数值分析上机报告
第四章

院 (系) 名 称：　微电子学院

学 生 姓 名：　周玉乾

学　　　　号：　220205764

二〇二〇年十二月

第四章

4.1 3 次样条插值函数

1. 问题

(1) 编制求第一型 3 次样条插值函数的通用程序；

(2) 已知汽车门曲线型值点的数据如下：

i	0	1	2	3	4	5	6	7	8	9	10
x_i	0	1	2	3	4	5	6	7	8	9	10
y_i	2.51	3.30	4.04	4.70	5.22	5.54	5.78	5.40	5.57	5.70	5.80

端点条件为 $y_0^{'} = 0.8$，$y_{10}^{'} = 0.2$，用所编程序求车门的三次样条插值函数 $S(x)$，并打印出 $S(i+0.5), i = 0, 1, \cdots, 9$。

2. 分析

根据 3 次样条插值的定义，插值函数 $S(x)$ 满足：1. $S(x)$ 在每一个小区间上式 3 次多项式，2. $S(x)$ 在区间 $[a, b]$ 上具有连续 2 阶导数，3. $S(x)$ 经过每一个给定的插值节点。

在区间 $[a, b]$ 上有 $n+1$ 个插值节点，因此可以设每个小区间上的插值函数 $S_i(x)$ 为：

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i \tag{4.1}$$

在 n 个区间上有 n 个插值函数，每个插值函数有 4 个参数，因此需要 $4n$ 个不相关的方程来求解：

(1) 每个插值函数都会经过它所在小区间上的插值节点，即小区间的两个端点，有 $2n$ 个方程：

$$S_i(x_i) = y_i, \quad i \in [0, n-1] \tag{4.2}$$

$$S_i(x_{i+1}) = y_{i+1}, \quad i \in [0, n-1] \tag{4.3}$$

(2) 一阶导数连续，即两个相邻的插值函数连接点处的一阶导数相等，有 $n-1$ 个方程：

$$S_i^{'}(x_{i+1}) = S_{i+1}^{'}(x_{i+1}), \quad i \in [0, n-2] \tag{4.4}$$

(3) 二阶导数连续，即两个相邻的插值函数连接点处的二阶导数相等，有 $n-1$ 个方程：

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}), \quad i \in [0, n-2] \tag{4.5}$$

