Student name: Abao Zhang (张宝)

Student number: 12332459

Homework 5

Question 1

最小二乘法的目标是找到直线$y = Ax + B$使得所有数据点(x_i, y_i)到直线之间的误差平方和最小化。误差平方和表示为

$$S(A,B) = \sum_{k=1}^{N} \left((Ax_k + B) - y_k\right)^2$$

对A和B分别求偏导数：

$$\frac{\partial S}{\partial A} = 2\sum_{k=1}^{N} x_k(Ax_k + B - y_k)$$
$$\frac{\partial S}{\partial B} = 2\sum_{k=1}^{N} (Ax_k + B - y_k)$$

分别另两者等于 0，可以联立方程组

$$2\sum_{k=1}^{N} x_k(Ax_k + B - y_k) = 0$$
$$2\sum_{k=1}^{N} (Ax_k + B - y_k) = 0$$

分别展开

$$A\sum_{k=1}^{N} x_k^2 + B\sum_{k=1}^{N} x_k - \sum_{k=1}^{N} x_k y_k = 0$$
$$A\sum_{k=1}^{N} x_k + NB - \sum_{k=1}^{N} y_k = 0$$

使用$N\bar{x} = \sum_{k=1}^{N} x_k$，$N\bar{y} = \sum_{k=1}^{N} y_k$替换，可以简化为

$$A\sum_{k=1}^{N} x_k^2 + NB\bar{x} - \sum_{k=1}^{N} x_k y_k = 0$$
$$A\bar{x} + B - \bar{y} = 0$$

消元法解方程组：根据第二个等式可以得到$B = \bar{y} - A\bar{x}$，代入等式一，解得

$$A = \frac{\sum_{k=1}^{N} (x_k - \bar{x})(y_k - \bar{y})}{\sum_{k=1}^{N} (x_k - \bar{x})^2}$$

命题得证，即

$$C = \sum_{k=1}^{N} (x_k - \bar{x})^2, \quad A = \frac{\sum_{k=1}^{N} (x_k - \bar{x})(y_k - \bar{y})}{C}, \quad B = \bar{y} - A\bar{x}$$

Question 2

(a)

$$S(A)=\sum_{k=1}^{N}\left(Ax_k-y_k\right)^2=\sum_{k=1}^{N}(A^2x_k^2-2Ax_ky_k+y_k^2)$$

对A求导数，并令倒数为 0

$$\frac{dS}{dA}=2\sum_{k=1}^{N}(Ax_k^2-x_ky_k)=0$$

解得

$$A=\frac{\sum_{k=1}^{N}x_ky_k}{\sum_{k=1}^{N}x_k^2}$$

(b)

$$S(A)=\sum_{k=1}^{N}\left(Ax_k^2-y_k\right)^2=\sum_{k=1}^{N}\left(A^2x_k^4-2Ax_k^2y_k+y_k^2\right)$$

对A求导数，并令倒数为 0

$$\frac{dS}{dA}=2\sum_{k=1}^{N}\left(Ax_k^4-x_k^2y_k\right)=0$$

解得

$$A=\frac{\sum_{k=1}^{N}x_k^2y_k}{\sum_{k=1}^{N}x_k^4}$$

(c)

$$S(A,B)=\sum_{k=1}^{N}\left(Ax_k^2+B-y_k\right)^2=\sum_{k=1}^{N}\left(A^2x_k^4+B^2+y_k^2+2ABx_k^2-2Ax_k^2y_k-2By_k\right)$$

对A和B 分别求偏导数，并分别令偏导数为 0，得到方程组如下：

$$\frac{\partial S}{\partial A}=2\sum_{k=1}^{N}x_k^2(Ax_k^2+B-y_k)=0$$
$$\frac{\partial S}{\partial B}=2\sum_{k=1}^{N}(Ax_k^2+B-y_k)=0$$

解二元一次方程组，可得

$$A=\frac{N\sum_i^N x_i^2y-N\bar{y}\sum_i^N x_i^2}{N\sum_i^N x_i^4-\left(\sum_i^N x_i^2\right)^2}$$
$$B=\bar{y}-\frac{A\sum_i^N x_i^2}{N}$$

