基于优化的切角递推算法重构平面曲线

摘要

本文以光纤传感器技术为研究基础，采用数学建模和算法分析方法，对光纤传感器在受力后的曲率变化进行估算，并探讨了基于这些曲率数据的平面曲线重构问题。研究工作主要分为三个核心问题，并围绕这些问题展开详细的分析与模型构建。

针对问题一，本文首先根据给定的波长测量数据，利用波长与曲率之间的关系公式，计算出测量数据点的曲率，通过**线性插值算法**和**三次样条插值算法**在每个测量数据点之间**插入一百个点**，对所有插值点的曲率进行估算并可视化。通过对比两种算法估算的曲率可视化图像发现，**三次样条插值算法**在处理非线性关系和**生成平滑曲线**方面具有**显著优势**，因此**基于三次样条插值算法建立曲率估算模型**。并通过该模型估算出 x 分别在[0.3　0.4　0.5　0.6　0.7]点处的曲率，**详见表 1**。

对于问题二，本文利用问题一得到的所有插值点的曲率数据，以及题目给出条件：光纤在平面受力后在初始位置的切线与水平方向的**夹角为 45°**，采用**斜率递推算法**和**切角递推算法**两种方法对平面曲线进行重构。根据**斜率递推算法重构出来的平面曲线**，发现当在曲线上某点切线的**斜率由负无穷到正无穷时**（或者由正无穷到负无穷）时**斜率会发生反转**，因此斜率递推算法**不适合重构斜率发生反转的曲线**。而**切角递推算法**重构出来的平面曲线结果更为准确，能够**更准确地反映曲线的真实形态**。利用切角递推算法逐步构建出整个曲线上每个点的坐标，实现了对曲线形状的有效重构。**对重构的曲线特点进行分析时**，发现有以下几个特点：1.**起始点与终止点不重合的类椭圆形。2.曲线内缩，有向中心靠拢的趋势。**

对于问题三，本文基于给定的平面曲线方程，通过**数值积分方法**计算函数在给定区间的弧长进行**等间距弧长采样**，并计算出这些采样点的 x 轴坐标，进而计算出采样点的曲率。接着构建了**优化的切角递推模型**。该模型考虑了步长与曲率之间的非线性关系，**通过动态调整步长**来优化重构过程，从而提高了重构曲线的精度。此外，还探讨了重构曲线与原始曲线之间可能**出现的偏差原因**，包括数值积分的精度、步长的确定、曲率计算的近似、重构算法的局限性以及舍入误差等，并**提出了相应的改进措施**。

关键词：**线性插值算法　三次样条插值　斜率递推算法　切角递推算法　优化切角递推算法**

目录

一、问题重述

1.1 问题背景

光纤传感技术是伴随着光纤及光通信技术发展起来的一种新型传感器技术。它是以光波为传感信号、光纤为传输载体来感知外界环境中的信号，其基本原理是当外界环境参数发生变化时，会引起光纤传感器中光波参量(如波长、相位、强度等)的变化，即外界信号变化会对光信号产生调制。光纤传感器具有质地轻、体积小、弯曲性能好，抗电磁干扰能力强，灵敏度高，易于安装使用等优点。光纤传感技术最重要的是实时获得结构实时应变信息，再通过解调出来的应变参数来重构得到结构的形变或位移。

光纤传感器已在许多领域有实际应用，比如能够对结肠部位进行形状重建等。通过光纤传感器解调系统解调出来的应变信息，间接求出曲率等信息，并基于离散曲率信息对曲线进行重构。

1.2 问题提出

根据以上背景，及题目所给的数据信息，需要解决以下三个问题：

问题一：根据提供的波长测量数据，计算出曲率，建立数学模型，估算表格中横坐标 x 轴位置处的曲率。

问题二：构建一个数学模型，使用问题一中得到的曲率数据，以及题目给出的条件重构平面上的曲线并探讨曲线特征。

问题三：根据平面曲线方程 $y = x^3 + x(0 \le x \le 1)$，通过等间距的弧长进行采样，确定采样点的曲率，然后基于采样点的曲率，优化题二中的数学模型来重构平面曲线，并探讨重构曲线与原始曲线之间可能出现的偏差原因。

二、问题分析

2.1 问题一的分析：

问题一要求依据表 1 的波长测量数据以及波长与曲线曲率之间的关系公式建立数学模型，估算平面光栅各个传感点的曲率。利用题目提供的波长与曲率之间的关系式，我们在原定区间内等距插入 100 个点以获得更细致的曲率变化，同时得到更为平滑的曲线拟合效果。并在特定的横坐标位置（[0.3, 0.4, 0.5, 0.6, 0.7]米）计算各点对应的曲率值。鉴于曲率与传感器位置之间可能存在的非线性关系，我们采用了三次样条曲线插值与线性插值算法来进行对比，为了获得准确性更高的曲率结果，最后采取了效果更优的三次样条曲线插值算法构建数学模型，这种方法能够处理复杂的非线性关系并生成平滑的曲

线，能够得到更为精确的数据以及拟合曲线。

2.2 问题二的分析：

问题二要求根据表 1 中波长测量数据以及问题 1 中所求出的曲率数据来构建数学模型，并且分别重构平面曲线，然后分析曲线的特点，首先我们根据题目中的初始条件已知确定了起始点为坐标原点$(0,0)$，并且设定了初始切线方向与水平方向的夹角为 45 度，利用问题一中使用三次样条插值法，在每 0.006 米处估算所得到曲率值，然后我们使用了两种方法来重构平面曲线，方法一是通过斜率递推算法，方法二是通过采用切角递推算法，从原点开始，根据当前曲率设置步长，计算切线方向的变化量，然后更新当前坐标，再沿着当前切线方向前进一个步长，以确定新点的位置，以此类推，逐步构建出整个曲线上每个点的坐标，由于方法一拟合效果与实际存在较大的差异，于是我们选择了方法二通过切角递推算法来重构平面曲线的模型，将计算得到的连续点坐标存储在一个二维数组中，最后进行结果的可视化，以分析曲线的特点。

2.3 问题三的分析：

问题三给出的函数 $y = x^3 + x(0 \le x \le 1)$ 要求我们根据给出的函数设置适当的采样点，然后计算这些采样点的曲率，接着根据这些采样点的曲率建立数学模型，重新构建平面曲线，重构的曲线与原函数曲线会有一定的差异，所以我们需要分析出现差异的原因。首先求出函数在 x=0 到 x=1 的函数弧长，然后将弧长平等的分为 50 小段，然后使用数值方法计算出每段弧长值的 x 坐标点，然后根据以下公式

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

计算出每个点的曲率，最后我们会得到 50 个 x 轴坐标和对应的曲率。然后我们根据第二题的模型，来重构平面曲线，但是为了使误差更小，对第二题的模型进行优化，在向切线方向延伸一个步长时，根据该点的曲率来优化向前延伸的步长，最后将重构的平面曲线可视化与原函数图像作对比，并分析产生误差的原因。

