# 数值分析 - 第三章实验报告

强基数学 2001 班樊睿

December 19,2022

## 1 abstract

本项目分别用 ppform 和 B-spline 实现了分段线性样条和三次样条的函数拟合和曲线拟合。

## 2 ppform 的理论及实现

ppform 使用传统的待定系数方法,对除右端点外的结点 $x_0, x_1, \ldots, x_{n-1}$（注：本文结点标号全部从 0 到 $n$，即结点总数为 $n+1$）求出样条在结点处的 0 至 $m$ 次导数值，然后利用泰勒展开式求出每段的函数解析式。

本节完整程序见 `Spline.h`。

### 2.1 分段线性样条 $\mathbb{S}_1^0$

#### 2.1.1 问题描述

已知结点处 $a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$ 的函数值 $f(x_0) = y_0, f(x_1) = y_1, \ldots, f(x_n) = y_n$，用分段线性样条 $s(x) \in \mathbb{S}_1^0$ 拟合 $f$。

#### 2.1.2 理论基础

设样条在结点 $(x_i, x_{i+1}$ 之间的解析式是 $s(x) = y_i + K_i(x_i)(x - x_i)$，则 $K_i = \dfrac{y_{i+1} - y_i}{x_{i+1} - x_i}$。

### 2.1.3 代码实现

设计 LinearSpline 类。

- vector<type> 型私有成员变量 x：结点坐标。
- vector<type> 型私有成员变量 f：结点函数值。
- 构造函数 LinearSpline(const vector<type>& x, const vector <type>& f)：默认构造。
- type 型公有成员函数 GetValue(const type& _x)：获取样条函数在 _x 处的拟合值。用 STL 自带的upper_bound函数定位到代求点所在区间后直接取值即可。

具体代码实现如下。同时还支持了对具体函数 $f$ 在区间 $[l,r]$ 进行 $n+1$ 个结点的等距插值。

```
template <class type>
class LinearSpline {
private:
    vector <type> x, f; // 结点，函数值
public:
    LinearSpline(const vector<type>& x, const vector<type>& f)
        : x(x), f(f) {}
    type GetValue(const type& _x) {
        int i = upper_bound(x.begin(), x.end(), _x) - x.begin()
            - 1;
        type t = _x - x[i], c0 = f[i], c1 = (f[i+1] - f[i]) / (
            x[i+1] - x[i]);
        return c0 + c1 * t;
    }
};

template <class type>
LinearSpline<type> LinearSplineInterpolation(const Function <
    type>& f, const type& l, const type& r, const int& n) {
    vector <type> x(n+1), y(n+1);
    for(int i = 0; i <= n; ++ i) x[i] = l + (r - l) / n * i, y[
        i] = f(x[i]);
```

```
18      return LinearSpline<type>(x, y);
19  }
```

## 2.2 分段三次样条

### 2.2.1 问题描述

已知结点处 $a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$ 的函数值 $f(x_0) = y_0, f(x_1) = y_1, \ldots, f(x_n) = y_n$ 和端点处对一阶或二阶导数的限制（自然样条、完全样条、端点处二阶导指定的样条），用分段线性样条 $s(x) \in \mathbb{S}_3^2$ 拟合 $f$。

### 2.2.2 理论基础

设样条在结点 $(x_i, x_{i+1})$ 之间的解析式是 $s(x) = y_i + m_i(x - x_i) + \frac{M_i}{2}(x - x_i)^2 + \frac{M_{i+1} - M_i}{6(x_{i+1} - x_i)}(x - x_1)^3$。

当求自然样条或完全样条，即 $m_0 = f[x_0, x_0]$ 和 $m_n = f[x_n, x_n]$ 给定时，由讲义的 Lem 3.4,Lem 3.6 可得 $M_i$ 满足如下关系式：

$$\begin{bmatrix} 2 & 1 & & & & \\ \lambda_1 & 2 & \mu_1 & & & \\ & \lambda_2 & 2 & \mu_2 & & \\ & \ddots & \ddots & \ddots & & \\ & & & \lambda_{n-1} & 2 & \mu_{n-1} \\ & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = 6 \begin{bmatrix} f[x_0, x_0, x_1] \\ f[x_0, x_1, x_2] \\ f[x_1, x_2, x_3] \\ \vdots \\ f[x_{n-2}, x_{n-1}, x_n] \\ f[x_{n-1}, x_n, x_n] \end{bmatrix} \tag{1}$$

其中，$\lambda_i = \dfrac{x_i - x_{i-1}}{x_{i+1} - x_{i-1}}$，$\mu_i = \dfrac{x_{i+1} - x_i}{x_{i+1} - x_{i-1}}$，右边所有的二阶差商都可以直接计算。

然后根据 Lem 3.4 求 $m_i$：

$$\begin{aligned} m_i &= f[x_i, x_{i+1}] - \frac{1}{6}(M_{i+1} + 2M_i)(x_{i+1} - x_i), && i = 0, 1, \ldots, n-1 \\ m_n &= f[x_n, x_{n-1}] - \frac{1}{6}(M_{n-1} + 2M_n)(x_{n-1} - x_n), && i = n \end{aligned} \tag{2}$$

当求端点处二阶导给定的样条，即 $M_0$ 和 $M_n$ 给定时，$f[x_0, x_0, x_1]$ 和 $f[x_{n-1}, x_{n-1}, x_n]$ 未知但不再需要，只需去掉第一个和最后一个方程，解一个 $n-1$ 阶线性方程组即可。

### 2.2.3 代码实现

设计 `CubicSpline` 类。

- `vector<type>` 型私有成员变量 `x,f,m,M`：结点坐标、函数值、一阶导、二阶导。
- 构造函数 `CubicSpline(const vector<type>& x, const vector <type>& f, const string& m` 构造函数。mode 可选择 `Natural`、`Complete` 或 `Specified_Second_Derivatives`，`m0` 和 `mn` 是端点处的一阶或二阶导数值。具体实现时先计算二阶差商，然后解三对角方程组求出 $M_i$，最后求 $m_i$。
- `type` 型公有成员函数 `GetValue(const type& _x)`：获取样条函数在 `_x` 处的拟合值。用 STL 自带的`upper_bound`函数定位到代求点所在区间后直接取值即可。

