数值分析项目作业报告

刘陈若 3200104872
信息与计算科学 2001

1 程序编译和运行说明

本次项目作业采用 Makefile 文件对编译进行统一管理。具体地，在 Makefile 所在目录下输入`make`即可完成编译，得到`report.pdf`，problemB的可执行文件`B`，problemC和 problemD共同的可执行文件`CD`，以及 problemE的可执行文件`E`。分别对其进行运行，即可得到各小题的程序运行结果。

需要说明两点：首先，本项目作业使用 eigen3 进行线性方程组求解，并使用 json file 进行参数输入，它们分别以`#include <eigen3/Eigen/...>`和`#include <jsoncpp/json/json.h>`的形式被调用，因此在编译时需要保持相应的文件关系；其次，报告使用 json file 进行参数输入时，参数的含义将在下文中详细给出，以便完成报告中的所有测试。

2 程序运行结果及简要分析

2.1 spline.h 设计思路

在`spline.h`头文件中，一共设计了`Function`，`Interpolation`两个基类，派生出不同的 class 以满足项目要求。以下将分别展开设计思路叙述。

2.1.1 class Function

`Function`类一共设计了`operator()`（括号重载），`diff`和`diff_2`三个虚函数，用于返回给定函数的值以及一二阶导数的值。对于无需求导数的函数，其导数默认为 0。总体而言，`Function`类主要功能有三：存储待样条插值的函数，存储 B 样条基函数以及存储 ppForm 插值后得到的多项式函数。

- `class Discrete_function`：对于待样条插值的函数，若给出了函数的解析表达式，直接派生相应的函数类即可；若没有给出函数的解析表达式，也就是以离散点集给出的函数，还设计了派生类`class Discrete_function`进行相应的处理。由此，有无显示表达式的函数得到了形式上的统一，这在曲线拟合中起到了重要作用。
- `class B_spline`：对于 B 样条基函数，设计了派生类`class B_spline`，利用 **Definition 3.23** 递归返回样条基函数的值：如果阶数为 1 则直接利用解析表达式返回值，否则利用课本式 (3.30) 递归。
- `class Polynomial`：由于进行 ppForm 插值后得到的是许多个分段多项式，因此还设计了派生类`class Polynomial`，存储形如 $a_0 + a_1(x-a) + \cdots + a_n(x-a)^n$ 的多项式（多项式的“分段”任务将在之后的 interpolation 类中完成）。

2.1.2 class Interpolation

`Interpolation`类主要功能是为了实现基于 ppForm 和 B 样条法的线性样条及 $\mathbb{S}_3^2$ 样条（以下简称三次样条）插值，并基于此完成曲线插值。`Interpolation`类一共设计了`solve`和`operator()`（括号重载）两个虚函数，分别用于插值算法求解以及返回求解后插值函数在某一点处的值。

• `class Bspline_interpolation`：对于 B 样条插值，设计了该派生类以支持线性样条和三次样条插值，其中三次样条插值支持 complete cubic spline（以下简称 CCS），cubic spline with specified second derivatives at its end points（以下简称 CS-SSD）和 natural cubic spline（以下简称 NCS）三种边界条件。

给定不一定等间距的插值点 $t_1,\dots,t_N$，待插值函数 $f(x)$，以及插值阶数和边界条件，`solve`函数利用 eigen3 的稀疏矩阵 LU 分解求解线性方程组 $Ax=b$ 得到 B 样条基函数的对应系数 x 从而求得插值函数 $S(x)$。经过数学推导，以下给出各种情况对应的矩阵 A 和向量 b 的具体表达式。由于推导过程非常繁琐，此处只给出最终结果。

对于线性样条，在添加额外的插值点 t_0,t_{N+1}（理论上是任意的，本算法采用等间隔添加，即使得 $t_1-t_0=t_2-t_1$，$t_{N+1}-t_N=t_N-t_{N-1}$）后可设

