new_cor)])
191             if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&new_cor
    (2)<1)
192                 logic = true;
193                 return
194             end
195             % 检查 3 号点
196             new_cor = (point_2_3 - new_Origin) / [
197                 Rectangle_Points(2, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(2, i, 2) - Rectangle_Points(1, i, 2);
198                 Rectangle_Points(4, i, 1) - Rectangle_Points(1, i, 1),
    Rectangle_Points(4, i, 2) - Rectangle_Points(1, i, 2);
199                 ];   % 坐标系转换
200             %disp(['板凳3号点 new_cor:', num2str(new_cor)])
201             if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&new_cor
    (2)<1)
202                 logic = true;
203                 return
204             end
205         end
206 
207     % 检查通过
208     logic = false;
209     return
210 end
211 
212 
```

```
213 function PrintResult_Q2(theta, Speed, b)
214     % theta: 224*1 矩阵
215     % Speed: 224*1 矩阵
216     % b: 用于转直角坐标
217     % 注：未进圈的数据本应是 nan，为方便考虑
218     %     这里未做处理，theta_all((theta_all - 16*2*pi) > 0) = nan 即可
        筛选
219
220     % 计算直角坐标位置
221     Coordinates = b*theta.*[cos(theta), sin(theta)];
222
223     % 遍历矩阵，将每个元素格式化为保留 6 位小数的字符串
224     Output = num2cell(zeros(224,3));
225     for i = 1:size(Coordinates, 1)
226             Output{i, 1} = num2str(Coordinates(i, 1), '%.6f');
227             Output{i, 2} = num2str(Coordinates(i, 2), '%.6f');
228     end
229     for i = 1:size(Speed, 1)
230         Output{i, 3} = num2str(Speed(i), '%.6f');
231     end
232
233     writecell(Output, 'Q2_Result.xlsx', 'Sheet', 'Sheet1'); % 输出结果
234 end
235
236
237 %% 问题一函数区 %%
238 %%%%%%%%%%%%%%%%%%
239
240 function dis = GetDistance(theta, theta_0, b)
241     dis = sqrt( b^2*(theta.^2 + theta_0.^2) -2*b^2.*theta.*theta_0.*cos
        (theta-theta_0) );
242 end
243
244 % 此函数已废弃，之后使用牛顿迭代法
245 function [theta, rho] = GetNextPoint( theta_0 , b, d , error_level)
246     % 第 1 层
247     theta_range = theta_0:10^(-1):(theta_0 + 10*pi);
248     for i = 2:length(theta_range)
249         if GetDistance(theta_range(i), theta_0, b) > d
250             break
251         end
252     end
```

```
253
254     % 第 2 ~ error_level 层
255     for j = 2:error_level
256         theta_range = theta_range(i-1):10^(-j):theta_range(i);
257         for i = 2:length(theta_range)
258             if GetDistance(theta_range(i), theta_0, b) > d
259                 break
260             end
261         end
262     end
263
264     % 输出结果
265     theta = 0.5*( theta_range(i-1) + theta_range(i) );
266     rho = b*theta;
267 end
268
269 function theta = GetNextPoint_newton( theta_0 , b, d , error_max_abs)
270     % 第 1 层
271     theta_range = theta_0:10^(-1):(theta_0 + 10*pi);
272     for i = 2:length(theta_range)
273         if GetDistance(theta_range(i), theta_0, b) > d
274             break
275         end
276     end
277
278     % 进行牛顿迭代
279     min = theta_range(i-1);
280     max = theta_range(i);
281
282     for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使
    得 4*pi/2^n < 10^(-20)
283         error_max = max - min;
284         temp = 0.5 * (max + min);
285         temp_dis = GetDistance(temp, theta_0, b);
286         if temp_dis < d
287             min = temp;
288         elseif temp_dis > d
289             max = temp;
290         elseif temp_dis == d
291             theta = temp;
292             return
293         end
```

```
294         if error_max <= error_max_abs
295             break
296         end
297     end
298 
299     % 输出结果
300     theta = 0.5*(min + max);
301 end
302 
303 function theta_all = GetAllPoints( theta_0 , b, d , error_max_abs)
304     theta_all = zeros(224, 1);
305     theta_all(1) = theta_0;
306     % 龙头板凳长不同
307     theta_all(2) = GetNextPoint_newton(theta_all(1), b, 2.86,
    error_max_abs);
308     for i = 3:224
309         theta_all(i) = GetNextPoint_newton(theta_all(i-1), b, d,
    error_max_abs);
310         %polarscatter(theta , r, 100, 'r.');
311     end
312 end
313 
314 
315 function stc = DrawPoints(theta_all , b)
316 
317     X = 0:0.1:(32*pi);
318     Rho = b*X;
319 
320     % 作图
321     figure
322     stc.line = polarplot(X, Rho);
323     hold on
324     stc.point_0 = polarscatter(theta_all(1), b*theta_all(1), 200, 'r.')
    ;
325     stc.points_rest = polarscatter(theta_all(2:end), b*theta_all(2:end)
    , 70, 'b.');
326 
327 % 设置样式
328     % 坐标轴
329         stc.fig = gcf;
330         stc.axes = gca;
331         stc.axes.FontName = "Times New Roman"; % 全局 FontName
```

```
332         stc.axes.Box = 'on';
333
334     % 标题
335         stc.axes.Title.String = 'Figure: Draw Points';
336         stc.axes.Title.FontSize = 17;
337         stc.axes.Title.FontWeight = 'bold';
338
339     % 线的样式
340         stc.line.LineWidth = 1;
341         stc.line.Color = [0 1 0];   % 绿色
342
343     % 收尾
344         hold(stc.axes,'off')
345 end
346
347
348 function Rectangle_Points = GetRectangles(theta_all, b)
349     Coordinates = b*theta_all.*[cos(theta_all), sin(theta_all)];
350     Vec_X = diff(Coordinates);
351     Vec_X = Vec_X ./ sqrt(sum(Vec_X.^2, 2));      % 单位化
352     Vec_N = [ -Vec_X(: ,2), Vec_X(: ,1) ];  % 法向量
353
354
355     % 计算矩形坐标
356     Rectangle_P1 = Coordinates(1:223, :) - Vec_X*0.275 + Vec_N*0.15;
        % 注意是 1:223
357     Rectangle_P2 = Coordinates(1:223, :) - Vec_X*0.275 - Vec_N*0.15;
358     Rectangle_P3 = Coordinates(2:224, :) + Vec_X*0.275 - Vec_N*0.15;
        % 注意是 2:224
359     Rectangle_P4 = Coordinates(2:224, :) + Vec_X*0.275 + Vec_N*0.15;
360
361     Rectangle_Points = zeros(4, 223, 2);
362     Rectangle_Points(1, :, :) = Rectangle_P1;
363     Rectangle_Points(2, :, :) = Rectangle_P2;
364     Rectangle_Points(3, :, :) = Rectangle_P3;
365     Rectangle_Points(4, :, :) = Rectangle_P4;
366 end
367
368
369 function stc = DrawPointsAndRectangles(theta_all, b)
370 % 作线和点
371     x = 0:0.1:(32*pi);
```

```
372     X = x';
373
374     % 转为直角坐标
375         coor_line = [
376             b*X.*cos(X), b*X.*sin(X)
377         ];
378         coor_all = [
379             b*theta_all.*cos(theta_all), b*theta_all.*sin(theta_all)
380         ];
381
382     % 作图
383         figure
384         stc.line = plot(coor_line(:, 1), coor_line(:, 2));
385         hold on
386         stc.point_0 = scatter(coor_all(1, 1), coor_all(1, 2), 200, 'r.'
    );
387         stc.points_rest = scatter(coor_all(2:end, 1), coor_all(2:end,
    2), 70, 'b.');
388
389     % 设置样式
390         % 坐标轴
391             stc.fig = gcf;
392             axis equal
393             stc.axes = gca;
394             stc.axes.FontName = "Times New Roman"; % 全局 FontName
395             stc.axes.Box = 'on';
396             stc.axes.FontSize = 14;
397             xline(0, 'LineWidth', 0.3, 'Color', [0.3, 0.3, 0.3]);
398             yline(0, 'LineWidth', 0.3, 'Color', [0.3, 0.3, 0.3]);
399             stc.label.x = xlabel(stc.axes, '$x$', 'Interpreter', 'latex
    ', 'FontSize', 15);
400             stc.label.y = ylabel(stc.axes, '$y$', 'Interpreter', 'latex
    ', 'FontSize', 15);
401
402
403         % 标题
404             stc.axes.Title.String = 'Figure: Draw Rectangles';
405             stc.axes.Title.FontSize = 17;
406             stc.axes.Title.FontWeight = 'bold';
407
408         % 线的样式
409             stc.line.LineWidth = 1;
```

```
410             stc.line.Color = [0 1 0];   % 绿色
411
412 % 作方框
413     Rectangle_Points = GetRectangles(theta_all, b);
414     Rectangle_Points(5, :, :) = Rectangle_Points(1, :, :);    % plot 围
    成闭合曲线
415     hold on
416     for i = 1:223
417         plot(Rectangle_Points(:, i, 1), Rectangle_Points(:, i, 2), '
    LineWidth', 0.3, 'Color', [1 0 1]);
418     end
419
420 % 收尾
421     hold off
422 end
423
424
425 function Speed_all = GetAllSpeed(theta_all, b, v_0)
426     Speed_all = zeros(224 ,1);
427     Speed_all(1) = v_0;
428
429     % 转为直角坐标
430     Coordinates = b*theta_all.*[cos(theta_all), sin(theta_all)];
431     Vec_X = diff(Coordinates);
432     % 下面注意要有负号
433     Vec_Tao = -[cos(theta_all) - theta_all.*sin(theta_all),  sin(
    theta_all) + theta_all.*cos(theta_all)];
434     for i = 1:223
435         Speed_all(i+1) = Speed_all(i) * ( norm(Vec_Tao(i+1, :))*abs(sum
    (Vec_Tao(i, :).*Vec_X(i, :)))  ) / ( norm(Vec_Tao(i, :))*abs(sum(
    Vec_Tao(i+1, :).*Vec_X(i, :))) );
436     end
437 end
438
439
440 % 此函数已废弃，因为 .xlsx 格式不同
441 function printLocation(theta_all, Speed_all, b, column_initial)
442     %logic_matrix = (theta_all - 16*2*pi) > 0;
443     theta_all((theta_all - 16*2*pi) > 0) = nan;    % 将不合法 (未进圈)
    的数据设为 nan
444     Speed_all((theta_all - 16*2*pi) > 0) = nan;
445     % 计算直角坐标位置
```

```
446     Coordinates = b*theta_all.*[cos(theta_all), sin(theta_all)];
447     rowHeaders = arrayfun(@(i) sprintf('第%d节点', i), 1:length(
        theta_all), 'UniformOutput', false);
448
449     Output = array2table([Coordinates, Speed_all], "RowNames",
        rowHeaders, "VariableNames", {'位置 x', '位置 y', '速率 v'});
450     writetable(Output, 'Q1_Result.xlsx', 'WriteRowNames', true, 'Range'
        , column_initial);
451 end
452
453 function PrintResult_Q1(theta, Speed, b)
454     % theta: 224*301 矩阵
455     % Speed: 224*301 矩阵
456     % b: 用于转直角坐标
457     % 注: 未进圈的数据本应是 nan, 为方便考虑
458     %     这里未做处理, theta_all((theta_all - 16*2*pi) > 0) = nan 即可
        筛选
459
460     t_length = size(theta, 2);
461
462     % 计算直角坐标位置
463     Coordinates = zeros(2, 224, t_length);
464     Coordinates(1, :, :) = b*theta.*cos(theta);
465     Coordinates(2, :, :) = b*theta.*sin(theta);
466     Location = zeros(2*224, t_length);
467     for i = 1:2*224
468         if mod(i, 2) == 1   % i 奇数, 对应 x
469             Location(i, :) = Coordinates(1, ceil(i/2), :);
470         else                % i 偶数, 对应 y
471             Location(i, :) = Coordinates(2, i/2, :);
472         end
473     end
474     % 遍历矩阵, 将每个元素格式化为保留 6 位小数的字符串
475     for i = 1:size(Location, 1)
476         for j = 1:size(Location, 2)
477             Location_str{i, j} = num2str(Location(i, j), '%.6f');
478         end
479     end
480     for i = 1:size(Speed, 1)
481         for j = 1:size(Speed, 2)
482             Speed_str{i, j} = num2str(Speed(i, j), '%.6f');
483         end
```

```
    end

    writecell(Location_str, 'Q1_Result.xlsx', 'Sheet', 'Sheet1'); % 输出位置
    writecell(Speed_str, 'Q1_Result.xlsx', 'Sheet', 'Sheet2'); % 输出速度
end

function theta = GetTheta(b, v, t)
    % 弧长公式
    S = @(theta) b/2 * (  theta.*sqrt(1+theta.^2) + log(theta + sqrt(1+theta.^2))  );
    S_real = @(theta) S(16*2*pi) - S(theta);

    % 第 1 层
    theta_range = 16*2*pi:(-10^(-1)):5*pi;
    for i = 2:length(theta_range)
        if GetDistance(theta_range(i), theta_0, b) > d
            break
        end
    end

    % 进行牛顿迭代
    min = theta_range(i-1);
    max = theta_range(i);

    for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使得 4*pi/2^n < 10^(-20)
        error_max = max - min;
        temp = 0.5 * (max + min);
        temp_dis = GetDistance(temp, theta_0, b);
        if temp_dis < d
            min = temp;
        elseif temp_dis > d
            max = temp;
        elseif temp_dis == d
            theta = temp;
            return
        end
        if error_max <= error_max_abs
            break
        end
```

```
    end
    % 输入参数
    % b = 55 / (2 * pi);     % 螺线递增参数 (b)
    % theta0 = 16 * 2 * pi; % 初始点的极坐标角度 (θ_0)
    % L = 500;               % 沿螺线走的距离 (L)
end

function theta_final = func_find_start(b, theta0, L)
    % 初始化参数
    % b = 55 / (2 * pi);     % 螺线递增参数 (b)
    % theta0 = 16 * 2 * pi; % 初始点的极坐标角度 (θ_0)
    % L = 500;               % 沿螺线走的距离 (L)

