赛区评阅编号（由赛区组委会填写）：

2024 高教社杯全国大学生数学建模竞赛

承　诺　书

我们仔细阅读了《全国大学生数学建模竞赛章程》和《全国大学生数学建模竞赛参赛规则》（以下简称“竞赛章程和参赛规则”，可从 http://www.mcm.edu.cn 下载）。

我们完全清楚，在竞赛开始后参赛队员不能以任何方式，包括电话、电子邮件、“贴吧”、QQ 群、微信群等，与队外的任何人（包括指导教师）交流、讨论与赛题有关的问题；无论主动参与讨论还是被动接收讨论信息都是严重违反竞赛纪律的行为。

我们完全清楚，在竞赛中必须合法合规地使用文献资料和软件工具，不能有任何侵犯知识产权的行为。否则我们将失去评奖资格，并可能受到严肃处理。

我们以中国大学生名誉和诚信郑重承诺，严格遵守竞赛章程和参赛规则，以保证竞赛的公正、公平性。如有违反竞赛章程和参赛规则的行为，我们将受到严肃处理。

我们授权全国大学生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号（从 A/B/C/D/E 中选择一项填写）：A

我们的报名参赛队号（12 位数字全国统一编号）：4321

参赛学校（完整的学校全称，不含院系名）：中山大学

参赛队员 (打印并签名)：1. 张洋

2. 邓俊辉

3. 殷润轩

指导教师或指导教师组负责人 (打印并签名)：

（指导教师签名意味着对参赛队的行为和论文的真实性负责）

日期：2024 年 09 月 22 日

(请勿改动此页内容和格式。此承诺书打印签名后作为纸质论文的封面，注意电子版论文中不得出现此页。以上内容请仔细核对，如填写错误，论文可能被取消评奖资格。)

赛区评阅编号:
（由赛区填写）_____

全国评阅编号:
（全国组委会填写）_____

2024 高教社杯全国大学生数学建模竞赛

编 号 专 用 页

赛区评阅记录（可供赛区评阅时使用）:

评阅人						
备注						

送全国评阅统一编号:
（赛区组委会填写）

（请勿改动此页内容和格式。此编号专用页仅供赛区和全国评阅使用，参赛队打印后装订到纸质论文的第二页上。注意电子版论文中不得出现此页。）

“板凳龙”模型实时仿真模拟

摘要

本文针对“板凳龙”运动问题，结合物理学相关规律，建立一套实时仿真模拟的“板凳龙”物理模型。通过近似圆弧法、叉积法、非线性动态规划，实时仿真模拟等方法进行计算机求解。在给定的不同的螺距、“板凳龙”运动方式、位置、速度等多种情况下，解决了“板凳龙”各节龙身位置、速度与调头转向等多个运动学问题。

对于给定螺距、前把手运动速度、龙头初始位置的各节龙身运动位置速度求解问题，将该运动过程划分为龙头运动过程与各节龙身运动过程，分别进行建模。龙头运动位置，通过建立极坐标系下螺线的轨迹方程、龙头把手极坐标关于时间的解析表达式、将瞬时的螺旋运动等效为圆弧运动等方法，从而计算出龙头的实时位置坐标。再利用速度差分法，将一小段微小时间内平均速度等效为瞬时速度的方法得到各把手实时速度表达式。

对于给定螺距、龙头初始位置的各节板凳碰撞问题，首先通过证明三个断言，给出了简化计算的理论依据。我们将碰撞问题转化为了龙头左前边界点与其后续 18 节板凳矩形位置关系问题，通过叉积法判断矩形与点的位置关系。结合问题 1 的实时仿真模拟，解得龙头在到达螺旋线中心前已和第八节龙身板凳相碰撞，该时刻为 413.36s。

对于逆时针盘出、发生调头情况的龙头运动求解问题，可以将该问题转化成在不同螺距下求解碰撞点螺旋半径的问题。通过不断减小螺距步长，使得碰撞点螺旋半径不断接近调头半径。循环调用求解问题 2 所使用的算法，直至螺距到达最优值 0.45m。

为了求出在掉头区域内完成调头的最短路程问题，我们首先分析在严格限制半径比值 1：2 情况下的最优调头路程，通过对其自由度的分析可知这是个非线性方程组求解问题，只有唯一解。为了最优化调头路径，我们释放关于半径比值的严格约束，使之增加一个自由度，从而变成非线性规划类问题。通过寻找图形中的几何关系，同时结合矩阵向量的思想，得出目标函数以及约束方程，最终求得最短路程长为 13.621224m。在此路径基础上，为了求解各把手实时位置坐标公式，先分析龙头的运动状态，接着设计了一套算法分类计算复杂路径下后续各个把手实时坐标位置，进而求得实时速度。

在问题 4 的基础上，不断缓慢增加龙头前把手的速度，直至在所有时刻下各把手的速度矩阵中出现一个大于 $2m/s$ 的临界值，对此时龙头前把手速度减去一个速度步长，求得符合要求的最大速度是 $1.10m/s$。

关键字：　仿真模拟　近似圆弧法　差分法　非线性规划　叉积法　循环迭代　几何法　算法设计

一、 问题背景与重述

1.1 问题背景

本文所研究的问题基于“板凳龙”这一独特的文化活动。板凳龙，是起源于河洛地区的传统民俗活动之一，普遍存在于我国南方各省市的年度盛大活动。这一元宵节习俗已有几个世纪。其中福建是当地人迎神、祈福的一项民俗活动，蕴含着深厚的地域文化。2008 年春节，中国福建省三明市大田县的“板凳龙”被列为国家级非物质文化遗产推荐项目。板凳龙相传源于汉代，由“舞龙求雨”的宗教活动演变而来。而大田的板凳龙的习俗据闻由八姓入闽时期中原的先民所带来，如今，大田板凳龙已成为远近闻名传统习俗。村民们把一节节的板凳钻孔连接，一户一节，组成板凳长龙，可达两百多米长，非常壮观。当地的居民将几十条乃至上百条的板凳首尾相连，形成蜿蜒曲折的“盘龙”，整体呈圆盘状。

本文所研究的问题正是基于这一文化活动给出的具体化数学问题。问题要求我们对一条整体为螺线结构的“盘龙”进行分析，从实际的物理现象中抽离出数学物理模型，建立方程并求解。题目给出了每条板凳的结构与相关数据。本文结合实际，建立数学模型，对以下问题进行了探讨与求解。

1.2 问题要求

问题 1：问题 1 要求我们建立合理的数学模型，用来描述舞龙队沿螺距为 55cm 的等螺距逆时针盘入的全过程。前把手的行进速度恒定为 1m/s，问题要求我们计算从初始时刻到 300 秒为止，每一秒龙头身和龙尾各前把手及龙尾后把手的位置和速度。问题要求我们将结果保存到指定的命名文件中。针对该问题，我们结合物理定律与仿真模拟，在极坐标系下建立了螺线方程，从而确定各处的位置与速度。该过程中我们将螺旋运动近似看作圆周运动，而螺旋线并非严格的圆弧，故而必然导致几度或分秒级的误差，随后我们引入误差度 η与修正系数 α得出了修正后的极坐标方程解析表达式。对于后续龙身把手位置，我们分别采用余弦法与近似圆弧法求解。针对把手速度，我们采用差分法求解。

问题 2：在问题 1 的基础上，本问题要求深入剖析终止时刻所对应的物理意义，并将其作为约束条件，确定舞龙队盘入的终止时刻。换言之，就是确定出即将发生碰撞而受迫无法再继续盘入的时间。我们需要计算并给出此时舞龙队的位置，并将结果保存到指定的文件中。针对该问题，我们将板凳的面近似为一个长矩形，因此需要给出一个点是否位于另一个矩形内部的判断依据。引入叉积法，用于判断点是否在矩形内，若在矩形外，即满足保持运动的条件，则增加时间步长继续求解；反之，则证明已经发生过碰

撞，发生碰撞的时刻即为终止时间。我们找出一个显然还未相撞的时间点，并以该时间点为起点，逐步迭代确定终止时间。

问题 3：问题 3 引入了一个全新的概念——“调头空间”，即以螺线中心为圆心、直径为 9m 的圆形区域。问题要求我们确定最小的螺距，使得龙头前把手能够沿着相应的螺线盘入到空间的边界，同时必须保证板凳之间不发生碰撞。与问题 2 相似，当螺距最小时，龙头左前方把手刚好在调头时与某一部分龙身发生碰撞。此时，我们可以从 0 开始逐渐增大螺距，找出刚好满足题意时的螺距，此时便确定了最小的螺距。

问题 4：问题 4 区别于前 3 个问题，设定了新的盘入和盘出条件。盘入螺线的螺距为 1.7m，盘螺线与盘入螺线关于螺线中心呈中心对称。在问题 3 设定的调头空间内，舞龙要完成调头。调头路径是由两段圆弧相切连接而成的 S 形曲线，其中前一段圆的半径是后一段的 2 倍，且与盘入、盘出螺线均相切。问题要求我们探讨是否可以通过圆弧来缩短调头曲线，同时保持各部分相切。还需要计算从-100 秒到 100 秒内每秒舞龙队的位置和速度，并在论文中给出特定时间点的数据。题目给出了前一段圆的半径是后一段的 2 倍时的条件，要求我们改变半径比，从而使调头曲线变短。很自然地，我们想到去求得调头曲线的最小值。因为此时调头曲线必然是变短的，很明显符合题意。详细的求解过程见下文的建模与求解。

问题 5：基于问题 4 设定的路径，问题 5 要求确定龙头的最大行进速度，使得舞龙队各把手的速度均不超过 2m/s。第四问求出的横纵坐标与时间相关的表达式，转化为以时间与速度为变量的表达式。由题意可知，所有板凳的速度均不超过 2m/s，则可从 v_{head}=1m/s 开始，以 dv 步长不断增大，遍历-100s 到 100s 的每一秒。当某一部分速度恰好达到 2m/s 时，取此时龙头的速度为最大速度。

二、 问题分析

2.1 问题 1：求解“板凳龙”各把手实时坐标和速度

问题一要求我们建立模型来计算“板凳龙”各个把手在螺线上的实时坐标和速度。首先，我们需要将具体的运动抽象成方便理解与运算的物理模型。分析了具体情境之后，我们发现，由于问题 1 侧重于各部分的运动，尚未考虑不同板凳之间的相互碰撞问题，因此我们可以将这 223 节“板凳龙”的运动质点化，从而利用题干信息求解结果。

根据螺线的性质，我们选择在极坐标中建立螺线的轨迹方程。即抽象成连接有 224 个把手的细杆刚体运动模型，其中每个把手均位于螺线上，各把手间欧式距离由实际意义保持恒定。考虑题干中所给的龙头信息，由于龙头线速度恒定为 1m/s, 根据角速度微分表达式和瞬时表达式，结合龙头初始值可以建立微分方程模型，等式两边定积分可求出龙头的角位移随时间变化关系表达式，再根据轨迹方程和平面坐标投影关系便得到龙头的实时坐标。

我们发现由于各把手与其上一个把手用“板凳”刚性连接，故它们的运动学特征存在关联。因此，我们可以建立循环迭代模型，只要前一个把手的运动学参数已知，就能在一次循环节中求得下一个把手的相关运动学参数。在处理相邻的把手时（即一个循环节中），由于螺线在一个微小角位移变化下其半径几乎不变，故而可以将相邻把手间的螺线等效为一小段微小的圆弧处理，且因连接把手的有效板凳长相对于半径为一个小量，因而可以进一步以直代曲，直接将有效板凳长度当做圆弧长来计算两把手对应的角位移差值，从而顺利得到了下一个把手的角位移大小，相应地，就不难求出其位置参数。在下面的章节中，我们还将介绍一种使用三角余弦定理结合超越方程数值解的迭代方法，并比较两种方法的误差程度，并且说明选择前者做法的理由。由于每个把手的实时位置坐标已知，接下来我们可以通过瞬时速度的近似定义，即一段微小时间内的位移除以时间长度来求出具体的速度参数。

2.2 问题 2：

问题二要求我们建立模型考虑“板凳龙”盘入的终止时刻。首先，我们需要明确，“板凳龙”的盘入终止可以分为两种情况。第一种为舞龙队龙头到达螺线的中心，无法再继续前进，则盘入自然终止；第二种为，早在龙头到达螺线的中心之前，舞龙队的龙头与其中一节龙身之间就已经发生了碰撞，则盘入的过程必然终止。我们需要将“碰撞”情况用数学的语言去区分和描述。因此，我们将板凳面等效为一个长矩形。故而，给出一个点是否位于另一个矩形内部的判断依据是必要的。我们引入叉积法来解决这个问题。该方法的详细步骤可分为：确定矩形的顶点、计算点到矩形各边的叉积以及判断点关于矩形的位置。若存在某一时刻，使得一个板凳的某个边界点出现在另一个矩形内部，则说明在该时刻之前，已经发生了碰撞，则运动必然终止。若这个时刻不存在，则龙头顺利到达终点，运动自然终止。我们应用 MATLAB 语言验证上述模型与假设，在逐步迭代的过程中，发现左前把手与第八节龙身在某一时刻发生了碰撞，运动也随之而终止。关于这一过程，详细的求解步骤将在下文中展示出来。

2.3 问题 3：

和问题 2 相似，问题 3 同样要求我们考虑碰撞条件。首先，我们需要明确，圆形的“调头空间”基于龙头先到达中心并转向的假设。因此，我们必须先筛选出满足龙头到达中心的时刻，才能进行后续操作。我们需要考虑“调头空间”所对应的物理意义。可以知道，“调头空间”越小，龙头前把手调头后与龙身发生碰撞的时刻越早。因此，我们可以通过发生碰撞的时刻，反向进一步确定“调头空间”的最小值。我们从 0 开始，不断增大“调头空间”的值，直到龙头前把手调头时刚好与龙身发生碰撞。此时，我们便成功确定了“调头空间”的最小值。

2.4 问题 4：

问题 4 给出了一套全新的框架，首先，我们需要读懂题意。可以明确的是，由盘入曲线中心与盘出曲线中心关于圆心对称可知，它们的连线必然经过圆心。另外，在盘入曲线与圆弧、盘出曲线与圆弧相交的两点，是相切的，即斜率相同，更进一步的说，关于时间的变化率相同。题目还给出大圆弧的半径为小圆弧的两倍。其次，当半径比固定时，盘入曲线的横纵坐标、盘出曲线的横纵坐标以及半径比五个变量对应五个已知条件，此时有关圆弧与螺旋线的所有条件都是逻辑自洽的。然而，题目要求我们改变半径比，即五个变量对应了四个已知条件。可以看出，调头曲线的长度可以用半径与角度求得。因此，问题 4 变成了我们所熟悉的非线性规划问题。最后，我们将约束条件改写为数学公式，目标函数正是题目要求我们计算的调头曲线，带入 MATLAB 求解即可得出答案。至此，我们将减小调头曲线的目标修正为求解调头曲线最小值的工作全部完成。

问题 4 的第二问与第一问相似，我们可以解出调头曲线的数学表达式。首先，我们需要对舞龙队的运动作分类，由于约束条件的存在，所有龙身的运动都可以由龙头的运动情况推出。因此，我们直接对龙头的运动分情况讨论即可。其次，龙头的运动状况可以分为在第一个圆弧上运动、在第二个圆弧上运动以及盘出曲线。值得注意的是，盘出曲线视为与盘入曲线关于螺旋线中心对称。因此只需考虑盘入曲线的结果并取相反数便可。利用约束条件求出所有龙身板凳的运动状况，将其列入表格当中。

2.5 问题 5：

问题 5 基于问题 4，令原本固定的速度 v 也成为变量。首先，我们需要明确，满足什么条件时，龙头达到了最大行进速度。题干中提出了要求，所有板凳的速度均不可超过 2m/s。因此，不难判断，当板凳龙的某一部分速度恰好达到 2m/s 时，此时龙头将达到最大行进速度。其次，我们需要将约束条件转化为数学表达式，若存在 $v_(t, v_{head}) > 2m/s$，那么取最大值 $v_{max} = v_{head} - dv$。结合物理知识，运用模拟仿真技术，从 $v_{head} = 1m/s$ 开始，以 dv 步长不断增大，遍历-100s 到 100s 的每一秒，顺利得出龙头的最大行进速度。

三、 模型假设

以下是我们在模型建立与求解时为了尽可能简便计算所作出的合理假设：

1. 一开始所有“板凳龙”的把手都已处于该螺旋线上。即在 0 时刻，不仅仅龙头前把手严格位于螺旋线上，后续把手依然位于该螺旋线上。
2. “板凳龙”可以被看作是一个由 224 个离散把手刚性连接的系统，每个把手代表一个板凳把手中心，这样我们可以忽略板凳的形变与内部结构，有助于模型的简化。

3. “板凳龙”龙头的速度被设定为恒定值（1m/s），每个把手严格按照路径运动，如均位于螺线或者圆弧线上。这种假设有助于使得运动更加理想化，尽管在现实中这种运动很难严格按照指定的速度与路线。
4. 板凳之间的连接完全光滑，忽略实际生活中存在的摩擦力，把手振动问题。在实际问题中，我们还需要考虑由摩擦等因素带来的能量损耗。
5. 在研究板凳间碰撞问题时，假设所有的板凳全部在同一平面内，尽管至少相邻的板凳连接一定会存在高度差。基于这种假设，我们只需要考虑平面内的碰撞，忽略三维运动和碰撞。
6. 在现实中，板凳的运动方程，包括其位置与速度参数一定是连续可微的，即可作图为一张连续光滑的曲线。在模型的建立与求解过程中，我们只能将时间假设为离散的分立数值，但是可以不断减小时间的步长，从而增强其连续性。

四、 符号说明

以下是建立模型时用到的主要符号及其说明

符号	含义说明
L_{head}	龙头长度（m）
L_{body}	龙身长度（m）
d	孔中心距离最近的板头距离（m）
L_{eff1}	龙头有效连接长度（m），这里为 $L_{head}-2d$
L_{eff2}	龙身有效连接长度，这里为 $L_{body}-2d$
v_{head}	龙头把手线速度（1m/s）
w	板宽（m）
p	螺距（m）
b	螺线系数，这里 $b=p/2\pi$
θ_i	第 i 个把手（第 $i-1$ 段龙身的前把手）的角位移（rad/s）
r_i	第 i 个把手的螺线半径（m）
ω	角速度（m/s）
(x_i,y_i)	第 i 个把手对应的横纵坐标（m）
(v_{ix},v_{iy},v_i)	第 i 个把手对应的三个速度参量（m/s）
$\begin{bmatrix}(x_{ifl},y_{ifl}) & (x_{ifr},y_{ifr})\\(x_{ibl},y_{ibl}) & (x_{ibr},y_{ibr})\end{bmatrix}$	第 i 个把手所处板凳四个边界点坐标矩阵
R	掉头区域半径（m）
γ	螺旋线与圆切线方向的夹角（rad）
η	速度误差
α	速度修正函数
S	调头曲线长度（m）
r_1,r_2	两条圆弧的半径（m）
θ_1,θ_2	两条圆弧对应的夹角（rad）
t_0	龙头进入第一条圆弧线的时刻（s）
t_{n1}	龙头离开第一条圆弧线的时刻（s）
t_{n2}	龙头离开第二条圆弧线的时刻（s）

五、 模型的建立与求解

5.1 问题 1 模型建立与求解

问题一要求求解任意 t 时刻所有把手的位置与速度参数。根据各个把手刚性相连的特点，只要已知龙头的运动参数，就可以通过循环迭代将龙头后连接的所有把手的运动参数求出。同时，应尽可能提高提高相关龙头运动参数的精确度，否则通过循环迭代求解出的其他后续把手必然也会存在较大误差。

5.1.1 问题 1 龙头位置求解

首先建立极坐标系下螺线的轨迹方程，相应地得出以 t 为变量的螺旋线上龙头把手的运动方程：

$$r(\theta) = \frac{11\theta}{40\pi} \tag{1}$$

$$r(t) = \frac{11\theta(t)}{40\pi} \tag{2}$$

下面建立微分方程模型进而求解出不同时刻 t 下龙头把手极坐标的解析表达式。分别展开角速度 $\omega(t)$ 的微分表达式和瞬时决定式，并使其相等从而得到一个微分方程。

$$\text{由}\vec{v} = \vec{\omega} \times \vec{r} \quad \text{有}\frac{v(t)}{r(t)} = -\omega(t) = -\frac{d\theta}{dt} \tag{3}$$

其中，根据题目所给条件，龙头把手从初始坐标 $(8.8, 32\pi)$ 开始顺时针盘入中心，故其角位移 $\theta(t)$ 逐渐减小，因此 $\omega(t)$ 前添符号。下面对 (3) 式两边求定积分：

$$\int_{32\pi}^{\theta} \theta \, d\theta = -\frac{40\pi}{11} \int_{0}^{t} dt \tag{4}$$

得到龙头把手中心极坐标表达式：

$$\begin{cases} \theta(t) = \sqrt{(32\pi)^2 - 80\pi t/11} \\ r(t) = 11\theta(t)/40\pi \end{cases} \tag{5}$$

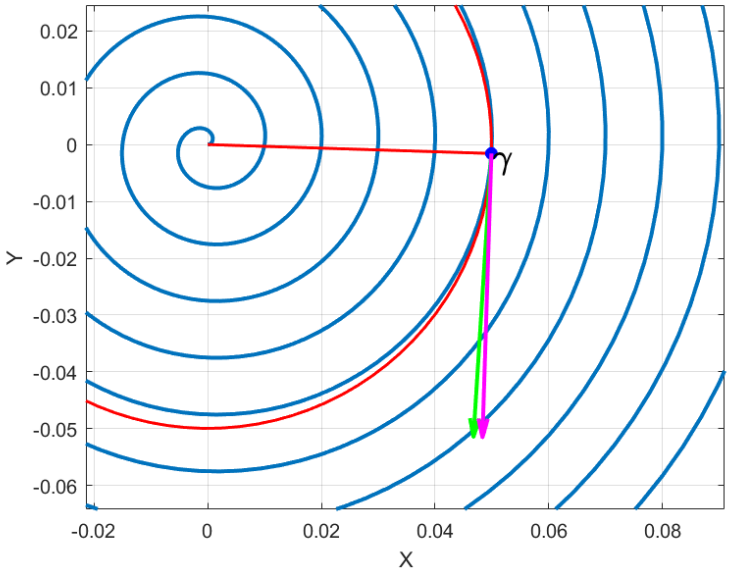
相应地，在平面直角坐标系投影螺旋半径可以求出龙头把手关于时间 t 的解析表达式：

$$\begin{cases} x(t) = r(t)\cos\theta(t) \\ y(t) = r(t)\sin\theta(t) \end{cases} \tag{6}$$

为了进一步求出当前时刻的横纵方向分速度 v_{1x} 和 v_{1y}，可以通过链式法则对上面的式子进行求导：

$$\begin{cases} v_{1x}(t) = \cos\theta(t) \cdot \frac{dr(t)}{dt} - r(t)\sin\theta(t) \cdot \frac{d\theta(t)}{dt} \\ v_{1y}(t) = \sin\theta(t) \cdot \frac{dr(t)}{dt} + r(t)\cos\theta(t) \cdot \frac{d\theta(t)}{dt} \end{cases} \tag{7}$$

图 1　螺旋运动和近似圆周运动夹角



其中：

$$\begin{cases} \frac{dr(t)}{dt} = -\frac{1}{\sqrt{(32\pi)^2 - \frac{80}{11}\pi t}} \\ \frac{d\theta(t)}{dt} = -\frac{4\pi}{110\sqrt{(32\pi)^2 - \frac{80}{11}\pi t}} \end{cases} \tag{8}$$

由于把手沿着螺旋线运动，而螺旋线并不是一个严格的圆弧，这就导致了其线速度和螺旋矢径之间的夹角并不是一个严格的直角，必然存在几度或分秒级别的误差。在 (3) 式中，我们是将这样的螺旋运动在各个时刻下的状态与其在同样半径的圆弧运动作了一步近似。因此在 (3) 式中，可以忽略径向速度 $v_r(t)$, 并直接将圆周运动的切向速度 $v_\theta(t)$ 等效为螺旋运动的线速度 $v(t)$。

假设任意一点处螺旋线的切线方向和相同半径圆弧切线方向夹角（即图中紫线和绿线之间的夹角）为 γ(见图 1)。我们可以通过先前的物理量推出 γ 与 θ 之前的关系式。为了拟合得到更好的数据，我们考虑螺线与圆弧之间的运动关系并用数学关系来定量表示。我们需要用到切向矢量来表达这两个量，等价关系可由以下式子导出：

$$\begin{cases} \vec{a} = (cos\theta - \theta sin\theta, sin\theta + \theta cos\theta) \\ \vec{b} = (cos\theta, sin\theta) \\ cos\gamma = \vec{a} \cdot \vec{b} / |\vec{a}||\vec{b}| \end{cases} \tag{9}$$

其中 $\vec{a}$ 为螺旋线的切向矢量，$\vec{b}$ 为圆周的切向矢量，联立并求解上述式子，有以下关系：

$$cos\gamma = \frac{1}{1+\theta^2} \tag{10}$$

对式子变形，有：

$$\gamma = arccos\frac{1}{1+\theta^2} \tag{11}$$

不难看出，在角位移较小时，γ 趋向于 0，此时将螺旋运动近似为圆周运动是十分合理的。因此我们所使用的近似法是科学而有效的（见图 2）。

根据这一重要的等价关系，我们可以计算在将瞬时时刻螺旋运动近似成圆周运动时所产生的速度误差 η。螺线运动的径向速度 $v_\theta(t)$，切向速度 $v_r(t)$，合速度 $v(t)$, 方向夹角 γ 和误差度 η 关系如下：

$$\begin{cases} tan\gamma(t) = v_r(t)/v_\theta(t) \\ v(t) = \sqrt{v_\theta(t)^2 + v_r(t)^2} \\ v(t) \equiv 1 \\ \eta(t) = 1 - v_r(t)/v(t) \end{cases} \tag{12}$$

