基于光纤传感器数据的曲率估算与平面曲线重构技术研究

摘要

光纤传感器在结构监测中至关重要。基于光的传输和反射，通过测量光信号的强度或相位变化来感知周围环境的物理量，如应变、温度和压力等。光纤传感器被广泛应用于桥梁、建筑、航空航天乃至医学仪器探测领域，具有高灵敏度、快速响应、抗干扰能力强等优点，在工程结构的监测与管理中发挥着重要作用。

本文运用曲率估算、**三次样条插值方法**、**四阶龙格-库塔法**，分析光纤传感器受力后的曲率变化，对形变的光纤进行平面曲线重建。曲率作为描述曲线弯曲程度的重要参数，在结构分析中具有重要意义。对光纤传感器曲率变化的估算和分析，能够更准确理解结构在受力作用下的形变，为工程实践提供可靠的数据支持。

针对问题一，运用**线性插值法**、**三次样条插值法**进行点位曲率估算。根据给定公式 $k = c\left(\frac{\lambda-\lambda_0}{\lambda_0}\right)$ 计算光纤传感器受力后各测点的曲率变化，采用线性插值和三次样条插值方法实现曲率的平滑估算。线性插值作为初步方法能够快速估算曲率，三次样条插值方法能够更好地模拟曲率在测点间的平滑过渡，提高估算精度和平滑度，从而全面地了解光纤在受力作用下的变形情况，计算得测试 1、2 各点位曲率值 **2.21**、**2.98** 左右。最后对公式中的 c 值进行**灵敏度分析**。

针对问题二，进行平面曲线重建与曲线特征分析。首先基于三次样条插值计算曲率函数，描述曲线的弯曲程度。接下来，运用**四阶龙格-库塔法**设立微分方程表述曲线 x 和 y 坐标以及切线方向 θ 随弧长 s 的变化关系。通过数值积分，从初始条件出发进行方程积分，重构曲线形状。使用 python 中的 seaborn 库绘制测试 1、2 的重构曲线，直观展示从理论模型到具体实现的过程。最终，对平面曲线进行几何构型与实用性分析。

针对问题三，依据给定函数进行误差分析与优化。首先，计算曲线的曲率和弧长参数化，理解曲线的特性。然后采用数值积分和四阶龙格-库塔法进行积分，重构曲线，并分析可能导致误差的因素，如数值积分误差、采样密度和计算中的近似，对采样密度进行 **MSE 均方误差检验**。通过比较重构曲线与原始曲线，深入探讨数值方法在解决实际数学问题中的应用，并提出优化建议，以提高计算精度和稳定性。

本文提出的模型能够准确估算曲率和重构曲线，为结构分析提供可靠数据支持，具有计算精度高、稳定性好的优点。通过对光纤传感器的曲率估算、曲线重构和误差分析，能够更全面地理解结构在受力作用下的变形情况，为工程实践提供重要参考依据。

关键字：**平面曲线重构　三次样条插值　四阶龙格-库塔法 常微分方程数值积分**

目录

1 问题重述

1.1 问题背景

光纤传感器技术，依托于光纤及光通信技术的飞速发展，已经成为结构健康监测领域的核心技术之一。这种技术利用光波参数（如波长、相位、强度）的变化来感应和响应外界环境的变动。其出色的灵敏度和抗电磁干扰能力使得光纤传感器在实时获取结构应变信息方面表现卓越，进而可以通过解调技术准确重建出结构的实际形态。

图 1　光纤传感器技术在各领域的应用

光纤传感器的应用范围广泛，除了在工业领域用于监控建筑和机械结构的稳定性外，还扩展到医疗领域，如用于人体内部结构的形态重建，展示了其技术的多样性和重要性。因此探究光纤的重构对生产生活具有重要的现实意义。

1.2 问题概述

问题 1 要求根据给定的波长测量数据，构建数学模型，估算平面光栅各个传感点 (FBG1-FBG6) 的曲率。进一步，在已知初始点坐标和初始光纤方向的情况下，估算出指定横坐标位置处的曲率。

问题 2 要求根据问题 1 中求得的曲率，构建数学模型，分别重构平面曲线，并对曲线的特点进行分析。

问题 3 要求在已知平面曲线方程的情况下，以适当的等间距弧长采样，计算采样点

的曲率，并基于这些曲率构建数学模型，重构平面曲线，并分析重构曲线与原始曲线出现误差的原因。

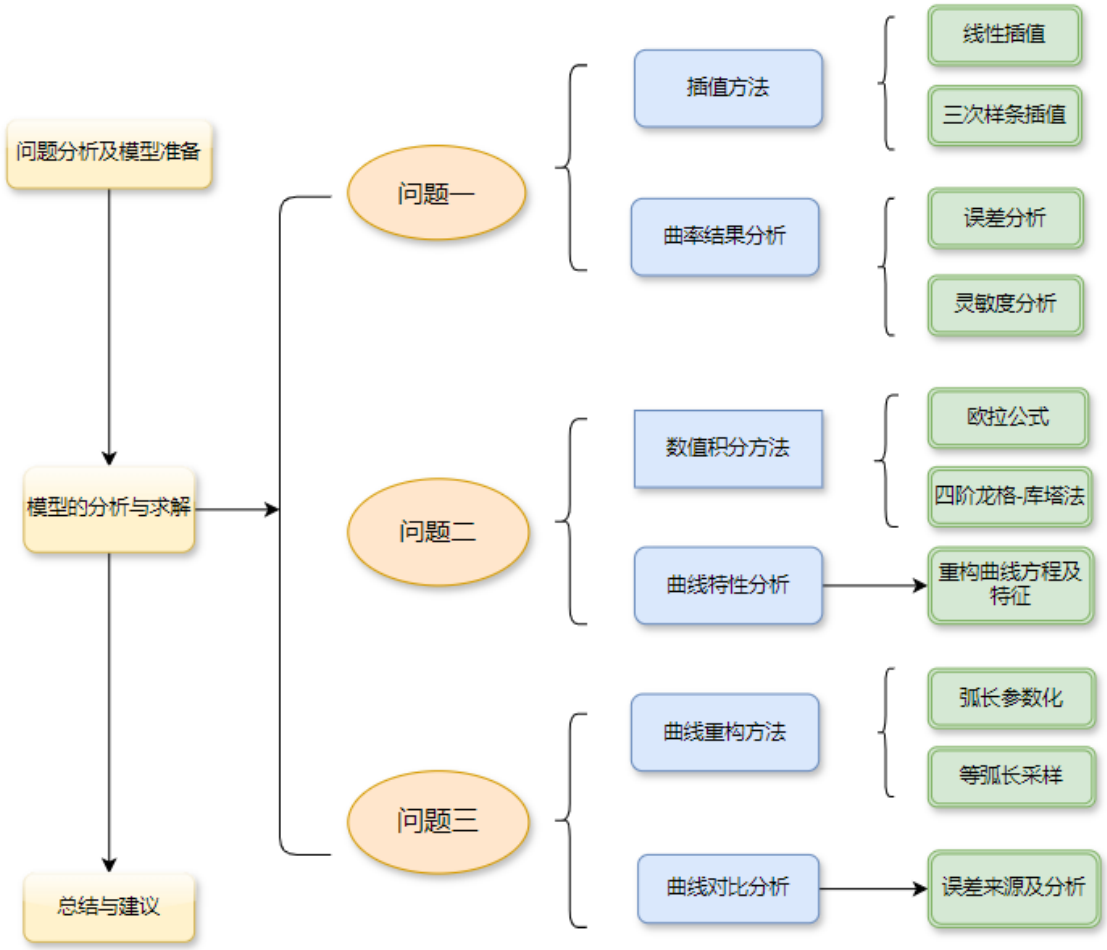


图 2　问题求解总思路

2 模型假设

1. 在受力前后，光纤传感器的形变可以近似为平面内的曲线变化。
2. 在进行插值计算时，假设光纤在任何两个测量点之间的变化是连续的，不存在突变或断裂。
3. 假设通过插值得到的曲率函数在整个曲线上是连续的，且足够光滑，即允许进行数值积分。重构的曲线具有足够的平滑性，即不会出现不合理的尖峰或异常变化。
4. 假设光纤材料的物理性质在其长度范围内是均匀的，即材料的密度、弹性模量等参数在整个光纤长度上保持不变。
5. 假设在插值计算中采用的数据点密度足够，能够充分反映曲线的形状特征，避免插值误差过大。同时采用的数值方法在近似计算过程中产生的误差是可控的，并且不

会引入过大的误差影响到最终结果的准确性。

3 符号说明

符号	含义	单位
λ_0	初始状态下的波长，表示未受外力影响时的波长	nm
λ	受力后的波长，表示受到外力影响后的波长	nm
c	比例常数，将波长变化转换为曲率的比例因子	m^{-1}
k	曲率，表示光纤在某一点的弯曲程度	m^{-1}
x	沿光纤的位置坐标，从光纤的起始点开始测量	m
$S_i(x)$	第 i 段三次多项式插值函数，用于在 x_i 到 x_{i+1} 之间近似曲率	m^{-1}
x_i	第 i 个传感器的位置坐标	m
$k(s)$	位置 s 处的曲率，从插值模型中得到	m^{-1}
$\theta(s)$	曲线在弧长 s 处切线与 x 轴的夹角	$\circ$
$x(s), y(s)$	曲线在弧长 s 处的笛卡尔坐标	/
s	曲线的弧长参数，用于描述曲线上的点	/
$\frac{dx}{ds}, \frac{dy}{ds}$	曲线的 x 和 y 坐标关于弧长的导数，表示曲线的切线方向	/
$x, y(x)$	曲线上的点及其对应的函数值	/
y', y''	曲线 y 关于 x 的一阶和二阶导数	/