    % 定义螺旋线方程函数
    r = @(theta) b * (theta0-theta);        % 极径方程：r(θ) = b(θ_0-θ)
    ds = @(theta) sqrt(b^2 + r(theta).^2); % 曲线长度微元
    r_real = @(theta) b * theta;        % 极径方程：r(θ) = b(θ_0-θ)
    % 使用Numerical Integration计算θ值
    theta_final = fzero(@(theta) integral(ds, 0, theta) - L, theta0 +
    2*pi);

end
```

B.3.2 模拟退火函数

```
function [stc_SA, stc_Figure] = MySimulatedAnnealing(stc_SA, objective,
    d, v_0, t_range, t_step1, t_step2, t_error_max_abs)
% 输入退火问题结构体，输出迭代结果（minimize）。
%
% 输入：
    % stc_SA：退火问题结构体
    % objective：目标函数
% 输出：迭代结果
% 注：迭代总次数为 1000 时，在 waitbar 上共耗时约 0.8 s
%%
    % 步骤一：初始化
        %Waitbar = waitbar(0, '1', 'Name', 'Simulated Annealing', '
    CreateCancelBtn', 'delete(gcbf)','Color', [0.9, 0.9, 0.9]);
        %Btn = findall(Waitbar, 'type', 'Uicontrol');
        %set(Btn, 'String', 'Cancle', 'FontSize', 10);
         Waitbar = waitbar(0, '1', 'Name', 'Simulated Annealing', 'Color
    ', [0.9, 0.9, 0.9]);
```

```
15          TK = stc_SA.Annealing.T_0;
16          T_end = stc_SA.Annealing.T_end;
17          mkv = stc_SA.Annealing.mkvlength;
18          alpha = stc_SA.Annealing.alpha;
19          num_Var = size(stc_SA.Var.range(:,3));
20          num_Var = num_Var(1);
21          %lam = stc.Annealing.lam;
22          % 迭代记录仪初始化
23          change_1 = 0;     % 随机到更优解的次数
24          change_2 = 0;     % 接收较差解的次数（两者约 10:1 时有较好寻优效果）
25          mytry = 1;        % 当前迭代次数
26          N = mkv*ceil(log(T_end/TK)/log(alpha));    % 迭代总次数
27          X = stc_SA.Var.range(:,3)';
28          X_best = stc_SA.Var.range(:,3)';
29          f_best = objective(X, d, v_0, t_range, t_step1, t_step2, t_error_max_abs); % 计算目标函数
30          process = zeros(1, floor(N));
31          process(1) = f_best;
32          process_best = zeros(1, floor(N));
33          process_best(1) = f_best;
34
35
36      % 步骤二：退火
37          disp("初始化完成，开始退火")
38          disp(['initial obj: ',num2str(f_best)])
39          start = tic; % 开始计时
40          while TK >= T_end
41              for i = 1:mkv  % 每个温度T下，我们都寻找 mkv 次新解 X，每一个新解都有可能被接受
42
43                  r = rand;
44                  if r>=0.5 % 在当前较优解附近扰动
45                      for j = 1:num_Var
46                          X(j) = X_best(j)+(rand-0.5)*(stc_SA.Var.range(j,2) - stc_SA.Var.range(j,1))*( 1-(mytry-1)/N )^2;
47                          X(j) = max(stc_SA.Var.range(j,1), min(X(j), stc_SA.Var.range(j,2)));    % 确保扰动后的 X 仍在范围内
48                      end
49                  else % 生成全局随机解
50                      X = (stc_SA.Var.range(:,1) + rand*(stc_SA.Var.range(:,2) - stc_SA.Var.range(:,1)))'; % 转置后才是行向量
```

```
51                  end
52
53                  mytry = mytry+1;
54                  f = objective(X, d, v_0, t_range, t_step1, t_step2,
    t_error_max_abs); % 计算目标函数
55
56                  if f < f_best   % 随机到更优解，接受新解
57                      f_best = f;
58                      X_best = X;
59                      change_1 = change_1+1;
60                     % disp(['较优参数为：',num2str(X_best)])
61                      disp(['    new obj: ',num2str(f_best)])
62                  elseif exp( - 10^7 * abs((f_best-f)/f_best) /TK  ) >
    rand  % 满足概率，接受较差解
63                      f_best = f;
64                      X_best = X;
65                     % disp(['较优参数为：',num2str(X_best)])
66                      disp(['    new obj: ',num2str(f_best)])
67                      change_2 = change_2 + 1;
68                  end
69
70                  process(mytry) = f;
71                  process_best(mytry) = f_best;
72              end
73             %disp(['进度：',num2str((mytry)/N*100),'%'])
74              TK = TK*alpha;
75              waitbar((mytry)/N*100, Waitbar, ['Computing: ', num2str(
    round((mytry)/N*100, 1)), '%']);
76          end
77
78      % 步骤三：退火结束，输出最终结果
79          % 进度条
80              time = toc(start);
81              waitbar(1, Waitbar, ['Simulated Annealing Completed (in ',
    num2str(time),' s)']);
82              Waitbar.Color = [1 1 1];
83          % 图像
84              stc_SA.process = process;
85              stc_SA.process_best = process_best;
86              stc_SA.X_best = X_best;
87              stc_SA.Object_best = f_best;
88
```

```
89
90          stc_Figure = MyYYPlot(1:length(process), process_best, 1:
    length(process), process);
91          stc_Figure.axes.Title.String = ['Simulated Annealing (in ',
     num2str(time),' s)'];
92          %stc_Figure.axes.YLimitMethod = "padded";
93          stc_Figure.leg.String = ["Best objective value";"Current
    objective value"];
94          stc_Figure.leg.Location = "northeast";
95          stc_Figure.p_left.LineWidth = 5;
96          stc_Figure.p_right.LineWidth = 1;
97          stc_Figure.label.x.String = 'times';
98          stc_Figure.label.y_left.String = '$obj_{\mathrm{best}}$';
99          stc_Figure.label.y_right.String = '$obj_{\mathrm{current}}$
    ';
100
101     % 文本
102         disp('----------------------------------')
103         disp('>> --------  模拟退火  -------- <<')
104         disp(['历时 ', num2str(time), ' 秒'])
105         disp(['一共寻找新解: ',num2str(mytry)])
106         disp(['change_1次数: ',num2str(change_1)])
107         disp(['change_2次数: ',num2str(change_2)])
108         disp(['最优参数: ', num2str(X_best)])
109         disp(['最优目标值: ', num2str(f_best)])
110         disp('>> --------  模拟退火  -------- <<')
111         disp('----------------------------------')
112
113     % 导出数据
114         output = cell(1, 6 + num_Var);
115         output{1, 1} = datestr(now, 'yyyy-mm-dd HH:MM:SS');
116         output{1, 2} = 'Spend (s)';
117         output{1, 3} = num2str(time, '%.6f');
118         output{1, 4} = 'X_best';
119         for i = 1:num_Var
120             output{1, 4+i} = num2str(stc_SA.X_best(i), '%.12f');
121         end
122         output{1, 5+num_Var} = 'Object_best';
123         output{1, 6+num_Var} = num2str(stc_SA.Object_best, '%.12f')
    ;
124         writecell(output, 'MySimulatedAnnealingResualts.xlsx', "
    WriteMode","append");
```

```
            %writematrix([datestr(now, 'yyyy-mm-dd HH:MM:SS'), "Spend (s)", time, 'X_best', vpa(stc_SA.X_best), 'Object_best', vpa(stc_SA.Object_best)], 'MySimulatedAnnealingResualts.xlsx', "WriteMode","append");
end
```

B.4 问题 4 代码

```
clc, clear,close all
%% 程序主代码区 %%
%%%%%%%%%%%%%%%%%%
% 获取最小调头半径, 考虑碰撞和速度限制 (实际是否能通过)
    t_step = 0.05;
    R_range_init = [0, 4.5];
    R_step1 = 0.5;
    R_step2 = 0.1;
    error_max_abs = 10^(-12);
    %R_t_min = 4.254673736180;
     R_t_min = 4.25467374;
    [R_t_min, ~] = GetMinRadius(t_step, R_range_init, R_step1, R_step2, error_max_abs);


% 根据此调头半径计算所需参数
    % 给定 R_turn_min
    global R_t; R_t = R_t_min;

    ChangeGlobalVaribles_Q4

% 由题目直接确定的参数
global d d_0 luoju b v_0 theta_max
    d = 1.65;    % 普通节点间距
    d_0 = 2.86;    % 龙头节点间距
    luoju = 1.7; % 螺距
    a = 0; % 阿基米德螺线的起始半径
    b = luoju / (2*pi); % 每圈的增长量 (与螺距有关)
    theta_max = 16 * 2 * pi;       % 初始点极坐标角度
    v_0 = 1;

if 0
% 依次求解全部所需全局变量
```

```
34      global theta_t coor_M coor_H coor_E dist_HE l_M l_H t_turn ...
35          l_HA coor_A coor_C R_small R_big lambda coor_HA kappa
36          theta_t = R_t/b;
37          func = @(x)b*sqrt(x.^2+1);
38          t_turn = 1/v_0*integral(func, theta_t , 32*pi); % 从进入开始计
    时, 到恰好开始转弯需要的时间
39          coor_M = b*theta_t*[cos(theta_t), sin(theta_t)];
40          coor_H = -coor_M;
41          coor_E = coor_H/3;
42          dist_HE = R_t/3;
43          l_M = - [cos(theta_t) - theta_t.*sin(theta_t), sin(theta_t) +
    theta_t.*cos(theta_t)];
44          l_H = - l_M;
45          l_HA = - cross([0,0,1], [l_H, 0]); l_HA = l_HA(1:2); l_HA =
    l_HA/norm(l_HA);    % cross 前补了个负号
46          func = @(t) norm(coor_E - (coor_H + t*l_HA)) - norm(coor_H - (
    coor_H + t*l_HA));
47          t = fzero(func, 0);
48          coor_A = coor_H + t*l_HA;
49          coor_C = 3*coor_E - 2*coor_A;
50          R_small = norm(coor_A - coor_H);
51          R_big = 2*R_small;
52          lambda = acos(1 - norm(coor_H-coor_E)^2/(2*R_small^2));
53          coor_HA = coor_A - coor_H;
54          kappa = pi - atan(coor_HA(2)/coor_HA(1));
55          % 注意 kappa 的范围, 需要作判断 if(-coor_HA(1) < 0){ kappa -->
    kappa - pi}以纠正
56          if ( - coor_HA(1) < 0)
57              kappa = kappa - pi;
58          end
59
60      % 构建全局函数
61      global P_C P_A
62          % 圆 C 的参数方程 (返回直角坐标)
63          P_C = @(beta) ( coor_C + R_big *cos(beta + kappa).*[1 0] -
    R_big *sin(beta + kappa).*[0 1] );
64          % 圆 A 的参数方程 (返回直角坐标)
65          P_A = @(beta) ( coor_A - R_small *cos((-beta) + kappa + lambda)
    .*[1 0] + R_small *sin((-beta) + kappa + lambda ).*[0 1]);
66
67      % 用于计算距离的全局函数族
68      global func_dist_21 func_dist_22 func_dist_32 func_dist_33
```

```
    func_dist_42 func_dist_43 func_dist_44
69          func_dist_21 = @(beta, theta, d) ... % 转直角后用距离公式, 并作
    映射 dist --> 2*d - dist 以满足 min 时 > d
70              2*d - norm(P_C(beta) - b*theta.*[cos(theta), sin(theta)]);
71          func_dist_22 = @(beta_1, beta_2) R_big * sqrt( 2*(1-cos(beta_1
    - beta_2)) ); % 半径是 R_big
72          func_dist_32 = @(beta_1, beta_2) norm(P_A(beta_1) - P_C(beta_2)
    ); % 注意 beta_1 是 P_A
73          func_dist_33 = @(beta_1, beta_2) R_small * sqrt( 2*(1-cos(
    beta_1 - beta_2)) ); % 半径是 R_small
74          func_dist_42 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称 (取负) 才是真实坐标
75              norm( - b*theta.*[cos(theta), sin(theta)] - P_C(beta));
76          func_dist_43 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称才是真实坐标
77              norm( - b*theta.*[cos(theta), sin(theta)] - P_A(beta));
78          func_dist_44 = @(theta_1, theta_2) ...
79              norm( b*theta_1.*[cos(theta_1), sin(theta_1)] - b*theta_2
    .*[cos(theta_2), sin(theta_2)]);
80  end
81
82  % 检查 R_turn_min 是否可行
83
84      [logic_crush, logic_speederror, t_crushing] = CheckCrushAndSpeed_Q4
    ([0, 40], 0.1);
85      if (logic_crush == true) || (logic_speederror == true)
86          disp(['R_t = ', num2str(R_t, '%.12f')]);
87          error(">> error:  当前 R_t 发生碰撞或无法通过, 须重新设置 R_t
    值")
88      end
89
90  % 输出 Q4 最终结果
91      t_array = -100:1:100;
92      coordinates_matrix = zeros(224*2, length(t_array));
93      speed_matrix = zeros(224, length(t_array));
94      for j = 1:length(t_array)
95          t = t_array(j);
96          % 求解此 t 下的直角坐标
97          angel_withflag_0 = GetHeadLocation(v_0, t);
98          angle_withflag_all = GetAllPoints_Q4(angel_withflag_0, 10^(-16)
    );
99          coordinates_array = Angle2Coor(angle_withflag_all);
```

```
100         speed_matrix(:, j) = GetAllSpeed_Q4(angle_withflag_all, v_0);
101         % 将坐标格式转为 xlsx 中的模版格式
102         for i = 1:2*224
103             if mod(i, 2) == 1   % i 奇数, 对应 x
104                 coordinates_matrix(i, j) = coordinates_array(ceil(i/2),
    1);
105             else                % i 偶数, 对应 y
106                 coordinates_matrix(i, j) = coordinates_array(i/2, 2);
107             end
108         end
109     end
110     PrintResult_Q4(coordinates_matrix, speed_matrix)
111
112 % 可视化部分
113     % 路径可视化
114         DrawWholePath
115     % t = 100 结果可视化
116         t = 100;
117         angle_withflag_0 = GetHeadLocation(v_0, t);
118         angle_withflag_all = GetAllPoints_Q4(angle_withflag_0, 10^(-16)
    );
119         coor_all = Angle2Coor(angle_withflag_all);
120         coor_Rectangles = GetRectangles_Q4(coor_all);
121         DrawPoints_Q4(coor_all);
122         DrawPointsAndRectangles_Q4(coor_all, coor_Rectangles);
123
124
125 %% 问题四函数区 %%
126 %%%%%%%%%%%%%%%%%%
127
128
129 function ChangeGlobalVaribles_Q4
130 % 引入全局变量
131 global R_t d d_0 luoju b v_0 theta_max
132 % 依次求解全部所需全局变量
133     global theta_t coor_M coor_H coor_E dist_HE l_M l_H t_turn ...
134             l_HA coor_A coor_C R_small R_big lambda coor_HA kappa
135
136         theta_t = R_t/b;
137         func = @(x)b*sqrt(x.^2+1);
138         t_turn = 1/v_0*integral(func, theta_t , 32*pi); % 从进入开始计
    时, 到恰好开始转弯需要的时间
```