最终就能求得速度误差函数 $\eta(t)$。只要知道了确切的误差就可以对 (3) 式进行速度修正。定义修正系数 $\alpha(t) = 1 - \eta(t)$，可以如下修改 (3) 式两边求定积分：

$$\begin{cases} v_\theta(t)/r(t) = -d\theta/dt = -\omega(t) \\ v_\theta(t) = \alpha(t) \cdot v(t) \end{cases} \tag{13}$$

相应定积分求解式变更为：

$$\int_{32\pi}^{\theta} \theta \, d\theta = -\frac{40\pi}{11} \int_0^t \alpha(t) \, dt \tag{14}$$

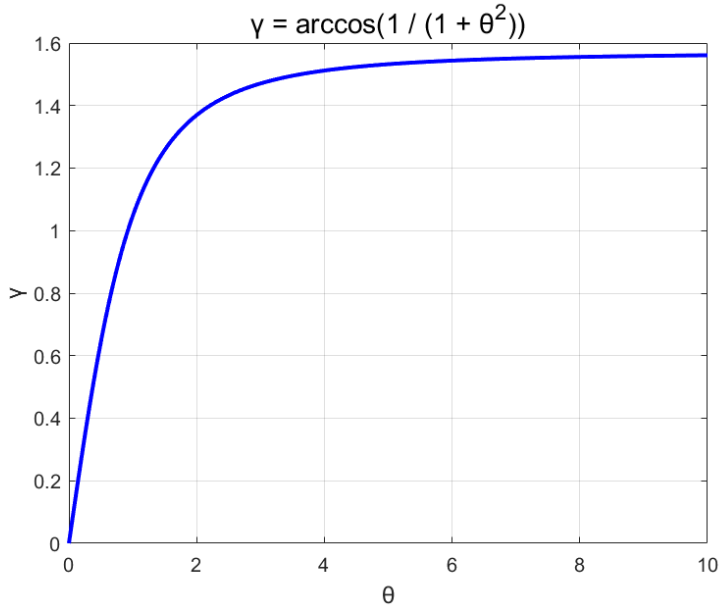
尽管相对误差函数 $\alpha(t)$ 是关于时间的函数，我们仍然很难求出其关于时间 t 的定积分，因为相对误差函数 $\alpha(t)$ 的表达式是十分复杂的，甚至涉及到通过参数代换。因此我们所作的一步近似是行之有效的。

5.1.2 问题 1 后续龙身把手位置求解（余弦法）

根据实际情形不难作出相邻把手间的位置关系图。对于两个相邻的把手，需要满足两个约束条件：两点均位于螺旋线上，且两点直线距离为一个给定的定值 L。根据牵引关系，不难判断后续把手的 θ_i 要大于当前把手的 θ_{i-1}。假设两者的夹角为 $\Delta\theta$ ，显然 $\Delta\theta > 0$。根据三角余弦定理，结合轨迹方程，由两点的螺旋半径长度和夹角来可列出方程：

$$\begin{cases} L^2 = r(\theta_{i-1})^2 + r(\theta_{i-1} + \Delta\theta)^2 - 2r(\theta_{i-1})r(\theta_{i-1} + \Delta\theta)\cos\Delta\theta \\ \theta_i = \theta_{i-1} + \Delta\theta \\ r(\theta_{i-1}) = 11\theta_{i-1}/40\pi \\ r(\theta_i) = 11\theta_i/40\pi \end{cases} \tag{15}$$

图 2　切线方向夹角与角位移的关系



其中，当前把手（第 i-1 个把手）的位置参数全部已知，即 $\theta_{i-1}, r(\theta_{i-1})$。实际上我们可以将后三个式子全部带入第一个式子，这样就得到方程:

$$L^2 = r(\theta_{i-1})^2 + \left(\frac{11}{40\pi}\right)^2 (\theta_{i-1} + \Delta\theta)^2 - 2 \cdot r(\theta_{i-1}) \cdot \left(\frac{11}{40\pi}\right) \cdot (\theta_{i-1} + \Delta\theta) \cdot \cos(\Delta\theta) \quad (16)$$

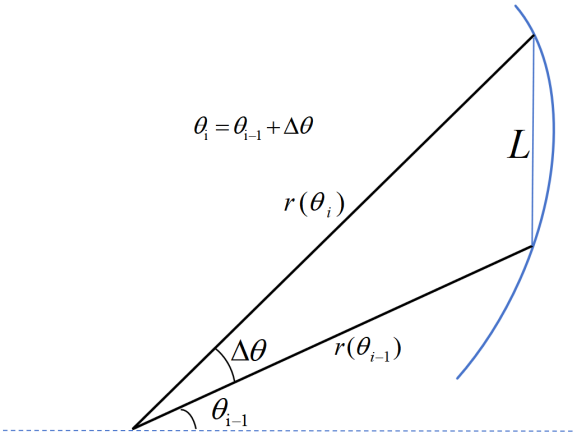
该方程属于超越方程，无法解出其解析解，只能使用计算机求一个较好的数值解。以下是一种可能的算法：我们可以让 $\Delta\theta$ 从 0 开始以一定步长增加，在接近 L 时换用更小的步长搜索数值解。下面我们需要衡量这种做法的误差大小。不妨定义 δ_1 为该种迭代方法的在一次循环中的误差程度。我们以第一个和第二个把手在 $t = 0$ 时刻运行一次得到结果为例（代码见附录）。求出了龙头至第一个把手间 $\Delta\theta = 0.3259$，单次循环精度 $\delta_1 \approx 10^{-3}$ 数量级左右。在半径较大时，由于 $\Delta\theta$ 普遍偏小，故其误差较大。因此随着迭代次数的增加，计算更外围的把手时，其误差更大，而当时刻 t 逐渐增加后，随着把手向内运动，其精度会越来越高。

在 $t \in [0, 300]$ 区间内，各个把手对应螺线半径较大，故该方法会产生较大误差。下面我们来介绍一种适合螺旋半径较大的解法。

5.1.3 问题 1 后续龙身把手位置求解（近似圆弧法）

下面介绍一种新解法，我们称之为近似圆弧法，其做法是在处理半径较大的螺线时我们可以将其等效为圆弧处理。理由是根据螺线轨迹方程，每当把手转过 2π 时，其半径衰减 55cm，而当半径较大时（5—8m），每次迭代一个有效板凳长作为螺旋线的弦所转过的角位移只有约 0.35 rad/s。我们定义这种方法的误差度为 δ_2，根据计算，在时间不太大（300s 内），后继把手数较多的情况下，$\delta_2 \approx 10^{-4}$ 数量级左右，精度明显更好。

图 3　余弦定理求下一个把手位置



下面给出一个具体算法：首先根据龙头极坐标解析表达式求得任意时刻 t 时的运动学状态，再应用近似圆弧法从任意时刻 t 的龙头坐标开始迭代，最终得到任意时刻“板凳龙”各把手坐标数值，即任意时刻“板凳龙”的运动状态。仿真模拟算法步骤如以下**算法 1** 所述。

算法 1：近似圆弧迭代法模拟舞龙的过程仿真：

输出：存储所有把手在所有整数时刻的位置，角位移矩阵。

S1：创建两个矩阵，用行存储所有把手，用列存储所有要求解的时刻 t。

S2：根据先前求得的修正后龙头把手实时极坐标表达式 (11)，可以得到任意时刻龙头把手的坐标，同时将求解结果填充进位置矩阵前两行。

S3：下面从每个时刻 t 入手，即已经进入第一层关于时间 t 的循环后，从龙头把手进入循环迭代过程，迭代对象和顺序依次是后续 223 个把手，从第 2 个把手开始循环迭代。对每个把手执行 **S4**。

S4：该步骤介绍从第 i 个把手到第 i+1 个把手的单次迭代循环过程。首先计算角位移的偏转值 $\Delta\theta_i$，两个把手之间的螺旋线近似作为圆弧处理，由圆周角三角形的几何公式，有：

$$\Delta\theta_i = 2\arcsin\frac{L}{2r(\theta_i)} \tag{17}$$

考虑到计算机编译器在求解反正弦数值的精确度和先前近似圆弧的误差，这里可以进一步作“以直代曲”的近似，直接将把手间有效连接长度 L 作为这段圆弧长。由于先前将 $r_(\theta_{i+1})$ 近似为 $r_(\theta_i)$ 会导致求解得到的 $\Delta\theta_i$ 偏大，而这里以直代曲又使得 $\Delta\theta_i$ 偏小，且两者误差值位于同一数量级的水平，故作两次趋势相反的近似不仅简化了计算难度，也使得求出的 $\Delta\theta_i$ 精度更高。角位移偏转量方程为：

$$\Delta\theta_i = \frac{L}{r(\theta_i)} \tag{18}$$

再通过以下方程组：

$$\begin{cases} \theta_{i+1} = \theta_i + \Delta\theta \\ r(\theta_{i+1}) = 11\theta_{i+1}/40\pi \end{cases} \tag{19}$$

最终得到了第 i+1 把手的极坐标 $(r(\theta_{i+1}), \theta_{i+1})$，可以通过一步平面直角坐标系投影得到最终的平面坐标 $(r(\theta_{i+1})\cos\theta_{i+1}, r(\theta_{i+1})\sin\theta_{i+1})$。

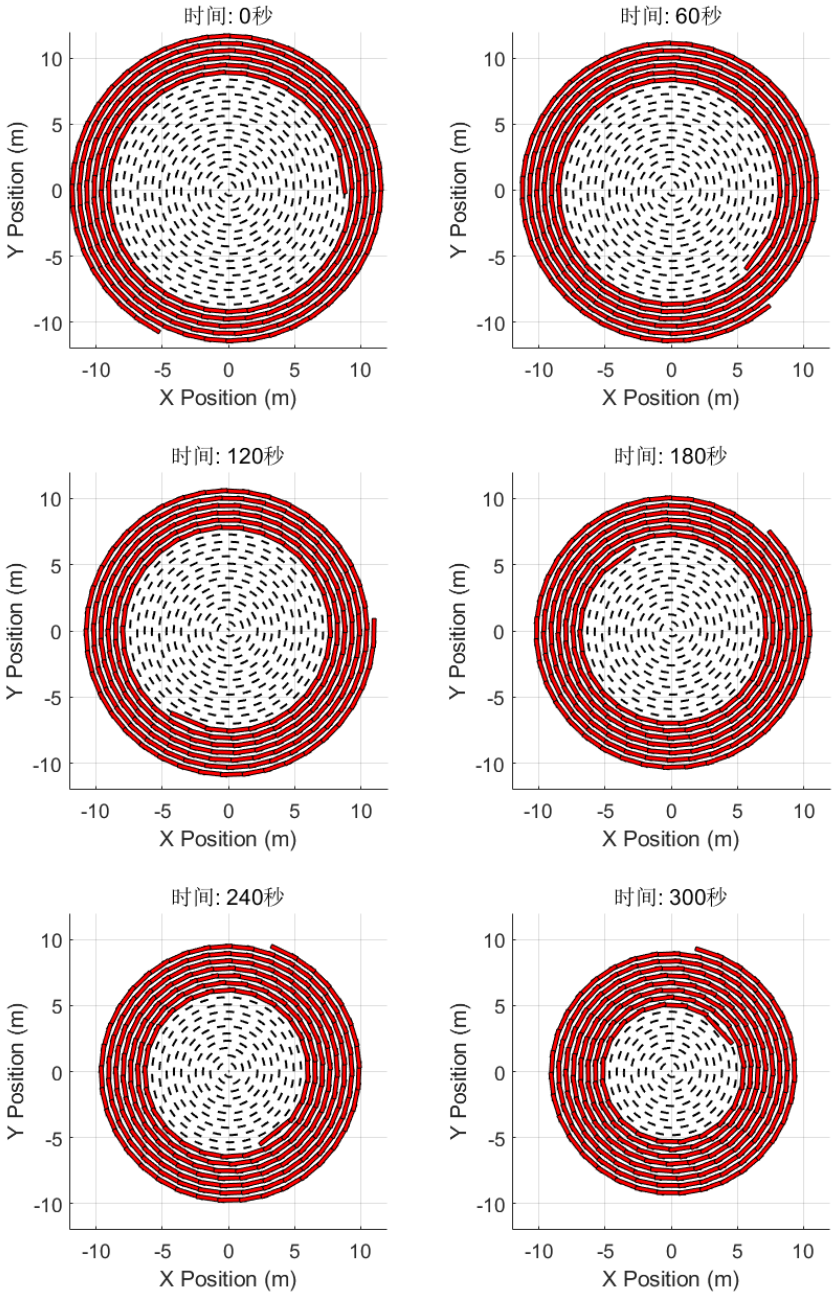
S5：对于时间循环 t 中的把手循环，循环执行 **S4**，由于龙头的存在，需要添加判断条件来区分不同板凳对应的有效连接长度。

S6：因为已经求出了完整的各个时刻每个把手的位置信息，这使得我们有能力对全过程进行仿真模拟，只需要不断减小时间步长 t，就可以逐渐趋近于一个连续过程。可以借助计算机编译器，将全过程绘制成动画，这也为问题 2 判断相撞时间提供了依据。通过计算机编译手段求得在给定时刻下把手的位置坐标，结果如表 1 所示。

表 1　位置结果

	$0s$	$60s$	$120s$	$180s$	$240s$	$300s$
龙头 x (m)	8.8	5.796934	-4.090654	-2.953259	2.578971	4.431365
龙头 y (m)	0.000000	-5.773329	-6.300643	6.099638	-5.363954	2.298233
第 1 节龙身 x	8.366286	3.456016	-6.111764	-0.194664	-191234	4.919684
第 1 节龙身 y	2.819001	-7.381867	-4.310255	6.737217	-5.906572	-0.466255
第 51 节龙身 x	-9.512018	5.486829	-1.43511	3.932879	0.397575	0.227612
第 51 节龙身 y	1.385148	4.680284	6.28097	3.942671	4.512326	-3.160608
第 101 节龙身 x	2.841064	3.68234	-5.07733	-3.454000	-2.47359	0.555436
第 101 节龙身 y	-9.938777	4.902125	1.148161	2.164192	0.20858	0.125698
第 151 节龙身 x	10.877538	0.124579	-3.54136	0.133836	-1.65013	2.255574
第 151 节龙身 y	1.727014	-4.81315	0.409225	-1.51431	0.002419	10.325175
第 201 节龙身 x	4.671434	-2.35762	4.940488	0.532093	-1.24801	8.958199
第 201 节龙身 y	10.67392	1.802727	0.406542	0.73535	1.916431	-4.124826
龙尾（后）x	-5.427245	1.281067	-0.406418	1.741699	0.019065	-4.909683
龙尾（后）y	-10.61406	-0.94886	-4.25146	0.734097	0.147523	8.173638

图 4　几个重要时刻“板凳龙”位置图



5.1.4 问题 1 把手速度求解（差分法）

由于在上一小节中已经可以对全过程进行模拟仿真，因此求解实时速度已经不再是问题。根据速度定义公式，有：

$$v_{ix} = \lim_{\Delta t \to 0} \frac{\Delta x_i}{\Delta t} \quad v_{iy} = \lim_{\Delta t \to 0} \frac{\Delta y_i}{\Delta t} \tag{20}$$

平方和求合速度：

$$v_i = \sqrt{v_{ix}^2 + v_{iy}^2} \tag{21}$$

由于无法对全过程进行连续的模拟仿真，我们只能取分立的值。重新设置一个关于时刻 t 新的步长 $\delta t = 0.0001$。用从这一时刻开始的微小时间内的平均速度近似代替在该时刻的瞬时速度。因而有计算公式：

$$v_{ix}(t) = \frac{x_i(t+\delta t) - x_i(t)}{\delta t} \quad v_{iy}(t) = \frac{y_i(t+\delta t) - y_i(t)}{\delta t} \tag{22}$$

为了方便计算，把**算法 1** 封装进一个函数方便调用，接受的自变量为时间 t，返回一个存储有该时刻所有点坐标的向量，通过这个函数不断向位置矩阵中写入向量。在循环分别对 t 和 $t+\delta$ 循环，得到两个存储有各个时刻和各个时刻后一段微小时间的矩阵，将两个矩阵相减并将全部元素除以 δt，就得到了所有把手的速度矩阵，题目要求时刻和把手的时刻数据如表 2，“板凳龙”位置图如图 4。

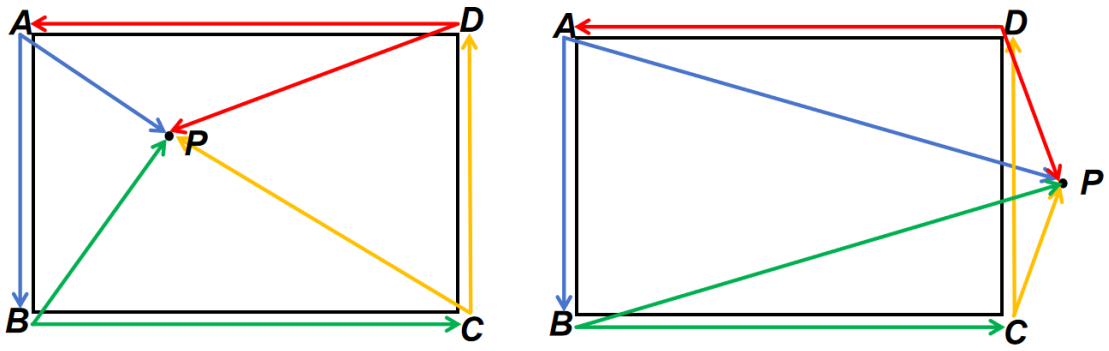
表 2　速度结果

	$0s$	$60s$	$120s$	$180s$	$240s$	$300s$
龙头	1.00005	1.00006	1.00007	1.00008	1.00011	1.00015
第 1 节龙身	1.00004	1.00004	1.00007	1.00005	1.00006	1.00005
第 51 节龙身	0.99989	0.99984	1.00005	0.99966	0.99945	0.99896
第 101 节龙身	0.99977	0.9997	0.99978	0.99942	0.99911	0.99845
第 151 节龙身	0.99969	0.9996	0.9996	0.99926	0.99889	0.99816
第 201 节龙身	0.9996	0.99952	0.99947	0.99914	0.99875	0.99796
龙尾（后）	0.99949	0.99949	0.99934	0.9991	0.9987	0.9979

5.2 问题 2 模型建立与求解

为了判断在龙头到达极坐标原点前板凳之间是否会发生碰撞，以及找出碰撞的判据，同时求出具体碰撞的板凳节及其相应地坐标速度等运动状态，需要建立一套有效的碰撞检测机制。由于“板凳龙”系统一共涉及到 224 节板凳，如果对每一节依次判断与其他 223 节是否相撞将极大地增加计算量，因此需要对这个预测碰撞系统做一些简化以降低计算复杂度。比如先证明出某一节板凳的某个把手最容易和其他板凳发生碰撞。其次，由于在问题 1 中已经建立了“板凳龙”运动状态的实时仿真模拟模型，通过对动画的观察，可以大致确定一个时间点 t_x，在 t_x 前一定不碰撞。最后，我们需要考虑数据

图 5　叉积法判断点是否在矩形内



计算的误差。在具体计算过程中，随着循环迭代的次数不断增加，单次计算产生的误差会随着循环次数增加不断累积，导致最终计算结果的偏差，因此需要找出一种方法尽可能地缩小误差。

综上，我们建立一套“板凳间实时位置判断”的实时仿真模拟模型来解决这一问题。下面先介绍一种判断单个点与矩阵位置关系的方法——叉积法。

5.2.1 建模准备：叉积法判断点与矩形位置关系

由于在本问题中涉及对板凳碰撞的讨论，而板凳被等效成了一个长矩形，因此需要给出一个点是否位于另外一个矩形内部的判断依据。如果在仿真模拟中发现在某一时刻 t_0，存在某一个板凳的某个边界点出现在另一个矩形内部，则说明至少在 t_0 以前发生了碰撞。

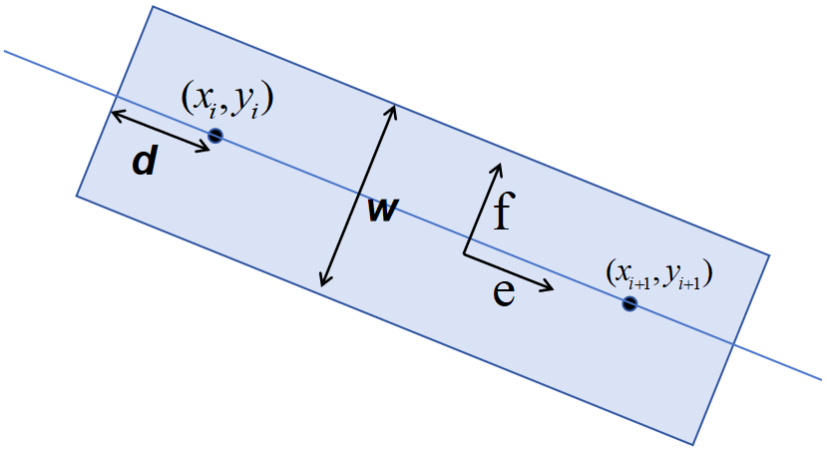
以下是叉积法的详细步骤：

1. **确定矩形的顶点**：假设矩形的四个顶点为 A、B、C、D（按顺时针或逆时针顺序排列）。
2. **计算点到矩形各边的叉积**：选择要判断的点 P。对于矩形的每一条边（假设边是由点 A 到点 B），计算向量 $\overrightarrow{AB}$ 和向量 $\overrightarrow{AP}$ 的叉乘，其中 $\overrightarrow{AB}$ 是矩形边的向量，$\overrightarrow{AP}$ 是矩形顶点到点 P 的向量。由于两个向量均位于平面上，所以仅需考虑叉乘在竖直方向上的正负，向量叉乘竖直方向竖直计算公式：

$$(\overrightarrow{AB} \times \overrightarrow{AP})_z = (x_B - x_A) \cdot (y_P - y_A) - (y_B - y_A) \cdot (x_P - x_A) \tag{23}$$

3. **判断点关于矩形的位置**：如果点 P 在矩形的每一边的相同一侧（即所有边的叉积结果都具有相同的符号），那么点 P 就在矩形内部。因此，若平面内一点位于矩形内部，则满足以下方程组：

图 6　板凳四个边界点求解示意图



$$\begin{cases} (\overrightarrow{AB} \times \overrightarrow{AP})_z > 0 \\ (\overrightarrow{BC} \times \overrightarrow{BP})_z > 0 \\ (\overrightarrow{CD} \times \overrightarrow{CP})_z > 0 \\ (\overrightarrow{DA} \times \overrightarrow{DP})_z > 0 \end{cases} \text{或} \begin{cases} (\overrightarrow{AB} \times \overrightarrow{AP})_z < 0 \\ (\overrightarrow{BC} \times \overrightarrow{BP})_z < 0 \\ (\overrightarrow{CD} \times \overrightarrow{CP})_z < 0 \\ (\overrightarrow{DA} \times \overrightarrow{DP})_z < 0 \end{cases} \tag{24}$$

5.2.2 判断最先发生碰撞的板凳

我们可以进行以下三个断言：

1. 一定是龙头那一节板凳最先与其他板凳发生碰撞。
2. 以“板凳龙”前进方向为正方向，一定是龙头板凳所对应矩形的左前方边界点最先发生碰撞。
3. 碰撞时刻，以龙头把手为起点，沿着螺线逆时针往回倒 $5\pi/2$，只需要检查角位移 θ 与龙头把手角位移 θ_1 相差不超过 $5\pi/2$ 的板凳把手，即与龙头相撞的板凳一定在这些板凳之间。

下面我们依次证明这三个断言使之成为正式的结论。

结论 1：证明：使用反证法，假设存在龙头后续某块板凳（设其前把手标号为 i）首先在时刻 t_2 与除龙头外的其他板凳相撞（设其前把手标号为 j），如果是龙头，就是龙头除去边界点的四个边。记录当前第 i 把手的角位移 θ_i。由于“板凳龙”的结构是由龙头牵引其他龙身运动，即龙头角位移 θ_1 一定比 θ_i 小，换言之龙头一定在先前的某一时刻 t_1 到达过角位移为 θ_i 的点，其中 $t_1 < t_2$。由于龙头部分的长度 L_{head} 显然要长于龙身长度 L_{body}，如果第 i 个龙身在 θ_i 处可以和其他板凳发生碰撞，那么龙头在 θ_i 处也一定可以与后面的板凳发生碰撞。这与所提假设矛盾。

结论 2：证明：以“板凳龙”前进方向为正方向，我们依次标定矩形的四个边界点：依次分为左前、右前、左后、右后。首先先证明左后和右后边界点一定不可能是龙头首先与其他板凳发生碰撞的把手。根据题目已知条件，螺距 $p = 55cm$，板凳后把手中心到后两个把手连线的距离（即与后板头的距离）为 27.5cm。可以作辅助线：相邻两条螺线的中间平分线。无论“板凳龙”盘入什么位置，后把手边界一定在相邻螺线的中心平分线内，更为通俗地说，这条螺线平分线完全可以将相邻螺线上的板凳矩形严格分割。

下面接着证明，除了边界点以外的边不可能与其他板凳的任何点发生碰撞。依然使用反证法，假设龙头矩形除了边界点以外的边可以与其他板凳的某个点发生碰撞。显然这个假设等价于结论 1，矛盾所以舍去。

最后证明，龙头右前把手不可能与其他板凳发生碰撞。根据实时仿真模拟图，龙头板凳右前边界点的相邻右侧螺线以及后续相邻的螺线上均不存在板凳节，故显然不可能发生碰撞。

结论 3：证明：一个显然的现象是，无论如何，碰撞只可能在相邻一个螺线范围内发生。即对于相碰的两个板凳，他们的角位移分别设为 θ_i 和 θ_j，由于他们不可能相隔一个螺线，故 θ_i 和 θ_j 的差值 $\Delta\theta_{ij} = \theta_i - \theta_j$ 一定在 2π 左右，为了保险起见，我们将这一数值设置为 $5/2\pi$。根据实时仿真模拟可以判断出，“板凳龙”系统在第 400s 时不会发生碰撞，此时综合考虑龙头把手对应的弧度及其螺旋半径长作放大估算，我们只需要从第 1 节龙身板凳遍历到第 18 龙身板凳即可。