注：表中未说明的符号以首次出现处为准

4 问题一模型的建立与求解

4.1 问题分析

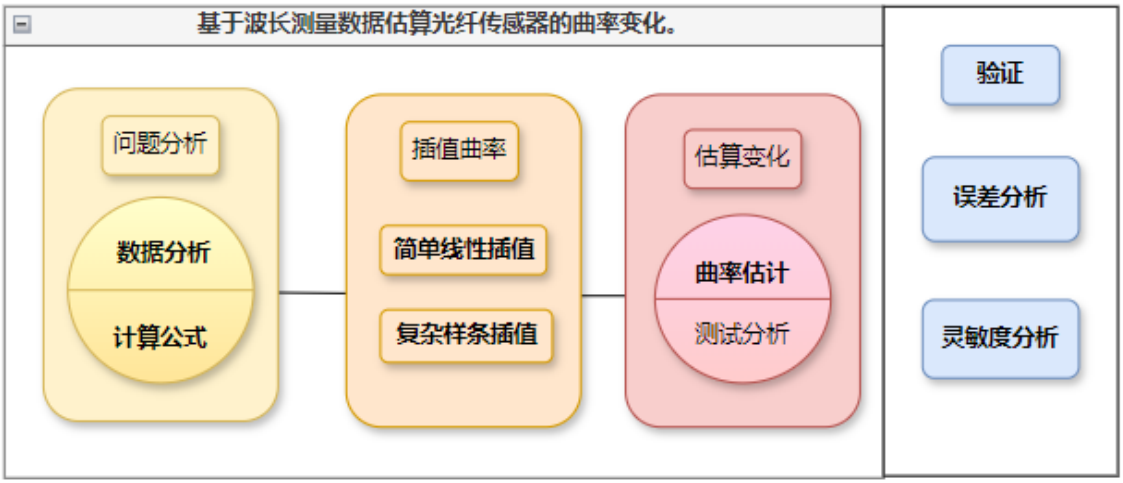


图 3　问题一流程图

首先根据题目中给出的关系 $k = c\left(\frac{\lambda-\lambda_0}{\lambda_0}\right)$，计算每个测量点传感器的曲率，其中 c 为常数，题目中给出的值为 4200，λ_0 和 λ 分别是各传感器受力前后的波长。曲率 k 与路径的曲率半径 R 有 $k = \frac{1}{R}$ 的关系。

表 1　测量点曲率

测量点	测试 1 曲率	测试 2 曲率
FBG1	2.219490	2.986364
FBG2	2.216743	2.978182
FBG3	2.233224	2.972727
FBG4	2.230477	2.980909
FBG5	2.235971	2.983636
FBG6	2.222237	2.975455

为简化计算，假设光纤段之间的曲率是恒定的。通过积分曲率可以估算光纤的形状。下图为各传感点曲率变化，由图可知各传感点曲率数据较为一致。

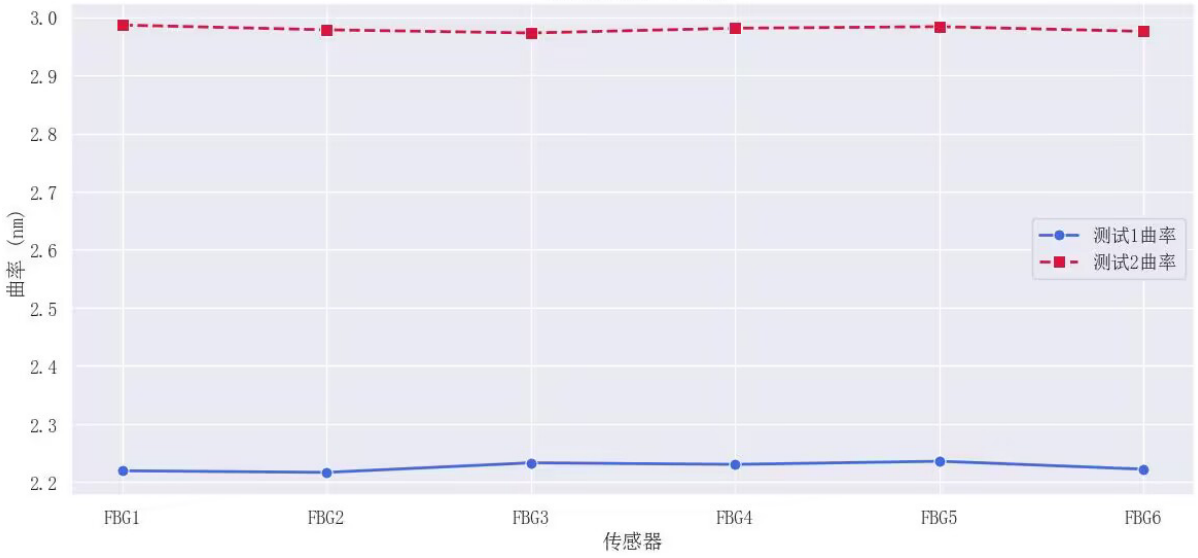


图 4　各传感点曲率变化

4.2 模型的建立与求解

4.2.1 线性插值法估计间断点曲率

线性插值来估计间断点的曲率。例如，如果要估计 0.3 米处的曲率，可以根据 FBG1（0 米）和 FBG2（0.6 米）之间的曲率来插值。

线性插值是一种简单而有效的建模方法，特别适用于平面曲线重建。其基本思想是假设相邻数据点之间的曲线是线性的，通过连接这些数据点来近似曲线形状。假设有一组数据点 $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$，其中 x_i 是自变量，y_i 是因变量。

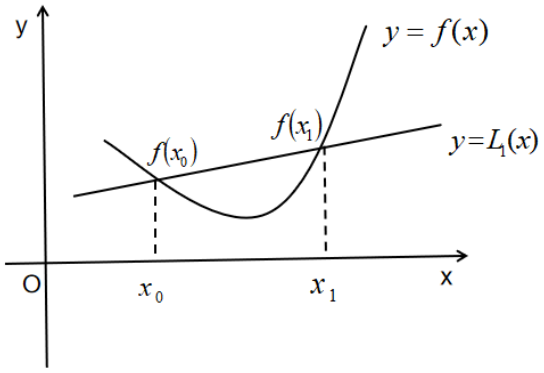


图 5　线性插值法示意图

线性插值的基本流程如下：

$$m_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \tag{1}$$

首先计算每相邻两点之间的斜率:(1) 所示

接着进行插值计算:

$$y = y_i + m_i \cdot (x - x_i) \tag{2}$$

对于给定的插值点 x，其对应的插值 y 由公式 (2) 给出，其中 $x_i \leq x \leq x_{i+1}$，y_i 是与 x_i 对应的因变量值。

其中:

$(x - x_1)$ 表示 x 与 x_1 的距离

$\frac{y_2 - y_1}{x_2 - x_1}$ 表示数据在 x 方向上的变化率

这个公式可以直观地理解为，在 (x_1, y_1) 和 (x_2, y_2) 之间的直线上，x 点处的 y 值由直线上相邻两点的线性关系来估计。

下图为经过线性插值法得到的光纤传感器曲率:

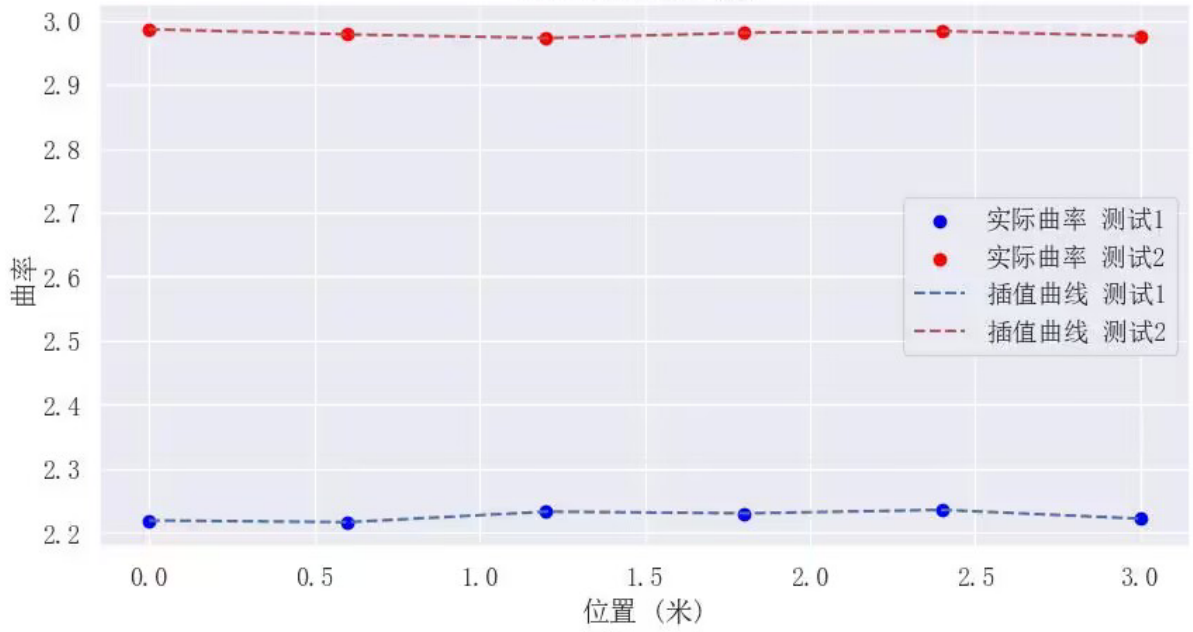


图 6　光纤传感器曲率线性插值

由此得到线性插值法测得的测试 1、2 对应各横坐标点的曲率:

表 2　线性插值法测得曲率

横坐标 (米)	0.3	0.4	0.5	0.6	0.7
测试 1 曲率	2.218116	2.217659	2.217201	2.216743	2.219490
测试 2 曲率	2.982273	2.980909	2.979545	2.978182	2.977273

4.2.2 三次样条插值法估计间断点曲率

采用线性插值，虽然计算简单，也有一致收敛性，但不能满足某些工程技术的高精度要求，因此有必要引入三次样条插值。通过使用三次样条插值，我们可以以光滑的方式逼近数据，同时保持较低的振荡和误差。

三次样条插值的基本思想是将插值区间分成若干段，并在每个段上使用一个三次多项式来逼近数据。这些多项式称为样条函数，它们在相邻段上保持连续性和光滑性，以确保整体插值函数的平滑性。在三次样条插值中，每个样条函数都由四个未知系数确定，因此对于每个数据段，我们有四个未知数，这些未知数通过插值条件得到。