$$S(x)=\sum_{i=1}^{N}a_iB_i^1(x),\quad x^T=[a_1,\dots,a_N], \tag{1}$$

则有

$$A=I_N,\quad b^T=[f(t_1),\dots,f(t_N)]. \tag{2}$$

对于三次样条，在添加额外的插值点 t_{-2} 至 t_0,t_{N+1} 至 t_{N+3}（方法同上）后可设

$$S(x)=\sum_{i=-1}^{N}a_iB_i^3(x),\quad x^T=[a_{-1},\dots,a_N], \tag{3}$$

并记

$$A=\begin{bmatrix} m_{1,1} & -m_{1,1}-m_{1,3} & m_{1,3} & & & \\ B_{-1}^3(t_1) & B_0^3(t_1) & B_1^3(t_1) & & & \\ & \ddots & \ddots & \ddots & & \\ & & B_{N-2}^3(t_N) & B_{N-1}^3(t_N) & B_N^3(t_N) & \\ & & m_{N+2,N} & -m_{N+2,N}-m_{N+2,N+2} & m_{N+2,N+2} \end{bmatrix},\quad b=\begin{bmatrix} b_1 \\ f(t_1) \\ \vdots \\ f(t_N) \\ b_{N+2} \end{bmatrix}$$

则对 CCS 有 $b_1 = f'(t_1)$，$b_{N+2} = f'(t_N)$，

$$m_{1,1} = -\frac{3B_0^2(t_1)}{t_2 - t_{-1}}, \quad m_{1,3} = \frac{3B_1^2(t_1)}{t_3 - t_0}, \quad m_{N+2,N} = -\frac{3B_{N-1}^2(t_N)}{t_{N+1} - t_{N-2}}, \quad m_{N+2,N+2} = \frac{3B_N^2(t_N)}{t_{N+2} - t_{N-1}}; \tag{4}$$

对 CS-SSD 有 $b_1 = f''(t_1)$，$b_{N+2} = f''(t_N)$，

$$\begin{aligned} m_{1,1} &= \frac{6}{(t_2 - t_{-1})(t_2 - t_0)}, \quad m_{1,3} = \frac{6}{(t_3 - t_0)(t_2 - t_0)}, \\ m_{N+2,N} &= \frac{6}{(t_{N+1} - t_{N-2})(t_{N+1} - t_{N-1})}, \quad m_{N+2,N+2} = \frac{6}{(t_{N+2} - t_{N-1})(t_{N+1} - t_{N-1})}; \end{aligned} \tag{5}$$

对 NCS 有 $b_1 = 0$，$b_{N+2} = 0$，

$$\begin{aligned} m_{1,1} &= \frac{6}{(t_2 - t_{-1})(t_2 - t_0)}, \quad m_{1,3} = \frac{6}{(t_3 - t_0)(t_2 - t_0)}, \\ m_{N+2,N} &= \frac{6}{(t_{N+1} - t_{N-2})(t_{N+1} - t_{N-1})}, \quad m_{N+2,N+2} = \frac{6}{(t_{N+2} - t_{N-1})(t_{N+1} - t_{N-1})}. \end{aligned} \tag{6}$$

由此可知，对于任一种情况此时的 A 和 b 都是已知的，从而解得对应的 x，进而唯一确定插值函数 S 完成插值过程。

进一步，`operator()`可以返回插值函数 S 在任一点的函数值，如有需要，`Get_coef`函数则用于返回系数 x（一般不使用）。需要注意的是，本算法使用`std::vector<B_spline>`存储全体 B 样条基函数至 vector 中以避免重复计算，并且在`operator()`函数中仅考虑函数值不为 0 的部分基函数进行累加以提高算法运算效率。

• `class ppForm_interpolation`：对于 ppForm 样条插值，设计了该派生类以支持线性样条和三次样条插值，其中三次样条插值同样支持 CCS，CS-SSD 和 NCS 三种边界条件。

给定不一定等间距的插值点 $t_1, \ldots, t_N$，待插值函数 $f(x)$，以及插值阶数和边界条件，设插值函数为 $S(x)$，并记 $p_i(x) = s|_{[t_i, t_{i+1}]}$，则对于线性样条插值，`solve`函数直接求解得到

$$p_i(x) = \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} x + f(t_{i+1}) - t_i \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i}. \tag{7}$$

对于三次样条插值，`solve`函数利用 eigen3 的稀疏矩阵 LU 分解求解类似 **Theorem 3.7** 的线性方程组 $Ax = b$ 得插值点处的导数值 x 从而求得插值函数 $S(x)$。经过数学推导，以下给出不同边界条件对应的矩阵 A 和向量 b 的具体表达式。

