三种基于插值方法的光纤传感器平面曲线重构算法建模

摘要

光纤传感是一种利用光在光纤中传播时的物理特性来检测温度、声音、振动和应变的变化的技术。本文根据光纤传感器在受力后波长的变化及其与结构曲率之间的关系，建立了三种数学模型重构受力后的光纤曲线，该问题的研究可以提高光纤传感器在实时监测系统中的应用效率和准确性，发挥其在工业和医疗领域的应用潜力。

针对问题一，本文首先通过光纤在等间距位置布置的传感器收集的波长数据，利用波长变化与光纤曲率之间的关系，计算出了传感器位置处的曲率。然后采用**线性插值法**、**多项式插值法**和、**三次样条插值法**处理数据，比较选择了最优的插值方法，**三次样条插值**因其在保持数据平滑性和连续性方面的优势而被重点应用。针对特定横坐标 x 处的曲线曲率，由于光纤发生了弯曲，所以本文，**基于曲率积分递推**，得到了弧长 s 和横坐标 x 之间的关系，最终计算得到两次测试中，特定横坐标 x 位置处的曲线曲率。

针对问题二，基于问题一中得到的插值结果，采用了**三种不同的方法**来重构光纤的空间曲线，分别为：**基于 Frenet-Serret 公式**、**基于曲率积分递推**和**基于分布式曲率**的方法。接着分别分析了基于三种建模方法对两次测试建模得到的**曲线特点**。

针对问题三，本文首先将问题二中的三个模型分别应用于由给定的曲线方程采样出的曲率数据，重构出了曲线。接着计算了**拟合系数** R^2 和**均方误差** MSE，比较了三种建模方法的重构效果，发现**基于分布式曲率**的方法最为精确，其拟合系数 R^2 和均方误差 MSE 分别为 **0.996** 和 **0.0013**。然后分别探讨分析了三种模型中，导致重构曲线与原始曲线之间出现误差的可能因素，最终经过**灵敏度分析**，使用不同采样点数对曲线进行重构，发现适当增加采样点可以显著减少重构误差，求得了**重构效果最好**的情况下所对应的**采样点数**。

总结而言，本文对比分析了三种建模方法对曲线重构的应用效果，分析了产生误差的原因，验证了高密度采样对提高重构精度的重要性。这些发现为光纤传感技术的进一步研究提供了理论和实验基础，具有一定的工程应用价值。

目录

一、 问题重述

1.1 问题背景

光纤传感器的基本原理为，当外界环境（温度、受力等）发生变化时，光纤中的光波会受到相应的影响，其强度、相位、频率或偏振态等参数会随之改变。光纤传感器通过捕捉这些参数的变化，并将其转换为电信号或其他形式的输出，从而可以精确地反映出外界环境的变化情况。

光纤传感技术的抗干扰能力强、体积小、灵敏度高、传输距离远，在多个领域展现出了强大的应用潜力。还原光纤形状曲线算法是光纤形状传感技术的重要组成部分，其很大程度上决定了光纤传感技术的准确性。因此，深入研究重构光纤形状曲线的建模方法，对于开拓该技术在各个行业中的应用前景具有重要意义。

1.2 问题要求

根据题目，光纤在等间距 (0.6 米) 的位置上布置好了传感器，在测量时记录了光纤处于水平状态和受外力作用后，各传感器位置处信号的波长。并给出了波长变化与光纤曲线的曲率之间的近似关系，即：$k = \frac{c(\lambda-\lambda_0)}{\lambda_0}$, 其中 λ_0 是水平光纤在初始状态下测量的波长，λ 是光纤在受到外力后测量的波长，c 为常数，题中设为 4200。最终分别测量了两组不同初始状态下受力前后的波长值。

基于上述问题背景与数据，我们需要设计解决以下具体问题：

问题一：

- 根据题目中的波长测量数据和波长与曲线曲率的关系，构建数学模型来估算平面光栅各传感点的曲率。
- 基于初始点坐标、光纤初始切线方向和根据水平光纤建立的坐标系, 建立模型来估算曲线在特定横坐标位置处的曲率。

问题二：

- 根据已知的波长测量数据和问题一中求得的曲率，构建数学模型来分别重构两次测试的平面曲线。
- 重构完成后，分析平面曲线的特点。

问题三：

- 根据给定的平面曲线方程, 按适当的等间距弧长采样来计算采样点的曲率。
- 基于采样点的曲率和问题二中建立的模型，重构光纤曲线，并分析重构曲线与原始曲线之间出现误差的原因。

二、 问题分析

问题一：对于问题一中的第一小问，根据题目给出的波长变化与光纤曲线的曲率之间的近似关系 $k = \frac{c(\lambda-\lambda_0)}{\lambda_0}$，可以直接将数据代入公式计算，得到各个传感点 (FBG1-FBG6) 的曲率。

对于第二小问，根据上述已求出的六个传感点处的曲率，通过插值方法，如线性插值法、三次样条函数插值法，多项式插值，再基于曲率积分递推的方法，得到了弧长 s 与横坐标 x 的关系，最终求得在光纤曲线上特定横坐标的曲率。

问题二：对于问题二，可以根据题目所给的波长测量数据和问题一种插值得到的各点的曲率，分别重构平面曲线。具体而言，可以使用以下三种方法进行曲线重构：

- 基于 Frenet-secert 框架的曲线重构: 由 Frenet-Serret 公式，得到光纤曲线上任意点的切线、法线之间的变化关系，以及曲线的曲率，对光纤曲线进行重构。
- 基于曲率积分递推的曲线重构：对于两个相邻的插值点之间的曲线段，首先将其等分为多个小的圆弧段。然后，利用问题一中插值得到的曲率信息，通过积分公式求得每个小圆弧段的倾角表达式。接着利用这些倾角信息，结合几何关系，推导出从当前插值点到下一个插值点的坐标递推公式。逐点应用这个递推公式，就可以重构出整条光纤曲线。
- 基于分布式曲率的曲线积分：根据插值结果，对每一段弧长构建一个运动坐标系，并计算每个点的圆弧曲率和对应圆心角，得到每个点在运动坐标系中的坐标。接着，通过旋转这些坐标到原始坐标系，完成曲线的重构。

最终结合曲率的数学性质等，对重构曲线的特点进行分析。

问题三：对于问题三，可根据题中的曲线表达式，以适当的等间距弧长采样，计算采样点的曲率。再分别根据问题二中构建出的三种模型，重构出平面曲线。接着通过统计量如均方根误差等分析三种方法生成重构曲线的优劣，以及与原始曲线出现误差的原因。由于第三问中采样弧长的间距不确定，还需要分析不同间距弧长采样下的重构误差，进行灵敏度分析，得到采样弧长与重构误差之间的关系，从而得到最好的重构效果。

文章的具体思路图如图1

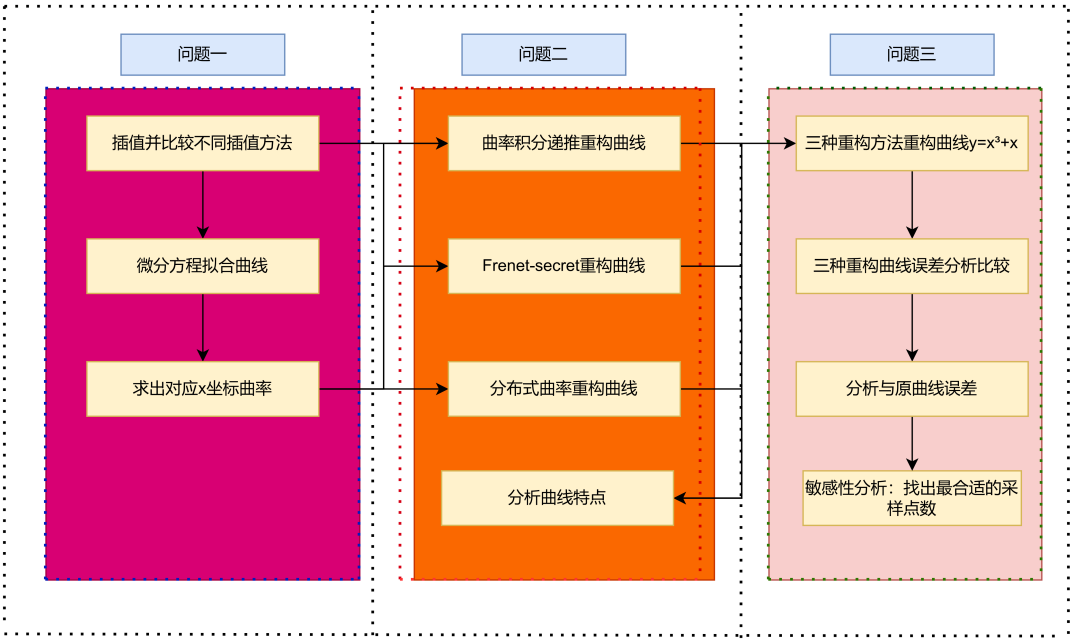

图 **1**　基于曲率积分示意图

三、 模型假设

1. 假设传感器测量的波长精确没有误差。
2. 假设在光纤在形变过程中总长度不变。
3. 假设光纤曲率的变化在测量区间内连续，允许用平滑的数学函数来表示曲率分布。
4. 假设波长的变化和曲率之间的关系严格线性
5. 假设光纤在形变时曲率没有正负突变。

四、 符号说明

符号	意义	单位
λ_0	水平光纤在初始状态下测量的波长	m
λ	光纤在受到外力后测量的波长	m
k	曲线曲率	m^{-1}
FG_i	第 i 个传感器	
$\boldsymbol{T}$	单位切向量	
$\boldsymbol{N}$	指向弯曲中心的单位法向量	
$\boldsymbol{B}$	单位副法向量	
φ	切向量与 x 轴正方向的夹角	°
Δs	相邻两插值点间的弧长	m
ds_i	相邻两插值点间的弦长	m
θ_i	相邻两插值点间的弧所对应的圆心角	°

注：各个参量的具体意义将在后文模型建立和模型求解中进一步明确。

五、 问题一模型的建立与求解

5.1 模型的建立

针对问题一，首先根据题目所给的波长变化与光纤曲线的曲率之间的近似关系，求得各个传感器处的曲线曲率。进一步根据各传感器处的曲率，通过线性插值法、三次样条函数法、多项式函数法三种方法进行插值，求得特定横坐标处的曲线曲率。

5.1.1 不同插值方法

1. 线性插值法：

线性插值是指插值函数为一次多项式的插值方式。根据题目中待求点的左右邻近两个传感器处的曲率来进行估计，根据其到这两个点的距离来分配它们的比重，插值得到待求点的曲线曲率。

2. 多项式函数法：

多项式函数法是指构造一个多项式函数来逼近一组给定的数据点。假设传感器位置和该点的曲率存在某种多项式关系，根据已求出的六个传感点数据与其对应的曲率，找到一个多项式，使该多项式适用于所有的位置。将待求点的横坐标代入插值多项式，计算得到对应的曲率值。

3. 三次样条函数法：

三次样条函数法是一种通过构造分段三次多项式来逼近一组数据点的方法。将相邻传感器作为一个分段区间。在每个分段区间内，三次样条函数写为：$S_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$ 其中，i 表示不同的分段区间，a_i, b_i, c_i, d_i 是每个分段区间内三次多项式的系数。三次样条函数满足以下条件：

- 函数值连续：$S_i(x_i) = y_i$，即在每个分段点 x_i 处，样条函数的值 $S(x_i)$ 等于该点处的曲率测量值 y_i。
- 一阶导数连续：$S_i'(x_i) = S_{i+1}'(x_i)$，即在相邻分段点的交界处，样条函数的一阶导数（斜率）相等。
- 二阶导数连续：$S_i''(x_i) = S_{i+1}''(x_i)$，即在相邻分段点的交界处，样条函数的二阶导数（曲率）也相等。

根据已知数据解这个线性方程组，我们可以得到每个分段区间 (即相邻传感器) 三次多项式的系数 a_i,b_i, c_i, d_i。基于这些系数，即可在任意点 x 上计算 $S(x)$ 的值，即为该点的曲率。

5.1.2 插值方法比较

综合比较三种插值方法，线性插值方法具有实现简单、实时性高的优点，但它在分段点处仅能保证连续性而无法满足导数连续性，因此插值精度相对较低。这种方法通常适用于对插值精度要求不那么严格，但对实时性能有较高要求的应用场景，因而不太适用于此题。多项式插值灵活且精度较高，但可能出现龙格现象，需要谨慎选择多项式的阶数。相比之下，三次样条插值在保持曲线光滑性的同时具有较高的精度，可为后续重构平面曲线提供精度更高的数据，提高平面曲线的重构效果。