假设我们有一组数据点 $\{(x_i, y_i)\}_{i=0}^{n}$，其中 x_i 是已知的数据点的自变量，y_i 是对应的因变量。我们的目标是找到一个三次样条函数 $S(x)$，它在每个相邻数据点间 $[x_i, x_{i+1}]$ 上都是三次多项式。假设在每个区间 $[x_i, x_{i+1}]$ 上，我们有：

1. $S_i(x)$ 是关于 x 的三次多项式，表示第 i 个区间内的样条函数。

2. $S_i(x_i) = y_i$ 和 $S_i(x_{i+1}) = y_{i+1}$，即样条函数经过相邻数据点。

3. $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$ 和 $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$，即相邻样条函数在连接处的一阶和二阶导数相等，确保连续性和光滑性。

综合以上条件，我们可以得到每个区间内的样条函数 $S_i(x)$ 表达式为：

$$S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \tag{3}$$

其中，a_i, b_i, c_i, d_i 是待定系数，通过插值条件求解。

为了求解每个区间内的待定系数，我们需要满足以下条件：

1. 在每个数据点 x_i 处，样条函数经过给定的因变量值：$S_i(x_i) = y_i$。

2. 在每个内部节点处 x_{i+1}，相邻样条函数的一阶和二阶导数相等：

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \tag{4}$$

通过构建线性方程组，可以解出每个区间内的待定系数，从而确定整个插值函数 $S(x)$。

当使用三次样条插值时，通常需要指定边界条件来完全确定插值函数。常见的三种边界条件是：

1、自然边界条件

自然边界条件要求在插值函数的端点处，二阶导数为零。这意味着在插值函数的端点处，插值曲线是一条直线，而不是弯曲的。自然边界条件的好处是插值函数在端点处的形状更加平滑，适用于大多数情况。

2、固定边界条件

固定边界条件可以指定插值函数在端点处的一阶导数或者函数值。例如，如果我们希望插值函数在端点处的一阶导数为 $f'(x_0)$ 和 $f'(x_n)$，则固定边界条件可以表示为：

$$S'(x_0) = f'(x_0), \quad S'(x_n) = f'(x_n) \tag{5}$$

或者，如果我们希望插值函数在端点处的函数值为 y_0 和 y_n，则固定边界条件可以表示为：

$$S(x_0) = y_0, \quad S(x_n) = y_n \tag{6}$$

3、周期边界条件

周期边界条件要求插值函数在给定区间的端点处具有相同的函数值和导数。对于一个周期性的插值问题，周期边界条件可以表示为：

$$S(x_0) = S(x_n), \quad S'(x_0) = S'(x_n) \tag{7}$$

选择合适的边界条件取决于具体的应用场景和问题需求。在实际应用中，需要根据数据的特性和问题的约束来选择最适合的边界条件。

此题我们采用自然边界条件，即假设端点处二阶导数为 0。下图为经过三次样条插值法得到的光纤传感器曲率：

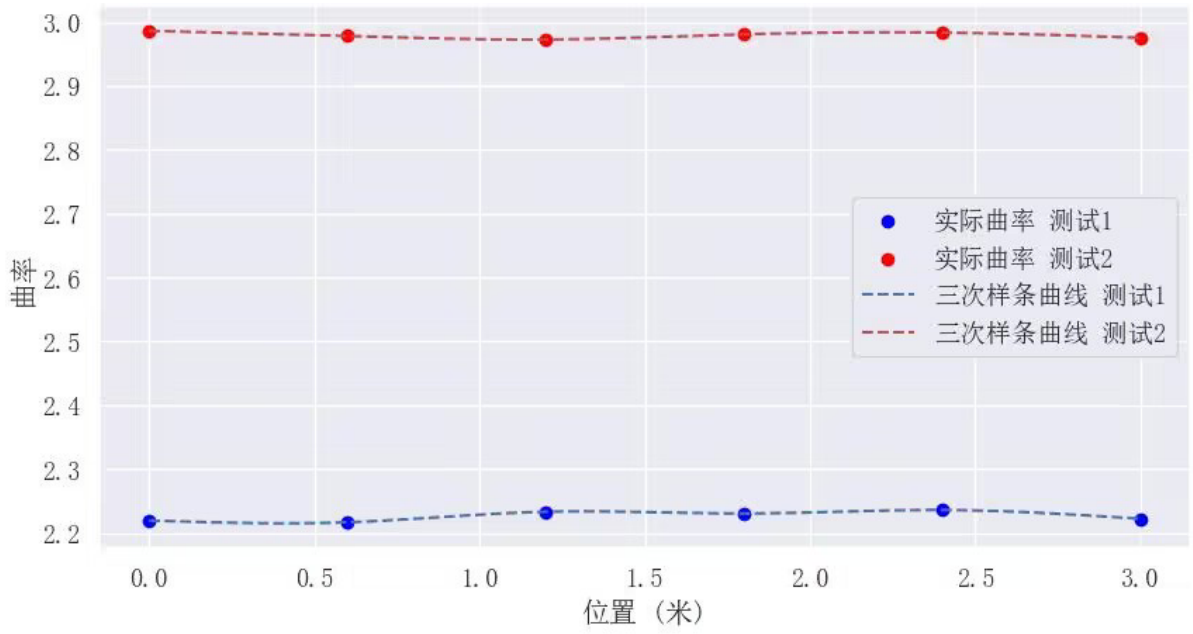


图 7　光纤传感器曲率三次样条插值

由此得到三次样条插值法测得的测试 1、2 对应各横坐标点的曲率：

表 3　三次样条插值法测得曲率

横坐标 (米)	0.3	0.4	0.5	0.6	0.7
测试 1 曲率 k	2.215573	2.215147	2.215474	2.216743	2.219043
测试 2 曲率 k	2.982385	2.981020	2.979622	2.978182	2.976713

4.3 c 值灵敏度分析

曲率估计中参数 c 的影响分析

图表显示两组测试数据的平均曲率随着 c 的增加都表现出线性上升的趋势，这意味着曲率与 c 值成正比关系。这符合曲率计算公式

$$\kappa = c\left(\frac{\lambda - \lambda_0}{\lambda_0}\right) \tag{8}$$

其中 c 是比例系数。对于 Test 1 数据，曲率的增加较为温和，这意味着在这个测试条件下，曲率对 c 的变化不是特别敏感。相比之下，Test 2 数据显示曲率对 c 的变化更为敏感，这表明在测试 2 的条件下，同样的 c 值变动会引起更大的曲率变化。

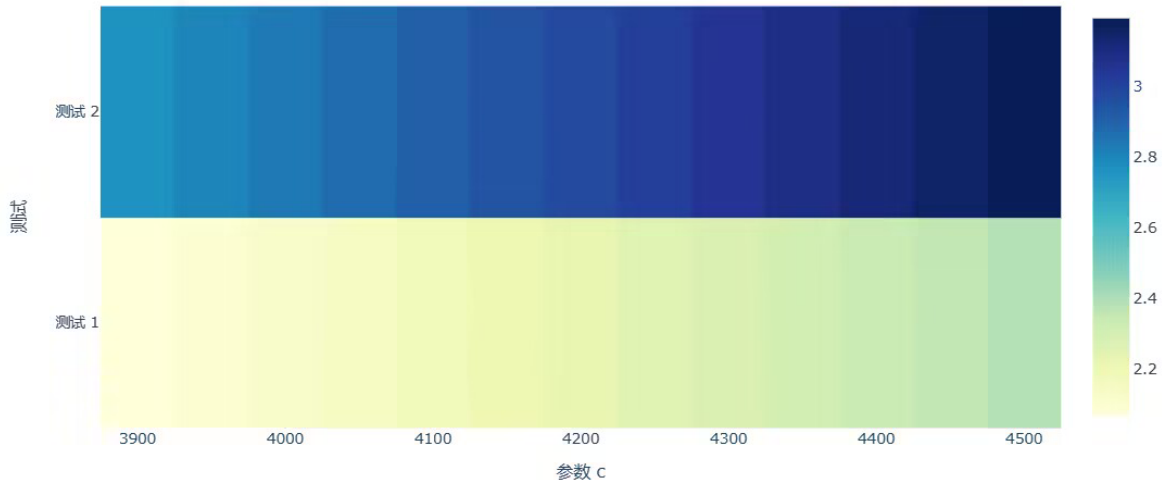


图 8　参数 c 的灵敏度检验

c 值的影响

c 值体现了波长变化到曲率变化的敏感度，即 c 值越大，同样的波长变化会导致更大的曲率估计变化。在实际应用中，如果 c 值选择不当，可能会导致曲率的估计不准确。这对于使用光纤传感器进行精确测量的工程应用来说至关重要，例如在结构健康监测或医疗诊断设备中。

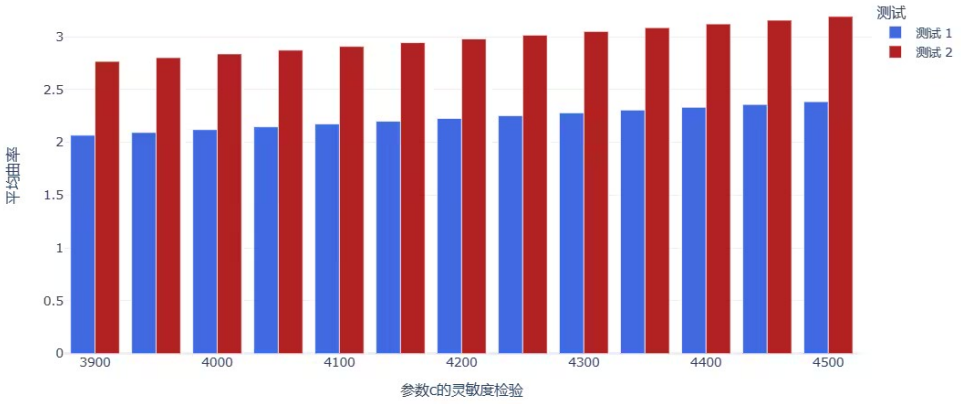


图 9　参数 c 的灵敏度检验

敏感度分析显示 c 值的选择对于曲率的准确估计非常关键。如果 c 的估计或选择过于保守（太小），则可能低估实际曲率，反之亦然。在设计传感器和进行实验设置时，选择正确的 c 值尤为重要。通过敏感度分析，我们可以确定一个安全的 c 值范围，以保证曲率估计在可接受的误差范围内。最终，确定 c 值的最优范围可以通过实验进一步校准和验证，以确保光纤传感器数据解释的准确性。