具体代码实现如下。同时还支持了对具体函数 $f$ 在区间 $[l, r]$ 进行 $n+1$ 个结点的等距插值。

```
template <class type>
class CubicSpline {
private:
    vector <type> x, f, m, M;   // 结点，函数值，一阶导，二阶导
public:
    CubicSpline(const vector<type>& x, const vector<type>& f,
        const string& mode = "Natural", const type& m0 = 0,
        const type& mn = 0) : x(x), f(f) {
        int n = x.size() - 1;
        m.resize(n+1);
        vector <type> l(n+1), u(n+1);   // lambda, mu
        for (int i = 1; i <= n-1; ++ i) {
            l[i] = (x[i] - x[i-1]) / (x[i+1] - x[i-1]);
            u[i] = (x[i+1] - x[i]) / (x[i+1] - x[i-1]);
        }

```

```
15          vector <type> dq1(n+1), dq2(n+2);   // 一阶差商，二阶差
                商
16          vector <type> a(n+1), b(n), c(n), y(n+1);
17
18          if (mode == "Natural" && (m0 != 0 || mn != 0)) throw "
                Invalid Parameter!";
19          if (mode == "Complete" || mode == "Natural") {
20              // 计算差商
21              for (int i = 1; i <= n; ++ i)
22                  dq1[i] = (f[i] - f[i-1]) / (x[i] - x[i-1]);
23              dq2[1] = (dq1[1] - m0) / (x[1] - x[0]);
24              for (int i = 2; i <= n; ++ i)
25                  dq2[i] = (dq1[i] - dq1[i-1]) / (x[i] - x[i-2]);
26              dq2[n+1] = (mn - dq1[n]) / (x[n] - x[n-1]);
27              // 构造三对角方程组
28              for (int i = 1; i <= n-1; ++ i) {
29                  a[i] = 2;
30                  b[i-1] = u[i];
31                  c[i] = l[i];
32                  y[i] = 6 * dq2[i+1];
33              }
34              a[0] = 2, c[0] = 1, y[0] = 6 * dq2[1];
35              a[n] = 2, b[n-1] = 1, y[n] = 6 * dq2[n+1];
36              // 调用 Thomas 算法解出结点处的二阶导 M
37              M = Thomas(a, b, c, y);
38              // 根据 M 的值计算 m
39              m[0] = m0, m[n] = mn;
40              for (int i = 1; i <= n-1; ++ i)
41                  m[i] = dq1[i+1] - (2 * M[i] + M[i+1]) * (x[i+1]
                        - x[i]) / 6;
42          }
43          else if (mode == "Specified_Second_Derivatives") {
44              // 计算差商
45              for (int i = 1; i <= n; ++ i)
46                  dq1[i] = (f[i] - f[i-1]) / (x[i] - x[i-1]);
47              for (int i = 2; i <= n; ++ i)
48                  dq2[i] = (dq1[i] - dq1[i-1]) / (x[i] - x[i-2]);
49              // 构造三对角方程组
50              for (int i = 1; i <= n-1; ++ i) {
```

```
51                  a[i] = 2;
52                  b[i-1] = u[i];
53                  c[i] = l[i];
54                  y[i] = 6 * dq2[i+1];
55              }
56              a[0] = 1, y[0] = m0;
57              a[n] = 1, y[n] = mn;
58              // 调用 Thomas 算法解出结点处的二阶导 M
59              M = Thomas(a, b, c, y);
60              // 根据 M 的值计算 m
61              for (int i = 0; i <= n-1; ++ i)
62                  m[i] = dq1[i+1] - (2 * M[i] + M[i+1]) * (x[i+1]
                        - x[i]) / 6;
63              m[n] = dq1[n] - (2 * M[n] + M[n-1]) * (x[n-1] - x[n
                    ]) / 6;
64          }
65      }
66      type GetValue(const type& _x) const {
67          // 求出 x 所在结点区间，根据该区间上的解析式计算样条函
                数的值
68          int i = upper_bound(x.begin(), x.end(), _x) - x.begin()
                 - 1;
69          type t = _x - x[i], c0 = f[i], c1 = m[i], c2 = M[i] /
                2, c3 = (M[i+1] - M[i]) / (x[i+1] - x[i]) / 6;
70          return c0 + t * (c1 + t * (c2 + c3 * t));
71      }
72  };
73
74  template <class type>
75  CubicSpline<type> CubicSplineInterpolation(const Function <type
        >& f, const type& l, const type& r, const int& n, const
        string& mode = "Natural") {
76      vector <type> x(n+1), y(n+1);
77      for(int i = 0; i <= n; ++ i) x[i] = l + (r - l) / n * i, y[
            i] = f(x[i]);
78      type m0, mn;
79      if (mode == "Natural") m0 = mn = 0;
80      else if (mode == "Complete") m0 = f.d(l), mn = f.d(r);
81      else if (mode == "Specified_Second_Derivatives") m0 = f.d(l
```

```
        , 2), mn = f.d(r, 2);
    return CubicSpline<type>(x, y, mode, m0, mn);
}
```

另外附上解三对角方程组的程序（见 Thomas.h）：

```
#include <bits/stdc++.h>
using namespace std;

template <class type>
vector<type> Thomas(vector<type>& a, vector<type>& b, vector<
    type>& c, vector<type> y) {
    int n = a.size();
    if (b.size() != n-1 || c.size() != n-1) throw "Invalid Size
        !";
    vector<type> p(n), q(n-1);
    p[0] = a[0];
    for (int i = 1; i < n; ++ i) {
        q[i-1] = c[i-1] / p[i-1];
        p[i] = a[i] - b[i-1] * q[i-1];
    }
    y[0] = y[0] / p[0];
    for (int i = 1; i < n; ++ i)
        y[i] = (y[i] - b[i-1] * y[i-1]) / p[i];
    for (int i = n-2; i >= 0 ; -- i)
        y[i] = y[i] - q[i] * y[i+1];
    return y;
}
```