三、模型假设

1. 假设题目所给数据真实有效；
2. 传感器在光纤上均匀分布，且每个传感器之间的间距固定；
3. 在波长与曲率的关系式中，常数是已知且固定的，不受其他环境因素影响；
4. 曲率在传感器之间是连续的，不存在突变；
5. 测量得到的波长数据是准确的，或者测量误差在可接受范围内，不会对曲率估算结果产生显著影响

四、符号说明

符号	符号说明
x_n	第 n 个点的横坐标
y_n	第 n 个点的纵坐标
Δx	x 轴上的变化量
Δy	y 轴上的变化量
θ_n	第 n 个点的切线与 x 轴正方向上的夹角
s_n	第 n 个点延伸出的圆弧长
Δs	延伸出的步长
k_n	第 n 个点的曲率
β_n	第 n 个点的切线与 x 轴正方向上的夹角

五、模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 光纤传感器曲率估算模型的建立

在问题一中，为了得到更为平滑的曲线，我们分别使用了三次样条曲线插值与线性插值算法进行效果对比，最后选择基于三次样条曲线插值算法来建立光纤传感器曲率估算模型。流程图如**图 1：**

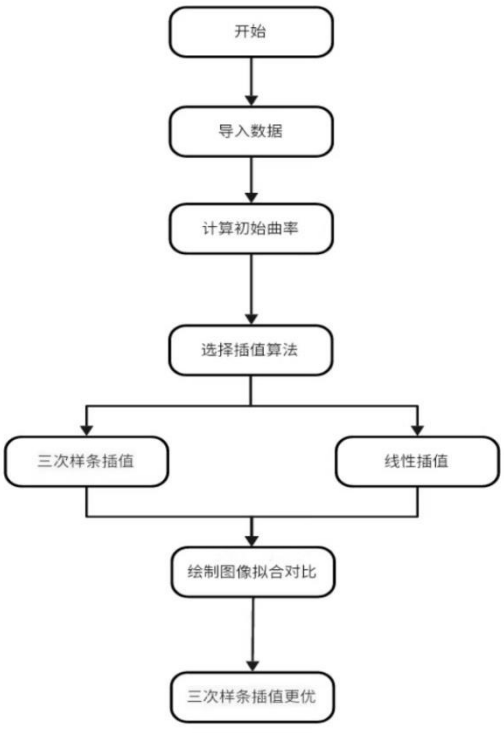

图 1

5.1.1.1 线性插值算法

线性插值算法是一种简单的数值计算分析方法，用于在两个已知的坐标点之间估算未知坐标点的坐标值。这种方法假设在两个已知点之间，数据的变化是线性的，即变化率恒定。

假设我们有两个已知的坐标点(x_0, y_0)和(x_1, y_1)，想要找到在$x_0 < x < x_1$范围内的某个x对于的y值。线性插值的公式如下：

$$y = y_0 + \frac{(y_1 - y_0)}{(x_1 - x_0)}(x - x_0) \tag{1}$$

这个公式的意思是，首先需要计算出两个已知点之间的斜率$\frac{y_1 - y_0}{x_1 - x_0}$，然后使用这个曲率和所求点的$x$的位置来计算对应的$y$值。

5.1.1.2 三次样条插值算法

三次样条插值是一种常用的插值方法，它通过构造一个三次多项式来估计未知数据点的值。三次样条插值的特殊之处在于它不仅保证了插值函数在已知数据点上的连续性，还保证了一阶导数和二阶导数的连续性。这使得插值曲线更加平滑，适用于需要曲线平滑处理的场合。

数学定义：假设我们有一组数据点$(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$，我们希望找到一个函数$S(x)$，它在每个数据点$x_i$上取到对应的$y_i$值。三次样条插值就是寻找一个三次多项式函数，它在每两个相邻数据点之间的区间上都是一个三次多项式。

$S(x)$在区间$[X_{i-1},\ X_i]$上的形式为：

$$S_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3 \tag{2}$$

其中，a_i, b_i, c_i, d_i都是待定系数，它们通过满足以下条件来确定：

1. $S(x_j) = y_j$对所有$j = 1, 2, ..., n$。
2. $S'(x_j)$和$S''(x_j)$在$j = 1, 2, ..., n-1$时连续。
3. $S''(x_1)$和$S(x_n)$的值通常由边界条件所确定。

为了保证连续性，我们引入连续性条件：

$$S_{i-1}^{''}(x_{i-1}) = S_{i}^{''}(x_{i-1}) \quad (3)$$

$$S_{i}^{''}(x_{i}) = S_{i+1}^{''}(x_{i}) \quad (4)$$

通过以上条件使得线性方程组可以通过数值的方法求解。

5.1.1.3 两种算法的求解得出精度高的算法

1）线性插值算法求解:

首先，我们根据给定的波长测量数据计算出传感器位置处的曲率。然后，为了获得更细致的曲率变化以及提高算法模型的准确性，我们在每个 0.6 米的区间内等距插入 100 个点。通过这些点进行线性插值算法的求解以及分别可视化（求解及可视化代码见附录 1）。得到测量值与插入的值的对比图如图 2，图 3：

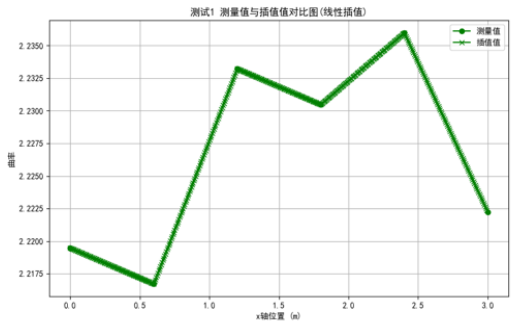

图 2

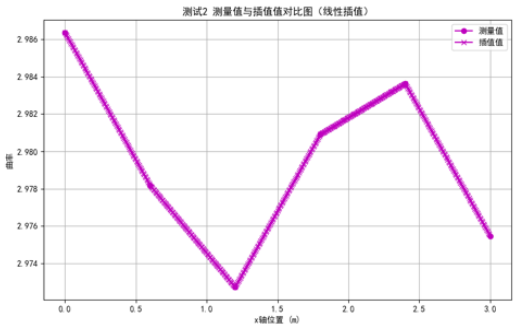

图 3

2）三次样条插值算法求解

同样的，首先，我们根据给定的波长测量数据计算出传感器位置处的曲率。然后，为了获得更细致的曲率变化以及提高算法模型的准确性，我们在每个 0.6 米的区间内等距插入 100 个点。通过这些点进行线性插值算法的求解以及分别可视化（求解及可视化代码见附录 2）得到测量值与插入的值的对比图如图 4，图 5：