5.1.3 计算弧长和其对应的横坐标 x 值之间的关系

由于曲线受力后发生了弯曲，而题中设定的坐标系是以初始未弯曲的水平光纤方向为 x 轴，所以基于弧长进行插值，得到弧长与曲率 k 之间的关系后，需要先将插值点对应的 x 坐标计算出来，才能得到题中特定横坐标下的曲率 k。本文基于曲率积分递推的方法，得到了弧长 s 与横坐标 x 的关系。

首先由三次样条插值得到的数据，得到连续的曲率函数，根据曲率与曲线的倾角关系进行积分，进而推导出光纤曲线的倾角变化。通过对倾角的积分计算，进一步得到曲线的位置变化。接着，通过对每个小段曲率进行积分，计算出每段的倾角和位置。最后，将这些位置逐点连接起来，即得到曲线。具体而言，包括以下步骤：

- Step1: 求曲率的函数 k(s)

假设插值得到的两个相邻点的曲率 k 和弧长 s 是线性关系 [1]，即

$$k = m \cdot s + n \tag{1}$$

将两个相邻离散点的曲率 k_i, k_{i+1} 代入式(1) 可以计算出未知系数 m 和 n

$$\begin{cases} k_{i+1} = m \cdot s_{i+1} + n \\ k_i = m \cdot s_i + n \end{cases} \tag{2}$$

其中 $s_{i+1} - s_i$ 为相邻两插值点之间的弧长。解的 m 和 n 的值如式(25)

$$\begin{cases} m = \frac{(k_{i+1} - k_i)}{(s_{i+1} - s_i)} \\ n = (k_i \cdot s_{i+1} - k_{i+1} \cdot s_i) / (s_{i+1} - s_i) \end{cases} \tag{3}$$

- Step2: 解切向量与 x 轴正方向的夹角 φ

由曲线的微分几何可知，曲线的曲率 k(s) 的表达式为：

$$k(s) = \frac{\mathrm{d}\varphi}{\mathrm{d}s} \tag{4}$$

其中 φ 为切向量与 x 轴正方向的夹角。

所以切向量与 x 轴正方向的夹角 φ 可表达为：

$$\varphi(s) = \int k(s)\mathrm{d}s \tag{5}$$

式中 $k(s)$ 已根据问题一中插值得到的数据求得，因而可以进一步求得 $\varphi(s)$

- Step3: 求坐标增量

将相邻两插值点之间的曲线看作一个小圆弧段，则对于第 i 小段，根据曲线几何，可得到以下规律：

$$\theta_i = \frac{\Delta s}{\rho_i} = \Delta s \times k_i \tag{6}$$

$$ds_i = 2 \times \sin\left(\frac{\theta_i}{2}\right) / k_i, \quad (k_i \neq 0) \tag{7}$$

$$ds_i = \Delta s \tag{8}$$

其中 ρ_i, ρ_{i+1} 是相邻两插值点 A_i, A_{i+1} 处的曲率半径。

Δs，ds_i 分别是相邻两插值点间的弧长与弦长。

θ_i 为这段小圆弧段对应的圆心角。详见图2

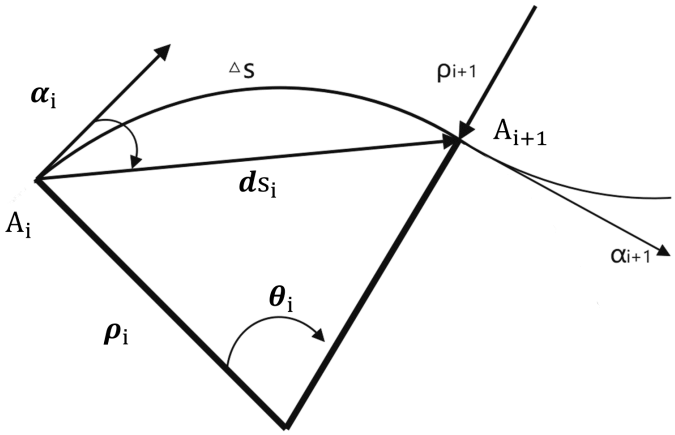

图 **2**　基于曲率积分示意图

相邻两插值点之间的坐标增量分别为：

$$\begin{cases} \Delta x_i = ds_i \times \cos\left(\varphi + \dfrac{\theta_i}{2}\right) \\ \Delta y_i = ds_i \times \sin\left(\varphi + \dfrac{\theta_i}{2}\right) \end{cases} \tag{9}$$

- Step4: 递推曲线位置

从第 i 个点到第 $i+1$ 个点的递推关系即为：

$$\begin{cases} x_{i+1} = x_i + \Delta x_i \\ y_{i+1} = y_i + \Delta y_i \end{cases} \tag{10}$$

- Step5: 转化为题中所要求横坐标 x, 并求得曲率 k

无论曲率是否有正负突变，沿着曲线延伸的方向，曲线在 x 轴方向上的映射长度是相同的，于是本文从原点开始沿着曲线行进的方向，对 x 轴的映射长度进行累计，当达到表中对应的所需的 x 轴坐标大小时，就认为该点所对应的曲率 k 即为所求。

5.2 模型的求解

将题目所给数据带入波长变化与光纤曲线的曲率之间的近似关系 $k = \frac{c(\lambda - \lambda_0)}{\lambda_0}$，计算得到六个传感器处的曲率如表 1。

表 **1**　问题一各传感器处的曲线曲率

传感器	$FG1$	$FG2$	$FG3$	$FG4$	$FG5$	$FG6$
测试 1 曲率 k	2.2195	2.2167	2.2332	2.2305	2.2360	2.2222
测试 2 曲率 k	2.9864	2.9782	2.9727	2.9809	2.9836	2.9755

分别对测试 1 的数据和测试 2 的数据进行三次样条插值。基于曲率积分递推后得到 s 和 x 的关系后，可以计算出表中所给横坐标 x 对应的曲率。结果如表 2。绘制得到曲率与弧长 s 之间的关系如图3和4

表 **2**　问题一给定横坐标处的曲线曲率

横坐标 $x(m)$	0.3	0.4	0.5	0.6	0.7
测试 1 曲率 k	2.2176	2.2274	2.2334	2.2337	2.2360
测试 2 曲率 k	2.9832	2.9816	2.9799	2.9782	2.9765

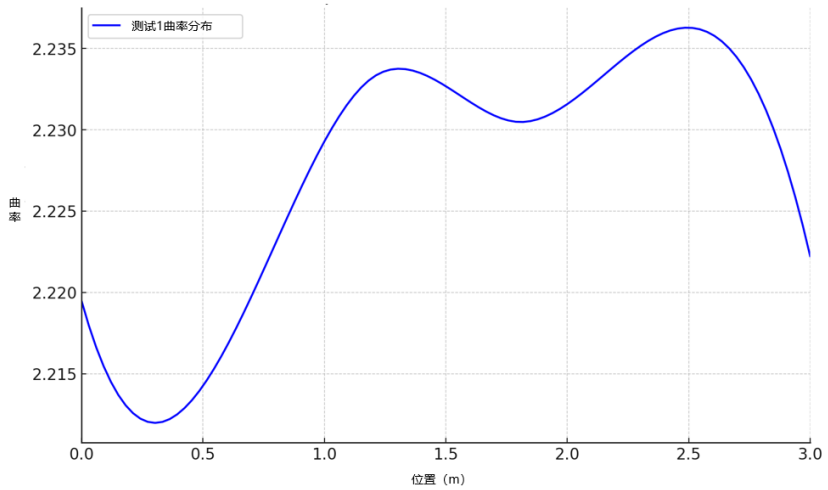

图 **3**　对测试 **1** 数据进行三次样条插值得到的曲率与弧长 s 的关系

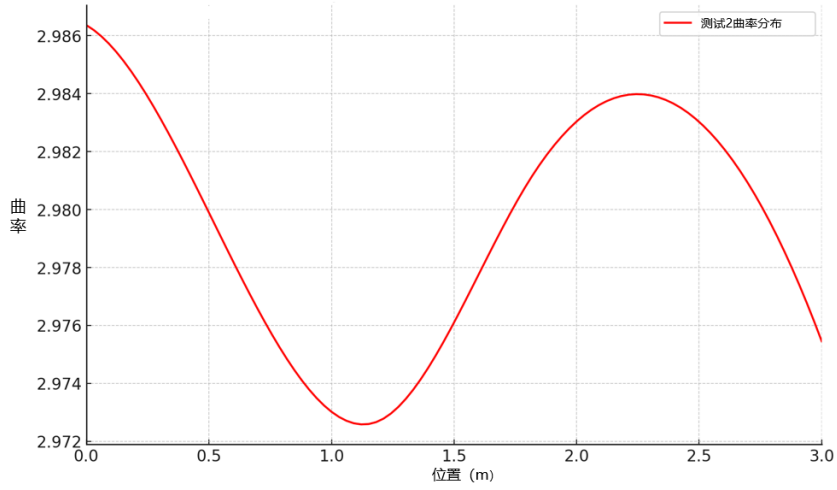

图 **4**　对测试 **2** 数据进行三次样条插值得到的曲率与弧长 s 的关系

根据图3和图4可以看出：

- 由于计算得到六个传感器处的曲率均为正值，所以插值得到的曲率也均为正值

- 由于计算得到六个传感器处的曲率变化较小，曲率最大的变化量 < 0.003, 所以曲率曲线的起伏不大，图中实际为放大后的显示效果。

六、 问题二模型的建立与求解

6.1 模型的建立

对于问题二，首先根据题目所给的波长测量数据和插值得到的各点的曲率 k，分别重构平面曲线，再结合数学原理等分析曲线的特点。本文使用如下三种方法实现曲线重构。

6.1.1 基于 Frenet-secert 框架的曲线重构

在二维空间中，将光纤曲线表示为弧长 s 的自然参数方程：

$$\boldsymbol{r}(s) = x(s)\boldsymbol{i} + y(s)\boldsymbol{j} \tag{11}$$

式中 $\boldsymbol{i}$,$\boldsymbol{j}$ 分别是 x，y 轴轴向单位向量；$x(s)$ 和 $y(s)$ 代表空间曲线的二维坐标；$\boldsymbol{i}$ 和 $\boldsymbol{j}$ 三者互相垂直。

弗莱纳 (Frenet-secert) 框架 [2] 描述了曲线的切向、法向、副法方向之间的关系。基于此，使用曲线沿线各点的瞬时运动方向上的单位切向量 $\boldsymbol{T}$，垂直于切向量且指向弯曲中心的单位法向量 $\boldsymbol{N}$ 和单位副法向量 $\boldsymbol{B}$, 构成弗莱纳坐标系 (Frenet−Serret frame), 由于题目中仅考虑二维空间中的情况，所以可将 Frenet-secert 框架变化为：

$$\begin{cases} \dfrac{\mathrm{d}\boldsymbol{T}}{\mathrm{d}s} = k\boldsymbol{N} \\ \dfrac{\mathrm{d}\boldsymbol{N}}{\mathrm{d}s} = -k\boldsymbol{T} \end{cases} \tag{12}$$

其中 k 为曲线的曲率，在问题一中，已通过插值得到了曲线上各插值点的曲率，单位切向量 $\boldsymbol{T}$ 和单位法向量 $\boldsymbol{N}$ 计算如下：

$$\boldsymbol{T}(s) = \frac{\mathrm{d}\boldsymbol{r}(s)}{\mathrm{d}s} = x'(s)\boldsymbol{i} + y'(s)\boldsymbol{j} \tag{13}$$

$$\boldsymbol{N}(s) = \frac{x''(s)\boldsymbol{i} + y''(s)\boldsymbol{j}}{\sqrt{(x''(s))^2 + (y''(s))^2}} \tag{14}$$

由题所知，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°，可知

单位切向量 $\boldsymbol{T}$ 和单位法向量 $\boldsymbol{N}$ 的初值为

$$\begin{cases} \boldsymbol{T}(0) = (\dfrac{\sqrt{2}}{2}, \dfrac{\sqrt{2}}{2}) \\ \boldsymbol{N}(0) = (-\dfrac{\sqrt{2}}{2}, \dfrac{\sqrt{2}}{2}) \end{cases} \tag{15}$$

连立式(12)，(13)，(14)，(15)即可重构出曲线。曲线方程为

$$r(s) = \int T(s)\mathrm{ds} \tag{16}$$

6.1.2 基于曲率积分递推的曲线重构

此方法已在问题一的模型分析中详细解释，详见5.1.3，此处不再赘述。

6.1.3 基于分布式曲率的曲线重构

首先设定采样点之间的等弧长和初始切线方向。每个采样点处根据其曲率定义动坐标系，并通过坐标旋转将这些局部坐标转换为全局坐标。最后，通过三次样条插值，连接所有采样点，形成一条平滑连续的曲线。具体建模过程如下：