Question 3

本题解题目思路为使用变量变换将数据变化，然后拟合线性方程$y = Ax + B$，然后将系数A、B变换回原方程。手动计算较为复杂，这里我使用了代码进行计算，可实际运行 python-3.py 查看结果。注意：我仅使用 numpy 表示数组，方便计算，并没有直接调库求解。下面给出部分计算逻辑代码。

首先是均方根误差E_2

```python
def E2(table, f):
    """计算均方根误差
    Args:
        table     [N, 2] 表示(xi, yi)
        f    fun: 函数
    """
    X, Y = table[:, 0], table[:, 1]
    f = np.vectorize(f)
    return np.sqrt(np.sum(np.subtract(f(X), Y) ** 2) / len(X))
```

是使用最小二乘法拟合线性方程$y = Ax + B$，我们需要计算系数A、B，这里使用的公式为第一问中得到的公式。

```python
def solve(X, Y) -> Tuple[float, float]:
    """使用最小二乘法进行线性拟合，得到 A 和 B
    Args:
        X    [N]:    数组，N 个元素
        Y    [N]:    数组，N 个元素
    """
    x_mean = np.mean(X)
    y_mean = np.mean(Y)
    C = np.sum((X - x_mean) ** 2)
    A = np.sum((X - x_mean) * (Y - y_mean)) / C
    B = y_mean - A * x_mean
    return A, B
```

下面给出最终的求解函数

```python
def resolve(table, input_mapper, coeff_mapper):
    """
    将数据按照 x_mapper, y_mapper 后，返回原始系数
    Args:
        table     [N, 2]
        input_mapper (x_mapper, y_mapper) 数据 mapper
            x_mapper (fun(x):X): 映射函数由 x 到 X
            y_mapper (fun(y):Y): 隐射函数由 y 到 Y
        coeff_mapper (A_mapper, B_mapper) 系数 mapper
            A_mapper (fun(A):raw_A): 映射系数 A 回到原参数
            B_mapper (fun(B):raw_B): 隐射系数 B 回到原参数
    """
    x_mapper, y_mapper = input_mapper
    x_mapper, y_mapper = np.vectorize(x_mapper), np.vectorize(y_mapper)
    X, Y = table[:, 0], table[:, 1]
    A, B = solve(x_mapper(X), y_mapper(Y))
    A_mapper, B_mapper = coeff_mapper
    return A_mapper(A), B_mapper(B)
```

调用求解函数，即可得最终结果。以问题（a）为例，下图中还包含了画图代码。

```
A, C = resolve(tab1, [lambda x: x, np.log], [do_nothing, np.exp])
def f(x):
    return C * np.exp(A * x)
e2 = E2(tab1, f)
plot(tab1, f)
print("(a) i")
print(f" A={A}, C={C}, E2={e2}")
```

(a)

i

$$A = -1.058567340022129, \quad C = 2.399508181612176, \quad E_2 = 3.4097854679724233,$$

ii

$$A = -1.058567340022129, \quad C = 2.399508181612176, \quad E_2 = 3.4097854679724233$$

(b)

i

$$A = 0.7573257893549833, \quad B = 0.7845232804008844, \quad E_2 = 669.2218637833518$$

ii

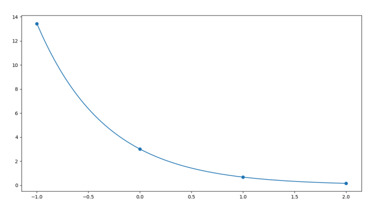
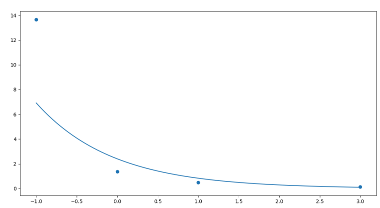
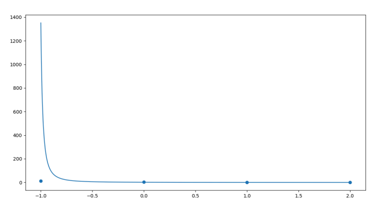
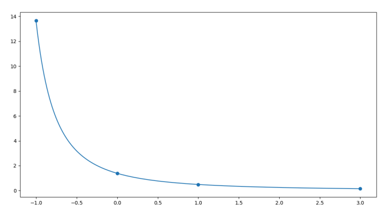
$$A = 0.5776582870479143, \quad B = 0.8498769941596073, \quad E_2 = 0.07767026951123651$$