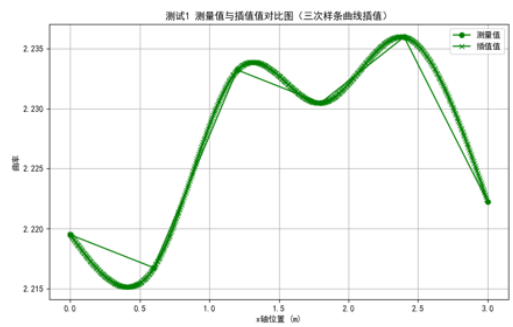

图 4

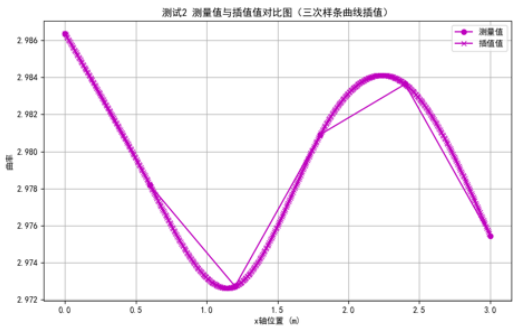

图 5

3）两种算法对比结论：

对于光纤传感器曲率生成的曲线往往不是由直线段组成的，而三次样条曲线插值法

通过三次多项式在数据点之间插值拟合出来的图像平滑过渡，所生成的曲线比线性插值更加平滑，对于计算光纤传感器曲率的应用来说是非常重要的。所以我们认为三次样条曲线插值法拟合的图像更加精确，精度更高。

5.1.2 模型的求解

上面我们通过对比得出三次样条曲线插值法更加精确，所以我们使用三次样条曲线插值算法得出的曲线，找出横坐标在 0.3, 0.4, 0.5, 0.6, 0.7 位置的点，分别得到测试一和测试二这五个点的曲率，并可视化，如图 6，图 7：

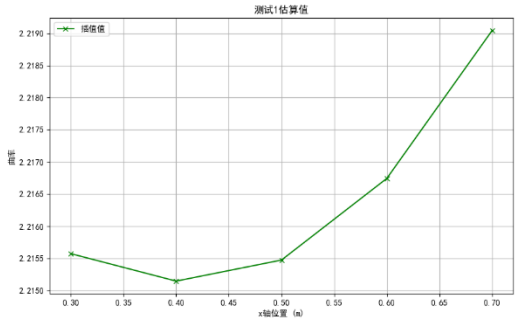

图 6

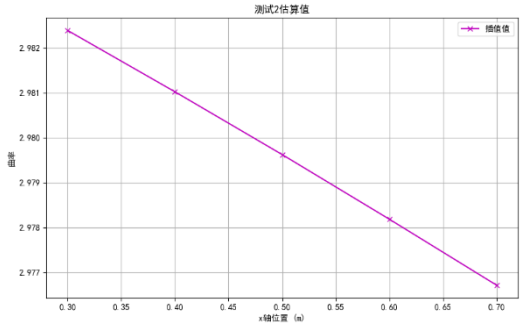

图 7

将其填入表格如表 1：

横坐标 x（米）	0.3	0.4	0.5	0.6	0.7
测试 1 曲率 k	2.2155732	2.2151468	2.2154739	2.2167429	2.2190429
测试 2 曲率 k	2.9823852	2.9810202	2.9796218	2.9781818	2.9767127

表 1

5.1.3 结果分析

通过三次样条插值得到的曲率结果，我们可以分析光纤传感器在不同位置的变形情况。这种插值方法提供了一种平滑且连续的曲线，有助于我们更准确地理解光纤在受力后的形态变化。

结论三次样条插值在本题中的应用展示了其在处理具有复杂曲率变化的数据时的有效性。通过这种方法，我们不仅能够估算出传感器位置处的曲率，还能在任意位置得到连续的曲率估计，这对于光纤传感器的平面曲线重建算法建模至关重要。

5.2 问题二模型的建立与求解

5.2.1 模型的建立

问题二我们选择使用了斜率递推算法和切角递推算法两种算法进行对比，流程图如

下：

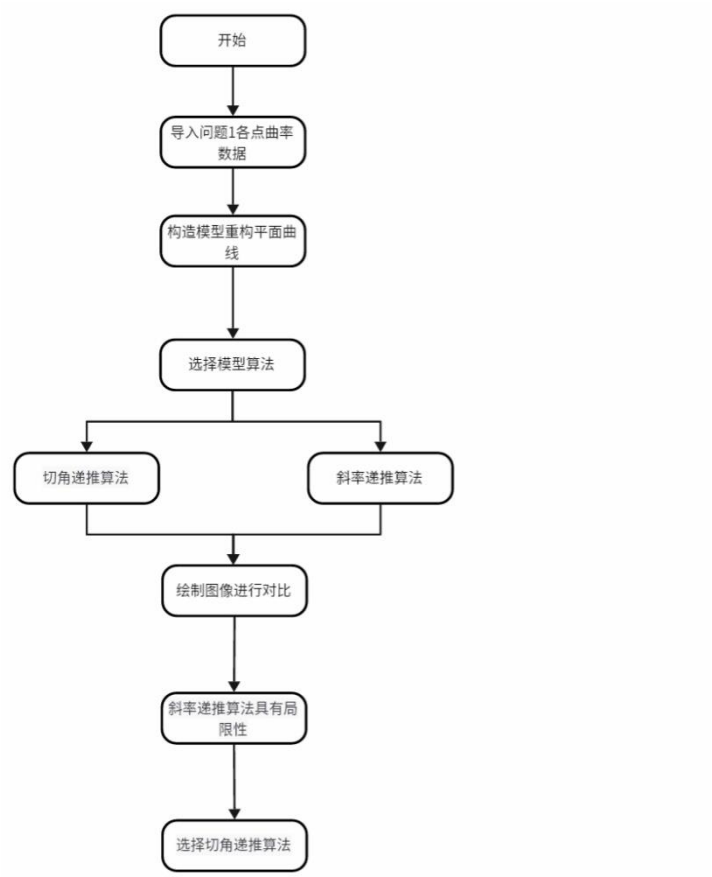

图 8

5.2.1.1 斜率递推算法介绍

斜率递推算法通常基于德卡斯特里奥算法（De Casteljau's algorithm），它是一种递归算法，用于计算贝塞尔曲线上的点。算法的基本思想是将曲线分成更小的部分，然后逐步计算每个点的位置。由于问题一中，我们将每个 0.6 米的区间内等距插入 100 个点，将曲线分成了更小的部分。如图所示，设第 n 、 $n+1$ 点的曲率分别为 κ_n 、 κ_{n+1} ；斜率分别为 k_n 、 k_{n+1} ；坐标分别为 (x_n, y_n) ， (x_{n+1}, y_{n+1}) ；该两点的斜率对 x 轴的夹角分别为： α_n ， α_{n+1} ； $\Delta\alpha_n$ 为两点切向角的变化值； Δs_n 为两点之间的弧长。

几何关系式：

$$\begin{cases} \theta_{\mathrm{n}} = arctg(k_n) \\ \theta_{\mathrm{n+1}} = arctg(k_{n+1}) \\ \Delta\theta_n = \theta_{n+1} - \theta \\ \kappa_n = \dfrac{\Delta\theta_n}{\Delta s_n} \end{cases} \tag{5}$$

