基于三次样条插值和累积曲率的积分重构方法研究

摘要

本研究聚焦于基于光纤传感器的平面曲线重建算法建模，旨在通过光纤传感器技术实时获取结构应变信息，进而重构结构的形变或位移。光纤传感器因其轻质、体积小、弯曲性能好、抗电磁干扰能力强等优点，在多个领域具有实际应用价值。

针对问题一，根据表一提供的数据及题目给出的曲率通式k计算出（FBG1~FBG6）的曲率，其中FBG1-FBG6两两间隔0.6m，当初始状态的波长为1529（简记测试一）时，对应的曲率k分别为2.2195，2.2167，2.2332，2.2305，2.2360，2.2222 ；当初始状态为1540（简记测试二）时,相同x所对应曲率为2.9864，2.9782，2.9727，2.9809，2.9836，2.9755，进而根据所求出来的曲率和已知x来构建弧长len_{arc}与曲率k的关系式，将给定x=0.3 ，0.4， 0.5， 0.6， 0.7转化为弧长len_{arc}进而解出在测试一的情况下，所对应的曲率k分别为2.2129，2.2158，2.2200，2.2247，2.2290；在测试二的情况下，所对应的曲率k分别为2.9812，2.9788 2.9764，2.9745，2.9731。

针对问题二，由问题一测出的曲率，可以拟合出一个曲率k和x曲线，进而根据曲率k在平面直角坐标系中的基本定义，在给定初始值，即与水平方向的夹角为45度，通过数值积分和弧长参数化的方法求解出6个测量点在二维坐标系下的具体坐标,并对这些点进行三次样条插值重构出曲线的形状近似椭圆。由于初始值是与水平方向的夹角为45度，即夹角应该有四种情况，$\theta=\frac{\pi}{4},\frac{3\pi}{4},-\frac{3\pi}{4},-\frac{\pi}{4}$分别对四种情况进行误差分析，求均方根RMSE，测试一中，指标分析依次为 5.15，0.49，5.18，0.49；测试二中，指标分析依次为0.15，0.31，0.31，0.15。从误差分析中可以看出在测试一中$\theta=\frac{3\pi}{4}$和$\theta=-\frac{\pi}{4}$时，构建出来的曲线更拟合；在测试二中$\theta=\frac{\pi}{4}$和$\theta=-\frac{\pi}{4}$时，构建出来的曲线更拟合，对坐标系进行升维处理后，构建出了一个光滑且单一的螺旋上升曲线。

针对问题三，旨在针对给定曲线，基于原始平面曲线定义的公式计算曲率，并在等弧长采样点（s=0.2）上进行采样。通过曲率半径和曲率的关系以及圆心角 θ和弧长s与曲率k的关系，采用累积曲率方法构建多项式拟合模型。拟合曲线方程的建立为研究重点，随后对重构曲线与原始曲线进行误差分析，包括均方误差（MSE）、均方根误差（RMSE）、平均绝对误差（MAE）和 R^2系数。实验结果表明，计算得到的误差分别为 0.0003、0.0159、0.0125、0.9990。误差分析结果清晰显示，通过累积曲率方法实现的重构更为准确和可靠。

关键词：三次样条插值；数值积分；微分方程；曲率；累积曲率；误差分析

目录

一、问题重述

光纤传感技术是一种新型的传感器技术，利用光波和光纤来感知外界环境的信号变化。通过测量光纤中光波的参数变化，可以实时获得结构的应变信息，从而重构结构的形变或位移。在这项研究中，我们通过解调光纤传感器获得的波长数据，估算平面光栅各个传感点的曲率，并利用这些曲率数据构建数学模型进行平面曲线的重构。

问题 1：根据给定的光纤传感器波长测量数据，构建数学模型以估算平面光栅各传感点（FBG1-FBG6）的曲率。假设初始状态的切线与水平方向的夹角为 45 度，求出在指定横坐标位置处（0.3 米、0.4 米、0.5 米、0.6 米、0.7 米）的曲率。

问题 2：利用问题 1 中求得的曲率及给定的波长测量数据，构建数学模型分别重构平面曲线，并对重构曲线的特点进行分析。

问题 3：根据给定的平面曲线方程 $y=x^3+x\left(0\le x\le 1\right)$ 以适当的等间距弧长采样，计算采样点的曲率。然后利用采样曲率构建数学模型，重构平面曲线，并分析重构曲线与原始曲线之间的误差产生原因。

二、问题分析

通过已知公式求曲率，并由各曲率值来求曲率函数，可以用插值法和多项式拟合法，由于多项式拟合可能会导致过拟合，且多项式拟合生成的拟合曲线可能在数据点之间产生不连续性或者导数不连续的情况，导致生成的拟合曲线不稳定，由公式求出的曲率点即已知的数据点，用插值法会确保生成的曲线通过所有给定的数据点，保证在数据点处的逼近性很高，因此选用插值法拟合的曲率函数会更精确。插值法的选取也很重要，相对于线性插值法和拉格朗日插值法，三次样条插值法产生的曲线通常比其他方法更加光滑，可以减少插值曲线的振荡现象，特别是在数据点之间有明显变化的情况下，并且还能防止龙格现象，选用三次样条插值法拟合的曲率函数会更精确。

问题 1

针对问题 1，需要先用通式$k=\frac{c(\lambda-\lambda_0)}{\lambda_0}$求出 FBG1-FBG6 的曲率，构建弧长$len_{arc}$与曲率 k 的关系式，用弧长与$len_{arc}$的关系$len_{arc}=\frac{x}{\cos\left(\frac{\pi}{4}\right)}$求出 x=0.3 米、0.4 米、0.5 米、0.6 米 、0.7 米的弧长值，通过三次样条插值估算出 x 对应的曲率 k。

问题 2

针对问题 2，首先通过问题 1 求出的曲率以及等距离的弧长使用三次样条插值法来构造曲率函数，然后把弧长参数化，再把曲率函数进行积分得到对应的参数坐标 x(s)，y(s)，θ(s)，转换成直角坐标后再次使用三次样条插值得到近似的拟合曲线。

问题 3

针对问题 3，首先我们需要确定等弧长的间距，以确保采样点在指定范围内均匀分布。然后利用弧长和横坐标的转换求得采样点的横坐标值，根据曲率公式计算采样点的曲率；接下来根据已知等间距弧长处的曲率值和设定的初始条件，建立累积曲率模型来重构曲面方程。

三、模型准备

模型假设

1、假设弧长近似为直线

2、假设曲线在每一段都可以近似为圆弧

3、假设插值节点是均匀分布的

4、假设曲线的两端点边界条件为自然边界条件（即曲线的二阶导数在端点处为零）

5、假设在每个由两个相邻数据点确定的子区间上，可以找到一个三次多项式来近似数据。

6、假设插值函数存在于某个函数空间中，这个空间由满足边界条件的所有三次多项式组成。

7、假设由曲率求得的 y 是 y_{true}

符号说明

符号	说明
t	参数化曲线的参数
$s(x_i)$	表示点x_i处的弧长
k	曲率
λ	波长
θ	曲线在s处的切线角度
a_i, b_i, c_i, d_i	均为几个待定系数
M_i	泰勒展开式的系数
len_{arc}	曲线的弧长
x_{end}	x的取值范围的末端（本题为 1）
x_{last}	上一个采样点的x坐标值
num_{points}	采样点的总数量
len_{points}	已加入的采样点个数
$step$	步长
x_{guess}	迭代中猜测点的x坐标值

MSE	均方误差
R^2	衡量模型拟合优度的指标
$RMSE$	均方根误差
MAE	平均绝对误差
y_{pred}	y 的预测值
y_{true}	y 的真实值

四、模型的建立与求解

问题 1 的分析与求解

1 计算表 1 给出的测量点的曲率

根据给出的波长λ与曲线曲率(x,y)之间的关系近似公式

$$k=\frac{c(\lambda-\lambda_0)}{\lambda_0} \tag{1}$$

代入表 1 的两个测试的受力前后的数据，且c=4200，　　求得结果如下：

表 1 测量点的曲率值

测量点	FBG1	FBG2	FBG3	FBG4	FBG5	FBG6
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

2 基于三次样条插值和弧长与水平方向的转换估计相应位置的曲率

2.1 定义初始状态

以测量点 FBG1 处为原点，曲线在该点的切线方向与水平位置的夹角为 45°

2.2 横坐标位置与弧长的转换

通过几何关系可以计算出特定的 x 坐标值（水平方向的位置），其在曲线上对应的弧长。这里我们使用了：

$$len_{arc} = \frac{x}{\cos\left(\frac{\pi}{4}\right)} \tag{2}$$

把表格中横坐标 x =0.3, 0.4, 0.5, 0.6, 0.7(米)位置到原点的弧长求出来，用已知弧长位置的点通过三次样条插值法，估算出上面所求的 x 位置的对应曲率。

表 2 x 对应的曲率值

横坐标 x（米）	0.3	0.4	0.5	0.6	0.7
测试 1 曲率 k	2.2129	2.2158	2.2200	2.2247	2.2290
测试 2 曲率 k	2.9812	2.9788	2.9764	2.9745	2.9731