## 2.3 实例测试（A 题）

对于题中函数，简单计算可得其一二阶导数如下：

$$f'(x) = \frac{-50x}{(1+25x^2)^2} \tag{3}$$

$$f''(x) = \frac{-50+1250x^2}{(1+25x^2)^3} \tag{4}$$

然后直接调用对应函数，将插值结果输出到 csv 中，再用 python 读取并输出，测试线性样条和三次样条的拟合效果。

```cpp
#include <bits/stdc++.h>
#include "Spline.h"
#include "../HW1/function.h"
using namespace std;

class F : public Function<double> {
    virtual double operator () (const double& x) const {
        return 1 / (1 + 25*x*x*x);
    }
    virtual double d (const double& x, const int& k = 1) const {
        if (k == 1) return -50 * x / (1 + 25*x*x*x) / (1 + 25*x*x*x);
        else if (k == 2) return (-50 + 1250*x*x*x) / (1 + 25*x*x*x) / (1 + 25*x*x*x) / (1 + 25*x*x*x);
        else throw 0;
    }
} f;

int main() {
    LinearSpline<double> A1 = LinearSplineInterpolation<double>(f, -1.0, 1.0, 5);
    LinearSpline<double> A2 = LinearSplineInterpolation<double>(f, -1.0, 1.0, 10);
    LinearSpline<double> A3 = LinearSplineInterpolation<double>(f, -1.0, 1.0, 20);
    LinearSpline<double> A4 = LinearSplineInterpolation<double>(f, -1.0, 1.0, 40);
    LinearSpline<double> A5 = LinearSplineInterpolation<double>(f, -1.0, 1.0, 80);

    ofstream out1("a1.csv");
    out1 << "x,y" << '\n';
    for(int i = -999; i <= 999; ++ i) out1 << i * 0.001 << ',' << A1.GetValue(i * 0.001) << '\n';
    ofstream out2("a2.csv");
```

```
28      out2 << "x,y" << '\n';
29      for(int i = -999; i <= 999; ++ i) out2 << i * 0.001 << ','
            << A2.GetValue(i * 0.001) << '\n';
30      ofstream out3("a3.csv");
31      out3 << "x,y" << '\n';
32      for(int i = -999; i <= 999; ++ i) out3 << i * 0.001 << ','
            << A3.GetValue(i * 0.001) << '\n';
33      ofstream out4("a4.csv");
34      out4 << "x,y" << '\n';
35      for(int i = -999; i <= 999; ++ i) out4 << i * 0.001 << ','
            << A4.GetValue(i * 0.001) << '\n';
36      ofstream out5("a5.csv");
37      out5 << "x,y" << '\n';
38      for(int i = -999; i <= 999; ++ i) out5 << i * 0.001 << ','
            << A5.GetValue(i * 0.001) << '\n';
39
40      CubicSpline<double> A6 = CubicSplineInterpolation<double>(f
            , -1.0, 1.0, 5, "Complete");
41      CubicSpline<double> A7 = CubicSplineInterpolation<double>(f
            , -1.0, 1.0, 10, "Complete");
42      CubicSpline<double> A8 = CubicSplineInterpolation<double>(f
            , -1.0, 1.0, 20, "Complete");
43      CubicSpline<double> A9 = CubicSplineInterpolation<double>(f
            , -1.0, 1.0, 40, "Complete");
44      CubicSpline<double> A10 = CubicSplineInterpolation<double>(
            f, -1.0, 1.0, 80, "Complete");
45
46      ofstream out6("a6.csv");
47      out6 << "x,y" << '\n';
48      for(int i = -999; i <= 999; ++ i) out6 << i * 0.001 << ','
            << A6.GetValue(i * 0.001) << '\n';
49      ofstream out7("a7.csv");
50      out7 << "x,y" << '\n';
51      for(int i = -999; i <= 999; ++ i) out7 << i * 0.001 << ','
            << A7.GetValue(i * 0.001) << '\n';
52      ofstream out8("a8.csv");
53      out8 << "x,y" << '\n';
54      for(int i = -999; i <= 999; ++ i) out8 << i * 0.001 << ','
            << A8.GetValue(i * 0.001) << '\n';
```

```
55      ofstream out9("a9.csv");
56      out9 << "x,y" << '\n';
57      for(int i = -999; i <= 999; ++ i) out9 << i * 0.001 << ','
            << A9.GetValue(i * 0.001) << '\n';
58      ofstream out10("a10.csv");
59      out10 << "x,y" << '\n';
60      for(int i = -999; i <= 999; ++ i) out10 << i * 0.001 << ','
             << A10.GetValue(i * 0.001) << '\n';
61
62      CubicSpline<double> A11 = CubicSplineInterpolation<double>(
            f, -1.0, 1.0, 5, "Natural");
63      CubicSpline<double> A12 = CubicSplineInterpolation<double>(
            f, -1.0, 1.0, 10, "Natural");
64      CubicSpline<double> A13 = CubicSplineInterpolation<double>(
            f, -1.0, 1.0, 20, "Natural");
65      CubicSpline<double> A14 = CubicSplineInterpolation<double>(
            f, -1.0, 1.0, 40, "Natural");
66      CubicSpline<double> A15 = CubicSplineInterpolation<double>(
            f, -1.0, 1.0, 80, "Natural");
67
68      ofstream out11("a11.csv");
69      out11 << "x,y" << '\n';
70      for(int i = -999; i <= 999; ++ i) out11 << i * 0.001 << ','
             << A11.GetValue(i * 0.001) << '\n';
71      ofstream out12("a12.csv");
72      out12 << "x,y" << '\n';
73      for(int i = -999; i <= 999; ++ i) out12 << i * 0.001 << ','
             << A12.GetValue(i * 0.001) << '\n';
74      ofstream out13("a13.csv");
75      out13 << "x,y" << '\n';
76      for(int i = -999; i <= 999; ++ i) out13 << i * 0.001 << ','
             << A13.GetValue(i * 0.001) << '\n';
77      ofstream out14("a14.csv");
78      out14 << "x,y" << '\n';
79      for(int i = -999; i <= 999; ++ i) out14 << i * 0.001 << ','
             << A14.GetValue(i * 0.001) << '\n';
80      ofstream out15("a15.csv");
81      out15 << "x,y" << '\n';
82      for(int i = -999; i <= 999; ++ i) out15 << i * 0.001 << ','
```

```
        << A15.GetValue(i * 0.001) << '\n';

    CubicSpline<double> A16 = CubicSplineInterpolation<double>(
        f, -1.0, 1.0, 5, "Specified_Second_Derivatives");
    CubicSpline<double> A17 = CubicSplineInterpolation<double>(
        f, -1.0, 1.0, 10, "Specified_Second_Derivatives");
    CubicSpline<double> A18 = CubicSplineInterpolation<double>(
        f, -1.0, 1.0, 20, "Specified_Second_Derivatives");
    CubicSpline<double> A19 = CubicSplineInterpolation<double>(
        f, -1.0, 1.0, 40, "Specified_Second_Derivatives");
    CubicSpline<double> A20 = CubicSplineInterpolation<double>(
        f, -1.0, 1.0, 80, "Specified_Second_Derivatives");

    ofstream out16("a16.csv");
    out16 << "x,y" << '\n';
    for(int i = -999; i <= 999; ++ i) out16 << i * 0.001 << ','
         << A16.GetValue(i * 0.001) << '\n';
    ofstream out17("a17.csv");
    out17 << "x,y" << '\n';
    for(int i = -999; i <= 999; ++ i) out17 << i * 0.001 << ','
         << A17.GetValue(i * 0.001) << '\n';
    ofstream out18("a18.csv");
    out18 << "x,y" << '\n';
    for(int i = -999; i <= 999; ++ i) out18 << i * 0.001 << ','
         << A18.GetValue(i * 0.001) << '\n';
    ofstream out19("a19.csv");
    out19 << "x,y" << '\n';
    for(int i = -999; i <= 999; ++ i) out19 << i * 0.001 << ','
         << A19.GetValue(i * 0.001) << '\n';
    ofstream out20("a20.csv");
    out20 << "x,y" << '\n';
    for(int i = -999; i <= 999; ++ i) out20 << i * 0.001 << ','
         << A20.GetValue(i * 0.001) << '\n';

}
```