沿用 **Lemma 3.3** 的记号，设 $x^T = [m_1, \ldots, m_N]$，那么对 CCS 有

$$A = \begin{bmatrix} 1 & & & & \\ \lambda_2 & 2 & \mu_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \lambda_{N-1} & 2 & \mu_{N-1} \\ & & & & 1 \end{bmatrix}, \quad b = \begin{bmatrix} f'(t_1) \\ 3\mu_2 f[t_2, t_3] + 3\lambda_2 f[t_1, t_2] \\ \vdots \\ 3\mu_{N-1} f[t_{N-1}, t_N] + 3\lambda_{N-1} f[t_{N-2}, t_{N-1}] \\ f'(t_N) \end{bmatrix};$$

对 CS-SSD 有

$$A = \begin{bmatrix} 4 & 2 & & & \\ \lambda_2 & 2 & \mu_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \lambda_{N-1} & 2 & \mu_{N-1} \\ & & & 2 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 6f[t_1, t_2] - f''(t_1)(t_2 - t_1) \\ 3\mu_2 f[t_2, t_3] + 3\lambda_2 f[t_1, t_2] \\ \vdots \\ 3\mu_{N-1} f[t_{N-1}, t_N] + 3\lambda_{N-1} f[t_{N-2}, t_{N-1}] \\ 6f[t_{N-1}, t_N] + f''(t_N)(t_N - t_{N-1}) \end{bmatrix};$$

对 NCS 有

$$A = \begin{bmatrix} 4 & 2 & & & \\ \lambda_2 & 2 & \mu_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \lambda_{N-1} & 2 & \mu_{N-1} \\ & & & 2 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 6f[t_1, t_2] \\ 3\mu_2 f[t_2, t_3] + 3\lambda_2 f[t_1, t_2] \\ \vdots \\ 3\mu_{N-1} f[t_{N-1}, t_N] + 3\lambda_{N-1} f[t_{N-2}, t_{N-1}] \\ 6f[t_{N-1}, t_N] \end{bmatrix}.$$

由此可知，对于任一种情况此时的 A 和 b 都是已知的，从而解得对应的 x，进而根据课本公式 (3.6) 即可确定每一个 $p_i(x)$，从而唯一确定 $S(x)$ 完成插值过程。

进一步，`operator()`可以返回插值函数 S 在任一点的函数值，如有需要，`Get_coef`函数则用于返回每一个分段多项式的系数（一般不使用）。需要注意的是，本算法使用`std::vector<Polynomial>`存储插值得到的 $N-1$ 个分段多项式，且在`operator()`函数中寻找待求值点所在区间的多项式并返回函数值，由此实现了多项式的“分段”。

• `class curve_spline`：曲线样条插值建立函数样条插值的基础上，支持 ppForm 和 B 样条插值两种算法，且支持线性和三次样条插值。由于曲线常常是以隐函数的形式给出，本程序以曲线上点集的形式进行输入。根据 **Algorithm 3.72**，`solve`函数首先计算 cumulative chordal lengths，并将点集的每一个维度分量视为 cumulative chordal lengths 的函数，由此进行给定的样条插值，得到最终结果。

`Get_Point`函数用于返回曲线插值结果关于 cumulative chordal lengths 等间隔采样得到的插值曲

线点，以便作图分析。需要说明的是，由于曲线插值结果是关于 cumulative chordal lengths 的参数方程，一般无法得到消参后的方程，故可以采用`Get_Point`函数的形式返回最终的插值结果。

2.2 Problem A

本题中，将对函数 f 运行所有设计的插值算法（ppForm 和 Bspline 算法下的线性和三次样条插值），对结果给出可视化的展示，并进行样条插值算法收敛阶的分析。

2.2.1 A.json 参数说明

在 A 题中，需要对函数 $f(x)=\frac{1}{1+25x^2}$ 进行大量测试，为此您可能需要修改`A.json`的参数。因此本节首先给出`A.json`中各参数的含义说明。

- `interval_left`：插值区间左端点。本题中无需修改。
- `interval_right`：插值区间右端点。本题中无需修改。
- `n`：插值点个数表。
- `method`：输入 1，2, 或 3, 代表插值边界条件。三次样条 1 表示 complete cubic spline，2 表示 cubic spline with specified second derivatives, 3 表示 natural cubic spline；线性样条输入 1-3 不作区分。
- `order`：样条阶数。输入 1 或 3，分别代表线性和三次样条。

2.2.2 插值结果展示

• **ppForm**：使用 ppForm 进行样条插值（在`ProblemA.cpp`第 10 行修改`Spline_Form`的定义即可切换 ppForm 和 B 样条之间插值方法）。首先测试 complete cubic spline，设置`A.json`中`method`为 1,`order`为 3，运行程序得到不同插值点个数对应的 $f(x)$ 插值结果，输出至`ProblemA_result_points.txt`，并使用 python 作图如下所示。由于不同插值点个数对应的图像非常相近，因此还给出了局部放大图。