```
        coor_M = b*theta_t*[cos(theta_t), sin(theta_t)];
        coor_H = -coor_M;
        coor_E = coor_H/3;
        dist_HE = R_t/3;
        l_M = - [cos(theta_t) - theta_t.*sin(theta_t),  sin(theta_t) + theta_t.*cos(theta_t)];
        l_H = - l_M;
        l_HA = - cross([0,0,1], [l_H, 0]); l_HA = l_HA(1:2); l_HA = l_HA/norm(l_HA);     % cross 前补了个负号
        func = @(t) norm(coor_E - (coor_H + t*l_HA)) - norm(coor_H - (coor_H + t*l_HA));
        t = fzero(func, 0);
        coor_A = coor_H + t*l_HA;
        coor_C = 3*coor_E - 2*coor_A;
        R_small = norm(coor_A - coor_H);
        R_big = 2*R_small;
        lambda = acos(1 - norm(coor_H-coor_E)^2/(2*R_small^2));
        coor_HA = coor_A - coor_H;
        kappa = pi - atan(coor_HA(2)/coor_HA(1));
       % 注意 kappa 的范围, 需要作判断 if(-coor_HA(1) < 0){ kappa --> kappa - pi}以纠正
        if ( - coor_HA(1) < 0)
            kappa = kappa - pi;
        end

    % 构建全局函数
     global P_C P_A
        % 圆 C 的参数方程 (返回直角坐标)
        P_C = @(beta) ( coor_C + R_big *cos(beta + kappa).*[1 0] - R_big *sin(beta + kappa).*[0 1] );
        % 圆 A 的参数方程 (返回直角坐标)
        P_A = @(beta) ( coor_A - R_small *cos((-beta) + kappa + lambda).*[1 0] + R_small *sin((-beta)  + kappa + lambda ).*[0 1]);

    % 用于计算距离的全局函数族
     global func_dist_21 func_dist_22 func_dist_32 func_dist_33 func_dist_42 func_dist_43 func_dist_44
        func_dist_21 = @(beta, theta, d) ... % 转直角后用距离公式, 并作映射 dist --> 2*d - dist 以满足 min 时 > d
            2*d - norm(P_C(beta) - b*theta.*[cos(theta), sin(theta)]);
        func_dist_22 = @(beta_1, beta_2) R_big * sqrt( 2*(1-cos(beta_1 - beta_2)) ); % 半径是 R_big
```

```
172         func_dist_32 = @(beta_1, beta_2) norm(P_A(beta_1) - P_C(beta_2)
    ); % 注意 beta_1 是 P_A
173         func_dist_33 = @(beta_1, beta_2) R_small * sqrt( 2*(1-cos(
    beta_1 - beta_2)) ); % 半径是 R_small
174         func_dist_42 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称 (取负) 才是真实坐标
175             norm( - b*theta.*[cos(theta), sin(theta)] - P_C(beta));
176         func_dist_43 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称才是真实坐标
177             norm( - b*theta.*[cos(theta), sin(theta)] - P_A(beta));
178         func_dist_44 = @(theta_1, theta_2) ...
179             norm( b*theta_1.*[cos(theta_1), sin(theta_1)] - b*theta_2
    .*[cos(theta_2), sin(theta_2)]);
180 end
181
182
183 function [R_t_min, t_crushing] = GetMinRadius(t_step, R_range_init,
    R_step1, R_step2, error_max_abs)
184 global R_t v_0 b
185     disp('函数 GetMinRadius 运行时会修改几乎所有全局变量')
186     disp('需在运行后对全局变量重新赋值!')
187     R_range = R_range_init
188
189     start = tic;
190
191     % 第一层步长: R_step1
192     R_array = R_range(1):R_step1:R_range(2);
193         for i = length(R_array):(-1):1
194             R_t = R_array(i);
195             %func = @(x)b*sqrt(x.^2+1);
196             %t_turn_now = 1/v_0*integral(func, R_t/b , 32*pi); % 从进入
    开始计时, 到恰好开始转弯需要的时间
197             %[logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4([0, t_turn_now/3], t_step);
198             [logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4([0, 40], t_step);
199             disp(['crush = ', num2str(logic_crush), ', speederror = ',
    num2str(logic_speederror), ', R = ', num2str(R_t), ', t_crushing = '
    , num2str(t_crushing)]);
200             if (logic_crush == true) || (logic_speederror == true)
201                 R_range = [R_array(i), R_array(i+1)]
202                 break
```

```
203             end
204         end
205
206     % 第二层步长: R_step2
207     R_array = R_range(1):R_step2:R_range(2);
208         for i = length(R_array):(-1):1
209             R_t = R_array(i);
210             %func = @(x)b*sqrt(x.^2+1);
211             %t_turn_now = 1/v_0*integral(func, R_t/b , 32*pi); % 从进入
    开始计时, 到恰好开始转弯需要的时间
212             %[logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4([0, t_turn_now/3], t_step);
213             [logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4([0, 40], t_step);
214             disp(['crush = ', num2str(logic_crush), ', speederror = ',
    num2str(logic_speederror), ', R = ', num2str(R_t), ', t_crushing = '
    , num2str(t_crushing)]);
215             if (logic_crush == true) || (logic_speederror == true)
216                 R_range = [R_array(i), R_array(i+1)]
217                 break
218             end
219         end
220
221     % 牛顿迭代法
222     min = R_range(1);
223     max = R_range(2);
224     for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使
    得 4*pi/2^n < 10^(-20)
225         error_max = max - min;
226         temp = 0.5 * (max + min);
227         R_t = temp;
228         %func = @(x)b*sqrt(x.^2+1);
229         %t_turn_now = 1/v_0*integral(func, R_t/b , 32*pi); % 从进入开始
    计时, 到恰好开始转弯需要的时间
230         %[logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4([0, t_turn_now/3], t_step);
231         [temp_logic_crush, temp_logic_speederror, temp_t] =
    CheckCrushAndSpeed_Q4([0, 40], t_step);
232         if (temp_logic_crush == true) || (temp_logic_speederror == true
    )
233             min = temp
234             t_crushing = temp_t;
```

```
235         else
236             max = temp
237         end
238         disp(['crush = ', num2str(temp_logic_crush), ', speederror = ',
        num2str(temp_logic_speederror), ', R = ', num2str(R_t), ',
        t_crushing = ', num2str(t_crushing)]);
239         if error_max <= error_max_abs
240             break
241         end
242     end
243
244     % 记录结果
245     R_t_min = 0.5*(min + max);
246     time = toc(start);
247     output_cell{1, 1} = num2str(time, '%.6f');
248     output_cell{1, 2} = num2str(R_t_min, '%.12f');
249     output_cell{1, 3} = num2str(t_crushing, '%.12f');
250     output_cell{1, 4} = num2str(t_step, '%.6f');
251     output_cell{1, 4} = num2str(R_range_init, '%.6f');
252     output_cell{1, 5} = num2str(R_step1, '%.6f');
253     output_cell{1, 6} = num2str(R_step2, '%.6f');
254     output_cell{1, 7} = num2str(error_max_abs, '%.20f');
255     disp(['time = ', num2str(time, '%.6f')])
256     disp(['R_t_min = ', num2str(R_t_min, '%.12f')])
257     disp(['t_crushing = ', num2str(t_crushing, '%.6f')])
258     disp(['t_step = ', num2str(t_step, '%.4f')])
259     disp(['R_range_init = [', num2str(R_range_init(1), '%.3f'), ' ,',
        num2str(R_range_init(2), '%.3f'), ']'])
260     disp(['R_step1 = ', num2str(R_step1, '%.4f')])
261     disp(['R_step2 = ', num2str(R_step2, '%.4f')])
262     disp(['R_error_max_abs = ', num2str(error_max_abs, '%.16f')])
263     writecell(output_cell, 'MinRadius_Result.xlsx', 'WriteMode', '
        append');
264 end
265
266 function [logic_crush, logic_speederror, t_crushing] =
        CheckCrushAndSpeed_Q4(t_range, t_step)
267
268     % 外层函数修改了 R_turn
269     global R_t;
270
271     % 题目确定的参数
```

```
272     % 参数设置
273     global d d_0 luoju b v_0 theta_max
274         d = 1.65;     % 普通节点间距
275         d_0 = 2.86;     % 龙头节点间距
276         luoju = 1.7; % 螺距
277         a = 0; % 阿基米德螺线的起始半径
278         b = luoju / (2*pi); % 每圈的增长量 (与螺距有关)
279         theta_max = 16 * 2 * pi;        % 初始点极坐标角度
280         v_0 = 1;
281
282     % 依次求解全部所需全局变量
283     global theta_t coor_M coor_H coor_E dist_HE l_M l_H t_turn ...
284             l_HA coor_A coor_C R_small R_big lambda coor_HA kappa
285         theta_t = R_t/b;
286         func = @(x)b*sqrt(x.^2+1);
287         t_turn = 1/v_0*integral(func, theta_t , 32*pi); % 从进入开始计
    时, 到恰好开始转弯需要的时间
288         coor_M = b*theta_t*[cos(theta_t), sin(theta_t)];
289         coor_H = -coor_M;
290         coor_E = coor_H/3;
291         dist_HE = R_t/3;
292         l_M = - [cos(theta_t) - theta_t.*sin(theta_t),  sin(theta_t) +
    theta_t.*cos(theta_t)];
293         l_H = - l_M;
294         l_HA = - cross([0,0,1], [l_H, 0]); l_HA = l_HA(1:2); l_HA =
    l_HA/norm(l_HA);     % cross 前补了个负号
295         func = @(t) norm(coor_E - (coor_H + t*l_HA)) - norm(coor_H - (
    coor_H + t*l_HA));
296         h = fzero(func, 0);
297         coor_A = coor_H + h*l_HA;
298         coor_C = 3*coor_E - 2*coor_A;
299         R_small = norm(coor_A - coor_H);
300         R_big = 2*R_small;
301         lambda = acos(1 - norm(coor_H-coor_E)^2/(2*R_small^2));
302         coor_HA = coor_A - coor_H;
303         kappa = pi - atan(coor_HA(2)/coor_HA(1));
304        % 注意 kappa 的范围, 需要作判断 if(-coor_HA(1) < 0){ kappa -->
    kappa - pi}以纠正
305         if ( - coor_HA(1) < 0)
306             kappa = kappa - pi;
307         end
308
```

```
309     % 构建全局函数
310     global P_C P_A
311         % 圆 C 的参数方程 (返回直角坐标)
312         P_C = @(beta) ( coor_C + R_big *cos(beta + kappa).*[1 0] -
    R_big *sin(beta + kappa).*[0 1] );
313         % 圆 A 的参数方程 (返回直角坐标)
314         P_A = @(beta) ( coor_A - R_small *cos((-beta) + kappa + lambda)
    .*[1 0] + R_small *sin((-beta)  + kappa + lambda ).*[0 1]);
315
316     % 用于计算距离的全局函数族
317     global func_dist_21 func_dist_22 func_dist_32 func_dist_33
    func_dist_42 func_dist_43 func_dist_44
318         func_dist_21 = @(beta, theta, d) ... % 转直角后用距离公式, 并作
    映射 dist --> 2*d - dist 以满足 min 时 > d
319             2*d - norm(P_C(beta) - b*theta.*[cos(theta), sin(theta)]);
320         func_dist_22 = @(beta_1, beta_2) R_big * sqrt( 2*(1-cos(beta_1
    - beta_2)) ); % 半径是 R_big
321         func_dist_32 = @(beta_1, beta_2) norm(P_A(beta_1) - P_C(beta_2)
    ); % 注意 beta_1 是 P_A
322         func_dist_33 = @(beta_1, beta_2) R_small * sqrt( 2*(1-cos(
    beta_1 - beta_2)) ); % 半径是 R_small
323         func_dist_42 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称 (取负) 才是真实坐标
324             norm( - b*theta.*[cos(theta), sin(theta)] - P_C(beta));
325         func_dist_43 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称才是真实坐标
326             norm( - b*theta.*[cos(theta), sin(theta)] - P_A(beta));
327         func_dist_44 = @(theta_1, theta_2) ...
328             norm( b*theta_1.*[cos(theta_1), sin(theta_1)] - b*theta_2
    .*[cos(theta_2), sin(theta_2)]);
329
330
331     for t = t_range(1):t_step:t_range(2)
332         angle_withflag_0 = GetHeadLocation(v_0, t);
333         angle_withflag_all = GetAllPoints_Q4(angle_withflag_0, 10^(-16)
    );
334         coor_all = Angle2Coor(angle_withflag_all);
335         logic_crush = IsCrushed_Q4(coor_all);
336         if logic_crush == true
337             t_crushing = t;
338             logic_speederror = ~isempty(GetAllSpeed_Q4(
    angle_withflag_all, v_0)<0);
```

```
339             return
340         end
341         speed_all = GetAllSpeed_Q4(angle_withflag_all, v_0);
342         if ~isempty(find(speed_all<0, 1))
343             t_crushing = t;
344             logic_speederror = true;
345             return
346         end
347     end
348
349     % no crush or speed error
350     logic_speederror = false;
351     t_crushing = -1;
352     return
353 end
354
355 function logic = IsCrushed_Q4(coor_all)
356     % 数据准备
357     d_special = sqrt(3.135^2 + 0.15^2) + sqrt(0.275^2 + 0.15^2);  % 龙
    头 3 号点最大判断间距
358     d_common = sqrt(1.925^2 + 0.15^2) + sqrt(0.275^2 + 0.15^2);  % 2, 3
     号点最大判断间距
359     coor_Rectangles = GetRectangles_Q4(coor_all);
360
361     % 与问题二不同，这里需要判断全部方框的四个点是否 legal
362
363     % 判断龙头 (第一个板凳)
364         point_four(1:4, 1:2) = coor_Rectangles(1:4, 1, :);
365
366         % 挑选可能的节点
367         distance_array = sqrt( sum( ( coor_all - coor_all(1,:) ).^2 ,
    2) );
368         Logic = (distance_array < d_special) ;
369         Logic(1:2) = 0; % 忽略第 1(本身),2(之后)
370         Logic(224) = 0; % 忽略最后一个节点 (无板凳)
371         for i = find(Logic)'    % 这里必须转置使得右值为行向量
372             % 新的坐标原点
373             new_Origin(1, 1:2) = coor_Rectangles(1, i, :);
374             % 依次检查 4 个点
375             for k = 1:4
376                 new_cor = (point_four(k, :) - new_Origin ) / [
377                     coor_Rectangles(2, i, 1) - coor_Rectangles(1, i, 1)
```

```
, coor_Rectangles(2, i, 2) - coor_Rectangles(1, i, 2);
                    coor_Rectangles(4, i, 1) - coor_Rectangles(1, i, 1)
, coor_Rectangles(4, i, 2) - coor_Rectangles(1, i, 2);
                    ];   % 坐标系转换
                %disp(['龙头2号点 new_cor:', num2str(new_cor)])
                if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&
new_cor(2)<1)
                    logic = true;
                    %disp(['龙头: i = ',num2str(i),', k = ',num2str(k)
])
                    return
                end
            end
        end