此分析揭示了参数 c 在曲率估计中的重要作用，并强调了准确确定该参数对于确保曲率测量准确性的重要性。未来的工作应当包括实验验证分析中确定的 c 值范围，并根

据应用领域对其进行适当的调整和优化。

5 问题二模型的建立与求解

5.1 问题分析

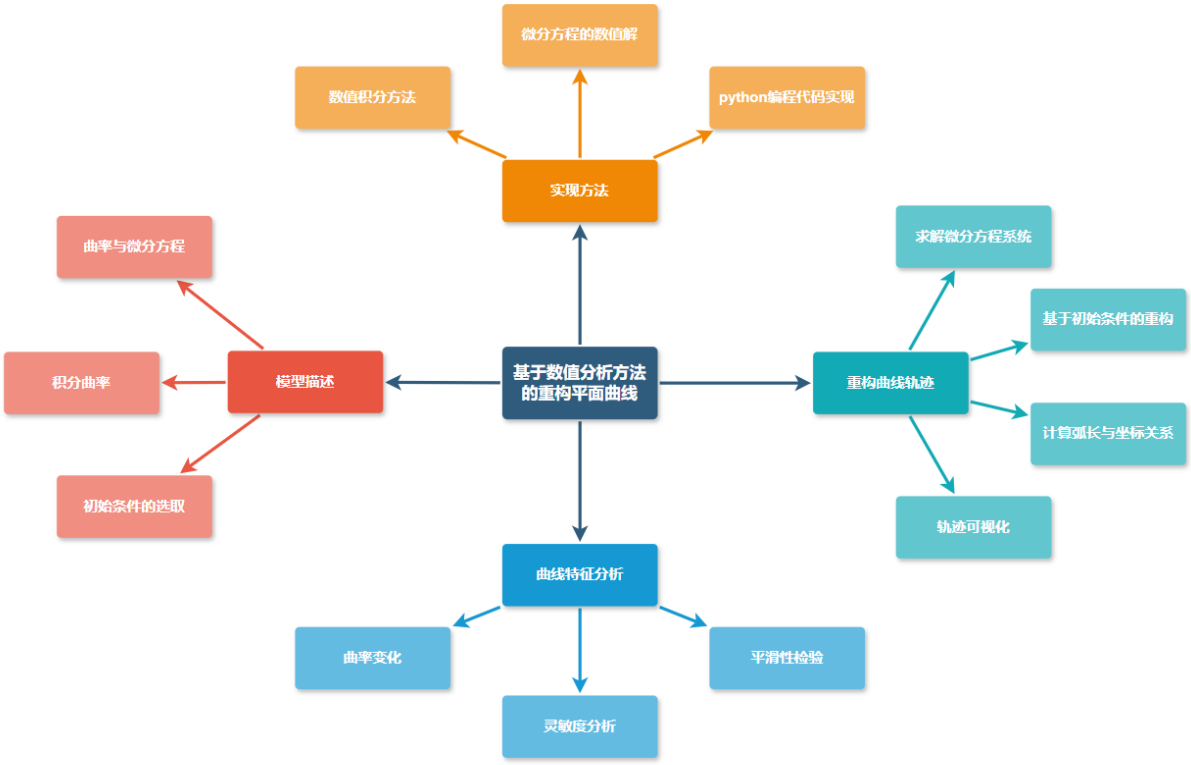


图 10　问题二流程图

问题二要求我们利用问题一中获取的曲率数据来重新构建平面曲线，并进一步分析这些曲线的特性。本文将采用曲率积分的方法来恢复光纤的平面形态。

曲率 k 是用来描述曲线在某一特定点处的弯曲程度的物理量。对于平面曲线而言，曲率 k 与曲率半径 R 之间存在直接的关系，即 $k = \frac{1}{R}$。这意味着，如果我们知道曲线的曲率函数，就可以通过积分的方式来确定曲线上任意点的位置。具体来说，从曲线的某一起点开始，结合该点的初始方向信息，我们可以逐步积分曲率函数，从而计算出曲线上每一点的具体方向和位置。

5.2 模型建立

龙格-库塔方法是解决一阶常微分方程组初值问题在区间 $[a,b]$ 上的数值解法。

$$\begin{cases} \frac{\mathrm{d}y}{\mathrm{d}x} = f(x,y) \\ y\left(x_0\right) = y_0 \end{cases} \tag{9}$$

龙格-库塔方法基于泰勒级数展开，但是它不需要直接计算高阶导数，而是通过多个中间点的估计来近似解的行为。其基本思想是利用当前点的信息，预测下一个点的值，具体通过以下步骤实现：

- Step1：估计斜率：在每一步，根据已知点的导数（斜率）来估计函数值。
- Step2：组合斜率：计算一步之内的多个斜率，然后将这些斜率组合起来，形成一个更好的总体斜率估计。
- Step3：更新解：使用这个组合斜率来更新解。

按照上述思想，龙格-库塔方法的一般形式设定为：

$$\begin{cases} y_{n+1} = y_n + \sum_{i=1}^{r} \omega_i K_i \\ K_1 = h y_{n+1} \\ K_i = hf\left(x_n + \alpha_i h, y_n + \sum_{j=1}^{i-1} \beta_{ij} K_j\right) \quad (i = 2,3,\cdots,r) \end{cases} \tag{10}$$

其中 $\omega_i, \alpha_i, \alpha_i$ 为待定常数。从上式可以看到用它计算 y_{n+1} 需要计算 r 次 $f(x,y)$ 的值，故此式称为 r 阶 R-K 方法。

由问题一的假设可知，曲线的原始点位于原点 $(0,0)$，初始的水平光纤方向为 x 轴，垂直方向为 y 轴，光纤在平面内受力后在初始位置的切线与水平方向的夹角为 $45°$，即 $\theta(0) = 45°$，其中 θ 是切线方向与水平方向的夹角。

由曲率的定义，我们有：$\frac{d\theta}{ds} = k(s)$，其中 s 是曲线的弧长参数。已知 $k(s)$，可以通过积分得到 $\theta(s)$。更进一步，曲线上任意点的坐标 $(x(s), y(s))$ 可以通过以下方程计算：$\frac{dx}{ds} = \cos(\theta(s))$，$\frac{dy}{ds} = \sin(\theta(s))$。

通过对这些方程进行积分，我们可以从已知的起始条件 $x(0) = 0, y(0) = 0$ 推导出曲线的形状。即求解三个常微分方程的初值问题：

$$\begin{cases} \frac{\mathrm{d}\theta}{\mathrm{d}s} = k(s) \\ \theta(0) = 45° \end{cases} \tag{11}$$

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}s} = \cos(\theta(s)) \\ x(0) = 0 \end{cases} \tag{12}$$

$$\begin{cases} \frac{\mathrm{d}y}{\mathrm{d}s} = \sin(\theta(s)) \\ y(0) = 0 \end{cases} \tag{13}$$

我们根据四阶龙格-库塔 RK4 方法来求解上述微分方程组，四阶提供了较高的精度，这个方法对每个积分步骤使用以下形式的迭代：

$$\begin{cases} y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ t_{n+1} = t_n + h \\ k_1 = h - f(t_n, y_n) \\ k_2 = h - f(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ k_3 = h - f(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\ k_4 = h - f(t_n + h, y_n + k_3) \end{cases} \tag{14}$$

5.3 模型求解

按照下述流程进行四阶龙格-库塔（RK4）模型的求解：

1. 定义初始条件：- $x_0 = 0$ - $y_0 = 0$ - $\theta_0 = 45$ - h = 步长 - $s_{\text{start}} = 0$ - s_{end} = 积分范围末端

2. 定义微分方程函数 $f(t, y)$：- $f(t, y) = [\cos(\theta), \sin(\theta), k(s)]$

3. 使用四阶龙格-库塔方法进行迭代：

- 对于每个步长 h，重复以下步骤直到达到积分范围末端：

- 计算 $k_1 = h \cdot f(t, y)$

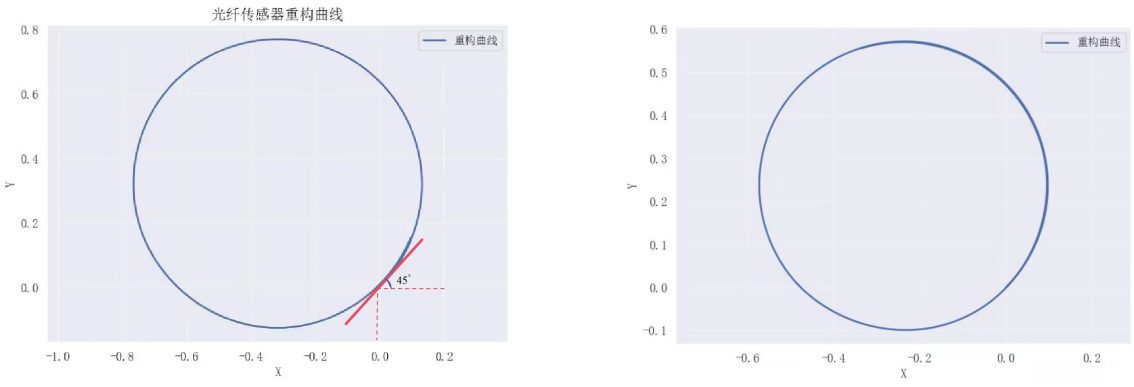
- 计算 $k_2 = h \cdot f(t + \frac{h}{2}, y + \frac{k_1}{2})$

- 计算 $k_3 = h \cdot f(t + \frac{h}{2}, y + \frac{k_2}{2})$

- 计算 $k_4 = h \cdot f(t + h, y + k_3)$

- 使用公式 $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ 更新 x、y 和 θ - 更新 t。

4. 将得到的 x 和 y 值存储起来，形成曲线的参数化表示，结果如下图。



(a) 测试 1 三次样条插值结果重构曲线　　**(b) 测试 2 三次样条插值结果重构曲线**

图 11　两个测试样本的三次样条插值结果重构曲线

5.4 平面曲线特点分析

几何特性分析：

1. 曲线的几何形态:

曲线的几何形态由曲率的分布决定。高曲率区域通常意味着曲线在较短的距离内实现了较大的方向改变，这可能指示了在这些区域光纤传感器感应到更大的外部力作用。反之，低曲率区域则表现为曲线更加接近直线，可能是外力作用较小或者材料自身性质导致的。

2. 敏感度与稳定性:

曲线的形状对初始条件的敏感度反映了系统的稳定性和可控性。例如，初始测量点的方向若有微小偏差，可能会导致曲线整体形态发生显著变化，这在工程应用中可能需要通过增加测量点或调整算法参数来控制。

3. 平滑过渡与物理真实性:

三次样条插值提供的平滑曲线保证了物理现象的真实性。在光纤曲线重构中，平滑性意味着力的作用和材料的响应在整个光纤中是连续的，没有突变。这有助于避免在实际应用中可能遇到的结构不稳定性。

实用性评估：

考虑到实际应用中的误差因素，如光纤的弯曲性和弹性，讨论这些因素对曲线形状的潜在影响。

光纤等弹性材料的弯曲会导致曲线的局部变形，从而影响重构曲线的几何属性。此外，制造和安装过程中的误差也会对曲线形状产生影响，使得其不能完全符合理想的圆形。这些因素可能导致实际曲线的长度略有偏差，并使得曲率在某些位置发生微小变化。因此，在实际应用中，需要综合考虑，对重构曲线的几何特性进行适当的调整和修正。

6 问题三模型的建立与求解

6.1 问题分析

图 12　问题三流程图

问题三要求我们找到满足方程组的平面曲线方程 $y = x^3 + x$。通过对方程曲线曲率的求解，计算求解的出发点并使用这些曲率数据来重构曲线，最后分析所得到的曲线与原始曲线之间的误差。

6.2 模型建立

首先，从判断给定的曲线方程 $y = x^3 + x$ 出发，计算曲线的切线斜率和曲率。问题三的核心在于利用曲率和弧长参数化技术来重构曲线。计算曲线上各点的曲率，然后以等间距弧长采样计算曲线上的点利用切线斜率和曲线的曲率来计算参数 k。下面使用对弧长的微分表示弧长参数化结果。

给定曲线为 $y = f(x) = x^3 + x$，首先计算微分 dy/dx:

$$\frac{dy}{dx} = 3x^2 + 1 \tag{15}$$

则弧长的微分 ds 的下式给出:

$$ds = \sqrt{1 + \left(\frac{dy}{dx}\right)^2}\, dx = \sqrt{1 + (3x^2 + 1)^2}\, dx \tag{16}$$

我们可以通过积分 $s = \int \sqrt{1 + (3x^2 + 1)^2} dx$ 来得到弧长关于 x 的函数表达式。这样，可以求解该函数的反函数，即 x 关于 s 的函数关系式。

求解过程可以使用数值求解方法。

曲率公式

弧长对应的曲率 k 可以通过以下公式计算：

$$k = \frac{|y''|}{(1 + (y')^2)^{\frac{3}{2}}} \tag{17}$$

其中 $y' = \frac{dy}{dx}$ 和 $y'' = \frac{d^2y}{dx^2}$，对于 $y = x^3 + x$:

$$\begin{aligned} y' &= 3x^2 + 1 \\ y'' &= 6x \end{aligned} \tag{18}$$

曲率半径的综合表达与曲线重构

曲率半径 R 是曲线在任一点处的局部几何属性，提供了曲线弯曲程度的度量。对于曲线 $y = x^3 + x$，曲率半径可以通过曲率 k 计算得到：

$$R = \frac{1}{k} = \frac{(1 + (y')^2)^{\frac{3}{2}}}{|y''|} \tag{19}$$

这一值对于曲线形状的精确重构至关重要。重构曲线需要利用曲率与弧长关系的逆映射，即运用第二问的龙格-库塔方法。通过 matplolib 工具实现重构曲线轨迹的可视化。

6.3 模型求解

首先运用绘图工具将原始曲线与各点曲率可视化：

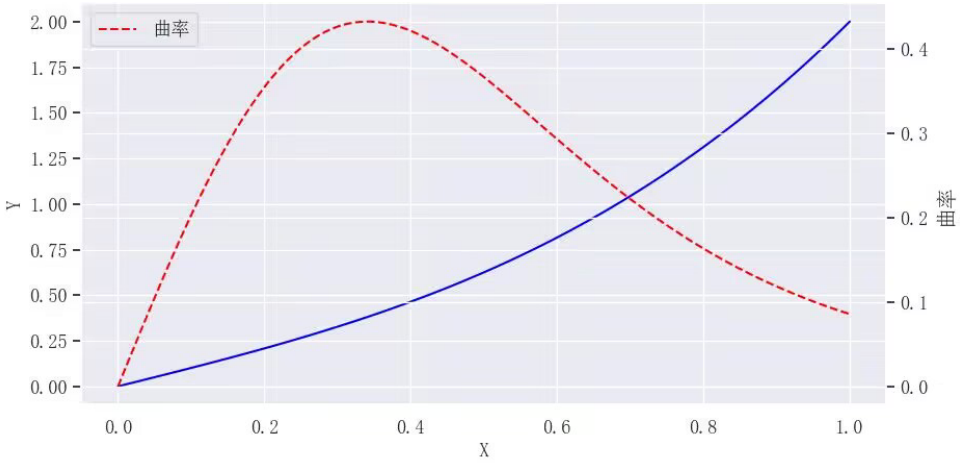


图 13　原始曲线与曲率

计算原始曲线弧长，等弧长取 10 个采样点并进行平面曲线重建，重建结果如下图：

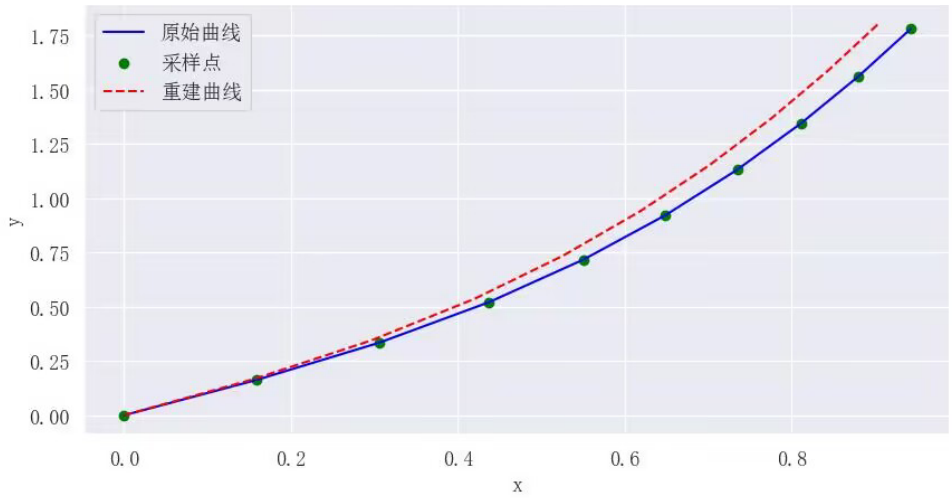


图 14　原始曲线与 10 采样点重建曲线比较

下表为 10 个采样点与对应曲率：

采样点	曲率	采样点	曲率
0.000	0.000	0.550	0.330
0.159	0.300	0.649	0.257
0.306	0.428	0.735	0.200
0.437	0.405	0.811	0.157
0.880	0.126	0.943	0.103

表 4　采样点和曲率的对应关系

计算原始曲线弧长，等弧长取 30 个采样点并进行平面曲线重建，重建结果如下图：

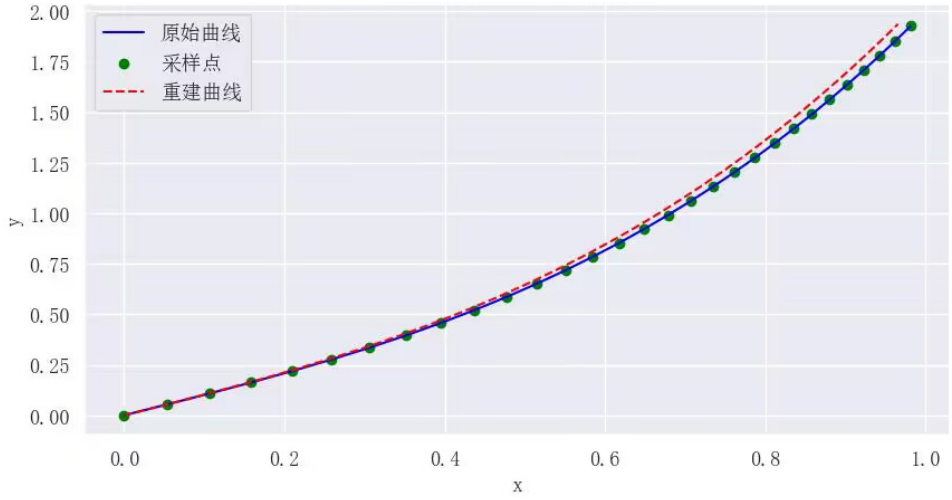


图 15　原始曲线与 30 采样点重建曲线比较

6.4 结果分析

误差分析与优化

在数值计算过程中，我们分析了可能导致误差的多个来源，包括曲率计算误差、计算过程中的数值方法误差，以及采样间隔的选取对曲线重构精度的影响。

1. **数值积分误差**：
   在求解曲线的弧长时，由于数值积分方法是基于对函数曲线局部线性或二次多项式的逼近，这一过程里会产生截断误差。由于重构曲线的方程在某些区域曲率迅速变化，常规数值积分方法可能无法捕捉这些变化，从而导致计算得到的弧长与真实值存在差异。
2. **曲率计算误差**：
   另一方面，曲率的计算依赖于函数的一阶导数和二阶导数，尽管对于解析表达式来说，这些导数可以精确计算，但数值方法在计算导数时会引入误差。这些误差源自于差分间隔的选择和数值舍入，尤其是在曲线拐点附近，即使微小的计算误差也会在曲率估计中产生较大的影响。
3. **采样密度误差**：