(4) 第一型边界条件，已知两个端点处的一阶导数值，有 2 个方程：

$$S_0'(x_0) = y_0' \tag{4.6}$$

$$S_{n-1}'(x_n) = y_n' \tag{4.7}$$

$4n$ 个方程，使用列主元高斯法求解方程，得到最终的插值函数。

3. 程序

```
import numpy as np
import matplotlib.pyplot as plt
from pylab import mpl
import sys, os

'''
DESCRIPTION:
PARAM {*} X N+1 个插值点
PARAM {*} Y N+1 个插值点
RETURN {*} N
'''
def Prejudgment(x, y):
    n1 = len(x)
    n2 = len(y)
    if n1 != n2:
        print('x 与 y 长度不相等')
        sys.exit()

    n = n1-1
    return n

'''
DESCRIPTION: 求三次样条差值的 4N 个方程
PARAM: {X[0,N], Y[0,N]} N+1 个插值点
PARAM: TYPE 三次样条边界条件 1 OR 2 OR 3
RETURN {A, B} [A0 B0 C0 D0 A1 B1 C1 D1 ... A(N-1) B(N-1) C(N-1) D(N-1)] = [B] 形式
    的方程组
'''
def calculateEquationParameters(x, y, Type=1, dy0=0, dyn=0):
    n = Prejudgment(x, y)

    parameterA = []
    parameterB = []
```

```
    # S_i(x_i) = y_i
    # S_i(x_{i+1}) = y_{i+1}
    # 0 <= i <= n-1
    for i in range(0, n):
        # S_i(x_i) = y_i
        data = np.zeros(n*4)
        data[i*4] = pow(x[i], 3)
        data[i*4+1] = pow(x[i], 2)
        data[i*4+2] = x[i]
        data[i*4+3] = 1
        parameterA.append(data.tolist())
        parameterB.append(y[i])

        # S_i(x_{i+1}) = y_{i+1}
        data1 = np.zeros(n*4)
        data1[i*4] = pow(x[(i+1)], 3)
        data1[i*4+1] = pow(x[(i+1)], 2)
        data1[i*4+2] = x[(i+1)]
        data1[i*4+3] = 1
        parameterA.append(data1.tolist())
        parameterB.append(y[i+1])

    # S'_i(x_{i+1}) = S'_{i+1}(x_{i+1})
    # 0 <= i <= n-2
    for i in range(0, n-1):
        data = np.zeros(n*4)

        data[i*4] = 3 * pow(x[i+1], 2)
        data[i*4+1] = 2 * x[i+1]
        data[i*4+2] = 1

        data[(i+1)*4] = -3 * pow(x[i+1], 2)
        data[(i+1)*4+1] = -2 * x[i+1]
        data[(i+1)*4+2] = -1

        parameterA.append(data.tolist())
        parameterB.append(0)

    # S''_i(x_{i+1}) = S''_{i+1}(x_{i+1})
    # 0 <= i <= n-2
    for i in range(0, n-1):
        data = np.zeros(n*4)

        data[i*4] = 6 * x[i+1]
        data[i*4+1] = 2

        data[(i+1)*4] = -6 * x[i+1]
```

```
        data[(i+1)*4+1] = -2

        parameterA.append(data.tolist())
        parameterB.append(0)

    if Type == 1:
        # S'_0(x_0) = y'_0
        data = np.zeros(n*4)
        data[0] = 3 * pow(x[0], 2)
        data[1] = 2 * x[0]
        data[2] = 1
        parameterA.append(data.tolist())
        parameterB.append(dy0)

        # S'_{n-1}(x_n) = y'_n
        data = np.zeros(n*4)
        data[(n-1)*4] = 3 * pow(x[n], 2)
        data[(n-1)*4+1] = 2 * x[n]
        data[(n-1)*4+2] = 1

        parameterA.append(data.tolist())
        parameterB.append(dyn)
    elif Type == 2:
        # S''(a) = S''(b) = 0

        # S''_0(x_0) = 0
        data = np.zeros(n*4)
        data[0] = 6 * x[0]
        data[1] = 2
        parameterA.append(data.tolist())
        parameterB.append(0)

        # S''_{n-1}(x_n) = 0
        data = np.zeros(n*4)
        data[(n-1)*4] = 6 * x[n]
        data[(n-1)*4+1] = 2
        parameterA.append(data.tolist())
        parameterB.append(0)

    elif Type == 3:
        # S'(a) = S'(b) and # S''(a) = S''(b)
        pass
    else:
        print('Error! Unknown "Type" Value!')

    return parameterA, parameterB

"""
```

```
功能：根据所给参数，计算三次函数的函数值：
参数：OriginalInterval为原始x的区间，parameters为二次函数的系数，x为自变量
返回值：为函数的因变量
"""
def calculate(OriginalInterval, paremeters, x):
    n = int(len(paremeters)/4)
    result=[]
    for data_x in x:

        Interval = 0
        if data_x <= OriginalInterval[0]:
            Interval = 0
        elif data_x >= OriginalInterval[-1]:
            Interval = n-1
        else:
            for i in range(0,n):
                if data_x >= OriginalInterval[i] and data_x < OriginalInterval[i+1]:
                    Interval = i
                    break

        result.append(paremeters[Interval*4+0]*data_x*data_x*data_x+paremeters[Interval*4+1]*data_x*data_x+paremeters[Interval*4+2]*data_x+paremeters[Interval*4+3])
    return result

"""
功能：将函数绘制成图像
参数：data_x,data_y为离散的点.new_data_x,new_data_y为由拉格朗日插值函数计算的值。x为函数的预测值。
返回值：空
"""
def Draw(data_x,data_y,new_data_x,new_data_y, title):
        plt.plot(new_data_x, new_data_y, label="拟合曲线", color="black")
        plt.scatter(data_x,data_y, label="离散数据",color="red")
        mpl.rcParams['font.sans-serif'] = ['SimHei']
        mpl.rcParams['axes.unicode_minus'] = False
        plt.title("三次样条函数")
        plt.legend(loc="upper left")
        plt.savefig(os.path.join(os.path.dirname(os.path.abspath(__file__)), title+'.png'), dpi=300)
        plt.show()

def PrintS(parameterX):
    n = int(len(parameterX)/4)
    print('S(x) = ')
    for i in range(0,n):
        print("%.6g & %.6g & %.6g & %.6g \\\\" % (parameterX[i*4], parameterX[i*4+1], parameterX[i*4+2], parameterX[i*4+3]))
```

```python
    print('\n\n')

def main():
    x = [0,   1,   2,   3,   4,   5,   6,   7,   8,   9,   10]
    y = [2.51, 3.30, 4.04, 4.70, 5.22, 5.54, 5.78, 5.40, 5.57, 5.70, 5.80]
    dy0 = 0.8
    dyn = 0.2
    parameterA, parameterB = calculateEquationParameters(x, y, 1, dy0, dyn)
    parameterX = MGauss_Caculate(parameterA, parameterB)

    PrintS(parameterX)

    # 画图
    new_data_x = np.arange(x[0]-0.5, x[-1]+0.6, 0.1)
    new_data_y = calculate(x, parameterX, new_data_x)
    Draw(x, y, new_data_x, new_data_y, '三次样条插值')

    # 打印
    new_data_x = np.arange(0.5, 10.5, 1)
    new_data_y = calculate(x, parameterX, new_data_x)
    # F4_5 = CALCULATE(PARAMETERX[8:12], [4.5])
    print(new_data_x)
    for i,data in enumerate(new_data_y):
        print("%.6g & " % data)

if __name__ == "__main__":

    # 获取当前文件路径
    current_path = os.path.abspath(__file__)
    sys.path.append(os.path.abspath(os.path.join(os.path.dirname(current_path), '
        ../')))
    # PRINT(SYS.PATH)
    # 调用 CHAPTER3 中的列主元高斯消去法
    from chapter3.q3 import MGauss_Caculate

    main()
```