在正式建立检测模型前，为了尽可能减少我们的计算复杂度，提高计算效率，首先得出了以上三个结论。根据第一个结论，我们可以减少一轮循环，不需要再去检查 224 个板凳的四个边界点中的每一个点是否与其他 223 个板凳相碰。根据第二个结论，我们不再需要一一遍历龙头板凳四个边界点，而是只需要去判断左前边界点是否与后续的板凳发生碰撞。而最后一个结论大大减少了检查循环中所需要的轮数，每轮遍历只需要对龙头板凳后十八节板凳进行判断即可。

5.2.3 碰撞条件和板凳矩形区域求解

根据上一小节的三个定理，我们把板凳间的相互碰撞问题转化成了点与矩形的碰撞问题，也就是点与矩阵的位置关系问题，可以通过叉积法解决。设有点 (x_0, y_0) 和矩形 $ABCD$，矩形的四个边界点坐标依次为 $(x_{11}, y_{11}), (x_{12}, y_{12}), (x_{21}, y_{21}), (x_{22}, y_{22})$。我们记区域 H 为由这四个点连线所形成的矩形，则关于碰撞条件，有下面的关系式（以相碰为例）：

$$H = \begin{bmatrix} (x_{11}, y_{11}) & (x_{12}, y_{12}) \\ (x_{21}, y_{21}) & (x_{22}, y_{22}) \end{bmatrix} \quad (x_0, y_0) \in H \tag{25}$$

将上面的式子用叉积法转换成坐标表达式，（两个方程组只满足其中一个）：

$$\begin{cases}(x_{11}-x_{12})\cdot(y_0-y_{11})-(y_{11}-y_{12})\cdot(x_0-x_{11})>0\\(x_{12}-x_{21})\cdot(y_0-y_{12})-(y_{12}-y_{21})\cdot(x_0-x_{12})>0\\(x_{21}-x_{22})\cdot(y_0-y_{21})-(y_{21}-y_{22})\cdot(x_0-x_{21})>0\\(x_{22}-x_{11})\cdot(y_0-y_{22})-(y_{22}-y_{11})\cdot(x_0-x_{22})>0\end{cases} \tag{26}$$

$$\begin{cases}(x_{11}-x_{12})\cdot(y_0-y_{11})-(y_{11}-y_{12})\cdot(x_0-x_{11})<0\\(x_{12}-x_{21})\cdot(y_0-y_{12})-(y_{12}-y_{21})\cdot(x_0-x_{12})<0\\(x_{21}-x_{22})\cdot(y_0-y_{21})-(y_{21}-y_{22})\cdot(x_0-x_{21})<0\\(x_{22}-x_{11})\cdot(y_0-y_{22})-(y_{22}-y_{11})\cdot(x_0-x_{22})<0\end{cases} \tag{27}$$

在具体代码实现时，可以将以上判断过程封装进一个函数，接口为龙头板凳左前边界点坐标和后续板凳矩形边界点坐标。下面的问题在于，如何求出所需要的相应矩形边界点坐标，即求出第 i 个把手所处板凳四个边界点坐标矩阵 $\begin{bmatrix}(x_{ifl},y_{ifl}) & (x_{ifr},y_{ifr})\\(x_{ibl},y_{ibl}) & (x_{ibr},y_{ibr})\end{bmatrix}$。由于相邻把手分别是一条板凳的前后把手，因此只要知道一对相邻板凳把手的坐标，就可以将该板凳的四个边界点坐标完全解出。

由于已经求出各时刻所有把手的位置坐标，因此在求解边点时不再需要通过循环迭代，只需要传入求好的位置矩阵即可。根据图 6，对于第 i 节板凳（前把手为第 i 个把手），需要求出能够表示其走向的方向向量，定义 $\vec{e_i}$ 来表示第 i 块板凳的方向（从第 i 个把手指向第 i+1 个把手），$\vec{f_i}$ 表示其法向有：

$$\vec{e_i}=\frac{(x_{i+1}-x_i,y_{i+1}-y_i)}{\sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2}} \tag{28}$$

$$\vec{f_i}=\frac{(y_i-x_{i+1},x_{i+1}-x_i)}{\sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2}} \tag{29}$$

下面可以通过 $\vec{e_i}$ 结合其他相应的参数如板宽 w，板头板尾长 d 可以求出该矩形四个边界点坐标：

$$(x_{ifl},y_{ifl})=(x_i,y_i)-d\cdot\vec{e_i}-\frac{w}{2}\cdot\vec{f_i} \tag{30}$$

$$(x_{ifr},y_{ifr})=(x_i,y_i)-d\cdot\vec{e_i}+\frac{w}{2}\cdot\vec{f_i} \tag{31}$$

$$(x_{ibl},y_{ibl})=(x_{i+1},y_{i+1})+d\cdot\vec{e_i}-\frac{w}{2}\cdot\vec{f_i} \tag{32}$$

$$(x_{ibr},y_{ibr})=(x_{i+1},y_{i+1})+d\cdot\vec{e_i}+\frac{w}{2}\cdot\vec{f_i} \tag{33}$$

可以将上述求解过程直接封装进一个函数，在函数中定义一个矩阵直接存储所有把手在所有时刻下对应板凳边点的坐标。

图 7　碰撞时刻全局示意图

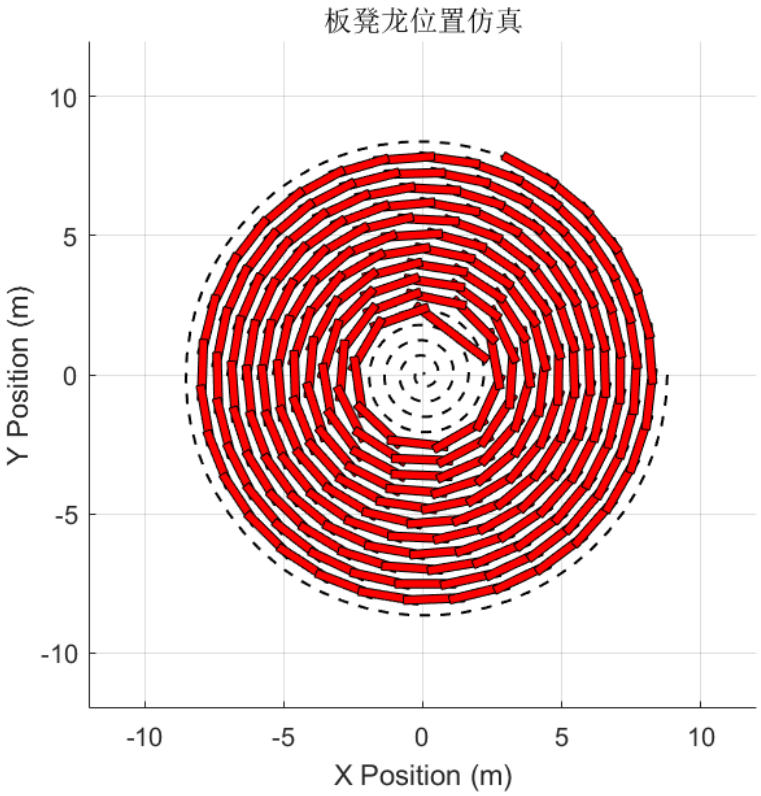
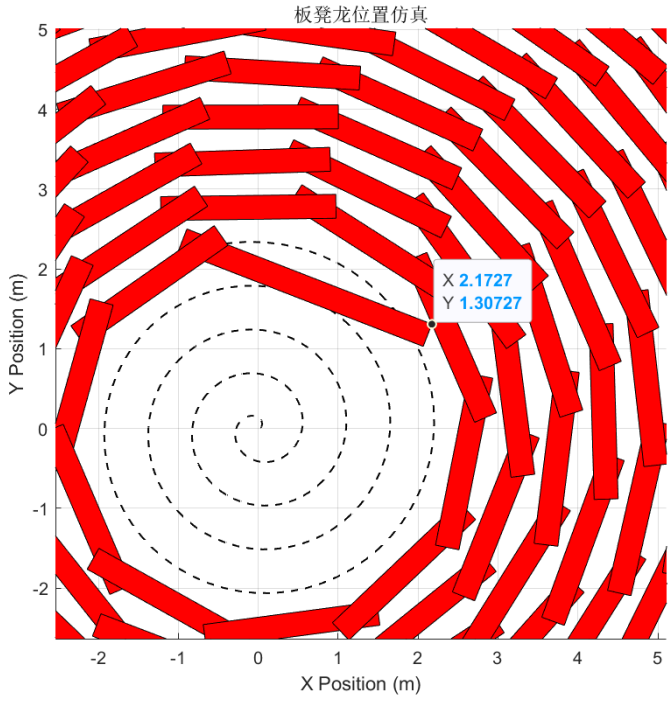


图 8　碰撞时刻局部示意图



5.2.4 判断板凳间实时位置的仿真模拟模型和结果

经过以上几个小节的铺垫，判断板凳间实时位置的仿真模拟模型已经呼之欲出了。我们可以只通过两轮循环就求解出所有的龙头板凳左前边点在任意时刻关于是否与后续板凳发生碰撞的布尔值矩阵。我们通过一个算法来描述这个“判断板凳间实时位置的仿真模拟模型”。为了后续得到精确的碰撞时刻，同时提高仿真模拟的精确程度，我们设置时间步长为 $\Delta t = 0.01$。

算法 2：判断板凳间实时位置的仿真模拟模型：

输入：存储所有把手在所有时刻的位置矩阵，板凳边点矩阵。

输出：“板凳龙”实时运动状态动画，碰撞时刻，碰撞时刻下各个把手的位置与速度矩阵。

S1：进入第一轮循环，每一个循环节对应一个时刻。时刻 t 从 400s 开始，以 Δt 的步长逐渐增加，然后再进入一轮循环。

S2：进入第二轮循环，每一个循环节对应一个板凳的判断过程，只需要遍历从龙头板凳开始的 18 节龙身板凳，由此进入第三轮小遍历。

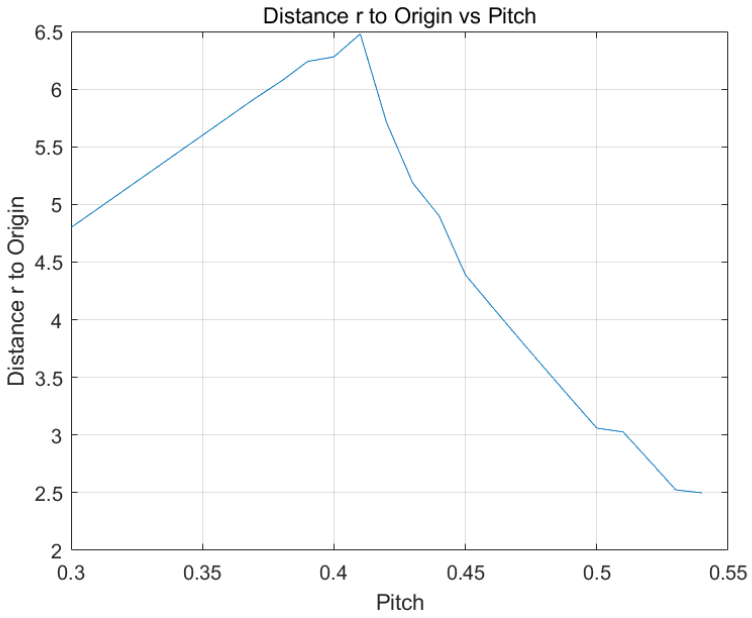
S3：进入第三轮小遍历，依次遍历一个板凳矩形的四个把手，对于每个板凳边点，调用叉积法判断点与矩形相碰与否，同时传入龙头板凳左前点坐标。

根据上述算法不难编程求解得到答案，最终求得在时刻 $t_{crash} = 413.36s$ 与第 8 节龙身，即从龙头开始的第九节板凳龙发生碰撞，此时龙头前把手的坐标为 (2.1727, 1.30727)。相应地碰撞时刻全局状态图和局部状态图分别如图 7，图 8 所示。此时龙头前把手、龙头后面第 1、51、101、151、201 条龙身前把手和龙尾后把手的位置和速度如表 3 所示。

表 3　碰撞时刻对应

	$x(m)$	$y(m)$	$v(m/s)$
龙头	1.856699	1.275657	1.000755
第 1 节龙身	-0.700321	2.257706	0.998250
第 51 节龙身	2.489526	3.731819	0.988250
第 101 节龙身	-1.960849	-5.546337	0.986675
第 151 节龙身	-0.569067	-6.982230	0.986033
第 201 节龙身	-7.962845	0.361181	0.985683
龙尾（后）	2.530747	7.968352	0.985576

图 9　碰撞时龙头与中心距离示意图



5.3 问题 3 模型建立与求解

问题三要求我们确定最小螺距，使得龙头前把手能够沿着相应的螺线盘入到调头空间的边界。这个问题本质上是在寻找一个平衡点，即螺距既要足够小以满足题目所需要求，又要足够大以避免在龙头前把手到达调头空间前发生板凳之间的碰撞。我们需要考虑的关键因素包括螺线的几何特性、舞龙队的物理结构以及调头空间的限制。

5.3.1 问题 3 模型建立

在问题二我们已经求出了当螺距 p=0.55m 时第一次发生碰撞时的时间，以及各把手在第一次碰撞时的位置和速度信息，若将 p 设为参数，则我们可以通过传入参数 p，求得在任意螺距下第一次发生碰撞时各把手的位置和速度参数。现在我们须求最小的螺距，使得龙头前把手在发生碰撞之前就已经到达了调头空间的边界。若此时假设没有调头空间，龙头依然按盘入螺旋线盘入，可知，当之后的某一时刻发生碰撞时，龙头的前把手一定位于调头空间之内。若此时龙头前把手的坐标为 (x_1, y_1)，则一定满足：

$$x_1^2 + y_1^2 \leq R^2 \tag{34}$$

其中，R 为调头空间的半径。显然，由于螺距越小发生碰撞的可能性越大。因此，我们可以从螺距 p=0.55 开始，以一个较为粗略的步长不断减小螺距，求出当前螺距下第一次发生碰撞时龙头前把手到圆心的距离。结果如图 9 所示，随着螺距逐渐增大，龙头与某一部分龙身发生碰撞时距离中心的距离也有所不同。可以看出，随着螺距逐渐增大，碰撞时龙头距离中心的距离先增加再减小。不难发现，螺距从 0.3m 到 0.41m 左右时，龙

头出发点与中心的距离不断增大；当螺距再增加时，由于间距过大，龙头与龙身之间更加难以碰撞，因此要到靠近中心的位置才可能发生碰撞。

我们的目标是求得最小的满足式 (34) 的螺距 p_{min}。由于 R=4.5m，从图 9 可以粗略估计：p_{min} 位于 0.4 到 0.45 这一区间内。因此我们可以从 p=0.4m 开始，不断增大进行较高精度的搜索，直至根据当前 p 求出的 (x_1,y_1) 满足式 (34) 方程。

5.3.2 问题 3 模型的求解

为了实现上述螺距优化模型，我们设计了一个迭代优化算法。这个算法的主要步骤如下: 首先，我们初始化系统参数，包括螺距的搜索范围、龙头和龙身的长度、把手间距等。然后，我们遍历所有的螺距来搜索满足条件的螺距。在每次迭代中，我们都会模拟舞龙队的盘入过程，检查第一次发生碰撞时龙头前把手是否位于调头空间之内，并根据结果调整螺距的搜索范围。这个过程会一直持续，直到找到满足条件的最小螺距或达到预设的迭代次数上限。具体来说，算法的步骤可以分为以下几步。

算法 3：迭代优化调整螺距：

S1：初始化参数：初始化碰撞时间，给出螺距初始值为 0.40，设定螺距增加的步长为 0.01m，给出调头空间的半径 R=4.5m，定义最大螺距为 1.0m，防止无限循环。开始主循环。

S2：在每一个循环节中，我们会进入一个子循环体，从时刻 0 开始不断前进，调用之前问题求得的函数来计算每时刻所有把手的位置，调用算法 3 中封装函数体“叉积法”检测当前位置是否发生了碰撞。若检测成功，则跳出子循环体。

S3：判断当前龙头前把手位置是否位于调头空间内。若判断成功，则输出当前的螺距，结束算法。反之，将当前螺距增加一个步长，回到步骤 **S2**。

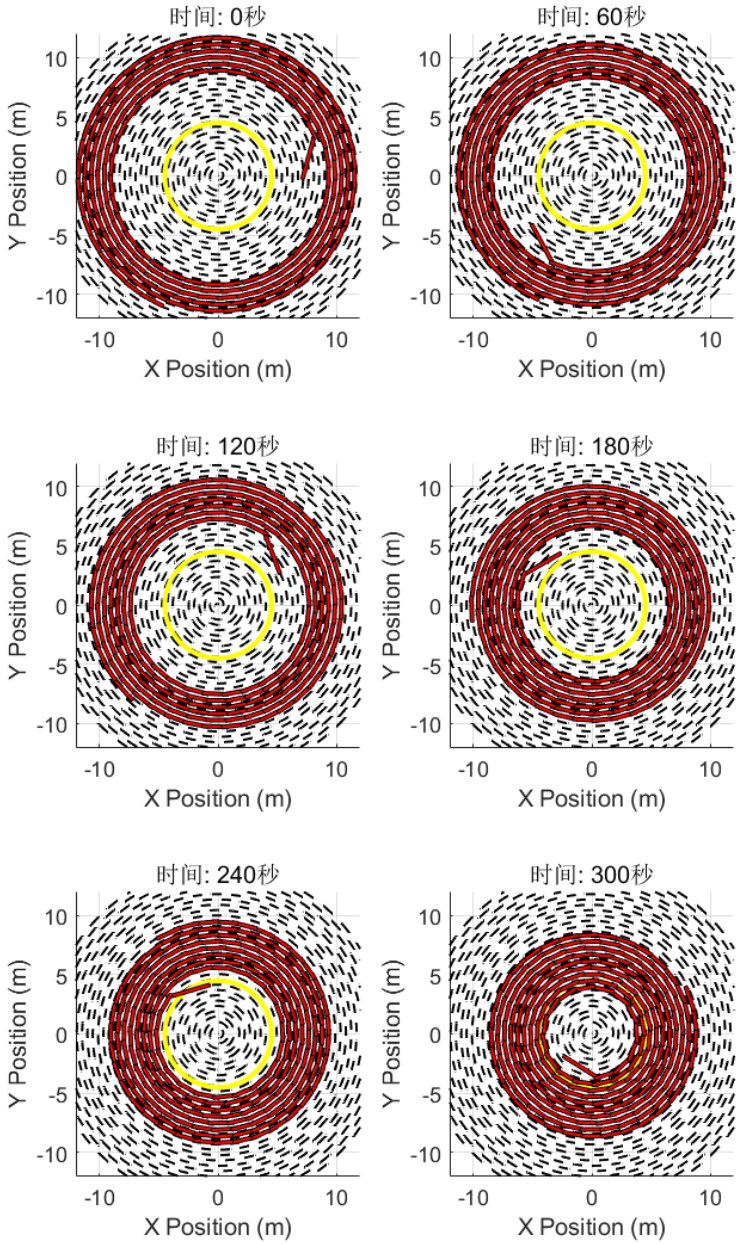
通过调用算法 3，我们求出 p_{min}=0.45m, 即当 p < 0.45m 时，龙头前把手在还未进入调头空间时就已经由于碰撞而停止前进了。

5.3.3 问题 3 仿真分析和验证

将当前螺距设置为 0.45m，我们从 0 时刻开始模拟各板凳龙的运动。中间过程如图 10 所示，可以看出，当螺距减少时，龙头的有效长度 L_{eff1} 和龙身有效长度 L_{eff2} 的差异被放大。具体表现在龙头前把手同龙头后把手之间的角位移增加，两个把手之间相隔了多圈螺旋线，而后面的龙身把手则依旧紧密。同时，从图中可以看出，碰撞发生在 180s 到 240s 之间。通过缩小迭代的步长，我们可以求出第一次碰撞具体发生在 227.41s：龙头左前把手同第十六节龙身发生碰撞。

在 227.41s 碰撞时刻的局部示意图如图 11 所示，从中可以验证，此时龙头左前角同第十六节龙身的有边界发生碰撞。从仿真可以看出，我们的结果是准确的，且具有较高的精度。

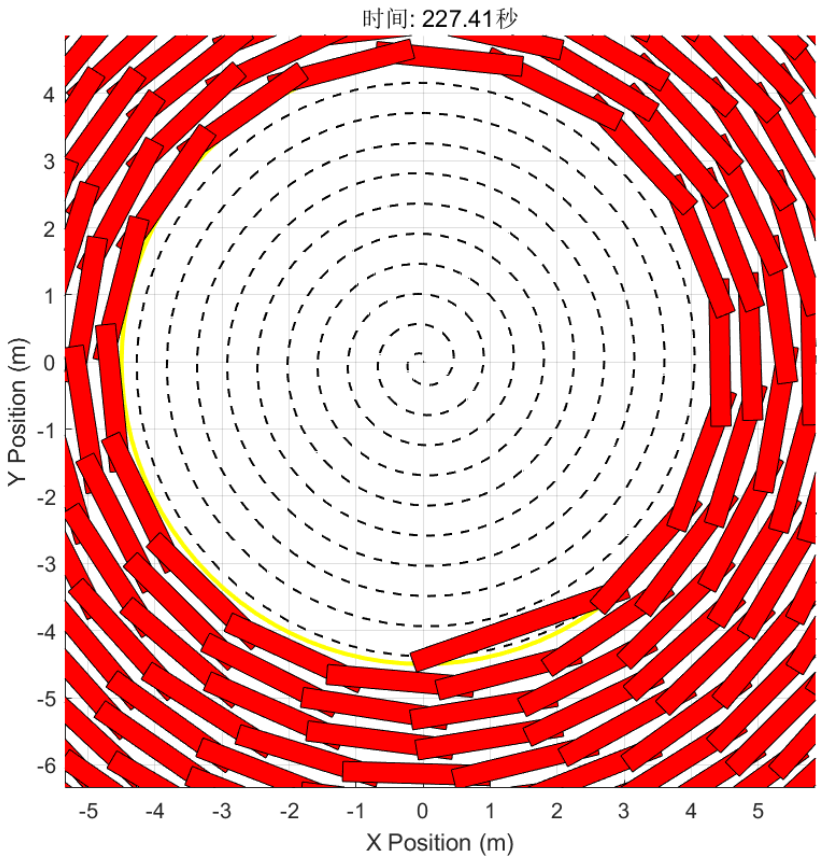
图 10　不同时刻相对于调头边界的示意图



5.4 问题 4 模型建立与求解

对于给定的掉头半径与两端掉头圆弧的半径比值，可以列出非线性方程，通过赋予计算机若干初始解的方法不断优化进而得到全局最优解。为了尽可能减小“板凳龙”在掉头圆内的掉头路程，可以放开两端掉头圆弧半径比值为 2 的严格限制，这样使得原本只存在最优解（数值解）的方程问题释放了一个自由度成为了一个动态规划问题，通过

图 11　碰撞时刻局部示意图



结合计算机数值求解与动态路径规划与优化，可以得到使得掉头路程尽可能短的两端掉头圆弧的数值解。该问题的关键在于如何设计一个更优的掉头曲线，在保证掉头过程更加高效的情况下同时保证运动的连续性和解的精确性。

5.4.1 严格限制半径比值 1:2 的掉头圆弧方程求解

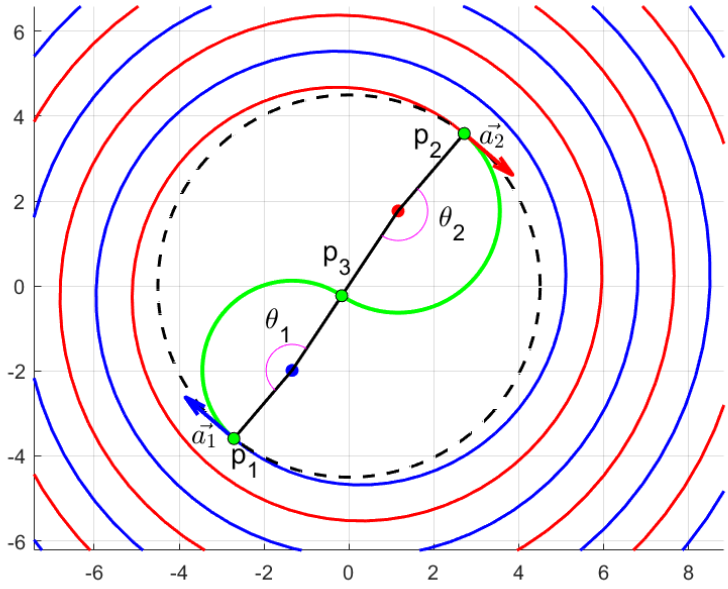
示例情形如图 12 所示。题目中已经给定了螺距大小为 $p = 1.7m$，相应地，盘入螺线轨迹为：

$$r(\theta) = \frac{1.7\theta}{2\pi} \tag{35}$$

又因为掉头区域半径为 $R = 4.5m$，故代入求解得到“板凳龙”盘入到掉头区域时的位置 $P_1(x_1, y_1)$ 极坐标为 $(4.5, 16.630)$，即角位移 $\theta_0 = 16.630$，通过平面直角坐标系投影得到 P_1 平面直角坐标为 $(-2.7119, -3.5911)$。根据盘出螺线与盘入螺线的中心对称性质，得到“板凳龙”盘出螺线和掉头区域的交点 $P_2(x_2, y_2)$ 相应坐标依次 $(2.7119, 3.5911)$。

下面设出各种未知数，并根据题目所给条件列出约束方程：掉头圆弧所对应的圆心坐标 O_1, O_2，半径分别为 (x_{o1}, y_{o1}) ,(x_{o2}, y_{o2}) 和 r_1, r_2，设两端圆弧线交点为 P_3。由于两

图 12　盘入盘出螺线与掉头圆弧示意图



端圆弧在内部相切，故它们的圆心连线长度应恰好等于半径之和：

$$d(O_1, O_2) = \sqrt{(x_{o1} - x_{o2})^2 + (y_{o1} - y_{o2})^2} = r_1 + r_2 \tag{36}$$