作图程序略，详见 `plot.ipynb`。

分段线性样条、自然三次样条、完全三次样条、端点处二阶导指定的三次样条的拟合效果见图 1、2、3、4。蓝色虚线为 $f(x)$，黄色、绿色、红色、

紫色和棕色分别为 $n = 5, 10, 20, 40, 80$ 的样条曲线。

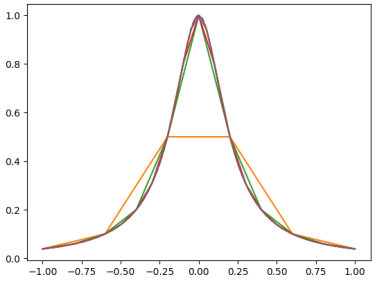


图 1: 线性

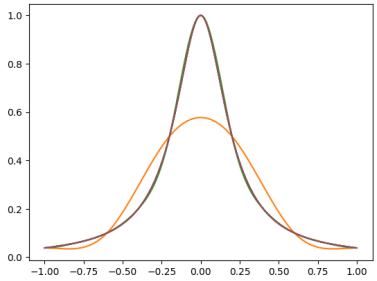


图 2: 自然

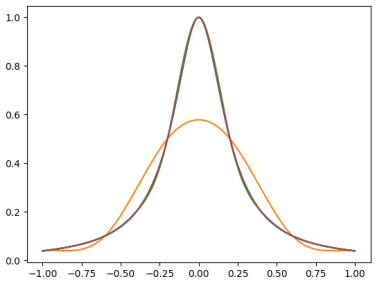


图 3: 完全

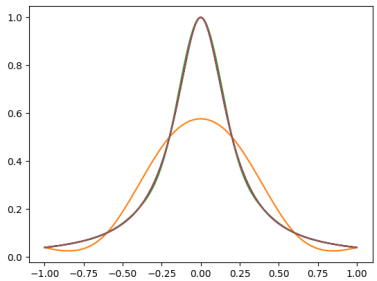


图 4: 端点处二阶导指定

分段样条的拟合偏差较大，但在 $n \geq 20$ 时基本无误差。其他三者的拟合在 $n \geq 10$ 及以上时基本无误差。

# 3 B-Spline 的理论及实现（B 题）

B-Spline 通过将样条函数分解成 $n + m$ 个 $m$ 阶 B-样条基函数的线性组合，可以解决结点为整数的插值问题。进而通过简单的平移和放缩变换也可以解决任意结点间隔相等的插值问题。多数情形下，我们只需要用到一阶、二阶和三阶 B-Spline。本项目仅支持结点为整数的 B-Spline 插值。

## 3.1 B-Spline 的基函数

经典 B-Spline 的基函数可通过递归定义。

$$B_{i,\mathbb{Z}}^{n+1}(x) = \frac{x - i + 1}{n + 1} B_{i,\mathbb{Z}}^{n}(x) + \frac{i + n + 1 - x}{n + 1} B_{i+1,\mathbb{Z}}^{n}(x) \tag{5}$$

当 $m=1,2,3$ 时，也可以直接写出它们的表达式。

$$B_{i,\mathbb{Z}}^{1}(x)=\begin{cases}x-i+1, & i-1\le x\le i\\ i+1-x, & i\le x\le i+1\\ 0, & \text{others}\end{cases} \tag{6}$$

$$B_{i,\mathbb{Z}}^{2}(x)=\begin{cases}\dfrac{(x-i+1)^2}{2}, & i-1\le x\le i\\ \dfrac{3}{4}-(x-(i+\dfrac{1}{2}))^2, & i\le x\le i+1\\ \dfrac{(i+2-x)^2}{2}, & i+1\le x\le i+2\\ 0, \text{others}\end{cases} \tag{7}$$

$$B_{i,\mathbb{Z}}^{3}(x)=\begin{cases}\dfrac{(x-i+1)^3}{6}, & i-1\le x\le i\\ \dfrac{2}{3}-\dfrac{1}{2}(x-i+1)(i+1-x)^2, & i\le x\le i+1\\ B_{i,\mathbb{Z}}^{3}(2i+2-x), & i+1\le x\le i+3\\ 0, & \text{others}\end{cases} \tag{8}$$

代码实现如下：

```
template <class type>
type B(const int& n, const int& i, const type& x) {
    if (n == 0) {
        if (i-1 < x && x <= i) return 1;
        else return 0;
    }
    // 根据 B 样条基的递归定义计算
    return (x-i+1) * B(n-1, i, x) / n + (i+n-x) * B(n-1, i+1, x
        ) / n;
}
```

## 3.2 用 B-Spline 实现分段线性样条

### 3.2.1 问题和理论基础

条件和 ppform 相同，但要求结点为连续整数 $l,l+1,\ldots,l+n$。

因为 $B_i^1(i)=1, B_i^1(j)=0, i\neq j$，所以

$$s(x)=\sum_{i=l}^{l+n} y_i B_i^1(i). \tag{9}$$

### 3.2.2 代码实现

设计 LinearBSpline 类。

- int 型私有成员变量 L,R：左右端点。
- vector<type> 型私有成员变量 coef：基函数的系数。
- 构造函数 LinearBSpline(const int& x0, const int& n, const vector<type>& f)：L=x0,R=x0+n,coef=f。
- type 型公有成员函数 GetValue(const type& _x)：获取样条函数在 _x 处的拟合值。用 STL 自带的upper_bound函数定位到代求点所在区间后，取左右两个基函数值求和即可。

```
template <class type>
class LinearBSpline{
private:
    int L, R;
    vector<type> coef;
public:
    LinearBSpline(const int& x0, const int& n, const vector<
        type>& f) : L(x0), coef(f) {
        R = L + n;
    }
    type GetValue(const type& _x) const{
        if (_x < L || _x > R) throw "Out of Range!";
        int i = floor(_x);
        type res = 0;
        if(i   >= L && i   <= R) res += coef[i  -L] * B(1, i,
              _x);
        if(i+1 >= L && i+1 <= R) res += coef[i+1-L] * B(1, i+1,
             _x);
        return res;
    }
```

```
};

template <class type>
LinearBSpline<type> LinearBSplineInterpolation(const Function <
    type>& f, const int& l, const int& n) {
    vector <type> y(n+1);
    for(int i = 0; i <= n; ++ i) y[i] = f(l+i);
    return LinearBSpline<type>(l, n, y);
}
```