4. 算例

对题目中的数据进行三次样条插值:

i	0	1	2	3	4	5	6	7	8	9	10
x_i	0	1	2	3	4	5	6	7	8	9	10
y_i	2.51	3.30	4.04	4.70	5.22	5.54	5.78	5.40	5.57	5.70	5.80

得到的插值函数系数为:

$$
\begin{bmatrix}
-0.00851404 & -0.00148596 & 0.8 & 2.51 \\
-0.00445789 & -0.0136544 & 0.812168 & 2.50594 \\
-0.00365441 & -0.0184753 & 0.82181 & 2.49952 \\
-0.0409245 & 0.316955 & -0.184482 & 3.50581 \\
0.107352 & -1.46237 & 6.93281 & -5.98391 \\
-0.268485 & 4.17519 & -21.255 & 40.9958 \\
0.426587 & -8.33611 & 53.8128 & -109.14 \\
-0.267865 & 6.24739 & -48.2717 & 129.057 \\
0.0548723 & -1.4983 & 13.6939 & -36.1842 \\
0.0583759 & -1.5929 & 14.5453 & -38.7383
\end{bmatrix}
$$

题目中要求的 $S(i+0.5), i=0,1,\cdots,9$ 值如表4.1所示，插值函数的图像如图4.1所示。

表 4.1 $S(i+0.5), i=0,1,\cdots,9$

i	0	1	2	3	4
x_i	0.5	1.5	2.5	3.5	4.5
$S(x)$	2.90856	3.67843	4.38147	4.98819	5.38328
i	5	6	7	8	9
x_i	5.5	6.5	7.5	8.5	9.5
$S(x)$	5.7237	5.59441	5.42989	5.65977	5.7323