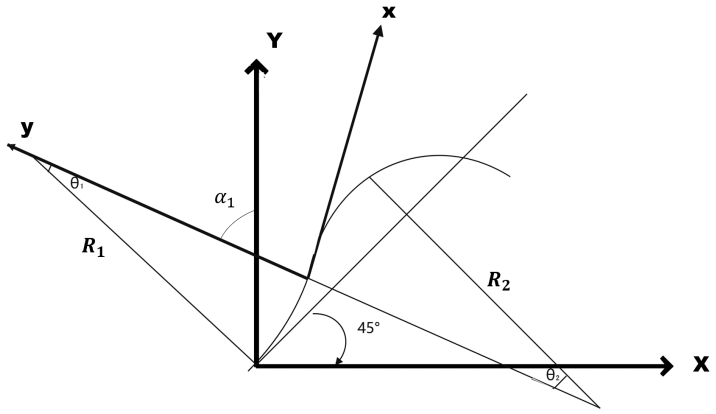

图 **5**　基于分布式曲率的曲线重构示意图

- Step1: 建立运动坐标系
  光纤曲线上相邻两个插值点之间的弧长均为 $\triangle s$。在每个插值点处, 以采样点为坐标原点，以该点的切线方向为 x 轴，将 x 轴逆时针旋转 90° 为 y 轴, 建立一个运动坐标系。
- Step2: 求任一点在运动坐标系中的坐标
  在原点 O_0 建立的坐标系为绝对坐标系，则根据曲线几何关系，任意一点 Oi 在运动坐标系中的坐标为：

$$(x_i, y_i) = (R_i sin\theta_i, R_i - R_i cos\theta_i) \tag{17}$$

$$\theta_i = \triangle s \cdot k_i \tag{18}$$

其中 $\triangle s$ 为每一段圆弧的弧长，且该弧长相等，k_i 为第 i 段的圆弧对应的曲率，θ_i 为第 i 段圆弧对应的圆心角,R_i 为第 i 段圆弧的曲率半径。

- Step3: 旋转坐标求得下一点的坐标

  由题目可知，初始点切线方向与 x 轴夹角为 45°，所以初始点后下一个插值点的坐标为：

$$(X_1, Y_1) = (\frac{\sqrt{2}}{2}R_1 - R_1 \cdot cos(\theta_1 + 45°), R_1 \cdot sin(\theta_1 + 45°) - \frac{\sqrt{2}}{2}R_1) \tag{19}$$

  其中 (X_1, Y_1) 为初始点后下一个插值点在绝对坐标系中的坐标

  将 O_i 处运动坐标系顺时针旋转 α_i 角度即可与绝对坐标系平行, 由几何关系可知：

$$\alpha = 45° + \theta_1 + \sum_{2}^{i} \theta_i \tag{20}$$

  通过坐标旋转变换即可得到下一点 O_{i+1} 在绝对坐标系中的坐标为:

$$(X_{i+1}, Y_{i+1}) = (X_i + x_{i+1} \cdot cos\alpha_i - y_{i+1} \cdot sin\alpha_i, Y_i + x_{i+1} \cdot sin\alpha_i + y_{i+1} \cdot cos\alpha_i) \tag{21}$$

  最终根据式(21)给出的关系，经过三次样条插值，即可重构出曲线。

6.2 模型的求解与曲线分析

6.2.1 基于 Frenet-secert 框架的曲线重构

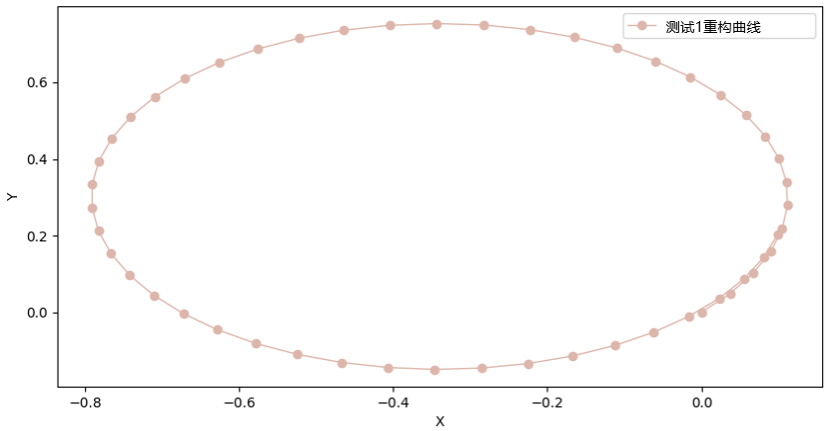

图 6　基于 **Frenet-secert** 框架对测试 **1** 的光纤曲线重构结果

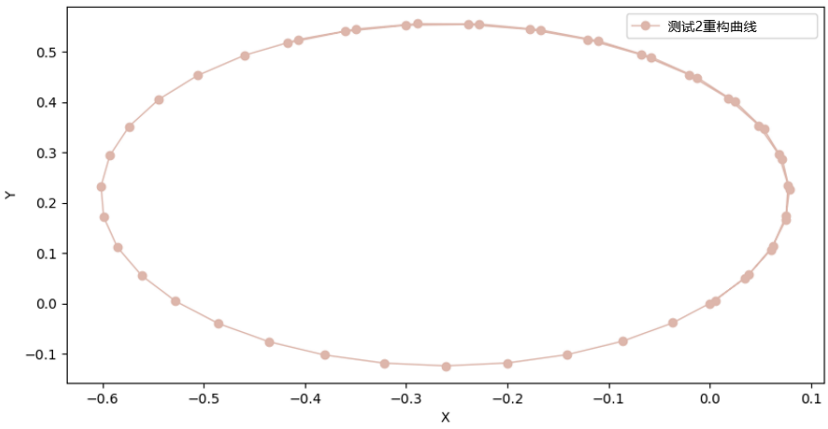

图 **7**　基于 **Frenet-secert** 框架对测试 **2** 的光纤曲线重构结果

基于 Frenet-secert 框架的曲线重构结果分别如图6和图7所示。详细分析见6.2.4。

6.2.2 基于曲率积分递推的曲线重构

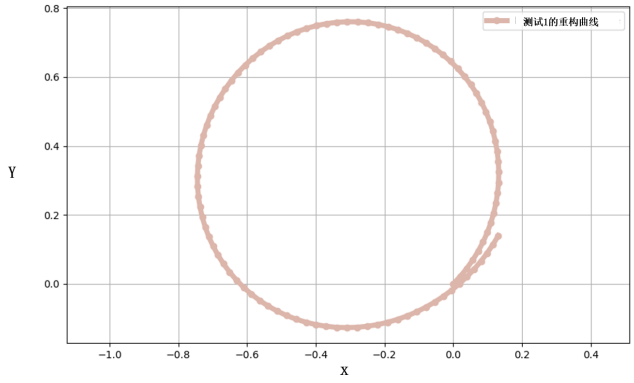

图 **8**　基于曲率积分递推对测试 **1** 的光纤曲线重构结果

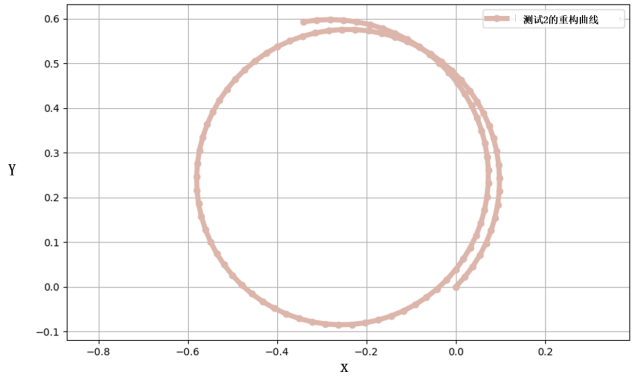

图 **9**　基于曲率积分递推对测试 **2** 的光纤曲线重构结果

基于曲率积分递推的曲线重构结果分别如图8和图9所示。详细分析见6.2.4。

6.2.3 基于分布式曲率的曲线重构

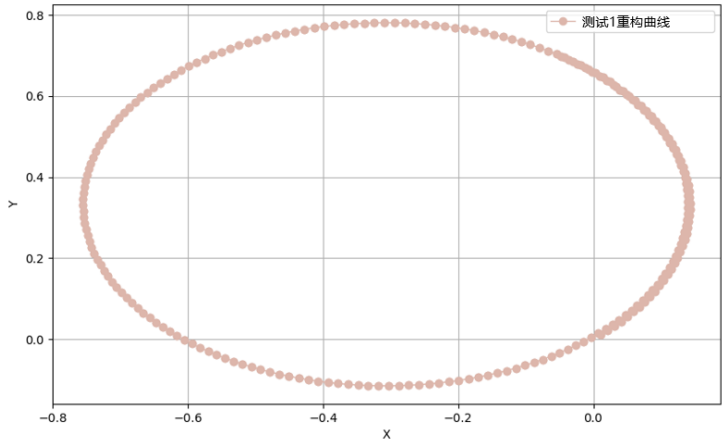

图 **10**　基于分布式曲率对测试 **1** 的光纤曲线重构结果

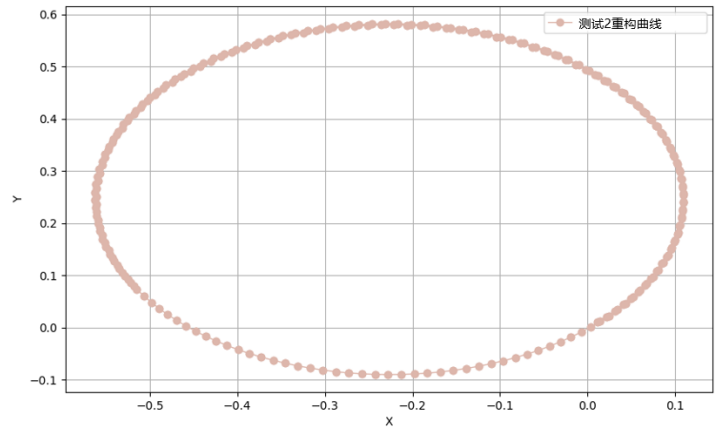

图 **11**　基于分布式曲率对测试 **2** 的光纤曲线重构结果

基于分布式曲率的曲线重构结果分别如图10和图11所示。详细分析见6.2.4。

6.2.4 对所有重构曲线的分析

分别分析用三种方法对测试 1 和测试 2 重建得到的曲线，总结出曲线的特点如下：

1. 三种建模方法的特点及比较：

- 整体上三种建模方法得到的曲线均为逆时针弯曲，未出现相反方向的弯曲。这是由于在问题一中插值得到的曲率均为正值，而根据曲率的数学意义，正曲率意味着曲线在该点处以逆时针方向弯曲，负曲率则表示曲线在该点处以顺时针方向弯曲。所以整体上曲线只往一个方向弯曲。

• 重构曲线光滑连续。这是由于问题一中插值得到的曲率变化较小，且随横坐标变化，曲率变化比较平滑，没有特殊断点或尖点，所以曲线比较平滑。

• 基于 Frenet-Serret 框架的曲线重构，依赖于初始条件和给定的曲率数据来重构曲线。由图像发现，曲线显示出较高的准确度，该方法更适合于曲线形态相对简单且变化规律明显的情况。

• 基于曲率积分递推的曲线重构，通过积分曲率信息来推导曲线的倾角变化，从而逐步构建曲线的路径。因为积分递推依赖于局部曲率的精确估计，受到插值曲率的影响较大。

• 基于分布式曲率的曲线重构考虑了每个测量点处的局部曲率信息，并通过坐标旋转将这些局部信息合成全局曲线形态。曲线更加平滑和连续，充分考虑了局部变化对整体形状的影响。

2. 测试 1 和测试 2 建模得到曲线对比：

• **测试 1 曲线：**曲线呈现出逆时针弯曲，但整体上比较平滑。测试 1 曲线在曲率变化较小时，曲线呈现出较小的弯曲，而在曲率变化较大时，曲线则会产生更明显的弯曲。但由于曲率的变化较小，最大的变化量 <0.03, 所以总体的图像弯曲程度的变化还是较为平缓。

• **测试 2 曲线：**测试 2 的曲线整体上比测试 1 的曲线更加弯曲，表现出相较于测试 1 更明显的弯曲特征。与测试仪相似，曲线同样在曲率变化较大时，呈现出更弯曲的特征。

• 测试 1 和测试 2 的曲线轨迹在整体上都呈现出了一定的弯曲特征，但测试 2 的曲线相较于测试 1 的弯曲程度更大。这可能是由于在测试 2 中施加给光纤的外力更大，所以其曲率变化较于测试 1 更剧烈，最终导致了两次测试弯曲程度的差异。