由（5）式得：

$$k_{n+1}=tg\left[\kappa_n\Delta s_n+arctg\left(k_n\right)\right] \tag{6}$$

只要给定最初的边界条件，就可以依据（6）式等式递推得出个点的斜率，并在此基础上递推下个点坐标：

$$\begin{cases}\Delta x=\dfrac{\Delta s_n}{\sqrt{1+\mathrm{k}_n^2}}\\ \Delta y=\dfrac{k_n\Delta s_n}{\sqrt{1+\mathrm{k}_n^2}}\end{cases} \tag{7}$$

综合上式得：

$$\begin{cases}x_{n+1}=x_n+\Delta x=x_n+\dfrac{\Delta s_n}{\sqrt{1+k_n^2}}\\ y_{n+1}=y_n+\Delta y=y_n+\dfrac{k_n\Delta s_n}{\sqrt{1+k_n^2}}\end{cases} \tag{8}$$

由上式可得各点坐标值，最后用光滑曲线将插值后的各点连起来，最终得到该算法重构曲线。

5.2.1.2 切角递推算法介绍

切线递推算法（TRA），这是一种用于解决递推序列问题的高效算法。通过利用切线的性质，该算法能够快速计算序列的特定项，尤其适用于那些具有复杂递推关系的序列。对于曲线而言，只要曲线上两点距离足够近，就可以近似认为这一段弧是一段微圆弧，同上所述，我们近似的将每一小段距离看作一段微圆弧。如图—所示，设曲线上一段微圆弧 Δs_n 的起点 o_n 处的曲率为 k_n，终点 o_{n+1} 处的曲率为 k_{n+1}，对应的坐标分别为 $\left(x_n,y_n\right)$、$\left(x_{n+1},y_{n+1}\right)$；$l_n$ 为弧对应的弧长，$\Delta\alpha\left(n\right)$ 为弧段在 o_n 和 o_{n+1} 点处两切线的夹角；β_n 和 β_{n+1} 分别表示 o_n 和 o_{n+1} 处弧段的切线与 x 轴正方向的夹角。原理图如图 9：

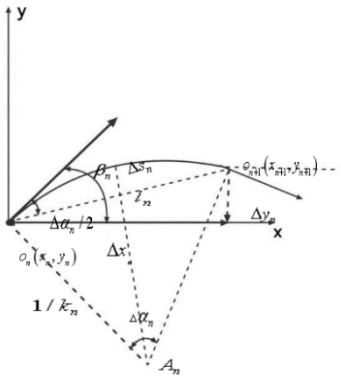

图 9

由几何关系得:

$$\left\{\begin{array}{c}\Delta\alpha_n=\beta_{n+1}-\beta_n\\ l_n=2\sin(\Delta\alpha_n/2)/\kappa_n,(\kappa_n\neq 0)\\ l_n=\Delta s_n,(\kappa=0)\\ \Delta x_n=l_n\cdot\cos(\alpha_n-\Delta\alpha_n/2)\\ \Delta y_n=l_n\cdot\sin(\alpha_n-\Delta\alpha_n/2)\\ x_{n+1}=x_n+\Delta x_n\\ y_{n+1}=y_n+\Delta y_n\end{array}\right\} \tag{9}$$

根据曲率的定义可得：

$$\beta=\int\kappa_n ds \tag{10}$$

根据问题一已知曲率 k 与弧长 s，由（9）（10）式即可得出 β，由此可以逐步构建出整个曲线上每个点的坐标，从而实现曲线的重构。

5.2.2 模型的求解

1）斜率递推算法重构平面曲线

根据斜率递推算法介绍公式，编写 python 程序（见附录 3）依次计算出下一个点的坐标，并将所有坐标可视化，得到如图 10 的重构曲线图：

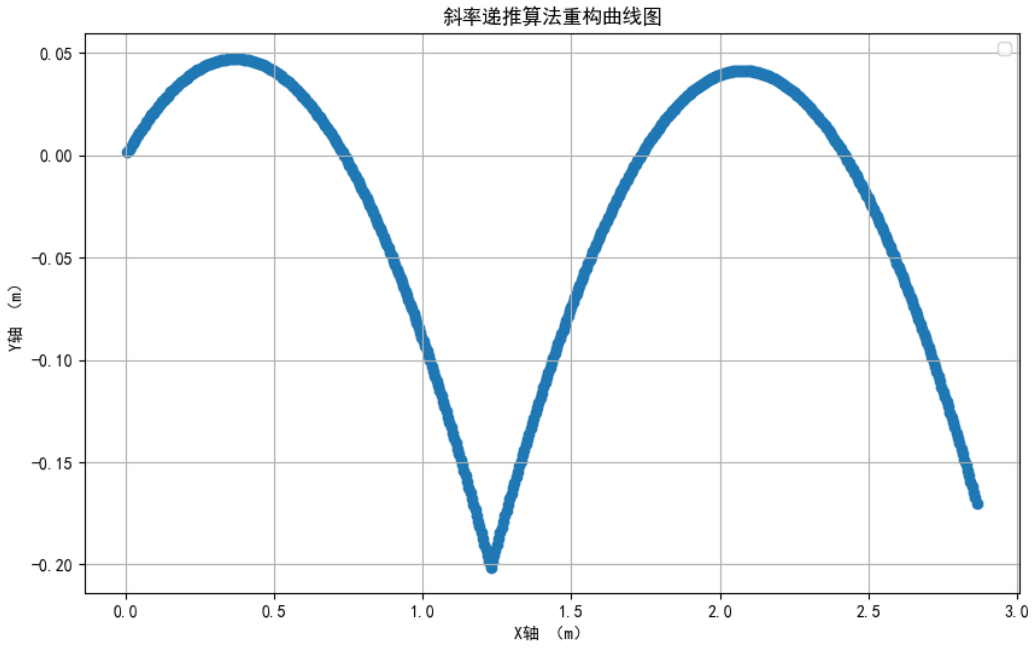

图 10

可以发现当在曲线上某点切线的斜率由负无穷到正无穷时（或者由正无穷到负无穷）时斜率会发生反转，所以此时下一个点的坐标就会出现误差，导致后面所有点均会发生反转，所以这种算法只适用于斜率没有发生反转时的情况。

2）切角递推算法重构平面曲线

根据切角递推算法介绍公式，将起始点设置为坐标原点，因为在原点的切线与 x 轴夹角为 45 度，所以将初始斜率设置为 1，并将第一题得到的曲率带入公式，编写 python 程序（见附录 4）分别依次计算出测试 1 和测试 2 下一个点的坐标，并将测试 1 和测试 2 所有坐标分别可视化，得到如图 11 和图 12 的重构曲线图：