两端圆弧在掉头区域的边界同盘入盘出螺线相切，故下面推导边界处的相切方程表达式。记 $\vec{a_1}$ 为把手盘入螺线在角位移 θ_1 处的切线方向向量，有：

$$\vec{a_1} = -(\frac{dx}{d\theta}, \frac{dy}{d\theta}) = -(\frac{d\frac{1.7}{2\pi}\theta\cos\theta}{d\theta}, \frac{d\frac{1.7}{2\pi}\theta\sin\theta}{d\theta}) = -\frac{1.7}{2\pi}(\cos\theta_0 - \theta_0\sin\theta_0, \sin\theta_0 + \theta_0\cos\theta_0) \tag{37}$$

这里由于把手顺时针向原点盘入，故描述把手在该点方向向量时需要添加负号。根据圆的相关知识，圆 $O_1 : (x - x_{O1})^2 + (y - y + O1) = r_1^2$ 在 $\theta_0, P_1(x_1, y_1)$ 的切线方向向量为 $\vec{b_1} = (y_1 - y_{O1}, x_{O1} - x_1)$。由于在该点两条曲线相切，因此它们的方向向量平行，即 $\vec{a_1} \parallel \vec{b_1}$，相应坐标表达式为：

$$\frac{x_{O1} - x_1}{y_1 - y_{O1}} = \frac{\sin\theta_0 + \theta_0\cos\theta_0}{\cos\theta_0 - \theta_0\sin\theta_0} \tag{38}$$

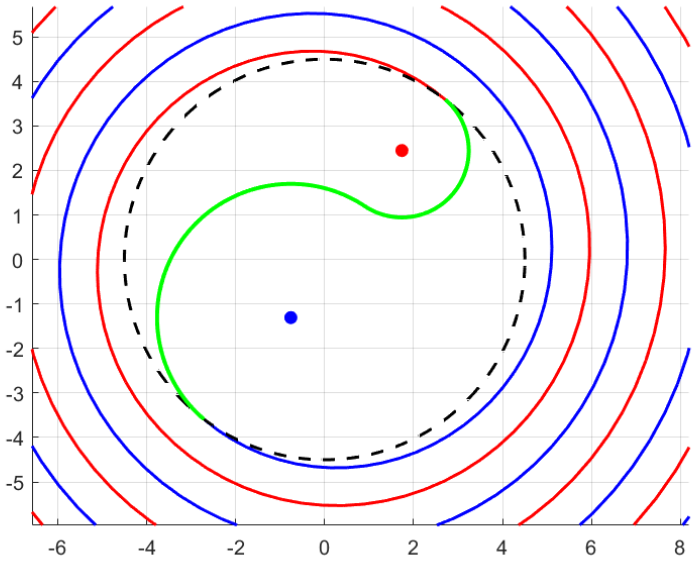
同样的，对于盘出螺线我们也有平行的结论：

$$\vec{a_2} = -\vec{a_1} = \frac{1.7}{2\pi}(-\cos\theta_0 + \theta_0\sin\theta_0, -\sin\theta_0 - \theta_0\cos\theta_0) \tag{39}$$

$$\vec{b_2} = (-y2 - y_{O2}, x_1 + x_{O2}) \parallel \vec{a_2} \tag{40}$$

$$-\frac{x_{O2} + x_1}{y_2 + y_{O2}} = \frac{-\sin\theta_0 - \theta_0\cos\theta_0}{-\cos\theta_0 + \theta_0\sin\theta_0} \tag{41}$$

图 13　半径比 1:2 的情形



又因为 P_1, P_2 本身分别位于 O_1, O_2 上，有：

$$(x_1 - x_{o1})^2 + (y_1 - y_{O1})^2 = r_1^2 \tag{42}$$

$$(x_2 - x_{o2})^2 + (y_2 - y_{O2})^2 = r_2^2 \tag{43}$$

再加上题目给出的有关半径的严格限制：

$$r_1 = 2r_2 \tag{44}$$

到现在为止，我们已经根据几何关系得出了所有的约束方程，现在结合方程与未知数个数来判断方程的类别。

我们选取式 (36) (38) (41) (42) (43) (44) 组成方程组，这里的未知数由 $x_{O1}, y_{O1}, x_{O2}, y_{O2}, r1, r2$ 共计 6 个组成，因此我们可以判断，理论上该方程由最优的唯一解。由于方程组的类型为非线性方程组，可行的求解方法只有只有设定若干有效初始解，通过计算机求解器从这些初始解出发，不断向周围优化迭代循环，直至找到一个最优化的数值解。

经过计算机求解器求解，得到了一组较为精确的数值解：

$$\begin{cases} \text{圆}O_1 \quad (-0.7600, -1.3058) \quad r_1 = 3.0054 \quad \theta_1 = 3.0215 \\ \text{圆}O_2 \quad (1.7358, 2.4485) \quad r_2 = 1.5027 \quad \theta_2 = 3.0215 \end{cases} \tag{45}$$

可视化结果如图 13 所示。

这里还有个有趣的现象，那就是半径比在某个范围内变化时，大部分有 $\theta_1 = \theta_2$，例如，当半径比 $\alpha \in [1.5, 3]$ 时（$\alpha = r_2/r_1$ 或 r_1/r_2），经过离散检验，$\theta_1 = \theta_2 = 3.0215$ 成立的可能性很高。

5.4.2 放开半径比值限制优化最短路径

如果要进一步优化圆弧，使得掉头的路程长度最小，唯一的选择是放开对半径比值严格为 1:2 的限制，即我们的方程组除去了式(44)，剩下式(36) (38) (41) (42) (43)。少去的方程释放了一个自由度，使得该问题变成了在给定约束条件下的一个非线性规划问题。如图 12 所示，掉头曲线长度表达式与规划目标为:

$$\min \quad S = \theta_1 r_1 + \theta_2 r_2 \tag{46}$$

通过对未知元和约束方程个数分析我们知道，这个非线性动态规划问题中一共涉及 4 个未知元，而与这些未知元相关的方程个数只有 3 个，因此只有一个自由度，但方程解的个数变成了无穷。

仍然延续上一小节中的各个字母变量符号。由于两个螺旋线相切于 P_3，可以给出 P_3 的两种不同表达式，最终令其相等。由题意可知，圆 O_1 在 P_1 的切线向量为 $\vec{a_1}$，现在将方向向量 $\vec{a_1}$ 顺时针方向旋转 $90°$，根据坐标顺时针旋转 ϕ 角公式:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{47}$$

旋转后的向量 $\vec{a_1}$ 是 $\overrightarrow{a_{irot}}$:

$$\overrightarrow{a_{irot}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \vec{a_1} \tag{48}$$

容易知道 $\overrightarrow{P_1O_1}$ 同 $\overrightarrow{a_{irot}}$ 方向相同。又因为圆 O_1 半径是 r_1，因此有:

$$\overrightarrow{P_1O_1} = r_1 \cdot \frac{\overrightarrow{a_{irot}}}{|\overrightarrow{a_{irot}}|} \tag{49}$$

通过这个方程可以求出对应的点坐标，我们通俗地表示为:

$$O_1 = P_1 + \overrightarrow{P_1O_1} \tag{50}$$

由于 $\overrightarrow{O_1P_3}$ 为 $\overrightarrow{O_1P_1}$ 顺时针旋转 θ_1 得到，因此有:

$$\overrightarrow{O_1P_3} = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix} (-\overrightarrow{P_1O_1}) \tag{51}$$

该式子可以得到 P_3 的坐标表达式:

$$P_3 = O_1 + \overrightarrow{O_1P_3} \tag{52}$$

同理对于盘出曲线，可以给出 P_3 坐标的另一种坐标表达形式 P_3'。依照上面的原理可以求出 $\overrightarrow{O_2\,P_3'}$ 表达式，相应的可以得到 P_3' 另外一种坐标表示式:

$$P_3' = O_2 + \overrightarrow{O_2\,P_3'} \tag{53}$$

然后根据相切条件，即圆 O_1 和圆 O_2 相切于 $P_3(P_3')$ ，因此有：

$$\begin{cases} P_{3x} = P'_{3x} \\ P_{3y} = P'_{3y} \end{cases} \tag{54}$$

观察 $\overrightarrow{O_1P_3}$ 和 $\overrightarrow{O_2P_3}$，它们方向相反且位于同一条直线上，因此其对应的单位向量点乘结果为-1。

$$\frac{\overrightarrow{O_1P_3}}{|\overrightarrow{O_1P_3}|} \cdot \frac{\overrightarrow{O_2P_3}}{|\overrightarrow{O_2P_3}|} = -1 \tag{55}$$

通过上面的分析，可以将求解最短路径弧长的非线性最优规划问题规划为以下方程组：

$$\begin{aligned} &\min \quad S = \theta_1 r_1 + \theta_2 r_2 \\ &\text{s.t.} \quad \begin{cases} P_{3x} = P'_{3x} \\ P_{3y} = P'_{3y} \\ \frac{\overrightarrow{O_1P_3}}{|\overrightarrow{O_1P_3}|} \cdot \frac{\overrightarrow{O_2P_3}}{|\overrightarrow{O_2P_3}|} = -1 \end{cases} \end{aligned} \tag{56}$$

为了解决这个非线性最优规划问题，调用计算机求解器关于该类型问题的库函数求得两组最优结果的数值解，这两组解互为中心对称情形解为：

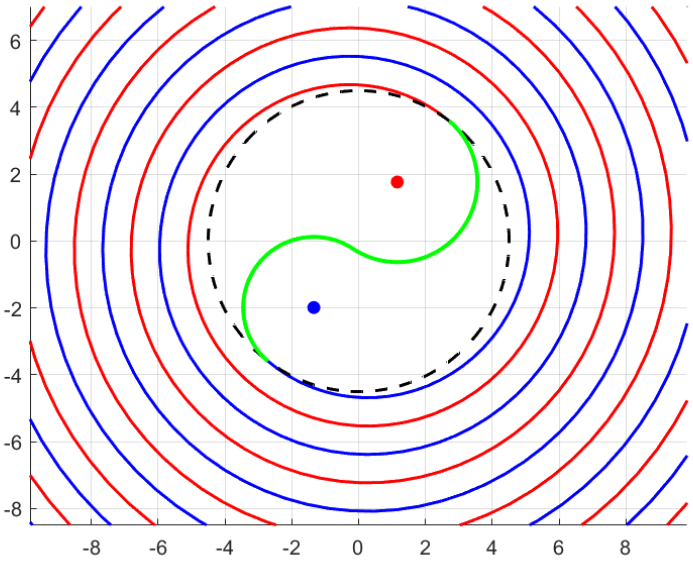
$$\begin{cases} \text{圆}O_1 \quad (-1.3404, -1.9852) \quad r_1 = 2.1118 \quad \theta_1 = 3.0215 \\ \text{圆}O_2 \quad (1.1556, 1.7689) \quad r_2 = 2.3963 \quad \theta_2 = 3.0215 \\ \alpha = r_1/r_2 = 0.883862 \\ S_min = \theta_1 r_1 + \theta_2 r_2 = 13.62122415 \end{cases} \tag{57}$$

$$\begin{cases} \text{圆}O_1 \quad (1.1556, 1.7689) \quad r_1 = 2.3963 \quad \theta_1 = 3.0215 \\ \text{圆}O_2 \quad (-1.3404, -1.9852) \quad r_2 = 2.1118 \quad \theta_2 = 3.0215 \\ \alpha = r_1/r_2 = 1.131396 \\ S_min = \theta_1 r_1 + \theta_2 r_2 = 13.62122415 \end{cases} \tag{58}$$

由于这两种解完全中心对称，可以视同完全等效，因此在求解下面问题的时候不妨使用数值解 (57)式。这一非线性规划优化问题解的可视化如图 14 ，同样使用数值解(57)式

在该问题中，同样需要求出的是 P_1, P_2, P_3 坐标，以及向量 $\overrightarrow{O_1P_1}, \overrightarrow{O_1P_3}, \overrightarrow{O_2P_2}, \overrightarrow{O_2P_3}$ 供下一小问求解使用。

图 14　动态调整半径比值后的最优解



5.4.3 给定路径下龙头分时段运动状态分析

结合问题 1 和问题 2，要求解所有把手的实时运动状态，问题的关键就在于求出龙头把手的实时位置坐标。一旦求出了龙头把手的实时坐标，就可以通过循环迭代依次从龙头把手开始遍历后续所有把手，得到后续所有把手的位置坐标，而一旦求出了所有把手的位置坐标，通过求解当前时刻 t 之后一段微小时间 δt 后所有把手的位置状态，作差值并将 δt 内的平均速度近似为时刻 t 时的瞬时速度，就可以得到所有把手实时速度关于时间 t 的函数。下面我们着重来研究龙头把手在任意时刻的位置信息。

根据题目要求，龙头把手刚刚到达 P_1 的时刻被重新记为 $t_0 = 0$，设龙头把手刚刚离开第一段圆弧的时刻是 t_{n1}，刚刚离开第二段圆弧的时刻是 t_{n2}，在这之后龙头就沿着盘出螺线逐渐盘出。

根据刚才的分析，龙头盘入掉头盘出的过程可以分成四个阶段如下所列：

1. **第一阶段：** $t < t_0$，龙头在盘入螺线上逐渐向中心盘入。
2. **第二阶段：** $t_0 \le t < t_{n1}$，龙头走出盘入螺线进入第一条圆弧线未进入第二条圆弧线。
3. **第三阶段：** $t_{n1} \le t < t_{n2}$，龙头出第一条圆弧线进入第二条圆弧线未进入盘出螺线。
4. **第四阶段：** $t \ge t_{n2}$，龙头出第二条圆弧线进入盘出螺线。

下面对四个阶段分别进行分析，从而得到龙头的实时位置坐标方程，从而可以进行仿真模拟分析。

第一阶段： $t < t_0 = 0$ 时盘入螺线方程保持不变：

$$C_1 : r(\theta) = \frac{1.7\theta}{2\pi} \tag{59}$$

仿照第一问，将运动时的切向速度近似成速度，从而得到微分方程：

$$\omega = \frac{v(t)}{r(t)} = \frac{d\theta}{dt} \tag{60}$$

注意，本题的初始时间点 $t = 0$ 时，质点角位移初始值是 θ_0（已经在上一小节中解出）。相比于问题 1，这里需要更改定积分上下限，有：

$$\int_0^t \frac{2\pi}{1.7}\, dt = \int_{\theta_0}^{\theta} \theta\, d\theta \tag{61}$$

考虑到 $t < t_0$，故积分结果还有 t 的项前面需要添加负号，即：

$$\theta(t) = \sqrt{\theta_0^2 - \frac{2\pi t}{1.7}} \tag{62}$$

通过平面投影得到其平面坐标：

$$\begin{cases} x_1(t) = 1.7\theta(t)\cos\theta(t)/2\pi \\ y_1(t) = 1.7\theta(t)\sin\theta(t)/2\pi \end{cases} \tag{63}$$

由于质点线速度恒为 1m/s，因此龙头走完两段弧线分别要

$$\begin{cases} t_n1 = \theta_1 r_1/v = \theta_1 r_1 \\ t_n2 = (\theta_1 r_1 + \theta_2 r_2)/v = \theta_1 r_1 + \theta_2 r_2 \end{cases} \tag{64}$$

第二阶段： $t_0 \le t < t_{n1}$，龙头在第一段圆弧上。假设圆弧位于点 h_1 处，则弧线长及对应圆心角 $\Delta\theta_1$ 分别是

$$\begin{cases} \overset{\frown}{P_1h_1} = vt = t \\ \Delta\theta_1 = \overset{\frown}{P_1b_1}/r_1 = t/r_1 \end{cases} \tag{65}$$

因此 $\overrightarrow{O_1h_1}$ 是由 $\overrightarrow{O_1P_1}$ 顺时针旋转 $\Delta\theta_1$ 得到。根据顺时针坐标旋转公式，可以得到位置坐标关系：

$$\begin{cases} \overrightarrow{O_1h_1} = \begin{bmatrix} \cos\Delta\theta_1 & \sin\Delta\theta_1 \\ -\sin\Delta\theta_1 & \cos\Delta\theta_1 \end{bmatrix} \cdot \overrightarrow{O_1P_1} \\ h_1 = O_1 + \overrightarrow{O_1h_1} \\ x_1(t) = h_{ix}, y_1(t) = h_{1y} \end{cases} \tag{66}$$

第三阶段： $t_{n1} \le t < t_{n2}$，龙头在第二条圆弧线上，设某时刻龙头位于 h_2 处。仿照上面的分析，显然有方程组：

$$\begin{cases} \overset{\frown}{P_3h_2} = v(t-t_{n1}) = t - t_{n1}, \\ \Delta\theta_2 = \overset{\frown}{P_3h_2}/r_2 = t - t_{n1}/r_2, \\ \overrightarrow{O_2h_2} = \begin{bmatrix} \cos\Delta\theta_2 & -\sin\Delta\theta_2 \\ \sin\Delta\theta_2 & \cos\Delta\theta_2 \end{bmatrix} \cdot \overrightarrow{O_2P_3}, \\ h_2 = O_2 + \overrightarrow{O_2h_2}, \\ x_1(t) = h_{2x}, \quad y_1(t) = h_{2y}. \end{cases} \tag{67}$$

第四阶段： $t \ge t_{n2}$，龙头进入盘出螺线。根据盘入螺线和盘出螺线的中心对称性质，可以求出相应盘入螺线上的点，再关于原点取中心对称。设某时刻龙头位于离开盘出螺线前进 $t - t_{n2}$ 时刻位于 h'，根据 h 与 h' 的中心对称性可知，有：

$$\begin{cases} h'_x = x_1(t_n - t) \\ h'_y = y_1(t_n - t) \\ h_x = -h'_x = -x_1(t_n - t) \\ h_y = -h'_y = -y_1(t_n - t) \end{cases} \tag{68}$$

求解上述方程组：

$$\begin{cases} x_1(t) = -x_1(t_n - t) \\ y_1(t) = -y_1(t_n - t) \end{cases} \tag{69}$$

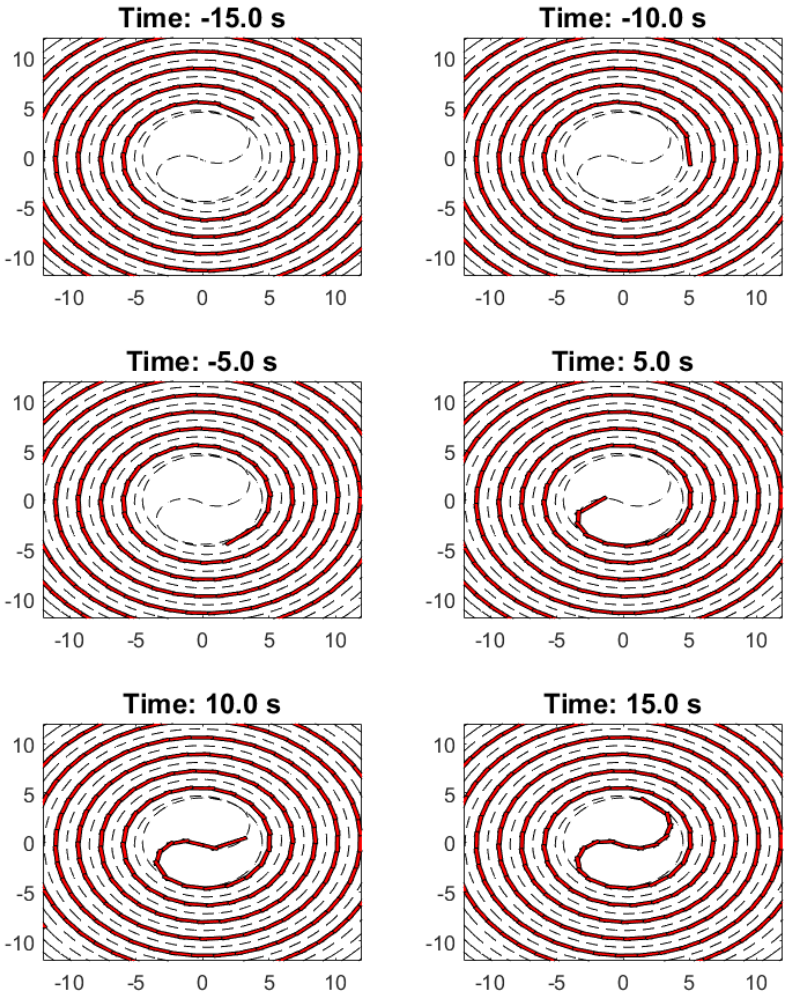
综上所述，龙头把手关于时间 t 的坐标 $(x_1(t), y_1(t))$ 函数如下：

$$\begin{cases} x_1(t) = 1.7\theta(t)\cos\theta(t)/2\pi & t < t_0 \\ x_1(t) = h_{ix} & t_0 \le t < t_{n1} \\ x_1(t) = h_{2x} & t_{n1} \le t < t_{n2} \\ x_1(t) = -x_1(t_n - t) & t \ge t_{n2} \end{cases} \quad \begin{cases} y_1(t) = 1.7\theta(t)\sin\theta(t)/2\pi & t < t_0 \\ y_1(t) = h_{1y} & t_0 \le t < t_{n1} \\ y_1(t) = h_{2y} & t_{n1} \le t < t_{n2} \\ y_1(t) = -y_1(t_n - t) & t \ge t_{n2} \end{cases} \tag{70}$$

5.4.4 递推法求解任意时刻所有把手的运动状态

正如问题 1 的思路，只要已知了龙头把手的位置状态，就可以通过一个循环遍历后续所有把手，得到在任意时刻下所有把手的位置状态。对于这些位置状态，赋予时间 t 一个微小的扰动 δt，用在 δt 内发生微小位移的平均速度来近似替代在 t 的时刻的平均速度。

图 15　调头前后重要时刻图



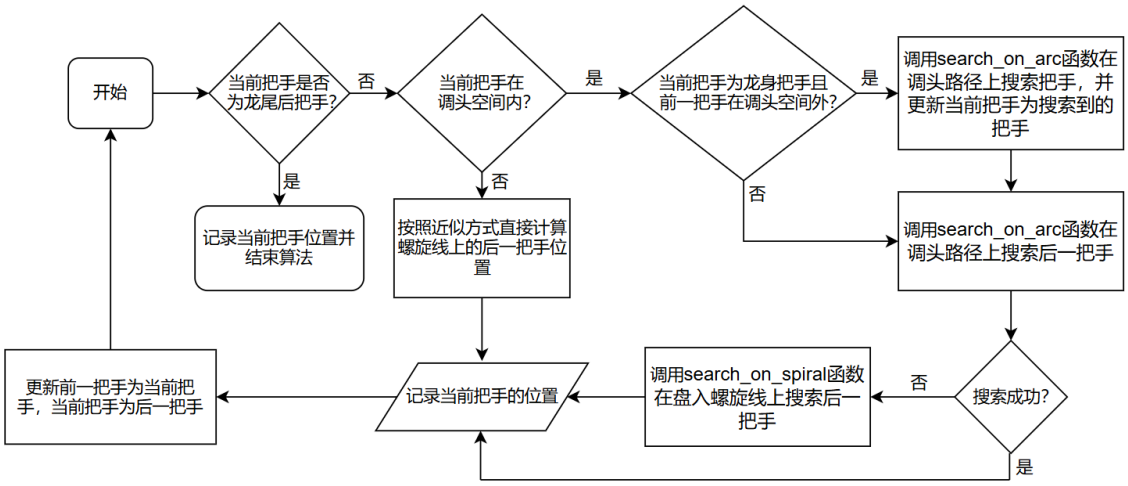
与算法 1（循环近似法求解下一把手的位置坐标）不同的是，对于下一个把手的计算，由于掉头半径的存在，前一把手和后一把手可能不再同一条曲线上，因此在迭代时还需要对把手所处曲线的类型进行判断，下面我们介绍一种新算法 4。如图 16 所示，算法 4 解决了在复杂路径下后续把手的分类计算问题，在通过代码实现的过程中，可以封装成一个函数体，嵌套至算法 1 中参加循环迭代过程。

首先需要介绍算法 4 所需要的几个相关辅助函数：

位置判断函数 1：接受当前把手的位置坐标，计算其矢径长度并与掉头半径 R 进行比较从而判断当前把手是否在掉头区域内，返回一个布尔值。是则传递并继续调用近似螺旋搜索函数，否则传递并调用 R 内搜索方向判断函数 3。

近似螺旋搜索函数 2：对应图中函数 search_on_spiral, 它接受当前把手的坐标，通过近似螺旋法求解下一把手对应角位移大小，相应计算出其位置坐标。

图 16　算法流程图分析



R 内搜索方向判断函数 3：对应图中函数 search_on_arc, 它接受当前把手 i 的坐标 (x_i, y_i)，若把手 i 调头空间内部，可以肯定把手 i 一定在掉头区域中的两段圆弧上。由于区域内有两段圆弧，时针方向不同。可以分别计算 (x_i, y_i) 与 (x_1, y_1) 和 (x_2, y_2) 的距离，比较与各自对应半径长的差值。若与圆 O_2 的差值更大，则说明把手在圆 O_1 上，返回布尔值并传递给“圆弧内搜索函数 4A”，反之传递给“圆弧内搜索函数 4B”。如果把手 i 不在调头空间内部，则直接调用圆弧内搜索函数 4A，从盘出螺旋线和调头空间的交点沿第二段圆弧进行搜索。

圆弧内搜索函数 4A：顺时针开始搜索第 i+1 个把手的位置，连接该把手与圆心 O_1，取 $\Delta\varphi$，从一个微小的步长开始递增 $\Delta\varphi$，直至两个把手的长度正好是 L，然后使用圆的相关知识求解具体点坐标。

圆弧内搜索函数 4B：逆时针开始搜索第 i+1 个把手的位置，原理同上。

有了以上五个函数，我们可以将相同功能的代码封装起来，只保留接口。这样也简化了该算法的流程。

算法 4：复杂路径下后续把手的分类计算模型：

输入：上一轮循环中求得的各个龙头任意时刻的位置坐标。

输出：“板凳龙”各个把手任意时刻的坐标（一旦坐标已知，就可以求出实时速度并作出模拟仿真动画）

S1：从龙头把手出发，依次向后迭代循环。首先传入“位置判断函数 1”，按照上面函数间的相互关系依次调用。最终在“位置判断函数 1”，“圆弧内搜索函数 4A”和“圆弧内搜索函数 4B”这三个函数中停止并全部返回布尔正值。