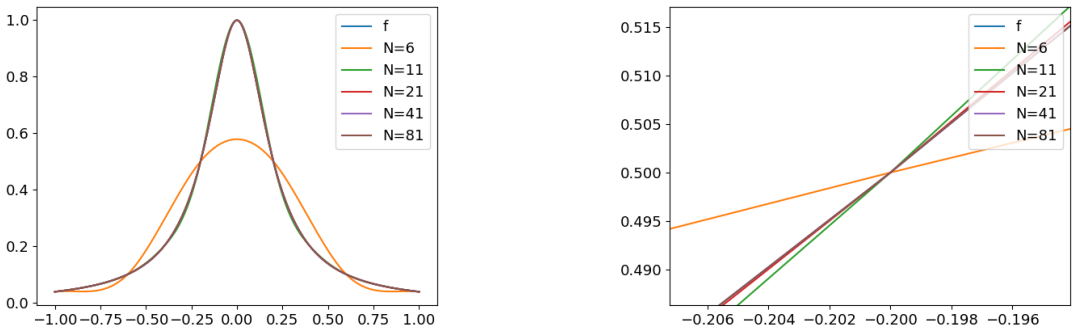

图 1: ppForm CCS 插值图像及其局部放大图

其次，改变`A.json`中`method`为 2 和 3，分别测试 cubic spline with specified second derivatives 和 natural cubic spline 相应的插值结果，使用 python 作图如图。可以看出三次样条插值无论何种边界条件，都具有较好的效果，且避免了 Runge 现象。

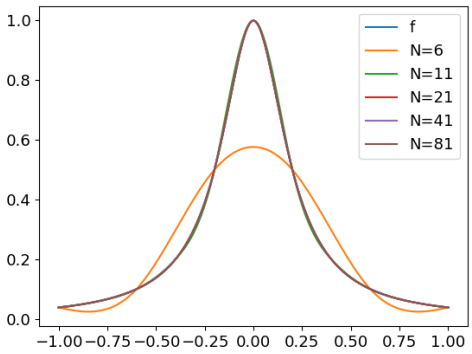

图 2: ppForm CS-SSD 插值图像

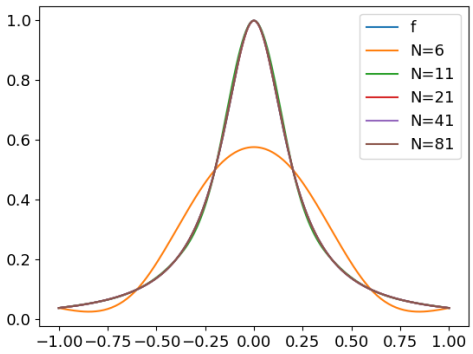

图 3: ppForm NCS 插值图像

最后，测试线性样条插值的结果。修改`A.json`中`order`为 1，得到线性样条插值结果，运行程序并绘制结果图像如下。

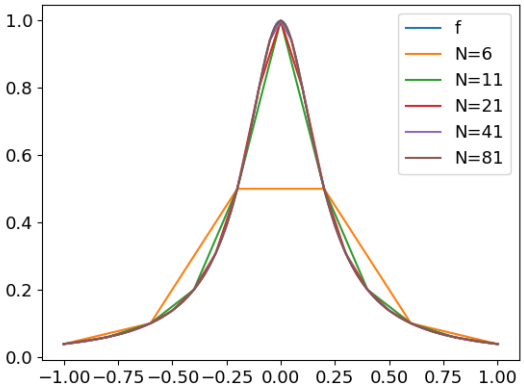

图 4: ppForm 线性样条插值图像

• **Bspline**：使用 B 样条基函数进行插值（在`ProblemA.cpp`第 10 行修改`Spline_Form`的定义即可切换 ppForm 和 B 样条之间插值方法）。与 ppForm 的操作方式相同，仍然绘制 complete cubic spline，cubic spline with specified second derivatives，natural cubic spline 和线性样条插值的结果图像，使用