## 3.3 用 B-Spline 实现分段三次样条

### 3.3.1 问题和理论基础

条件和 ppform 相同，但要求结点为连续整数 $l, l+1, \ldots, l+n$。

设样条为 $s(x) = \sum_{i=-2}^{n} a_i B_{i,\mathbb{Z}}^3(x)$。

对于自然样条和完全样条，根据讲义 Thm 3.57 可知，B-Spline 的系数满足方程组

$$\begin{bmatrix} 2 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{-1} \\ a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 3f(l) + f'(l) \\ 6f(l+1) \\ 6f(l+2) \\ \vdots \\ 6f(l+n-1) \\ 3f(l+n) - f'(l+n) \end{bmatrix}, \tag{10}$$

$$\begin{aligned} a_{-2} &= a_0 - 2f'(l), \\ a_n &= a_{n-2} + 2f'(l+n). \end{aligned}$$

对于端点处二阶导指定的样条，根据三阶 B-Spline 的解析式可得它在整点处的二阶导数值为

$$B_{i,\mathbb{Z}}^3(i) = 1, B_{i,\mathbb{Z}}^3(i+1) = -2, B_{i,\mathbb{Z}}^3(i+2) = 1. \tag{11}$$

因此有

$$f''(l) = a_{-2} - 2a_{-1} + a_0, f''(l+n) = a_{n-2} - 2a_{n-1} + a_n. \tag{12}$$

进而可得

$$6a_{-1} = 6f(l) - f''(l), 6a_{n-1} = 6f(l+n) - f''(l+n). \tag{13}$$

用这两个方程代替上面矩阵中的第一个和最后一个方程即可。进而可求出 $a_{-2}$ 和 $a_n$ 的值。

### 3.3.2 代码实现

设计类 `CubicBSpline`，

- `int` 型私有成员变量 `L,R`：左右端点。
- `vector<type>` 型私有成员变量 `coef`：基函数的系数。
- 构造函数 `CubicSpline(const int& x, const int& n, const vector <type>& f, const stri` 构造函数。mode 可选择 `Natural`、`Complete` 或 `Specified_Second_Derivatives`，`m0` 和 `mn` 是端点处的一阶或二阶导数值。具体实现时先解三对角方程组求得中间 $n+1$ 个系数，再根据边界方程求出边界两个系数。
- `type` 型公有成员函数 `GetValue(const type& _x)`：获取样条函数在 `_x` 处的拟合值。用 STL 自带的`upper_bound`函数定位到代求点所在区间后，取左右四个基函数值求和即可。

具体代码如下。

```
template <class type>
class CubicBSpline{
private:
    int L, R;
    vector <type> coef;
public:
    CubicBSpline(const int& x0, const int& n, const vector<type
        >& f, const string& mode = "Natural", const type& m0 =
        0, const type& mn = 0) : L(x0) {
        R = L + n;
        vector <type> a(n+1), b(n), c(n), y(n+1);
        // 构造三对角方程组的中间 n-1 行
        for (int i = 1; i < n; ++ i) {
            a[i] = 4;
```

```
13              b[i-1] = 1;
14              c[i] = 1;
15              y[i] = 6 * f[i];
16          }
17          // 根据边界条件构造首尾两行
18          if (mode == "Natural" || mode == "Complete") {
19              a[0] = 2, c[0] = 1, y[0] = 3 * f[0] + m0;
20              a[n] = 2, b[n-1] = 1, y[n] = 3 * f[n] - mn;
21              // 解出中间 n+1 个系数
22              vector <type> t = Thomas(a, b, c, y);
23              coef.resize(n+3);
24              for (int i = 0; i <= n; ++ i) coef[i+1] = t[i];
25              // 计算首尾两个系数
26              coef[0] = coef[2] - 2 * m0;
27              coef[n+2] = coef[n] + 2 * mn;
28          }
29          else if (mode == "Specified_Second_Derivatives") {
30              a[0] = 6, y[0] = 6 * f[0] - m0;
31              a[n] = 6, y[n] = 6 * f[n] - mn;
32              // 解出中间 n+1 个系数
33              vector <type> t = Thomas(a, b, c, y);
34              coef.resize(n+3);
35              for (int i = 0; i <= n; ++ i) coef[i+1] = t[i];
36              // 计算首尾两个系数
37              coef[0] = m0 - coef[2] + 2 * coef[1];
38              coef[n+2] = mn - coef[n] + 2 * coef[n-1];
39          }
40      }
41      type GetValue(const type& _x) const {
42          if (_x < L || _x > R) throw "Out of Range!";
43          int i = floor(_x);
44          type res = 0;
45          if(i-2 >= L-2 && i-2 <= R) res += coef[i-2-L+2] * B(3,
                i-2, _x);
46          if(i-1 >= L-2 && i-1 <= R) res += coef[i-1-L+2] * B(3,
                i-1, _x);
47          if(i   >= L-2 && i   <= R) res += coef[i  -L+2] * B(3,
                i  , _x);
48          if(i+1 >= L-2 && i+1 <= R) res += coef[i+1-L+2] * B(3,
```

```
            i+1, _x);
        return res;
    }
};

template <class type>
CubicBSpline<type> CubicBSplineInterpolation(const Function <
    type>& f, const int& l, const int& n, const string& mode =
    "Natural") {
    vector <type> y(n+1);
    for(int i = 0; i <= n; ++ i) y[i] = f(l+i);
    type m0, mn;
    if (mode == "Natural") m0 = mn = 0;
    else if (mode == "Complete") m0 = f.d(l), mn = f.d(l+n);
    else if (mode == "Specified_Second_Derivatives") m0 = f.d(l
        , 2), mn = f.d(l+n, 2);
    return CubicBSpline<type>(l, n, y, mode, m0, mn);
}
```

## 3.4 用 B-Spline 实现分段二次样条

Thm 3.58 还提供了用分段二次样条在仅知道函数在端点和 $l+\frac{1}{2}, l+\frac{3}{2}, \ldots, l+n-\frac{1}{2}$ 处的取值时的插值方法。这部分不是本项目的重点，因此仅给出类的设计代码。