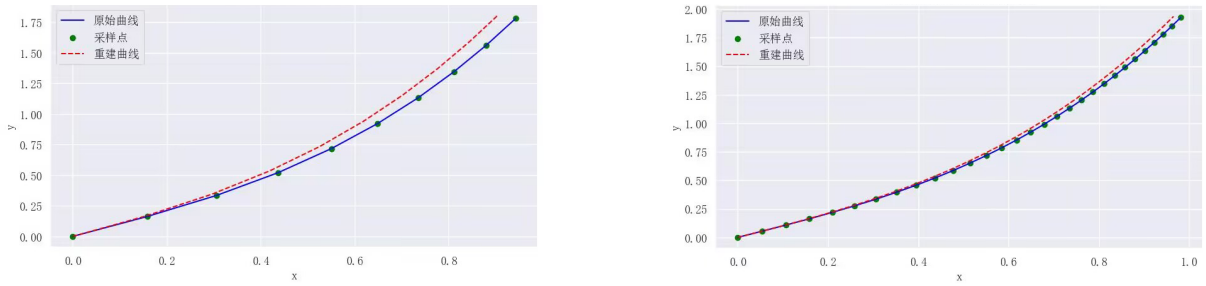


图 16　取点密度与拟合效果比对

同时，在对曲线进行离散采样时，采样点的密度决定了重构曲线的精确度。如果采样点过于稀疏，可能会错过曲线的关键特征。反之，过于密集的采样可能导致计算量过大，而且增加数值误差的风险。在某些曲率变化大的区域，等弧长采样的密度可能存在不能精确捕捉曲线的所有细节。

为了减少这些误差，我们可以采用更高精度的数值方法，如高阶 Runge-Kutta 方法，以及适当调整采样间隔，尤其是在曲率变化大的区域。我们还绘制了原始曲线和重构曲线的轨迹对比图，通过这种可视化的形式我们可以更直观的理解曲率的变化导致的误差以及分析重构曲线的拟合精度效果。

MSE 均方误差分析

均方误差计算：

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{20}$$

其中，n 是采样点个数，y_i 是第 i 个样本的实际目标值，$\hat{y}_i$ 是第 i 个采样点的模型预测值。

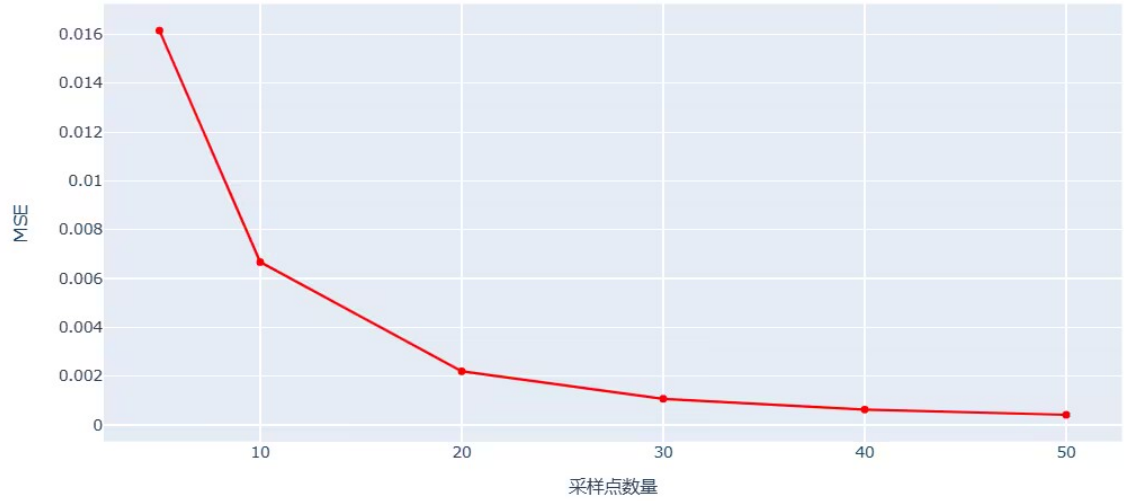


图 17　MSE 均方误差分析

通过计算每个样本预测值与实际值之间的差的平方后取平均值，可以得到均方误差。通过上图可知，随着采样点增加，重构曲线与原始曲线的拟合度增强，均方误差在采样点超过 20 后显著降低，最终趋向 0。

7 模型评价

7.1 模型优点

1. 模型简化与快速部署：
   线性插值模型以其明了的数学结构和易于操作的特性，为我们在面临需要迅速作出估算时提供了方便，它简化了问题，加快了解决方案的实施速度。
2. 平滑过渡与精细呈现：
   采用三次样条插值的方法，得到的曲线在各数据点间过渡自然，平滑连贯，这为描绘复杂的数据形态提供了更为细腻和真实的模拟，尤其适用于需要高精度建模的场合。
3. 非线性适配的准确性：
   在处理具有非线性特征的数据时，三次样条插值以其优越的适应能力展现出高准确度，它能够有效捕捉并再现物理世界中的连续动态和曲率变化，从而提升模型的实

际应用价值。

4. 结构分析的精确工具：
   三次样条插值不仅在整个应用区域内确保了曲率连续和可导，还为工程领域中的力学分析和结构监测提供了精确的分析手段，是一个在技术实施中不可或缺的工具。

7.2 模型缺点

1. 数据质量对模型精准性的显著影响：
   模型的精细度及其预测精确性，受制于所依赖数据的质量与其分布的均匀性。数据的任何偏差或是其稀疏性，都是诱发插值偏误的潜在源泉。
2. 边界处的不稳定性问题：
   在边缘领域，三次样条插值法往往面临振荡的挑战，特别是在数据采样较为稀疏的区域，这一现象尤为凸显，成为模型应用的一个不可忽视的局限。
3. 泛化与适应未知环境的受限性：
   缺乏实验验证的模型，其在未知或未充分探究的物理环境下，所显示出的泛化能力和适应性受到限制，这在应用于多变环境时，可能会成为一个显著的弱点。

7.3 模型改进及未来展望

1. 数据采集精度的量子跃进：
   通过融入最尖端的测量科技，例如分布式光纤传感器，可以极大地提升数据点的密集度和测量频率，进而达到量子级的数据精度提升。这种技术的引进不仅增强了数据的实时性，也为数据的高维度分析提供了坚实的基础。
2. 模型演化的深度递进：
   探索引入更为高级的插值手段及数值求解策略，如有限元方法，以应对更为复杂的几何形态和动态加载条件。这样的发展不仅优化了现有模型的处理能力，更是对其功能性的多维度扩展，为解决高阶问题提供了新的解决框架。
3. 人工智能技术的创新融合：
   在预测曲率变化这一多维复杂系统中，采用机器学习技术作为一个创新的途径，特别是对于大数据和非线性变化的场景。利用这些算法能够显著提升预测的精准度和模型的自适应性，为传统模型的局限性提供了智能化的解决方案。
4. 多物理场交互作用的集成分析：
   通过在模型中整合多物理场耦合作用的分析，如热力学、湿度学等，来全方位评估并优化光纤传感器在不同环境下的性能表现。这种多维耦合的分析方法，能够实现对结构健康监测系统的全面革新，确保了监测结果的高度综合性和可靠性。

参考文献

[1] 田金容. 基于多芯光纤和光频域反射的三维曲线重构方法研究 [D]. 华中科技大学, 2022.

[2] Shuyang C, Fengze T, Weimin L, et al. Deep learning-based ballistocardiography reconstruction algorithm on the optical fiber sensor[J]. Optics express, 2022, 30 (8): 13121-13133.

[3] 吕安强, 黄崇武, 乐彦杰, 等. 基于分布式应变的三芯光纤形态重构算法研究 [J]. 光电子 · 激光, 2021, 32 (07): 784-790.

[4] 闫洁, 李伟, 姜明顺, 等. 基于光纤光栅传感器的板状结构形态感知与三维重构技术 [J]. 中国激光,2020,47(11):231-240.

[5] 冯荻. 基于光纤光栅应变传感的结构变形重构技术研究 [D]. 大连理工大学, 2020.

[6] 陈世凯. 光纤光栅重构方法研究及实验 [D]. 国防科学技术大学, 2016.

[7] 章亚男, 肖海, 沈林勇. 用于光纤光栅曲线重建算法的坐标点拟合 [J]. 光学精密工程, 2016, 24(09): 2149-2157.

[8] 肖海, 章亚男, 沈林勇, 等. 光纤光栅曲线重建算法中的曲率连续化研究 [J]. 仪器仪表学报, 2016, 37(05): 993-999.

[9] Chaojie M ,Jianglei D ,Yi Z , et al.Reconstruction of structured laser beams through a multimode fiber based on digital optical phase conjugation[J].Optics Letters,2018,43(14):3333-3333.

[10] 司亚文. 基于分布式光纤传感器的变形监测与应变场重构方法研究 [D]. 南京航空大学学,2019.

[11] SHI C，GIANNA R OU S，LEE S L，et al. Simultaneous catheter and environment modeling for trans-catheter aortic valve implantation［C］. IEEE/R SJ International Conference on Intelligent R obots and Systems，2014: 2014-2029.

[12] 孙广开, 曲道明, 闫光, 等. 软体气动驱动器弯曲变形光纤传感与形状重构 [J]. 光学精密工程,2019,27(05):1052-1059.