python 展示如下所示。

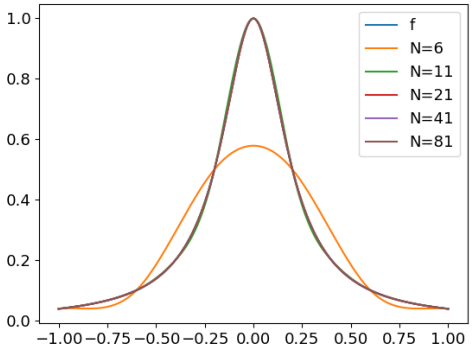

图 5: Bspline CCS 插值图像

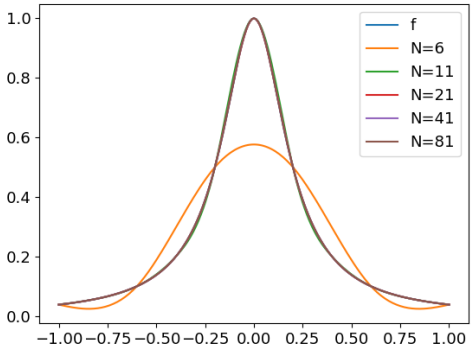

图 6: Bspline CS-SSD 插值图像

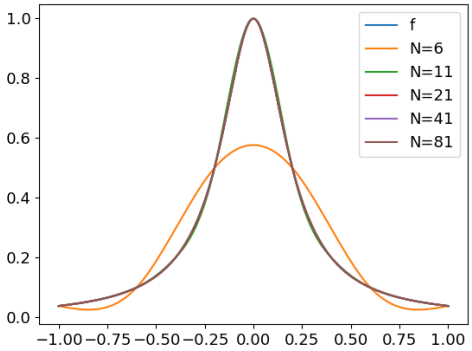

图 7: Bspline NCS 插值图像

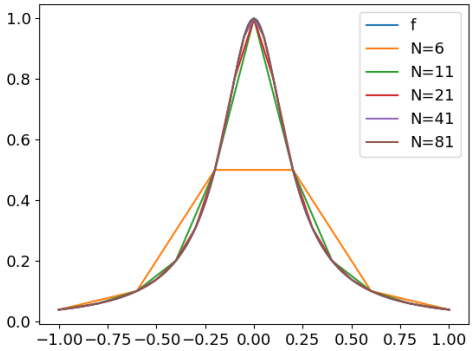

图 8: Bspline 线性样条插值图像

事实上，通过输出文件`ProblemA_result_points.txt`的比对，可以发现 ppForm 和 Bspline 两种方法得到的结果是几乎一样的，这也从侧面说明了`spline.h`中两种算法的正确性和结果的精确性。

2.2.3 收敛阶分析

根据题目要求，对于 CCS，CS-SSD，NCS 和线性样条四种情况，分别记录不同插值点个数 n 所对应的子区间中点误差最大值，输出至`ProblemA_result_error.txt`中。根据上一节分析，ppForm 和 Bspline 两种方法结果相同，故此处以 ppForm 为例进行分析。对于固定的 n，对三次样条的三种边界条件的最大误差取平均值，与线性样条一并绘制最大误差和插值点个数的关系图如下。

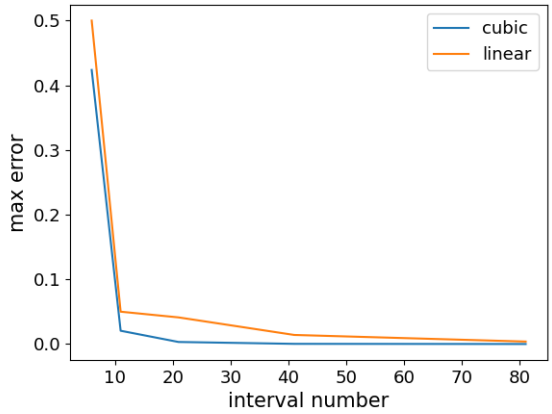

图 9: 最大插值误差和插值点个数关系图

为了对线性样条和三次样条进行收敛阶分析，我们需要对获得的最大误差进行进一步处理。首先，修改`A.json`中`n`为`[6,11,21,41,81,161,321,641,1281]`，增加插值点个数以便观察趋势；其次，绘制最大误差序列前一项与后一项的比值，以便观察插值点个数翻倍之后最大误差缩减的比例。基于此，根据得到的数据绘制图像如图。