    % 判断其它板凳
        for j = 2:223
            point_four(1:4, 1:2) = coor_Rectangles(1:4, j, :);

            % 挑选可能的节点
            distance_array = sqrt( sum( ( coor_all - coor_all(1,:) ).^2
 , 2) );
            Logic = (distance_array < d_common) ;
            Logic([j-1, j, j+1]) = 0; % 忽略第 j-1 (之前), j (本身), j
+1(之后) 号板凳
            Logic(224) = 0; % 忽略最后一个节点 (无板凳)
            for i = find(Logic)'    % 这里必须转置使得右值为行向量
                % 新的坐标原点
                new_Origin(1, 1:2) = coor_Rectangles(1, i, :);
                % 依次检查 4 个点
                for k = 1:4
                    new_cor = (point_four(k, :) - new_Origin ) / [
                        coor_Rectangles(2, i, 1) - coor_Rectangles(1, i
, 1), coor_Rectangles(2, i, 2) - coor_Rectangles(1, i, 2);
                        coor_Rectangles(4, i, 1) - coor_Rectangles(1, i
, 1), coor_Rectangles(4, i, 2) - coor_Rectangles(1, i, 2);
                        ];   % 坐标系转换
                    if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&
new_cor(2)<1)
                        logic = true;
                        %disp(['j = ',num2str(j), ': i = ',num2str(i)
,', k = ',num2str(k)])
```

```
                        return
                    end
                end
            end
        end

    % 检查通过
    logic = false;
    return
end



function DrawPointsAndRectangles_Q4(coor_all, coor_Rectangles)
% 作线和点
    DrawPoints_Q4(coor_all);
% 作方框
    coor_Rectangles(5, :, :) = coor_Rectangles(1, :, :);    % plot 围成闭合曲线
    hold on
    for i = 1:223
        plot(coor_Rectangles(:, i, 1), coor_Rectangles(:, i, 2), 'LineWidth', 0.3, 'Color', [1 0 1]);
    end
% 收尾
    hold off
end

function DrawWholePath
global theta_t b lambda P_C P_A R_t
    % 生成全路径曲线
    theta_array = theta_t:0.1:32*pi;
    beta_array = 0:0.05:lambda;
    coor_1_array = b*theta_array'.*[cos(theta_array'), sin(theta_array')];
    coor_4_array = - coor_1_array;
    coor_2_array = zeros(length(beta_array), 2);
    coor_3_array = zeros(length(beta_array), 2);
    for i = 1:length(beta_array)
        coor_2_array(i, :) = P_C(beta_array(i));
        coor_3_array(i, :)  = P_A(beta_array(i));
    end
```

```
449     % 生成调头区域
450     coor_5 = 4.5*[cos(0:0.02:2*pi); sin(0:0.02:2*pi)]'; % 给定调头区域
451     coor_6 = R_t*[cos(0:0.02:2*pi); sin(0:0.02:2*pi)]'; % 实际调头区域
452
453     % 作图
454         figure('Color', [1 1 1])
455         % 作出全路径曲线
456         stc.line1 = plot(coor_1_array(:, 1), coor_1_array(:, 2));
457         hold on
458         stc.line23 = plot([coor_2_array(:, 1); coor_3_array(:, 1)], [
    coor_2_array(:, 2); coor_3_array(:, 2)]);
459         stc.line4 = plot(coor_4_array(:, 1), coor_4_array(:, 2));
460         stc.line5 = plot(coor_5(:, 1), coor_5(:, 2), 'black--');
461         stc.line6 = plot(coor_6(:, 1), coor_6(:, 2), 'r--');
462
463     % 设置样式
464         % 坐标轴
465             stc.fig = gcf;
466             axis equal
467             stc.axes = gca;
468             stc.axes.FontName = "Times New Roman"; % 全局 FontName
469             stc.axes.Box = 'on';
470             stc.axes.FontSize = 14;
471             xline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
472             yline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
473             stc.label.x = xlabel(stc.axes, '$x$', 'Interpreter', 'latex
    ', 'FontSize', 15);
474             stc.label.y = ylabel(stc.axes, '$y$', 'Interpreter', 'latex
    ', 'FontSize', 15);
475
476
477         % 标题
478             stc.axes.Title.String = '';
479             stc.axes.Title.FontSize = 17;
480             stc.axes.Title.FontWeight = 'bold';
481
482         % 线的样式
483             stc.line1.LineWidth = 0.8;
484             stc.line23.LineWidth = 0.8;
485             stc.line4.LineWidth = 0.8;
486             stc.line5.LineWidth = 0.6;
487             stc.line6.LineWidth = 0.6;
```

```
            stc.line1.Color = [0 1 0];    % 绿色
            stc.line23.Color = [1 0 0];    % 红色
            stc.line4.Color = [0 0 1];    % 蓝色

    % 收尾
        hold off
end


function Rectangle_Points = GetRectangles_Q4(coor_all)
    Vec_X = diff(coor_all);
    Vec_X = Vec_X ./ sqrt(sum(Vec_X.^2, 2));      % 单位化
    Vec_N = [ -Vec_X(: ,2), Vec_X(: ,1) ];  % 法向量


    % 计算矩形坐标
    Rectangle_P1 = coor_all(1:223, :) - Vec_X*0.275 + Vec_N*0.15;     % 注意是 1:223
    Rectangle_P2 = coor_all(1:223, :) - Vec_X*0.275 - Vec_N*0.15;
    Rectangle_P3 = coor_all(2:224, :) + Vec_X*0.275 - Vec_N*0.15;     % 注意是 2:224
    Rectangle_P4 = coor_all(2:224, :) + Vec_X*0.275 + Vec_N*0.15;

    Rectangle_Points = zeros(4, 223, 2);
    Rectangle_Points(1, :, :) = Rectangle_P1;
    Rectangle_Points(2, :, :) = Rectangle_P2;
    Rectangle_Points(3, :, :) = Rectangle_P3;
    Rectangle_Points(4, :, :) = Rectangle_P4;
end


function PrintResult_Q4(coor_matrix, speed_matrix)
    % coor_matrix: (224*2)*(201) 矩阵
    % speed_matrix: 224*(201) 矩阵

    % 遍历矩阵，将每个元素格式化为保留 6 位小数的字符串
    coor_xlsx = cell(size(coor_matrix));
    speed_xlsx = cell(size(speed_matrix));
    for i = 1:size(coor_matrix, 1)
        for j = 1:size(coor_matrix, 2)
            coor_xlsx{i, j} = num2str(coor_matrix(i, j), '%.6f');
        end
```

```
    end
    for i = 1:size(speed_matrix, 1)
        for j = 1:size(speed_matrix, 2)
            speed_xlsx{i, j} = num2str(speed_matrix(i, j), '%.6f');
        end
    end
    writecell(coor_xlsx, 'Q4_Result.xlsx', 'Sheet', 'Sheet1'); % 输出位置
    writecell(speed_xlsx, 'Q4_Result.xlsx', 'Sheet', 'Sheet2'); % 输出速度
end



function Speed_all = GetAllSpeed_Q4(angle_withflag_all, v_0)
% 引入全局变量
global P_C P_A coor_C coor_A
    Speed_all = zeros(224 ,1);
    Speed_all(1) = v_0;

    % 计算 Vec_X
    Coordinates = Angle2Coor(angle_withflag_all);
    Vec_X = diff(Coordinates);

    % 计算 Vec_Tao
    Vec_Tao = zeros(224, 2);
        % flag 1 上的点
        for i = find(angle_withflag_all(:, 2) == 1)'
            theta = angle_withflag_all(i, 1);
             Vec_Tao(i, :) = - [cos(theta) - theta*sin(theta), sin(theta) + theta*cos(theta) ];
        end
        % flag 2 上的点
        for i = find(angle_withflag_all(:, 2) == 2)'
            vec_r = [P_C(angle_withflag_all(i, 1)) - coor_C, 0];
            vec_tao = cross(vec_r, [0 0 1]);
            Vec_Tao(i, :) = vec_tao(1:2);
        end
        % flag 3 上的点
        for i = find(angle_withflag_all(:, 2) == 3)'
            vec_r = [P_A(angle_withflag_all(i, 1)) - coor_A, 0];
            vec_tao = cross([0 0 1], vec_r);    % [0 0 1] 在前
```

```
567             Vec_Tao(i, :) = vec_tao(1:2);
568         end
569         % flag 4 上的点
570         for i = find(angle_withflag_all(:, 2) == 4)'
571             theta = angle_withflag_all(i, 1);
572             Vec_Tao(i, :) = - [cos(theta) - theta*sin(theta), sin(theta
    ) + theta*cos(theta) ];
573         end
574 
575         Vec_Tao = Vec_Tao ./sqrt(sum( (Vec_Tao).^2 ,2));
576     % 计算速度
577     for i = 1:223
578         % 速度正负
579         mp = (    (sum((Vec_Tao(i, :).*Vec_X(i, :))) * sum((Vec_Tao(i+1,
    :).*Vec_X(i, :))))    >= 0     )*2 + -1;
580         Speed_all(i+1) = mp * Speed_all(i) * ( norm(Vec_Tao(i+1, :))*
    abs(sum(Vec_Tao(i, :).*Vec_X(i, :)))  ) / ( norm(Vec_Tao(i, :))*abs(
    sum(Vec_Tao(i+1, :).*Vec_X(i, :))) );
581     end
582 end
583 
584 
585 function stc = DrawPoints_Q4(coordinates_array)
586 % 引入全局变量
587 global theta_t b lambda P_C P_A R_t
588 
589     % 生成全路径曲线
590     theta_array = theta_t:0.1:32*pi;
591     beta_array = 0:0.005:lambda;
592     coor_1_array = b*theta_array'.*[cos(theta_array'), sin(theta_array
    ')];
593     coor_4_array = - coor_1_array;
594     coor_2_array = zeros(length(beta_array), 2);
595     coor_3_array = zeros(length(beta_array), 2);
596     for i = 1:length(beta_array)
597         coor_2_array(i, :) = P_C(beta_array(i));
598         coor_3_array(i, :)  = P_A(beta_array(i));
599     end
600     % 生成调头区域
601     coor_5 = 4.5*[cos(0:0.02:2*pi); sin(0:0.02:2*pi)]';
602     coor_6 = R_t*[cos(0:0.02:2*pi); sin(0:0.02:2*pi)]'; % 实际调头区域
603 
```

```
604
605     % 作图
606         figure('Color', [1 1 1])
607         % 作出全路径曲线
608         stc.line1 = plot(coor_1_array(:, 1), coor_1_array(:, 2));
609         hold on
610         stc.line23 = plot([coor_2_array(:, 1); coor_3_array(:, 1)], [
    coor_2_array(:, 2); coor_3_array(:, 2)]);
611         stc.line4 = plot(coor_4_array(:, 1), coor_4_array(:, 2));
612         stc.line5 = plot(coor_5(:, 1), coor_5(:, 2), 'black--');
613         stc.line6 = plot(coor_6(:, 1), coor_6(:, 2), 'r--');
614         % 作出所有节点
615         stc.point_0 = scatter(coordinates_array(1, 1),
    coordinates_array(1, 2), 150, '.');
616         stc.point_0.CData = [1 0 1];   % 粉色
617         stc.points_rest = scatter(coordinates_array(2:end, 1),
    coordinates_array(2:end, 2), 50, 'black.');
618
619     % 设置样式
620         % 坐标轴
621             stc.fig = gcf;
622             axis equal
623             stc.axes = gca;
624             stc.axes.FontName = "Times New Roman"; % 全局 FontName
625             stc.axes.Box = 'on';
626             stc.axes.FontSize = 14;
627             xline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
628             yline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
629             stc.label.x = xlabel(stc.axes, '$x$', 'Interpreter', 'latex
    ', 'FontSize', 15);
630             stc.label.y = ylabel(stc.axes, '$y$', 'Interpreter', 'latex
    ', 'FontSize', 15);
631
632
633         % 标题
634             stc.axes.Title.String = '';
635             stc.axes.Title.FontSize = 17;
636             stc.axes.Title.FontWeight = 'bold';
637
638         % 线的样式
639             stc.line1.LineWidth = 0.8;
640             stc.line23.LineWidth = 0.8;
```

```
641             stc.line4.LineWidth = 0.8;
642             stc.line5.LineWidth = 0.6;
643             stc.line1.Color = [0 1 0];   % 绿色
644             stc.line23.Color = [1 0 0];   % 红色
645             stc.line4.Color = [0 0 1];   % 蓝色
646 
647     % 收尾
648         hold off
649 end
650 
651 
652 function coordinates_array = Angle2Coor(angle_withflag_array)
653 % 将 angle_withflag_array (n*2 矩阵, 也即一列向量) 转为 直角坐标
    coordinates (n*2 矩阵, 也即一列坐标向量)
654 % 引入全局变量
655 global P_C P_A b
656     coordinates_array = zeros(size(angle_withflag_array));
657     for i = 1 : size(angle_withflag_array, 1)
658         switch angle_withflag_array(i, 2)
659             case 1
660                 coordinates_array(i, :) = b*angle_withflag_array(i, 1)
    ...
661                                           *[cos(angle_withflag_array(i
    , 1)), sin(angle_withflag_array(i, 1))];
662             case 2
663                 coordinates_array(i, :) = P_C(angle_withflag_array(i,
    1));
664             case 3
665                 coordinates_array(i, :) = P_A(angle_withflag_array(i,
    1));
666             case 4
667                 coordinates_array(i, :) = - b*angle_withflag_array(i,
    1) ...
668                                            *[cos(angle_withflag_array(
    i, 1)), sin(angle_withflag_array(i, 1))];
669         end
670     end
671 end
672 
673 
674 function angle_withflag_all = GetAllPoints_Q4(angle_withflag_0,
    error_max_abs)
```

```
675 % 引入全局变量
676 global d d_0
677     angle_withflag_all = zeros(224, 2);
678     angle_withflag_all(1, :) = angle_withflag_0;
679     % 龙头板凳长不同, 单独计算 第二节点
680     angle_withflag_all(2, :) = GetNextPoint_newton_Q4(
    angle_withflag_all(1, :), d_0, error_max_abs);
681     % 其它节点
682     for i = 3:224
683         angle_withflag_all(i, :) = GetNextPoint_newton_Q4(
    angle_withflag_all(i-1, :), d, error_max_abs);
684     end
685 end
686
687
688 function angel_withflag = GetHeadLocation(v_0, t)
689     % 引入全局变量
690     global lambda R_big R_small b t_turn
691     % 判断龙头 flag 标志位, 并给出对应 angle
692     if t < 0
693         angel_withflag(2) = 1;
694         angel_withflag(1) = 16*2*pi - func_find_start(b, 16*2*pi, v_0*(
    t + t_turn));
695     elseif ( 0 <= t ) && ( t < lambda*R_big/v_0 )
696         angel_withflag(2) = 2;
697         angel_withflag(1) = v_0*t/R_big;
698     elseif ( lambda*R_big/v_0 <= t ) && ( t < lambda*(R_big+R_small)/
    v_0 )
699         angel_withflag(2) = 3;
700         angel_withflag(1) = v_0*(t - lambda*R_big/v_0) / R_small;
701     elseif (t >= lambda*(R_big+R_small)/v_0)
702         angel_withflag(2) = 4;
703         angel_withflag(1) = 16*2*pi - func_find_start(b, 16*2*pi, v_0*(
    t_turn + lambda*(R_big + R_small)/v_0 - t));
704     end
705     return
706 end
707
708
709 function angle_withflag_next = GetNextPoint_newton_Q4(angle_withflag, d
    , error_max_abs)
710     % 引入全局变量
```