5. 结论

(1) 编写程序实现了第一型边界条件和第二型边界条件下的三次样条插值计算；

(2) 三次样条插值得到的函数二阶导数连续，因此其光滑性较好；

(3) 目前算法使用的是待定系数法求解，当插值节点较多时，计算量较大，应加以改进。

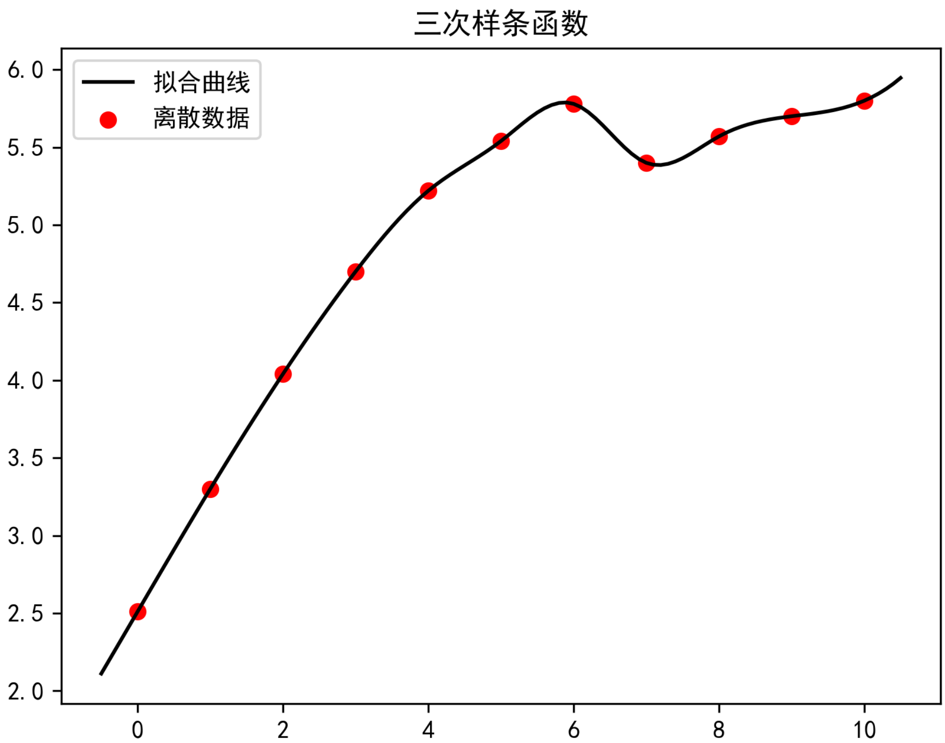


图 4.1　三次样条插值图像

4.2 离散数据的最佳平方逼近

1. 问题

一种商品的需求量和其价格有一定关系，先对一定时期内的商品价格 x 与需求量 y 进行观察, 取得如下的样本数据，如表4.2所示。

表 4.2 样本数据

价格	2	3	4	5	6	7	8	9	10	11
需求量	58	50	44	38	34	30	29	26	25	24

(1) 对上述数据，分别求出其 2, 3, 4 次最佳平方逼近多项式。画出图形，并比较拟合误差:

$$Q = \sum_{k=1}^{n}(q(x_k) - y_k)^2 \tag{4.8}$$

(2) 假设拟合函数分别为:

$$a + \frac{b}{x}, \quad a + b\ln x, \quad ae^{bx}, \quad \frac{1}{a + bx} \tag{4.9}$$

分别求出 a, b 的值。绘图，计算误差，并比较优劣。

2. 分析

本题是一个离散数据的最佳平方逼近问题，对于这类问题，首先要将拟合函数化为标准形式，即 $p(x) = \sum_{k=1}^{n}(c_i\varphi_i(x))$，然后求出正规方程，通过求解正规方程得到拟合函数系数 c_i，如有需要再将标准形式的拟合函数化为需要的形式。

在第二问中，四个拟合函数都不是标准形式。

1. 对于 $y = a + b/x$，令 $t = 1/x$，则 $y = a + bt$;
2. 对于 $y = a + b\ln x$，令 $t = \ln x$，则 $y = a + bt$;
3. 对于 $y = ae^{bx}$，两边取对数，得到 $\ln y = \ln a + bx$;
4. 对于 $y = 1/(a + bx)$，令 $t = 1/y$，则 $t = a + bx$。

3. 程序

```python
# 离散数据的最佳平方逼近
import numpy as np
import sys, os
import matplotlib.pyplot as plt
from pylab import mpl
import math
```

```python

'''
DESCRIPTION:
PARAM {*} X N+1 个插值点
PARAM {*} Y N+1 个插值点
RETURN {*} N
'''
def PrejudgmentShape(x, y):
    n1 = len(x)
    n2 = len(y)
    if n1 != n2:
        print('x 与 y 长度不相等')
        sys.exit()

    n = n1-1
    return n

'''
DESCRIPTION: 求最佳平方逼近的正规方程
PARAM {*} M M次最佳平方逼近
PARAM {*} X N个自变量X
PARAM {*} Y N个因变量Y
RETURN {*} (A, B)
'''
def GetNormalEquation(m, x, y):
    A = []
    b = []
    PrejudgmentShape(x, y)
    for j in range(0,m+1):
        AAline = []
        for i in range(0, m+1):
            tmp = np.dot(np.power(x, j), np.power(x, i))
            AAline.append(tmp)
        A.append(AAline)
        tmp = np.dot(y, np.power(x, j))
        b.append(tmp)
    return A, b

'''
DESCRIPTION: 获取最佳平方逼近的多项式
PARAM {*} A 正规方程系数A
PARAM {*} B 正规方程系数B
RETURN {*} C 多项式系数, Q(X) = C0 + C1*X + C2*X^2 + ... + CM*X^M
'''
def GetExpression(A, b):
    c = MGauss_Caculate(A, b)
    return c