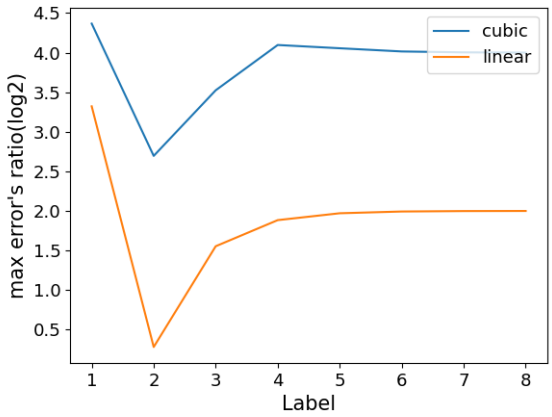

图 10: 最大插值误差比值图（以 2 为底）

图中 Label 为 1 对应的数值指的是 $n = 6$ 和 $n = 11$ 对应的最大误差的比值进行以 2 为底的对数规范化处理后的结果，以此类推。从图像中可以看出，随着插值点个数 n 的增大，三次样条的最大误差比值（以 2 为底）趋向于 4，线性样条的最大误差比值（以 2 为底）趋向于 2。换言之，当 n 增大一倍

时，三次样条的最大误差缩小至原来的十六分之一，而线性样条的最大误差缩小至原来的四分之一。

综上所述，可以得到三次样条的收敛阶为 4，线性样条的收敛阶为 2。

2.3 Problem C and Problem D

由于问题 B 已经在头文件中得到更一般化的实现，故不再单独列出；问题 C 和问题 D 是针对同一个函数进行的分析，因此在本报告将其一起进行展示。另外，经向胡双助教确认，对应的二次样条只需实现线性样条进行对应分析即可，故在以下出现的二次样条部分均由线性样条进行替代。

2.3.1 CD.json 参数说明

以下给出`CD.json`中各参数的含义说明。

- `point_1`：三次 Bspline 插值点。在误差分析的补充中需要修改。
- `point_2`：线性 Bspline 插值点。本题中无需修改。
- `output_point`：误差函数需要输出值的点。本题中无需修改。

2.3.2 插值结果展示

对函数

$$f(x) = \frac{1}{1+x^2} \tag{8}$$

分别根据`CD.json`中给定的插值点使用三次 B 样条插值（根据题目要求，使用 complete cubic spline）和线性 B 样条插值，得到结果输出至`ProblemC_result_points.txt`，利用 python 绘制结果图像和实际函数图像如图所示。

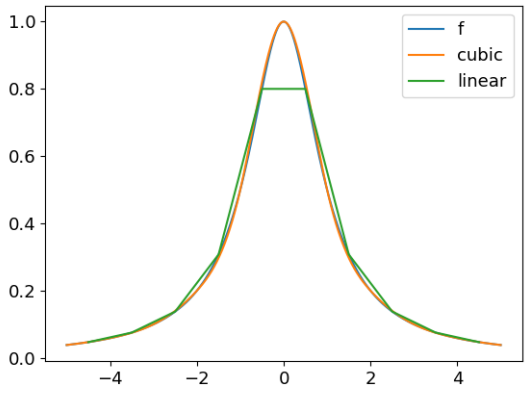

图 11: Bspline CCS 和线性样条插值图像

从图像可以看出，三次样条插值相较于线性样条插值，对于 f 的拟合结果更加精确。具体的误差分析将在下一节中展开叙述。

2.3.3 误差展示和分析

对于 C 题中得到的两个插值函数 S, 分别计算误差函数 $E_S(x) = |S(x) - f(x)|$，将结果采样输出至`ProblemD_result_errorpoints.txt`中，并基于此绘图如下。

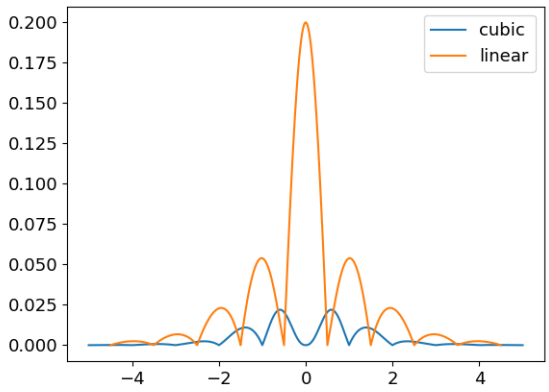

图 12: Bspline CCS 和线性样条插值误差图像

从图像中可以清晰地看出三次样条的误差要小于线性样条插值，故三次 Bspline 的结果更加精确。进一步，根据题目要求，输出三次样条和线性样条在 $x = -3.5, -3, -0.5, 0, 0.5, 3, 3.5$ 处的误差函数值于`ProblemD_result_error.txt`，展示如下：

```
Error values of cubic spline at required points
0.000669568, 0, 0.0205289, 0, 0.0205289, 0, 0.000669568
Error values of linear spline at required points
0, 0.00670137, 0, 0.2, 0, 0.00670137, 0
```