S2：判断当前把手是否为最后一个把手，如果是，则停止循环。

在已知所有把手的运动状态后，不难对其进行实时过程仿真模拟，图 15 为调头前后重要时刻的模拟图，从中可以看出，根据我们的算法求出的位置极好地拟合了所有板

凳龙的正确位置。

5.5 问题 5 模型建立与求解

问题五要求我们确定舞龙队沿问题四设定的路径行进时，龙头的最大行进速度，使得舞龙队各把手的速度均不超过 2m/s。因此，这个问题的本质是一个约束优化问题，我们需要在保证所有板凳速度不超过限制的前提下，最大化龙头的行进速度。

首先，我们需要明确问题的约束条件。龙头的行进速度是我们的优化目标，而所有板凳 (包括龙头) 的速度不超过 2m/s 是我们的约束条件。这意味着我们需要建立一个模型，能够根据龙头的行进速度计算出板凳龙所有部分的速度。其次，我们需要考虑路径的几何特性如何影响板凳龙的运动。问题四中设定的路径是一个 S 形曲线，由两段圆弧组成。在这种曲线路径上，即使龙头保持恒定速度，其他部分的速度也会因为路径曲率的变化而发生变化。我们需要仔细分析这种关系，以确保在整个运动过程中所有部分的速度都不会超过限制。再者，我们需要考虑板凳龙的结构特性。板凳之间的连接方式和相对位置关系会影响速度的传递。特别是，由于板凳龙是一个长链状结构，靠近尾部的板凳会经历更大的速度变化，这成为限制最大速度的关键因素。

在问题 4 的基础上，我们将 v 作为变量，将问题 4 中求出的函数转化为关于 t，v 的函数。

5.5.1 问题 5 模型的分析与求解

算法 5：迭代求速度：

S1：初始化参数：记时间步长为 1 秒，终止时间为 100 秒，确保结束时间足够大而不会遗漏结果。

S2：进入第一轮循环，每一个循环节将会对应一个时刻。调用之前问题求得的函数来计算每时刻所有把手的位置。具体来说，先计算龙头前把手的位置，再从龙头开始依次计算每个把手的位置。

S3：第一轮循环结束，即将进入第二轮循环。每一个循环节对应一个板凳的判断过程，根据当前把手是第一个把手还是后续把手确定距离，判断当前把手是否在调头空间内。若不在当前空间内，按照近似方式计算下一个把手。

S4：按照第一段圆弧、第二段圆弧以及盘出对把手的运动进行分类和筛选。调头过程中，搜寻在螺旋线上的下一个点。设定一个小的角度步长，如果距离达到了目标距离，则返回当前螺旋线上的点。

S5：定义初始龙头速度和步长，龙头速度初始为 1m/s。开始迭代速度，从 1m/s 逐步增加，获取所有把手的位置信息，检查是否有把手速度超过了 2m/s，若没有，则继续循环；若出现这种情况，则跳出循环，并且输出此时的龙头速度，即最终的最大速度。

问题 4 导出的函数：

$$\mathbf{X(t)} = \begin{bmatrix} x_1(t) \\ x_3(t) \\ \vdots \\ x_n(t) \end{bmatrix} \tag{71}$$

$$\mathbf{Y(t)} = \begin{bmatrix} y_1(t) \\ y_3(t) \\ \vdots \\ y_n(t) \end{bmatrix} \tag{72}$$

上述两个函数是关于 t，v 的函数，即可以写成：

$$\mathbf{X}(t) = \mathbf{X}(t, v_{head}) \tag{73}$$

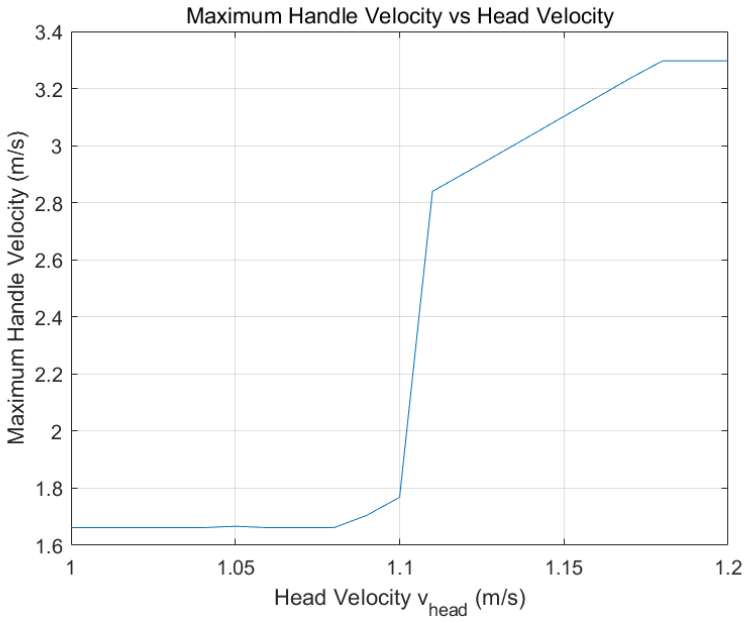
$$\mathbf{Y}(t) = \mathbf{Y}(t, v_{head}) \tag{74}$$

同理可得：

$$\mathbf{v}(t) = v(t, v_{head}) \tag{75}$$

由问题 4 所导出的结果可知，当 v_{head}=1m/s 时，所有把手在 [-100,100] 这一区间上的运动速度均不超过 2m/s，因此，我们不妨选取 v_{head}=1m/s 为起点，以 $dv = 0.01m/s$ 为步长不断增加。直到某一点 $v(t, v_{head}) > 2m/s$ 时，有最大值 $v = v_{head} - dv$。经过多次迭代，我们计算出的结果为 1.10m/s。

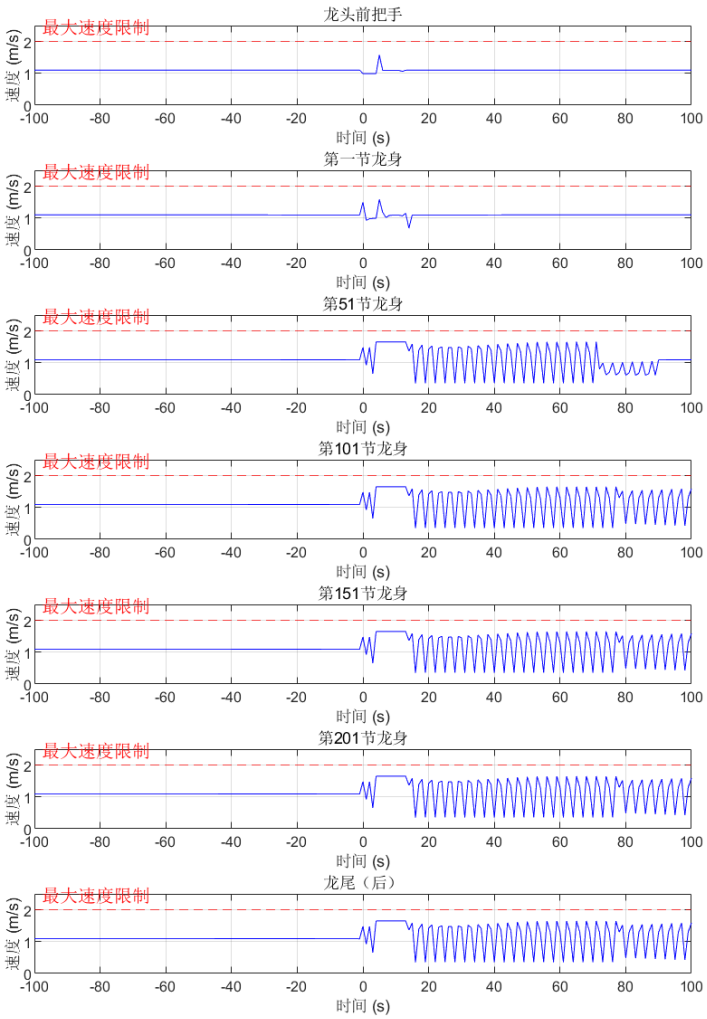
图 17　龙身最大速度随龙头速度的变化曲线



5.5.2 问题 5 结果的验证与处理

我们结合物理学知识，利用 MATLAB 进行模拟仿真，绘制出龙身最大速度与龙头最大速度之间的关系曲线，如图 17 所见。可以看出，随着龙头最大速度的增加，龙身的最大速度也不断增加，这是符合真实的物理情景的。值得注意的是，当龙头最大速度达到 1.1m/s 时，龙身最大速度立刻发生了突跃，迅速突破 2m/s 的上限。因此，我们确定龙头的最大速度为 1.1m/s.

图 18　龙头速度最大时，七节龙身的速度变化曲线



确定龙头的最大行进速度后，我们选取了七节龙身，并研究其速度变化特征，如图 18 所示。可以发现，在 0s（初始时刻即将运动）时，七节龙身的速度均约为 1m/s，这与问题 1 和问题 2 的给定条件不谋而合。当龙头达到最大速度后，七节龙身都突然获得了一定的速度。不同之处在于，龙头前把手与第一节龙身的速度很快便回到 1m/s 附近，

然后不再变动；第 51、101、151、201 节龙身的速度则开始以 1m/s 为准线上下振荡起来。由之前的分析不难联想到，这与板凳之间的连锁运动有着密不可分的关系。

六、 模型评估

6.1 模型误差及其校正

6.1.1 问题 1 相关误差与修正方法

1. 由于“板凳龙”的把手沿着螺旋线运动，而螺旋线并不是一个严格的圆弧，这就导致了其线速度和螺旋矢径之间的夹角并不是一个严格的直角，必然存在几度或分秒级别的误差。我们将这样的螺旋运动在各个时刻下的状态与其在同样半径的圆弧运动作了一步近似。近似之后，我们便可以忽略径向速度，并将圆周的切向速度等效为螺旋运动的线速度。
2. 我们使用余弦法求后续龙身把手位置，得到的迭代方程属于超越方程，无法求出其解析解。因此我们只能通过尽可能优化的近似手段来求出近似解。我们令 $\Delta\theta$ 从 0 开始逐渐增加步长，遍历到 L 的全过程。随着迭代次数的增加，计算更外围的把手时，其误差更大，而当时刻 t 逐渐增加后，随着把手向内运动，其精度会越来越高。
3. 除了余弦法之外，我们另外还使用了近似圆弧法求解后续龙身把手位置，即处理半径较大的螺线时，我们可以将其等效为圆弧处理。这样做的合理性是，转过一圈时，半径衰减的幅度所对应的数量级远远小于半径所对应的数量级，误差约为小数点后四位，精度较好，并且通过近似圆弧法相较于余弦法更容易被计算机实现。
4. 我们使用差分法来求解把手的速度。由于无法对全过程进行连续的模拟仿真，我们只能取多个相隔很近、分立的值。我们重新设置一个关于时刻 t 新的步长 δt=0.0001。用从这一时刻开始的微小时间内的平均速度近似代替在该时刻的瞬时速度。这是一种非常经典却又精度较高的近似方法。

6.1.2 问题 2,3 相关误差与修正方法

1. 问题 2 是基于问题 1 的仿真模拟所给出的实时数据来解决的，因此问题 2 的求解在先天上便存在有问题 1 带来的全部问题。例如由于龙头把手的坐标是由切向速度近似成线速度得到的，而这一数据又在后面作为循环头，开启若干轮遍历，故一开始的误差会随着时间逐渐累积。
2. 问题 2 需要求解出龙头后续 18 个板凳的 4 个边界点的坐标，由于计算这些坐标值需要大量中间量，例如计算方向向量，单位方向向量，单位切线向量，这个计算过程会导致其中间变量的误差逐渐累积到最终数值中。
3. 受限于循环迭代操作中时间步长的大小影响，永远没有办法得到精确的碰撞时刻，

因为进行求解的只能是离散的时间状态，所以距离真实过程中连续的物体运动过程始终有一定差值。想要解决这个问题，只需要再计算机允许求解的复杂度的情况下，不断缩小其步长，逼近连续过程。

6.1.3 问题 4,5 相关误差与修正方法

1. 问题 4 在求解掉头最短路径一问误差值较小，原因在于我们没有对该非线性规划问题的任何数值或者约束条件作任何近似处理，在传入计算机后，其误差仅仅来自于编译器求解的误差，一般来说 matlab 求解器精度较高，可以达到小数点后六位。
2. 问题 4 第二小问误差相当更大了，与 1 类似，这种根据龙头节点开始循环得到的位置矩阵在经过多轮循环后误差会逐渐累积，同时时间由于取值离散化的原因导致精确时间测不准。这些误差和问题 2 类似。
3. 在这里，“板凳龙”是在由 4 条曲线拼接而成的曲线内运动，在计算相邻节点运动状态时，还需要考虑一个节点在圆弧内，一个节点在圆弧外，在涉及这样的一轮循环后，需要调用更多的函数体进行求解。同时，对于横跨两段曲线的求解过程，使用的近似螺旋在这里的计算误差更大误差更大。
4. 在最后求解实时速度时，在选取一段微小的时间时发生了较大的误差。在计算机编译时，若选取小于 0.5 的微小时间，会使得计算出的某些相邻时间内数值发生突变跃升至几千米每秒，这显然是不可能得，因此，为了同时兼顾数值的准确性与可计算型，最终取得一段微小的时间长度为 1m/s。

6.2 模型优缺点及推广

1. 优点：模型被极大地简化了，问题 1 使用近似螺旋法，将螺旋线等价为圆弧处理。在计算速度时不去探究速度精准的解析表达式，而是直接用一段微小时间内的平均速度进行近似。问题本身是一个连续型的仿真模拟问题，我们取离散的时刻值，不断缩小时间步长来趋近于连续。最后一问将问题逐步分解为若干个函数块，彼此之间相互调用，简单方便。
2. 缺点：误差较大，第一轮近似螺旋法求得的龙头坐标本身存在误差，这些误差经过一步步循环逐步累积。无法得到精确的位置坐标和速度的解析表达式，而根据物理原理，确定的过程具有确定的解，理论上其确定的解析表达式一定存在，而这里只研究数值解。不是连续类型模型，与实际有所差距。
3. 启发：交通运输过程中的车辆运动问题，更精确地认识车流量、碰撞的物理意义，从而减少交通事故的发生。对自然界中某些生物现象的研究，比如昆虫的成群移动等，推进人们对于自然界的认识。对中国传统文化的弘扬，从数学等专业角度给出了“板凳龙”舞动更加美丽的策略。

七、 附录

Listing 1: 计算问题 1 的代码并将计算结果写入到 result1.xlsx 中

```
clc, clear;
time_total = 423; % 总时间 300秒
dt = 1; % 时间步长 1秒
dt_small = 0.0001; % 用于计算速度的小时间步长
n_sections = 224; % 把手数量
v = 1; %龙头的速度

% 时间数组
time_steps = 0:dt:time_total;

% 初始化存储位置的矩阵
positions = zeros(n_sections * 2, length(time_steps));
velocity = zeros(n_sections, length(time_steps)); % 因为速度在相邻的整数秒之间计算

% 定义一个函数，输入时间t，返回t时刻的所有把手位置
function positions = calculate_positions_at_time(t)
    % 设定参数
    n_sections = 224; % 把手数量
    pitch = 0.55; % 螺距55cm
    v = 1; %龙头的速度
    distance_1 = 2.86; %龙头两个把手的距离
    distance_2 = 1.65; %龙身两个把手的距离
    theta_initial = 2 * pi * 16; % 第16圈起始角度
    % 初始化存储位置的矩阵
    positions = zeros(n_sections * 2, 1);

    % 计算当前时刻的龙头角度和半径
    theta = sqrt(theta_initial^2 - 4 * pi *v / pitch * t); % 当前时刻的角度变化
    radius = pitch * theta / (2 * pi); % 螺线半径

    % 计算龙头的位置
    x_head = radius * cos(theta); % x 位置
    y_head = radius * sin(theta); % y 位置

    % 将龙头位置保存到矩阵（第1行和第2行分别为x和y）
    positions(1) = x_head;
    positions(2) = y_head;

    % 计算每一节龙身的位置
    for i = 2:n_sections
        %两个把手之间的距离
        if (i == 2)
            distance = distance_1;
        else
            distance = distance_2;
        end
        delta_theta = distance / radius;
        theta = theta + delta_theta;
        radius = 0.55 * theta / (2 * pi);
        % 计算该节板凳的 x 和 y 坐标
        x_i = radius * cos(theta);
```

```
        y_i = radius * sin(theta);

        % 保存该节板凳的位置 (注意行号是 2i-1 和 2i)
        positions(2*i-1) = x_i;
        positions(2*i) = y_i;
    end
end

% 计算每秒钟的位置信息
for t_idx = 1:length(time_steps)
    t = time_steps(t_idx);
    positions(:, t_idx) = calculate_positions_at_time(t);
end

% 计算整数秒之间的速度，使用 t 和 t+0.0001 之间的位移来计算
for t_idx = 1:length(time_steps)
    % 当前时刻 t_now 和 t+0.001 秒时刻
    t_now = time_steps(t_idx);
    t_small = t_now + dt_small;

    % 获取所有把手在 t_now 时刻的位置
    pos_now = calculate_positions_at_time(t_now); % 返回一个 2*n_sections 大小的向量

    % 获取所有把手在 t+0.001s 时刻的位置
    pos_next = calculate_positions_at_time(t_small); % 返回一个 2*n_sections 大小的向量

    % 将 pos_now 和 pos_next 重新分为 x, y 坐标
    x_now = pos_now(1:2:end); % 当前时刻所有把手的 x 坐标
    y_now = pos_now(2:2:end); % 当前时刻所有把手的 y 坐标
    x_next = pos_next(1:2:end); % t+0.001s 时刻所有把手的 x 坐标
    y_next = pos_next(2:2:end); % t+0.001s 时刻所有把手的 y 坐标

    % 计算速度 (vx, vy)，然后计算总速度
    v_x = (x_next - x_now) / dt_small;
    v_y = (y_next - y_now) / dt_small;

    % 计算每个把手的速度并存储
    velocity(:, t_idx) = sqrt(v_x.^2 + v_y.^2);
end


%将结果保留六位小数
positions = round(positions, 6);
velocity = round(velocity, 6);

% 构建第一页行头，表示每节板凳的 x(m) 和 y(m) 坐标
headers_rows_1 = {'', '龙头x(m)', '龙头y(m)'};
for i = 1:n_sections-3
    headers_rows_1 = [headers_rows_1, {['第' num2str(i) '节龙身x(m)'], ['第' num2str(i) '节龙身y(m)']}];
end
headers_rows_1 = [headers_rows_1, {'龙尾x(m)', '龙尾y(m)'}];
headers_rows_1 = [headers_rows_1, {'龙尾（后）x(m)', '龙尾（后）y(m)'}];

% 构建第一页列头，表示每个时间点 (0s, 1s, 2s, ...)
headers_columns_1 = arrayfun(@(x) [num2str(x) 's'], time_steps, 'UniformOutput', false);

```

```
% 将 headers_rows_1 和 headers_columns_1 拼接到数据中
final_data_1 = [headers_rows_1', [headers_columns_1; num2cell(positions)]];

% 构建第二页行头
headers_rows_2 = {'', '龙头 (m/s) '};
for i = 1:n_sections-3
    headers_rows_2 = [headers_rows_2, {['第' num2str(i) '节龙身 (m/s) ']}];
end
headers_rows_2 = [headers_rows_2, {'龙尾 (m/s) '}];
headers_rows_2 = [headers_rows_2, {'龙尾 (后) (m/s) '}];

% 构建第二页列头，表示每个时间点 (1s, 2s, 3s, ...)
headers_columns_2 = arrayfun(@(x) [num2str(x) 's'], time_steps(1:end), 'UniformOutput', false);

% 将 headers_rows_2 和 headers_columns_2 拼接到数据中
final_data_2 = [headers_rows_2', [headers_columns_2; num2cell(velocity)]];

% 保存到Excel文件，位置数据到第一页，速度数据到第二页
writecell(final_data_1, 'result1.xlsx', 'Sheet', '位置');
writecell(final_data_2, 'result1.xlsx', 'Sheet', '速度');
```

Listing 2: 求解问题 2 的代码，将碰撞时间打印到控制台，并将位置，速度等信息写入到 result2.xlsx 中

```
clc; clear;
start_time = 400; %开始时间
dt = 0.01; % 默认时间步长
end_time = 440;%结束时间
dt_small = 0.000001; % 用于计算速度的小时间步长
n_sections = 224; % 把手数量
width = 0.3; % 板凳宽度30cm
offset = 0.275; % 把手到边界的距离为 0.275m
collision_time = -1; % 初始化碰撞时间

% 时间数组
time_steps = start_time:dt:end_time;

% 定义函数计算每时刻所有把手位置
function positions = calculate_positions_at_time(t)
    n_sections = 224;
    pitch = 0.55; % 螺距55cm
    v = 1; %龙头的速度
    distance_1 = 2.86; %龙头两个把手的距离
    distance_2 = 1.65; %龙身两个把手的距离
    theta_initial = 2 * pi * 16; % 初始角度
    positions = zeros(n_sections * 2, 1);

    theta = sqrt(theta_initial^2 - 4 * pi *v/ pitch * t); % 当前角度
    radius = pitch * theta / (2 * pi); % 螺旋半径

    x_head = radius * cos(theta);
    y_head = radius * sin(theta);
    positions(1) = x_head;
    positions(2) = y_head;

```

```
    for i = 2:n_sections
        if (i == 2)
            distance = distance_1;
        else
            distance = distance_2;
        end
        delta_theta = distance / radius;
        theta = theta + delta_theta;
        radius = 0.55 * theta / (2 * pi);
        x_i = radius * cos(theta);
        y_i = radius * sin(theta);
        positions(2*i-1) = x_i;
        positions(2*i) = y_i;
    end
end

% 定义辅助函数，检查点是否在矩形内
function inside = is_point_in_rectangle(px, py, rect)
    x1 = rect(1,1); y1 = rect(1,2);
    x2 = rect(2,1); y2 = rect(2,2);
    x3 = rect(3,1); y3 = rect(3,2);
    x4 = rect(4,1); y4 = rect(4,2);

    % 使用向量叉积判断点是否在矩形内
    v1 = [x2 - x1, y2 - y1]; % 向量 v1
    v2 = [x3 - x2, y3 - y2]; % 向量 v2
    v3 = [x4 - x3, y4 - y3]; % 向量 v3
    v4 = [x1 - x4, y1 - y4]; % 向量 v4

    % 点相对于每条边的叉积
    cp1 = (px - x1)*(y2 - y1) - (py - y1)*(x2 - x1); % 点在边1左侧
    cp2 = (px - x2)*(y3 - y2) - (py - y2)*(x3 - x2); % 点在边2左侧
    cp3 = (px - x3)*(y4 - y3) - (py - y3)*(x4 - x3); % 点在边3左侧
    cp4 = (px - x4)*(y1 - y4) - (py - y4)*(x1 - x4); % 点在边4左侧

    % 如果所有叉积都为正或都为负，则点在矩形内
    inside = (cp1 >= 0 && cp2 >= 0 && cp3 >= 0 && cp4 >= 0) || ...
             (cp1 <= 0 && cp2 <= 0 && cp3 <= 0 && cp4 <= 0);
end

% 主循环，只检测龙头的碰撞
for t_idx = 1:length(time_steps)
    t = time_steps(t_idx);
    positions_now = calculate_positions_at_time(t);
    % 检查螺旋中心，当 theta 接近 0 时，停止计算
    theta_2 = (2 * pi * 16)^2 - 4 * pi / 0.55 * t;
    if theta_2 < 0
        disp(['龙头在 t = ', num2str(t), ' 秒到达螺旋中心，无法继续旋入']);
        collision_time = t;
        break;
    end

    % 计算龙头的两个前角，基于前后把手的位置
    x_head_front = positions_now(1); % 龙头前把手的 x 坐标
    y_head_front = positions_now(2); % 龙头前把手的 y 坐标
    x_head_back = positions_now(3); % 龙头后把手的 x 坐标
```

```
y_head_back = positions_now(4); % 龙头后把手的 y 坐标

% 计算龙头的方向向量
dx_head = x_head_front - x_head_back;
dy_head = y_head_front - y_head_back;
length_vector_head = sqrt(dx_head^2 + dy_head^2);

% 归一化方向向量（使其长度为1）
direction_x_head = dx_head / length_vector_head;
direction_y_head = dy_head / length_vector_head;

% 垂直方向向量（用于计算龙头的左右两侧）
perpendicular_x_head = -direction_y_head;
perpendicular_y_head = direction_x_head;

% 计算龙头的前左和后左角坐标，考虑宽度和偏移量
x_head_front_left = x_head_front + (width / 2) * perpendicular_x_head + offset * direction_x_head;
y_head_front_left = y_head_front + (width / 2) * perpendicular_y_head + offset * direction_y_head;

x_head_back_left = x_head_front + (width / 2) * perpendicular_x_head - offset * direction_x_head;
y_head_back_left = y_head_front + (width / 2) * perpendicular_y_head - offset * direction_y_head;

% 遍历之后的所有板凳，检查是否发生碰撞
for i = 3:n_sections
    % 当前节的后把手和前把手的位置
    x_back = positions_now(2*i-1); % 后把手
    y_back = positions_now(2*i); % 后把手
    x_front = positions_now(2*(i-1)-1); % 前把手
    y_front = positions_now(2*(i-1)); % 前把手

    % 计算板凳的方向向量（dx, dy）
    dx = x_back - x_front; % 计算前把手到后把手的向量
    dy = y_back - y_front;
    length_vector = sqrt(dx^2 + dy^2);

    % 归一化方向向量（使其长度为1）
    direction_x = dx / length_vector;
    direction_y = dy / length_vector;

    % 垂直方向向量（用于计算板凳的左右两侧）
    perpendicular_x = -direction_y;
    perpendicular_y = direction_x;

    % 计算板凳的四个角，考虑偏移量 offset 和宽度 width
    % 后右角
    x_back_left = x_back + (width / 2) * perpendicular_x + offset * direction_x;
    y_back_left = y_back + (width / 2) * perpendicular_y + offset * direction_y;

    % 后左角
    x_back_right = x_back - (width / 2) * perpendicular_x + offset * direction_x;
    y_back_right = y_back - (width / 2) * perpendicular_y + offset * direction_y;

    % 前右角
    x_front_left = x_front + (width / 2) * perpendicular_x - offset * direction_x;
    y_front_left = y_front + (width / 2) * perpendicular_y - offset * direction_y;