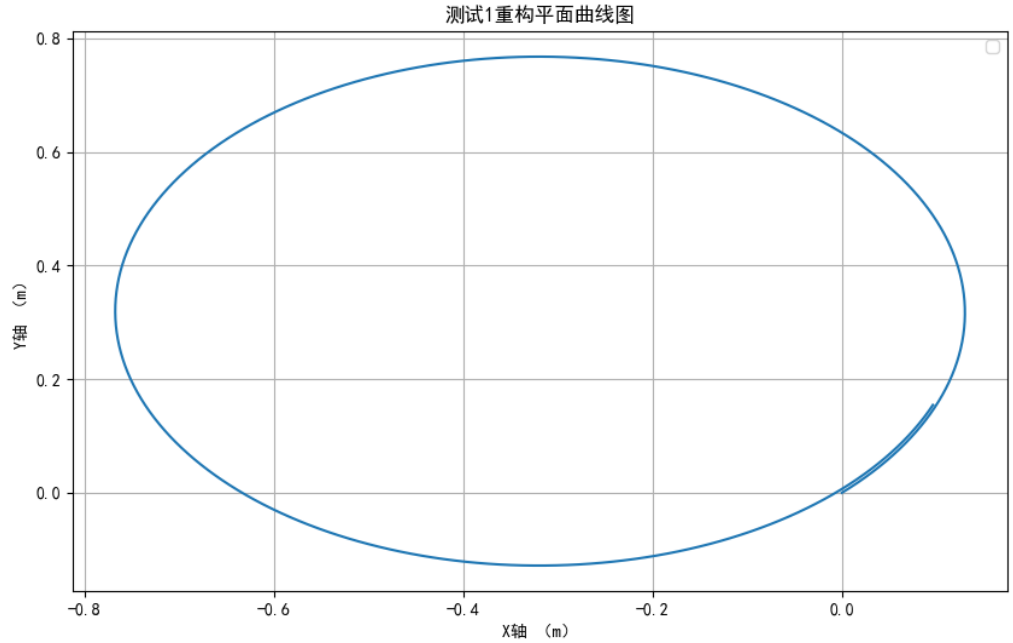

图 11

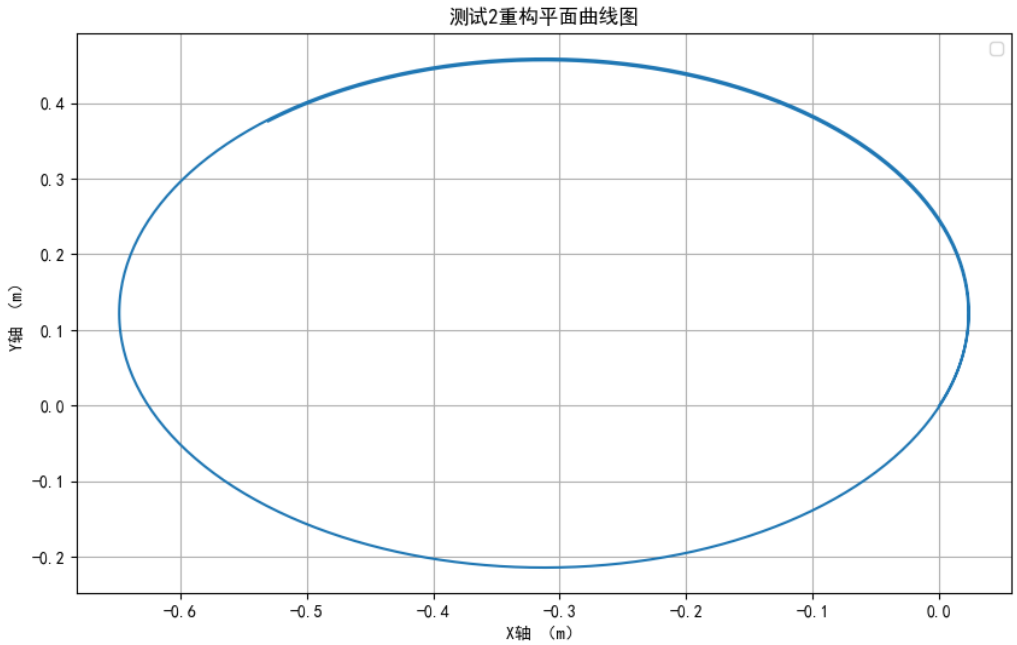

图 12

可以发现此时并没有发生很离谱的斜率反转，我们认为此时重构的平面曲线图最接近原始图像。

然后对曲线特点进行分析如下：

1. 方向改变：曲线在延伸的过程中逐渐改变方向，曲线较为陡峭，该曲线在垂直方向上的变化大于水平方向上的变化。并且曲线覆盖正负 x 值，正负 y 值，表明了曲线方向的变化，这与初始角度设定为 45° 有关。

2. 路径开放：起始点与终点不重合，表示曲线可能是一个开放性的路径，表示结构在受力后发生的非封闭形变。

3. 内缩特征：曲线的内缩倾向表明结构在受力后有向中心靠拢的趋势。

5.2.3 结果分析

通过上述的模型求解，很明显可以看出来斜率递推算法和切角递推算法差别不大，但是这种算法只适用于当斜率没有反转的情况，所以我们在问题二中使用切角递推算法来构建模型。并重构出测试 1 和测试 2 的平面曲线图为首位不相连的近似椭圆形。

5.3 问题三模型的建立与求解

5.3.1 模型的建立与求解

5.3.1.1 计算采样点的曲率

1）计算给定函数在区间（0,1）之间的弧长

弧长的计算公式:

$$s=\int_a^b\sqrt{1+\left(y'\right)^2}dx \tag{11}$$

对于

$$y=x^3+x \tag{12}$$

在（0,1）之间，需要计算：

$$s=\int_0^1\sqrt{1+\left(3x^2+1\right)^2}dx \tag{13}$$

2）将弧长等分成 50 段

$$\Delta s=\frac{s}{50} \tag{14}$$

3）通过 python 的 brentq 方法计算出每段弧长端点对应的 x 轴坐标

4）根据 x 轴坐标计算曲率

$$k = \frac{|y''|}{(1+(y')^2)^{\frac{3}{2}}} \tag{15}$$

5）得到 50 个曲率和 50 个 x 轴坐标对应为 50 个采样点。

5.3.1.2 构建优化切角递推模型并求解

由于原先的切角递推算法中步长是设定的一个固定值，这样就会导致不论曲率大小下一个点的坐标总是会在本次迭代点的基础上加上步长乘以曲率，当曲率点过于密集或者曲率值过大时，会导致重构出来的平面曲线和原本的曲线误差较大，所有我们这里将步长设置为一个动态的值，并且和曲率相关，这里我们将步长和曲率的关系设置为下面的非线性关系：

$$\Delta s = \frac{max}{1.3+k} \tag{15}$$

这里的max为最大步长，即 $\frac{s}{50}$，定值 1.3 是经过优化后得到的最佳值，此时重构的平面曲线和原函数最接近。优化第二题中的代码（见附录 5），并将计算出的采样点的曲率带入模型后所得到重构平面曲线与原函数的对比图如图 13：

