第四章作业

目录

一、问题叙述

根据给定数据表，分别用插值和拟合的方法：

（1）确定 x=3 时对应的 y 值

（2）确定 y=25 时对应的 x 值

xi	**0.000**	**1.445**	**2.890**	**4.335**	**5.780**
yi	**1.8419**	**2.9633**	**18.2360**	**98.7410**	**529.2178**

二、Langrange 插值

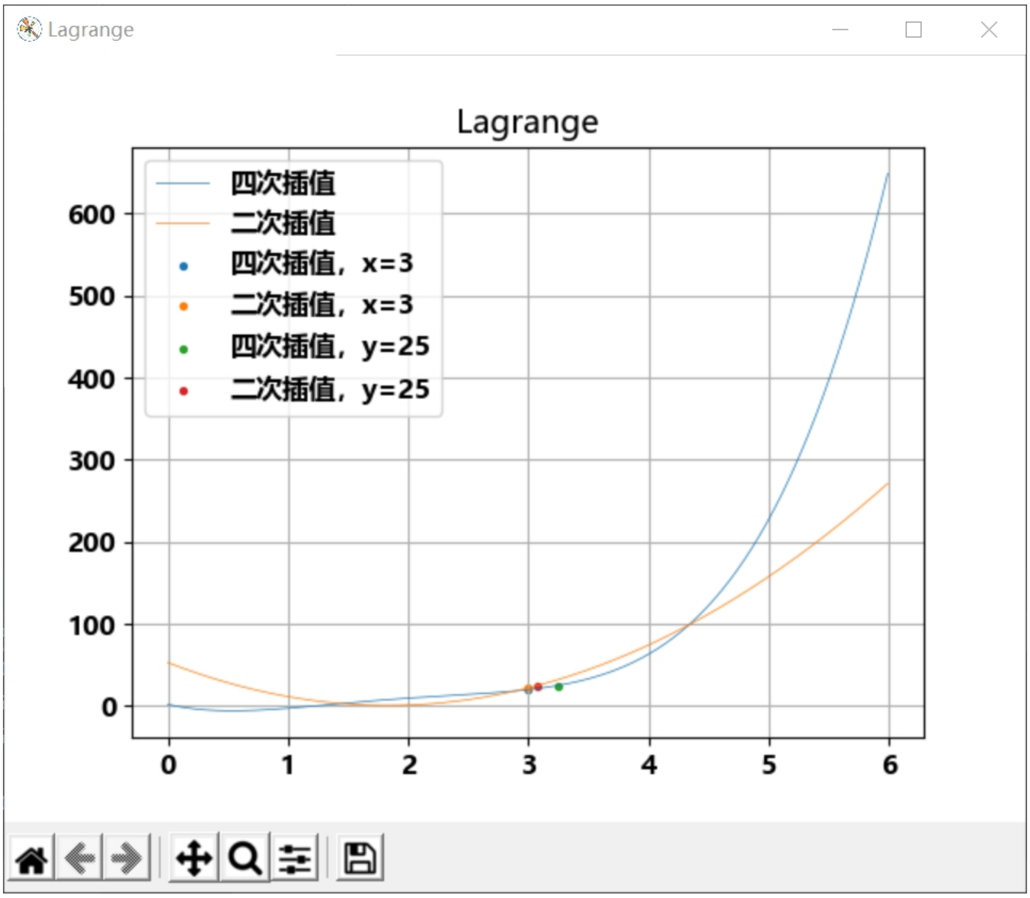
分别选取五个点做四次插值和三个靠近求解值的点做二次插值：

```python
x_1 = [0.0, 1.445, 2.89, 4.335, 5.78]
y_1 = [1.8419, 2.9633, 18.236, 98.741, 529.2178]
n_1 = 4      #四次插值

x_2 = [1.445, 2.89, 4.335]
y_2 = [2.9633, 18.236, 98.741]
n_2 = 2      #二次插值

x_r = 3
y_r = 25

# 求拉格朗日插值多项式的系数
def l_1(i, x):
    mul = 1
    for j in range(n_1 + 1):
        if j != i:
            mul *= (x-x_1[j]) / (x_1[i]-x_1[j])
    return mul
def l_2(i, x):
    mul = 1
    for j in range(n_2 + 1):
        if j != i:
            mul *= (x-x_2[j]) / (x_2[i]-x_2[j])
    return mul

# 拉格朗日插值公式
def f_1(x0):
    return y_1[0]*l_1(0, x0) + y_1[1]*l_1(1, x0) + y_1[2]*l_1(2, x0) + \
           y_1[3]*l_1(3, x0) + y_1[4]*l_1(4, x0)

def f_2(x0):
    return y_2[0]*l_2(0, x0) + y_2[1]*l_2(1, x0) + y_2[2]*l_2(2, x0)
```

求得结果如下：

	x = 3	y = 25
四次插值	y1 = 19.8964	x1 = 3.2467
二次插值	y2 = 22.0705	x2 = 3.0776

图形如下：



两种方式求得的Δx=0.1691，Δy=2.1741，相比于各自的跨度 0-6 和 0-600 来说是很小的，但由于没有真实值，所以无法探究哪一种方法得出的结果精度更高，不过总体来说都可以适用。