```

```
'''
DESCRIPTION: 计算最佳平方逼近多项式
PARAM {*} M M次最佳平方逼近
PARAM {*} X N个自变量X
PARAM {*} Y N个因变量Y
RETURN {*} C 最佳平方逼近多项式系数, Q(X) = C0 + C1*X + C2*X^2 + ... + CM*X^M
'''
def GetLeastSquaresApproximationExpression(m, x, y):
    A, b = GetNormalEquation(m, x, y)
    c = GetExpression(A, b)
    return c

'''
DESCRIPTION: 计算多项式, Q(X) = C0 + C1*X + C2*X^2 + ... + CM*X^M
PARAM {*} C 最佳平方逼近多项式系数
RETURN {*} 多项式结果
'''
def CaculateExpression(c, x):
    res = 0
    for i, ci in enumerate(c):
        res = res + ci * pow(x, i)
    return res

'''
DESCRIPTION: 计算多项式, X FORM X0 TO XN, Q(X) = C0 + C1*X + C2*X^2 + ... + CM*X^M
PARAM {*} C 最佳平方逼近多项式系数
PARAM {LIST} XN
RETURN {*}
'''
def CaculateInterval(c, xn):
    res = []
    for x in xn:
        res.append(CaculateExpression(c, x))
    return res

"""
功能: 将函数绘制成图像
参数: DATA_X, DATA_Y 为原始值, NEW_DATA_X, NEW_DATA_Y 为预测计算的值。
返回值: 空
"""
def Draw(data_x,data_y,new_data_x,new_data_y, title):
    plt.plot(new_data_x, new_data_y, label="拟合曲线", color="black")
    plt.scatter(data_x,data_y, label="离散数据",color="red")
    mpl.rcParams['font.sans-serif'] = ['SimHei']
    mpl.rcParams['axes.unicode_minus'] = False
    plt.title(title)
    plt.legend(loc="upper right")
    plt.savefig(os.path.join(os.path.dirname(os.path.abspath(__file__)), title+'.
```

```
        png'), dpi=300)
    plt.show()

'''
DESCRIPTION: 计算拟合误差 Q = \SIGMA_{K=1}^{N} (Q(X_K) - Y_K)^2
PARAM {*} C 最佳平方逼近多项式系数, Q(X) = C0 + C1*X + C2*X^2 + ... + CM*X^M
PARAM {*} X
PARAM {*} Y
RETURN {*} Q
'''
def GetFittingError(c, x, y):
    Err = 0
    for n, xn in enumerate(x):
        qx = CaculateExpression(c, xn)
        Err += pow((qx - y[n]), 2)
    return Err

'''
DESCRIPTION: 别求出其 2, 3, 4 次最佳平方逼近多项式.画出图形, 并比较拟合误差
PARAM {*}
RETURN {*}
'''
def Q1():
    x = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
    y = [58, 50, 44, 38, 34, 30, 29, 26, 25, 24]

    result_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), '最佳平
        方逼近_result.txt')

    if os.path.exists(result_file):
        os.remove(result_file)

    # 最佳平方逼近
    for m in range(2, 5):
        c = GetLeastSquaresApproximationExpression(m, x, y)
        # 画图
        new_x = np.arange(x[0]-0.5, x[-1]+0.6, 0.1)
        new_y = CaculateInterval(c, new_x)
        Draw(x, y, new_x, new_y, "{0}次最佳平方逼近".format(m))
        # 拟合误差
        FittingErr = GetFittingError(c, x, y)
        # 写入文件
        with open(result_file, "a+") as fo:
            fo.write("\n
                \\************************************************************\\\n")
            fo.write("{0} 次最佳平方逼近, m = {1}\n".format(m, m))
            fo.write("x = ")
            for xi in x:
```

```python
                fo.write(str(xi))
                fo.write("\t")
            fo.write("\n")
            fo.write("y = ")
            for yi in y:
                fo.write(str(yi))
                fo.write("\t")
            fo.write("\n")
            fo.write("近似函数: q(x) = c0 + c1*x + c2*x^2 + ... + cm*x^m \n")
            fo.write("c = ")
            for ci in c:
                fo.write(str(format(ci, '.6g')))
                fo.write("\t")
            fo.write("\n")
            fo.write("拟合误差: {0}\n".format(format(FittingErr, '.8g')))
            fo.write("\\**********************************************************\\n
                ")

# A+B÷x近似
def Q2_1():
    x = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
    y = [58, 50, 44, 38, 34, 30, 29, 26, 25, 24]

    result_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'a+b÷x近
        似_result.txt')

    # A + B/x
    x1 = [1.0/xi for xi in x]
    c = GetLeastSquaresApproximationExpression(1, x1, y)
    # 画图
    new_x = np.arange(x[0]-0.5, x[-1]+0.6, 0.1)
    new_x1 = [1.0/xi for xi in new_x]
    new_y = CaculateInterval(c, new_x1)
    Draw(x, y, new_x, new_y, "a+b÷x近似")
    # 拟合误差
    FittingErr = GetFittingError(c, x1, y)
    # 写入文件
    with open(result_file, "w+") as fo:
        fo.write("\n\\**********************************************************\\n"
            )
        fo.write("a+b÷x近似\n")
        fo.write("x = ")
        for xi in x:
            fo.write(str(xi))
            fo.write("\t")
        fo.write("\n")
        fo.write("y = ")
        for yi in y:
```

```python
            fo.write(str(yi))
            fo.write("\t")
        fo.write("\n")
        fo.write("近似函数: q(x) = {} + {} / x \n".format(c[0], c[1]))
        fo.write("拟合误差: {0}\n".format(FittingErr))
        fo.write("\\**********************************************************\\\n")

# a+blnx近似
def Q2_2():
    x = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
    y = [58, 50, 44, 38, 34, 30, 29, 26, 25, 24]