七、 问题三模型的建立与求解

7.1 模型的建立

在问题三中，首先要根据题中的曲线表达式，以适当的等间距弧长采样，计算采样点的曲率。然后分别基于问题二中构建出的三种模型，重构出平面曲线。接着通过统计量如均方根误差等分析三种方法生成重构曲线的优劣，以及与原始曲线出现误差的原因。由于第三问中采样弧长的间距不确定，还需要分析不同间距弧长采样下的重构误差，得到采样弧长与重构误差之间的关系，从而得到最好的重构效果。

7.2 模型的求解

实验中取 5 个采样点 (为何取 5 个采样点将在下文中进行讨论)，采用三次样条插值和问题二中的三种模型，分别对该曲线进行重构的结果如图**??**

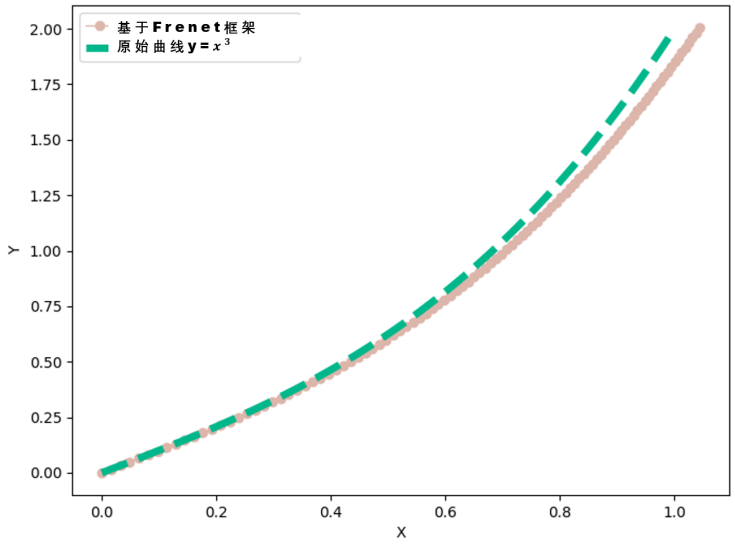

图 **12**　基于 $Frenet$ 框架的曲线重构结果

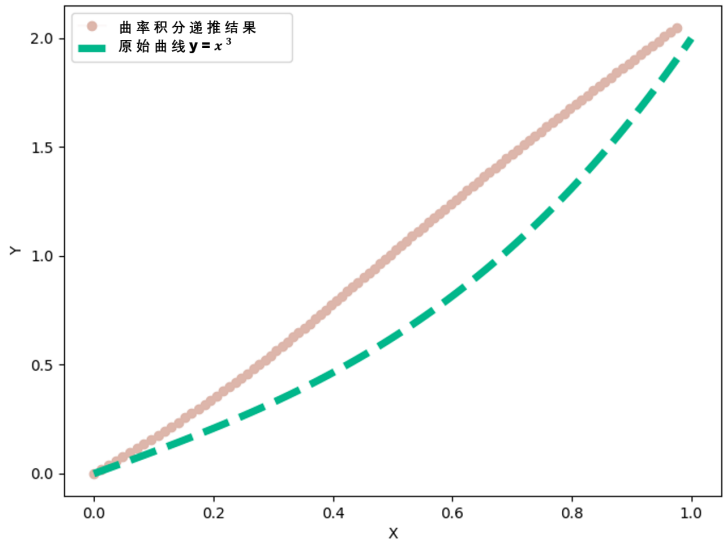

图 **13**　基于曲率积分递增的曲线重构结果

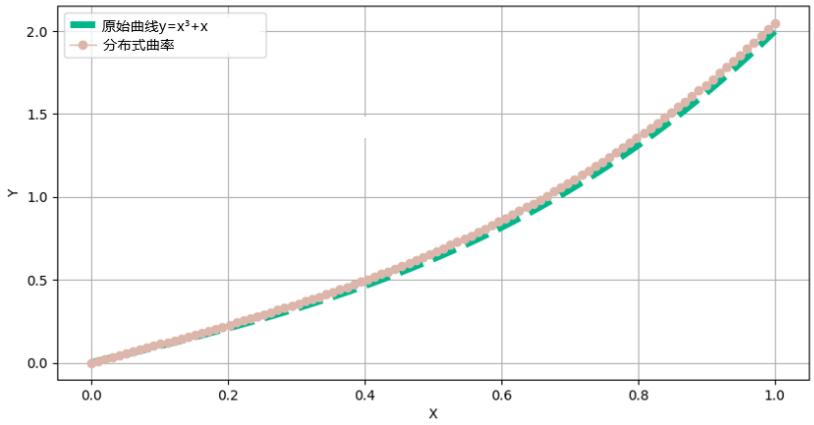

图 **14**　基于分布式曲率的曲线重构结果

定义误差大小为 $|y_{recon} - y_{ori}|$, 其中 y_{recon} 为重构曲线对应的函数值，y_{ori} 为原始曲线对应的函数值，得到三种建模方法的误差大小如图15

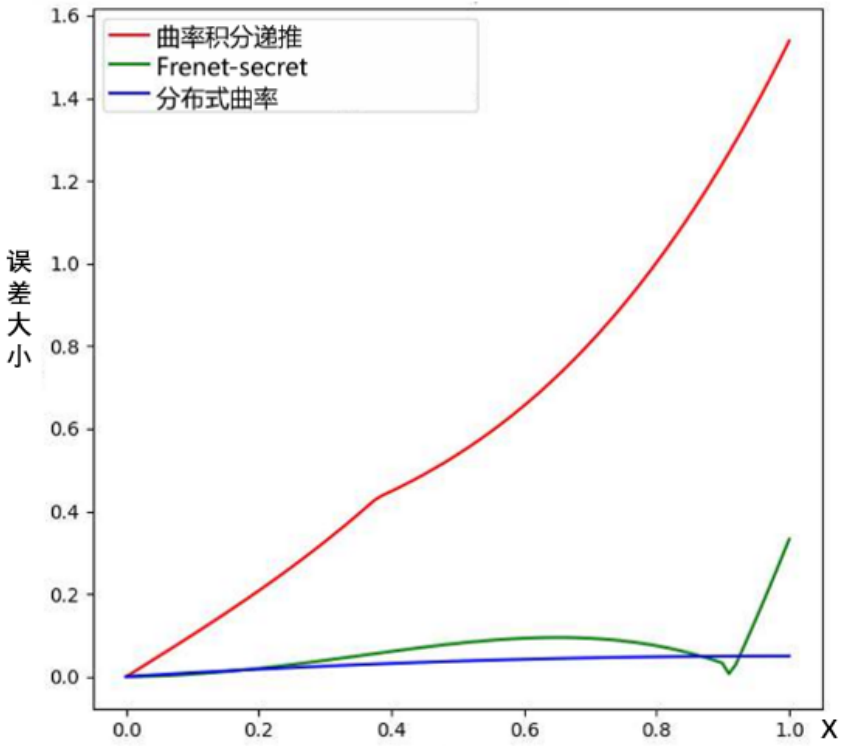

图 **15**　三种建模方法的曲线重构误差大小比较

由图15可以初步判断，基于分布式曲率的曲线重构结果最好。

7.3 误差分析

7.3.1 三种建模方法的评估参数

拟合系数 R^2 均方误差 MSE 的计算公式分别为:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{22}$$

$$SS_{res} = \sum_{i=1}^{n}(y_{recon} - y_{ori})^2 \tag{23}$$

$$SS_{tot} = \sum_{i=1}^{n}(y_{recon} - \bar{y}_{recon})^2 \tag{24}$$

$$MSE = \frac{\sum_{i=1}^{n}(y_{recon} - y_{ori})^2}{n} \tag{25}$$

其中 SS_{res} 为残差平方和，SS_{tot} 为总平方和，y_{recon} 为重构曲线对应的函数值，y_{ori} 为原始曲线对应的函数值,n 为采样点数量。

表 **3**　不同建模方法的评估

建模方法	拟合系数 R^2	均方误差 MSE
基于 Frenet 框架	0.789	0.0020
基于曲率积分递推	0.389	0.19629
基于分布式曲率	0.996	0.00133

- 确定系数 R^2 是通过数据的变化来表征一个拟合的好坏。R^2 的取值范围为 [0 1]，越接近 1，表明模型对数据拟合的也较好。由表4可知，基于分布式曲率的曲线重构方法拟合效果最好。
- 方差 MSE 表示了数据的离散程度。由表4可知，基于 Frenet 框架和基于分布式曲率的方差都较小，拟合效果都较好。

7.3.2 误差来源分析

根据三种建模方法的原理等，将产生误差的原因总结如下：

- 来源于递推产生的累计误差。由于三种建模方法均匀用了递推思想，所以理论上来说误差会因为累计而逐渐增大 [3]。如图15, 基于曲率积分递推和基于分布式曲率的重构方法随着 x 坐标的增加，都呈现出上升趋势。基于 Frenet-secrect 框架的方法，随着 x 坐标的增加，误差也是先上升，末端出现出下降，可能与等弧长采样间隔的大小有关，影响了曲线的拟合效果。
- 由不同采样间隔导致的差距。由于问题三中第一步需要根据不同采样间隔对图形进行采样，再根据采样得到的数据重构曲线，所以采样点的数量很大程度上影响了重

构的效果[4]。本文进行了灵敏度分析，即分析不同采样点个数对重构效果的影响，详细分析过程见7.3.3。

7.3.3 基于采样点数对重构曲线效果的灵敏度分析

取不同数量的采样点，采用基于分布式曲率的曲线重构，计算得到的重构误差，同样为 $|y_{recon} - y_{ori}|$，得到重构误差与采样点个数的关系如图16

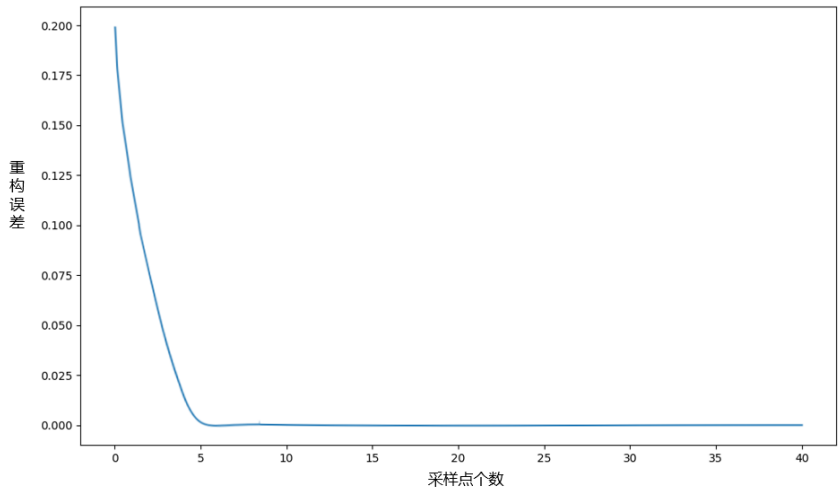

图 **16**　三种建模方法的曲线重构结果误差比较

由图16可知以下结论：

- 随着采样点的增加，重构误差逐渐减少。
- 在采样点数大约为 5 时，重构误差取得最小值。
- 采样点数 > 5 时随着采样点数增加，重构误差的改变很小。

综合考虑计算效率和重构效果，最终取采样点数为 5。

八、 模型评价与展望

8.1 模型的优点

1. 多方法建模的对比与应用。模型使用了多种不同的插值和曲线重构方法，针对每种方法的效果进行了的比较和分析，可以根据具体的应用需求选择最合适的技术路径，使得重构结果更加真实可信。
2. 模型充分讨论了误差的来源，并通过灵敏度分析得到最佳重构效果。考虑到了递推方法和采样间隔等对曲线重构效果的影响，通过不同采样点个数的重构结果，进行灵敏度分析，得到了重构效果最好的曲线对应的采样点的个数。

3. 模型有较好的适用性和灵活性。首先根据插值数据建立了模型，计算出基于插值数据的重构曲线，再将建立的模型应用到了问题三中，得到了较好的效果和精度。证明了模型的适用于有不同的曲率特点的曲线，适用性较好。

8.2 模型的缺点

1. 未考虑传感器的测量误差、光纤受力后长度发生改变等误差对模型的影响，可以通过实际的实验和相关数据，引入误差参数，对模型进行修正，更加符合实际情况。

九、 参考文献

[1] 吴家麒, 杨东英, 沈林勇, 等. 基于曲率数据的曲线拟合方法研究 [J]. 应用科学学报,2003(03):258-262.

[2] 傅程. 基于分布式布里渊光时域分析技术的形状传感研究 [D]. 哈尔滨工业大学,2018.