可以发现，部分 x 处的误差为 0。仔细观察，发现误差接近 0 对应的 x 恰巧是插值结点，它们的误差很小是因为样条插值要求节点处的插值函数值等于原函数的值，因此在求解精确度高的条件下，插值结点的误差应当接近 0。这也可以从图像中看出：误差函数在插值结点的误差很小, 而在每一个插值子区间的中点处取到极大值。

2.3.4 误差分析补充

第四章理论题 XIII 中提到，如果某两个插值点间距远小于其他插值点，会造成结果不准确。在此给出对其的验证。

首先将CD.json中point_1修改为

```
"point_1" : [-5,-3.000001,-2.999999,-1,1,3,5]
```

显然，-3 附近的两个插值点的间距要远小于其余各插值点的间距。然后再扩大该插值点间距，将point_1修改（作为对照）为

```
"point_1" : [-5,-3.1,-2.9,-1,1,3,5]
```

对于以上两组插值点，分别计算其误差函数 $E_S(x)$，绘制其在 $[-3,-1]$ 区间附近的放大图像如下。

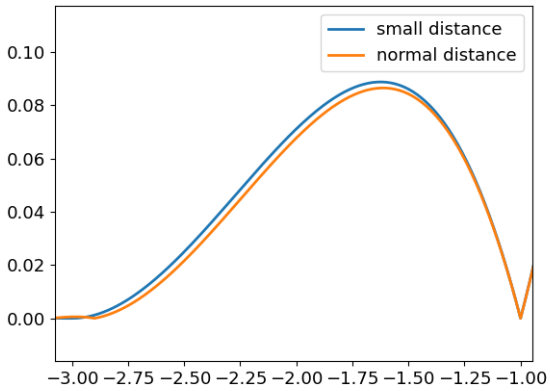

图 13: 不同插值点间距对应插值误差对比图（局部）

由图像可知，若某两点的间距远小于其他插值点间距时，对应的插值误差会略有增大但不明显。插值误差会略有增大验证了第四章对应理论的正确性，而误差增大不明显可能是因为算法和计算的精度都相对较高。

2.4 Problem E

本题中，对于心形曲线

$$x^2 + (\frac{3}{2}y - \sqrt{|x|})^2 = 3, \tag{9}$$

程序将根据不同的插值点个数，分别使用三次和线性样条进行曲线插值，并对结果给出可视化的展示。

2.4.1 E.json 参数说明

以下给出`E.json`中各参数的含义说明。

- `n`：插值点个数表。本题中无需修改。
- `order`：样条阶数。输入 1 或 3，分别代表线性和三次样条。
- `spline_form`：插值方式。输入`"Bspline"`或者`"PP"`，分别代表 Bspline 插值法和 ppForm 插值。

本报告的结果**默认**使用 Bspline 插值，如需改成 ppForm 可进行相应修改，也可得到相同的结果。

2.4.2 思考题

- **样条组数**：事实上，`spline.h`可以自动实现任意维度的曲线样条插值。对于本题的而言，心形曲线是二维的，因此需要两组样条进行插值。
- **边界条件**：对于线性样条，无需边界条件；对于三次样条，我认为 natural cubic spline 的边界条件是最合适的。曲线插值的自变量是 cumulative chordal lengths，而根据定义，cumulative chordal lengths 依赖于插值点的选取（且本身就是离散形式的），因此一般是无法给出待插值函数的解析表达式，当然也无法给出其在某一点的一阶导或者二阶导数的值。从这个角度而言，选择 CCS 或者 CS-SSD 作为边界条件是不妥的，而 NCS 规定在插值区间端点处的二阶导数为 0，既具有其实际意义，又避免了导数的计算，因此相对更为合理。
- **插值点选取**：本程序采用

$$(0, \pm\frac{2}{3}\sqrt{3}), \quad (\pm\sqrt{3}, \frac{2}{3}\sqrt[4]{3}) \tag{10}$$