3 弧长与曲率函数关系可视化

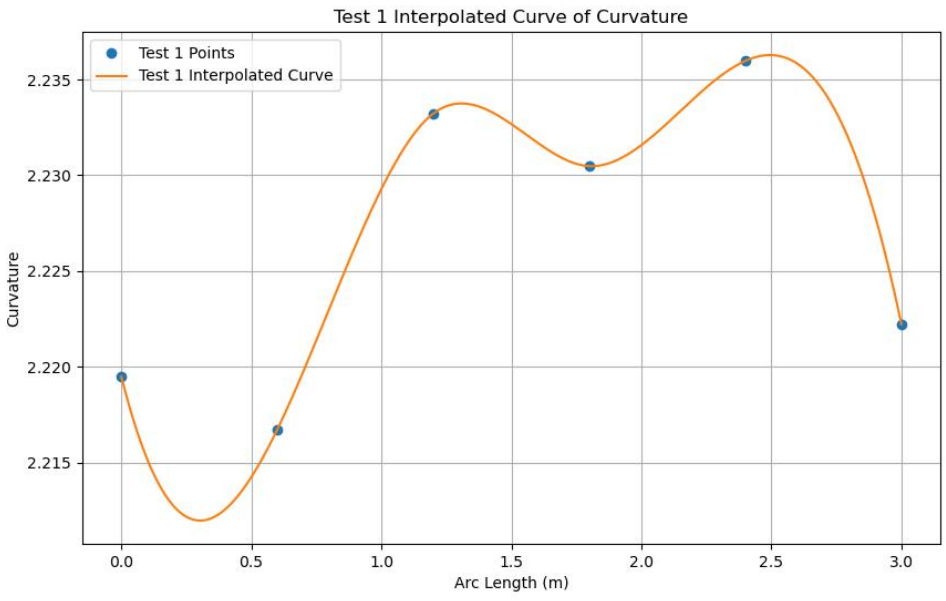


图 1 测试一弧长 x 与曲率 k 的拟合曲线

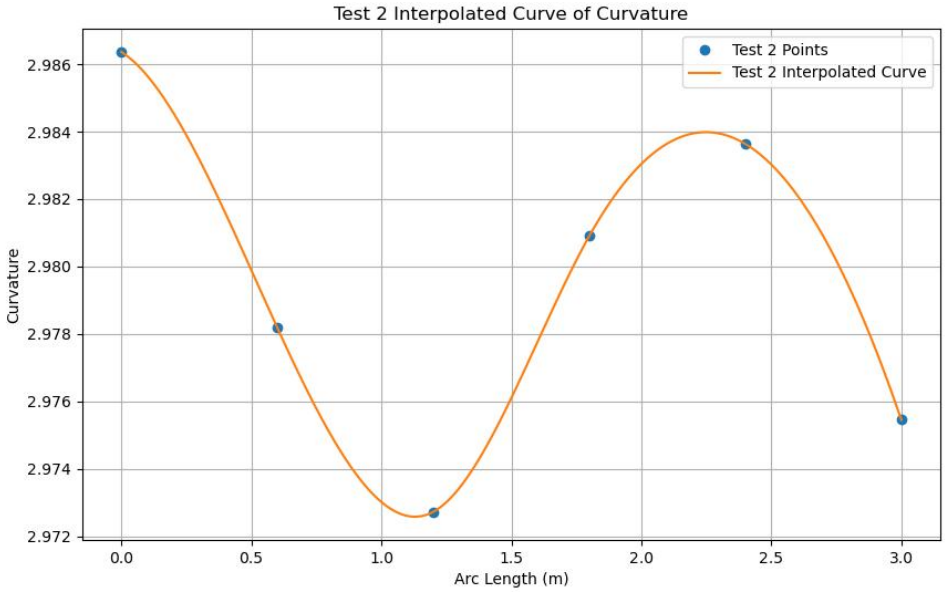


图 2 测试二弧长 x 与曲率 k 的拟合曲线

问题 2 的分析与求解

要求根据表 1 波长测量数据和问题 1 求出的曲率，构建数学模型，并分别重构平面曲线，需要先根据给出的曲率以及等距离的弧长通过三次样条插值法来构造曲率函数，然后把弧长参数化，再把曲率函数进行积分得到对应的参数坐标 $x(s),y(s),\theta(s)$，转换成直角坐标后再次使用三次样条插值得到近似的拟合曲线（以下步骤均以测试一为例）。

1 使用三次样条插值构造曲率函数

函数 $S(x_i)=f(x_i)$，$i=0,1,\cdots,\mathrm{n}$ 满足下列条件：

(1)在每一个子区间 $[x_{i-1},x_i](i=1,2,\cdots,n)$ 上，$f(x)$ 是一个不超过三次的多项式。

(2)在每一个结点上满足 $\mathrm{S}(x_i)=f,i=0,1,2,\cdots,\mathrm{n}$ 。

(3) $S(x_i)$ 在开区间 (a,b) 上为二次连续可导函数。

则称 $S(x_i)=f(x_i)$ 为区间 $[a,b]$ 对应划分 $[x_{i-1},x_i]$ 的三次样条插值。

设三次样条函数在每个子区间$[x_{i-1},x_i]$上有表达式

$$S(x)=S(x_i)=a_ix^3+b_ix^2+c_ix+d_i \quad x\in(x_{i-1},x_i),i=1,2,\cdots,n \tag{3}$$

利用三弯矩法令$S''(x_i)=M_i\,(i=0,1,2,\cdots,n)$。因为$S(x)$在 $[x_{i-1},x_i]$上是三次多项式

故有

$$S'''(x)=\frac{M_{i+1}-M_i}{x_{i+1}-x_i} \qquad \forall x\in[x_{i-1},x_i]$$

为第二边界条件$S''(x_0)=f''(x_0)=M_0,S''(x_n)=f''(x_n)=M_n$， 得

$$\begin{cases}2M_1+\lambda_1M_2=6f[x_o,x_1,x_2]-\mu_1M_0\\ \mu_iM_{i-1}+2M_i+\lambda_iM_{i+1}=6f[x_{i-1},x_i,x_{i+1}] \quad (i=2,3\cdots,n-2)\\ \mu_{n-1}M_{n-2}+2M_{n-1}=6f[x_{n-2},x_{n-1},x_n]-\lambda_{n-1}M_n\end{cases}$$

由方程组可表示为矩阵方程如下：

$$\begin{pmatrix}2 & \lambda_1 & & & \\ \mu_2 & 2 & \lambda_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mu_{n-2} & 2 & \lambda_{n-2}\\ & & & \mu_{n-1} & 2\end{pmatrix}\begin{pmatrix}M_1\\ M_2\\ \vdots\\ M_{n-2}\\ M_{n-1}\end{pmatrix}=\begin{pmatrix}6f[x_o,x_1,x_2]-\mu_1f''(x_0)\\ 6f[x_1,x_2,x_3]\\ \vdots\\ 6f[x_{n-3},x_{n-2},x_{n-1}]\\ 6f[x_{n-2},x_{n-1},x_n]-\lambda_{n-1}f''(x_n)\end{pmatrix}$$

最后使用三次样条插值得到连续的曲率函数，并允许外插

2 弧长参数化

弧长$P(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$表示从曲线起点到某点的距离，对于参数$t$的曲线$r(t)$，弧长的表达式为：$s(t)=\int_{t0}^{t}|r'(u)|du$

在曲线上某点t的切线方向的单位向量$\mathbf{T}(t)$可以由曲线的导数标准化得到：

$$\mathbf{T}(t)=\frac{r'(t)}{|r'(t)|}$$

曲线在s方向上的变化可以表示为：$\frac{dr}{ds}=\mathbf{T}(t)$

由于$\mathbf{T}(t)$是单位向量，所以：$\frac{dr}{ds}=\cos(\theta)\mathbf{i}+\sin(\theta)\mathbf{j}$

其中θ是曲线在s处的切线角度。

3 通过数值积分求坐标 x, y

由曲率的公式

$$k(t)=\frac{\left|r'(t)\times r''(t)\right|}{\left|r'(t)\right|^3}=\frac{\left|x'(t)y''(t)-x''(t)y'(t)\right|}{\left(x'(t)^2+y'(t)^2\right)^{\frac{3}{2}}}$$

进行数值积分可得到$x(t),y(t)$的参数坐标，再转化为坐标可得到每一个曲率对应的坐标（以测试一角度为45°为例）：

表 3 曲率值对应的坐标

曲率	x	y
2.219490	0.000000	0.000000
2.216743	0.067273	0.552799
2.233224	-0.453266	0.749278
2.230477	-0.764126	0.288132
2.235971	-0.386709	-0.120319
2.222237	0.097051	0.153961

4 用三次样条插值求近似曲线图像

使用 3 求出来的点的坐标，基于三次样条插值曲线重构的数学模型，在坐标上拟合出图像，题目假设初始点坐标为原点，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°，可知初始方向有 4 种情况，分别是$\frac{\pi}{4}$，$\frac{3\pi}{4}$，$-\frac{3\pi}{4}$，$-\frac{\pi}{4}$ 计算了四个方向x与y的值，构造了四个方向上的曲线，图像如下：