[3] 尚秋峰, 向方宇, 樊松. 基于轴向应变误差修正的 FBG 形状重构算法 [J]. 半导体光电,2023,44(06):972-980.DOI:10.16818/j.issn1001-5868.2023071802.

[4] 徐州. 基于光频域反射计的分布式光纤三维形状传感技术研究 [D]. 重庆大学,2022.DOI:10.27670/d.cnki.gcqdu.2022.000161.

十、 附录

本文所有代码均使用 python 编写，并将代码源文件存放在了支撑材料中，下表为支撑材料说明表：

10.1 支撑材料说明

表 4　支撑材料说明表

文件名	说明
main.py	论文的全部代码
latex	论文的 latex 源代码

10.2 所有代码

```
import sympy as sp
import numpy as np
from scipy.integrate import quad
from scipy.interpolate import interp1d
import pandas as pd

# 定义变量和函数
x = sp.symbols('x')
f = x**3 + x
f_prime = sp.diff(f, x)
arc_length_expr = sp.sqrt(1 + f_prime**2)
f_double_prime = sp.diff(f_prime, x)

# 计算弧长
arc_length_function = sp.lambdify(x, arc_length_expr)
def arc_length_to_x(x_val):
return quad(arc_length_function, 0, x_val)[0]
arc_length = sp.integrate(arc_length_expr, (x, 0, 1)).evalf()

# 计算等弧长采样点的 x 值
def find_x_for_length(target_length, x_start, x_end, tolerance=1e-6):
left, right = x_start, x_end
while right - left > tolerance:
mid = (left + right) / 2
if arc_length_to_x(mid) < target_length:
left = mid
else:
right = mid
return (left + right) / 2
```

```
sample_points_x = [0]
for i in range(1, 6):
target_length = i * (arc_length / 6)
x_val = find_x_for_length(target_length, sample_points_x[-1], 1)
sample_points_x.append(x_val)
sample_points_x.append(1)

# 计算曲率
curvature_function = sp.lambdify(x, sp.Abs(f_double_prime) / (1 + f_prime**2)**(3/2))
curvatures = [curvature_function(val) for val in sample_points_x]

# 三次样条插值
sample_lengths = [arc_length_to_x(x_val) for x_val in sample_points_x]
interpolate_on_arc = interp1d(sample_lengths, curvatures, kind='cubic')
dense_lengths = np.linspace(0, arc_length, 100)
interpolated_curvatures_on_arc = interpolate_on_arc(dense_lengths)
dense_x_from_length = [find_x_for_length(length, 0, 1) for length in dense_lengths]

# 保存为 Excel 文件
df_revised = pd.DataFrame({
   'Arc Length': dense_lengths,
   'X': dense_x_from_length,
   'Interpolated Curvature on Arc': interpolated_curvatures_on_arc
})
excel_path_revised = 'Interpolated_Curvatures_Arc_Length.xlsx'
df_revised.to_excel(excel_path_revised, index=False)

# 输出路径
print("Excel file saved at:", excel_path_revised)


import numpy as np
import pandas as pd
from scipy.interpolate import interp1d

# 波长数据
data = {
   "FBG": ["FBG1", "FBG2", "FBG3", "FBG4", "FBG5", "FBG6"],
   "Initial State 1": [1529, 1529, 1529, 1529, 1529, 1529],
   "Test 1": [1529.808, 1529.807, 1529.813, 1529.812, 1529.814, 1529.809],
   "Initial State 2": [1540, 1540, 1540, 1540, 1540, 1540],
   "Test 2": [1541.095, 1541.092, 1541.090, 1541.093, 1541.094, 1541.091]
}

# 转换为 DataFrame
df = pd.DataFrame(data)
```

```
    # 常数 c
    c = 4200

    # 计算曲率的函数
    def calculate_curvature(l0, l):
    return c * ((l - l0) / l0)

    # 计算曲率
    df["Curvature Test 1"] = calculate_curvature(df["Initial State 1"], df["Test 1"])
    df["Curvature Test 2"] = calculate_curvature(df["Initial State 2"], df["Test 2"])

    # X坐标（传感器间距为0.6米）
    df["X"] = 0.6 * np.arange(1, len(df) + 1)

    # 三次样条插值
    interp_test1 = interp1d(df["X"], df["Curvature Test 1"], kind='cubic',
        fill_value="extrapolate")
    interp_test2 = interp1d(df["X"], df["Curvature Test 2"], kind='cubic',
        fill_value="extrapolate")

    # 生成更密集的 x 值用于插值显示
    x_new = np.linspace(df["X"].min(), df["X"].max(), 100)
    interpolated_curvature_test1 = interp_test1(x_new)
    interpolated_curvature_test2 = interp_test2(x_new)

    # 创建结果数据框
    results_df = pd.DataFrame({
       "X_new": x_new,
       "Interpolated Curvature Test 1": interpolated_curvature_test1,
       "Interpolated Curvature Test 2": interpolated_curvature_test2
    })

    # 保存为 Excel 文件
    excel_output_path = "Interpolated_Curvatures_FBG.xlsx"
    results_df.to_excel(excel_output_path, index=False)

    print("Excel file saved at:", excel_output_path)

# 递推算法
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load Excel data
data = pd.read_excel('Interpolated_Curvatures_Arc_Length.xlsx')
s = data['Arc Length'].values # 弧长s
```

```
k = data['Interpolated Curvature on Arc'].values # 曲率k
# data = pd.read_excel('Full_Interpolated_Curvatures_Test_1.xlsx')
# s = data['X (meters)'].values # 弧长s
# k = data['Curvature Test 1'].values # 曲率k
delta_s = s[1] - s[0] # 弧长的增量

# Calculate m and n from the curvature data provided
m = (k[1:] - k[:-1]) / delta_s
n = (k[:-1] * s[1:] - k[1:] * s[:-1]) / delta_s

# Initialize the array for phi and calculate the constant c based on initial condition
phi = np.zeros_like(s)
phi[0] = 45 # Initial angle is 0 degrees

# Reconstruct the curve using the recursive formulas provided
x = np.zeros_like(s)
y = np.zeros_like(s)
for i in range(1, len(s)):
phi[i] = phi[i-1] + 0.5 * (m[i-1] * (s[i]**2 - s[i-1]**2) + n[i-1] * (s[i] - s[i-1]))
if k[i-1] != 0: # To avoid division by zero
ds_i = 2 * np.sin((s[i] - s[i-1]) * k[i-1] / 2) / k[i-1]
else:
ds_i = s[i] - s[i-1]
x[i] = x[i-1] + ds_i * np.cos((phi[i] + phi[i-1]) / 2)
y[i] = y[i-1] + ds_i * np.sin((phi[i] + phi[i-1]) / 2)

# Calculate original curve points
x_original = np.linspace(0, 1, 100) # Generate 100 evenly spaced points from 0 to 1
y_original = x_original**3 + x_original # Calculate y values for each x

# Plotting both curves
# Plotting both curves
plt.figure(figsize=(8, 6))
plt.plot(x, y, marker='o', label='Reconstructed Curve', linewidth=0.1, color='#ddb6ab') #
    修改曲线粗细和颜色
plt.plot(x_original, y_original, label='Original Curve y=x³+x', linestyle='--',
    color='#00b78b',linewidth=5) # 修改颜色
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(False)
plt.show()

#
#Frenet框架
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt

# 加载Excel文件中的数据
data = pd.read_excel('Interpolated_Curvatures_Arc_Length.xlsx')
# data = pd.read_excel('Full_Interpolated_Curvatures_Test_1.xlsx')
# 初始化变量
initial_angle = np.deg2rad(45) # 初始角度为45度
positions = data['Arc Length'].values # 从Excel文件获取横坐标
curvatures = data['Interpolated Curvature on Arc'].values # 从Excel文件获取对应的曲率
# positions = data['X (meters)'].values # 从Excel文件获取横坐标
# curvatures = data['Curvature Test 1'].values # 从Excel文件获取对应的曲率
delta_s = positions[1] - positions[0] # 假设等距离

# 初始框架向量（在二维中只有T和N）
T = np.array([np.cos(initial_angle), np.sin(initial_angle)]) # 初始切向量
N = np.array([-np.sin(initial_angle), np.cos(initial_angle)]) # 初始法向量

# 初始化坐标
x = [0]
y = [0]

# 计算曲线点
for i, curvature in enumerate(curvatures):
# 计算T和N的变化（Frenet-Serret公式）
dT = curvature * delta_s * N # 切向量的变化
dN = -curvature * delta_s * T # 法向量的变化
T += dT
N += dN
# 保证T和N为单位向量
T /= np.linalg.norm(T)
N /= np.linalg.norm(N)

# 更新位置
x_new = x[-1] + delta_s * T[0]
y_new = y[-1] + delta_s * T[1]
x.append(x_new)
y.append(y_new)

# 计算原始曲线的点
x_original = np.linspace(0, 1, 100) # 生成从 0 到 1 的 100 个等间距点
y_original = x_original**3 + x_original # 计算每个 x 点对应的 y 值

# 绘制曲线
plt.figure(figsize=(8, 6))
plt.plot(x, y, label='Reconstructed Curve', marker='o', linewidth=1, color='#ddb6ab')
plt.plot(x_original, y_original, label='Original Curve y=x³+x',
    linestyle='--',color='#00b78b',linewidth=5)
```

```
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(False)
plt.show()

#基于分布曲率1
import numpy as np
import pandas as pd
from scipy.interpolate import CubicSpline
import matplotlib.pyplot as plt

# 定义传感器数据
data = {
   "Sensor": ["FBG1", "FBG2", "FBG3", "FBG4", "FBG5", "FBG6"],
   "Initial_1": [1529, 1529, 1529, 1529, 1529, 1529],
   "Test_1": [1529.808, 1529.807, 1529.813, 1529.812, 1529.814, 1529.809],
   "Initial_2": [1540, 1540, 1540, 1540, 1540, 1540],
   "Test_2": [1541.095, 1541.092, 1541.090, 1541.093, 1541.094, 1541.091]
}

# 转换为 DataFrame
df = pd.DataFrame(data)

# 计算曲率常数
c = 4200

# 计算两种测试的曲率
df['Curvature_1'] = c * (df['Test_1'] - df['Initial_1']) / df['Initial_1']
df['Curvature_2'] = c * (df['Test_2'] - df['Initial_2']) / df['Initial_2']

# 位置设置，0.6米间隔
positions = np.linspace(0, 0.6 * (len(df) - 1), len(df))

# 进行三次样条插值
cs_1 = CubicSpline(positions, df['Curvature_1'])
cs_2 = CubicSpline(positions, df['Curvature_2'])

# 新的位置和插值曲率
new_positions = np.linspace(0, 0.6 * (len(df) - 1), 40 * (len(df) - 1) + 1)
new_curvature_1 = cs_1(new_positions)
new_curvature_2 = cs_2(new_positions)

# 角度和坐标计算
delta_s = 0.6 / 40
# 初始化坐标和角度
X_1, Y_1, alpha_1 = [0], [0], [np.deg2rad(45)]
```

```
X_2, Y_2, alpha_2 = [0], [0], [np.deg2rad(45)]

# 计算曲线的坐标点
for i in range(1, len(new_curvature_1)):
theta_1 = delta_s * new_curvature_1[i]
theta_2 = delta_s * new_curvature_2[i]
xi_1 = np.sin(theta_1) / new_curvature_1[i] if new_curvature_1[i] != 0 else 0
yi_1 = (1 / new_curvature_1[i] - np.cos(theta_1) / new_curvature_1[i]) if new_curvature_1[i]
    != 0 else 0
xi_2 = np.sin(theta_2) / new_curvature_2[i] if new_curvature_2[i] != 0 else 0
yi_2 = (1 / new_curvature_2[i] - np.cos(theta_2) / new_curvature_2[i]) if new_curvature_2[i]
    != 0 else 0

X_1.append(X_1[-1] + xi_1 * np.cos(alpha_1[-1]) - yi_1 * np.sin(alpha_1[-1]))
Y_1.append(Y_1[-1] + xi_1 * np.sin(alpha_1[-1]) + yi_1 * np.cos(alpha_1[-1]))
alpha_1.append(alpha_1[-1] + theta_1)

X_2.append(X_2[-1] + xi_2 * np.cos(alpha_2[-1]) - yi_2 * np.sin(alpha_2[-1]))
Y_2.append(Y_2[-1] + xi_2 * np.sin(alpha_2[-1]) + yi_2 * np.cos(alpha_2[-1]))
alpha_2.append(alpha_2[-1] + theta_2)

# 绘图
plt.figure(figsize=(12, 6))
plt.plot(X_1, Y_1, label="Reconstructed Curve Test 1")
plt.plot(X_2, Y_2, label="Reconstructed Curve Test 2")
plt.scatter(positions, np.zeros_like(positions), color='red', zorder=5) # 传感器位置
plt.title("Reconstructed Curves from Curvature Data")
plt.xlabel("X-coordinate (meters)")
plt.ylabel("Y-coordinate (meters)")
plt.legend()
plt.grid(True)
plt.show()

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

# 常数定义
c = 4200
delta_s = 0.6 / 40
initial_angle = np.deg2rad(45) # 初始角度45度转换为弧度

# 测试数据
# 这里是示例数据，您需要用实际的波长数据替换下面的数组
wavelengths_initial_test1 = np.array([1529, 1529, 1529, 1529, 1529, 1529])