    result_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'a+blnx近似_result.txt')

    # a + blnx
    x1 = [math.log(xi) for xi in x]
    c = GetLeastSquaresApproximationExpression(1, x1, y)
    # 画图
    new_x = np.arange(x[0]-0.5, x[-1]+0.6, 0.1)
    new_x1 = [math.log(xi) for xi in new_x]
    new_y = CaculateInterval(c, new_x1)
    Draw(x, y, new_x, new_y, "a+blnx近似")
    # 拟合误差
    FittingErr = GetFittingError(c, x1, y)
    # 写入文件
    with open(result_file, "w+") as fo:
        fo.write("\n\\**********************************************************\\\n")
        fo.write("a+blnx近似\n")
        fo.write("x = ")
        for xi in x:
            fo.write(str(xi))
            fo.write("\t")
        fo.write("\n")
        fo.write("y = ")
        for yi in y:
            fo.write(str(yi))
            fo.write("\t")
        fo.write("\n")
        fo.write("近似函数: q(x) = {} + {} ln(x) \n".format(c[0], c[1]))
        fo.write("拟合误差: {0}\n".format(FittingErr))
        fo.write("\\**********************************************************\\\n")

'''
description: 计算拟合误差 Q = \Sigma_{k=1}^{n} (q(x_k) - y_k)^2
param {*} c 最佳平方逼近多项式系数, q(x) = a*exp(bx)
param {*} x
```

```python
    PARAM {*} Y
    RETURN {*} Q
    '''
def GetFittingError_q2_3(c, x, y):
    Err = 0
    for n, xn in enumerate(x):
        qx1 = CaculateExpression(c, xn)
        qx = math.exp(qx1)
        Err += pow((qx - y[n]), 2)
    return Err

# A*EXP(BX)近似
def Q2_3():
    x = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
    y = [58, 50, 44, 38, 34, 30, 29, 26, 25, 24]

    result_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'a*exp(bx)近似_result.txt')

    # A + BLNX
    y1 = [math.log(yi) for yi in y]
    c = GetLeastSquaresApproximationExpression(1, x, y1)
    # 画图
    new_x = np.arange(x[0]-0.5, x[-1]+0.6, 0.1)
    new_y1 = CaculateInterval(c, new_x)
    new_y = [math.exp(yi) for yi in new_y1]
    Draw(x, y, new_x, new_y, "a*exp(bx)近似")
    # 拟合误差
    FittingErr = GetFittingError_q2_3(c, x, y)
    # 写入文件
    with open(result_file, "w+") as fo:
        fo.write("\n\\********************************************************\\\n")
        fo.write("a*exp(bx)近似\n")
        fo.write("x = ")
        for xi in x:
            fo.write(str(xi))
            fo.write("\t")
        fo.write("\n")
        fo.write("y = ")
        for yi in y:
            fo.write(str(yi))
            fo.write("\t")
        fo.write("\n")
        fo.write("近似函数: q(x) = {0} * exp({1} * x) \n".format(math.exp(c[0]), c[1]))
        fo.write("拟合误差: {0}\n".format(FittingErr))
        fo.write("\\********************************************************\\\n")
```

```python

'''
description: 计算拟合误差 Q = \Sigma_{k=1}^{n} (q(x_k) - y_k)^2
param {*} c 最佳平方逼近多项式系数, q(x) = 1/(a+bx)
param {*} x
param {*} y
return {*} Q
'''
def GetFittingError_q2_4(c, x, y):
    Err = 0
    for n, xn in enumerate(x):
        qx1 = CaculateExpression(c, xn)
        qx = 1.0/qx1
        Err += pow((qx - y[n]), 2)
    return Err

# 1÷(a+bx)近似
def Q2_4():
    x = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
    y = [58, 50, 44, 38, 34, 30, 29, 26, 25, 24]

    result_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), '1÷(a+bx)近似_result.txt')

    # a + blnx
    y1 = [1.0/yi for yi in y]
    c = GetLeastSquaresApproximationExpression(1, x, y1)
    # 画图
    new_x = np.arange(x[0]-0.5, x[-1]+0.6, 0.1)
    new_y1 = CaculateInterval(c, new_x)
    new_y = [1.0/yi for yi in new_y1]
    Draw(x, y, new_x, new_y, "1÷(a+bx)近似")
    # 拟合误差
    FittingErr = GetFittingError_q2_4(c, x, y)
    # 写入文件
    with open(result_file, "w+") as fo:
        fo.write("\n\\*************************************************************\\\n")
        fo.write("1÷(a+bx)近似\n")
        fo.write("x = ")
        for xi in x:
            fo.write(str(xi))
            fo.write("\t")
        fo.write("\n")
        fo.write("y = ")
        for yi in y:
            fo.write(str(yi))
            fo.write("\t")
```

```python
            fo.write("\n")
            fo.write("近似函数: q(x) = 1 ÷ ( {0} + {1} * x) \n".format(math.exp(c[0]),
                c[1]))
            fo.write("拟合误差: {0}\n".format(FittingErr))
            fo.write("\\*********************************************************\\\n")

def main():
    Q1()
    Q2_1()
    Q2_2()
    Q2_3()
    Q2_4()

if __name__ == "__main__":

    # 获取当前文件路径
    current_path = os.path.abspath(__file__)
    sys.path.append(os.path.abspath(os.path.join(os.path.dirname(current_path), '
        ../')))
    # PRINT(SYS.PATH)
    # 调用 CHAPTER3 中的列主元高斯消去法
    from chapter3.q3 import MGauss_Caculate

    main()
```