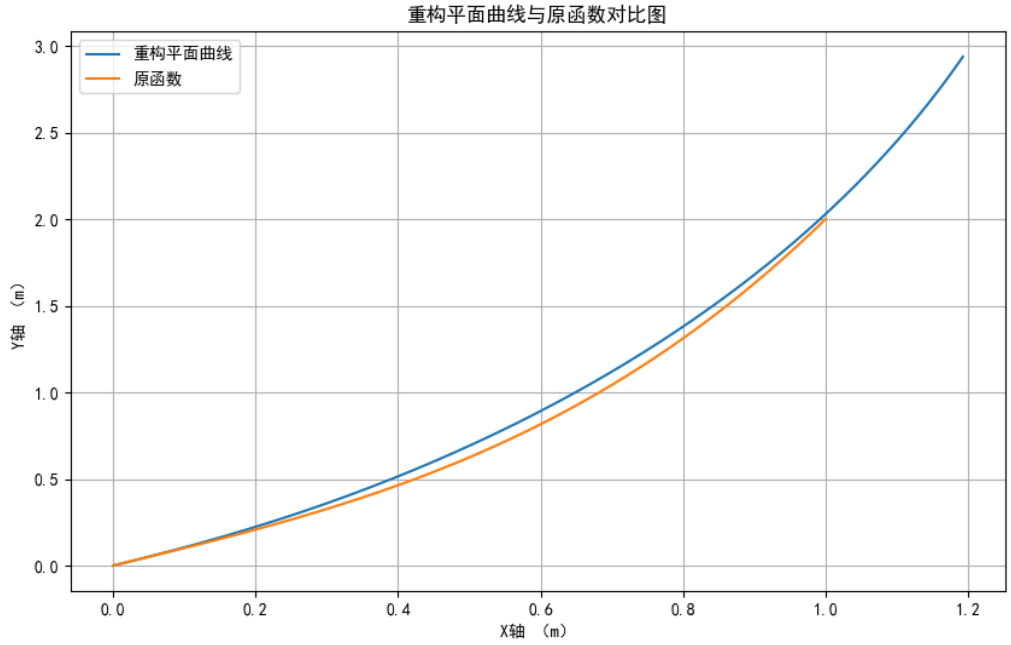

图 13

5.3.2 结果分析

重构曲线与原始曲线产生误差的原因：

1.数值积分的精度：在计算弧长时，使用了数值积分方法（如 quad 函数）。数值

积分只能近似原始积分的精确值，因此会引入误差。

2.步长的确定：在重构曲线的过程中，步长的确定对最终结果的精确度有直接影响。如果步长过大或过小，都可能导致重构的曲线与原函数之间的差异。

3.曲率计算的近似：曲率的计算是基于一阶导数和二阶导数的，如果这些导数的计算存在误差，那么曲率的计算也会受到影响。

4.重构算法的局限性：切角递推算法是一种基于曲率和步长来重构曲线的方法，但它可能无法完全捕捉到原函数的所有细节，特别是当原函数在某些区域变化很快时。

5.舍入误差：在进行大量浮点数运算时，由于计算机的浮点数精度限制，会产生舍入误差。

六、模型的评价与改进

6.1 模型的优点

1.本文中建立的光线传感器曲率估算模型中的三次样条曲线插值算法，因其光滑性、灵活性和准确性等特点，在高平滑度曲线拟合场景中是一种非常有效的工具。光纤受力时弯曲，呈曲线状。该模型具备精度高，拟合效果理想。

2.问题二和问题三通过数值计算和构建基于切角递推算法的模型，适当选择递推步长，可以控制每一步的局部误差，从而使得整个算法的全局误差保持在可接受的范围内，具有适应性强的特点。

6.2 模型的缺点

1.在信息点数量有限的情况下，可能会导致插值结果的准确性和可靠性受到影响。

2.由于算法的收敛性，基于切角递推算法的模型在处理大规模密集数据时可能需要考虑更为高效的算法或者对切角递推算法进行优化。

6.3 模型的改进

本文构建的优化切角递推模型如果仅仅是建立步长与曲率之间的关系，可能只会调整曲率较大的点的步长，但对面更为密集且曲率较小的点时无法得到理想的效果。因此在面对密集的点时我们也需要对步长作出调整，使其在面对密集且曲率较小的点时能得到理想的结果。

七、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]程文胜.基于超弱光纤光栅的曲线重构方法研究[D].三峡大学, 2021.

[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.

附录

附录 1：

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

# 给定的常数 c 和初始波长 λ0
c = 4200
lambda0_1 = 1529
lambda0_2 = 1540

# 传感器位置
sensor_positions = np.array([0, 0.6, 1.2, 1.8, 2.4, 3.0])

# 测试 1 的波长测量数据
lambda1_measured = np.array([1529.808, 1529.807, 1529.813, 1529.812,
1529.814, 1529.809])

# 计算测试 1 的曲率 k
delta_lambda1 = lambda1_measured - lambda0_1
k1 = c * delta_lambda1 / lambda0_1

# 为 test1 建立线性插值函数
```

```
interp_test1 = interp1d(sensor_positions, k1, kind='linear',
bounds_error=False)

# 横坐标的中间点
x_new_test1 = np.array(np.arange(0, 3, 0.006))
k1_interp = interp_test1(x_new_test1)

# 测试 2 的波长测量数据
lambda2_measured = np.array([1541.095, 1541.092, 1541.090, 1541.093,
1541.094, 1541.091])

# 计算测试 2 的曲率 k
delta_lambda2 = lambda2_measured - lambda0_2
k2 = c * delta_lambda2 / lambda0_2

# 对测试 2 使用三次样条曲线插值
interp_test2 = interp1d(sensor_positions, k2, kind='linear',
bounds_error=False)

# 横坐标的中间点
x_new_test2 = np.array(np.arange(0, 3, 0.006))
k2_interp = interp_test2(x_new_test2)

# 设置字体为黑体
matplotlib.rcParams["font.family"] = "Simhei"

# 绘制曲率插值结果
plt.figure(figsize=(10, 6))
plt.plot(sensor_positions, k1, 'go-', label='测量值')
plt.plot(x_new_test1, k1_interp, 'gx-', label='插值值')
plt.legend()
plt.title('测试 1 测量值与插值值对比图(线性插值)')
plt.xlabel('x 轴位置 (m)')
plt.ylabel('曲率')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(sensor_positions, k2, 'mo-', label='测量值')
plt.plot(x_new_test2, k2_interp, 'mx-', label='插值值')
plt.legend()
plt.title('测试 2 测量值与插值值对比图（线性插值）')
plt.xlabel('x 轴位置 (m)')
plt.ylabel('曲率')
plt.grid(True)
```