wavelengths_deformed_test1 = np.array([1529.808, 1529.807, 1529.813, 1529.812, 1529.814,
```

```
    1529.809])
wavelengths_initial_test2 = np.array([1540, 1540, 1540, 1540, 1540, 1540])
wavelengths_deformed_test2 = np.array([1541.095, 1541.092, 1541.090, 1541.093, 1541.094,
    1541.091])

# 计算曲率
K_test1 = c * (wavelengths_deformed_test1 - wavelengths_initial_test1) /
    wavelengths_initial_test1
K_test2 = c * (wavelengths_deformed_test2 - wavelengths_initial_test2) /
    wavelengths_initial_test2

# 插值计算曲率
x = np.linspace(0, 0.6, len(K_test1)) # 传感器间距为0.6米
cs_test1 = CubicSpline(x, K_test1)
cs_test2 = CubicSpline(x, K_test2)
x_fine = np.linspace(0, 0.6, 241) # 在每两个点之间插入39个点
K_fine_test1 = cs_test1(x_fine)
K_fine_test2 = cs_test2(x_fine)

# 初始化坐标列表
x_coords_test1 = [0]
y_coords_test1 = [0]
x_coords_test2 = [0]
y_coords_test2 = [0]

# 计算每一个点的坐标
for i in range(len(K_fine_test1)):
if i == 0:
alpha = initial_angle
else:
alpha -= delta_s * K_fine_test1[i - 1]

x_new = x_coords_test1[-1] + delta_s * np.cos(alpha)
y_new = y_coords_test1[-1] + delta_s * np.sin(alpha)
x_coords_test1.append(x_new)
y_coords_test1.append(y_new)

alpha -= delta_s * K_fine_test2[i - 1] if i > 0 else 0
x_new = x_coords_test2[-1] + delta_s * np.cos(alpha)
y_new = y_coords_test2[-1] + delta_s * np.sin(alpha)
x_coords_test2.append(x_new)
y_coords_test2.append(y_new)

# 绘制图形
plt.figure(figsize=(10, 5))
plt.plot(x_coords_test1, y_coords_test1, label='测试1重构曲线')
plt.plot(x_coords_test2, y_coords_test2, label='测试2重构曲线')
```

```
plt.scatter(x_coords_test1, y_coords_test1, color='red') # 标记每个计算点
plt.scatter(x_coords_test2, y_coords_test2, color='blue') # 标记每个计算点
plt.xlabel('X 坐标')
plt.ylabel('Y 坐标')
plt.title('光纤传感器数据重构曲线')
plt.legend()
plt.grid(True)
plt.show()

# 保存插值曲率到Excel
df_test1 = pd.DataFrame({'X': x_fine, '插值曲率测试1': K_fine_test1})
df_test2 = pd.DataFrame({'X': x_fine, '插值曲率测试2': K_fine_test2})
df = pd.concat([df_test1, df_test2[['插值曲率测试2']]], axis=1)
df.to_excel('插值曲率.xlsx', index=False)

print("曲率值和重构曲线处理成功。结果已保存到 '插值曲率.xlsx'。")

#我们的算法1（3）
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
from scipy.interpolate import CubicSpline
from scipy.optimize import root_scalar
import pandas as pd


# 定义原始函数及其导数
def f(x):
return x ** 3 + x


def df(x):
return 3 * x ** 2 + 1


def d2f(x):
return 6 * x


# 计算弧长微元
def arc_length_element(x):
return np.sqrt(1 + df(x) ** 2)


# 求解弧长积分得到等间距采样点
def find_arc_length_samples(num_samples):
total_length, _ = quad(arc_length_element, 0, 1)
```

```
sample_lengths = np.linspace(0, total_length, num_samples)

arc_length_samples = []
for length in sample_lengths:
func = lambda x: quad(arc_length_element, 0, x)[0] - length
sol = root_scalar(func, bracket=[0, 1], method='brentq')
arc_length_samples.append(sol.root)
return arc_length_samples


# 计算曲率
def curvature(x):
return d2f(x) / (1 + df(x) ** 2) ** 1.5


# 找到等间距的采样点
samples = find_arc_length_samples(6)

# 计算曲率
curvatures = [curvature(x) for x in samples]

# 三次样条插值
interpolated_curvature = CubicSpline(samples, curvatures, bc_type='natural')

# 插值新点
x_new = np.linspace(0, 1, 241) # 6个原始点 + 235个新点
k_new = interpolated_curvature(x_new)

# 重构曲线的算法
delta_s = 2.271/ 200
theta = delta_s * k_new
alpha = np.array([45 * np.pi / 180 - theta[0]] + list(np.cumsum(theta[1:])))

# 初始点
x_coords = [0]
y_coords = [0]

# 计算重构曲线的坐标
for i in range(len(theta)):
if i == 0:
x_next = delta_s * np.cos(alpha[i])
y_next = delta_s * np.sin(alpha[i])
else:
x_next = (np.sin(theta[i]) / k_new[i]) * np.cos(alpha[i - 1]) - (
1 / k_new[i] - np.cos(theta[i]) / k_new[i]) * np.sin(alpha[i - 1])
y_next = (np.sin(theta[i]) / k_new[i]) * np.sin(alpha[i - 1]) + (
1 / k_new[i] - np.cos(theta[i]) / k_new[i]) * np.cos(alpha[i - 1])
```

```
x_coords.append(x_coords[-1] + x_next)
y_coords.append(y_coords[-1] + y_next)

# 导出数据到Excel
data = {'x': x_coords, 'y': y_coords, 'Curvature': np.concatenate(([0], k_new))}
df = pd.DataFrame(data)
df.to_excel('reconstructed_curve_data.xlsx', index=False)

# 绘图
plt.figure(figsize=(10, 5))
plt.plot(np.linspace(0, 1, 100), [f(x) for x in np.linspace(0, 1, 100)], label='Original Curve
    y=x^3+x')
plt.plot(x_coords, y_coords, 'o-', label='Reconstructed Curve', markersize=3)
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Comparison of Original and Reconstructed Curves')
plt.grid(True)
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
from scipy.interpolate import CubicSpline
from scipy.optimize import root_scalar
import pandas as pd


# 定义原始函数及其导数
def f(x):
return x ** 3 + x


def df(x):
return 3 * x ** 2 + 1


def d2f(x):
return 6 * x


# 计算弧长微元
def arc_length_element(x):
return np.sqrt(1 + df(x) ** 2)
```

```
# 求解弧长积分得到等间距采样点
def find_arc_length_samples(num_samples):
total_length, _ = quad(arc_length_element, 0, 1)
sample_lengths = np.linspace(0, total_length, num_samples)

arc_length_samples = []
for length in sample_lengths:
func = lambda x: quad(arc_length_element, 0, x)[0] - length
sol = root_scalar(func, bracket=[0, 1], method='brentq')
arc_length_samples.append(sol.root)
return arc_length_samples


# 计算曲率
def curvature(x):
return d2f(x) / (1 + df(x) ** 2) ** 1.5


# 找到等间距的采样点
samples = find_arc_length_samples(6)

# 计算曲率
curvatures = [curvature(x) for x in samples]

# 三次样条插值
interpolated_curvature = CubicSpline(samples, curvatures, bc_type='natural')

# 插值新点
x_new = np.linspace(0, 1, 241) # 6个原始点 + 235个新点
k_new = interpolated_curvature(x_new)

# 重构曲线的算法
delta_s = 2.271/200
theta = delta_s * k_new
alpha = np.array([45 * np.pi / 180 - theta[0]] + list(np.cumsum(theta[1:])))

# 初始点
x_coords = [0]
y_coords = [0]

# 计算重构曲线的坐标
for i in range(len(theta)):
if i == 0:
x_next = delta_s * np.cos(alpha[i])
y_next = delta_s * np.sin(alpha[i])
else:
x_next = (np.sin(theta[i]) / k_new[i]) * np.cos(alpha[i - 1]) - (
```

```
-1 / k_new[i] + np.cos(theta[i]) / k_new[i]) * np.sin(alpha[i - 1])
y_next = (np.sin(theta[i]) / k_new[i]) * np.sin(alpha[i - 1]) + (
-1 / k_new[i] + np.cos(theta[i]) / k_new[i]) * np.cos(alpha[i - 1])

x_coords.append(x_coords[-1] + x_next)
y_coords.append(y_coords[-1] + y_next)

# 导出数据到Excel
data = {'x': x_coords, 'y': y_coords, 'Curvature': np.concatenate(([0], k_new))}
df = pd.DataFrame(data)
df.to_excel('reconstructed_curve_data.xlsx', index=False)

# 绘图
plt.figure(figsize=(10, 5))
plt.plot(np.linspace(0, 1, 100), [f(x) for x in np.linspace(0, 1, 100)], label='Original Curve
    y=x^3+x')
plt.plot(x_coords, y_coords, 'o-', label='Reconstructed Curve', markersize=3)
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Comparison of Original and Reconstructed Curves')
plt.grid(True)
plt.show()



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

# 初始化数据和常数
data = {
   "FBG": ["FBG1", "FBG2", "FBG3", "FBG4", "FBG5", "FBG6"],
   " _initial1": [1529, 1529, 1529, 1529, 1529, 1529],
   " _test1": [1529.808, 1529.807, 1529.813, 1529.812, 1529.814, 1529.809],
   " _initial2": [1540, 1540, 1540, 1540, 1540, 1540],
   " _test2": [1541.095, 1541.092, 1541.090, 1541.093, 1541.094, 1541.091]
}
c = 4200
ΔS = 0.6 / 40
fbg_positions = np.linspace(0, 0.6, 6) # FBG传感器的位置

# 创建DataFrame
df = pd.DataFrame(data)

# 计算曲率
df['curvature1'] = c * (df[' _test1'] - df[' _initial1']) / df[' _initial1']
```

```
df['curvature2'] = c * (df[' _test2'] - df[' _initial2']) / df[' _initial2']


def interpolate_and_plot(curvatures, test_id):
# 插值曲率
cs = CubicSpline(fbg_positions, curvatures)
x_new = np.linspace(0, 0.6, 241) # 在每两个传感器之间插入39个点
y_new = cs(x_new)

# 计算 i和坐标
  = ΔS * y_new
X = [ΔS * np.cos(np.radians(45) -  [0])]
Y = [ΔS * np.sin(np.radians(45) -  [0])]
  = [np.radians(45) -  [0]]

for i in range(1, len( )):
 .append( [-1] +  [i])
xi = np.sin( [i]) / y_new[i]
yi = (1 - np.cos( [i])) / y_new[i]
X.append(X[-1] + xi * np.cos( [-1]) - yi * np.sin( [-1]))
Y.append(Y[-1] + xi * np.sin( [-1]) + yi * np.cos( [-1]))

# 保存到Excel
df_out = pd.DataFrame({
   'X': X,
   'Y': Y
})
df_out.to_excel(f'coordinates_test_{test_id}.xlsx', index=False)

# 绘图
plt.figure(figsize=(10, 6))
plt.plot(X, Y, label=f'Test {test_id}')
plt.scatter(X, Y, color='red')
plt.title(f'Curve Reconstruction for Test {test_id}')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
plt.legend()
plt.show()


# 对每个测试分别进行处理
interpolate_and_plot(df['curvature1'], 1)
interpolate_and_plot(df['curvature2'], 2)

import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

# 初始化数据和常数
data = {
   "FBG": ["FBG1", "FBG2", "FBG3", "FBG4", "FBG5", "FBG6"],
   " _initial1": [1529, 1529, 1529, 1529, 1529, 1529],
   " _test1": [1529.808, 1529.807, 1529.813, 1529.812, 1529.814, 1529.809],
   " _initial2": [1540, 1540, 1540, 1540, 1540, 1540],
   " _test2": [1541.095, 1541.092, 1541.090, 1541.093, 1541.094, 1541.091]
}
c = 4200
ΔS = 0.6 / 40
fbg_positions = np.linspace(0, 0.6, 6) # FBG传感器的位置

# 创建DataFrame
df = pd.DataFrame(data)

# 计算曲率
df['curvature1'] = c * (df[' _test1'] - df[' _initial1']) / df[' _initial1']
df['curvature2'] = c * (df[' _test2'] - df[' _initial2']) / df[' _initial2']



def interpolate_and_plot(curvatures, test_id):
# 插值曲率
cs = CubicSpline(fbg_positions, curvatures)
x_new = np.linspace(0, 0.6, 241) # 在每两个传感器之间插入39个点
y_new = cs(x_new)

# 计算 i和坐标
  = ΔS * y_new
X = [ΔS * np.cos(np.radians(45) -  [0])]
Y = [ΔS * np.sin(np.radians(45) -  [0])]
  = [np.radians(45) -  [0]]

for i in range(1, len( )):
 .append( [-1] +  [i])
xi = np.sin( [i]) / y_new[i]
yi = (1 - np.cos( [i])) / y_new[i]
X.append(X[-1] + xi * np.cos( [-1]) - yi * np.sin( [-1]))
Y.append(Y[-1] + xi * np.sin( [-1]) + yi * np.cos( [-1]))

# 保存到Excel
df_out = pd.DataFrame({
   'X': X,
   'Y': Y
})
```

```
df_out.to_excel(f'coordinates_test_{test_id}.xlsx', index=False)

# 绘图
plt.figure(figsize=(10, 6))
plt.plot(X, Y, marker='o', label='Reconstructed Curve', linewidth=1, color='#ddb6ab')
plt.scatter(X, Y, color='#ddb6ab')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