```

```
        % 前左角
        x_front_right = x_front - (width / 2) * perpendicular_x - offset * direction_x;
        y_front_right = y_front - (width / 2) * perpendicular_y - offset * direction_y;

        % 将这四个角点存储为矩形
        rect = [x_back_left, y_back_left;
                x_back_right, y_back_right;
                x_front_right, y_front_right;
                x_front_left, y_front_left];

        % 检查龙头前左、后左是否进入该矩形内
        if is_point_in_rectangle(x_head_front_left, y_head_front_left, rect) || ...
           is_point_in_rectangle(x_head_back_left, y_head_back_left, rect)
            collision_time = t;
            disp(['和第',num2str(i-2),'节龙身碰撞'])
            disp(['碰撞发生在 t = ', num2str(collision_time), ' 秒']);
            break;
        end
    end
    if collision_time > 0
        break;
    end
end

% 在碰撞时刻之前的时刻和下一个时刻计算速度
if collision_time > 0
    t_previous = collision_time - dt;

    % 计算碰撞前一时刻的位置
    positions_previous = calculate_positions_at_time(t_previous);

    % 计算下一小段的位置
    positions_next = calculate_positions_at_time(t_previous + dt_small);

    % 初始化存储位置和速度的矩阵（224行，3列）
    result_matrix = zeros(n_sections, 3);

    % 逐节计算每个把手的x, y和速度
    for i = 1:n_sections
        % 获取前一时刻的 x 和 y 坐标
        x_previous = positions_previous(2*i-1);
        y_previous = positions_previous(2*i);

        % 获取碰撞时刻的 x 和 y 坐标
        x_collision = positions_next(2*i-1);
        y_collision = positions_next(2*i);

        % 计算速度（基于前后两个时刻的位移）
        v_x = (x_collision - x_previous) / dt_small;
        v_y = (y_collision - y_previous) / dt_small;
        velocity_i = sqrt(v_x^2 + v_y^2); % 计算总速度

        % 将结果存储到矩阵中
        result_matrix(i, 1) = x_previous; % x 坐标
        result_matrix(i, 2) = y_previous; % y 坐标
        result_matrix(i, 3) = velocity_i; % 速度
```

```
    end

    %保留六位小数
    result_matrix = round(result_matrix, 6);
    % 构建行头
    headers_rows = {'', '龙头'};
    for i = 1:n_sections-3
        headers_rows = [headers_rows, {['第' num2str(i) '节龙身']}];
    end
    headers_rows = [headers_rows, {'龙尾'}];
    headers_rows = [headers_rows, {'龙尾（后）'}];

    % 构建列头，表示x,y和速度
    headers_columns = {'横坐标x(m)','纵坐标y(m)','速度（m/s）'};

    % 将 headers_rows 和 headers_columns 拼接到数据中
    final_data = [headers_rows', [headers_columns; num2cell(result_matrix)]];

    % 保存到Excel文件
    writecell(final_data, 'result2.xlsx');
end
```

Listing 3: 问题 2 中对板凳龙运动及碰撞进行模拟的代码

```
clc,clear;

% 自定义开始时间、结束时间和步长
start_time =0;   % 开始时间（秒）
end_time = 414;   % 结束时间（秒）
dt = 1;           % 时间步长（秒）

% 设定总时长和步长，生成时间数组
time_steps = start_time:dt:end_time;
n_sections = 224; % 总板凳数量
theta_initial = 2 * pi * 16; % 初始角度（第16圈），即 theta = 32*pi
pitch = 0.55; % 螺距

% 初始化存储位置的矩阵
positions = zeros(n_sections * 2, length(time_steps));

% 定义一个函数，输入时间t，返回t时刻的所有把手位置
function positions = calculate_positions_at_time(t)
    n_sections = 224; % 把手数量
    pitch = 0.55; % 螺距 55cm
    v = 1; %龙头的速度
    distance_1 = 2.86; % 龙头两个把手的距离
    distance_2 = 1.65; % 龙身两个把手的距离
    theta_initial = 2 * pi * 16; % 初始角度（第16圈）
    positions = zeros(n_sections * 2, 1); % 存储位置数据

    theta = sqrt(theta_initial^2 - 4 * pi * v / pitch * t); % 当前时刻的角度变化
    radius = pitch * theta / (2 * pi); % 螺旋半径

    % 计算龙头位置
    x_head = radius * cos(theta);
```

```
    y_head = radius * sin(theta);
    positions(1) = x_head;
    positions(2) = y_head;

    % 计算每节龙身的位置
    for i = 2:n_sections
        if i == 2
            distance = distance_1;
        else
            distance = distance_2;
        end
        delta_theta = distance / radius;
        theta = theta + delta_theta;
        radius = 0.55 * theta / (2 * pi);

        % 计算每节板凳的位置
        x_i = radius * cos(theta);
        y_i = radius * sin(theta);
        positions(2*i-1) = x_i;
        positions(2*i) = y_i;
    end
end

% 生成每秒的位置信息
for t_idx = 1:length(time_steps)
    t = time_steps(t_idx);
    positions(:, t_idx) = calculate_positions_at_time(t);
end

% 计算完整的螺旋轨迹：从 theta = 32*pi 到 theta = 0
theta_values = linspace(32*pi, 0, 1000); % 从 32*pi 到 0 的角度变化
spiral_x = zeros(size(theta_values));
spiral_y = zeros(size(theta_values));

for i = 1:length(theta_values)
    theta = theta_values(i);
    radius = pitch * theta / (2 * pi); % 螺旋半径
    spiral_x(i) = radius * cos(theta);
    spiral_y(i) = radius * sin(theta);
end

% 位置数据生成完成后，直接用于分析和绘图
% ---------------- 动态仿真 ------------------

figure_handle = figure; % 获取图像句柄
hold on;
grid on;
axis equal;
xlim([-12, 12]); % 调整显示区域
ylim([-12, 12]); % 调整显示区域
title('板凳龙位置仿真');
xlabel('X Position (m)');
ylabel('Y Position (m)');

% 在图上绘制螺旋线轨迹（虚线）
plot(spiral_x, spiral_y, '--k', 'LineWidth', 1); % 用虚线显示完整的螺旋轨迹
```

```
88
89  % 动态显示板凳龙的运动
90  for t_idx = 1:length(time_steps)
91      % 如果图像窗口已关闭，则中止动画
92      if ~isvalid(figure_handle)
93          disp('窗口已关闭，动画中止');
94          break;
95      end
96
97      % 清除之前的绘图（不清除螺旋轨迹）
98      cla;
99      plot(spiral_x, spiral_y, '--k', 'LineWidth', 1); % 重新绘制螺旋轨迹
100
101     % 获取当前时间步的所有板凳位置信息
102     x_positions = positions(1:2:end, t_idx); % x坐标
103     y_positions = positions(2:2:end, t_idx); % y坐标
104
105     % 绘制每节板凳的位置
106     for i = 2:n_sections
107         % 获取当前节的前后把手位置
108         x_back = x_positions(i);
109         y_back = y_positions(i);
110         x_front = x_positions(i - 1);
111         y_front = y_positions(i - 1);
112
113         % 计算方向向量
114         dx = x_front - x_back;
115         dy = y_front - y_back;
116         length_vector = sqrt(dx^2 + dy^2);
117         if length_vector == 0
118             continue; % 如果前后把手重合，则跳过当前节
119         end
120         direction_x = dx / length_vector;
121         direction_y = dy / length_vector;
122
123         % 垂直方向向量
124         perpendicular_x = -direction_y;
125         perpendicular_y = direction_x;
126
127         % 计算矩形四个角的位置
128         board_width = 0.3;
129         handle_offset = 0.275;
130
131         x_back_left = x_back + (board_width / 2) * perpendicular_x - handle_offset * direction_x;
132         y_back_left = y_back + (board_width / 2) * perpendicular_y - handle_offset * direction_y;
133         x_back_right = x_back - (board_width / 2) * perpendicular_x - handle_offset * direction_x;
134         y_back_right = y_back - (board_width / 2) * perpendicular_y - handle_offset * direction_y;
135         x_front_left = x_front + (board_width / 2) * perpendicular_x + handle_offset * direction_x;
136         y_front_left = y_front + (board_width / 2) * perpendicular_y + handle_offset * direction_y;
137         x_front_right = x_front - (board_width / 2) * perpendicular_x + handle_offset * direction_x;
138         y_front_right = y_front - (board_width / 2) * perpendicular_y + handle_offset * direction_y;
139
140         % 绘制矩形（表示板凳）
141         fill([x_back_left, x_back_right, x_front_right, x_front_left], ...
142             [y_back_left, y_back_right, y_front_right, y_front_left], 'r');
143     end
```

```
144
145     % 显示当前时间
146     current_time = time_steps(t_idx); % 获取当前时间
147     time_text = sprintf('Time: %.1f s', current_time); % 格式化时间文本
148     text(-11, 11, time_text, 'FontSize', 12, 'FontWeight', 'bold'); % 在图像左上角显示时间
149     % 暂停一段时间来创建动画效果
150     pause(0.1);
151 end
152
153 % Hold the final plot
154 hold off;
```

Listing 4: 求解问题 3 的代码，将求解过程和结果打印到控制台

```
1  clc; clear;
2  start_time = 0; %开始时间
3  dt = 0.01; % 默认时间步长
4  end_time = 440;%结束时间
5  n_sections = 224; % 把手数量
6  width = 0.3; % 板凳宽度30cm
7  offset = 0.275; % 把手到边界的距离为 0.275m
8  collision_time = -1; % 初始化碰撞时间
9  initial_pitch = 0.4; % 螺距初始值
10 pitch_step = 0.01; % 螺距增加的步长
11 radius_threshold = 4.5; % 半径阈值
12 max_pitch = 1.0; % 螺距最大值，防止无限循环
13
14 % 时间数组
15 time_steps = start_time:dt:end_time;
16
17 % 定义函数计算每时刻所有把手位置
18 function positions = calculate_positions_at_time(t, pitch)
19     n_sections = 224;
20     v = 1; %龙头的速度
21     distance_1 = 2.86; %龙头两个把手的距离
22     distance_2 = 1.65; %龙身两个把手的距离
23     theta_initial = 2 * pi * 16; % 初始角度
24     positions = zeros(n_sections * 2, 1);
25
26     theta = sqrt(theta_initial^2 - 4 * pi * v/ pitch * t); % 当前角度
27     radius = pitch * theta / (2 * pi); % 螺旋半径
28
29     x_head = radius * cos(theta);
30     y_head = radius * sin(theta);
31     positions(1) = x_head;
32     positions(2) = y_head;
33
34     for i = 2:n_sections
35         if (i == 2)
36             distance = distance_1;
37         else
38             distance = distance_2;
39         end
40         delta_theta = distance / radius;
41         theta = theta + delta_theta;
```

```
        radius = pitch * theta / (2 * pi);
        x_i = radius * cos(theta);
        y_i = radius * sin(theta);
        positions(2*i-1) = x_i;
        positions(2*i) = y_i;
    end
end

% 定义辅助函数，检查点是否在矩形内
function inside = is_point_in_rectangle(px, py, rect)
    x1 = rect(1,1); y1 = rect(1,2);
    x2 = rect(2,1); y2 = rect(2,2);
    x3 = rect(3,1); y3 = rect(3,2);
    x4 = rect(4,1); y4 = rect(4,2);

    % 使用向量叉积判断点是否在矩形内
    v1 = [x2 - x1, y2 - y1]; % 向量 v1
    v2 = [x3 - x2, y3 - y2]; % 向量 v2
    v3 = [x4 - x3, y4 - y3]; % 向量 v3
    v4 = [x1 - x4, y1 - y4]; % 向量 v4

    % 点相对于每条边的叉积
    cp1 = (px - x1)*(y2 - y1) - (py - y1)*(x2 - x1); % 点在边1左侧
    cp2 = (px - x2)*(y3 - y2) - (py - y2)*(x3 - x2); % 点在边2左侧
    cp3 = (px - x3)*(y4 - y3) - (py - y3)*(x4 - x3); % 点在边3左侧
    cp4 = (px - x4)*(y1 - y4) - (py - y4)*(x1 - x4); % 点在边4左侧

    % 如果所有叉积都为正或都为负，则点在矩形内
    inside = (cp1 >= 0 && cp2 >= 0 && cp3 >= 0 && cp4 >= 0) || ...
             (cp1 <= 0 && cp2 <= 0 && cp3 <= 0 && cp4 <= 0);
end

% 初始化最小螺距
found_pitch = -1;

% 开始调整 pitch 直到找到最小满足条件的 pitch
pitch = initial_pitch;
while pitch <= max_pitch
    collision_time = -1; % 重置碰撞时间

    % 主循环，检测碰撞
    for t_idx = 1:length(time_steps)
        t = time_steps(t_idx);
        positions_now = calculate_positions_at_time(t, pitch);
        % 检查螺旋中心，当 theta 接近 0 时，停止计算
        theta_2 = (2 * pi * 16)^2 - 4 * pi / 0.55 * t;
        if theta_2 < 0
            disp(['龙头在 t = ', num2str(t), ' 秒到达螺旋中心，无法继续旋入']);
            collision_time = t;
            break;
        end

        % 计算龙头的两个前角，基于前后把手的位置
        x_head_front = positions_now(1); % 龙头前把手的 x 坐标
        y_head_front = positions_now(2); % 龙头前把手的 y 坐标
        x_head_back = positions_now(3); % 龙头后把手的 x 坐标
```

```
    y_head_back = positions_now(4); % 龙头后把手的 y 坐标

    % 计算龙头的方向向量
    dx_head = x_head_front - x_head_back;
    dy_head = y_head_front - y_head_back;
    length_vector_head = sqrt(dx_head^2 + dy_head^2);

    % 归一化方向向量（使其长度为1）
    direction_x_head = dx_head / length_vector_head;
    direction_y_head = dy_head / length_vector_head;

    % 垂直方向向量（用于计算龙头的左右两侧）
    perpendicular_x_head = -direction_y_head;
    perpendicular_y_head = direction_x_head;

    % 计算龙头的前左和后左角坐标，考虑宽度和偏移量
    x_head_front_left = x_head_front + (width / 2) * perpendicular_x_head + offset * direction_x_head;
    y_head_front_left = y_head_front + (width / 2) * perpendicular_y_head + offset * direction_y_head;

    x_head_back_left = x_head_front + (width / 2) * perpendicular_x_head - offset * direction_x_head;
    y_head_back_left = y_head_front + (width / 2) * perpendicular_y_head - offset * direction_y_head;
    % 遍历之后的所有板凳，检查是否发生碰撞
    for i = 3:n_sections
        % 当前节的后把手和前把手的位置
        x_back = positions_now(2*i-1); % 后把手
        y_back = positions_now(2*i); % 后把手
        x_front = positions_now(2*(i-1)-1); % 前把手
        y_front = positions_now(2*(i-1)); % 前把手

        % 计算板凳的方向向量（dx, dy）
        dx = x_back - x_front; % 计算前把手到后把手的向量
        dy = y_back - y_front;
        length_vector = sqrt(dx^2 + dy^2);

        % 归一化方向向量（使其长度为1）
        direction_x = dx / length_vector;
        direction_y = dy / length_vector;

        % 垂直方向向量（用于计算板凳的左右两侧）
        perpendicular_x = -direction_y;
        perpendicular_y = direction_x;

        % 计算板凳的四个角，考虑偏移量 offset 和宽度 width
        % 后右角
        x_back_left = x_back + (width / 2) * perpendicular_x + offset * direction_x;
        y_back_left = y_back + (width / 2) * perpendicular_y + offset * direction_y;

        % 后左角
        x_back_right = x_back - (width / 2) * perpendicular_x + offset * direction_x;
        y_back_right = y_back - (width / 2) * perpendicular_y + offset * direction_y;

        % 前右角
        x_front_left = x_front + (width / 2) * perpendicular_x - offset * direction_x;
        y_front_left = y_front + (width / 2) * perpendicular_y - offset * direction_y;

        % 前左角
```

```
            x_front_right = x_front - (width / 2) * perpendicular_x - offset * direction_x;
            y_front_right = y_front - (width / 2) * perpendicular_y - offset * direction_y;

            % 将这四个角点存储为矩形
            rect = [x_back_left, y_back_left;
                    x_back_right, y_back_right;
                    x_front_right, y_front_right;
                    x_front_left, y_front_left];

            % 检查龙头前左、后左是否进入该矩形内
            if is_point_in_rectangle(x_head_front_left, y_head_front_left, rect) || ...
               is_point_in_rectangle(x_head_back_left, y_head_back_left, rect)
                collision_time = t;
                disp(['和第',num2str(i-2),'节龙身碰撞'])
                disp(['碰撞发生在 t = ', num2str(collision_time), ' 秒']);
                break;
            end
        end
        if collision_time > 0
            break;
        end
    end

    % 如果发生碰撞，检查龙头是否位于目标圆内
    if collision_time > 0
        x_head = positions_now(1); % 龙头前把手的 x 坐标
        y_head = positions_now(2); % 龙头前把手的 y 坐标

        if sqrt(x_head^2 + y_head^2) <= radius_threshold
            found_pitch = pitch; % 找到满足条件的最小螺距
            disp(['找到了满足条件的最小螺距: pitch = ', num2str(found_pitch)]);
            break;
        end
    end

    % 增加 pitch
    pitch = pitch + pitch_step;
end

% 如果没有找到满足条件的螺距，输出提示
if found_pitch == -1
    disp('未找到使得龙头位于圆内的最小螺距');
end
```

Listing 5: 求解问题 4 中最短调头曲线的代码

```
clc; clear;

% 螺旋线方程参数
a = 1.7; % 螺距常数
R_space = 4.5; % 调头空间半径

% 1. 盘入螺旋线交点 (极坐标 -> 笛卡尔坐标)
theta_in_intersection = (R_space * 2 * pi) / a; % 盘入交点角度
r_in_intersection = a * theta_in_intersection / (2 * pi); % 盘入交点极径
```

```
x_in_intersection = r_in_intersection * cos(theta_in_intersection); % 转换为 x 坐标
y_in_intersection = r_in_intersection * sin(theta_in_intersection); % 转换为 y 坐标

% 2. 盘出螺旋线交点（极坐标 -> 笛卡尔坐标）
x_out_intersection = -x_in_intersection;
y_out_intersection = -y_in_intersection;

% 3. 计算盘入螺旋线的切线向量并旋转 90 度（顺时针旋转）
dr_dtheta_in = a / (2 * pi);
dx_dtheta_in = -(dr_dtheta_in * cos(theta_in_intersection) - r_in_intersection * sin(theta_in_intersection));
dy_dtheta_in = -(dr_dtheta_in * sin(theta_in_intersection) + r_in_intersection * cos(theta_in_intersection));
tangent_in_unit_vector = [dx_dtheta_in, dy_dtheta_in] / norm([dx_dtheta_in, dy_dtheta_in]); % 单位向量

% 顺时针旋转90度
tangent_in_rotated = [tangent_in_unit_vector(2), -tangent_in_unit_vector(1)];

% 4. 盘出螺旋线的切线向量（与盘入螺旋线对称）
tangent_out_rotated = -tangent_in_rotated;

% 优化目标函数
objective = @(x) x(1) * x(3) + x(2) * x(4); % r1*theta1 + r2*theta2

% 约束函数定义
function [c, ceq] = constraints(x, x_in_intersection, y_in_intersection, x_out_intersection, y_out_intersection,
    ...
                          tangent_in_rotated, tangent_out_rotated)

   r1 = x(1); r2 = x(2); theta1 = x(3); theta2 = x(4);

   % 盘入螺旋线的圆心坐标
   x_in_circle_center = x_in_intersection + r1 * tangent_in_rotated(1);
   y_in_circle_center = y_in_intersection + r1 * tangent_in_rotated(2);

   % 盘出螺旋线的圆心坐标
   x_out_circle_center = x_out_intersection + r2 * tangent_out_rotated(1);
   y_out_circle_center = y_out_intersection + r2 * tangent_out_rotated(2);

   % 从圆心到交点的向量计算
   vector_in = [x_in_intersection - x_in_circle_center, y_in_intersection - y_in_circle_center];
   vector_out = [x_out_intersection - x_out_circle_center, y_out_intersection - y_out_circle_center];

   % 顺时针旋转后的盘入向量
   vector_in_rotated = [vector_in(1) * cos(theta1) + vector_in(2) * sin(theta1), ...
                       -vector_in(1) * sin(theta1) + vector_in(2) * cos(theta1)];

   % 顺时针旋转后的盘出向量
   vector_out_rotated = [vector_out(1) * cos(theta2) + vector_out(2) * sin(theta2), ...
                        -vector_out(1) * sin(theta2) + vector_out(2) * cos(theta2)];

   % 单位化两个旋转后的向量
   vector_in_rotated = vector_in_rotated / norm(vector_in_rotated);
   vector_out_rotated = vector_out_rotated / norm(vector_out_rotated);

   % 约束1：两个向量方向相反
   c = dot(vector_in_rotated, vector_out_rotated) + 1;

```

```
    % 约束2: 两个终点坐标相等
    x_in_rotated_end = x_in_circle_center + r1 * vector_in_rotated(1);
    y_in_rotated_end = y_in_circle_center + r1 * vector_in_rotated(2);

    x_out_rotated_end = x_out_circle_center + r2 * vector_out_rotated(1);
    y_out_rotated_end = y_out_circle_center + r2 * vector_out_rotated(2);

    ceq = [x_in_rotated_end - x_out_rotated_end, y_in_rotated_end - y_out_rotated_end];
end

% 初始猜测值 [r1, r2, theta1, theta2]
x0 = [1, 1, pi/4, pi/4];

% 优化求解选项
options = optimoptions('fmincon', 'Display', 'iter', 'Algorithm', 'sqp', 'MaxFunctionEvaluations', 1000);

% 定义边界，确保r1和r2为正数
lb = [0, 0, -Inf, -Inf]; % r1 和 r2 的下界
ub = []; % 不设置上界

% 定义优化问题的约束函数
constr_fun = @(x) constraints(x, x_in_intersection, y_in_intersection, x_out_intersection, y_out_intersection,
    ...
                        tangent_in_rotated, tangent_out_rotated);

% 使用fmincon进行优化求解
x_opt = fmincon(objective, x0, [], [], [], [], lb, ub, constr_fun, options);

% 输出结果
fprintf('优化后的 r1: %.4f, r2: %.4f, theta1: %.4f, theta2: %.4f\n', x_opt(1), x_opt(2), x_opt(3), x_opt(4));
```

Listing 6: 对问题 4 中求得的最短调头曲线进行验证的代码

```
clc; clear;

% 螺旋线方程参数
a = 1.7; % 螺距常数
R_space = 4.5; % 调头空间半径

% 1. 盘入螺旋线交点 (极坐标 -> 笛卡尔坐标)
theta_in_intersection = (R_space * 2 * pi) / a; % 盘入交点角度
r_in_intersection = a * theta_in_intersection / (2 * pi); % 盘入交点极径
x_in_intersection = r_in_intersection * cos(theta_in_intersection); % 转换为 x 坐标
y_in_intersection = r_in_intersection * sin(theta_in_intersection); % 转换为 y 坐标

% 2. 盘出螺旋线交点 (极坐标 -> 笛卡尔坐标)
x_out_intersection = -x_in_intersection;
y_out_intersection = -y_in_intersection;

% 3. 计算盘入螺旋线的切线向量并旋转 90 度 (顺时针旋转)
dr_dtheta_in = a / (2 * pi);
dx_dtheta_in = -(dr_dtheta_in * cos(theta_in_intersection) - r_in_intersection * sin(theta_in_intersection));
dy_dtheta_in = -(dr_dtheta_in * sin(theta_in_intersection) + r_in_intersection * cos(theta_in_intersection));
tangent_in_unit_vector = [dx_dtheta_in, dy_dtheta_in] / norm([dx_dtheta_in, dy_dtheta_in]); % 单位向量

```

```
% 顺时针旋转90度
tangent_in_rotated = [tangent_in_unit_vector(2), -tangent_in_unit_vector(1)];

% 4. 盘出螺旋线的切线向量（与盘入螺旋线对称）
tangent_out_rotated = -tangent_in_rotated;

% 优化后的 r1, r2, theta1, theta2（替换为优化结果）
r1 = 2.1118;
r2 = 2.3963;
theta1 = 3.0215;
theta2 = 3.0215;
s = theta1 * r1 + theta2 * r2;
fprintf("最短调头路径是为%0.6f \n",s);

% 5. 盘入螺旋线的圆心坐标
x_in_circle_center = x_in_intersection + r1 * tangent_in_rotated(1);
y_in_circle_center = y_in_intersection + r1 * tangent_in_rotated(2);

% 6. 盘出螺旋线的圆心坐标
x_out_circle_center = x_out_intersection + r2 * tangent_out_rotated(1);
y_out_circle_center = y_out_intersection + r2 * tangent_out_rotated(2);

% 7. 从圆心到交点的向量计算
vector_in = [x_in_intersection - x_in_circle_center, y_in_intersection - y_in_circle_center];
vector_out = [x_out_intersection - x_out_circle_center, y_out_intersection - y_out_circle_center];

% 8. 顺时针旋转后的盘入向量
vector_in_rotated = [vector_in(1) * cos(theta1) + vector_in(2) * sin(theta1), ...
                    -vector_in(1) * sin(theta1) + vector_in(2) * cos(theta1)];

% 9. 顺时针旋转后的盘出向量
vector_out_rotated = [vector_out(1) * cos(theta2) + vector_out(2) * sin(theta2), ...
                     -vector_out(1) * sin(theta2) + vector_out(2) * cos(theta2)];

% 10. 单位化两个旋转后的向量
vector_in_rotated = vector_in_rotated / norm(vector_in_rotated);
vector_out_rotated = vector_out_rotated / norm(vector_out_rotated);

% 11. 检查方向相反的约束
dot_product = dot(vector_in_rotated, vector_out_rotated);
fprintf('方向约束：内积 = %.6f（应接近 -1）\n', dot_product);

% 12. 检查终点坐标相等的约束
x_in_rotated_end = x_in_circle_center + r1 * vector_in_rotated(1);
y_in_rotated_end = y_in_circle_center + r1 * vector_in_rotated(2);

x_out_rotated_end = x_out_circle_center + r2 * vector_out_rotated(1);
y_out_rotated_end = y_out_circle_center + r2 * vector_out_rotated(2);

% 13. 终点差距
x_diff = abs(x_in_rotated_end - x_out_rotated_end);
y_diff = abs(y_in_rotated_end - y_out_rotated_end);
fprintf('终点约束：x 坐标差 = %.6f，y 坐标差 = %.6f（应接近 0）\n', x_diff, y_diff);
```