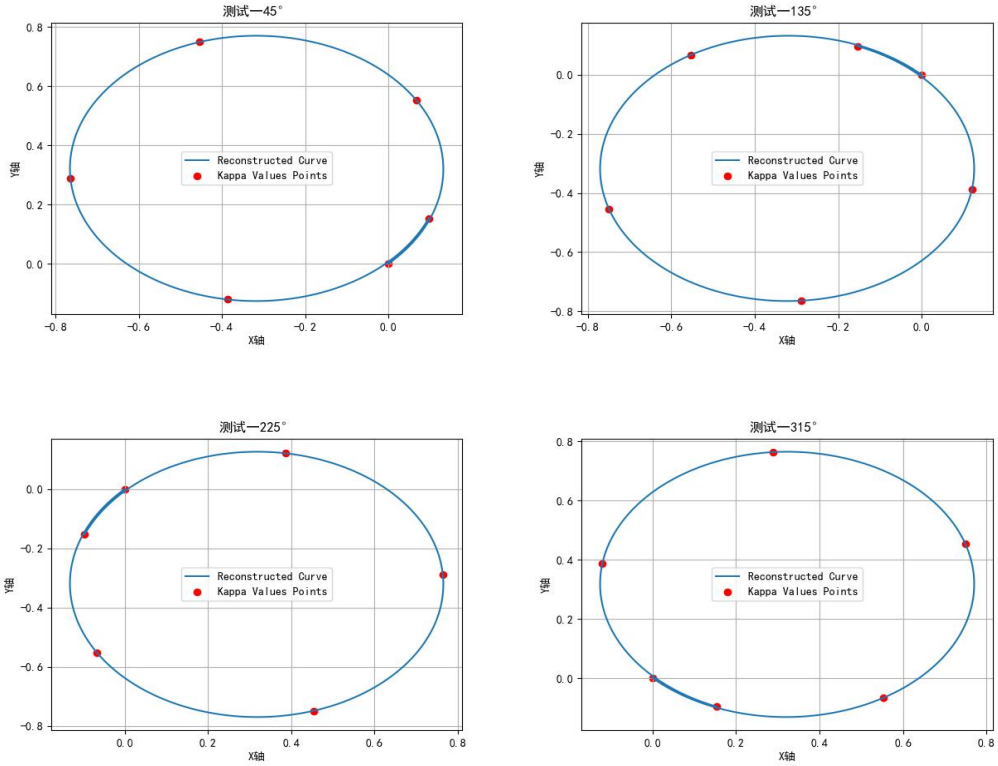


图 3 测试一重构曲线

同理可得出测试二的重构曲线图像:

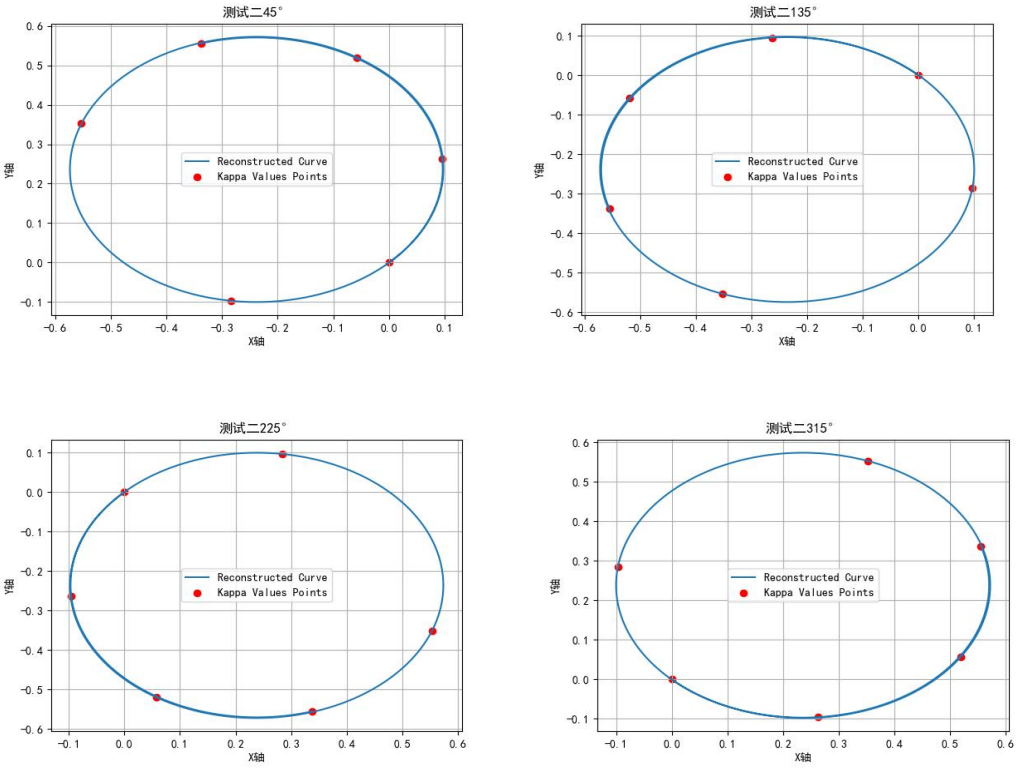


图 4 测试二重构曲线

5 误差分析

由于没有原始曲线，所以没有办法很精确的估计重构曲线的拟合程度，但是可以根据已知的曲率通过参数方程可求得y_{true}，得到重构的曲线，将曲率对应 x 值反代回重构曲线，得到y_{pred}，衡量指标采用均方根误差。

均方根误差公式：

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_{true,i} - y_{pred,i}\right)^2}$$

RMSE 越小，表示模型的预测值与实际观测值之间的差异越小，模型的预测性能越好。相反，RMSE 越大，表示预测误差越大，模型的性能越差。

可以得出测试 1 和测试 2 的均方根误差：

表 4 测试一各度数对应的误差

度数	*RMSE*
$\frac{\pi}{4}$	5.15
$\frac{3\pi}{4}$	0.49
$-\frac{3\pi}{4}$	5.18
$-\frac{\pi}{4}$	0.49

表 5 测试二各度数对应的误差

度数	*RMSE*
$\frac{\pi}{4}$	0.15
$\frac{3\pi}{4}$	0.31
$-\frac{3\pi}{4}$	0.31
$-\frac{\pi}{4}$	0.15

6 优化重构曲线

1.建立三维坐标轴

从图 4 可以看出，重构曲线形状类似椭圆，但是曲线有重合的部分，所以二维坐标系不能很好的可视化该曲线，尝试采用三维坐标系进行可视化，重构曲线在三维坐标系上是光滑且单一的，可视化该曲线：

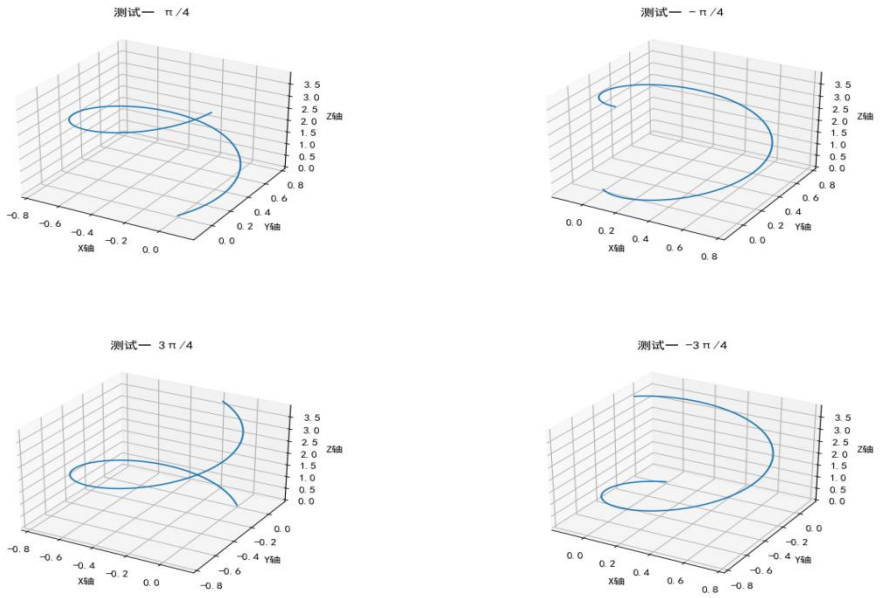
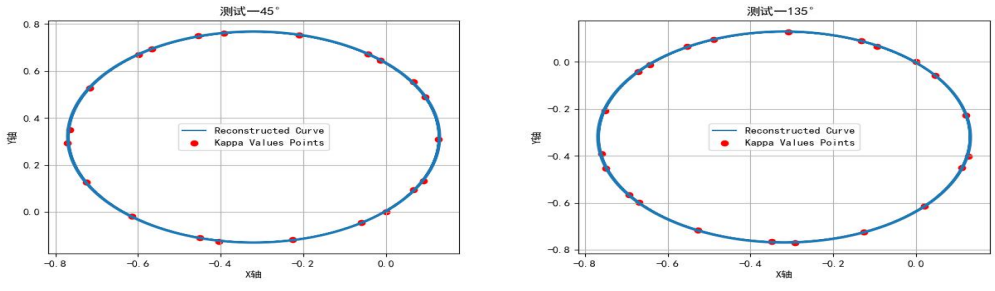


图 5 三维坐标系下测试一重构曲线

2. 选取 20 个坐标点进行拟合，得到重构曲线:



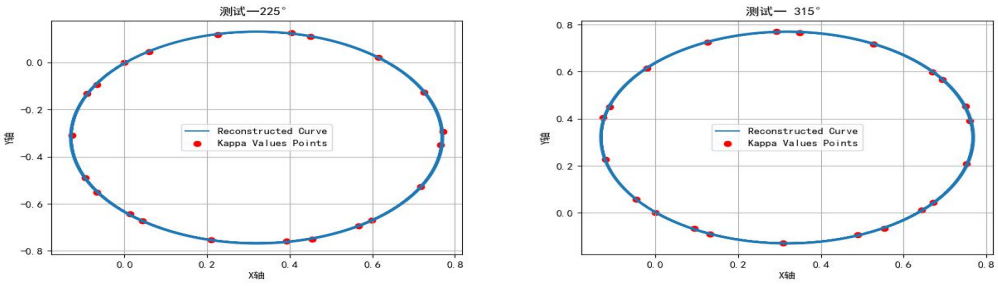


图 6 多样本点测试一重构曲线

在三维坐标下进行拟合，得到重构曲线:

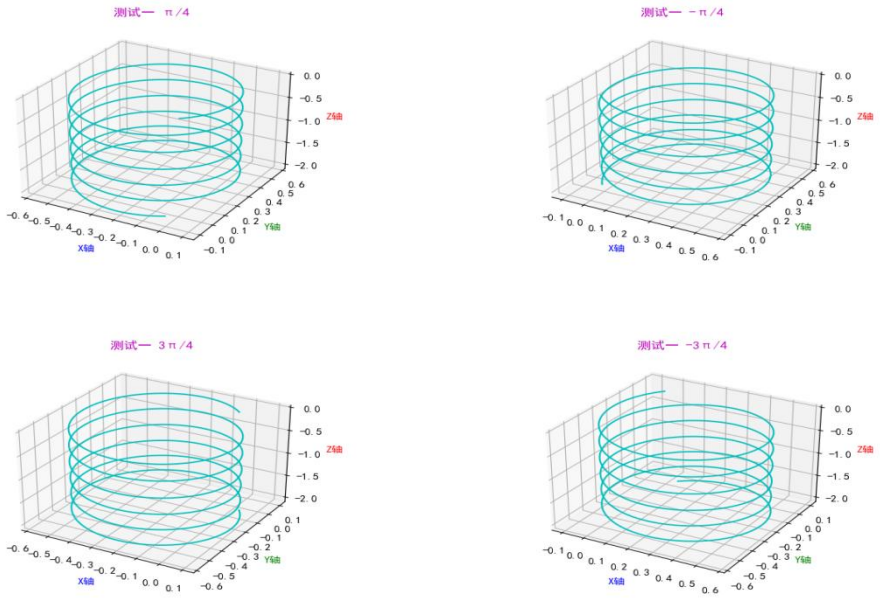


图 7 三维坐标系下多样本点测试一重构曲线

7 分析重构曲线的特点（以测试一$\frac{\pi}{4}$为例）

7.1 视觉上，这个曲线在三维坐标下是一个光滑且单一的螺旋上升曲线，当投影到二维坐标上时，类似于一个椭圆。

7.2 对称性，从图像中看，曲线本身似乎不具有传统的对称性，但是它的产生过程中可能有一个旋转对称性，沿着它的中心轴旋转曲线，曲线看起来会保持不变。

7.3 曲率，在这个图像中，曲线是均匀螺旋形状，且题目 1 各个点给出的曲率和插值求出来的曲率，可以推测曲率在整个曲线上是相差不大的，这意味着曲

线的每个部分都以相同的方式弯曲。

7.4 周期性，在这个螺旋线中，每完成一个完整的旋转就标志着一个周期的结束，并开始下一个周期。

问题 3 的分析与求解

1 通过牛顿法（使用 *fsolve* 函数）计算采样点的曲率

1.1 采样点的确定

根据题目要求在平面曲线方程 $y=x^3+x\left(0\le x\le 1\right)$ 上以适当的等间距弧长采样，考虑到 x 的取值范围和计算时间复杂度，这里设置等弧长 $s=0.2$，从原点开始在 $0\le x\le 1$ 上等间距弧长取 10 个采样点。

1.2 数学基础公式

对于函数 $y=f\left(x\right)$，曲率 k 的计算公式为：

$$k=\frac{\left|f''\left(x\right)\right|}{\left(1+\left(f'\left(x\right)\right)^2\right)^{3/2}}$$

对于给定函数 $y=f\left(x\right)$，弧长 s 在区间 $\left[a,b\right]$ 上可以通过以下公式计算：

$$s=\int_a^b\sqrt{1+\left(\frac{dy}{dx}\right)^2}dx$$

1.3 迭代计算逐步添加采样点

以第二个采样点的确定为例：

step1 先给定一个步长，这里用$\frac{1-0}{10-0}$作为开始的步长，迭代通式如下：

$$\frac{x_{end}-x_{last}}{num_{points}-len_{points}}$$

step2 计算x_{guess}（猜测点），计算公式是$x_{last}+step$

step3 计算出猜测点到原点（上一个采样点）的弧长距离

step4 如果弧长距离太短就增加步长（$2*step$），距离太长就减少步长（$step$ /2）

step5 使用 fsolve 函数找到精确的 x 值，使得弧长接近目标值

以此类推找到每一个采样点的 x 值，直到找到 10 个采样点。

1.4 根据曲率公式和 10 个采样点的 x 值，计算出这些点的曲率值

结果展示：

表 6 采样点 x 与曲率 k 的对应值

x	0.0000	0.1400	0.2724	0.3923	0.4984	0.5918	0.6744	0.7482	0.8147	0.8754
k	0.0000	0.2720	0.4148	0.4237	0.3675	0.2990	0.2392	0.1919	0.1558	0.1282

2 基于累积曲率方法重构曲线方程

该问题的求解是已知等弧长处曲线的曲率值，重构曲线方程。曲率描述了曲线在某一点的弯曲程度，而曲线上点的坐标则是由曲线的形状和长度共同决定的。所以我们根据已知的等弧长条件和曲率值，就可以得到点的坐标，从而根据多项式拟合来重构曲线方程。

2.1 设置初始条件：

等弧长间距 $s=0.2$；

第一个采样点为原点 $(0,0)$；

曲线在原点的切线方向与水平方向的夹角为 45°：

进行简单的求导计算，得出原方程 $y=x^3+x\left(0\le x\le 1\right)$ 在原点处的切线方向与水平方向的夹角为 45°

为了减少求出曲线方程与原方程的误差，设置待求曲线在原点的切线方向与水平方向的夹角与原方程一致

2.2 通过累积曲率方法根据前一个采样点迭代计算后一个采样点

以第二个坐标点的计算为例：

由于弧长$s=0.2$，且已知曲率，可以使用曲率半径公式来计算这一小段曲线的近似形状。曲率半径R与曲率k的关系是

$$R=\frac{1}{k}$$

假设这一小段曲线可以近似为圆弧，可以计算出这一圆弧对应的圆心角θ：

$$\theta=\left(\text{弧长}/\text{曲率半径}\right)=s*k\text{。}$$

有了圆心角和切线方向，就可以计算出第二个点的坐标。圆心位于切线的法线方向上，距离原点为曲率半径R。第二个点的坐标可以通过在切线方向上移动弧长s的距离来得到。

$$\begin{cases}x=x_0+s*\cos\left(\theta\right)\\y=y_0+s*\sin\left(\theta\right)\end{cases}$$

迭代计算后续点的坐标：

对于后续的每个点，都以前一个点为起点，以前一个点的切线方向为基础，重复上述步骤来计算新的坐标。

2.3 多项式拟合构建曲线方程与原曲线方程对比结果展示

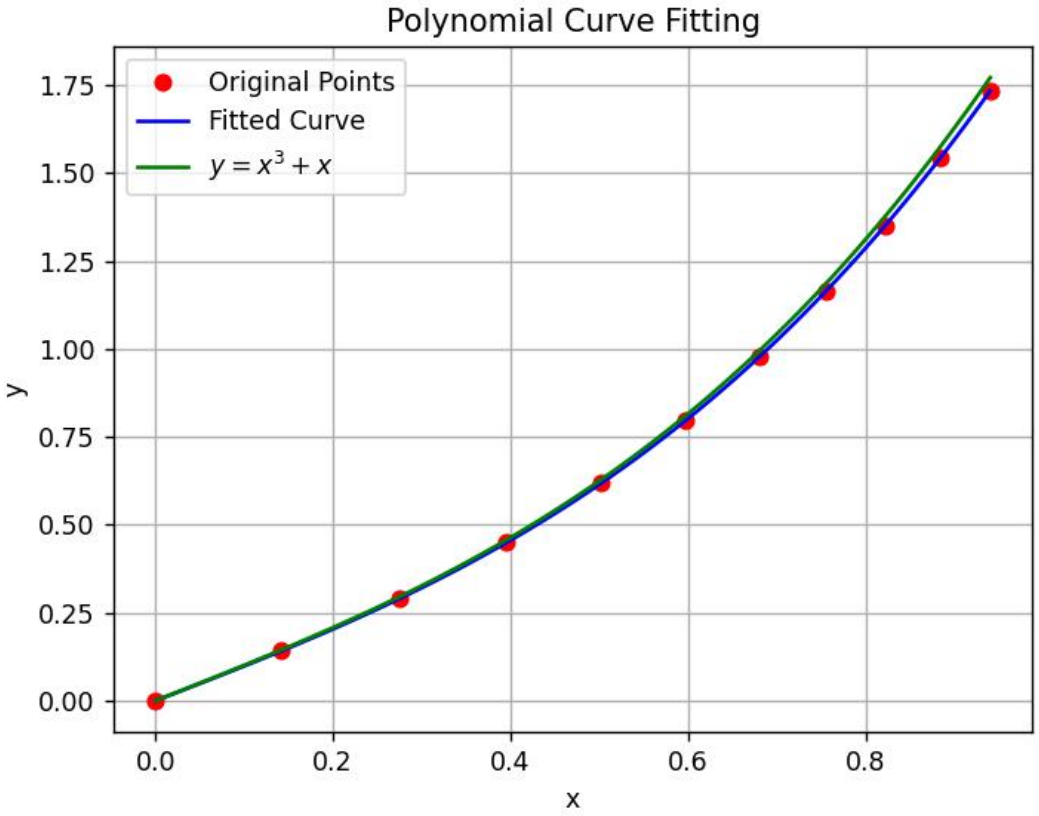


图 8 重构方程与原方程对比

重构曲线的方程：y=0.9347 x3 + 0.05026 x2 + 0.9705 x + 0.000178

由图像可知，重构方程与原方程有一定的偏差，特别是在偏离原点越远的位置。

3 误差分析

3.1 误差测量标准

均方误差计算公式：$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_{true,i} - y_{pred,i}\right)^2$