设计 `QuadraticBSpline` 类。

- `int` 型私有成员变量 `L,R`：左右端点。
- `vector<type>` 型私有成员变量 `coef`：基函数的系数。
- 构造函数 `QuadraticBSpline(const int& x0, const int& n, const vector<type>& f, const` 解三对角方程组求得基函数的中间 $n$ 个系数，再根据边界条件求出边界两个系数。
- `type` 型公有成员函数 `GetValue(const type& _x)`：获取样条函数在 `_x` 处的拟合值。用 STL 自带的`upper_bound`函数定位到代求点所在区间后，取左右三个基函数值求和即可。

具体代码如下。

```
template <class type>
class QuadraticBSpline{
private:
    int L, R;
    vector <type> coef;
public:
    QuadraticBSpline(const int& x0, const int& n, const vector<
        type>& f, const type& f0, const type& fn) : L(x0) {
        R = L + n;
        vector <type> a(n), b(n-1), c(n-1), y(n);
        // 构造三对角方程组
        for (int i = 1; i < n-1; ++ i) {
            a[i] = 6;
            b[i-1] = 1;
            c[i] = 1;
            y[i] = 8 * f[i];
        }
        a[0] = 5, c[0] = 1, y[0] = 8 * f[0] - 2 * f0;
        a[n-1] = 5, b[n-2] = 1, y[n-1] = 8 * f[n-1] - 2 * fn;
        // 解出中间 n 个系数
        vector <type> t = Thomas(a, b, c, y);
        coef.resize(n+2);
        for (int i = 0; i < n; ++ i) coef[i+1] = t[i];
        // 计算首尾两个系数
        coef[0] = 2 * f0 - coef[1];
        coef[n+1] = 2 * fn - coef[n];
    }
    type GetValue(const type& _x) const {
        if (_x < L || _x > R) throw "Out of Range!";
        int i = floor(_x);
        type res = 0;
        if(i-1 >= L-1 && i-1 <= R) res += coef[i-1-L+1] * B(2,
            i-1, _x);
        if(i   >= L-1 && i   <= R) res += coef[i  -L+1] * B(2,
            i  , _x);
        if(i+1 >= L-1 && i+1 <= R) res += coef[i+1-L+1] * B(2,
            i+1, _x);
```

```
34          return res;
35      }
36  };
37
38  template <class type>
39  QuadraticBSpline<type> QuadraticBSplineInterpolation(const
        Function <type>& f, const int& l, const int& n) {
40      vector <type> y(n);
41      for(int i = 0; i < n; ++ i) y[i] = f(l+i+0.5);
42      type f0 = f(l), fn = f(l+n);
43      return QuadraticBSpline<type>(l, n, y, f0, fn);
44  }
```

### 3.5 实例测试（C、D 题）

对于题中函数，简单计算可得其一二阶导数如下：

$$f'(x) = \frac{-2x}{(1+x^2)^2} \tag{14}$$

$$f''(x) = \frac{-2+6x^2}{(1+x^2)^3} \tag{15}$$

然后直接调用对应函数，将插值结果输出到 `csv` 中，再用 python 读取并输出，测试线性样条、三次样条和二次样条的拟合效果。

```
1   #include <bits/stdc++.h>
2   #include "BSpline.h"
3   #include "../HW1/function.h"
4   using namespace std;
5
6   class F : public Function<double> {
7   public:
8       virtual double operator () (const double& x) const {
9           return 1 / (1 + x*x);
10      }
11      virtual double d (const double& x, const int& k) const {
12          if (k == 1) return -2*x / (1 + x*x) / (1 + x*x);
13          else if (k == 2) return (-2 + 6*x*x) / (1 + x*x) / (1 +
                x*x) / (1 + x*x);
```

```
14          else throw 0;
15      }
16  } f;
17
18  int main() {
19      LinearBSpline<double> CL = LinearBSplineInterpolation(f,
            -5, 10);
20      ofstream outL("c1.csv");
21      outL << "x,y" << endl;
22      for(int i = -4999; i <= 4999; ++ i) outL << i * 0.001 << ',
            ' << CL.GetValue(i * 0.001) << endl;
23
24      CubicBSpline<double> C1 = CubicBSplineInterpolation<double
            >(f, -5, 10, "Natural");
25      ofstream out1("c1.csv");
26      out1 << "x,y" << endl;
27      for(int i = -4999; i <= 4999; ++ i) out1 << i * 0.001 << ',
            ' << C1.GetValue(i * 0.001) << endl;
28
29      CubicBSpline<double> C2 = CubicBSplineInterpolation<double
            >(f, -5, 10, "Complete");
30      ofstream out2("c2.csv");
31      out2 << "x,y" << endl;
32      for(int i = -4999; i <= 4999; ++ i) out2 << i * 0.001 << ',
            ' << C2.GetValue(i * 0.001) << endl;
33
34      CubicBSpline<double> C3 = CubicBSplineInterpolation<double
            >(f, -5, 10, "Specified_Second_Derivatives");
35      ofstream out3("c3.csv");
36      out3 << "x,y" << endl;
37      for(int i = -4999; i <= 4999; ++ i) out3 << i * 0.001 << ',
            ' << C3.GetValue(i * 0.001) << endl;
38
39      QuadraticBSpline<double> CQ = QuadraticBSplineInterpolation
            <double>(f, -5, 10);
40      ofstream outQ("cq.csv");
41      outQ << "x,y" << endl;
42      for(int i = -4999; i <= 4999; ++ i) outQ << i * 0.001 << ',
            ' << CQ.GetValue(i * 0.001) << endl;
```

```
43
44      ofstream outd("d.csv");
45      vector<double> a = {-3.5,-3,-0.5,0,0.5,3,3.5};
46      outd << "a_i,ESL,ES1,ES2,ES3,ESQ" << endl;
47      for(int i = 0; i < 7; ++ i) outd << a[i] << ',' << fabs(CL.
            GetValue(a[i]) - f(a[i])) << ',' << fabs(C1.GetValue(a[
            i]) - f(a[i])) << ',' << fabs(C2.GetValue(a[i]) - f(a[i
            ])) << ',' << fabs(C3.GetValue(a[i]) - f(a[i])) << ','
            << fabs(CQ.GetValue(a[i]) - f(a[i])) << endl;
48  }
```

画图程序略，见 `plot.ipynb`。用上面五个样条插值的效果如下。

分段线性样条、自然三次样条、完全三次样条、端点处二阶导指定的三次样条、分段二次样条的拟合效果见图 5、6、7、8、9。蓝色虚线为 $f(x)$，黄色曲线为拟合曲线。