这四个点作为 characteristic points，对于其他插值点，程序采用顺时针方向，根据 x 坐标均匀间隔依次分配插值点。

2.4.3 插值结果展示

- **三次样条**：结合上述分析，设定`E.json`中参数`order`为 3，即可分别根据插值点个数 $n = 10, 40, 160$ 进行心形曲线三次样条插值，得到结果输出至`ProblemE_result_points.txt`，后利用 python 绘制插值图像及其对比图如下所示。

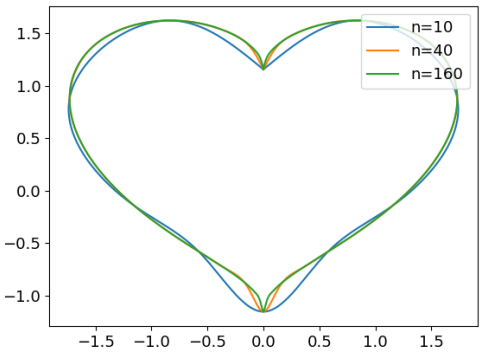

图 14: 三次样条插值结果对比图

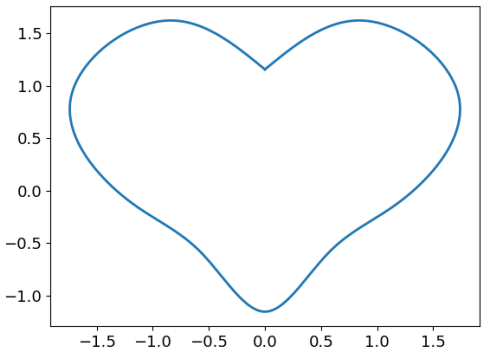

图 15: 三次样条插值结果（$n = 10$)

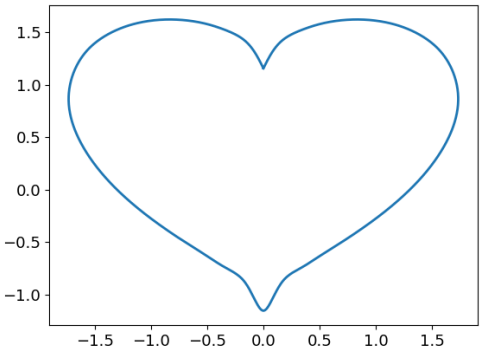

图 16: 三次样条插值结果（$n = 40$)

图 17: 三次样条插值结果（$n = 160$)

可以看出，插值曲线的轮廓是较为光滑的，并且随着插值点个数 n 的增大，曲线三次样条插值结果与心形越来越接近，同时可以很好地反映出心形曲线的轮廓特征。这也从侧面说明了选择 characteristic points 的合理性。

- **线性样条**：同样地，修改`E.json`中参数`order`为 1 后，即可分别根据插值点个数 $n = 10, 40, 160$ 进行心形曲线线性样条插值，得到结果输出至`ProblemE_result_points.txt`，后利用 python 绘制插值图像及其对比图如下所示。

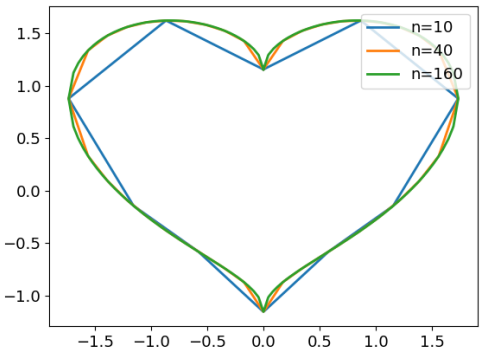

图 18: 线性样条插值结果对比图

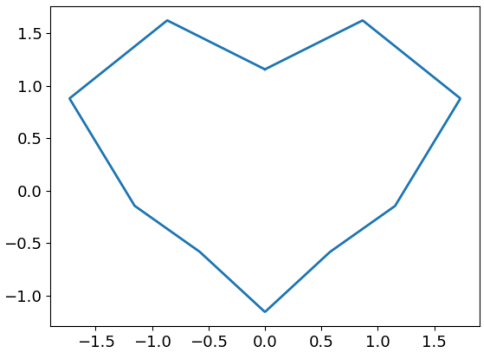

图 19: 线性样条插值结果（$n = 10$)

图 20: 线性样条插值结果（$n = 40$)

图 21: 线性样条插值结果（$n = 160$)

对比线性样条和三次样条曲线插值的图像，可以较明显地看出三次样条的拟合结果更加光滑且精确。