```
711     global b func_dist_21 func_dist_22 func_dist_32 func_dist_33
        func_dist_42 func_dist_43 func_dist_44 theta_t lambda
712     % 逻辑判断并求解
713     switch angle_withflag(2)
714         case 1
715             angle_withflag_next = [GetNextPoint_newton(angle_withflag
        (1), b, d, error_max_abs), 1];
716         case 2
717             dist = func_dist_22(angle_withflag(1), 0); % 计算距离
718             if dist > d     % NextPoint 仍在 flag 2, 使用函数
        func_dist_22
719                 % 其中 angle_1 定死, angle_2_range 用于遍历
720                 angle = GetPriciseAngle(angle_withflag(1), [0,
        angle_withflag(1)], d, error_max_abs, func_dist_22);
721                 angle_withflag_next = [angle, 2];
722             else    % NextPoint 在 flag 1, 使用函数 func_dist_21
723                 angle = GetPriciseAngle(angle_withflag(1), [theta_t,
        theta_t + pi], d, error_max_abs, @(x,y) func_dist_21(x,y,d));
724                 angle_withflag_next = [angle, 1];
725             end
726         case 3
727             dist = func_dist_33(angle_withflag(1), 0); % 计算距离
728             if dist > d     % NextPoint 仍在 flag 3, 使用函数
        func_dist_33
729                 % 其中 angle_1 定死, angle_2_range 用于遍历
730                 angle = GetPriciseAngle(angle_withflag(1), [0,
        angle_withflag(1)], d, error_max_abs, func_dist_33);
731                 angle_withflag_next = [angle, 3];
732             else    % NextPoint 在 flag 2, 使用函数 func_dist_32
733                 angle = GetPriciseAngle(angle_withflag(1), [0, lambda],
         d, error_max_abs, func_dist_32);
734                 angle_withflag_next = [angle, 2];
735             end
736         case 4
737             dist = func_dist_44(angle_withflag(1), theta_t); % 计算距
        离
738             if (dist > d) || (angle_withflag(1) > theta_t + pi*2/3)
        % NextPoint 仍在 flag 4, 使用函数 func_dist_44
739                 % 其中 angle_1 定死, angle_2_range 用于遍历
740                 % 这里 angle_range 选取不当会导致结果出错
741                 range_min = angle_withflag(1) - pi/3;
742                 if func_dist_44(angle_withflag(1), range_min) < d     %
```

```
    range_min 太大，还需缩小
                    range_min = range_min - pi/3;
                    if func_dist_44(angle_withflag(1), range_min) < d
                        error("func_dist_44(angle_withflag(1),
    angle_withflag(1) - 2*pi/3) < d, range 传入错误!");
                    end
                end
                angle = GetPriciseAngle(angle_withflag(1), [range_min,
    angle_withflag(1)], d, error_max_abs, func_dist_44);
                angle_withflag_next = [angle, 4];
            else    % NextPoint 在 flag 3 或 flag 2
                if func_dist_43(angle_withflag(1), 0) < d
                    % NextPoint 在 flag 2，使用函数 func_dist_42
                    angle = GetPriciseAngle(angle_withflag(1), [0,
    lambda], d, error_max_abs, func_dist_42);
                    angle_withflag_next = [angle, 2];
                else
                    % NextPoint 在 flag 3，使用函数 func_dist_43
                    angle = GetPriciseAngle(angle_withflag(1), [0,
    lambda], d, error_max_abs, func_dist_43);
                    angle_withflag_next = [angle, 3];
                end
            end
    end
    return
end

function angel = GetPriciseAngle(angel_1, angle_2_range, d,
    error_max_abs, func_dist)
% func_dist(angle_1, angle_2, b)，其中 angle_1 定死，angle_2 用于遍历
% 注：此函数已完成去耦分离
% 函数要求 range 的 min 角度对应距离 > d，max 角度对应距离 < d
    % 第 1 层
    angle_array = linspace(angle_2_range(1), angle_2_range(2), 100);
    for i = length(angle_array):(-1):1
        % 注意这里的判断是 < d，和之前的算法不同
        % 这里要求 range 的 min 角度对应距离 > d，max 角度对应距离 < d
        if func_dist(angel_1, angle_array(i)) > d
            break
        end
    end

```

```
779     % 牛顿迭代法获得高精度解
780
781     % 进行牛顿迭代
782     if i == 1
783         min = angle_2_range(1);
784         max = angle_array(1) + 0.1;
785     else
786         min = angle_array(i);
787         max = angle_array(i+1);
788     end
789
790     for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使
    得 4*pi/2^n < 10^(-20)
791         error_max = max - min;
792         temp = 0.5 * (max + min);
793         temp_dis = func_dist(angel_1, temp);
794         if temp_dis < d
795             max = temp;
796         elseif temp_dis > d
797             min = temp;
798         elseif temp_dis == d
799             angel = temp;
800             return
801         end
802         if error_max <= error_max_abs
803             break
804         end
805     end
806
807     % 输出结果
808     angel = 0.5*(min+max);
809 end
810
811 %% 问题一函数区 %%
812 %%%%%%%%%%%%%%%%%%
813
814 function theta = GetNextPoint_newton( theta_0 , b, d , error_max_abs)
815     % 第 1 层
816     theta_range = theta_0:10^(-1):(theta_0 + 10*pi);
817     for i = 2:length(theta_range)
818         if GetDistance(theta_range(i), theta_0, b) > d
819             break
```

```
820         end
821     end
822
823     % 进行牛顿迭代
824     min = theta_range(i-1);
825     max = theta_range(i);
826
827     for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使
    得 4*pi/2^n < 10^(-20)
828         error_max = max - min;
829         temp = 0.5 * (max + min);
830         temp_dis = GetDistance(temp, theta_0, b);
831         if temp_dis < d
832             min = temp;
833         elseif temp_dis > d
834             max = temp;
835         elseif temp_dis == d
836             theta = temp;
837             return
838         end
839         if error_max <= error_max_abs
840             break
841         end
842     end
843
844     % 输出结果
845     theta = 0.5*(min + max);
846 end
847
848
849 function dis = GetDistance(theta, theta_0, b)
850     dis = sqrt( b^2*(theta.^2 + theta_0.^2) -2*b^2.*theta.*theta_0.*cos
    (theta-theta_0) );
851 end
852
853
854 function theta_final = func_find_start(b, theta0, L)
855     % 初始化参数
856     % b = 55 / (2 * pi);     % 螺线递增参数 (b)
857     % theta0 = 16 * 2 * pi; % 初始点的极坐标角度 (θ_0)
858     % L = 500;              % 沿螺线走的距离 (L)
859
```

```
    % 定义螺旋线方程函数
    r = @(theta) b * (theta0-theta);        % 极径方程：r(θ) = b(θ_0-θ)
    ds = @(theta) sqrt(b^2 + r(theta).^2); % 曲线长度微元
    r_real = @(theta) b * theta;        % 极径方程：r(θ) = b(θ_0-θ)
    % 使用Numerical Integration计算θ值
    theta_final = fzero(@(theta) integral(ds, 0, theta) - L, theta0 +
    2*pi);

end
```

B.5 问题 5 代码

```
clc, clear, close all
%% 主程序代码 %%
%%%%%%%%%%%%%%%%

% 根据此调头半径计算所需参数
    % 给定 R_turn_min
    R_t_min = 4.254674;
    global R_t; R_t = ceil(R_t_min*10^(8))/10^(8);

% 由题目直接确定的参数
global d d_0 luoju b v_0 theta_max
    d = 1.65;    % 普通节点间距
    d_0 = 2.86;    % 龙头节点间距
    luoju = 1.7; % 螺距
    a = 0; % 阿基米德螺线的起始半径
    b = luoju / (2*pi); % 每圈的增长量 (与螺距有关)
    theta_max = 16 * 2 * pi;       % 初始点极坐标角度
    v_0 = 1;

% 依次求解全部所需全局变量
    global theta_t coor_M coor_H coor_E dist_HE l_M l_H t_turn ...
           l_HA coor_A coor_C R_small R_big lambda coor_HA kappa
        theta_t = R_t/b;
        func = @(x)b*sqrt(x.^2+1);
        t_turn = 1/v_0*integral(func, theta_t , 32*pi); % 从进入开始计
    时，到恰好开始转弯需要的时间
        coor_M = b*theta_t*[cos(theta_t), sin(theta_t)];
        coor_H = -coor_M;
        coor_E = coor_H/3;
        dist_HE = R_t/3;
```

```
30      l_M = - [cos(theta_t) - theta_t.*sin(theta_t),  sin(theta_t) +
    theta_t.*cos(theta_t)];
31      l_H = - l_M;
32      l_HA = - cross([0,0,1], [l_H, 0]); l_HA = l_HA(1:2); l_HA =
    l_HA/norm(l_HA);     % cross 前补了个负号
33      func = @(t) norm(coor_E - (coor_H + t*l_HA)) - norm(coor_H - (
    coor_H + t*l_HA));
34      t = fzero(func, 0);
35      coor_A = coor_H + t*l_HA;
36      coor_C = 3*coor_E - 2*coor_A;
37      R_small = norm(coor_A - coor_H);
38      R_big = 2*R_small;
39      lambda = acos(1 - norm(coor_H-coor_E)^2/(2*R_small^2));
40      coor_HA = coor_A - coor_H;
41      kappa = pi - atan(coor_HA(2)/coor_HA(1));
42     % 注意 kappa 的范围, 需要作判断 if(-coor_HA(1) < 0){ kappa -->
    kappa - pi}以纠正
43       if ( - coor_HA(1) < 0)
44           kappa = kappa - pi;
45       end
46
47  % 构建全局函数
48   global P_C P_A
49      % 圆 C 的参数方程 (返回直角坐标)
50      P_C = @(beta) ( coor_C + R_big *cos(beta + kappa).*[1 0] -
    R_big *sin(beta + kappa).*[0 1] );
51      % 圆 A 的参数方程 (返回直角坐标)
52      P_A = @(beta) ( coor_A - R_small *cos((-beta) + kappa + lambda)
    .*[1 0] + R_small *sin((-beta)  + kappa + lambda ).*[0 1]);
53
54  % 用于计算距离的全局函数族
55   global func_dist_21 func_dist_22 func_dist_32 func_dist_33
    func_dist_42 func_dist_43 func_dist_44
56      func_dist_21 = @(beta, theta, d) ... % 转直角后用距离公式, 并作
    映射 dist --> 2*d - dist 以满足 min 时 > d
57          2*d - norm(P_C(beta) - b*theta.*[cos(theta), sin(theta)]);
58      func_dist_22 = @(beta_1, beta_2) R_big * sqrt( 2*(1-cos(beta_1
    - beta_2)) ); % 半径是 R_big
59      func_dist_32 = @(beta_1, beta_2) norm(P_A(beta_1) - P_C(beta_2)
    ); % 注意 beta_1 是 P_A
60      func_dist_33 = @(beta_1, beta_2) R_small * sqrt( 2*(1-cos(
    beta_1 - beta_2)) ); % 半径是 R_small
```

```
61          func_dist_42 = @(theta, beta) ...    % 这里给的 theta 转直角后再
    对称 (取负) 才是真实坐标
62              norm( - b*theta.*[cos(theta), sin(theta)] - P_C(beta));
63          func_dist_43 = @(theta, beta) ...    % 这里给的 theta 转直角后再
    对称才是真实坐标
64              norm( - b*theta.*[cos(theta), sin(theta)] - P_A(beta));
65          func_dist_44 = @(theta_1, theta_2) ...
66              norm( b*theta_1.*[cos(theta_1), sin(theta_1)] - b*theta_2
    .*[cos(theta_2), sin(theta_2)]);
67
68  % 检查 R_turn_min 是否可行
69      [logic_crush, logic_speederror, t_crushing] = CheckCrushAndSpeed_Q4
    ([0, 50], 0.1);
70      if (logic_crush == true) || (logic_speederror == true)
71          error(">> error:  当前 R_t 发生碰撞或无法通过, 须重新设置 R_t
    值")
72      else
73          disp("调头半径检验通过!")
74      end
75
76  % 求解最大速度
77  v_0_range = [0.1, 2];
78  v0_error_max = 10^(-10);
79  v0_times_t_step = 0.05;  % 用于确定 t_range
80  v_0_max = GetMax_V0_Q5(v_0_range, v0_error_max, v0_times_t_step);
81
82  disp(['v_0_max = ', num2str(v_0_max, '%.12f')])
83
84  %% 问题五函数区 %%
85  %%%%%%%%%%%%%%%%%%
86
87  function v_0_max = GetMax_V0_Q5(v_0_range, v0_error_max,
    v0_times_t_step)
88      % 第一层步长: R_step1
89      global v_0
90      v_0_array = linspace(v_0_range(1), v_0_range(2), 6);
91          for i = 1:length(v_0_array)
92              v_0_new = v_0_array(i);
93              t_range = [0, 40/v_0_new];
94              t_step = v0_times_t_step/v_0_new;
95              if t_step > 0.5
96                  t_step = 0.5;
```

```
            end
            v_max = GetVMax(v_0_new, t_range, t_step);
            disp(['v_max = ', num2str(v_max), ', v_0 = ', num2str(v_0)
]);
            if v_max > 2
                v_0_range = [v_0_array(i-1), v_0_array(i)];
                break
            end
        end


    % 第一层步长: R_step1
    v_0_array = linspace(v_0_range(1), v_0_range(2), 10);
        for i = 1:length(v_0_array)
            v_0_new = v_0_array(i);
            t_range = [0, 40/v_0_new];
            t_step = v0_times_t_step/v_0_new;
            if t_step > 0.5
                t_step = 0.5;
            end
            v_max = GetVMax(v_0_new, t_range, t_step);
            disp(['v_max = ', num2str(v_max), ', v_0 = ', num2str(v_0)
]);
            if v_max > 2
                v_0_range = [v_0_array(i-1), v_0_array(i)];
                break
            end
        end