(c)

基于上面结果，基于$E_2(f)$，我们可以发现：

- 对于数据表 1，曲线$f(x) = Ce^{Ax}$更好
- 对于数据表 2，曲线$f(x) = (Ax+B)^{-2}$更好

从下表格中的图我们也可以印证

	表格 1	表格 2
$f(x) = Ce^{Ax}$		
$f(x) = (Ax+B)^{-2}$		

Question 4

首先将E进行展开，方便后续计算

$$
\begin{aligned}
E(A,B,C) = \sum_{k=1}^{N} \big(& A^2 x_k^2 + B^2 y_k^2 + C^2 + z_k^2 \\
& +2ABx_k y_k + 2ACx_k - 2Ax_k z_k + 2BCy_k - 2By_k z_k - 2Cz_k \big)
\end{aligned}
$$

将E分别对A、B、C求偏导。

$$
\begin{aligned}
\frac{\partial E}{\partial A} &= 2A\sum_{k=1}^{N} x_k^2 + 2B\sum_{k=1}^{N} x_k y_k + 2C\sum_{k=1}^{N} x_k - 2\sum_{k=1}^{N} x_k z_k \\
\frac{\partial E}{\partial A} &= 2B\sum_{k=1}^{N} y_k^2 + 2A\sum_{k=1}^{N} x_k y_k + C\sum_{k=1}^{N} y_k - 2\sum_{k=1}^{N} y_k z_k \\
\frac{\partial E}{\partial A} &= 2NC + 2A\sum_{k=1}^{N} x_k + 2B\sum_{k=1}^{N} y_k - 2\sum_{k=1}^{N} z_k
\end{aligned}
$$

分别另其等于 0，可得到方程组

$$
\begin{gathered}
2A\sum_{k=1}^{N} x_k^2 + 2B\sum_{k=1}^{N} x_k y_k + 2C\sum_{k=1}^{N} x_k - 2\sum_{k=1}^{N} x_k z_k = 0 \\
2B\sum_{k=1}^{N} y_k^2 + 2A\sum_{k=1}^{N} x_k y_k + C\sum_{k=1}^{N} y_k - 2\sum_{k=1}^{N} y_k z_k = 0 \\
2NC + 2A\sum_{k=1}^{N} x_k + 2B\sum_{k=1}^{N} y_k - 2\sum_{k=1}^{N} z_k = 0
\end{gathered}
$$

整理可得

$$
\begin{gathered}
A\sum_{k=1}^{N} x_k^2 + B\sum_{k=1}^{N} x_k y_k + C\sum_{k=1}^{N} x_k = \sum_{k=1}^{N} x_k z_k \\
A\sum_{k=1}^{N} x_k y_k + B\sum_{k=1}^{N} y_k^2 + C\sum_{k=1}^{N} y_k = \sum_{k=1}^{N} y_k z_k \\
A\sum_{k=1}^{N} x_k + B\sum_{k=1}^{N} y_k + NC = \sum_{k=1}^{N} z_k
\end{gathered}
$$

命题得证

Question 5

假设三段曲线由系数$c = \{c_0, c_1, ..., c_{11}\}$进行表征，具体如下

$$
\begin{aligned}
f_0(x) &= c_0 + c_1 x + c_2 x^2 + c_3 x^3,\ x \in (-3, -2) \\
f_1(x) &= c_4 + c_5 x + c_6 x^2 + c_7 x^3,\ x \in (-2, 1) \\
f_2(x) &= c_8 + c_9 x + c_{10} x^2 + c_{11} x^3,\ x \in (1, 4)
\end{aligned}
$$

求一阶导数

$$
\begin{aligned}
{f_0}'(x) &= c_1 + 2c_2 x + 3c_3 x^2,\ x \in (-3, -2) \\
{f_1}'(x) &= c_5 + 2c_6 x + 3c_7 x^2,\ x \in (-2, 1) \\
{f_2}'(x) &= c_9 + 2c_{10} x + 3c_{11} x^2,\ x \in (1, 4)
\end{aligned}
$$