D 题的答案存入了 `d.csv` 中。如下表。

$E_L, E_C, E_N, E_S, E_Q$ 分别为分段线性样条、自然样条、完全样条、端点处二阶导指定的样条、分段二次样条的误差。

| $x$ | $E_L$ | $E_C$ | $E_N$ | $E_S$ | $E_Q$ |
|---|---|---|---|---|---|
| -3.5 | 0.00394007 | 4.14009e-05 | 0.000669568 | 0.00068675 | 2.87991e-18 |
| -3 | 5.54976e-18 | 5.54976e-18 | 5.54976e-18 | 5.54976e-18 | 0.00141838 |
| -0.5 | 0.05 | 0.02052 | 0.0205289 | 0.0205291 | 4.43981e-17 |
| 0 | 0 | 0 | 1.11022e-16 | 0 | 0.120238 |
| 0.5 | 0.05 | 0.02052 | 0.0205289 | 0.0205291 | 6.66242e-17 |
| 3 | 5.54976e-18 | 1.94275e-17 | 5.54976e-18 | 1.94275e-17 | 0.00141838 |
| 3.5 | 0.00394007 | 4.14009e-05 | 0.000669568 | 0.00068675 | 1.67577e-17 |

在线性样条和三次样条的结点（-3,0,3）处，$E_L, E_C, E_N, E_S$ 的值都为零或接近系统误差；在二次样条的结点（-3.5,-0.5,0.5,3,5）处，$E_Q$ 的值接近系统误差。而在线性样条和三次样条两个结点的中点处，线性样条和三次样条都有较大的误差，特别是在 $\pm 0.5$ 处分别达到了 0.05 和 0.02。在二次样条两个结点的中点处，二次函数则有较大误差，特别是在 0 处达到了 0.12。

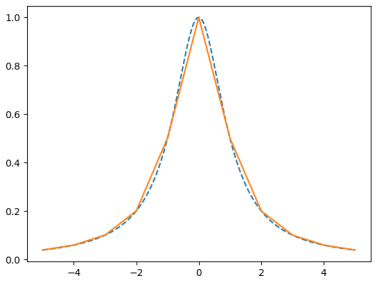


图 5: 线性

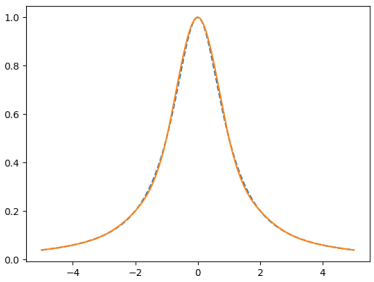


图 6: 自然

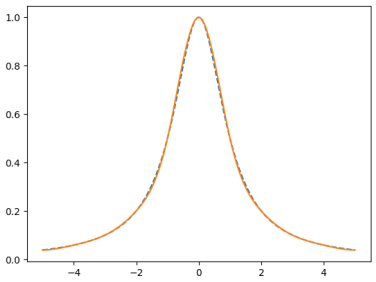


图 7: 完全

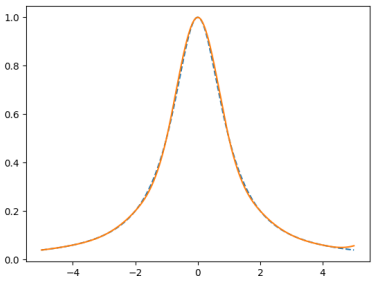


图 8: 端点处二阶导指定

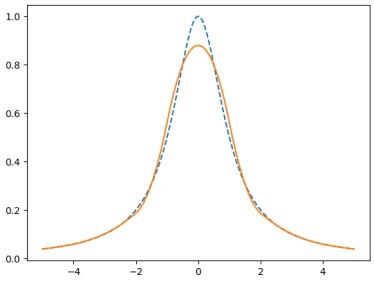


图 9: 二次

# 4 曲线拟合（E 题）

对于曲线拟合的问题,一般的处理方法是建立曲线的参数方程 $x(t), y(t)$, 然后对这两个函数分别进行拟合得到两个样条函数 $\hat{x}(t), \hat{y}(t)$。由于参数方程的建立方法多种多样，且实际应用时曲线会存在奇点（例如本题），因此很难对一般曲线拟合问题设计出通用的方法，只能具体问题具体分析。

## 4.1 心形曲线拟合

心形线曲线的方程为

$$x^2 + (\frac{3}{2}y - \sqrt{|x|})^2 = 3. \tag{16}$$

由极坐标可得

$$x(t) = \sqrt{3}sint, y(t) = \frac{2}{3}(\sqrt{3}\cos t + \sqrt{\sqrt{3}|\sin t|}) \tag{17}$$

这样就建立了心形曲线的参数方程。

按题目要求使用累计弧长参数，故将曲线重新参数化。

$$s(t) = \int_0^t \sqrt{x'(\tau)^2 + y'(\tau)^2}d\tau \tag{18}$$

但这个积分是没有原函数的，更不能显示解出 $t$。但注意到我们并不需要求出 $t$ 关于 $s$ 的表达式，只需求出 $s$ 取值为 $\frac{k}{n}L, k = 1, 2, \ldots, n-1$ 的值进而求出结点处的 $t$ 值（其中 $L$ 为总弧长），然后以这些 $t$ 为结点构造样条函数。

所以问题转化为求解 $n-1$ 个方程 $s(t) = \frac{k}{n}L$。它等价于

$$\int_{t_k}^{t_{k+1}} \sqrt{x'(\tau)^2 + y'(\tau)^2}d\tau = \frac{k}{n}L \tag{19}$$

这是一个非线性方程。因为 $s$ 的导数难以求出，故可以使用第一章的二分法或割线法求解（本文使用二分法）。

最后一个问题就是 $s(t)$ 是一个不能直接计算的积分，因此我们利用第六章的辛普森积分计算。但被积分式在积分边界处是奇异的，直接用二次函数拟合效果很差，即使用复化辛普森公式分段收敛速度也非常慢。因此采用自适应辛普森积分，即当 $|I_{l,r}^S(s) - (I_{l,\frac{l+r}{2}}^S(s) + I_{\frac{l+r}{2},r}^S(s)| < \varepsilon$ 时直接取 $I_{l,r}^S(s)$ 作为 $(l, r)$ 上积分的近似，否则取中点，两边分别继续分治计算。