    % 牛顿迭代法
    min = v_0_range(1);
    max = v_0_range(2);
    for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使
得 4*pi/2^n < 10^(-20)
        error_max = max - min;
        temp = 0.5 * (max + min);
        t_range = [0, 40/temp];
        t_step = v0_times_t_step/temp;
        v_max = GetVMax(temp, t_range, t_step);
        if v_max < 2
            min = temp;
        else
```

```
136             max = temp;
137         end
138         disp(['v_max = ', num2str(v_max), ', v_0 = ', num2str(v_0) ]);
139         if error_max <= v0_error_max
140             break
141         end
142     end
143 
144     % 输出结果
145     v_0_max = 0.5*(min+max);
146     disp(['v_max = ', num2str(v_max, '%.12f'), ', v_0 = ', num2str(v_0,
        '%.12f') ]);
147 end
148 
149 
150 function v_max = GetVMax(v_0_now, t_range, t_step)
151     global v_0; v_0 = v_0_now;
152     ChangeGlobalVaribles
153     t_array = t_range(1):t_step:t_range(2);
154     speed_matrix = zeros(224, length(t_array));
155     for i = 1:length(t_array)
156         t = t_array(i);
157         angel_withflag_0 = GetHeadLocation(v_0, t);
158         angle_withflag_all = GetAllPoints_Q4(angel_withflag_0, 10^(-12)
        );
159         speed_matrix(:, i) = GetAllSpeed_Q4(angle_withflag_all, v_0);
160     end
161     v_max = max(speed_matrix, [], "all");
162 end
163 
164 
165 function ChangeGlobalVaribles
166 % 引入全局变量
167 global R_t d d_0 luoju b v_0 theta_max
168 % 依次求解全部所需全局变量
169     global theta_t coor_M coor_H coor_E dist_HE l_M l_H t_turn ...
170             l_HA coor_A coor_C R_small R_big lambda coor_HA kappa
171 
172         theta_t = R_t/b;
173         func = @(x)b*sqrt(x.^2+1);
174         t_turn = 1/v_0*integral(func, theta_t , 32*pi); % 从进入开始计
        时, 到恰好开始转弯需要的时间
```

```
175         disp(['t_turn = ', num2str(t_turn)])
176         coor_M = b*theta_t*[cos(theta_t), sin(theta_t)];
177         coor_H = -coor_M;
178         coor_E = coor_H/3;
179         dist_HE = R_t/3;
180         l_M = - [cos(theta_t) - theta_t.*sin(theta_t),  sin(theta_t) +
    theta_t.*cos(theta_t)];
181         l_H = - l_M;
182         l_HA = - cross([0,0,1], [l_H, 0]); l_HA = l_HA(1:2); l_HA =
    l_HA/norm(l_HA);     % cross 前补了个负号
183         func = @(t) norm(coor_E - (coor_H + t*l_HA)) - norm(coor_H - (
    coor_H + t*l_HA));
184         t = fzero(func, 0);
185         coor_A = coor_H + t*l_HA;
186         coor_C = 3*coor_E - 2*coor_A;
187         R_small = norm(coor_A - coor_H);
188         R_big = 2*R_small;
189         lambda = acos(1 - norm(coor_H-coor_E)^2/(2*R_small^2));
190         coor_HA = coor_A - coor_H;
191         kappa = pi - atan(coor_HA(2)/coor_HA(1));
192        % 注意 kappa 的范围, 需要作判断 if(-coor_HA(1) < 0){ kappa -->
    kappa - pi}以纠正
193          if ( - coor_HA(1) < 0)
194              kappa = kappa - pi;
195          end
196
197     % 构建全局函数
198      global P_C P_A
199         % 圆 C 的参数方程 (返回直角坐标)
200         P_C = @(beta) ( coor_C + R_big *cos(beta + kappa).*[1 0] -
    R_big *sin(beta + kappa).*[0 1] );
201         % 圆 A 的参数方程 (返回直角坐标)
202         P_A = @(beta) ( coor_A - R_small *cos((-beta) + kappa + lambda)
    .*[1 0] + R_small *sin((-beta)  + kappa + lambda ).*[0 1]);
203
204     % 用于计算距离的全局函数族
205      global func_dist_21 func_dist_22 func_dist_32 func_dist_33
    func_dist_42 func_dist_43 func_dist_44
206         func_dist_21 = @(beta, theta, d) ... % 转直角后用距离公式, 并作
    映射 dist --> 2*d - dist 以满足 min 时 > d
207             2*d - norm(P_C(beta) - b*theta.*[cos(theta), sin(theta)]);
208         func_dist_22 = @(beta_1, beta_2) R_big * sqrt( 2*(1-cos(beta_1
```

```
    - beta_2)) ); % 半径是 R_big
209         func_dist_32 = @(beta_1, beta_2) norm(P_A(beta_1) - P_C(beta_2)
    ); % 注意 beta_1 是 P_A
210         func_dist_33 = @(beta_1, beta_2) R_small * sqrt( 2*(1-cos(
    beta_1 - beta_2)) ); % 半径是 R_small
211         func_dist_42 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称 (取负) 才是真实坐标
212             norm( - b*theta.*[cos(theta), sin(theta)] - P_C(beta));
213         func_dist_43 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称才是真实坐标
214             norm( - b*theta.*[cos(theta), sin(theta)] - P_A(beta));
215         func_dist_44 = @(theta_1, theta_2) ...
216             norm( b*theta_1.*[cos(theta_1), sin(theta_1)] - b*theta_2
    .*[cos(theta_2), sin(theta_2)]);
217
218 end
219
220 %% 问题四函数区 %%
221 %%%%%%%%%%%%%%%%%%
222
223 function ChangeGlobalVaribles_Q4
224 % 引入全局变量
225 global R_t d d_0 luoju b v_0 theta_max
226 % 依次求解全部所需全局变量
227     global theta_t coor_M coor_H coor_E dist_HE l_M l_H t_turn ...
228             l_HA coor_A coor_C R_small R_big lambda coor_HA kappa
229
230         theta_t = R_t/b;
231         func = @(x)b*sqrt(x.^2+1);
232         t_turn = 1/v_0*integral(func, theta_t , 32*pi); % 从进入开始计
    时, 到恰好开始转弯需要的时间
233         coor_M = b*theta_t*[cos(theta_t), sin(theta_t)];
234         coor_H = -coor_M;
235         coor_E = coor_H/3;
236         dist_HE = R_t/3;
237         l_M = - [cos(theta_t) - theta_t.*sin(theta_t),  sin(theta_t) +
    theta_t.*cos(theta_t)];
238         l_H = - l_M;
239         l_HA = - cross([0,0,1], [l_H, 0]); l_HA = l_HA(1:2); l_HA =
    l_HA/norm(l_HA);    % cross 前补了个负号
240         func = @(t) norm(coor_E - (coor_H + t*l_HA)) - norm(coor_H - (
    coor_H + t*l_HA));
```

```
241         t = fzero(func, 0);
242         coor_A = coor_H + t*l_HA;
243         coor_C = 3*coor_E - 2*coor_A;
244         R_small = norm(coor_A - coor_H);
245         R_big = 2*R_small;
246         lambda = acos(1 - norm(coor_H-coor_E)^2/(2*R_small^2));
247         coor_HA = coor_A - coor_H;
248         kappa = pi - atan(coor_HA(2)/coor_HA(1));
249        % 注意 kappa 的范围, 需要作判断 if(-coor_HA(1) < 0){ kappa -->
    kappa - pi}以纠正
250         if ( - coor_HA(1) < 0)
251             kappa = kappa - pi;
252         end
253
254     % 构建全局函数
255      global P_C P_A
256         % 圆 C 的参数方程 (返回直角坐标)
257         P_C = @(beta) ( coor_C + R_big *cos(beta + kappa).*[1 0] -
    R_big *sin(beta + kappa).*[0 1] );
258         % 圆 A 的参数方程 (返回直角坐标)
259         P_A = @(beta) ( coor_A - R_small *cos((-beta) + kappa + lambda)
    .*[1 0] + R_small *sin((-beta) + kappa + lambda ).*[0 1]);
260
261     % 用于计算距离的全局函数族
262      global func_dist_21 func_dist_22 func_dist_32 func_dist_33
    func_dist_42 func_dist_43 func_dist_44
263         func_dist_21 = @(beta, theta, d) ... % 转直角后用距离公式, 并作
    映射 dist --> 2*d - dist 以满足 min 时 > d
264             2*d - norm(P_C(beta) - b*theta.*[cos(theta), sin(theta)]);
265         func_dist_22 = @(beta_1, beta_2) R_big * sqrt( 2*(1-cos(beta_1
    - beta_2)) ); % 半径是 R_big
266         func_dist_32 = @(beta_1, beta_2) norm(P_A(beta_1) - P_C(beta_2)
    ); % 注意 beta_1 是 P_A
267         func_dist_33 = @(beta_1, beta_2) R_small * sqrt( 2*(1-cos(
    beta_1 - beta_2)) ); % 半径是 R_small
268         func_dist_42 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称 (取负) 才是真实坐标
269             norm( - b*theta.*[cos(theta), sin(theta)] - P_C(beta));
270         func_dist_43 = @(theta, beta) ...   % 这里给的 theta 转直角后再
    对称才是真实坐标
271             norm( - b*theta.*[cos(theta), sin(theta)] - P_A(beta));
272         func_dist_44 = @(theta_1, theta_2) ...
```

```
273             norm( b*theta_1.*[cos(theta_1), sin(theta_1)] - b*theta_2
    .*[cos(theta_2), sin(theta_2)]);
274 end
275
276
277 function [R_t_min, t_crushing] = GetMinRadius(t_step, R_range_init,
    R_step1, R_step2, error_max_abs)
278 global R_t v_0 b
279     disp('函数 GetMinRadius 运行时会修改几乎所有全局变量')
280     disp('需在运行后对全局变量重新赋值!')
281     R_range = R_range_init
282
283     start = tic;
284
285     % 第一层步长: R_step1
286     R_array = R_range(1):R_step1:R_range(2);
287         for i = length(R_array):(-1):1
288             R_t = R_array(i);
289             %func = @(x)b*sqrt(x.^2+1);
290             %t_turn_now = 1/v_0*integral(func, R_t/b , 32*pi); % 从进入
    开始计时, 到恰好开始转弯需要的时间
291             %[logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4([0, t_turn_now/3], t_step);
292             [logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4([0, 40], t_step);
293             disp(['crush = ', num2str(logic_crush), ', speederror = ',
    num2str(logic_speederror), ', R = ', num2str(R_t), ', t_crushing = '
    , num2str(t_crushing)]);
294             if (logic_crush == true) || (logic_speederror == true)
295                 R_range = [R_array(i), R_array(i+1)]
296                 break
297             end
298         end
299
300     % 第二层步长: R_step2
301     R_array = R_range(1):R_step2:R_range(2);
302         for i = length(R_array):(-1):1
303             R_t = R_array(i);
304             %func = @(x)b*sqrt(x.^2+1);
305             %t_turn_now = 1/v_0*integral(func, R_t/b , 32*pi); % 从进入
    开始计时, 到恰好开始转弯需要的时间
306             %[logic_crush, logic_speederror, t_crushing] =
```

```
    CheckCrushAndSpeed_Q4([0, t_turn_now/3], t_step);
307         [logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4([0, 40], t_step);
308         disp(['crush = ', num2str(logic_crush), ', speederror = ',
    num2str(logic_speederror), ', R = ', num2str(R_t), ', t_crushing = '
    , num2str(t_crushing)]);
309         if (logic_crush == true) || (logic_speederror == true)
310             R_range = [R_array(i), R_array(i+1)]
311             break
312         end
313     end
314
315 % 牛顿迭代法
316  min = R_range(1);
317  max = R_range(2);
318  for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使
    得 4*pi/2^n < 10^(-20)
319     error_max = max - min;
320     temp = 0.5 * (max + min);
321     R_t = temp;
322     %func = @(x)b*sqrt(x.^2+1);
323     %t_turn_now = 1/v_0*integral(func, R_t/b , 32*pi); % 从进入开始
    计时, 到恰好开始转弯需要的时间
324     %[logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4([0, t_turn_now/3], t_step);
325     [temp_logic_crush, temp_logic_speederror, temp_t] =
    CheckCrushAndSpeed_Q4([0, 40], t_step);
326     if (temp_logic_crush == true) || (temp_logic_speederror == true
    )
327         min = temp
328         t_crushing = temp_t;
329     else
330         max = temp
331     end
332     disp(['crush = ', num2str(temp_logic_crush), ', speederror = ',
     num2str(temp_logic_speederror), ', R = ', num2str(R_t), ',
    t_crushing = ', num2str(t_crushing)]);
333     if error_max <= error_max_abs
334         break
335     end
336  end
337
```

```
338     % 记录结果
339     R_t_min = 0.5*(min + max);
340     time = toc(start);
341     output_cell{1, 1} = num2str(time, '%.6f');
342     output_cell{1, 2} = num2str(R_t_min, '%.12f');
343     output_cell{1, 3} = num2str(t_crushing, '%.12f');
344     output_cell{1, 4} = num2str(t_step, '%.6f');
345     output_cell{1, 4} = num2str(R_range_init, '%.6f');
346     output_cell{1, 5} = num2str(R_step1, '%.6f');
347     output_cell{1, 6} = num2str(R_step2, '%.6f');
348     output_cell{1, 7} = num2str(error_max_abs, '%.20f');
349     disp(['time = ', num2str(time, '%.6f')])
350     disp(['R_t_min = ', num2str(R_t_min, '%.12f')])
351     disp(['t_crushing = ', num2str(t_crushing, '%.6f')])
352     disp(['t_step = ', num2str(t_step, '%.4f')])
353     disp(['R_range_init = [', num2str(R_range_init(1), '%.3f'), ' ,',
    num2str(R_range_init(2), '%.3f'), ']'])
354     disp(['R_step1 = ', num2str(R_step1, '%.4f')])
355     disp(['R_step2 = ', num2str(R_step2, '%.4f')])
356     disp(['R_error_max_abs = ', num2str(error_max_abs, '%.16f')])
357     writecell(output_cell, 'MinRadius_Result.xlsx', 'WriteMode', '
    append');
358 end
359
360 function [logic_crush, logic_speederror, t_crushing] =
    CheckCrushAndSpeed_Q4(t_range, t_step)
361
362     % 外层函数修改了 R_turn
363     global R_t;
364
365     % 题目确定的参数
366     % 参数设置
367     global d d_0 luoju b v_0 theta_max
368         d = 1.65;     % 普通节点间距
369         d_0 = 2.86;     % 龙头节点间距
370         luoju = 1.7; % 螺距
371         a = 0; % 阿基米德螺线的起始半径
372         b = luoju / (2*pi); % 每圈的增长量 (与螺距有关)
373         theta_max = 16 * 2 * pi;        % 初始点极坐标角度
374         v_0 = 1;
375
376     % 依次求解全部所需全局变量
```