求二阶导数

$$
\begin{aligned}
{f_0}''(x) &= 2c_2 + 6c_3 x,\ x \in (-3, -2) \\
{f_1}''(x) &= 2c_6 + 6c_7 x,\ x \in (-2, 1) \\
{f_2}''(x) &= 2c_{10} + 6c_{11} x,\ x \in (1, 4)
\end{aligned}
$$

根据通过给定端点、内部端点二阶导连续、给定边界条件，进行代入，即可构建包含 12 个未知数、12 个方程的方程组，显然，给定的是克拉姆边界条件，方程是有唯一解的

$$
\begin{aligned}
f_1(-3) &= 2 \\
f_1(-2) &= 0 \\
f_2(-2) &= 0 \\
f_2(1) &= 3 \\
f_3(1) &= 3 \\
f_3(4) &= 1 \\
{f_1}'(-2) &= {f_2}'(-2) \\
{f_1}''(-2) &= {f_2}''(-2) \\
{f_2}'(1) &= {f_2}'(1) \\
{f_2}''(1) &= {f_2}''(1) \\
{f_1}'(-3) &= -1 \\
{f_3}'(4) &= 1
\end{aligned}
$$

手动计算耗费量巨大，且无意义，这里我使用编程方法进行$Ac = B$的计算，这里的A也就是由方程组得到的矩阵，c即为我们要求的系数，B 也就是对于方程组等号右边的数值。下面分别介绍A、B是如何构造的。

对于任意给定的端点数组points (2d 数组，shape=[N, 2])和起始点一阶导start_der 和最后一个点的一阶导end_der，第一步我们简单获取参数信息：

```
point_num = len(points)
interval_num = point_num - 1
coeff_num = (point_num - 1) * 4
```

接下来我们定义三个函数，分别返回长度为未知数个数(coeff_num)的数组，用于后续矩阵A的构建

```
def f_i(i, x):
    """f_i(x) = y 对应的系数, 即 Ac =b 的某一行
    Args:
        i (int): 第 i 条曲线
    """
    row = np.zeros((coeff_num))
    start = i * 4
    row[start : start + 4] = [1, x, x**2, x**3]
    return row
def fd_i(i, x):
    """一阶倒数系数"""
    row = np.zeros(coeff_num)
    start = i * 4
    row[start : start + 4] = [0, 1, 2 * x, 3 * x**2]
    return row
def fdd_i(i, x):
    """二阶倒数系数"""
    row = np.zeros(coeff_num)
    start = i * 4
    row[start : start + 4] = [0, 0, 2, 6 * x]
    return row
```

对于$f_{i(x)} = y$的方程，我可以得到将如下方程系数和b加入到我们的A和b中

```
A, b = [], []
for i, (x, y) in enumerate(points):
    if i != point_num - 1:
        print(f"point{i} 通过 spine{i}")
        A.append(f_i(i, x))
        b.append(y)
    if i != 0:
        print(f"point{i} 通过 spine{i-1}")
        A.append(f_i(i - 1, x))
        b.append(y)
```

对于中间端点一阶导数和二阶导数连续（相等）的方程，我们使用如下代码，注意，我们这里将$f_{i-1}'(x) = f_i'(x)$ 转换为了 $f_{i-1}'(x) - f_i'(x) = 0$，二阶导数同理

```
for i in range(1, point_num - 1):
    (x, y) = points[i]
    print(f"point{i} spine{i-1} 和 spine{i} 一阶导数连续（相等）")
    A.append(fd_i(i - 1, x) - fd_i(i, x))
    b.append(0)
    print(f"point{i} spine{i-1} 和 spine{i} 二阶导数连续（相等）")
    A.append(fdd_i(i - 1, x) - fdd_i(i, x))
    b.append(0)
```

两个边界条件对应的两个方程

```
A.append(fd_i(0, points[0, 0]))
b.append(start_der)
A.append(fd_i(interval_num - 1, points[-1, 0]))
b.append(end_der)
```

最后，我们获得了A和B，只需要解方程组就可以求到所有的c_i，我这里使用的使用我在作业三-题目四中手撸的 `gs_partial_scaled_pivot(A, B)`方法。