附录 A　问题一源代码

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimSun'] # 设置中文字体为宋体
plt.rcParams['axes.unicode_minus'] = False # 解决负数坐标轴显示问题

# 设置 Seaborn 样式
sns.set(font='SimSun', font_scale=1.2) # 设置 Seaborn 中文字体为宋体，并缩放字体大小

# 定义常数
c = 4200

# 光纤传感器数据
wavelengths = {
    "FBG1": {"initial1": 1529, "test1": 1529.808, "initial2": 1540, "test2": 1541.095},
    "FBG2": {"initial1": 1529, "test1": 1529.807, "initial2": 1540, "test2": 1541.092},
    "FBG3": {"initial1": 1529, "test1": 1529.813, "initial2": 1540, "test2": 1541.090},
    "FBG4": {"initial1": 1529, "test1": 1529.812, "initial2": 1540, "test2": 1541.093},
    "FBG5": {"initial1": 1529, "test1": 1529.814, "initial2": 1540, "test2": 1541.094},
    "FBG6": {"initial1": 1529, "test1": 1529.809, "initial2": 1540, "test2": 1541.091}
}

# 计算曲率的函数
def calculate_curvature(initial, test):
    lambda_0 = initial
    lambda_ = test
    return c * (lambda_ - lambda_0) / lambda_0

# 输出每个传感器的曲率
curvatures = {}
for sensor, data in wavelengths.items():
    curvatures[sensor] = {
        "curvature1": calculate_curvature(data["initial1"], data["test1"]),
        "curvature2": calculate_curvature(data["initial2"], data["test2"])
    }
# 打印曲率结果
for sensor, curvature in curvatures.items():
    print(f"{sensor} - 测试1曲率: {curvature['curvature1']:.6f}, 测试2曲率:
        {curvature['curvature2']:.6f}")
# 准备绘图数据
sensors = list(curvatures.keys())
curvature1 = [curvatures[s]["curvature1"] for s in sensors]
```

```
curvature2 = [curvatures[s]["curvature2"] for s in sensors]

# 绘制图表
plt.figure(figsize=(12, 6))
sns.lineplot(x=sensors, y=curvature1, label='测试1曲率', marker='o', markersize=8, linewidth=2,
    color='royalblue')
sns.lineplot(x=sensors, y=curvature2, label='测试2曲率', marker='s', markersize=8, linewidth=2,
    color='crimson', linestyle='--')
plt.title('各传感点的曲率变化')
plt.xlabel('传感器')
plt.ylabel('曲率 (nm)')
plt.legend()
plt.tight_layout()
plt.show()
from scipy.interpolate import interp1d

# 光纤传感器位置 (以米为单位)
positions = np.array([0, 0.6, 1.2, 1.8, 2.4, 3.0])

# 提取曲率数据
curvature_test1 = np.array([curvatures[f"FBG{i+1}"]["curvature1"] for i in range(6)])
curvature_test2 = np.array([curvatures[f"FBG{i+1}"]["curvature2"] for i in range(6)])

# 创建插值函数
interpolate_curvature1 = interp1d(positions, curvature_test1, kind='linear',
    fill_value="extrapolate")
interpolate_curvature2 = interp1d(positions, curvature_test2, kind='linear',
    fill_value="extrapolate")
# 指定的x坐标点
x_coords = np.array([0.3, 0.4, 0.5, 0.6, 0.7])
# 绘制图表
plt.figure(figsize=(10, 5))
plt.scatter(positions, curvature_test1, color='blue', label='实际曲率 测试1')
plt.scatter(positions, curvature_test2, color='red', label='实际曲率 测试2')
plt.plot(np.linspace(0, 3, 300), interpolate_curvature1(np.linspace(0, 3, 300)), 'b--',
    label='插值曲线 测试1')
plt.plot(np.linspace(0, 3, 300), interpolate_curvature2(np.linspace(0, 3, 300)), 'r--',
    label='插值曲线 测试2')
plt.title('光纤传感器曲率插值')
plt.xlabel('位置 (米)')
plt.ylabel('曲率')
plt.grid(True)
plt.legend()
plt.show()
# 绘制线性插值后的函数图像
plt.figure(figsize=(10, 5))
plt.scatter(positions, curvature_test1, color='blue', label='实际曲率 测试1')
```

```
plt.scatter(positions, curvature_test2, color='red', label='实际曲率 测试2')
plt.plot(np.linspace(0, 3, 300), interpolate_curvature1(np.linspace(0, 3, 300)), 'b--',
    label='线性插值函数 测试1')
plt.plot(np.linspace(0, 3, 300), interpolate_curvature2(np.linspace(0, 3, 300)), 'r--',
    label='线性插值函数 测试2')
plt.title('线性插值后的函数图像')
plt.xlabel('位置 (米)')
plt.ylabel('曲率')
plt.grid(True)
plt.legend()
plt.show()
# 计算这些点的曲率
curvatures_at_x_test1 = interpolate_curvature1(x_coords)
curvatures_at_x_test2 = interpolate_curvature2(x_coords)

# 打印计算结果
print("测试1的曲率:")
for x, k in zip(x_coords, curvatures_at_x_test1):
    print(f"横坐标{x}米处的曲率: {k:.6f}")

print("\n测试2的曲率:")
for x, k in zip(x_coords, curvatures_at_x_test2):
    print(f"横坐标{x}米处的曲率: {k:.6f}")
import numpy as np
import plotly.graph_objects as go

# 初始波长和变化后的波长数据
lambda_0_test1 = 1529
lambda_0_test2 = 1540
lambdas_test1 = np.array([1529.808, 1529.807, 1529.813, 1529.812, 1529.814, 1529.809])
lambdas_test2 = np.array([1541.095, 1541.092, 1541.090, 1541.093, 1541.094, 1541.091])

# 参数 c 的范围
c_values = np.linspace(3900, 4500, num=13)
kappa_values_test1 = []
kappa_values_test2 = []

# 计算两组测试数据的曲率变化
for c in c_values:
    kappas_test1 = c * (lambdas_test1 - lambda_0_test1) / lambda_0_test1
    kappas_test2 = c * (lambdas_test2 - lambda_0_test2) / lambda_0_test2
    kappa_values_test1.append(np.mean(kappas_test1))
    kappa_values_test2.append(np.mean(kappas_test2))

# 创建图表
fig = go.Figure(data=[
    go.Bar(name='测试 1', x=c_values, y=kappa_values_test1, marker_color='royalblue'),
```

```
    go.Bar(name='测试 2', x=c_values, y=kappa_values_test2, marker_color='firebrick')
])

# 更新布局
fig.update_layout(
    title='',
    xaxis_title='参数c的灵敏度检验',
    yaxis_title='平均曲率',
    legend_title='测试',
    barmode='group',
    template='plotly_white'
)

fig.show()
import numpy as np
import plotly.graph_objects as go

# 初始波长和变化后的波长数据
lambda_0_test1 = 1529
lambda_0_test2 = 1540
lambdas_test1 = np.array([1529.808, 1529.807, 1529.813, 1529.812, 1529.814, 1529.809])
lambdas_test2 = np.array([1541.095, 1541.092, 1541.090, 1541.093, 1541.094, 1541.091])

# 参数 c 的范围
c_values = np.linspace(3900, 4500, num=13)
kappa_values_test1 = []
kappa_values_test2 = []

# 计算两组测试数据的曲率变化
for c in c_values:
    kappas_test1 = c * (lambdas_test1 - lambda_0_test1) / lambda_0_test1
    kappas_test2 = c * (lambdas_test2 - lambda_0_test2) / lambda_0_test2
    kappa_values_test1.append(np.mean(kappas_test1))
    kappa_values_test2.append(np.mean(kappas_test2))

kappa_matrix = np.array([kappa_values_test1, kappa_values_test2])

# 创建热点图
fig = go.Figure(data=go.Heatmap(
    z=kappa_matrix,
    x=c_values,
    y=['测试 1', '测试 2'],
    colorscale='YlGnBu'
))

# 更新布局
fig.update_layout(
```

```
    title='参数c的灵敏度检验',
    xaxis_title='参数 c',
    yaxis_title='测试',
    template='plotly_white'
)

fig.show()
```

附录 B　问题二源代码

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# 使用三次样条插值得到的曲率函数
def curvature_function(s, cubic_spline_curvature):
    return cubic_spline_curvature(s)

# 求解曲线的方程
def curve_equations(s, y, cubic_spline_curvature):
    theta = y[2]
    k = curvature_function(s, cubic_spline_curvature)
    dxds = np.cos(theta)
    dyds = np.sin(theta)
    dthetads = k
    return [dxds, dyds, dthetads]

# 初始条件
initial_conditions = [0, 0, np.deg2rad(45)]

# 定义曲线的弧长范围
s_span = (0, 3)
s_values = np.linspace(s_span[0], s_span[1], 300)

# 解方程
sol = solve_ivp(curve_equations, s_span, initial_conditions, args=(cubic_spline_curvature1,),
    t_eval=s_values)

# 绘制结果
plt.figure(figsize=(8, 6))
plt.plot(sol.y[0], sol.y[1], color='b', linestyle='-', linewidth=2, label='重构曲线') #
plt.title('光纤传感器重构曲线', fontsize=16)
plt.xlabel('X', fontsize=14)
plt.ylabel('Y', fontsize=14)
```

```
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7) #
plt.axis('equal')
plt.tight_layout()
plt.show()
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# 使用三次样条插值得到的曲率函数
def curvature_function(s, cubic_spline_curvature):
    return cubic_spline_curvature(s)

# 求解曲线的方程
def curve_equations(s, y, cubic_spline_curvature):
    theta = y[2]
    k = curvature_function(s, cubic_spline_curvature)
    dxds = np.cos(theta)
    dyds = np.sin(theta)
    dthetads = k
    return [dxds, dyds, dthetads]

# 初始条件
initial_conditions = [0, 0, np.deg2rad(45)]

# 定义曲线的弧长范围
s_span = (0, 3)
s_values = np.linspace(s_span[0], s_span[1], 300)

# 解方程
sol = solve_ivp(curve_equations, s_span, initial_conditions, args=(cubic_spline_curvature2,),
    t_eval=s_values)

# 绘制结果
plt.figure(figsize=(8, 6))
plt.plot(sol.y[0], sol.y[1], color='b', linestyle='-', linewidth=2, label='重构曲线')
plt.title('光纤传感器重构曲线', fontsize=16)
plt.xlabel('X', fontsize=14)
plt.ylabel('Y', fontsize=14)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.axis('equal')
plt.tight_layout()
plt.show()
import sympy as sp
import numpy as np
from scipy.integrate import quad
```

```
from scipy.optimize import fsolve

# 定义符号和函数
x = sp.symbols('x')
y = x**3 + x
y_prime = sp.diff(y, x)
y_double_prime = sp.diff(y_prime, x)
curvature = sp.Abs(y_double_prime) / (1 + y_prime**2)**(3/2)

# 弧长积分函数
arc_length_integrand = sp.lambdify(x, sp.sqrt(1 + y_prime**2))

# 计算总弧长
total_arc_length, _ = quad(arc_length_integrand, 0, 1)

# 定义采样点数
num_samples = 30
arc_length_per_sample = total_arc_length / num_samples

# 寻找等弧长的x值
def find_next_sample_x(start_x, target_length):
    def length_up_to_x(x):
        return quad(arc_length_integrand, start_x, x)[0] - target_length
    next_x = fsolve(length_up_to_x, start_x + arc_length_per_sample / 10)[0]
    return next_x

# 计算所有采样点
sample_x_values = [0]
for _ in range(num_samples - 1):
    next_x = find_next_sample_x(sample_x_values[-1], arc_length_per_sample)
    sample_x_values.append(next_x)

# 计算曲率
curvature_function = sp.lambdify(x, curvature)
sample_curvatures = [curvature_function(xi) for xi in sample_x_values]

sample_x_values, sample_curvatures
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# 使用三次样条插值得到的曲率函数
def curvature_function(s, cubic_spline_curvature):
    return cubic_spline_curvature(s)

# 求解曲线的方程
def curve_equations(s, y, cubic_spline_curvature):
```

```
    theta = y[2]
    k = curvature_function(s, cubic_spline_curvature)
    dxds = np.cos(theta)
    dyds = np.sin(theta)
    dthetads = k
    return [dxds, dyds, dthetads]

# 初始条件
initial_conditions = [0, 0, np.deg2rad(45)]

# 定义曲线的弧长范围
s_span = (0, 3)
s_values = np.linspace(s_span[0], s_span[1], 300)

# 解方程
sol = solve_ivp(curve_equations, s_span, initial_conditions, args=(cubic_spline_curvature2,),
    t_eval=s_values)

# 绘制结果
plt.figure(figsize=(8, 6))
plt.plot(sol.y[0], sol.y[1], color='b', linestyle='-', linewidth=2, label='重构曲线')
plt.title('光纤传感器重构曲线', fontsize=16)
plt.xlabel('X', fontsize=14)
plt.ylabel('Y', fontsize=14)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.axis('equal')
plt.tight_layout()
plt.show()
```