Listing 7: 画出求得的最短调头路径的代码

```
clc; clear;

% 螺旋线方程参数
a = 1.7; % 螺距常数
R_space = 4.5; % 调头空间半径

% 1. 盘入螺旋线的角度范围 (从 theta_in_start 到 theta_in_end)
theta_in_start = 16.6320;
theta_in_end = 32 * pi; % 盘入螺旋线的结束角度
theta_in = linspace(theta_in_start, theta_in_end, 1000); % 盘入螺旋线的角度范围
r_in = a * theta_in / (2 * pi); % 盘入螺旋线的半径

% 转换为笛卡尔坐标
x_in = r_in .* cos(theta_in);
y_in = r_in .* sin(theta_in);

% 2. 盘出螺旋线 (与盘入螺旋线中心对称)
x_out = -x_in;
y_out = -y_in;

% 3. 调头空间的圆
theta_circle = linspace(0, 2 * pi, 500); % 角度范围
x_circle = R_space * cos(theta_circle); % 圆的 x 坐标
y_circle = R_space * sin(theta_circle); % 圆的 y 坐标

% 优化后的 r1, r2, theta1, theta2 (替换为优化结果)
r1 = 2.1118;
r2 = 2.3963;
theta1 = 3.0215;
theta2 = 3.0215;

% 4. 计算圆心坐标
% 盘入螺旋线切线向量
dr_dtheta_in = a / (2 * pi);
theta_in_intersection = theta_in_start; % 交点发生在盘入螺旋线的起始角度
r_in_intersection = a * theta_in_intersection / (2 * pi);
x_in_intersection = r_in_intersection * cos(theta_in_intersection);
y_in_intersection = r_in_intersection * sin(theta_in_intersection);
dx_dtheta_in = -(dr_dtheta_in * cos(theta_in_intersection) - r_in_intersection * sin(theta_in_intersection));
dy_dtheta_in = -(dr_dtheta_in * sin(theta_in_intersection) + r_in_intersection * cos(theta_in_intersection));
tangent_in_unit_vector = [dx_dtheta_in, dy_dtheta_in] / norm([dx_dtheta_in, dy_dtheta_in]);

% 顺时针旋转90度的切线向量
tangent_in_rotated = [tangent_in_unit_vector(2), -tangent_in_unit_vector(1)];

% 盘入螺旋线的圆心坐标
x_in_circle_center = x_in_intersection + r1 * tangent_in_rotated(1)
y_in_circle_center = y_in_intersection + r1 * tangent_in_rotated(2)

% 盘出螺旋线的切线向量 (与盘入螺旋线对称)
tangent_out_rotated = -tangent_in_rotated;

% 盘出螺旋线的圆心坐标
x_out_intersection = -x_in_intersection;
```

```
y_out_intersection = -y_in_intersection;
x_out_circle_center = x_out_intersection + r2 * tangent_out_rotated(1)
y_out_circle_center = y_out_intersection + r2 * tangent_out_rotated(2)

% 5. 计算两段圆弧
% 第一段圆弧从盘入螺旋线的圆心出发，沿盘入方向顺时针旋转 theta1
% 第二段圆弧从盘出螺旋线的圆心出发，沿盘出方向顺时针旋转 theta2

% 调头弧线1: 从圆心到螺旋线交点的向量，顺时针旋转 theta1
theta_arc1 = linspace(0, -theta1, 100); % 顺时针旋转 theta1
arc1_x = x_in_circle_center + r1 * cos(theta_arc1 + atan2(y_in_intersection - y_in_circle_center,
    x_in_intersection - x_in_circle_center));
arc1_y = y_in_circle_center + r1 * sin(theta_arc1 + atan2(y_in_intersection - y_in_circle_center,
    x_in_intersection - x_in_circle_center));

% 调头弧线2: 从盘出螺旋线的圆心出发，顺时针旋转 theta2
theta_arc2 = linspace(0, -theta2, 100); % 顺时针旋转 theta2
arc2_x = x_out_circle_center + r2 * cos(theta_arc2 + atan2(y_out_intersection - y_out_circle_center,
    x_out_intersection - x_out_circle_center));
arc2_y = y_out_circle_center + r2 * sin(theta_arc2 + atan2(y_out_intersection - y_out_circle_center,
    x_out_intersection - x_out_circle_center));

% 6. 画图
figure;
hold on;
axis equal;
grid on;

% 盘入螺旋线
plot(x_in, y_in, 'b', 'LineWidth', 1.5);

% 盘出螺旋线
plot(x_out, y_out, 'r', 'LineWidth', 1.5);

% 调头空间的圆
plot(x_circle, y_circle, 'k--', 'LineWidth', 1.5);

% 圆心
plot(x_in_circle_center, y_in_circle_center, 'bo', 'MarkerFaceColor', 'b');
plot(x_out_circle_center, y_out_circle_center, 'ro', 'MarkerFaceColor', 'r');

% 调头弧线第一段
plot(arc1_x, arc1_y, 'g', 'LineWidth', 2);

% 调头弧线第二段
plot(arc2_x, arc2_y, 'g', 'LineWidth', 2);

title('螺旋线与调头空间的图示');
legend('盘入螺旋线', '盘出螺旋线', '调头空间', '盘入圆心', '盘出圆心', '调头弧线');
hold off;
```

Listing 8: 求解问题 4 中-100s 到 100s 位置和速度的代码，并将计算结果保存到 result4.xlsx 中

```
clc, clear;
```

```
start_time = -100; % 总时间 300秒
dt = 1; % 时间步长 1秒
end_time = 100;
dt_small = 1; % 用于计算速度的小时间步长
n_sections = 224; % 把手数量

% 时间数组
time_steps = start_time:dt:end_time;

% 初始化存储位置的矩阵
positions = zeros(n_sections * 2, length(time_steps));
velocity = zeros(n_sections, length(time_steps)); % 因为速度在相邻的整数秒之间计算


%定义一个函数，用于计算任意时刻所有把手的坐标
function positions = calculate_positions_at_time(t)
    % 参数设定
    v = 1; %龙头的速度
    pitch = 1.7; %螺距
    n_sections = 224;   % 总共224节
    distance_1 = 2.86;  % 龙头前两个把手的距离
    distance_2 = 1.65;  % 龙身和龙尾每两个把手的距离
    R_space = 4.5;      % 调头空间半径
    positions = zeros(n_sections * 2, 1); % 初始化存储位置的矩阵

    % 计算龙头前把手的位置
    [x_head, y_head] = calculate_head_position(t);

    % 从龙头开始，依次计算每个把手的位置
    current_x = x_head;
    current_y = y_head;
    previous_x = x_head;
    previous_y = y_head;

    for i = 1:n_sections
        % 根据当前是第一个把手还是后续把手确定距离
        if i == 1
            distance = distance_1; % 如果是龙头前把手，使用龙头前两个把手的距离
        else
            distance = distance_2; % 龙身和龙尾两个把手的距离
        end
        % 判断当前把手是否位于调头空间内
        if sqrt(current_x^2 + current_y^2) <= R_space
            % 如果当前把手位于调头空间内
            if i>=2 && sqrt(previous_x^2 + previous_y^2) >R_space
                % 如果是龙身把手且上一把手在调头空间外，代表当前位置不准确，需要更新当前位置
                if i==2
                    [current_x, current_y,~] = search_on_arc(previous_x, previous_y, distance_1);
                else
                    [current_x, current_y,~] = search_on_arc(previous_x, previous_y, distance_2);
                end
            end
            %如果是最后一个节点，直接存储
            if (i==n_sections)
                break;
            end
```

```
            %在圆弧内计算下一把手
            [next_x, next_y, found] = search_on_arc(current_x, current_y, distance);

            %如果在圆弧内没有找到
            if(~found)
                [next_x, next_y] = search_on_spiral(current_x, current_y, distance);
            end

        else
            % 如果当前把手不在调头空间内，按照近似方式计算下一把手
            current_r = sqrt(current_x^2 + current_y^2);
            current_theta = 2 * pi * current_r / 1.7;
            delta_theta = distance / current_r;

            %如果在盘入螺旋线
            if(cos(current_theta) * current_x >= 0 && sin(current_theta) * current_y >= 0)
                next_theta = current_theta + delta_theta;
                next_r = pitch * next_theta / (2 * pi);
                next_x = next_r * cos(next_theta);
                next_y = next_r * sin(next_theta);

            %如果在盘出螺旋线
            else
                next_theta = current_theta - delta_theta;
                next_r = pitch * next_theta / (2 * pi);
                next_x = -next_r * cos(next_theta);
                next_y = -next_r * sin(next_theta);
            end
        end

        %记录当前位置并更新坐标
        positions(2*i-1) = current_x;
        positions(2*i) = current_y;
        previous_x = current_x;
        previous_y = current_y;
        current_x = next_x;
        current_y = next_y;

    end
end

% 计算龙头位置的函数
function [x_head, y_head] = calculate_head_position(t)
    % 参数设定
    pitch = 1.7; % 螺距
    v = 1; %龙头的速度
    R_space = 4.5; % 调头空间半径
    r1 = 2.1118; % 调头弧线1的半径
    r2 = 2.3963; % 调头弧线2的半径
    theta1 = 3.0215; % 第一段圆弧旋转的角度
    theta2 = 3.0215; % 第二段圆弧旋转的角度
    T_turn = (r1 * theta1 + r2 * theta2) / 1; % 调头阶段总时间（速度1 m/s）

    % 1. 螺旋线阶段，适用于 -100s 到 0s
    if t <= 0
        theta_initial_in = 16.6320; % 螺旋线初始角度
```

```
        theta = sqrt(theta_initial_in^2 - 4 * pi * v / pitch * t); % 角度
        radius = pitch * theta / (2 * pi); % 半径
        x_head = radius * cos(theta); % x坐标
        y_head = radius * sin(theta); % y坐标

    % 2. 调头阶段，适用于 0s 到 T_turn
    elseif t > 0 && t <= T_turn
        % 第一段圆弧运动：0 到 r1 * theta1 时间内
        if t <= r1 * theta1
            % 圆心坐标
            x_in_circle_center = -1.3404;
            y_in_circle_center = -1.9852;

            % 圆弧和盘入螺旋线的交点
            x_in_intersection = -2.7119;
            y_in_intersection = -3.5911;

            % 当前时间对应的角度
            theta_arc1 = t / r1;
            % 从圆心到交点的向量方向
            theta_start_arc1 = atan2(y_in_intersection - y_in_circle_center, ...
                                x_in_intersection - x_in_circle_center);

            % 计算龙头在第一段圆弧上的位置
            x_head = x_in_circle_center + r1 * cos(theta_start_arc1 - theta_arc1); % 顺时针旋转
            y_head = y_in_circle_center + r1 * sin(theta_start_arc1 - theta_arc1); % 顺时针旋转

        % 第二段圆弧运动：从 r1 * theta1 到 T_turn
        else
            t_remaining = t - r1 * theta1; % 剩余时间
            x_out_circle_center = 1.1556;
            y_out_circle_center = 1.7986;

            % 两段圆弧交点坐标
            x_intersection = -0.1711;
            y_intersection = -0.2267;

            theta_arc2 = t_remaining / r2; % 当前时间对应的圆弧角度
            % 从交点到圆心的向量方向
            theta_start_arc2 = atan2(y_intersection - y_out_circle_center, ...
                                x_intersection - x_out_circle_center);

            % 计算龙头在第二段圆弧上的位置 (逆时针旋转 theta_arc2)
            x_head = x_out_circle_center + r2 * cos(theta_start_arc2 + theta_arc2);
            y_head = y_out_circle_center + r2 * sin(theta_start_arc2 + theta_arc2);
        end

    % 3. 盘出螺旋线阶段，适用于 T_turn 到 100s
    else
        t_out = t - T_turn; % 调头结束后的时间
        [x_head, y_head] = calculate_head_position(-t_out);
        x_head = -x_head;
        y_head = -y_head;

    end
end
```

```
170
171 %搜索在螺旋线上的下一个点
172 function [next_x, next_y] = search_on_spiral(x, y, distance)
173     % 初始theta值为16.6320
174     theta = 16.6320;
175
176     % 设定一个小的角度步长
177     delta_theta = 0.00001; % 可以根据需求调整步长的大小
178
179     while true
180         % 根据螺旋线方程 r = 1.7 * theta / (2 * pi) 计算当前的半径
181         current_r = 1.7 * theta / (2 * pi);
182         % 根据当前半径和theta计算螺旋线上对应的x, y坐标
183         current_x = current_r * cos(theta);
184         current_y = current_r * sin(theta);
185
186         % 计算螺旋线上的点与给定点(x, y)的距离
187         d = sqrt((current_x - x)^2 + (current_y - y)^2);
188
189         % 如果距离达到了目标距离，则返回当前螺旋线上的点
190         if abs(d - distance) < 0.5
191             next_x = current_x;
192             next_y = current_y;
193             break;
194         end
195
196         % 增加theta，继续沿着螺旋线移动
197         theta = theta + delta_theta;
198     end
199 end
200
201
202 %搜索在圆弧上的下一个把手
203 function [next_x, next_y, found] = search_on_arc(current_x, current_y, distance)
204     % 圆弧的参数设定
205     r1 = 2.1118; % 第一段圆弧的半径
206     r2 = 2.3963; % 第二段圆弧的半径
207     x_in_circle_center = -1.3404;
208     y_in_circle_center = -1.9852;
209     x_out_circle_center = 1.1556;
210     y_out_circle_center = 1.7986;
211
212     % 圆弧交点和圆弧与调头空间交点坐标
213     x_intersection = -0.1711;
214     y_intersection = -0.2267;
215     x_start_arc2 = 2.7119;
216     y_start_arc2 = 3.5911;
217     x_start_arc1 = -2.7119;
218     y_start_arc1 = -3.5911;
219
220     % 初始角度步长
221     angle_step = 0.00001; % 使用小步长搜索
222
223     % 1. 检查当前点是否位于第二段圆弧上
224     dist_to_out_center = sqrt((current_x - x_out_circle_center)^2 + (current_y - y_out_circle_center)^2);
225     if abs(dist_to_out_center - r2) < 0.5
```

```
        % 计算当前点的极角
        theta_current = atan2(current_y - y_out_circle_center, current_x - x_out_circle_center);
        % 计算第二段圆弧的角度范围
        theta_start_arc2 = atan2(y_start_arc2 - y_out_circle_center, x_start_arc2 - x_out_circle_center);
        theta_end_arc2 = atan2(y_intersection - y_out_circle_center, x_intersection - x_out_circle_center);

        % 判断当前点的角度是否在第二段圆弧的角度范围内
        if theta_current >= theta_end_arc2 && theta_current <= theta_start_arc2
            %先在第二段圆弧上搜索
            [next_x, next_y, found] = findNextArcPoint( ...
                                    current_x, current_y, ...          % 当前点的坐标
                                    x_out_circle_center, y_out_circle_center, ... % 外圆的圆心坐标
                                    r2, ...                            % 外圆的半径
                                    current_x, current_y, ...     % 第二段圆弧的起点坐标
                                    x_intersection, y_intersection, ... % 圆弧终点交点坐标
                                    distance, ...                      % 目标距离
                                    angle_step);                        % 角度步长
            if (found)
                return;
            end
            %再在第二段圆弧上搜索
            [next_x, next_y, found] = findNextArcPoint_r( ...
                                    current_x, current_y, ...          % 当前点的坐标
                                    x_in_circle_center, y_in_circle_center, ... % 内圆的圆心坐标
                                    r1, ...                            % 内圆的半径
                                    x_start_arc1, y_start_arc1, ...    % 第一段圆弧的起点坐标
                                    x_intersection, y_intersection, ... % 圆弧终点交点坐标
                                    distance, ...                      % 目标距离
                                    angle_step);
            return;
        end
    end

    % 2. 检查当前点是否位于第一段圆弧上
    dist_to_in_center = sqrt((current_x - x_in_circle_center)^2 + (current_y - y_in_circle_center)^2);
    if abs(dist_to_in_center - r1) < 0.5
        % 计算当前点的极角
        theta_current = atan2(current_y - y_in_circle_center, current_x - x_in_circle_center);
        % 计算第一段圆弧的角度范围
        theta_start_arc1 = atan2(y_start_arc1 - y_in_circle_center, x_start_arc1 - x_in_circle_center) + 2*pi;
        theta_end_arc1 = atan2(y_intersection - y_in_circle_center, x_intersection - x_in_circle_center);

        % 判断当前点的角度是否在第一段圆弧的角度范围内
        if theta_current >= theta_end_arc1 && theta_current <= theta_start_arc1
            %在第一段圆弧上搜索
            [next_x, next_y, found] = findNextArcPoint_r( ...
                                    current_x, current_y, ...          % 当前点的坐标
                                    x_in_circle_center, y_in_circle_center, ... % 内圆的圆心坐标
                                    r1, ...                            % 内圆的半径
                                    x_start_arc1, y_start_arc1, ...    % 第一段圆弧的起点坐标
                                    current_x, current_y, ... % 圆弧终点交点坐标
                                    distance, ...                      % 目标距离
                                    angle_step);                        % 角度步长
            if (found)
                return
            end
```

```
            found = false;
            next_x = NaN;
            next_y = NaN;
            return;
        end
    end
    [next_x, next_y, found] = findNextArcPoint( ...
        current_x, current_y, ...          % 当前点的坐标
        x_out_circle_center, y_out_circle_center, ... % 外圆的圆心坐标
        r2, ...                            % 外圆的半径
        x_start_arc2, y_start_arc2, ...    % 第二段圆弧的起点坐标
        x_intersection, y_intersection, ... % 圆弧终点交点坐标
        distance, ...                      % 目标距离
        angle_step);                        % 角度步长

end

%顺时针搜索圆弧上的下一个点
function [next_x, next_y, found] = findNextArcPoint( ...
    current_x, current_y, ...          % 当前点的坐标
    x_out_circle_center, y_out_circle_center, ... % 外圆的圆心坐标
    r2, ...                            % 外圆的半径
    x_start_arc2, y_start_arc2, ...    % 第二段圆弧的起点坐标
    x_intersection, y_intersection, ... % 圆弧终点交点坐标
    distance, ...                      % 目标距离
    angle_step)                        % 角度步长

    % 初始化搜索起始角度
    theta_current = atan2(y_start_arc2 - y_out_circle_center, x_start_arc2 - x_out_circle_center);
    % 初始化返回值
    found = false;
    next_x = NaN;
    next_y = NaN;

    % 搜索圆弧上的点
    while theta_current > atan2(y_intersection - y_out_circle_center, x_intersection - x_out_circle_center)
        % 计算当前圆弧点的位置
        next_x = x_out_circle_center + r2 * cos(theta_current);
        next_y = y_out_circle_center + r2 * sin(theta_current);

        % 计算该点与 current_x, current_y 的距离
        dist = sqrt((next_x - current_x)^2 + (next_y - current_y)^2);
        if abs(dist - distance) < 0.5
            % 找到符合条件的点，返回结果
            found = true;
            return;
        end

        % 没找到则继续沿着圆弧向内移动，减小角度
        theta_current = theta_current - angle_step;
    end
end


%逆时针搜索
function [next_x, next_y, found] = findNextArcPoint_r( ...
```

```
    current_x, current_y, ...          % 当前点的坐标
    x_in_circle_center, y_in_circle_center, ... % 内圆的圆心坐标
    r1, ...                            % 内圆的半径
    x_start_arc1, y_start_arc1, ...    % 第一段圆弧的起点坐标
    x_intersection, y_intersection, ... % 圆弧终点交点坐标
    distance, ...                      % 目标距离
    angle_step)                        % 角度步长

    % 初始化搜索起始角度
    theta_current = atan2(y_intersection - y_in_circle_center, x_intersection - x_in_circle_center);

    % 搜索圆弧上的点（逆时针）
    while theta_current < atan2(y_start_arc1 - y_in_circle_center, x_start_arc1 - x_in_circle_center) + 2*pi
        % 计算当前圆弧点的位置
        next_x = x_in_circle_center + r1 * cos(theta_current);
        next_y = y_in_circle_center + r1 * sin(theta_current);

        % 计算该点与 current_x, current_y 的距离
        dist = sqrt((next_x - current_x)^2 + (next_y - current_y)^2);
        if abs(dist - distance) < 0.5
            % 找到符合条件的点，返回结果
            found = true;
            return;
        end

        % 没找到则继续沿着圆弧向外移动，增加角度
        theta_current = theta_current + angle_step;
    end

    % 如果没有找到符合条件的点，返回 false
    found = false;
    next_x = NaN;
    next_y = NaN;
end


% 计算每秒钟的位置信息
for t_idx = 1:length(time_steps)
    t = time_steps(t_idx);
    positions(:, t_idx) = calculate_positions_at_time(t);
end

% 计算整数秒之间的速度，使用 t 和 t+0.0001 之间的位移来计算
for t_idx = 1:length(time_steps)
    % 当前时刻 t_now 和 t+0.001 秒时刻
    t_now = time_steps(t_idx);
    delta_t = dt_small;
    if t_now == 0
        delta_t = 1;
    end
    t_small = t_now + delta_t;

    % 获取所有把手在 t_now 时刻的位置
    pos_now = calculate_positions_at_time(t_now); % 返回一个 2*n_sections 大小的向量

    % 获取所有把手在 t+0.001s 时刻的位置
```

```
    pos_next = calculate_positions_at_time(t_small); % 返回一个 2*n_sections 大小的向量

    % 将 pos_now 和 pos_next 重新分为 x, y 坐标
    x_now = pos_now(1:2:end); % 当前时刻所有把手的 x 坐标
    y_now = pos_now(2:2:end); % 当前时刻所有把手的 y 坐标
    x_next = pos_next(1:2:end); % t+0.001s 时刻所有把手的 x 坐标
    y_next = pos_next(2:2:end); % t+0.001s 时刻所有把手的 y 坐标

    % 计算速度 (vx, vy)，然后计算总速度
    v_x = (x_next - x_now) / delta_t;
    v_y = (y_next - y_now) / delta_t;

    % 计算每个把手的速度并存储
    velocity(:, t_idx) = sqrt(v_x.^2 + v_y.^2);
end


%将结果保留六位小数
positions = round(positions, 6);
velocity = round(velocity, 6);

% 构建第一页行头，表示每节板凳的 x(m) 和 y(m) 坐标
headers_rows_1 = {'', '龙头x(m)', '龙头y(m)'};
for i = 1:n_sections-3
    headers_rows_1 = [headers_rows_1, {['第' num2str(i) '节龙身x(m)'], ['第' num2str(i) '节龙身y(m)']}];
end
headers_rows_1 = [headers_rows_1, {'龙尾x(m)', '龙尾y(m)'}];
headers_rows_1 = [headers_rows_1, {'龙尾（后）x(m)', '龙尾（后）y(m)'}];

% 构建第一页列头，表示每个时间点 (0s, 1s, 2s, ...)
headers_columns_1 = arrayfun(@(x) [num2str(x) 's'], time_steps, 'UniformOutput', false);

% 将 headers_rows_1 和 headers_columns_1 拼接到数据中
final_data_1 = [headers_rows_1', [headers_columns_1; num2cell(positions)]];

% 构建第二页行头
headers_rows_2 = {'', '龙头（m/s）'};
for i = 1:n_sections-3
    headers_rows_2 = [headers_rows_2, {['第' num2str(i) '节龙身（m/s）']}];
end
headers_rows_2 = [headers_rows_2, {'龙尾（m/s）'}];
headers_rows_2 = [headers_rows_2, {'龙尾（后）（m/s）'}];

% 构建第二页列头，表示每个时间点 (1s, 2s, 3s, ...)
headers_columns_2 = arrayfun(@(x) [num2str(x) 's'], time_steps(1:end), 'UniformOutput', false);

% 将 headers_rows_2 和 headers_columns_2 拼接到数据中
final_data_2 = [headers_rows_2', [headers_columns_2; num2cell(velocity)]];

% 保存到Excel文件，位置数据到第一页，速度数据到第二页
writecell(final_data_1, 'result4.xlsx', 'Sheet', '位置');
writecell(final_data_2, 'result4.xlsx', 'Sheet', '速度');
```