```
plt.show()

x_new_test1 = np.array([0.3, 0.4, 0.5, 0.6, 0.7])
k1_interp = interp_test1(x_new_test1)
print(k1_interp)
x_new_test2 = np.array([0.3, 0.4, 0.5, 0.6, 0.7])
k2_interp = interp_test2(x_new_test2)
print(k2_interp)
```

附录 2

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

# 给定的常数 c 和初始波长 λ0
c = 4200
lambda0_1 = 1529
lambda0_2 = 1540

# 测试 1 的波长测量数据
lambda1_measured = np.array([1529.808, 1529.807, 1529.813, 1529.812,
1529.814, 1529.809])

# 传感器位置
sensor_positions = np.array([0, 0.6, 1.2, 1.8, 2.4, 3.0])



# 计算测试 1 的曲率 k
delta_lambda1 = lambda1_measured - lambda0_1
k1 = c * delta_lambda1 / lambda0_1

# 拟合横坐标与 k 值的曲线模型
# 由于曲率 k 与位置 x 的关系可能是非线性的，我们使用三次样条曲线插值
cs_test1 = CubicSpline(sensor_positions, k1, bc_type='natural')

# 横坐标的中间点
x_new_test1 = np.array(np.arange(0, 3, 0.006))
k1_interp = cs_test1(x_new_test1)

# 测试 2 的波长测量数据
lambda2_measured = np.array([1541.095, 1541.092, 1541.090, 1541.093,
1541.094, 1541.091])
```

```
# 计算测试 2 的曲率 k
delta_lambda2 = lambda2_measured - lambda0_2
k2 = c * delta_lambda2 / lambda0_2

# 对测试 2 使用三次样条曲线插值
cs_test2 = CubicSpline(sensor_positions, k2, bc_type='natural')

# 横坐标的中间点
x_new_test2 = np.array(np.arange(0, 3, 0.006))
k2_interp = cs_test2(x_new_test2)

# 设置字体为黑体
matplotlib.rcParams["font.family"] = "Simhei"

# 绘制曲率插值结果
plt.figure(figsize=(10, 6))
plt.plot(sensor_positions, k1, 'go-', label='测量值')
plt.plot(x_new_test1, k1_interp, 'gx-', label='插值值')
plt.legend()
plt.title('测试 1 测量值与插值值对比图（三次样条曲线插值）')
plt.xlabel('x 轴位置 (m)')
plt.ylabel('曲率')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(sensor_positions, k2, 'mo-', label='测量值')
plt.plot(x_new_test2, k2_interp, 'mx-', label='插值值')
plt.legend()
plt.title('测试 2 测量值与插值值对比图（三次样条曲线插值）')
plt.xlabel('x 轴位置 (m)')
plt.ylabel('曲率')
plt.grid(True)
plt.show()


x_new_test1 = np.array([0.3, 0.4, 0.5, 0.6, 0.7])

k1_interp = cs_test1(x_new_test1)
print(k1_interp)
x_new_test2 = np.array([0.3, 0.4, 0.5, 0.6, 0.7])

k2_interp = cs_test2(x_new_test2)
print(k2_interp)
```

```
# 绘制曲率插值结果
plt.figure(figsize=(10, 6))
plt.plot(x_new_test1, k1_interp, 'gx-', label='插值值')
plt.legend()
plt.title('测试 1 估算值（三次样条曲线插值）')
plt.xlabel('x 轴位置 (m)')
plt.ylabel('曲率')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(x_new_test2, k2_interp, 'mx-', label='插值值')
plt.legend()
plt.title('测试 2 估算值（三次样条曲线插值）')
plt.xlabel('x 轴位置 (m)')
plt.ylabel('曲率')
plt.grid(True)
plt.show()
```

附录 3

```
import matplotlib
import numpy as np
from scipy.interpolate import CubicSpline
matplotlib.rcParams['axes.unicode_minus']=False

# 给定的常数 c 和初始波长 λ0
c = 4200
lambda0_1 = 1529
lambda0_2 = 1540

# 测试 1 的波长测量数据
lambda1_measured = np.array([1529.808, 1529.807, 1529.813, 1529.812,
1529.814, 1529.809])

# 传感器位置
sensor_positions = np.array([0, 0.6, 1.2, 1.8, 2.4, 3.0])

# 计算测试 1 的曲率 k
delta_lambda1 = lambda1_measured - lambda0_1
k1 = c * delta_lambda1 / lambda0_1

# 拟合横坐标与 k 值的曲线模型
# 由于曲率 k 与位置 x 的关系可能是非线性的，我们使用三次样条曲线插值
```

```
cs_test1 = CubicSpline(sensor_positions, k1, bc_type='natural')

# 横坐标的中间点
x_new_test1 = np.array(np.arange(0, 3, 0.006))
k1_interp = cs_test1(x_new_test1)

# 测试 2 的波长测量数据
lambda2_measured = np.array([1541.095, 1541.092, 1541.090, 1541.093,
1541.094, 1541.091])

# 计算测试 2 的曲率 k
delta_lambda2 = lambda2_measured - lambda0_2
k2 = c * delta_lambda2 / lambda0_2

# 对测试 2 使用三次样条曲线插值
cs_test2 = CubicSpline(sensor_positions, k2, bc_type='natural')

# 横坐标的中间点
x_new_test2 = np.array(np.arange(0, 3, 0.006))
k2_interp = cs_test2(x_new_test2)

import matplotlib.pyplot as plt
import math

# 假设的曲率数据，您需要根据实际测量数据替换这些值
curvatures = k1_interp

xAll = []
yAll = []
# 初始化坐标和角度
x, y = 0, 0
# 初始化斜率
slope = 1
# 一小段弧长的绝对值，您需要根据实际情况选择这个值
delta_s = 0.006

for i in range(len(curvatures) - 1):
    k = curvatures[i]
    R = 1 / abs(k)  # 曲率半径
    s = delta_s  # 弧长（可以根据需要调整）
    theta = s / R  # 圆心角（弧度）

    # 计算圆心坐标
    if slope >= 0:
        center_x = x + R * slope / (math.sqrt(1 + slope ** 2))
```

```
        center_y = y - R / math.sqrt(1 + slope ** 2)
    else:
        center_x = x + R * slope / (math.sqrt(1 + slope ** 2))
        center_y = y - R / math.sqrt(1 + slope ** 2)

    # 计算另一个端点 B 的坐标
    # 根据斜率 k 计算切线与 x 轴夹角
    alpha_1 = math.atan(slope)
    # 根据几何方法计算另一个端点 B 的坐标
    alpha_2 = alpha_1 - theta/2
    if slope >= 0 :
        end_point_x = x + 2 * R * math.sin(theta/2) *
math.cos(alpha_2/math.pi)
        end_point_y = y + 2 * R * math.sin(theta/2) *
math.sin(alpha_2/math.pi)
    else:
        end_point_x = x + 2 * R * math.sin(theta/2) *
math.cos(alpha_2/math.pi)
        end_point_y = y + 2 * R * math.sin(theta/2) *
math.sin(alpha_2/math.pi)

    # 更新坐标
    xAll.append(end_point_x)
    yAll.append(end_point_y)
    x, y = end_point_x, end_point_y
    print(slope)
    # 计算另一个点的斜率
    slope = - (x - center_x) / (y - center_y)

# 设置字体为黑体
matplotlib.rcParams["font.family"] = "Simhei"
# 绘制曲线
plt.figure(figsize=(10, 6))
plt.scatter(xAll, yAll)
plt.legend()
plt.title('斜率递推算法重构曲线图')
plt.xlabel('X 轴 （m）')
plt.ylabel('Y 轴 （m）')
plt.grid(True)
plt.show()
```