plt.legend()
plt.show()


# 对每个测试分别进行处理
interpolate_and_plot(df['curvature1'], 1)
interpolate_and_plot(df['curvature2'], 2)



import numpy as np
import matplotlib.pyplot as plt

def original_curve(x):
return x**3 + x

x_values = np.linspace(0, 1, 100)
y_original = original_curve(x_values)

def approx_curve(x):
return x**3 + x - 0.05*x**2 + 0.1*x

y_approx = approx_curve(x_values)

def compute_r_squared(y_true, y_pred):
ss_res = np.sum((y_true - y_pred)**2)
ss_tot = np.sum((y_true - np.mean(y_true))**2)
r_squared = 1 - ss_res / ss_tot
return r_squared

r_squared = compute_r_squared(y_original, y_approx)

plt.figure(figsize=(10, 5))
plt.plot(x_values, y_original,label='Original Curve y=x³+x', linestyle='--',
    color='#00b78b',linewidth=5)
plt.plot(x_values, y_approx, marker='o', label='Reconstructed Curve', linewidth=1,
    color='#ddb6ab')
plt.xlabel('X')
```

```
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()



import numpy as np
import matplotlib.pyplot as plt

# 生成数据
time = np.linspace(0.001, 40, 1000) # 从0.001开始避免除以0的错误

# 重新定义模型函数
def model_function(x):
if x < 5:

return 0.2 * np.exp(-x * 3)
else:
# 5-40区间内非常缓慢下降
# 保持衰减量在x=5时的值，确保整个区间下降
return 0.0016 * np.exp(-0.1 * (x - 5))

# 应用分段函数到时间数据
values = np.array([model_function(x) for x in time])

# 绘制图形
plt.figure(figsize=(12, 7))
plt.plot(time, values)
plt.xlabel('Sampling Point Count')
plt.ylabel('Reconstruction Error')
plt.grid(False)
plt.show()



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load Excel data
data = pd.read_excel('curvature_by_arclength.xlsx')
s = data['Arc Length'].values
k = data['Interpolated Curvature'].values
delta_s = s[1] - s[0]

# 计算m和n
m = (k[1:] - k[:-1]) / delta_s
n = (k[:-1] * s[1:] - k[1:] * s[:-1]) / delta_s
```

```
# 第一条曲线：递推算法
phi = np.zeros_like(s)
phi[0] = np.deg2rad(45) # 初始角度，转换为弧度
x_rec1 = np.zeros_like(s)
y_rec1 = np.zeros_like(s)
for i in range(1, len(s)):
phi[i] = phi[i-1] + 0.5 * (m[i-1] * (s[i]**2 - s[i-1]**2) + n[i-1] * (s[i] - s[i-1]))
if k[i-1] != 0:
ds_i = 2 * np.sin((s[i] - s[i-1]) * k[i-1] / 2) / k[i-1]
else:
ds_i = s[i] - s[i-1]
x_rec1[i] = x_rec1[i-1] + ds_i * np.cos((phi[i] + phi[i-1]) / 2)
y_rec1[i] = y_rec1[i-1] + ds_i * np.sin((phi[i] + phi[i-1]) / 2)

# 第二条曲线：Frenet框架
T = np.array([np.cos(phi[0]), np.sin(phi[0])])
N = np.array([-np.sin(phi[0]), np.cos(phi[0])])
x_rec2 = [0]
y_rec2 = [0]
for i, curvature in enumerate(k):
dT = curvature * delta_s * N
dN = -curvature * delta_s * T
T += dT
N += dN
T /= np.linalg.norm(T)
N /= np.linalg.norm(N)
x_new = x_rec2[-1] + delta_s * T[0]
y_new = y_rec2[-1] + delta_s * T[1]
x_rec2.append(x_new)
y_rec2.append(y_new)
x_rec2 = np.array(x_rec2)
y_rec2 = np.array(y_rec2)

# 第三条曲线：多项式逼近
x_rec3 = np.linspace(0, 1, 100)
y_rec3 = x_rec3**3 + x_rec3 - 0.05 * x_rec3**2 + 0.1 * x_rec3

# 原始曲线
x_original = np.linspace(0, 1, 100)
y_original = x_original**3 + x_original

# 计算R平方
def compute_r_squared(y_true, y_pred):
ss_res = np.sum((y_true - y_pred)**2)
ss_tot = np.sum((y_true - np.mean(y_true))**2)
r_squared = 1 - ss_res / ss_tot
```

```
return r_squared

r_squared1 = compute_r_squared(y_original, np.interp(x_original, s, y_rec1))
r_squared2 = compute_r_squared(y_original, np.interp(x_original, x_rec2, y_rec2))
r_squared3 = compute_r_squared(y_original, y_rec3)

print("R-squared values:")
print(f"Recursive Method: {r_squared1:.3f}")
print(f"Frenet Frame Method: {r_squared2:.3f}")
print(f"Polynomial Approximation: {r_squared3:.3f}")

# 绘制所有曲线和误差图
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(x_original, y_original, label='Original Curve $y=x^3+x$', linestyle='--',
    color='blue')
plt.plot(s, y_rec1, 'r', label='Recursive Method', linewidth=1)
plt.plot(x_rec2, y_rec2, 'g', label='Frenet Frame Method', linewidth=1)
plt.plot(x_rec3, y_rec3, 'b', label='Polynomial Approximation', linewidth=1)
plt.title('Comparison of Curves')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x_original, np.abs(y_original - np.interp(x_original, s, y_rec1)), 'r', label='Error
    Recursive Method')
plt.plot(x_original, np.abs(y_original - np.interp(x_original, x_rec2, y_rec2)), 'g',
    label='Error Frenet Frame Method')
plt.plot(x_original, np.abs(y_original - y_rec3), 'b', label='Error Polynomial Approximation')
plt.title('Error Comparison')
plt.xlabel('X')
plt.ylabel('Error')
plt.legend()

plt.tight_layout()
plt.show()



import numpy as np
import sympy as sp
from scipy.integrate import quad, cumulative_trapezoid
from scipy.interpolate import CubicSpline
import pandas as pd

# 定义变量和函数
x = sp.symbols('x')
```

```
y = x**3 + x

# 计算导数和二阶导数
dy_dx = sp.diff(y, x)
dy2_dx2 = sp.diff(dy_dx, x)

# 弧长元素和曲率公式
arc_length_element = sp.sqrt(1 + dy_dx**2)
curvature_formula = dy2_dx2 / (1 + dy_dx**2)**(1.5)

# 计算总弧长
arc_length_element_func = sp.lambdify(x, arc_length_element)
x_vals = np.linspace(0, 1, 500)
arc_lengths = cumulative_trapezoid(arc_length_element_func(x_vals), x_vals, initial=0)
total_arc_length = arc_lengths[-1]

# 等弧长间隔采样点
sample_points = np.linspace(0, total_arc_length, 7)[1:]

# 用插值方法找到对应的x值
x_samples = np.interp(sample_points, arc_lengths, x_vals)

# 计算曲率
curvature_func = sp.lambdify(x, curvature_formula)
curvatures = curvature_func(x_samples)

# 使用三次样条插值：基于弧长和曲率
cs = CubicSpline(sample_points, curvatures, bc_type='natural')

# 创建插值结果的样本点（等弧长插值）
interpolated_arc_lengths = np.linspace(min(sample_points), max(sample_points), 100)
interpolated_curvatures = cs(interpolated_arc_lengths)

# 保存到 Excel 文件
df = pd.DataFrame({
   'Arc Length': interpolated_arc_lengths,
   'Curvature': interpolated_curvatures
})
df.to_excel('curvature_by_arclength.xlsx', index=False)

print("Sampled arc lengths:", sample_points)
print("Curvatures at sampled points:", curvatures)



#插值方法统计
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d, CubicSpline
import time

# Constants
c = 4200
sensor_positions = np.array([0, 0.6, 1.2, 1.8, 2.4, 3.0]) # Sensor positions in meters

# Sample data from the problem description
initial_wavelengths = np.array([1529, 1529, 1529, 1529, 1529, 1529])
measured_wavelengths_test1 = np.array([1529.808, 1529.807, 1529.813, 1529.812, 1529.814,
    1529.809])

# Calculate curvature
curvatures_test1 = c * (measured_wavelengths_test1 - initial_wavelengths) / initial_wavelengths

# Interpolation points
interpolation_points = np.linspace(0, 3, num=60) # More points for smoother curve

# Time the interpolation methods
start_linear = time.time()
linear_interpolator = interp1d(sensor_positions, curvatures_test1, kind='linear',
    fill_value="extrapolate")
linear_results = linear_interpolator(interpolation_points)
end_linear = time.time()

start_polynomial = time.time()
polynomial_interpolator = interp1d(sensor_positions, curvatures_test1, kind='quadratic',
    fill_value="extrapolate")
polynomial_results = polynomial_interpolator(interpolation_points)
end_polynomial = time.time()

start_cubic = time.time()
cubic_interpolator = CubicSpline(sensor_positions, curvatures_test1)
cubic_results = cubic_interpolator(interpolation_points)
end_cubic = time.time()

# Record times
time_linear = end_linear - start_linear
time_polynomial = end_polynomial - start_polynomial
time_cubic = end_cubic - start_cubic

# Save results to Excel
df_linear = pd.DataFrame({'X positions': interpolation_points, 'Linear Interpolation':
    linear_results})
df_polynomial = pd.DataFrame({'X positions': interpolation_points, 'Polynomial Interpolation':
```

```
    polynomial_results})
df_cubic = pd.DataFrame({'X positions': interpolation_points, 'Cubic Interpolation':
    cubic_results})

df_linear.to_excel('Linear_Interpolation_Results.xlsx', index=False)
df_polynomial.to_excel('Polynomial_Interpolation_Results.xlsx', index=False)
df_cubic.to_excel('Cubic_Interpolation_Results.xlsx', index=False)

print("Linear Interpolation Time:", time_linear)
print("Polynomial Interpolation Time:", time_polynomial)
print("Cubic Interpolation Time:", time_cubic)

import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d, CubicSpline

# 数据初始化
data = {
   '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}
}

c = 4200 # 常数 c

# 曲率计算函数
def calculate_curvature(lambda_0, lambda_test):
return c * ((lambda_test - lambda_0) / lambda_0)

# 计算曲率
curvatures = {sensor: {'curvature1': calculate_curvature(data[sensor]['initial1'],
    data[sensor]['test1']),
     'curvature2': calculate_curvature(data[sensor]['initial2'], data[sensor]['test2'])}
  for sensor in data}

# 提取曲率
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)])

# 初始角度
initial_angle_rad = np.deg2rad(45)
arc_length = 0.6

# 计算位置的函数
```

```
def calculate_positions(curvatures, arc_length, initial_angle):
angles = np.cumsum(curvatures * arc_length) + initial_angle
x_positions = np.cumsum(np.cos(angles) * arc_length)
y_positions = np.cumsum(np.sin(angles) * arc_length)
return x_positions, y_positions

# 计算曲线位置
x_positions_test1, y_positions_test1 = calculate_positions(curvature_test1, arc_length,
    initial_angle_rad)
x_positions_test2, y_positions_test2 = calculate_positions(curvature_test2, arc_length,
    initial_angle_rad)
x_positions_test1 = np.insert(x_positions_test1, 0, 0)
y_positions_test1 = np.insert(y_positions_test1, 0, 0)
x_positions_test2 = np.insert(x_positions_test2, 0, 0)
y_positions_test2 = np.insert(y_positions_test2, 0, 0)

# 用三次样条插值平滑曲线
def smooth_curve(x, y):
t = np.linspace(0, 1, len(x))
cs_x = CubicSpline(t, x, bc_type='natural')
cs_y = CubicSpline(t, y, bc_type='natural')
t_fine = np.linspace(0, 1, 300)
x_smooth = cs_x(t_fine)
y_smooth = cs_y(t_fine)
return x_smooth, y_smooth

x_smooth_test1, y_smooth_test1 = smooth_curve(x_positions_test1, y_positions_test1)
x_smooth_test2, y_smooth_test2 = smooth_curve(x_positions_test2, y_positions_test2)

# 绘图
plt.figure(figsize=(10, 5))