MSE 越小，表示模型的预测误差越小，预测性能越好。

平均绝对误差公式：$MAE = \frac{1}{n}\left|y_{true,i} - y_{pred,i}\right|$

MAE 衡量的是预测值与实际值之间差的绝对值的平均。它对所有误差一视同仁，不像 MSE 那样对大误差给予更大的权重。因此，MAE 也是越小越好。

R^2的公式：　　$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y_{true,i} - y_{pred,i}\right)^2}{\sum_{i=1}^{n}\left(y_{true,i} - \bar{y}\right)^2}$

R^2 分数通常介于 0 和 1 之间，表示模型解释的方差占总方差的比例。一个 R^2 分数接近 1 表示模型拟合得非常好，能够解释大部分的数据变异性。

计算拟合曲线与原始曲线的均方误差，均方根误差，平均绝对误差和R^2,可得到以下结果：

表 7 曲线拟合效果评估

MSE	*RMSE*	*MAE*	R^2
0.0003	0.0159	0.0125	0.9990

均方误差（MSE），均方根误差（RMSE）、平均绝对误差（MAE）值很小，R^2的值接近 1，这意味着模型能够准确地预测数据，并且与真实值的偏差很小，但是仍需考虑误差存在的原因：

3.2 误差分析

3.2.1 曲线假设的局限性：将曲线视为由一系列圆弧段组成的假设存在局限性，并不完全符合实际情况。

3.2.2 采样点数量较少：采样点的数量对于最终曲线方程的准确性至关重要。本题只取了 10 个样本点，数量较少，特别是在曲线变化较剧烈的区域，可能会导致曲线方程的近似精度下降。可以增加采样点以减少误差。

3.2.3 累积误差：累积曲率的方法是根据前一个点的坐标来迭代后一个点的坐标，如果前一个坐标有误差，这个误差会随着点的增多而累加，并最终导致曲线方程的不准确。为了减小累积误差的影响，可以使用一些校正手段，比如基于最小二乘法的拟合，或者在计算过程中引入一些校正因子来平衡累积误差。

3.2.4 弧长间隔较大：曲线上采样点的间距会影响使用切线方向的位移来近似圆

周上点运动的精确度，间隔越大，精确度越小。

4 优化

以更小的弧长间隔来选取样本点，即从 0.2 变为 0.1，并将样本点数量从 10 个增加到 20 个：

4.1 计算 20 个样本点对应的坐标 x 值和曲率（与 10 个采样点所用方法一致）

表 8 20 个样本点的 x 值和曲率值

x	0.0000	0.0705	0.1400	0.2076	0.2724	0.3341	0.3923	0.4470	0.4984	5465
k	0.0000	0.1463	0.2720	0.3629	0.4148	0.4320	0.4237	0.3996	0.3675	0.3329

x	0.5918	0.6343	0.6744	0.7123	0.7482	0.7823	0.8147	0.8457	0.8754	0.9038
k	0.2990	0.2676	0.2392	0.2140	0.1919	0.1726	0.1558	0.1411	0.1282	0.1170

4.2 增加样本点后的重构曲线

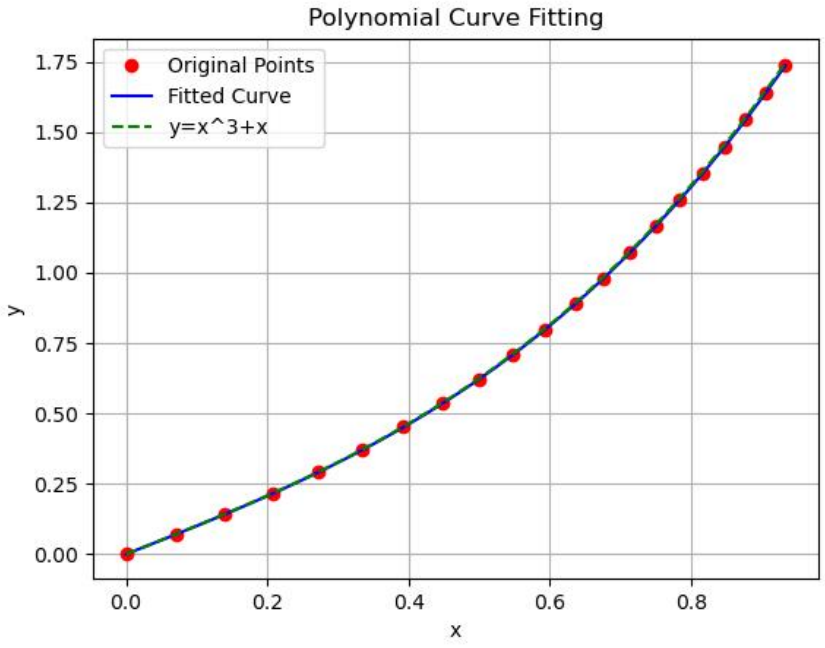


图 9 增加采样点后重构方程与原方程对比

改进后重构曲线的方程：$y=0.9833\ x^3 + 0.01298\ x^2 + 0.9924\ x + 0.00008481$

从图上可以看出，拟合效果良好，较之前有很大改进。

计算改进后的重构曲线与原始曲线的均方误差，均方根误差，平均绝对误差和R^2,可得到以下结果：

表 9 曲线拟合效果评估

MSE	*RMSE*	*MAE*	R^2
2.0109e-09	0.0004	0.0004	0.9999

与优化前相比，均方误差（MSE），均方根误差（RMSE）、平均绝对误差（MAE）值更小了，R^2的值也更接近 1，这意味着模型有很大的改进，预测结果与实际结果的偏差非常小。

4.3 增加样本点对模型的改进：

精度提高：更多的样本点可以更精细地捕捉到曲线的局部特性，特别是在曲率变化较大的区域，减小曲线形状对模型的影响；

误差累积减小：减小弧长间隔可以使每次迭代的近似误差减小。在迭代求解采样点坐标的过程中,是通过在每一步使用切线方向的位移来近似的,间隔越小,精度越准。

五、参考文献

[1]马雷,田中旭,邸义.地下管线检测中的曲线重构算法[J].机床与液压,2006,(11):13-15.

[2]吴云帆,张仕明,吴玉国,等.基于三次样条插值法的凸轮型线误差算法研究[J].制造业自动化,2023,45(10):66-70.

[3]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.

[4]金峰.基于V-系统的曲线拟合方法研究[D].江南大学,2022.DOI:10.27169/d.cnki.gwqgu.2022.000251.

[5]褚宝增,齐良平.平面曲线与空间曲线曲率及其算法[J].德州学院学

报,2013,29(02):18-21.
[6]徐应祥.通用可调形三次三角 Hermit 插值样条[J/OL].仲恺农业工程学院学报,1-6[2024-04-21].http://kns.cnki.net/kcms/detail/44.1660.s.20240415.1848.004.html.
[7] 于潇雁,蓝兆辉. 三次样条插值在平面凸轮廓线曲率半径求解中的应用[J].机械传动,2008,32(1):50-51. DOI:10.3969/j.issn.1004-2539.2008.01.015.
[8] 王飞. 插值曲率线与特征线的 B 样条曲面构造[D]. 安徽:中国科学技术大学,2018.

六、附录

第一题代码：

1. 计算给定了两组不同初始状态下受力前后的波长值即每组 6 个数据所对应的曲率

```
def compute_curvature(lamba, lamba0, c):
    # 计算曲率
    k = (c * (lamba - lamba0)) / lamba0
    return k
# 已知的 lamba 和 lamba0
# 测试 1
lamba_values_1 = [1529.808, 1529.807, 1529.813, 1529.812, 1529.814, 1529.809]
# 曲线上点的 lamba 值
lamba0_values_1 = [1529, 1529, 1529, 1529, 1529, 1529]  # 曲线上点的 lamba0 值
# 测试 2
lamba_values_2 = [1541.095, 1541.092, 1541.090, 1541.093, 1541.094, 1541.091]
# 曲线上点的 lamba 值
lamba0_values_2 = [1540, 1540, 1540, 1540, 1540, 1540]  # 曲线上点的 lamba0 值
# 常数 c
```

```
c = 4200.0
curvature_values_1 = []
curvature_values_2 = []
# 计算每个点的曲率(测试 1)
for lamba, lamba0 in zip(lamba_values_1, lamba0_values_1):
    k = compute_curvature(lamba, lamba0, c)
    curvature_values_1.append(k)
# 计算每个点的曲率(测试 2)
curvature_values = []
for lamba, lamba0 in zip(lamba_values_2, lamba0_values_2):
    k = compute_curvature(lamba, lamba0, c)
    curvature_values_2.append(k)
# 打印每个点的曲率(测试 1)
print("测试 1:")
for i, k in enumerate(curvature_values_1):
    print(f"FBG {i+1} curvature: {k}")
# 打印每个点的曲率(测试 2)
print("测试 2:")
for i, k in enumerate(curvature_values_2):
    print(f"FBG {i + 1} curvature: {k}")
```