附录 4

```
import matplotlib
import numpy as np
from scipy.interpolate import CubicSpline
```

```
import matplotlib.pyplot as plt
matplotlib.rcParams['axes.unicode_minus']=False

# 给定的常数 c 和初始波长 λ0
c = 4200
lambda0_1 = 1529
lambda0_2 = 1540

# 测试 1 的波长测量数据
lambda1_measured = np.array([1529.808, 1529.807, 1529.813, 1529.812,
1529.814, 1529.809])

# 传感器位置
sensor_positions = np.array([0, 0.6, 1.2, 1.8, 2.4, 3.0])

# 计算测试 1 的曲率 k
delta_lambda1 = lambda1_measured - lambda0_1
k1 = c * delta_lambda1 / lambda0_1

# 拟合横坐标与 k 值的曲线模型
# 由于曲率 k 与位置 x 的关系可能是非线性的，我们使用三次样条曲线插值
cs_test1 = CubicSpline(sensor_positions, k1, bc_type='natural')

# 横坐标的中间点
x_new_test1 = np.array(np.arange(0, 3, 0.006))
k1_interp = cs_test1(x_new_test1)

# 测试 2 的波长测量数据
lambda2_measured = np.array([1541.095, 1541.092, 1541.090, 1541.093,
1541.094, 1541.091])

# 计算测试 2 的曲率 k
delta_lambda2 = lambda2_measured - lambda0_2
k2 = c * delta_lambda2 / lambda0_2

# 对测试 2 使用三次样条曲线插值
cs_test2 = CubicSpline(sensor_positions, k2, bc_type='natural')

# 横坐标的中间点
x_new_test2 = np.array(np.arange(0, 3, 0.006))
k2_interp = cs_test2(x_new_test2)

# 曲率数据转存
curvatures_test1 = k1_interp
curvatures_test2 = k2_interp
```

```
# 设置步长
step = 0.006

# 初始化曲线的起始点和第一个切线角度（45 度转换为弧度）
curve_points_test1 = [(0, 0)]
curve_points_test2 = [(0, 0)]
current_angle = np.radians(45)

# 测试一
# 使用切角递推算法根据曲率数据重构曲线
for curvature in curvatures_test1:
    # 计算切线角度的变化
    angle_change = curvature * step

    # 更新切线角度
    current_angle += angle_change

    # 计算新的点的坐标
    x_new = curve_points_test1[-1][0] + step * np.cos(current_angle)
    y_new = curve_points_test1[-1][1] + step * np.sin(current_angle)

    # 添加新的点到曲线点集合中
    curve_points_test1.append((x_new, y_new))

# 测试二
# 使用切角递推算法根据曲率数据重构曲线
for curvature in curvatures_test2:
    # 计算切线角度的变化
    angle_change = curvature * step

    # 更新切线角度
    current_angle += angle_change

    # 计算新的点的坐标
    x_new = curve_points_test2[-1][0] + step * np.cos(current_angle)
    y_new = curve_points_test2[-1][1] + step * np.sin(current_angle)

    # 添加新的点到曲线点集合中
    curve_points_test2.append((x_new, y_new))

# 将曲线点转换为数组以便于绘图
curve_points_test1 = np.array(curve_points_test1)
curve_points_test2 = np.array(curve_points_test2)
```

```
# 设置字体为黑体
matplotlib.rcParams["font.family"] = "Simhei"
# 绘制 test1 重构的曲线
plt.figure(figsize=(10, 6))
plt.plot(curve_points_test1[:, 0], curve_points_test1[:, 1])
plt.legend()
plt.title('测试 1 重构平面曲线图')
plt.xlabel('X 轴 （m）')
plt.ylabel('Y 轴 （m）')
plt.grid(True)
plt.show()

# 绘制 test2 重构的曲线
plt.figure(figsize=(10, 6))
plt.plot(curve_points_test2[:, 0], curve_points_test2[:, 1])
plt.legend()
plt.title('测试 2 重构平面曲线图')
plt.xlabel('X 轴 （m）')
plt.ylabel('Y 轴 （m）')
plt.grid(True)
plt.show()
```

附录 5

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
from scipy.optimize import brentq
import matplotlib


# 定义函数 y = x^3 + x
def f(x):
    return x ** 3 + x


# 定义一阶导数 y'
def f_prime(x):
    return 3 * x ** 2 + 1


# 定义二阶导数 y''
def f_double_prime(x):
    return 6 * x
```

```python
# 定义计算曲率 k
def curvature(x):
    y_prime = f_prime(x)
    y_double_prime = f_double_prime(x)
    return abs(y_double_prime) / ((1 + y_prime ** 2) ** (3 / 2))


# 计算从 x=0 到 x=a 的弧长
def arc_length(a):
    # 使用积分计算弧长
    def integrand(t):
        return (f_prime(t)) ** 2

    return quad(integrand, 0, a)[0]


# 找到使得弧长等分成五十段的 x 坐标点
def find_div_points():
    div_points = []
    total_length = arc_length(1)
    length_per_segment = total_length / 50
    for i in range(1, 51):
        # 使用 brentq 方法找到满足条件的 x 坐标点
        x = brentq(lambda x: arc_length(x) - length_per_segment * i, 0, 1)
        div_points.append(x)
    return div_points


# 计算每个分段点的曲率
def calculate_curvatures(div_points):
    return [curvature(x) for x in div_points]


# 执行计算
div_points = find_div_points()
curvatures = calculate_curvatures(div_points)

# 设置字体为黑体
matplotlib.rcParams["font.family"] = "Simhei"

# 绘制曲率插值结果
plt.figure(figsize=(10, 6))
plt.plot(div_points, curvatures, 'go-', label='曲率')
plt.legend()
```

```
plt.title('等分弧长上的曲率与 x 轴关系图')
plt.xlabel('x 轴位置 (m)')
plt.ylabel('曲率')
plt.grid(True)
plt.show()

# 设置步长
max_step = 0.096

# 初始化曲线的起始点和第一个切线角度（45 度转换为弧度）
curve_points = [(0, 0)]
current_angle = np.radians(45)

# 测试一
# 使用切角递推算法根据曲率数据重构曲线
for curvature in curvatures:
    # 根据曲率调整步长
    step = max_step / (1.3 + curvature)

    # 计算切线角度的变化
    angle_change = curvature * step

    # 更新切线角度
    current_angle += angle_change

    # 计算新的点的坐标
    x_new = curve_points[-1][0] + step * np.cos(current_angle)
    y_new = curve_points[-1][1] + step * np.sin(current_angle)

    # 添加新的点到曲线点集合中
    curve_points.append((x_new, y_new))

# 将曲线点转换为数组以便于绘图
curve_points_test1 = np.array(curve_points)


# 定义函数
def f(x):
    return x ** 3 + x


# 生成 x 的值
x = np.linspace(0, 1, 400)  # 生成从-1 到 2 的 400 个点，用于更平滑的曲线

# 计算对应的 y 值
```

```
y = f(x)

# 设置字体为黑体
matplotlib.rcParams["font.family"] = "Simhei"
# 绘制 test1 重构的曲线
plt.figure(figsize=(10, 6))
plt.plot(curve_points_test1[:, 0], curve_points_test1[:, 1], label='重构平面曲线')
plt.plot(x, y, label='原函数')
plt.legend()
plt.title('重构平面曲线与原函数对比图')
plt.xlabel('X 轴 （m）')
plt.ylabel('Y 轴 （m）')
plt.grid(True)
plt.show()
```