4. 算例

对于题目中的第一问，分别求出 2, 3, 4 次最佳平方逼近多项式的拟合函数如下，拟合误差分别为 $3.2166667, 1.5102564, 1.4679487$，拟合函数的图像如图4.2所示。

$$p(x) = 74.3258 - 9.31136 * x + 0.435606 * x^2 \tag{4.10}$$

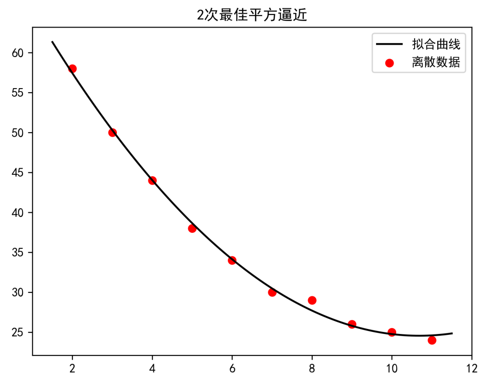
$$p(x) = 78.5424 - 11.9462 * x + 0.893939 * x^2 - 0.0235043 * x^3 \tag{4.11}$$

$$p(x) = 76.9924 - 10.6129 * x + 0.520542 * x^2 + 0.0181624 * x^3 - 0.00160256 * x^4 \tag{4.12}$$

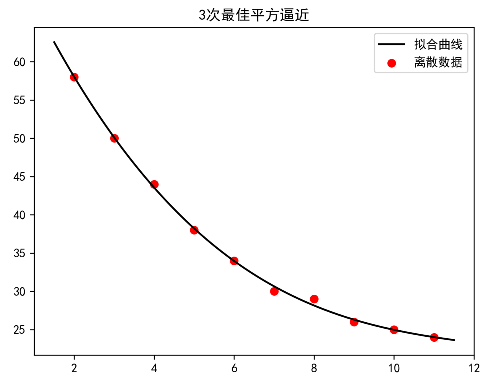
对于题目中的第二问，拟合函数如下，拟合的图像如图4.3所示，拟合误差分别为 $57.302270, 8.7788712, 41.844596, 7.7324292$。

$$q(x) = 18.160410958809152 + \frac{87.330000932616}{x} \tag{4.13}$$

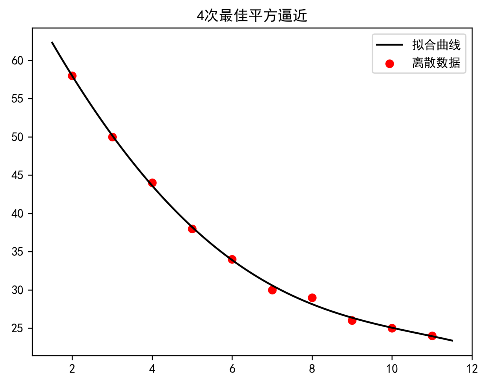
$$q(x) = 72.13901063630205 - 20.762410852503ln(x) \tag{4.14}$$