Listing 9: 对问题 4 整个调头过程进行仿真模拟的代码

```
clc; clear;

% 第一段代码中的螺旋线方程参数
a = 1.7; % 螺距常数
R_space = 4.5; % 调头空间半径

% 1. 盘入螺旋线的角度范围 (从 theta_in_start 到 theta_in_end)
theta_in_start = 16.6320;
theta_in_end = 32 * pi; % 盘入螺旋线的结束角度
theta_in = linspace(theta_in_start, theta_in_end, 1000); % 盘入螺旋线的角度范围
r_in = a * theta_in / (2 * pi); % 盘入螺旋线的半径

% 转换为笛卡尔坐标
x_in = r_in .* cos(theta_in);
y_in = r_in .* sin(theta_in);

% 2. 盘出螺旋线 (与盘入螺旋线中心对称)
x_out = -x_in;
y_out = -y_in;

% 3. 调头空间的圆
theta_circle = linspace(0, 2 * pi, 500); % 角度范围
x_circle = R_space * cos(theta_circle); % 圆的 x 坐标
y_circle = R_space * sin(theta_circle); % 圆的 y 坐标

% 优化后的 r1, r2, theta1, theta2 (假设的数值)
r1 = 2.1118;
r2 = 2.3963;
theta1 = 3.0215;
theta2 = 3.0215;

% 计算圆心坐标
% 盘入螺旋线的切线向量
theta_in_intersection = theta_in_start; % 交点发生在盘入螺旋线的起始角度
r_in_intersection = a * theta_in_intersection / (2 * pi);
x_in_intersection = r_in_intersection * cos(theta_in_intersection);
y_in_intersection = r_in_intersection * sin(theta_in_intersection);
dr_dtheta_in = a / (2 * pi);
dx_dtheta_in = -(dr_dtheta_in * cos(theta_in_intersection) - r_in_intersection * sin(theta_in_intersection));
dy_dtheta_in = -(dr_dtheta_in * sin(theta_in_intersection) + r_in_intersection * cos(theta_in_intersection));
tangent_in_unit_vector = [dx_dtheta_in, dy_dtheta_in] / norm([dx_dtheta_in, dy_dtheta_in]);

% 顺时针旋转90度的切线向量
tangent_in_rotated = [tangent_in_unit_vector(2), -tangent_in_unit_vector(1)];

% 盘入螺旋线的圆心坐标
x_in_circle_center = x_in_intersection + r1 * tangent_in_rotated(1);
y_in_circle_center = y_in_intersection + r1 * tangent_in_rotated(2);

% 盘出螺旋线的切线向量 (与盘入螺旋线对称)
tangent_out_rotated = -tangent_in_rotated;

% 盘出螺旋线的圆心坐标
x_out_intersection = -x_in_intersection;
y_out_intersection = -y_in_intersection;
x_out_circle_center = x_out_intersection + r2 * tangent_out_rotated(1);
```

```
y_out_circle_center = y_out_intersection + r2 * tangent_out_rotated(2);

% 计算调头弧线
theta_arc1 = linspace(0, -theta1, 100); % 第一段圆弧的角度范围
arc1_x = x_in_circle_center + r1 * cos(theta_arc1 + atan2(y_in_intersection - y_in_circle_center,
    x_in_intersection - x_in_circle_center));
arc1_y = y_in_circle_center + r1 * sin(theta_arc1 + atan2(y_in_intersection - y_in_circle_center,
    x_in_intersection - x_in_circle_center));

theta_arc2 = linspace(0, -theta2, 100); % 第二段圆弧的角度范围
arc2_x = x_out_circle_center + r2 * cos(theta_arc2 + atan2(y_out_intersection - y_out_circle_center,
    x_out_intersection - x_out_circle_center));
arc2_y = y_out_circle_center + r2 * sin(theta_arc2 + atan2(y_out_intersection - y_out_circle_center,
    x_out_intersection - x_out_circle_center));

% ---------------- 动态仿真设置 ------------------

% 读取 Excel 文件中的位置数据
data = readmatrix('result4.xlsx', 'Sheet', '位置');

% 数据的第一行是列头，第一列是行头，所以需要去掉
positions_data = data(1:end, 2:end);

% 将数据转换为矩阵
positions = reshape(positions_data', [], size(positions_data, 1))';

% 设置时间段参数
start_time = -100; % 开始时间（秒）
end_time = 100;  % 结束时间（秒）
time_steps = start_time:1:end_time; % 时间数组，对应201个时刻
n_sections = 224; % 总板凳数量

% ---------------- 创建图像和动画 ------------------

figure_handle = figure; % 获取图像句柄
hold on;
grid on;
axis equal;
xlim([-12, 12]); % 调整显示区域
ylim([-12, 12]); % 调整显示区域
title('板凳龙位置仿真与螺旋线图示');
xlabel('X Position (m)');
ylabel('Y Position (m)');

% 绘制静态的螺旋线轨迹和调头空间
plot(x_in, y_in, 'b', 'LineWidth', 1.5); % 盘入螺旋线
plot(x_out, y_out, 'r', 'LineWidth', 1.5); % 盘出螺旋线
plot(x_circle, y_circle, 'k--', 'LineWidth', 1.5); % 调头空间的圆

% 绘制调头弧线
plot(arc1_x, arc1_y, 'g', 'LineWidth', 2); % 第一段调头弧线
plot(arc2_x, arc2_y, 'g', 'LineWidth', 2); % 第二段调头弧线

% 动态绘制板凳龙的位置
for t_idx = 100:201 % 时间范围对应的位置索引，从第101个时间步到第201个
    % 如果图像窗口已关闭，则中止动画
```

```
    if ~isvalid(figure_handle)
        disp('窗口已关闭，动画中止');
        break;
    end

    % 清除之前的板凳绘图（不清除螺旋线、调头空间和调头弧线）
    cla;
    plot(x_in, y_in, 'b', 'LineWidth', 1.5); % 重新绘制螺旋轨迹
    plot(x_out, y_out, 'r', 'LineWidth', 1.5); % 重新绘制盘出螺旋线
    plot(x_circle, y_circle, 'k--', 'LineWidth', 1.5); % 重新绘制调头圆
    plot(arc1_x, arc1_y, 'g', 'LineWidth', 2); % 重新绘制第一段调头弧线
    plot(arc2_x, arc2_y, 'g', 'LineWidth', 2); % 重新绘制第二段调头弧线

    % 获取当前时间步的所有板凳位置信息
    x_positions = positions(1:2:end, t_idx); % x坐标
    y_positions = positions(2:2:end, t_idx); % y坐标

    % 绘制每节板凳的位置
    for i = 2:n_sections
        % 获取当前节的前后把手位置
        x_back = x_positions(i);
        y_back = y_positions(i);
        x_front = x_positions(i - 1);
        y_front = y_positions(i - 1);

        % 计算方向向量
        dx = x_front - x_back;
        dy = y_front - y_back;
        length_vector = sqrt(dx^2 + dy^2);
        if length_vector == 0
            continue; % 如果前后把手重合，则跳过当前节
        end
        direction_x = dx / length_vector;
        direction_y = dy / length_vector;

        % 垂直方向向量
        perpendicular_x = -direction_y;
        perpendicular_y = direction_x;

        % 计算矩形四个角的位置
        board_width = 0.3;
        handle_offset = 0.275;

        x_back_left = x_back + (board_width / 2) * perpendicular_x - handle_offset * direction_x;
        y_back_left = y_back + (board_width / 2) * perpendicular_y - handle_offset * direction_y;
        x_back_right = x_back - (board_width / 2) * perpendicular_x - handle_offset * direction_x;
        y_back_right = y_back - (board_width / 2) * perpendicular_y - handle_offset * direction_y;
        x_front_left = x_front + (board_width / 2) * perpendicular_x + handle_offset * direction_x;
        y_front_left = y_front + (board_width / 2) * perpendicular_y + handle_offset * direction_y;
        x_front_right = x_front - (board_width / 2) * perpendicular_x + handle_offset * direction_x;
        y_front_right = y_front - (board_width / 2) * perpendicular_y + handle_offset * direction_y;

        % 绘制矩形（表示板凳）
        fill([x_back_left, x_back_right, x_front_right, x_front_left], ...
             [y_back_left, y_back_right, y_front_right, y_front_left], 'r');
    end
```

```
165
166     % 显示当前时间
167     current_time = time_steps(t_idx); % 获取当前时间
168     time_text = sprintf('Time: %.1f s', current_time); % 格式化时间文本
169     text(-11, 11, time_text, 'FontSize', 12, 'FontWeight', 'bold'); % 在图像左上角显示时间
170
171     % 暂停一段时间来创建动画效果
172     pause(0.1); % 调整 pause 值可以改变动画速度
173 end
174
175 % Hold the final plot
176 hold off;
```

Listing 10: 求解问题 5 的代码，将过程和结果打印到控制台上

```
1  clc, clear;
2  start_time = 0; % 从进入调头空间开始计算
3  dt = 1; % 时间步长 1秒
4  end_time = 100;%结束时间足够大，确保不会遗漏
5  dt_small = 1; % 用于计算速度的小时间步长
6  n_sections = 224; % 把手数量
7
8  % 时间数组
9  time_steps = start_time:dt:end_time;
10
11 %定义一个函数，用于计算任意时刻所有把手的坐标
12 function positions = calculate_positions_at_time(t,v)
13     % 参数设定
14     pitch = 1.7; %螺距
15     n_sections = 224;   % 总共224节
16     distance_1 = 2.86;  % 龙头前两个把手的距离
17     distance_2 = 1.65;  % 龙身和龙尾每两个把手的距离
18     R_space = 4.5;      % 调头空间半径
19     positions = zeros(n_sections * 2, 1); % 初始化存储位置的矩阵
20
21     % 计算龙头前把手的位置
22     [x_head, y_head] = calculate_head_position(t,v);
23
24     % 从龙头开始，依次计算每个把手的位置
25     current_x = x_head;
26     current_y = y_head;
27     previous_x = x_head;
28     previous_y = y_head;
29
30     for i = 1:n_sections
31         % 根据当前是第一个把手还是后续把手确定距离
32         if i == 1
33             distance = distance_1; % 如果是龙头前把手，使用龙头前两个把手的距离
34         else
35             distance = distance_2; % 龙身和龙尾两个把手的距离
36         end
37         % 判断当前把手是否位于调头空间内
38         if sqrt(current_x^2 + current_y^2) <= R_space
39             % 如果当前把手位于调头空间内
40             if i>=2 && sqrt(previous_x^2 + previous_y^2) >R_space
```

```
            % 如果是龙身把手且上一把手在调头空间外，代表当前位置不准确，需要更新当前位置
            if i==2
                [current_x, current_y,~] = search_on_arc(previous_x, previous_y, distance_1);
            else
                [current_x, current_y,~] = search_on_arc(previous_x, previous_y, distance_2);
            end
        end
        %如果是最后一个节点，直接存储
        if (i==n_sections)
            break;
        end
        %在圆弧内计算下一把手
        [next_x, next_y, found] = search_on_arc(current_x, current_y, distance);

        %如果在圆弧内没有找到
        if(~found)
            [next_x, next_y] = search_on_spiral(current_x, current_y, distance);
        end

    else
        % 如果当前把手不在调头空间内，按照近似方式计算下一把手
        current_r = sqrt(current_x^2 + current_y^2);
        current_theta = 2 * pi * current_r / pitch;
        delta_theta = distance / current_r;

        %如果在盘入螺旋线
        if(cos(current_theta) * current_x >= 0 && sin(current_theta) * current_y >= 0)
            next_theta = current_theta + delta_theta;
            next_r = pitch * next_theta / (2 * pi);
            next_x = next_r * cos(next_theta);
            next_y = next_r * sin(next_theta);

        %如果在盘出螺旋线
        else
            next_theta = current_theta - delta_theta;
            next_r = pitch * next_theta / (2 * pi);
            next_x = -next_r * cos(next_theta);
            next_y = -next_r * sin(next_theta);
        end
    end

    %记录当前位置并更新坐标
    positions(2*i-1) = current_x;
    positions(2*i) = current_y;
    previous_x = current_x;
    previous_y = current_y;
    current_x = next_x;
    current_y = next_y;

  end
end

% 计算龙头位置的函数
function [x_head, y_head] = calculate_head_position(t,v)
    % 参数设定
    pitch = 1.7; % 螺距
```

```
r1 = 2.1118; % 调头弧线1的半径
r2 = 2.3963; % 调头弧线2的半径
theta1 = 3.0215; % 第一段圆弧旋转的角度
theta2 = 3.0215; % 第二段圆弧旋转的角度
T_turn = (r1 * theta1 + r2 * theta2) / v; % 调头阶段总时间

% 1. 螺旋线阶段，适用于 -100s 到 0s
if t <= 0
    theta_initial_in = 16.6320; % 螺旋线初始角度
    theta = sqrt(theta_initial_in^2 - 4 * pi * v / pitch * t); % 角度
    radius = pitch * theta / (2 * pi); % 半径
    x_head = radius * cos(theta); % x坐标
    y_head = radius * sin(theta); % y坐标

% 2. 调头阶段，适用于 0s 到 T_turn
elseif t > 0 && t <= T_turn
    % 第一段圆弧运动：0 到 r1 * theta1 时间内
    if t <= (r1 * theta1)/v
        % 圆心坐标
        x_in_circle_center = -1.3404;
        y_in_circle_center = -1.9852;

        % 圆弧和盘入螺旋线的交点
        x_in_intersection = -2.7119;
        y_in_intersection = -3.5911;

        % 当前时间对应的角度
        theta_arc1 = t / r1;
        % 从圆心到交点的向量方向
        theta_start_arc1 = atan2(y_in_intersection - y_in_circle_center, ...
                                 x_in_intersection - x_in_circle_center);

        % 计算龙头在第一段圆弧上的位置
        x_head = x_in_circle_center + r1 * cos(theta_start_arc1 - theta_arc1); % 顺时针旋转
        y_head = y_in_circle_center + r1 * sin(theta_start_arc1 - theta_arc1); % 顺时针旋转

    % 第二段圆弧运动：从 r1 * theta1 到 T_turn
    else
        t_remaining = t - r1 * theta1 / v; % 剩余时间
        x_out_circle_center = 1.1556;
        y_out_circle_center = 1.7986;

        % 两段圆弧交点坐标
        x_intersection = -0.1711;
        y_intersection = -0.2267;

        theta_arc2 = t_remaining * v / r2; % 当前时间对应的圆弧角度
        % 从交点到圆心的向量方向
        theta_start_arc2 = atan2(y_intersection - y_out_circle_center, ...
                                 x_intersection - x_out_circle_center);

        % 计算龙头在第二段圆弧上的位置 (逆时针旋转 theta_arc2)
        x_head = x_out_circle_center + r2 * cos(theta_start_arc2 + theta_arc2);
        y_head = y_out_circle_center + r2 * sin(theta_start_arc2 + theta_arc2);
    end

```

```
    % 3. 盘出螺旋线阶段，适用于 T_turn 到 100s
    else
        t_out = t - T_turn; % 调头结束后的时间
        [x_head, y_head] = calculate_head_position(-t_out, v);
        x_head = -x_head;
        y_head = -y_head;

    end
end

%搜索在螺旋线上的下一个点
function [next_x, next_y] = search_on_spiral(x, y, distance)
    % 初始theta值为16.6320
    theta = 16.6320;

    % 设定一个小的角度步长
    delta_theta = 0.00001; % 可以根据需求调整步长的大小
    while true
        % 根据螺旋线方程 r = 1.7 * theta / (2 * pi) 计算当前的半径
        current_r = 1.7 * theta / (2 * pi);
        % 根据当前半径和theta计算螺旋线上对应的x, y坐标
        current_x = current_r * cos(theta);
        current_y = current_r * sin(theta);

        % 计算螺旋线上的点与给定点(x, y)的距离
        d = sqrt((current_x - x)^2 + (current_y - y)^2);

        % 如果距离达到了目标距离，则返回当前螺旋线上的点
        if abs(d - distance) < 0.5
            next_x = current_x;
            next_y = current_y;
            break;
        end

        % 增加theta，继续沿着螺旋线移动
        theta = theta + delta_theta;

    end
end


%搜索在圆弧上的下一个把手
function [next_x, next_y, found] = search_on_arc(current_x, current_y, distance)
    % 圆弧的参数设定
    r1 = 2.1118; % 第一段圆弧的半径
    r2 = 2.3963; % 第二段圆弧的半径
    x_in_circle_center = -1.3404;
    y_in_circle_center = -1.9852;
    x_out_circle_center = 1.1556;
    y_out_circle_center = 1.7986;

    % 圆弧交点和圆弧与调头空间交点坐标
    x_intersection = -0.1711;
    y_intersection = -0.2267;
    x_start_arc2 = 2.7119;
    y_start_arc2 = 3.5911;
```

```
    x_start_arc1 = -2.7119;
    y_start_arc1 = -3.5911;

    % 初始角度步长
    angle_step = 0.00001; % 使用小步长搜索

    % 1. 检查当前点是否位于第二段圆弧上
    dist_to_out_center = sqrt((current_x - x_out_circle_center)^2 + (current_y - y_out_circle_center)^2);
    if abs(dist_to_out_center - r2) < 0.5
        % 计算当前点的极角
        theta_current = atan2(current_y - y_out_circle_center, current_x - x_out_circle_center);
        % 计算第二段圆弧的角度范围
        theta_start_arc2 = atan2(y_start_arc2 - y_out_circle_center, x_start_arc2 - x_out_circle_center);
        theta_end_arc2 = atan2(y_intersection - y_out_circle_center, x_intersection - x_out_circle_center);

        % 判断当前点的角度是否在第二段圆弧的角度范围内
        if theta_current >= theta_end_arc2 && theta_current <= theta_start_arc2
            %先在第二段圆弧上搜索
            [next_x, next_y, found] = findNextArcPoint( ...
                                    current_x, current_y, ...          % 当前点的坐标
                                    x_out_circle_center, y_out_circle_center, ... % 外圆的圆心坐标
                                    r2, ...                            % 外圆的半径
                                    current_x, current_y, ...     % 第二段圆弧的起点坐标
                                    x_intersection, y_intersection, ... % 圆弧终点交点坐标
                                    distance, ...                      % 目标距离
                                    angle_step);                        % 角度步长
            if (found)
                return;
            end
            %再在第二段圆弧上搜索
            [next_x, next_y, found] = findNextArcPoint_r( ...
                                    current_x, current_y, ...          % 当前点的坐标
                                    x_in_circle_center, y_in_circle_center, ... % 内圆的圆心坐标
                                    r1, ...                            % 内圆的半径
                                    x_start_arc1, y_start_arc1, ...    % 第一段圆弧的起点坐标
                                    x_intersection, y_intersection, ... % 圆弧终点交点坐标
                                    distance, ...                      % 目标距离
                                    angle_step);
            return;
        end
    end

    % 2. 检查当前点是否位于第一段圆弧上
    dist_to_in_center = sqrt((current_x - x_in_circle_center)^2 + (current_y - y_in_circle_center)^2);
    if abs(dist_to_in_center - r1) < 0.5
        % 计算当前点的极角
        theta_current = atan2(current_y - y_in_circle_center, current_x - x_in_circle_center);
        % 计算第一段圆弧的角度范围
        theta_start_arc1 = atan2(y_start_arc1 - y_in_circle_center, x_start_arc1 - x_in_circle_center) + 2*pi;
        theta_end_arc1 = atan2(y_intersection - y_in_circle_center, x_intersection - x_in_circle_center);

        % 判断当前点的角度是否在第一段圆弧的角度范围内
        if theta_current >= theta_end_arc1 && theta_current <= theta_start_arc1
            %在第一段圆弧上搜索
            [next_x, next_y, found] = findNextArcPoint_r( ...
                                    current_x, current_y, ...          % 当前点的坐标
```

```
                                x_in_circle_center, y_in_circle_center, ... % 内圆的圆心坐标
                                r1, ...                                % 内圆的半径
                                x_start_arc1, y_start_arc1, ...    % 第一段圆弧的起点坐标
                                current_x, current_y, ... % 圆弧终点交点坐标
                                distance, ...                          % 目标距离
                                angle_step);                           % 角度步长
            if (found)
                return
            end
            found = false;
            next_x = NaN;
            next_y = NaN;
            return;
        end
    end
    [next_x, next_y, found] = findNextArcPoint( ...
        current_x, current_y, ...          % 当前点的坐标
        x_out_circle_center, y_out_circle_center, ... % 外圆的圆心坐标
        r2, ...                            % 外圆的半径
        x_start_arc2, y_start_arc2, ...    % 第二段圆弧的起点坐标
        x_intersection, y_intersection, ... % 圆弧终点交点坐标
        distance, ...                      % 目标距离
        angle_step);                       % 角度步长

end

%顺时针搜索圆弧上的下一个点
function [next_x, next_y, found] = findNextArcPoint( ...
    current_x, current_y, ...          % 当前点的坐标
    x_out_circle_center, y_out_circle_center, ... % 外圆的圆心坐标
    r2, ...                            % 外圆的半径
    x_start_arc2, y_start_arc2, ...    % 第二段圆弧的起点坐标
    x_intersection, y_intersection, ... % 圆弧终点交点坐标
    distance, ...                      % 目标距离
    angle_step)                        % 角度步长

    % 初始化搜索起始角度
    theta_current = atan2(y_start_arc2 - y_out_circle_center, x_start_arc2 - x_out_circle_center);
    % 初始化返回值
    found = false;
    next_x = NaN;
    next_y = NaN;

    % 搜索圆弧上的点
    while theta_current > atan2(y_intersection - y_out_circle_center, x_intersection - x_out_circle_center)
        % 计算当前圆弧点的位置
        next_x = x_out_circle_center + r2 * cos(theta_current);
        next_y = y_out_circle_center + r2 * sin(theta_current);

        % 计算该点与 current_x, current_y 的距离
        dist = sqrt((next_x - current_x)^2 + (next_y - current_y)^2);
        if abs(dist - distance) < 0.5
            % 找到符合条件的点，返回结果
            found = true;
            return;
        end
```

```
321
322         % 没找到则继续沿着圆弧向内移动，减小角度
323         theta_current = theta_current - angle_step;
324     end
325 end
326
327
328 %逆时针搜索
329 function [next_x, next_y, found] = findNextArcPoint_r( ...
330     current_x, current_y, ...          % 当前点的坐标
331     x_in_circle_center, y_in_circle_center, ... % 内圆的圆心坐标
332     r1, ...                            % 内圆的半径
333     x_start_arc1, y_start_arc1, ...    % 第一段圆弧的起点坐标
334     x_intersection, y_intersection, ... % 圆弧终点交点坐标
335     distance, ...                      % 目标距离
336     angle_step)                        % 角度步长
337
338     % 初始化搜索起始角度
339     theta_current = atan2(y_intersection - y_in_circle_center, x_intersection - x_in_circle_center);
340
341     % 搜索圆弧上的点（逆时针）
342     while theta_current < atan2(y_start_arc1 - y_in_circle_center, x_start_arc1 - x_in_circle_center) + 2*pi
343         % 计算当前圆弧点的位置
344         next_x = x_in_circle_center + r1 * cos(theta_current);
345         next_y = y_in_circle_center + r1 * sin(theta_current);
346
347         % 计算该点与 current_x, current_y 的距离
348         dist = sqrt((next_x - current_x)^2 + (next_y - current_y)^2);
349         if abs(dist - distance) < 0.5
350             % 找到符合条件的点，返回结果
351             found = true;
352             return;
353         end
354
355         % 没找到则继续沿着圆弧向外移动，增加角度
356         theta_current = theta_current + angle_step;
357     end
358
359     % 如果没有找到符合条件的点，返回 false
360     found = false;
361     next_x = NaN;
362     next_y = NaN;
363 end
364
365
366 % 定义初始龙头速度和步长
367 v_head = 1.00; % 龙头速度初始为1m/s
368 v_step = 0.01; % 每次增加的速度步长
369 max_v = 2; % 把手速度的最大值
370 R_space = 4.5; % 调头空间半径
371
372 % 标记是否找到最大速度
373 found_max_speed = false;
374
375 % 开始迭代速度，从1m/s逐步增加
376 while ~found_max_speed
```

```
% 初始化时间数组
time_steps = start_time:dt:end_time;
fprintf("当前速度:%0.2f m/s\n", v_head);

% 循环时间步长
for t_idx = 1:length(time_steps)
    % 当前时刻 t_now 和 t+0.001 秒时刻
    t_now = time_steps(t_idx);
    fprintf("当前时刻:%0.2f s\n", t_now);
    delta_t = dt_small;
    if t_now == 0
        delta_t = 1;
    end
    t_small = t_now + delta_t;

    % 获取所有把手在 t_now 时刻的位置
    pos_now = calculate_positions_at_time(t_now, v_head); % 返回一个 2*n_sections 大小的向量

    % 获取所有把手在 t+0.001s 时刻的位置
    pos_next = calculate_positions_at_time(t_small, v_head); % 返回一个 2*n_sections 大小的向量

    % 将 pos_now 和 pos_next 重新分为 x, y 坐标
    x_now = pos_now(1:2:end); % 当前时刻所有把手的 x 坐标
    y_now = pos_now(2:2:end); % 当前时刻所有把手的 y 坐标
    x_next = pos_next(1:2:end); % t+0.001s 时刻所有把手的 x 坐标
    y_next = pos_next(2:2:end); % t+0.001s 时刻所有把手的 y 坐标

    % 计算速度 (vx, vy), 然后计算总速度
    v_x = (x_next - x_now) / delta_t;
    v_y = (y_next - y_now) / delta_t;

    % 计算每个把手的速度并存储
    velocity = sqrt(v_x.^2 + v_y.^2);

    % 检查是否有把手速度超过了2m/s
    exceeding_indices = find(velocity > max_v); % 超过2m/s的把手索引
    if ~isempty(exceeding_indices)
        % 记录第一个超出2m/s的把手信息
        exceeding_speed = velocity(exceeding_indices(1));
        exceeding_section = exceeding_indices(1);
        exceeding_time = t_now;
        found_max_speed = true;
        break;
    end

    % 检查所有把手是否都在调头空间外
    in_space = sqrt(x_now.^2 + y_now.^2) - R_space <= 0.5;
    if all(~in_space) % 如果所有把手都在调头空间外
        break; % 结束当前速度的检测
    end
end

% 如果发现速度超过2m/s, 跳出循环
if found_max_speed
    v_head = v_head - v_step; % 回退到上一个速度作为最大速度
    break;
```

```
    else
        v_head = v_head + v_step; % 增加速度
    end
end

% 输出最终的最大速度
fprintf('龙头的最大行进速度为: %.2f m/s\n', v_head);
% 输出超出2m/s的具体速度、把手编号和时刻
if found_max_speed
    fprintf('超出2m/s的速度为: %.6f m/s\n', exceeding_speed);
    fprintf('超速的把手编号为: %d\n', exceeding_section);
    fprintf('超速发生的时间为: %.2f 秒\n', exceeding_time);
end
```