我还编写了最终的曲线函数，对于给定x都可以计算响应的值

```
def S(x):
    if x < points[0, 0] and x > points[-1, 0]:
        assert False, "out of range"
    for i in range(interval_num):
        if x <= points[i + 1, 0]:
            return np.sum(C[i * 4: i * 4 + 4] @ [1, x, x**2, x**3])
```

最后我给出所有的系数

$$c_0 = 16.12903226, c_1 = 23.48387097, c_2 = 10.61290323, c_3 = 1.4516129$$
$$c_4 = 1.70609319, c_5 = 1.84946237, c_6 = -0.20430108, c_7 = -0.35125448$$
$$c_8 = 1.0525687, c_9 = 3.81003584, c_{10} = -2.16487455, c_{11} = 0.30227001$$

即对应

$$f_1(x) = 16.12903226 + 23.48387097x + 10.61290323x^2 + 1.4516129x^3$$
$$f_2(x) = 1.70609319 + 1.84946237x - 0.20430108x^2 - 0.35125448x^3$$
$$f_3(x) = 1.0525687 + 3.81003584x - 2.16487455x^2 + 0.30227001x^3$$

最后，我绘制了曲线进行可视化，具体代码可以参考 `question-5.py`

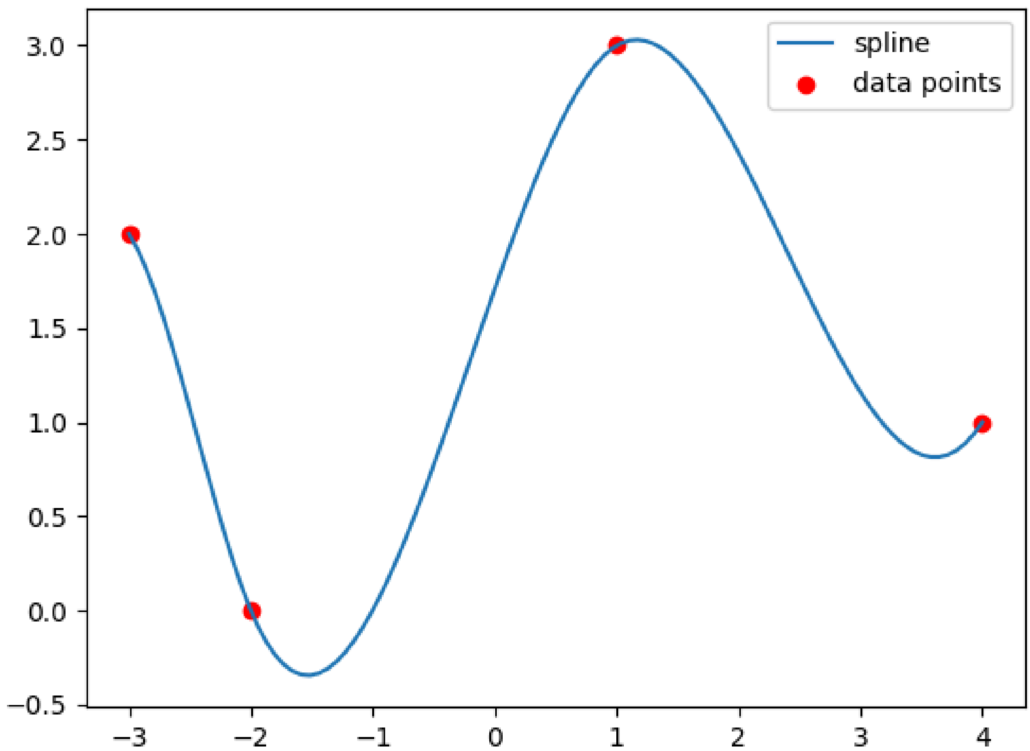


Figure 1: 可视化曲线

Question 6

代码从 question 5 修改得到，代码可以查看 `question-6.py` 具体有以下不同

```
~~A.append(fd_i(0, points[0, 0]))~~
~~b.append(start_der)~~
~~A.append(fd_i(interval_num - 1, points[-1, 0]))~~
~~b.append(end_der)~~
```

```
A.append(fdd_i(0, points[0, 0]))
b.append((points[1, 1] - 2 * points[0, 1]) / ((points[1, 0] - points[0, 0]) ** 2))
A.append(fdd_i(interval_num - 1, points[-1, 0]))
b.append(
    (points[-2, 1] - 2 * points[-1, 1]) / ((points[-2, 0] - points[-1, 0]) ** 2)
)
```