2. 根据计算出来的曲率，用三次样条插值法拟合 x 和 k 的曲线：

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
# 测试一的曲率值点
curvature_points_test1 = [(0.0, 2.21948986265531), (0.6, 2.216742969261), (1.2, 2.233224329627487), (1.8, 2.230477436232552), (2.4, 2.2359712230217976), (3.0, 2.222236756049621)]
```

```
# 测试二的曲率值点
curvature_points_test2 = [(0.0, 2.986363636363711), (0.6, 2.9781818181820863),
(1.2, 2.9727272727270493), (1.8, 2.9809090909092943), (2.4, 2.9836363636365024),
(3.0, 2.975454545454258)]
# 三次样条插值函数的创建函数
def create_interpolated_curve(curvature_points):
    arc_length = [point[0] for point in curvature_points]
    curvature = [point[1] for point in curvature_points]
    return interp1d(arc_length, curvature, kind='cubic')
# 创建测试一的插值函数
f_test1 = create_interpolated_curve(curvature_points_test1)
# 创建测试二的插值函数
f_test2 = create_interpolated_curve(curvature_points_test2)
# 要求的 x 坐标位置
x_values = [0.3, 0.4, 0.5, 0.6, 0.7]
# 打印测试一结果
print("Testing 1:")
for x in x_values:
    arc_length_x = x / np.cos(np.pi/4)
    curvature_x = f_test1(arc_length_x)
    print(f"At x = {x} meters, curvature is approximately {curvature_x:.4f}")
# 打印测试二结果
print("\nTesting 2:")
for x in x_values:
    arc_length_x = x / np.cos(np.pi/4)
    curvature_x = f_test2(arc_length_x)
    print(f"At x = {x} meters, curvature is approximately {curvature_x:.4f}")
# 绘制测试一的插值图像
plt.figure(figsize=(10, 6))
plt.plot([point[0] for point in curvature_points_test1], [point[1] for point in
```

```
curvature_points_test1], 'o', label='Test 1 Points')
x_interp = np.linspace(min([point[0] for point in curvature_points_test1]), max([point[0] for point in curvature_points_test1]), 1000)
plt.plot(x_interp, f_test1(x_interp), label='Test 1 Interpolated Curve')
plt.xlabel('Arc Length (m)')
plt.ylabel('Curvature')
plt.title('Test 1 Interpolated Curve of Curvature')
plt.legend()
plt.grid(True)
plt.show()
# 绘制测试二的插值图像
plt.figure(figsize=(10, 6))
plt.plot([point[0] for point in curvature_points_test2], [point[1] for point in curvature_points_test2], 'o', label='Test 2 Points')
x_interp = np.linspace(min([point[0] for point in curvature_points_test2]), max([point[0] for point in curvature_points_test2]), 1000)
plt.plot(x_interp, f_test2(x_interp), label='Test 2 Interpolated Curve')
plt.xlabel('Arc Length (m)')
plt.ylabel('Curvature')
plt.title('Test 2 Interpolated Curve of Curvature')
plt.legend()
plt.grid(True)
plt.show()
```

第二题代码

1.根据测试一的数据打印平面曲线

```
import csv
from math import pi
```

```
import numpy as np
from scipy.interpolate import interp1d
from scipy.integrate import odeint
import matplotlib.pyplot as plt
# 设置全局字体为宋体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 已知的曲率值列表和对应的弧长（这里假设弧长是等间距的）
kappa_values = [2.21948986265531, 2.216742969261, 2.233224329627487, 2.230477436232552, 2.2359712230217976, 2.222236756049621]
s_values = np.linspace(0, len(kappa_values) - 1, len(kappa_values)) * 0.6  # 等间距的弧长
# 使用三次样条插值得到连续的曲率函数，并允许外插
kappa_func = interp1d(s_values, kappa_values, kind='cubic', bounds_error=False, fill_value='extrapolate')
# 定义曲线重构的微分方程系统
def reconstruct_curve(w, s):
    x, y, theta = w
    kappa = kappa_func(s)
    dx = np.cos(theta)
    dy = np.sin(theta)
    dtheta = kappa
    return [dx, dy, dtheta]
# 初始条件
initial_conditions = [0, 0, pi/ 4]  # 原点，且初始角度为 45 度
#initial_conditions = [0, 0, 3*pi/ 4]#初始角度为 135 度
#initial_conditions = [0, 0, -3*pi/ 4]#初始角度为 225 度
#initial_conditions = [0, 0, -pi/ 4]#初始角度为 315 度
# 细化弧长 s 的范围以得到更平滑的曲线
s_fine = np.linspace(0, s_values[-1], 1000)
```

```
# 使用 odeint 进行数值积分
solution = odeint(reconstruct_curve, initial_conditions, s_fine)
x_curve, y_curve, theta_curve = solution.T
# 将 x 和 y 坐标保存到 CSV 文件
with open('curve_points.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['X', 'Y'])  # 写入标题行
    for x, y in zip(x_curve, y_curve):
        writer.writerow([x, y])  # 写入数据行
# 创建插值函数
interp_y_curve = interp1d(x_curve, y_curve, kind='cubic', fill_value="extrapolate")
# 给定的 x 值列表
x_values_to_interpolate = [0.000000, -0.552799,-0.749278, -0.288132, 0.120319, -0.153961]
# 打印给定 x 值和对应的 y 值
for x in x_values_to_interpolate:
    y = interp_y_curve(x)
    print(f"X: {x:.6f}, Y: {y:.6f}")
# 使用 interp1d 对 s_fine 上的 x_curve 和 y_curve 做插值，以找到与 s_values 对应的点
x_kappa_points = interp1d(s_fine, x_curve, kind='cubic')(s_values)
y_kappa_points = interp1d(s_fine, y_curve, kind='cubic')(s_values)
# 打印曲率对应的 6 个坐标值
for i in range(len(kappa_values)):
    print(f"Kappa: {kappa_values[i]:.6f}, X: {x_kappa_points[i]:.6f}, Y: {y_kappa_points[i]:.6f}")
# 绘制重构的曲线
plt.plot(x_curve, y_curve, label='Reconstructed Curve')
# 绘制散点图显示曲率值对应的坐标点
plt.scatter(x_kappa_points, y_kappa_points, color='red', marker='o', label='Kappa
```

```
Values Points')
# 设置图标的标题和标签
plt.xlabel('X 轴')
plt.ylabel('Y 轴')
plt.title('测试一 45°')  # 添加图像标题
#plt.title('测试一 135°')  # 添加图像标题
#plt.title('测试一 225°')  # 添加图像标题
#plt.title('测试一 315°')  # 添加图像标题
# 显示图例和网格
plt.legend()
plt.grid(True)
# 显示图像
plt.show()
```

2.**根据测试二的数据打印平面曲线**

```
import csv
from math import pi
import numpy as np
from scipy.interpolate import interp1d
from scipy.integrate import odeint
import matplotlib.pyplot as plt
# 设置全局字体为宋体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 已知的曲率值列表和对应的弧长（这里假设弧长是等间距的）
kappa_values = [2.986363636363711,2.9781818181820863,2.9727272727270493,
2.9809090909092943, 2.9836363636365024,2.975454545454258]
s_values = np.linspace(0, len(kappa_values) - 1, len(kappa_values)) * 0.6  # 等间距
的弧长
```

```
# 使用三次样条插值得到连续的曲率函数，并允许外插
kappa_func = interp1d(s_values, kappa_values, kind='cubic', bounds_error=False, fill_value='extrapolate')
# 定义曲线重构的微分方程系统
def reconstruct_curve(w, s):
    x, y, theta = w
    kappa = kappa_func(s)
    dx = np.cos(theta)
    dy = np.sin(theta)
    dtheta = kappa
    return [dx, dy, dtheta]
# 初始条件
initial_conditions = [0, 0, pi/ 4]  # 原点，且初始角度为 45 度
#initial_conditions = [0, 0, 3*pi/ 4]#初始角度为 135 度
#initial_conditions = [0, 0, -3*pi/ 4]#初始角度为 225 度
#initial_conditions = [0, 0, -pi/ 4]#初始角度为 315 度
# 细化弧长 s 的范围以得到更平滑的曲线
s_fine = np.linspace(0, s_values[-1], 1000)
# 使用 odeint 进行数值积分
solution = odeint(reconstruct_curve, initial_conditions, s_fine)
x_curve, y_curve, theta_curve = solution.T
# 将 x 和 y 坐标保存到 CSV 文件
with open('curve_points.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['X', 'Y'])  # 写入标题行
    for x, y in zip(x_curve, y_curve):
        writer.writerow([x, y])  # 写入数据行
# 创建插值函数
interp_y_curve = interp1d(x_curve, y_curve, kind='cubic', fill_value="extrapolate")
```

```
# 给定的 x 值列表
x_values_to_interpolate = [0.000000, -0.552799,-0.749278, -0.288132, 0.120319, -0.153961]
# 打印给定 x 值和对应的 y 值
for x in x_values_to_interpolate:
    y = interp_y_curve(x)
    print(f"X: {x:.6f}, Y: {y:.6f}")
# 使用 interp1d 对 s_fine 上的 x_curve 和 y_curve 做插值，以找到与 s_values 对应的点
x_kappa_points = interp1d(s_fine, x_curve, kind='cubic')(s_values)
y_kappa_points = interp1d(s_fine, y_curve, kind='cubic')(s_values)
# 打印曲率对应的 6 个坐标值
for i in range(len(kappa_values)):
    print(f"Kappa: {kappa_values[i]:.6f}, X: {x_kappa_points[i]:.6f}, Y: {y_kappa_points[i]:.6f}")
# 绘制重构的曲线
plt.plot(x_curve, y_curve, label='Reconstructed Curve')
# 绘制散点图显示曲率值对应的坐标点
plt.scatter(x_kappa_points, y_kappa_points, color='red', marker='o', label='Kappa Values Points')
# 设置图标的标题和标签
plt.xlabel('X 轴')
plt.ylabel('Y 轴')
plt.title('测试一 45°')  # 添加图像标题
#plt.title('测试一 135°')  # 添加图像标题
#plt.title('测试一 225°')  # 添加图像标题
#plt.title('测试一 315°')  # 添加图像标题
# 显示图例和网格
plt.legend()
plt.grid(True)
```