```
377     global theta_t coor_M coor_H coor_E dist_HE l_M l_H t_turn ...
378            l_HA coor_A coor_C R_small R_big lambda coor_HA kappa
379         theta_t = R_t/b;
380         func = @(x)b*sqrt(x.^2+1);
381         t_turn = 1/v_0*integral(func, theta_t , 32*pi); % 从进入开始计
    时, 到恰好开始转弯需要的时间
382         coor_M = b*theta_t*[cos(theta_t), sin(theta_t)];
383         coor_H = -coor_M;
384         coor_E = coor_H/3;
385         dist_HE = R_t/3;
386         l_M = - [cos(theta_t) - theta_t.*sin(theta_t),  sin(theta_t) +
    theta_t.*cos(theta_t)];
387         l_H = - l_M;
388         l_HA = - cross([0,0,1], [l_H, 0]); l_HA = l_HA(1:2); l_HA =
    l_HA/norm(l_HA);    % cross 前补了个负号
389         func = @(t) norm(coor_E - (coor_H + t*l_HA)) - norm(coor_H - (
    coor_H + t*l_HA));
390         h = fzero(func, 0);
391         coor_A = coor_H + h*l_HA;
392         coor_C = 3*coor_E - 2*coor_A;
393         R_small = norm(coor_A - coor_H);
394         R_big = 2*R_small;
395         lambda = acos(1 - norm(coor_H-coor_E)^2/(2*R_small^2));
396         coor_HA = coor_A - coor_H;
397         kappa = pi - atan(coor_HA(2)/coor_HA(1));
398        % 注意 kappa 的范围, 需要作判断 if(-coor_HA(1) < 0){ kappa -->
    kappa - pi}以纠正
399         if ( - coor_HA(1) < 0)
400             kappa = kappa - pi;
401         end
402
403     % 构建全局函数
404     global P_C P_A
405        % 圆 C 的参数方程 (返回直角坐标)
406         P_C = @(beta) ( coor_C + R_big *cos(beta + kappa).*[1 0] -
    R_big *sin(beta + kappa).*[0 1] );
407        % 圆 A 的参数方程 (返回直角坐标)
408         P_A = @(beta) ( coor_A - R_small *cos((-beta) + kappa + lambda)
    .*[1 0] + R_small *sin((-beta)  + kappa + lambda ).*[0 1]);
409
410     % 用于计算距离的全局函数族
411     global func_dist_21 func_dist_22 func_dist_32 func_dist_33
```

```
func_dist_42 func_dist_43 func_dist_44
412         func_dist_21 = @(beta, theta, d) ... % 转直角后用距离公式, 并作映射 dist --> 2*d - dist 以满足 min 时 > d
413             2*d - norm(P_C(beta) - b*theta.*[cos(theta), sin(theta)]);
414         func_dist_22 = @(beta_1, beta_2) R_big * sqrt( 2*(1-cos(beta_1 - beta_2)) ); % 半径是 R_big
415         func_dist_32 = @(beta_1, beta_2) norm(P_A(beta_1) - P_C(beta_2)); % 注意 beta_1 是 P_A
416         func_dist_33 = @(beta_1, beta_2) R_small * sqrt( 2*(1-cos(beta_1 - beta_2)) ); % 半径是 R_small
417         func_dist_42 = @(theta, beta) ...   % 这里给的 theta 转直角后再对称 (取负) 才是真实坐标
418             norm( - b*theta.*[cos(theta), sin(theta)] - P_C(beta));
419         func_dist_43 = @(theta, beta) ...   % 这里给的 theta 转直角后再对称才是真实坐标
420             norm( - b*theta.*[cos(theta), sin(theta)] - P_A(beta));
421         func_dist_44 = @(theta_1, theta_2) ...
422             norm( b*theta_1.*[cos(theta_1), sin(theta_1)] - b*theta_2.*[cos(theta_2), sin(theta_2)]);
423
424
425     for t = t_range(1):t_step:t_range(2)
426         angle_withflag_0 = GetHeadLocation(v_0, t);
427         angle_withflag_all = GetAllPoints_Q4(angle_withflag_0, 10^(-16));
428         coor_all = Angle2Coor(angle_withflag_all);
429         logic_crush = IsCrushed_Q4(coor_all);
430         if logic_crush == true
431             t_crushing = t;
432             logic_speederror = ~isempty(GetAllSpeed_Q4(angle_withflag_all, v_0)<0);
433             return
434         end
435         speed_all = GetAllSpeed_Q4(angle_withflag_all, v_0);
436         if ~isempty(find(speed_all<0, 1))
437             t_crushing = t;
438             logic_speederror = true;
439             return
440         end
441     end
442
443     % no crush or speed error
```

```
444         logic_speederror = false;
445         t_crushing = -1;
446         return
447     end
448
449     function logic = IsCrushed_Q4(coor_all)
450         % 数据准备
451         d_special = sqrt(3.135^2 + 0.15^2) + sqrt(0.275^2 + 0.15^2);  % 龙
        头 3 号点最大判断间距
452         d_common = sqrt(1.925^2 + 0.15^2) + sqrt(0.275^2 + 0.15^2);  % 2, 3
         号点最大判断间距
453         coor_Rectangles = GetRectangles_Q4(coor_all);
454
455         % 与问题二不同，这里需要判断全部方框的四个点是否 legal
456
457         % 判断龙头 (第一个板凳)
458             point_four(1:4, 1:2) = coor_Rectangles(1:4, 1, :);
459
460             % 挑选可能的节点
461             distance_array = sqrt( sum( ( coor_all - coor_all(1,:) ).^2 ,
        2) );
462             Logic = (distance_array < d_special) ;
463             Logic(1:2) = 0; % 忽略第 1(本身),2(之后)
464             Logic(224) = 0; % 忽略最后一个节点 (无板凳)
465             for i = find(Logic)'    % 这里必须转置使得右值为行向量
466                 % 新的坐标原点
467                 new_Origin(1, 1:2) = coor_Rectangles(1, i, :);
468                 % 依次检查 4 个点
469                 for k = 1:4
470                     new_cor = (point_four(k, :) - new_Origin ) / [
471                         coor_Rectangles(2, i, 1) - coor_Rectangles(1, i, 1)
        , coor_Rectangles(2, i, 2) - coor_Rectangles(1, i, 2);
472                         coor_Rectangles(4, i, 1) - coor_Rectangles(1, i, 1)
        , coor_Rectangles(4, i, 2) - coor_Rectangles(1, i, 2);
473                         ];   % 坐标系转换
474                     %disp(['龙头2号点 new_cor:', num2str(new_cor)])
475                     if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&
        new_cor(2)<1)
476                         logic = true;
477                         %disp(['龙头: i = ',num2str(i),', k = ',num2str(k)
        ])
478                         return
```

```
                end
            end
        end

    % 判断其它板凳
        for j = 2:223
            point_four(1:4, 1:2) = coor_Rectangles(1:4, j, :);

            % 挑选可能的节点
            distance_array = sqrt( sum( ( coor_all - coor_all(1,:) ).^2
    , 2) );
            Logic = (distance_array < d_common) ;
            Logic([j-1, j, j+1]) = 0; % 忽略第 j-1 (之前), j (本身), j
    +1(之后) 号板凳
            Logic(224) = 0; % 忽略最后一个节点 (无板凳)
            for i = find(Logic)'    % 这里必须转置使得右值为行向量
                % 新的坐标原点
                new_Origin(1, 1:2) = coor_Rectangles(1, i, :);
                % 依次检查 4 个点
                for k = 1:4
                    new_cor = (point_four(k, :) - new_Origin ) / [
                        coor_Rectangles(2, i, 1) - coor_Rectangles(1, i
    , 1), coor_Rectangles(2, i, 2) - coor_Rectangles(1, i, 2);
                        coor_Rectangles(4, i, 1) - coor_Rectangles(1, i
    , 1), coor_Rectangles(4, i, 2) - coor_Rectangles(1, i, 2);
                        ];   % 坐标系转换
                    if (0<new_cor(1)&&new_cor(1)<1) && (0<new_cor(2)&&
    new_cor(2)<1)
                        logic = true;
                        %disp(['j = ',num2str(j), ': i = ',num2str(i)
    ,', k = ',num2str(k)])
                        return
                    end
                end
            end
        end

    % 检查通过
    logic = false;
    return
end

```

```
515
516
517 function DrawPointsAndRectangles_Q4(coor_all, coor_Rectangles)
518 % 作线和点
519     DrawPoints_Q4(coor_all);
520 % 作方框
521     coor_Rectangles(5, :, :) = coor_Rectangles(1, :, :);    % plot 围成
    闭合曲线
522     hold on
523     for i = 1:223
524         plot(coor_Rectangles(:, i, 1), coor_Rectangles(:, i, 2), '
    LineWidth', 0.3, 'Color', [1 0 1]);
525     end
526 % 收尾
527     hold off
528 end
529
530 function DrawWholePath
531 global theta_t b lambda P_C P_A R_t
532     % 生成全路径曲线
533     theta_array = theta_t:0.1:32*pi;
534     beta_array = 0:0.05:lambda;
535     coor_1_array = b*theta_array'.*[cos(theta_array'), sin(theta_array
    ')];
536     coor_4_array = - coor_1_array;
537     coor_2_array = zeros(length(beta_array), 2);
538     coor_3_array = zeros(length(beta_array), 2);
539     for i = 1:length(beta_array)
540         coor_2_array(i, :) = P_C(beta_array(i));
541         coor_3_array(i, :)  = P_A(beta_array(i));
542     end
543     % 生成调头区域
544     coor_5 = 4.5*[cos(0:0.02:2*pi); sin(0:0.02:2*pi)]'; % 给定调头区域
545     coor_6 = R_t*[cos(0:0.02:2*pi); sin(0:0.02:2*pi)]'; % 实际调头区域
546
547     % 作图
548         figure('Color', [1 1 1])
549         % 作出全路径曲线
550         stc.line1 = plot(coor_1_array(:, 1), coor_1_array(:, 2));
551         hold on
552         stc.line23 = plot([coor_2_array(:, 1); coor_3_array(:, 1)], [
    coor_2_array(:, 2); coor_3_array(:, 2)]);
```

```
553         stc.line4 = plot(coor_4_array(:, 1), coor_4_array(:, 2));
554         stc.line5 = plot(coor_5(:, 1), coor_5(:, 2), 'black--');
555         stc.line6 = plot(coor_6(:, 1), coor_6(:, 2), 'r--');
556 
557     % 设置样式
558         % 坐标轴
559             stc.fig = gcf;
560             axis equal
561             stc.axes = gca;
562             stc.axes.FontName = "Times New Roman"; % 全局 FontName
563             stc.axes.Box = 'on';
564             stc.axes.FontSize = 14;
565             xline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
566             yline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
567             stc.label.x = xlabel(stc.axes, '$x$', 'Interpreter', 'latex
    ', 'FontSize', 15);
568             stc.label.y = ylabel(stc.axes, '$y$', 'Interpreter', 'latex
    ', 'FontSize', 15);
569 
570 
571         % 标题
572             stc.axes.Title.String = '';
573             stc.axes.Title.FontSize = 17;
574             stc.axes.Title.FontWeight = 'bold';
575 
576         % 线的样式
577             stc.line1.LineWidth = 0.8;
578             stc.line23.LineWidth = 0.8;
579             stc.line4.LineWidth = 0.8;
580             stc.line5.LineWidth = 0.6;
581             stc.line6.LineWidth = 0.6;
582             stc.line1.Color = [0 1 0];   % 绿色
583             stc.line23.Color = [1 0 0];   % 红色
584             stc.line4.Color = [0 0 1];   % 蓝色
585 
586     % 收尾
587         hold off
588 end
589 
590 
591 function Rectangle_Points = GetRectangles_Q4(coor_all)
592     Vec_X = diff(coor_all);
```

```
    Vec_X = Vec_X ./ sqrt(sum(Vec_X.^2, 2));      % 单位化
    Vec_N = [ -Vec_X(:,2), Vec_X(:,1) ];  % 法向量


    % 计算矩形坐标
    Rectangle_P1 = coor_all(1:223, :) - Vec_X*0.275 + Vec_N*0.15;     % 注意是 1:223
    Rectangle_P2 = coor_all(1:223, :) - Vec_X*0.275 - Vec_N*0.15;
    Rectangle_P3 = coor_all(2:224, :) + Vec_X*0.275 - Vec_N*0.15;     % 注意是 2:224
    Rectangle_P4 = coor_all(2:224, :) + Vec_X*0.275 + Vec_N*0.15;

    Rectangle_Points = zeros(4, 223, 2);
    Rectangle_Points(1, :, :) = Rectangle_P1;
    Rectangle_Points(2, :, :) = Rectangle_P2;
    Rectangle_Points(3, :, :) = Rectangle_P3;
    Rectangle_Points(4, :, :) = Rectangle_P4;
end


function PrintResult_Q4(coor_matrix, speed_matrix)
    % coor_matrix: (224*2)*(201) 矩阵
    % speed_matrix: 224*(201) 矩阵

    % 遍历矩阵，将每个元素格式化为保留 6 位小数的字符串
    coor_xlsx = cell(size(coor_matrix));
    speed_xlsx = cell(size(speed_matrix));
    for i = 1:size(coor_matrix, 1)
        for j = 1:size(coor_matrix, 2)
            coor_xlsx{i, j} = num2str(coor_matrix(i, j), '%.6f');
        end
    end
    for i = 1:size(speed_matrix, 1)
        for j = 1:size(speed_matrix, 2)
            speed_xlsx{i, j} = num2str(speed_matrix(i, j), '%.6f');
        end
    end
    writecell(coor_xlsx, 'Q4_Result.xlsx', 'Sheet', 'Sheet1'); % 输出位置
    writecell(speed_xlsx, 'Q4_Result.xlsx', 'Sheet', 'Sheet2'); % 输出速度
end
```