这里使用公式

$$f''(x_0) \approx \frac{y_1 - 2y_0}{(x_1 - x_0)^2}$$

作为边界二阶导数的近似。具体结果如下

$$f_1(x) = 13.563218390804604 + 20.02681992337165x + 9.086206896551724x^2 + 1.2318007662835249x^3$$

$$f_2(x) = 1.4423158790974888 + 1.845466155810983x + -0.004469987228608 03x^2 + -0.2833120476798638x^3$$

$$f_3(x) = 1.0578969774372076 + 2.998722860791826x + -1.157726692209451x^2 + 0.1011068539804172x^3$$

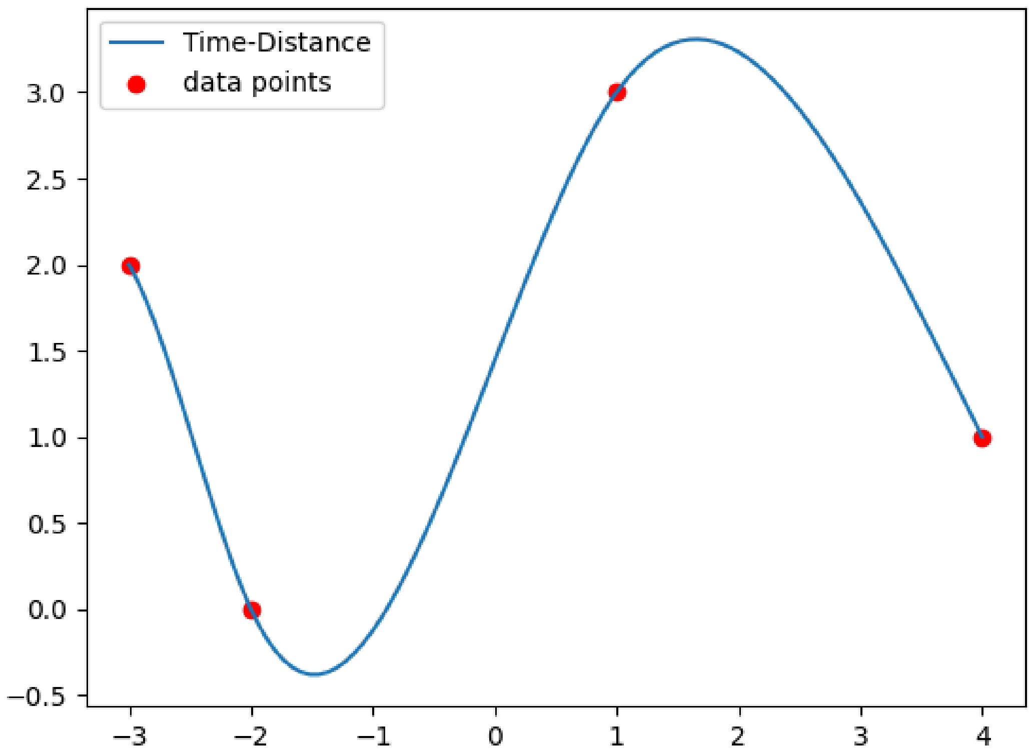


Figure 2: 可视化曲线

Question 7

(a)

对于$f(x) = x^3 - x$，求其一阶导、二阶导数分别为

$$f'(x) = 3x^2 - 1$$
$$f''(x) = 6x$$

可以发现$f(x)$、$f'(x)$ 和$f''(x)$在$x_0 = -2$和$x_1 = 0$处均连续，所以$f(x)$可以表示区间$[-2, 0]$上的三次样条曲线

(b)

同样的，$f(x)$、$f'(x)$ 和$f''(x)$在$x_0 = -2$、$x_1 = 0$和$x_2 = 2$处均连续，命题得证

(c)