```
# 显示图像
plt.show()
```

3.根据测试一的数据构建三维曲线

```
import csv
from math import pi
import numpy as np
from scipy.interpolate import interp1d
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# 设置全局字体为宋体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 初始化不同初始角度
initial_angles = [pi/4, -pi/4, 3*pi/4, -3*pi/4]
plot_labels = ['π/4', '-π/4', '3π/4', '-3π/4']

for angle, label in zip(initial_angles, plot_labels):
    # 已知的曲率值列表和对应的弧长（这里假设弧长是等间距的）
    kappa_values =[2.21948986265531, 2.216742969261, 2.233224329627487, 2.230477436232552, 2.2359712230217976, 2.222236756049621]
    #kappa_values = [2.98636364,2.97818182,2.97272727,2.98090909,2.98363636,2.97545455]
    s_values = np.linspace(0, len(kappa_values) - 1, len(kappa_values)) * 0.6  # 等间距的弧长
    # 使用三次样条插值得到连续的曲率函数，并允许外插
    kappa_func = interp1d(s_values, kappa_values, kind='cubic', bounds_error=False, fill_value='extrapolate')
```

```
    # 定义曲线重构的微分方程系统
    def reconstruct_curve(w, s):
        x, y, z, theta = w
        kappa = kappa_func(s)
        dx = np.cos(theta)
        dy = np.sin(theta)
        dz = np.abs(np.tan(kappa))  # 将 Z 轴的负值改为正值
        dtheta = kappa
        return [dx, dy, dz, dtheta]
    # 初始条件
    initial_conditions = [0, 0, 0, angle]  # 原点，且初始角度为指定角度
    # 细化弧长 s 的范围以得到更平滑的曲线
    s_fine = np.linspace(0, s_values[-1], 1000)
    # 使用 odeint 进行数值积分
    solution = odeint(reconstruct_curve, initial_conditions, s_fine)
    x_curve, y_curve, z_curve, theta_curve = solution.T
    # 创建插值函数
    interp_y_curve = interp1d(x_curve, y_curve, kind='cubic',
fill_value="extrapolate")
    # 绘制重构的曲线
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(x_curve, y_curve, z_curve, label=f'Reconstructed Curve, θ = {label}')
    # 设置图例的标题和标签
    ax.set_xlabel('X 轴')
    ax.set_ylabel('Y 轴')
    ax.set_zlabel('Z 轴')
    ax.set_title(f'测试一 {label}')  # 添加图像标题
# 显示图像
plt.show()
```

4.根据测试点二的数据构建三维曲线

```
import csv
from math import pi
import numpy as np
from scipy.interpolate import interp1d
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# 设置全局字体为宋体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 初始化不同初始角度
initial_angles = [pi/4, -pi/4, 3*pi/4, -3*pi/4]
plot_labels = ['π/4', '-π/4', '3π/4', '-3π/4']
for angle, label in zip(initial_angles, plot_labels):
    # 已知的曲率值列表和对应的弧长（这里假设弧长是等间距的）
     #kappa_values  =[2.21948986265531,  2.216742969261,  2.233224329627487, 2.230477436232552, 2.2359712230217976, 2.222236756049621]#测试一
    kappa_values = [2.98636364, 2.97818182, 2.97272727, 2.98090909, 2.98363636, 2.97545455]#测试二
    s_values = np.linspace(0, len(kappa_values) - 1, len(kappa_values)) * 0.6  # 等间距的弧长
    # 使用三次样条插值得到连续的曲率函数，并允许外插
    kappa_func = interp1d(s_values, kappa_values, kind='cubic', bounds_error=False, fill_value='extrapolate')
    # 定义曲线重构的微分方程系统
    def reconstruct_curve(w, s_a):
        x, y, z, theta = w
        kappa = kappa_func(s_a)
```

```
            dx = np.cos(theta)
            dy = np.sin(theta)
            dz = np.abs(np.tan(kappa))  # 将 Z 轴的负值改为正值
            # 缩放 Z 轴的值到 0 到 1 之间
            if np.min(dz) != np.max(dz):
                dz = (dz - np.min(dz)) / (np.max(dz) - np.min(dz))
            else:
                dz = dz / np.max(dz)  # 避免除以零的情况
            dtheta = kappa
            return [dx, dy, dz, dtheta]
        # 初始条件
        initial_conditions = [0, 0, 0, angle]  # 原点，且初始角度为指定角度
        # 细化弧长 s 的范围以得到更平滑的曲线
        s_fine = np.linspace(0, s_values[-1], 1000)
        # 使用 odeint 进行数值积分
        solution = odeint(reconstruct_curve, initial_conditions, s_fine)
        x_curve, y_curve, z_curve, theta_curve = solution.T
        # 创建插值函数
        interp_y_curve = interp1d(x_curve, y_curve, kind='cubic',
fill_value="extrapolate")
        # 绘制重构的曲线
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.plot(x_curve, y_curve, z_curve, label=f'Reconstructed Curve, θ = {label}')
        # 设置图例的标题和标签
        ax.set_xlabel('X 轴')
        ax.set_ylabel('Y 轴')
        ax.set_zlabel('Z 轴')
        ax.set_title(f'测试二 {label}')  # 添加图像标题
```

```
# 显示图像
plt.show()
```

5.根据第一问拟合出来的曲线，以步长为 0.1，取 20 个数据点

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
# 测试一的曲率值点
curvature_points_test1 = [(0.0, 2.21948986265531), (0.6, 2.216742969261), (1.2, 2.233224329627487), (1.8, 2.230477436232552), (2.4, 2.2359712230217976), (3.0, 2.222236756049621)]
# 测试二的曲率值点
curvature_points_test2 = [(0.0, 2.986363636363711), (0.6, 2.9781818181820863), (1.2, 2.9727272727270493), (1.8, 2.9809090909092943), (2.4, 2.9836363636365024), (3.0, 2.975454545454258)]
# 三次样条插值函数的创建函数
def create_interpolated_curve(curvature_points):
    arc_length = [point[0] for point in curvature_points]
    curvature = [point[1] for point in curvature_points]
    return interp1d(arc_length, curvature, kind='cubic')
# 创建测试一的插值函数
f_test1 = create_interpolated_curve(curvature_points_test1)
# 创建测试二的插值函数
f_test2 = create_interpolated_curve(curvature_points_test2)
# 要求的 x 坐标位置
x_values = np.arange(0, 3.1, 0.1)
# 打印测试一结果
print("Testing 1:")
for x in x_values:
```

```
    arc_length_x = x / np.cos(np.pi/4)
    if arc_length_x >= curvature_points_test1[0][0] and arc_length_x <= curvature_points_test1[-1][0]:
        curvature_x = f_test1(arc_length_x)
        print(f"At x = {x} meters, curvature is approximately {curvature_x:.4f}")
# 打印测试二结果
print("\nTesting 2:")
for x in x_values:
    arc_length_x = x / np.cos(np.pi/4)
    if arc_length_x >= curvature_points_test2[0][0] and arc_length_x <= curvature_points_test2[-1][0]:
        curvature_x = f_test2(arc_length_x)
        print(f"At x = {x} meters, curvature is approximately {curvature_x:.4f}")
```

6.根据由测试一的曲线求出来的 20 个数据点，再次进行曲线拟合

```
import csv
from math import pi
import numpy as np
from scipy.interpolate import interp1d
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# 设置全局字体为宋体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 初始化不同初始角度
initial_angles = [pi/4, -pi/4, 3*pi/4, -3*pi/4]
plot_labels = ['π/4', '-π/4', '3π/4', '-3π/4']
```

```
for angle, label in zip(initial_angles, plot_labels):
    # 已知的曲率值列表和对应的弧长（这里假设弧长是等间距的）
    kappa_values = [2.2195,2.2139,2.2120,2.2129,2.2158,2.2200,2.2247,2.2290,2.2323,2.2337,2.2334,2.2321,2.2309,2.2305,2.2314,2.2330,2.2347,2.2360,2.2362,2.2347,2.2308,
     2.2241]
    s_values = np.linspace(0, len(kappa_values) - 1, len(kappa_values)) * 0.6  # 等间距的弧长
    # 使用三次样条插值得到连续的曲率函数，并允许外插
    kappa_func = interp1d(s_values, kappa_values, kind='cubic', bounds_error=False, fill_value='extrapolate')
    # 定义曲线重构的微分方程系统
    def reconstruct_curve(w, s):
        x, y, z, theta = w
        kappa = kappa_func(s)
        dx = np.cos(theta)
        dy = np.sin(theta)
        dz = np.tan(kappa)
        dtheta = kappa
        return [dx, dy, dz, dtheta]
    # 初始条件
    initial_conditions = [0, 0, 0, angle]  # 原点，且初始角度为指定角度
    # 细化弧长 s 的范围以得到更平滑的曲线
    s_fine = np.linspace(0, s_values[-1], 1000)
    # 使用 odeint 进行数值积分
    solution = odeint(reconstruct_curve, initial_conditions, s_fine)
    x_curve, y_curve, z_curve, theta_curve = solution.T
    # 创建插值函数
    interp_y_curve = interp1d(x_curve, y_curve, kind='cubic', fill_value="extrapolate")
```

```
    # 绘制重构的曲线
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(x_curve, y_curve, z_curve, label=f'Reconstructed Curve, θ = {label}')
    # 设置图例的标题和标签
    ax.set_xlabel('X 轴')
    ax.set_ylabel('Y 轴')
    ax.set_zlabel('Z 轴')
    ax.set_title(f'测试一 {label}')  # 添加图像标题
# 显示图像
plt.show()
```