```
631
632
633
634 function Speed_all = GetAllSpeed_Q4(angle_withflag_all, v_0)
635 % 引入全局变量
636 global P_C P_A coor_C coor_A
637     Speed_all = zeros(224 ,1);
638     Speed_all(1) = v_0;
639
640     % 计算 Vec_X
641     Coordinates = Angle2Coor(angle_withflag_all);
642     Vec_X = diff(Coordinates);
643
644     % 计算 Vec_Tao
645     Vec_Tao = zeros(224, 2);
646         % flag 1 上的点
647         for i = find(angle_withflag_all(:, 2) == 1)'
648             theta = angle_withflag_all(i, 1);
649              Vec_Tao(i, :) = - [cos(theta) - theta*sin(theta), sin(
    theta) + theta*cos(theta) ];
650         end
651         % flag 2 上的点
652         for i = find(angle_withflag_all(:, 2) == 2)'
653             vec_r = [P_C(angle_withflag_all(i, 1)) - coor_C, 0];
654             vec_tao = cross(vec_r, [0 0 1]);
655             Vec_Tao(i, :) = vec_tao(1:2);
656         end
657         % flag 3 上的点
658         for i = find(angle_withflag_all(:, 2) == 3)'
659             vec_r = [P_A(angle_withflag_all(i, 1)) - coor_A, 0];
660             vec_tao = cross([0 0 1], vec_r);    % [0 0 1] 在前
661             Vec_Tao(i, :) = vec_tao(1:2);
662         end
663         % flag 4 上的点
664         for i = find(angle_withflag_all(:, 2) == 4)'
665             theta = angle_withflag_all(i, 1);
666             Vec_Tao(i, :) = - [cos(theta) - theta*sin(theta), sin(theta
    ) + theta*cos(theta) ];
667         end
668
669         Vec_Tao = Vec_Tao ./ sqrt(sum( (Vec_Tao).^2 ,2));
670     % 计算速度
```

```
671     for i = 1:223
672         % 速度正负
673         mp = (    (sum((Vec_Tao(i, :).*Vec_X(i, :))) * sum((Vec_Tao(i+1,
    :).*Vec_X(i, :))))    >= 0     )*2 + -1;
674         Speed_all(i+1) = mp * Speed_all(i) * ( norm(Vec_Tao(i+1, :))*
    abs(sum(Vec_Tao(i, :).*Vec_X(i, :)))  ) / ( norm(Vec_Tao(i, :))*abs(
    sum(Vec_Tao(i+1, :).*Vec_X(i, :))) );
675     end
676 end
677
678
679 function stc = DrawPoints_Q4(coordinates_array)
680 % 引入全局变量
681 global theta_t b lambda P_C P_A R_t
682
683     % 生成全路径曲线
684     theta_array = theta_t:0.1:32*pi;
685     beta_array = 0:0.005:lambda;
686     coor_1_array = b*theta_array'.*[cos(theta_array'), sin(theta_array
    ')];
687     coor_4_array = - coor_1_array;
688     coor_2_array = zeros(length(beta_array), 2);
689     coor_3_array = zeros(length(beta_array), 2);
690     for i = 1:length(beta_array)
691         coor_2_array(i, :) = P_C(beta_array(i));
692         coor_3_array(i, :)  = P_A(beta_array(i));
693     end
694     % 生成调头区域
695     coor_5 = 4.5*[cos(0:0.02:2*pi); sin(0:0.02:2*pi)]';
696     coor_6 = R_t*[cos(0:0.02:2*pi); sin(0:0.02:2*pi)]'; % 实际调头区域
697
698
699     % 作图
700         figure('Color', [1 1 1])
701         % 作出全路径曲线
702         stc.line1 = plot(coor_1_array(:, 1), coor_1_array(:, 2));
703         hold on
704         stc.line23 = plot([coor_2_array(:, 1); coor_3_array(:, 1)], [
    coor_2_array(:, 2); coor_3_array(:, 2)]);
705         stc.line4 = plot(coor_4_array(:, 1), coor_4_array(:, 2));
706         stc.line5 = plot(coor_5(:, 1), coor_5(:, 2), 'black--');
707         stc.line6 = plot(coor_6(:, 1), coor_6(:, 2), 'r--');
```

```
708         % 作出所有节点
709         stc.point_0 = scatter(coordinates_array(1, 1),
    coordinates_array(1, 2), 150, '.');
710         stc.point_0.CData = [1 0 1];   % 粉色
711         stc.points_rest = scatter(coordinates_array(2:end, 1),
    coordinates_array(2:end, 2), 50, 'black.');
712
713     % 设置样式
714         % 坐标轴
715             stc.fig = gcf;
716             axis equal
717             stc.axes = gca;
718             stc.axes.FontName = "Times New Roman"; % 全局 FontName
719             stc.axes.Box = 'on';
720             stc.axes.FontSize = 14;
721             xline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
722             yline(0, 'LineWidth', 0.3, 'Color', [0.7, 0.7, 0.7]);
723             stc.label.x = xlabel(stc.axes, '$x$', 'Interpreter', 'latex
    ', 'FontSize', 15);
724             stc.label.y = ylabel(stc.axes, '$y$', 'Interpreter', 'latex
    ', 'FontSize', 15);
725
726
727         % 标题
728             stc.axes.Title.String = '';
729             stc.axes.Title.FontSize = 17;
730             stc.axes.Title.FontWeight = 'bold';
731
732         % 线的样式
733             stc.line1.LineWidth = 0.8;
734             stc.line23.LineWidth = 0.8;
735             stc.line4.LineWidth = 0.8;
736             stc.line5.LineWidth = 0.6;
737             stc.line1.Color = [0 1 0];   % 绿色
738             stc.line23.Color = [1 0 0];   % 红色
739             stc.line4.Color = [0 0 1];   % 蓝色
740
741     % 收尾
742         hold off
743 end
744
745
```

```
function coordinates_array = Angle2Coor(angle_withflag_array)
% 将 angle_withflag_array (n*2 矩阵, 也即一列向量) 转为 直角坐标
    coordinates (n*2 矩阵, 也即一列坐标向量)
% 引入全局变量
global P_C P_A b
    coordinates_array = zeros(size(angle_withflag_array));
    for i = 1 : size(angle_withflag_array, 1)
        switch angle_withflag_array(i, 2)
            case 1
                coordinates_array(i, :) = b*angle_withflag_array(i, 1)
    ...
                                          *[cos(angle_withflag_array(i
    , 1)), sin(angle_withflag_array(i, 1))];
            case 2
                coordinates_array(i, :) = P_C(angle_withflag_array(i,
    1));
            case 3
                coordinates_array(i, :) = P_A(angle_withflag_array(i,
    1));
            case 4
                coordinates_array(i, :) = - b*angle_withflag_array(i,
    1) ...
                                           *[cos(angle_withflag_array(
    i, 1)), sin(angle_withflag_array(i, 1))];
        end
    end
end


function angle_withflag_all = GetAllPoints_Q4(angle_withflag_0,
    error_max_abs)
% 引入全局变量
global d d_0
    angle_withflag_all = zeros(224, 2);
    angle_withflag_all(1, :) = angle_withflag_0;
    % 龙头板凳长不同, 单独计算 第二节点
    angle_withflag_all(2, :) = GetNextPoint_newton_Q4(
    angle_withflag_all(1, :), d_0, error_max_abs);
    % 其它节点
    for i = 3:224
        angle_withflag_all(i, :) = GetNextPoint_newton_Q4(
    angle_withflag_all(i-1, :), d, error_max_abs);
```

```
778     end
779 end
780
781
782 function angel_withflag = GetHeadLocation(v_0, t)
783     % 引入全局变量
784     global lambda R_big R_small b t_turn
785     % 判断龙头 flag 标志位，并给出对应 angle
786     if t < 0
787         angel_withflag(2) = 1;
788         angel_withflag(1) = 16*2*pi - func_find_start(b, 16*2*pi, v_0*(
    t + t_turn));
789     elseif ( 0 <= t ) && ( t < lambda*R_big/v_0 )
790         angel_withflag(2) = 2;
791         angel_withflag(1) = v_0*t/R_big;
792     elseif ( lambda*R_big/v_0 <= t ) && ( t < lambda*(R_big+R_small)/
    v_0 )
793         angel_withflag(2) = 3;
794         angel_withflag(1) = v_0*(t - lambda*R_big/v_0) / R_small;
795     elseif (t >= lambda*(R_big+R_small)/v_0)
796         angel_withflag(2) = 4;
797         angel_withflag(1) = 16*2*pi - func_find_start(b, 16*2*pi, v_0*(
    t_turn + lambda*(R_big + R_small)/v_0 - t));
798     end
799     return
800 end
801
802
803 function angle_withflag_next = GetNextPoint_newton_Q4(angle_withflag, d
    , error_max_abs)
804     % 引入全局变量
805     global b func_dist_21 func_dist_22 func_dist_32 func_dist_33
    func_dist_42 func_dist_43 func_dist_44 theta_t lambda
806     % 逻辑判断并求解
807     switch angle_withflag(2)
808         case 1
809             angle_withflag_next = [GetNextPoint_newton(angle_withflag
    (1), b, d, error_max_abs), 1];
810         case 2
811             dist = func_dist_22(angle_withflag(1), 0);  % 计算距离
812             if dist > d     % NextPoint 仍在 flag 2，使用函数
    func_dist_22
```

```
813                 % 其中 angle_1 定死, angle_2_range 用于遍历
814                 angle = GetPriciseAngle(angle_withflag(1), [0,
    angle_withflag(1)], d, error_max_abs, func_dist_22);
815                 angle_withflag_next = [angle, 2];
816             else    % NextPoint 在 flag 1, 使用函数 func_dist_21
817                 angle = GetPriciseAngle(angle_withflag(1), [theta_t,
    theta_t + pi], d, error_max_abs, @(x,y) func_dist_21(x,y,d));
818                 angle_withflag_next = [angle, 1];
819             end
820         case 3
821             dist = func_dist_33(angle_withflag(1), 0); % 计算距离
822             if dist > d     % NextPoint 仍在 flag 3, 使用函数
    func_dist_33
823                 % 其中 angle_1 定死, angle_2_range 用于遍历
824                 angle = GetPriciseAngle(angle_withflag(1), [0,
    angle_withflag(1)], d, error_max_abs, func_dist_33);
825                 angle_withflag_next = [angle, 3];
826             else    % NextPoint 在 flag 2, 使用函数 func_dist_32
827                 angle = GetPriciseAngle(angle_withflag(1), [0, lambda],
    d, error_max_abs, func_dist_32);
828                 angle_withflag_next = [angle, 2];
829             end
830         case 4
831             dist = func_dist_44(angle_withflag(1), theta_t); % 计算距
    离
832             if (dist > d) || (angle_withflag(1) > theta_t + pi*2/3)
    % NextPoint 仍在 flag 4, 使用函数 func_dist_44
833                 % 其中 angle_1 定死, angle_2_range 用于遍历
834                 % 这里 angle_range 选取不当会导致结果出错
835                 range_min = angle_withflag(1) - pi/3;
836                 if func_dist_44(angle_withflag(1), range_min) < d     %
    range_min 太大, 还需缩小
837                     range_min = range_min - pi/3;
838                     if func_dist_44(angle_withflag(1), range_min) < d
839                         error("func_dist_44(angle_withflag(1),
    angle_withflag(1) - 2*pi/3) < d, range 传入错误!");
840                     end
841                 end
842                 angle = GetPriciseAngle(angle_withflag(1), [range_min,
    angle_withflag(1)], d, error_max_abs, func_dist_44);
843                 angle_withflag_next = [angle, 4];
844             else    % NextPoint 在 flag 3 或 flag 2
```

```
                if func_dist_43(angle_withflag(1), 0) < d
                    % NextPoint 在 flag 2, 使用函数 func_dist_42
                    angle = GetPriciseAngle(angle_withflag(1), [0,
    lambda], d, error_max_abs, func_dist_42);
                    angle_withflag_next = [angle, 2];
                else
                    % NextPoint 在 flag 3, 使用函数 func_dist_43
                    angle = GetPriciseAngle(angle_withflag(1), [0,
    lambda], d, error_max_abs, func_dist_43);
                    angle_withflag_next = [angle, 3];
                end
            end
    end
    return
end


function angel = GetPriciseAngle(angel_1, angle_2_range, d,
    error_max_abs, func_dist)
% func_dist(angle_1, angle_2, b), 其中 angle_1 定死, angle_2 用于遍历
% 注: 此函数已完成去耦分离
% 函数要求 range 的 min 角度对应距离 > d, max 角度对应距离 < d
    % 第 1 层
    angle_array = linspace(angle_2_range(1), angle_2_range(2), 100);
    for i = length(angle_array):(-1):1
        % 注意这里的判断是 < d, 和之前的算法不同
        % 这里要求 range 的 min 角度对应距离 > d, max 角度对应距离 < d
        if func_dist(angel_1, angle_array(i)) > d
            break
        end
    end

    % 牛顿迭代法获得高精度解

    % 进行牛顿迭代
    if i == 1
        min = angle_2_range(1);
        max = angle_array(1) + 0.1;
    else
        min = angle_array(i);
        max = angle_array(i+1);
    end
```

```
884
885     for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使
        得 4*pi/2^n < 10^(-20)
886         error_max = max - min;
887         temp = 0.5 * (max + min);
888         temp_dis = func_dist(angel_1, temp);
889         if temp_dis < d
890             max = temp;
891         elseif temp_dis > d
892             min = temp;
893         elseif temp_dis == d
894             angel = temp;
895             return
896         end
897         if error_max <= error_max_abs
898             break
899         end
900     end
901
902     % 输出结果
903     angel = 0.5*(min+max);
904 end
905
906 function theta = GetNextPoint_newton( theta_0 , b, d , error_max_abs)
907     % 第 1 层
908     theta_range = theta_0:10^(-1):(theta_0 + 10*pi);
909     for i = 2:length(theta_range)
910         if GetDistance(theta_range(i), theta_0, b) > d
911             break
912         end
913     end
914
915     % 进行牛顿迭代
916     min = theta_range(i-1);
917     max = theta_range(i);
918
919     for i = 1:71      % ceil( (log(4*pi)+20*log(10))/log(2) ) = 71, 这使
        得 4*pi/2^n < 10^(-20)
920         error_max = max - min;
921         temp = 0.5 * (max + min);
922         temp_dis = GetDistance(temp, theta_0, b);
923         if temp_dis < d
```

```
            min = temp;
        elseif temp_dis > d
            max = temp;
        elseif temp_dis == d
            theta = temp;
            return
        end
        if error_max <= error_max_abs
            break
        end
    end

    % 输出结果
    theta = 0.5*(min + max);
end


function dis = GetDistance(theta, theta_0, b)
    dis = sqrt( b^2*(theta.^2 + theta_0.^2) -2*b^2.*theta.*theta_0.*cos
   (theta-theta_0) );
end


function theta_final = func_find_start(b, theta0, L)
    % 初始化参数
    % b = 55 / (2 * pi);     % 螺线递增参数 (b)
    % theta0 = 16 * 2 * pi; % 初始点的极坐标角度 (θ_0)
    % L = 500;               % 沿螺线走的距离 (L)

    % 定义螺旋线方程函数
    r = @(theta) b * (theta0-theta);        % 极径方程: r(θ) = b(θ_0-θ)
    ds = @(theta) sqrt(b^2 + r(theta).^2); % 曲线长度微元
    r_real = @(theta) b * theta;        % 极径方程: r(θ) = b(θ_0-θ)
    % 使用Numerical Integration计算θ值
    theta_final = fzero(@(theta) integral(ds, 0, theta) - L, theta0 +
   2*pi);

end
```