分别用分段线性样条和自然样条进行拟合（代码见 E.cpp）：

```cpp
#include <bits/stdc++.h>
#include "Spline.h"
#include "../HW1/function.h"
using namespace std;

const double PI = acos(-1), Q = sqrt(3), dt = 1e-8, eps = 1e-12;

class X : public Function<double> {
public:
    virtual double operator () (const double& t) const {
        return Q * sin(t);
    }
    virtual double d(const double& t) const {
        return Q * cos(t);
    }
} x;

class Y : public Function<double>{
public:
    virtual double operator () (const double& t) const {
        return 2 * (Q * cos(t) + sqrt(Q * fabs(sin(t)))) / 3;
    }
    virtual double d(const double& t) const {
        if (sin(t) > 0) return 2 * (-Q * sin(t) + sqrt(Q) * cos(t) / 2 / sqrt(fabs(sin(t)))) / 3;
        else return 2 * (-Q * sin(t) - sqrt(Q) * cos(t) / sqrt(fabs(sin(t)))) / 3;
    }
} y;

class S : public Function<double>{
public:
    virtual double operator () (const double& t) const {
        return sqrt(x.d(t) * x.d(t) + y.d(t) * y.d(t));
    }
} s;
```

```
35
36 // (l, r) 上的辛普森积分
37 double IS(const Function<double>& f, const double& l, const
       double& r) {
38     return (r-l) * (f(l) + 4 * f((l+r)/2) + f(r)) / 6;
39 }
40 // 自适应辛普森积分
41 double Simpson(const Function<double>& f, const double& l,
       const double& r) {
42     double mid = (l+r) / 2, s = IS(f, l, r), t = IS(f, l, mid)
           + IS(f, mid, r);
43     if (fabs(s-t) < eps) return s;
44     return Simpson(f, l, mid) + Simpson(f, mid, r);
45 }
46
47 // 求出曲线的 n 等分点
48 vector<double> Get_Knots(int n) {
49     double I = Simpson(s, dt, PI - dt) * 2;
50     vector<double> x(n+1), y(n+1);
51     x[0] = dt;
52     // 在曲线上按长度均匀取插值结点
53     for (int i = 1; i <= n/2-1; ++ i) {
54         double l = x[i-1], r = 2*PI;
55         while(r-l>=1e-8){
56             double mid = (l+r) / 2;
57             double Im = Simpson(s, x[i-1], mid);
58             if (Im < I/n) l = mid;
59             else r = mid;
60         }
61         x[i] = l;
62     }
63     x[n/2] = PI - dt;
64     for (int i = 0; i <= n/2-1; ++ i) {
65         x[n-i] = 2*PI - x[i];
66     }
67     return x;
68 }
69
70 // 用线性样条拟合曲线
```

```
71  // file 曲线数据的文件名
72  void Linear_heart_plot(int n, int m, const char* file) {
73      vector<double> t = Get_Knots(n), f(n+1), g(n+1);
74      for (int i = 0; i <= n; ++ i) f[i] = x(t[i]);
75      for (int i = 0; i <= n; ++ i) g[i] = y(t[i]);
76      LinearSpline<double> X = LinearSpline<double>(t, f);
77      LinearSpline<double> Y = LinearSpline<double>(t, g);
78      ofstream out(file);
79      out << "x,y" << '\n';
80      for (int i = 0; i <= m; ++ i) out << X.GetValue(dt+(2*PI-2*
            dt)*i/m) << ',' << Y.GetValue(dt+(2*PI-2*dt)*i/m) << '\
            n';
81  }
82
83  // 用三次样条拟合曲线（这里只实现了自然三次样条）
84  void Cubic_heart_plot(int n, int m, const string& mode, const
        char* file) {
85      vector<double> t = Get_Knots(n), f(n+1), g(n+1);
86      for (int i = 0; i <= n; ++ i) f[i] = x(t[i]);
87      for (int i = 0; i <= n; ++ i) g[i] = y(t[i]);
88      double m0 = 0, mn = 0;
89      CubicSpline<double> X = CubicSpline<double>(t, f, mode, m0,
             mn);
90      CubicSpline<double> Y = CubicSpline<double>(t, g, mode, m0,
             mn);
91      ofstream out(file);
92      out << "x,y" << '\n';
93      for (int i = 0; i <= m; ++ i) out << X.GetValue(dt+(2*PI-2*
            dt)*i/m) << ',' << Y.GetValue(dt+(2*PI-2*dt)*i/m) << '\
            n';
94  }
95
96  int main() {
97      Linear_heart_plot(10, 50000, "e1.csv");
98      Linear_heart_plot(40, 50000, "e2.csv");
99      Linear_heart_plot(160, 50000, "e3.csv");
100     Cubic_heart_plot(10, 50000, "Natural", "e4.csv");
101     Cubic_heart_plot(40, 50000, "Natural", "e5.csv");
102     Cubic_heart_plot(160, 50000, "Natural", "e6.csv");
```

```
103 }
```

拟合效果如下。线性和自然样条见图 10 和图 11。蓝色、黄色、绿色的曲线分别是拟合点数为 $n = 10, 40, 160$ 的曲线：

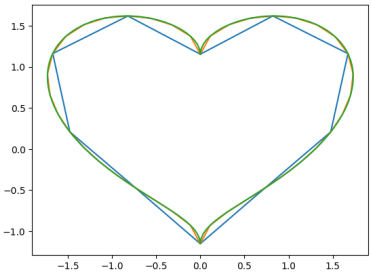


图 10: 线性

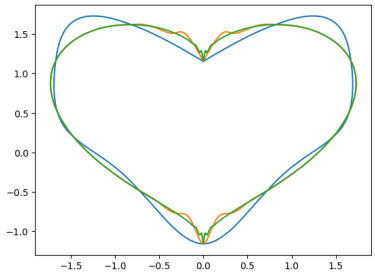


图 11: 自然

$n = 10$ 和 $n = 40$ 时，两种样条的表现都很差；$n = 160$ 时，样条在远离 $x = 0$ 的区域拟合效果很好，但在 $x = 0$ 附近拟合效果很差。这是因为在 $x = 0$ 附近 $y$ 的导数趋向于无穷大，拟合问题本身的条件数很大，不能期望算法的拟合效果很好。

需要注意的是，对于心形曲线，不能直接用完全样条或端点处二阶导指定的样条拟合，因为它们需要取到 $t = 0$ 处的导数值，而此时 $|y'(t)| \to \infty$。当然，注意到本题拟合的曲线是周期的，因此可任选其他的端点（例如 $\frac{\pi}{2}$），这样得到的结果和自然样条是类似的。但这样就失去了完全样条的优势（固定端点的斜率），所以这里不做实现。