对于任意三次多项式$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ ，在闭区间$[a, b]$是是否可以作为对应的样条曲线取决于其在端点$x = a$和$x = b$上是否存在一阶导数、二阶导数并连续， 具体而言就是下面$f'(x), f''(x)$存在，且在 a 和 b 上连续

$$f'(x) = a_1 + 2a_2 x + 3a_3 x^2$$
$$f''(x) = 2a_2 + 6a_3 x$$

Question 8

首先，根据傅里叶级数的定义，对于一个周期为 $2P$的周期函数 $f(x)$，其傅里叶级数展开

Question 9

对于贝塞尔曲线 $P(t)$，我们有一组控制点 $P_0, P_1, \cdot, P_N$ 和贝塞尔曲线的参数方程：

$$P(t) = \sum_{i=0}^{N} B_{i,N}(t) P_i$$

其中，$B_{i,N}(t)$ 是贝塞尔基函数，定义为：

$$B_{i,N}(t) = \binom{N}{i} t^i (1-t)^{N-i}$$

贝塞尔曲线的一阶导数和二阶导数分别是：

$$\frac{dP}{dt} = \sum_{i=0}^{N} \frac{dB}{dt} P_i$$
$$\frac{d^2P}{dt^2} = \sum_{i=0}^{N} \frac{d^2B}{dt^2} P_i$$

在 t=0 时，只有控制点P_0, P_1, P_2对其有贡献，其他项均为 0，

$$P''(0) = N(N-1)P_0 - 2N(N-1)P_1 + N(N-1)P_2 = N(N-1)(P_2 - 2P_1 + P_0)$$

同理在 t=1 时，只有控制点P_N, P_{N-1}, P_{N-2}对其有贡献，其他项均为 0，

$$P''(1) = N(N-1)(P_N - 2P_{N-1} + P_{N-2})$$

命题得证

Question 10

这里调用第 5 问中手撸的程序即可，这里不过多赘述，具体可以查看 question-10.py，下面给出调用的代码和

```
p = np.array(
    [
        [0, 0],
        [2, 40],
        [4, 160],
        [6, 300],
        [8, 480],
    ]
)
model = solve_cubic_coeff(p, 0, 98)

x_vals = np.linspace(0, 8, 100)

plt.plot(x_vals, model(x_vals), label="Time-Distance")
plt.scatter(p[:, 0], p[:, 1], color="red", label="data points")
plt.legend()
plt.show()
```

下面函数的对应区间不言自明

$$f_0(x) = 0.0 + 0.0x + 8.374999999999993x^2 + 0.8125000000000023x^3$$
$$f_1(x) = 26.000000000000043 + -39.00000000000006x + 27.875000000000002x^2 + -2.4375000000000002x^3$$
$$f_2(x) = -222.00000000000023 + 147.00000000000014x + -18.625000000000003x^2 + 1.4375000000000018x^3$$
$$f_3(x) = 263.9999999999999 + -95.99999999999994x + 21.874999999999986x^2 + -0.8124999999999999x^3$$