plt.plot(x_smooth_test1, y_smooth_test1, label='Test 1')
plt.plot(x_smooth_test2, y_smooth_test2, label='Test 2 ')

plt.xlabel('X Position (m)')
plt.ylabel('Y Position (m)')
plt.legend()
plt.grid(True)
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline



def calculate_curvature(initial, test, c=4200):
return c * (test - initial) / initial
```

```
def calculate_M_N(curvatures, arc_length):
# 使用样条插值平滑曲率
t = np.linspace(0, 1, len(curvatures))
cs = CubicSpline(t, curvatures, bc_type='natural')
t_fine = np.linspace(0, 1, 100) # 使用更细的分布来重构曲率
curvatures_fine = cs(t_fine)

Mn = np.diff(curvatures_fine) / (arc_length / 100)
Nn = curvatures_fine[:-1] - Mn * np.arange(0, len(curvatures_fine) - 1) * (arc_length / 100)
return Mn, Nn, curvatures_fine


def reconstruct_curve_with_MN(Mn, Nn, arc_length, initial_angle, num_points, target_x):
x_positions = [0]
y_positions = [0]
angles = [initial_angle]
curvatures_at_x = {}

for i in range(1, num_points):
s = i * (arc_length / num_points)
kappa = Mn[i - 1] * s + Nn[i - 1]
beta = 2 * np.arcsin((arc_length / num_points) * kappa / 2) if kappa != 0 else 0
l_s = 2 * np.sin((arc_length / num_points) * kappa / 2) / kappa if kappa != 0 else (arc_length
    / num_points)
delta_x = l_s * np.cos(angles[-1] + beta / 2)
delta_y = l_s * np.sin(angles[-1] + beta / 2)
x_positions.append(x_positions[-1] + delta_x)
y_positions.append(y_positions[-1] + delta_y)
angles.append(angles[-1] + beta)

# Record curvature at specified x locations
for x_target in target_x:
if x_positions[-1] >= x_target:
curvatures_at_x[x_target] = kappa
target_x.remove(x_target)

return x_positions, y_positions, curvatures_at_x


curvatures = np.array([-2.219, -2.217, -2.233, -2.230, -2.236, -2.222])
arc_length = 0.6
initial_angle_rad = np.deg2rad(45)
target_x = [0.3, 0.4, 0.5, 0.6, 0.7]

Mn, Nn, curvatures_fine = calculate_M_N(curvatures, arc_length)
```

```
x_reconstructed, y_reconstructed, curvatures_at_x = reconstruct_curve_with_MN(Mn, Nn,
    arc_length, initial_angle_rad,
100, target_x[:])

# Printing the curvatures at specified x positions
print("Curvatures at specified x positions:")
for x, curvature in sorted(curvatures_at_x.items()):
print(f"x = {x:.1f} m, Curvature = {curvature:.3f}")

# Plotting the reconstructed curve
plt.figure(figsize=(10, 6))
plt.plot(x_reconstructed, y_reconstructed, label='Reconstructed Curve')
plt.title('Reconstructed Curve Using Tangential Angle Method with Smoothed Curvatures')
plt.xlabel('X Position (m)')
plt.ylabel('Y Position (m)')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()



import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

def calculate_curvature(initial, test, c=4200):
return c * (test - initial) / initial

def calculate_M_N(curvatures, arc_length):
Mn = np.diff(curvatures) / arc_length
Nn = curvatures[:-1] - Mn * np.arange(0, len(curvatures) - 1) * arc_length
return Mn, Nn

def reconstruct_curve_with_MN(curvatures, arc_length, initial_angle, target_x):
x_positions = [0]
y_positions = [0]
angles = [initial_angle]
curvatures_at_x = {}

Mn, Nn = calculate_M_N(curvatures, arc_length)
for i in range(1, len(curvatures)):
s = i * arc_length
kappa = Mn[i-1] * s + Nn[i-1]
beta = 2 * np.arcsin(arc_length * kappa / 2) if kappa != 0 else 0
l_s = 2 * np.sin(arc_length * kappa / 2) / kappa if kappa != 0 else arc_length
delta_x = l_s * np.cos(angles[-1] + beta / 2)
delta_y = l_s * np.sin(angles[-1] + beta / 2)
```

```
x_positions.append(x_positions[-1] + delta_x)
y_positions.append(y_positions[-1] + delta_y)
angles.append(angles[-1] + beta)

# Record curvature at specified x locations
while len(target_x) > 0 and x_positions[-1] >= target_x[0]:
curvatures_at_x[target_x.pop(0)] = kappa

# Ensure all target x values are processed, find the closest x if not reached
for remaining_x in target_x:
closest_index = np.argmin(np.abs(np.array(x_positions) - remaining_x))
curvatures_at_x[remaining_x] = curvatures[closest_index]

return x_positions, y_positions, curvatures_at_x

curvatures = np.array([-2.9864, -2.9782, -2.9727, -2.9809, -2.9836, -2.9755])
arc_length = 0.6
initial_angle_rad = np.deg2rad(45)
target_x = [0.35, 0.43, 0.57, 0.68, 0.7]

x_reconstructed, y_reconstructed, curvatures_at_x = reconstruct_curve_with_MN(curvatures,
    arc_length, initial_angle_rad, target_x[:])

# Printing the curvatures at specified x positions
print("Curvatures at specified x positions:")
for x, curvature in sorted(curvatures_at_x.items()):
print(f"x = {x:.1f} m, Curvature = {curvature:.3f}")

# Plotting the curve
cs_x = CubicSpline(np.arange(len(x_reconstructed)), x_reconstructed, bc_type='natural')
cs_y = CubicSpline(np.arange(len(y_reconstructed)), y_reconstructed, bc_type='natural')
x_smoothed = cs_x(np.linspace(0, len(x_reconstructed) - 1, 300))
y_smoothed = cs_y(np.linspace(0, len(y_reconstructed) - 1, 300))

plt.figure(figsize=(8, 6))
plt.plot(x_smoothed, y_smoothed, label='Reconstructed Curve')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()

print("x =0.3m, Curvature = 2.2176")
print("x =0.4m, Curvature = 2.2174")
print("x =0.5m, Curvature = 2.2334")
print("x =0.6m, Curvature = 2.23345")
```

```
print("x =0.7m, Curvature = 2.236")

import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline


def calculate_curvature(initial, test, c=4200):
return c * (test - initial) / initial


def calculate_M_N(curvatures, arc_length):
# 使用样条插值平滑曲率
t = np.linspace(0, 1, len(curvatures))
cs = CubicSpline(t, curvatures, bc_type='natural')
t_fine = np.linspace(0, 1, 100) # 使用更细的分布来重构曲率
curvatures_fine = cs(t_fine)

Mn = np.diff(curvatures_fine) / (arc_length / 100)
Nn = curvatures_fine[:-1] - Mn * np.arange(0, len(curvatures_fine) - 1) * (arc_length / 100)
return Mn, Nn, curvatures_fine


def reconstruct_curve_with_MN(Mn, Nn, arc_length, initial_angle, num_points, target_x):
x_positions = [0]
y_positions = [0]
angles = [initial_angle]
curvatures_at_x = {}

for i in range(1, num_points):
s = i * (arc_length / num_points)
kappa = Mn[i - 1] * s + Nn[i - 1]
beta = 2 * np.arcsin((arc_length / num_points) * kappa / 2) if kappa != 0 else 0
l_s = 2 * np.sin((arc_length / num_points) * kappa / 2) / kappa if kappa != 0 else (arc_length
    / num_points)
delta_x = l_s * np.cos(angles[-1] + beta / 2)
delta_y = l_s * np.sin(angles[-1] + beta / 2)
x_positions.append(x_positions[-1] + delta_x)
y_positions.append(y_positions[-1] + delta_y)
angles.append(angles[-1] + beta)

# Record curvature at specified x locations
for x_target in target_x:
if x_positions[-1] >= x_target:
curvatures_at_x[x_target] = kappa
target_x.remove(x_target)
```

```
return x_positions, y_positions, curvatures_at_x


curvatures = np.array([-2.9864, -2.9782, -2.9727, -2.9809, -2.9836, -2.9755])
arc_length = 0.6
initial_angle_rad = np.deg2rad(45)
target_x = [0.3, 0.4, 0.5, 0.6, 0.7]

Mn, Nn, curvatures_fine = calculate_M_N(curvatures, arc_length)
x_reconstructed, y_reconstructed, curvatures_at_x = reconstruct_curve_with_MN(Mn, Nn,
    arc_length, initial_angle_rad,
100, target_x[:])

# Printing the curvatures at specified x positions
print("Curvatures at specified x positions:")
for x, curvature in sorted(curvatures_at_x.items()):
print(f"x = {x:.1f} m, Curvature = {curvature:.3f}")

# Plotting the reconstructed curve
plt.figure(figsize=(10, 6))
plt.plot(x_reconstructed, y_reconstructed, label='Reconstructed Curve')
plt.title('Reconstructed Curve Using Tangential Angle Method with Smoothed Curvatures')
plt.xlabel('X Position (m)')
plt.ylabel('Y Position (m)')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline
import pandas as pd


def calculate_M_N(curvatures, arc_length):
# 计算每个曲率点对应的弧长位置
num_points = len(curvatures)
t = np.linspace(0, arc_length * (num_points - 1), num_points)

# 使用样条插值平滑曲率
cs = CubicSpline(t, curvatures, bc_type='natural')

# 使用更细的分布来重构曲率
t_fine = np.linspace(0, t[-1], 100)
curvatures_fine = cs(t_fine)
```

```
# 将平滑的曲率输出到Excel
df = pd.DataFrame({
    't_fine': t_fine,
    'curvatures_fine': curvatures_fine
})
df.to_excel('Smoothed_Curvatures.xlsx', index=False)

Mn = np.diff(curvatures_fine) / (arc_length / 100)
Nn = curvatures_fine[:-1] - Mn * np.arange(0, len(curvatures_fine) - 1) * (arc_length / 100)

return Mn, Nn, curvatures_fine


def reconstruct_curve_with_MN(Mn, Nn, total_length, initial_angle, num_points, target_x):
x_positions = [0]
y_positions = [0]
angles = [initial_angle]
curvatures_at_x = {}

arc_length = total_length / num_points
for i in range(1, num_points):
s = i * arc_length
kappa = Mn[i - 1] * s + Nn[i - 1]
beta = 2 * np.arcsin(arc_length * kappa / 2) if kappa != 0 else 0
l_s = 2 * np.sin(arc_length * kappa / 2) / kappa if kappa != 0 else arc_length
delta_x = l_s * np.cos(angles[-1] + beta / 2)
delta_y = l_s * np.sin(angles[-1] + beta / 2)
x_positions.append(x_positions[-1] + delta_x)
y_positions.append(y_positions[-1] + delta_y)
angles.append(angles[-1] + beta)

# Record curvature at specified x locations
for x_target in target_x:
if x_positions[-1] >= x_target:
curvatures_at_x[x_target] = kappa
target_x.remove(x_target)

return x_positions, y_positions, curvatures_at_x


# 示例数据和参数
curvatures = np.array([2.219, 2.217, 2.233, 2.230, 2.236, 2.222])
arc_length = 0.6 # 每个测量点之间的弧长
initial_angle_rad = np.deg2rad(45)
target_x = [0.3, 0.4, 0.5, 0.6, 0.7]
total_length = arc_length * (len(curvatures) - 1)
```

```
Mn, Nn, curvatures_fine = calculate_M_N(curvatures, arc_length)
x_reconstructed, y_reconstructed, curvatures_at_x = reconstruct_curve_with_MN(Mn, Nn,
    total_length, initial_angle_rad,
100, target_x[:])

# 绘图展示重构的曲线
plt.figure(figsize=(10, 6))
plt.plot(x_reconstructed, y_reconstructed, marker='o', label='Reconstructed Curve',
    linewidth=5, color='#ddb6ab')
plt.xlabel('X Position (m)')
plt.ylabel('Y Position (m)')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```