(a) 2 次最佳平方逼近



(b) 3 次最佳平方逼近

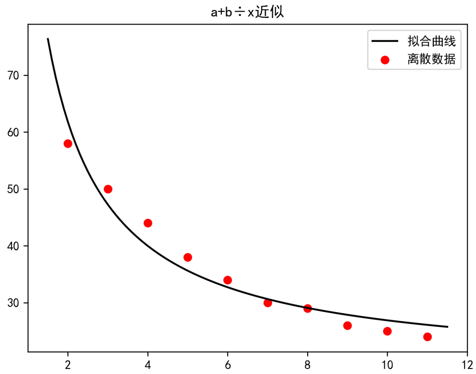


(c) 4 次最佳平方逼近

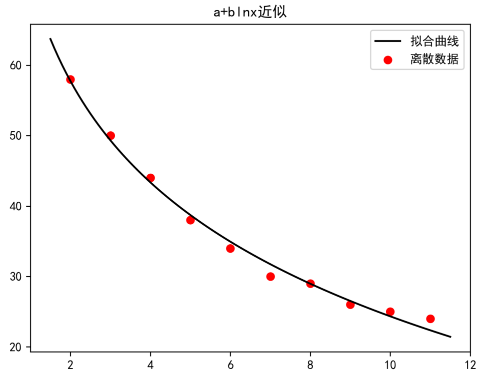
图 4.2　最佳平方逼近

$$q(x) = 65.29387947714481 * e^{-0.09915171210523323 * x} \tag{4.15}$$

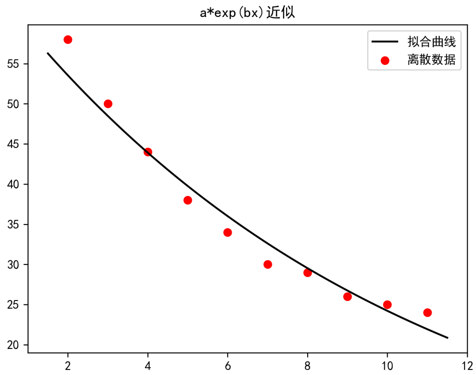
$$q(x) = \frac{1}{1.012042106641784 + 0.002829826611798 1353 * x} \tag{4.16}$$



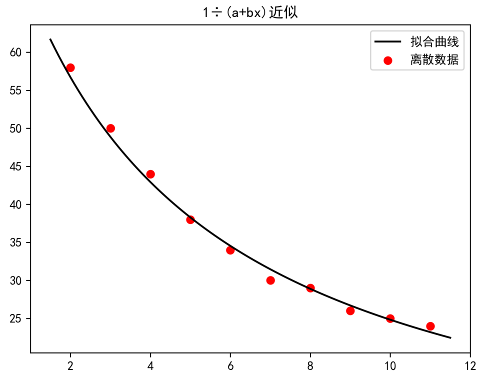
(a) $y = a + b/x$



(b) $y = a + b\ln x$



(c) $y = ae^{bx}$



(d) $y = 1/(a + bx)$

图 4.3　最佳平方逼近变形

5. 结论

(1) 7 种不同的拟合函数的拟合误差对比如表4.3所示。通过对比可以看到 4 次最佳平方逼近的拟合误差最小，但也是计算量最大的一个。

(2) 4 次最佳平方逼近的拟合误差相比于 3 次最佳平方逼近的拟合误差没有太多提升，反而加大了计算量。

(3) 在 4 种变形的拟合函数里，$y = a + b\ln x$ 和 $y = 1/(a + bx)$ 的闭合误差也比较小，但是都比 2 次最佳平方逼近的误差大。

(4) 整体上来说，相比于插值函数，次数不太高的最佳平方逼近的计算量不大。

表 4.3　拟合误差

拟合函数	2 次最佳平方逼近	3 次最佳平方逼近	4 次最佳平方逼近	
拟合误差	3.2166667	1.5102564	1.4679487	
拟合函数	$y = a + b/x$	$y = a + b\ln x$	$y = ae^{bx}$	$y = 1/(a + bx)$
拟合误差	57.302270	8.7788712	41.844596	7.7324292