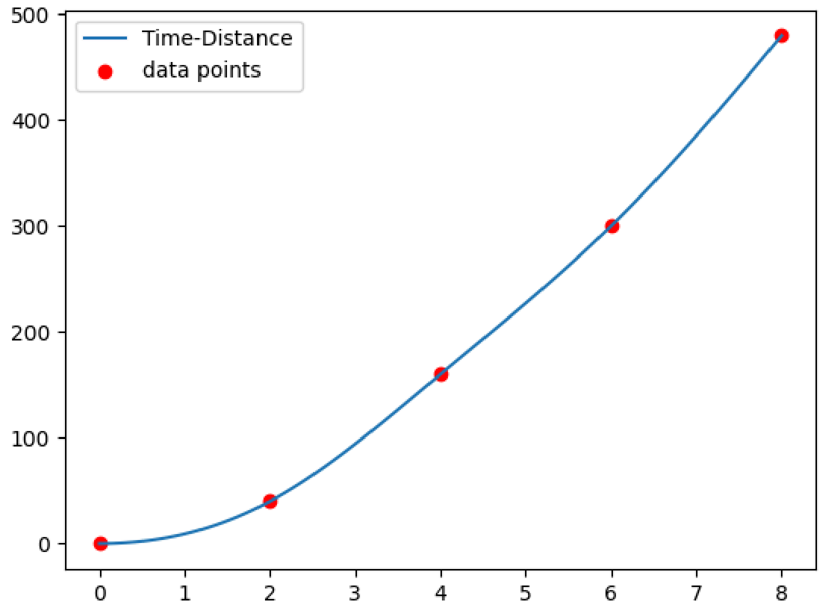


Figure 3: 时间-距离曲线

Question 11

(a) 自然边界条件

即$S''(x_1) = 0$和$S''(x_n) = 0$在第 5 问中手撸的程序中稍微修改最后两个方程（边界条件）即可，下面给出 需要修改的部分代码，完整版本可以查看 question-11-a.py

```
~~A.append(fd_i(0, points[0, 0]))~~
~~b.append(start_der)~~
~~A.append(fd_i(interval_num - 1, points[-1, 0]))~~
~~b.append(end_der)~~
```

```
A.append(fdd_i(0, points[0, 0]))
b.append(0)
A.append(fdd_i(interval_num - 1, points[-1, 0]))
b.append(0)
```

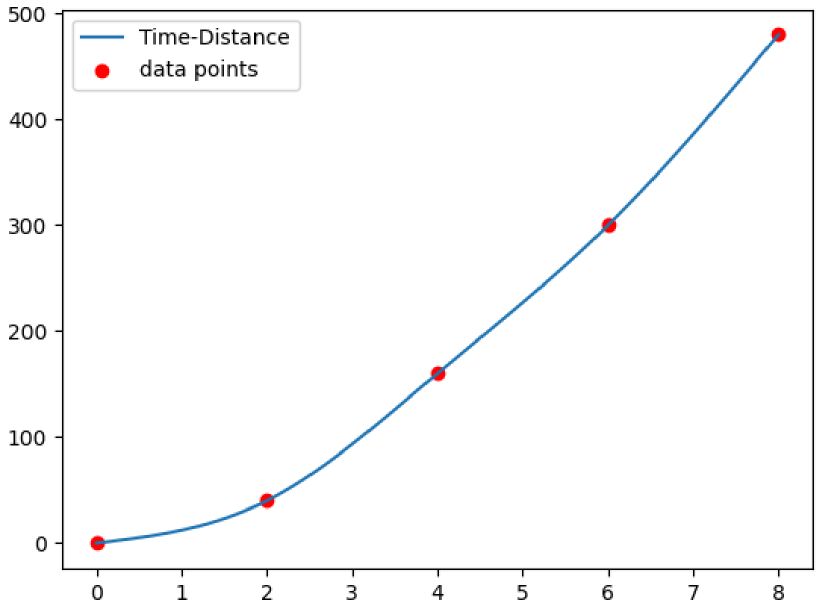


Figure 4: 时间-距离曲线

(b) 外推边界条件

跟题目 6 一样，下面仅给出结果

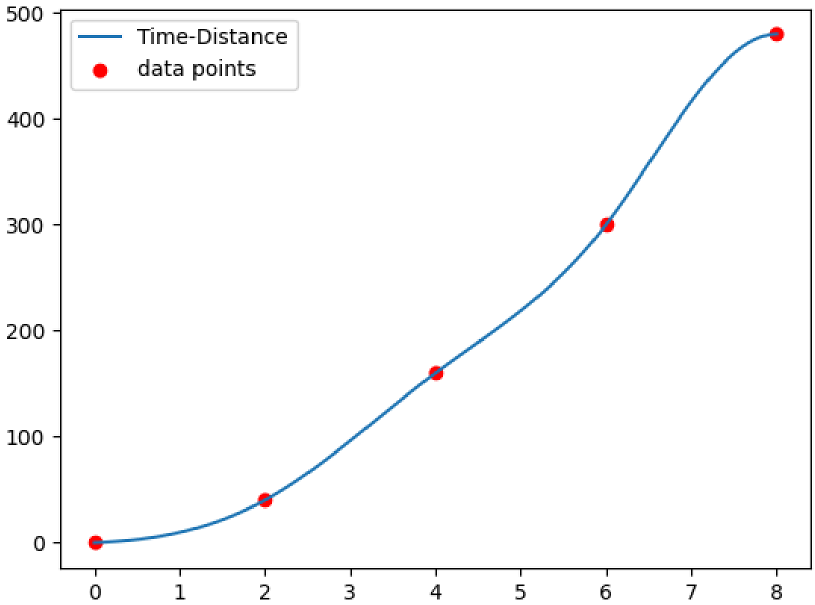


Figure 5: 时间-距离曲线