7.根据由测试二预测出的数据点进行曲线拟合

```
import csv
from math import pi
import numpy as np
from scipy.interpolate import interp1d
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# 设置全局字体为宋体
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 初始化不同初始角度
initial_angles = [pi/4, -pi/4, 3*pi/4, -3*pi/4]
plot_labels = ['π/4', '-π/4', '3π/4', '-3π/4']
for angle, label in zip(initial_angles, plot_labels):
    # 已知的曲率值列表和对应的弧长（这里假设弧长是等间距的）
    kappa_values = [2.9864, 2.9852, 2.9834, 2.9812, 2.9788, 2.9764, 2.9745, 2.9731,
```

```
2.9726, 2.9732, 2.9748, 2.9770, 2.9794, 2.9814, 2.9829, 2.9837, 2.9840, 2.9836, 2.9826, 2.9811, 2.9789, 2.9761]
    s_values = np.linspace(0, len(kappa_values) - 1, len(kappa_values)) * 0.6  # 等间距的弧长
    # 使用三次样条插值得到连续的曲率函数，并允许外插
    kappa_func = interp1d(s_values, kappa_values, kind='cubic', bounds_error=False, fill_value='extrapolate')
    # 定义曲线重构的微分方程系统
    def reconstruct_curve(w, s):
        x, y, z, theta = w
        kappa = kappa_func(s)
        dx = np.cos(theta)
        dy = np.sin(theta)
        dz = np.tan(kappa)
        dtheta = kappa
        return [dx, dy, dz, dtheta]
    # 初始条件
    initial_conditions = [0, 0, 0, angle]  # 原点，且初始角度为指定角度
    # 细化弧长 s 的范围以得到更平滑的曲线
    s_fine = np.linspace(0, s_values[-1], 1000)
    # 使用 odeint 进行数值积分
    solution = odeint(reconstruct_curve, initial_conditions, s_fine)
    x_curve, y_curve, z_curve, theta_curve = solution.T
    # 创建插值函数
    interp_y_curve = interp1d(x_curve, y_curve, kind='cubic', fill_value="extrapolate")
    # 绘制重构的曲线
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(x_curve, y_curve, z_curve, label=f'Reconstructed Curve, θ = {label}')
```

```
    # 设置图例的标题和标签
    ax.set_xlabel('X 轴')
    ax.set_ylabel('Y 轴')
    ax.set_zlabel('Z 轴')
    ax.set_title(f'测试二 {label}')  # 添加图像标题
# 显示图像
plt.show()
```

第三题的代码

1. 计算给定曲线的曲率

```
import numpy as np
from scipy.integrate import quad
from scipy.optimize import fsolve
# 定义函数及其导数
def f(x):
    return x ** 3 + x
def df(x):
    return 3 * x ** 2 + 1
def ddf(x):
    return 6 * x
# 计算曲率
def curvature(x):
    return abs(ddf(x)) / (1 + df(x) ** 2) ** (3 / 2)
# 计算弧长（从 x0 到 x）
def arc_length(x0, x):
    integrand = lambda t: np.sqrt(1 + df(t) ** 2)
    return quad(integrand, x0, x)[0]
# 初始化参数
x_start = 0
```

```
x_end = 1
arc_length_target = 0.2
num_points = 10
points = [x_start]
# 逐步添加采样点
while len(points) < num_points:
    last_x = points[-1]
    # 尝试增加一个小步长，并计算新的弧长
    step_size = (x_end - last_x) / (num_points - len(points))
    guess_x = last_x + step_size
    guess_arc_length = arc_length(last_x, guess_x)
    # 如果弧长太短，增加步长；如果太长，减少步长
    if guess_arc_length < arc_length_target:
        step_size *= 2
    else:
        step_size /= 2
        # 使用 fsolve 找到精确的 x 值，使得弧长接近目标值
    def find_x(x):
        return arc_length(last_x, x) - arc_length_target
  new_x = fsolve(find_x, guess_x)[0]
 # 添加新点到列表中，并检查是否超出范围
    if new_x <= x_end:
        points.append(new_x)
    else:
        break  # 如果超出范围，则停止添加点
# 计算每个采样点的曲率
curvatures = [curvature(x) for x in points]
# 打印结果
for x, K in zip(points, curvatures):
```

```
    print(f"x = {x:.4f}，曲率 K = {K:.4f}")
```

2. 根据曲率来拟合曲线

```
import math
import numpy as np
import matplotlib.pyplot as plt
# 初始点
x0, y0 = 0, 0
# 每个点之间的间距
interval = 0.2
# 每个点的曲率
curvatures = [0.0000, 0.2720, 0.4148, 0.4237, 0.3675, 0.2990, 0.2392, 0.1919, 0.1558,
0.1282]
# 计算每个点的坐标
points = [(x0, y0)]
angle = math.radians(45)  # 初始角度为 45 度
for curvature in curvatures:
    # 计算下一个点的角度
    angle += interval * curvature
    # 计算下一个点的坐标
    x = points[-1][0] + interval * math.cos(angle)
    y = points[-1][1] + interval * math.sin(angle)
    points.append((x, y))
# 输出每个点的坐标
for i, point in enumerate(points):
    print(f"Point {i + 1}: ({point[0]}, {point[1]})")
# 提取 x 和 y 坐标
x = [point[0] for point in points]
y = [point[1] for point in points]
```

```
# 多项式拟合
degree = 3  # 多项式的阶数，可以根据需要调整
coefficients = np.polyfit(x, y, degree)
# 构建多项式曲线方程
polynomial_equation = np.poly1d(coefficients)
# 输出多项式曲线方程
print("Polynomial Equation:")
print(polynomial_equation)
# 绘制原始点
plt.plot(x, y, 'ro', label='Original Points')
# 生成拟合曲线的 x 值
x_fit = np.linspace(min(x), max(x), 100)
# 使用多项式曲线方程计算 y 值
y_fit = polynomial_equation(x_fit)
# 绘制拟合曲线
plt.plot(x_fit, y_fit, 'b-', label='Fitted Curve')
# 绘制  y = x^3 + x  曲线
x_curve = np.linspace(min(x), max(x), 100)
y_curve = x_curve ** 3 + x_curve
plt.plot(x_curve, y_curve, 'g-', label='$y = x^3 + x$')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polynomial Curve Fitting')
plt.legend()
plt.grid(True)
plt.show()
```

3.误差估计

```
import math
import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# 初始点
x0, y0 = 0, 0
# 每个点之间的间距
interval = 0.1
# 每个点的曲率
curvatures = [0.0000, 0.1463, 0.2720, 0.3629, 0.4148, 0.4320, 0.4237, 0.3996, 0.3675,
 0.3329, 0.2990, 0.2676, 0.2392, 0.2140, 0.1919, 0.1726, 0.1558, 0.1411, 0.1282, 0.11
70]

# 计算每个点的坐标
points = [(x0, y0)]
angle = math.radians(45)  # 初始角度为 45 度
for curvature in curvatures:
    # 计算下一个点的角度
    angle += interval * curvature
    # 计算下一个点的坐标
    x = points[-1][0] + interval * math.cos(angle)
    y = points[-1][1] + interval * math.sin(angle)
    points.append((x, y))

# 提取 x 和 y 坐标
x = [point[0] for point in points]
y = [point[1] for point in points]

# 多项式拟合
degree = 3  # 多项式的阶数，可以根据需要调整
coefficients = np.polyfit(x, y, degree)
```

```
# 构建多项式曲线方程
polynomial_equation = np.poly1d(coefficients)

# 打印重构的曲线方程
print("重构的曲线方程:")
print(polynomial_equation)

# 绘制原始点
plt.plot(x, y, &apos;ro&apos;, label=&apos;Original Points&apos;)

# 生成拟合曲线的 x 值
x_fit = np.linspace(min(x), max(x), 100)
# 使用多项式曲线方程计算 y 值
y_fit = polynomial_equation(x_fit)
# 绘制拟合曲线
plt.plot(x_fit, y_fit, &apos;b-&apos;, label=&apos;Fitted Curve&apos;)

# 绘制与 y=x^3+x 的曲线对比
x_ref = np.linspace(min(x), max(x), 100)
y_ref = x_ref ** 3 + x_ref
plt.plot(x_ref, y_ref, &apos;g--&apos;, label=&apos;y=x^3+x&apos;)

# 计算 R²值
r_squared = r2_score(y, polynomial_equation(x))
print("R²:", r_squared)

# 计算均方误差（MSE）
mse = mean_squared_error(y, polynomial_equation(x))
print("Mean Squared Error (MSE):", mse)
```

```
# 计算均方根误差（RMSE）
rmse = math.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# 计算平均绝对误差（MAE）
mae = mean_absolute_error(y, polynomial_equation(x))
print("Mean Absolute Error (MAE):", mae)

plt.xlabel(&apos;x&apos;)
plt.ylabel(&apos;y&apos;)
plt.title(&apos;Polynomial Curve Fitting&apos;)
plt.legend()
plt.grid(True)
plt.show()
```