附录 C　问题三源代码

```
import sympy as sp
import numpy as np
from scipy.integrate import quad
from scipy.optimize import fsolve

# 定义符号和函数
x = sp.symbols('x')
y = x**3 + x
y_prime = sp.diff(y, x)
y_double_prime = sp.diff(y_prime, x)
curvature = sp.Abs(y_double_prime) / (1 + y_prime**2)**(3/2)

# 弧长积分函数
```

```
arc_length_integrand = sp.lambdify(x, sp.sqrt(1 + y_prime**2))

# 计算总弧长
total_arc_length, _ = quad(arc_length_integrand, 0, 1)

# 定义采样点数
num_samples = 30
arc_length_per_sample = total_arc_length / num_samples

# 寻找等弧长的x值
def find_next_sample_x(start_x, target_length):
    def length_up_to_x(x):
        return quad(arc_length_integrand, start_x, x)[0] - target_length
    next_x = fsolve(length_up_to_x, start_x + arc_length_per_sample / 10)[0]
    return next_x

# 计算所有采样点
sample_x_values = [0]
for _ in range(num_samples - 1):
    next_x = find_next_sample_x(sample_x_values[-1], arc_length_per_sample)
    sample_x_values.append(next_x)

# 计算曲率
curvature_function = sp.lambdify(x, curvature)
sample_curvatures = [curvature_function(xi) for xi in sample_x_values]

sample_x_values, sample_curvatures

import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimSun'] # 指定英文字体为Times New Roman
plt.rcParams['axes.unicode_minus'] = False # 解决负数坐标轴显示问题
plt.rcParams['font.size'] = 15          # 指定字号
x_original = sample_x_values
y_original = [xi**3 + xi for xi in sample_x_values]

# RK4积分步骤
def rk4_step(x, y, theta, kappa, ds):
    # 定义增量函数
    def delta(x, y, theta, kappa):
        return np.cos(theta) * ds, np.sin(theta) * ds, kappa * ds

    # 计算四个斜率
    k1x, k1y, k1t = delta(x, y, theta, kappa)
    k2x, k2y, k2t = delta(x + 0.5 * k1x, y + 0.5 * k1y, theta + 0.5 * k1t, kappa)
    k3x, k3y, k3t = delta(x + 0.5 * k2x, y + 0.5 * k2y, theta + 0.5 * k2t, kappa)
    k4x, k4y, k4t = delta(x + k3x, y + k3y, theta + k3t, kappa)
```

```
    # 更新x, y, theta
    x += (k1x + 2 * k2x + 2 * k3x + k4x) / 6
    y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6
    theta += (k1t + 2 * k2t + 2 * k3t + k4t) / 6
    return x, y, theta

# 初始条件
x_reconstructed = [0]
y_reconstructed = [0]
theta = float(sp.atan(y_prime.subs(x, 0))) # 初始切线方向

# 重构曲线使用RK4
for i in range(1, len(sample_x_values)):
    xi, yi, theta = rk4_step(x_reconstructed[-1], y_reconstructed[-1], theta,
        sample_curvatures[i], arc_length_per_sample)
    x_reconstructed.append(xi)
    y_reconstructed.append(yi)

# 绘制图形
plt.figure(figsize=(10, 5))
plt.plot(x_original, y_original, label='原始曲线', color='blue')
plt.scatter(sample_x_values, [xi**3 + xi for xi in sample_x_values], color='green',
    label='采样点')
plt.plot(x_reconstructed, y_reconstructed, label='重建曲线', color='red', linestyle='--')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('原始曲线和重建曲线的比较')
plt.grid(True)
plt.show()

import numpy as np
import matplotlib.pyplot as plt

# 原始曲线数据
x_vals = np.linspace(0, 1, 100)
y_vals = x_vals**3 + x_vals

# 计算曲率
y_prime = 3*x_vals**2 + 1
y_double_prime = 6*x_vals
curvatures = np.abs(y_double_prime) / (1 + y_prime**2)**1.5

# 绘制原始曲线
plt.figure(figsize=(10, 5))
plt.plot(x_vals, y_vals, label='Original Curve $y = x^3 + x$', color='blue')
```

```
plt.title('原始曲线和曲率')
plt.xlabel('X')
plt.ylabel('Y')

# 添加曲率分布图
plt.twinx() # 使用双坐标轴
plt.plot(x_vals, curvatures, label='曲率', color='red', linestyle='--')
plt.ylabel('曲率')
plt.legend(loc='upper left')

plt.grid(True)
plt.show()

import numpy as np
import sympy as sp
from scipy.integrate import quad
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go

# 定义符号和函数
x = sp.symbols('x')
y = x**3 + x
y_prime = sp.diff(y, x)
y_double_prime = sp.diff(y_prime, x)
curvature = sp.Abs(y_double_prime) / (1 + y_prime**2)**(3/2)

# 弧长积分函数
arc_length_integrand = sp.sqrt(1 + y_prime**2)

# 计算总弧长
total_arc_length, _ = quad(lambda x: arc_length_integrand.subs('x', x), 0, 1)

# 定义采样点数
num_samples = 50
arc_length_per_sample = total_arc_length / num_samples

# 寻找等弧长的x值
def find_next_sample_x(start_x, target_length):
    def length_up_to_x(x):
        return quad(lambda t: arc_length_integrand.subs('x', t), start_x, x)[0] - target_length
    next_x = fsolve(length_up_to_x, start_x + arc_length_per_sample / 10)[0]
    return next_x

# 计算曲率的函数
curvature_function = sp.lambdify(x, curvature)
```

```
# RK4积分步骤
def rk4_step(x, y, theta, kappa, ds):
    def delta(x, y, theta, kappa):
        return np.cos(theta) * ds, np.sin(theta) * ds, kappa * ds

    k1x, k1y, k1t = delta(x, y, theta, kappa)
    k2x, k2y, k2t = delta(x + 0.5 * k1x, y + 0.5 * k1y, theta + 0.5 * k1t, kappa)
    k3x, k3y, k3t = delta(x + 0.5 * k2x, y + 0.5 * k2y, theta + 0.5 * k2t, kappa)
    k4x, k4y, k4t = delta(x + k3x, y + k3y, theta + k3t, kappa)

    x += (k1x + 2 * k2x + 2 * k3x + k4x) / 6
    y += (k1y + 2 * k2y + 2 * k3y + k4y) / 6
    theta += (k1t + 2 * k2t + 2 * k3t + k4t) / 6
    return x, y, theta

def reconstruct_curve(num_samples):
    # 重新计算采样点和曲率
    arc_length_per_sample = total_arc_length / num_samples
    sample_x_values = [0]
    for _ in range(num_samples - 1):
        next_x = find_next_sample_x(sample_x_values[-1], arc_length_per_sample)
        sample_x_values.append(next_x)
    sample_curvatures = [curvature_function(xi) for xi in sample_x_values]

    # RK4重构曲线
    x_reconstructed = [0]
    y_reconstructed = [0]
    theta = float(sp.atan(y_prime.subs(x, 0))) # 初始切线方向
    for i in range(1, len(sample_x_values)):
        xi, yi, theta = rk4_step(x_reconstructed[-1], y_reconstructed[-1], theta,
            sample_curvatures[i], arc_length_per_sample)
        x_reconstructed.append(xi)
        y_reconstructed.append(yi)

    # 计算MSE
    x_original = np.linspace(0, 1, 1000)
    y_original = x_original**3 + x_original
    y_interp = np.interp(x_original, x_reconstructed, y_reconstructed)
    mse = np.mean((y_original - y_interp)**2)

    return mse, x_original, y_original, x_reconstructed, y_reconstructed, sample_x_values

# 可视化MSE随采样点数变化
sample_counts = [5, 10, 20, 30, 40, 50]
mse_values = []
```

```
for count in sample_counts:
    mse, _, _, _, _, _ = reconstruct_curve(count)
    mse_values.append(mse)


trace = go.Scatter(x=sample_counts, y=mse_values, mode='markers+lines',
     marker=dict(color='red'))


layout = go.Layout(title='MSE vs Sample Counts',
                   xaxis=dict(title='采样点数量'),
                   yaxis=dict(title='MSE'))

fig = go.Figure(data=[trace], layout=layout)

fig.show()
```