三、Newton 插值

牛顿插值相较于拉格朗日插值的优势是具有承袭性，无需因节点的增加减少而重新计算基函数，但二者在给定的相同有限个点上得出的插值函数是一样的，因此最终结果也相同。

代码主体部分如下：

```python
# 创建差商表
for i in range(n_1+1):
    table_1[i][0] = y_1[i]
for i in range(1, n_1+1):
    for j in range(i, n_1+1):
        table_1[j][i] = (table_1[j][i-1] - table_1[j-1][i-1]) / (x_1[j] - x_1[j-i])

for i in range(n_2+1):
    table_2[i][0] = y_2[i]
for i in range(1, n_2+1):
    for j in range(i, n_2+1):
        table_2[j][i] = (table_2[j][i-1] - table_2[j-1][i-1]) / (x_2[j] - x_2[j-i])

a_1 = np.diagonal(table_1)
a_2 = np.diagonal(table_2)

# 牛顿插值公式
def newton(x, x_temp, n, a):
    total = a[0]
    for k in range(1, n+1):
        product = 1
        for j in range(0, k):
            product *= x - x_temp[j]
        total += a[k] * product
    return total
```

四、二次样条插值

二次样条默认第一个节点处的二阶导为 0，即 $a_1=0$，通过节点处的函数值和内节点一阶导的连续性，可求解 3*(n-1)-1 个未知系数。

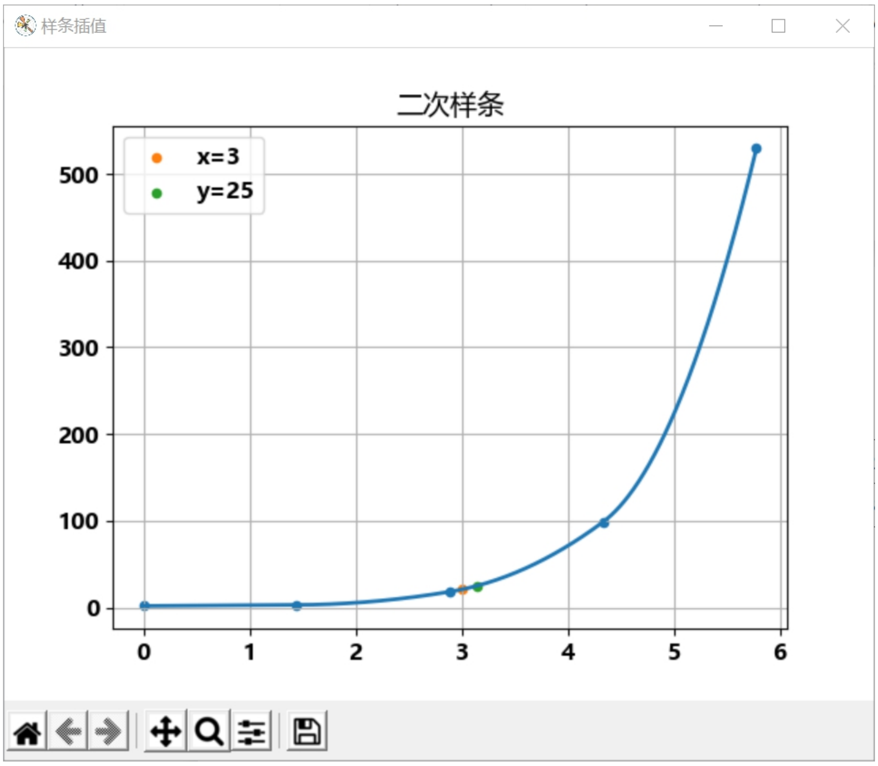
	0	1	2	3	4	5	6	7	8	9	10		
0	x[0]	1										b_1	y[0]
1	x[1]	1										c_1	y[1]
2			$x^2[1]$	x[1]	1							a_2	y[1]
3			$x^2[2]$	x[2]	1							b_2	y[2]
4						$x^2[2]$	x[2]	1				c_2	y[2]
5						$x^2[3]$	x[3]	1				a_3	y[3]
6									$x^2[3]$	x[3]	1	b_3	y[3]
7									$x^2[4]$	x[4]	1	c_3	y[4]
8	1	0	-2x[1]	-1								a_4	0
9			2x[2]	1	0	-2x[2]	1					b_4	0
10						2x[3]	1	0	-2x[3]	1		c_4	0

代码主体部分如下：

```python
# 二次样条系数矩阵
A = np.array([[x[0], 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
              [x[1], 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
              [0, 0, x[1]**2, x[1], 1, 0, 0, 0, 0, 0, 0],
              [0, 0, x[2]**2, x[2], 1, 0, 0, 0, 0, 0, 0],
              [0, 0, 0, 0, 0, x[2]**2, x[2], 1, 0, 0, 0],
              [0, 0, 0, 0, 0, x[3]**2, x[3], 1, 0, 0, 0],
              [0, 0, 0, 0, 0, 0, 0, 0, x[3]**2, x[3], 1],
              [0, 0, 0, 0, 0, 0, 0, 0, x[4]**2, x[4], 1],
              [1, 0, -2*x[1], -1, 0, 0, 0, 0, 0, 0, 0],
              [0, 0, 2*x[2], 1, 0, -2*x[2], -1, 0, 0, 0, 0],
              [0, 0, 0, 0, 0, 2*x[3], 1, 0, -2*x[3], -1, 0]])
B = np.array([y[0], y[1], y[1], y[2], y[2], y[3], y[3], y[4], 0, 0, 0])
C = (np.linalg.inv(A) @ B)

# 二次样条函数
def f(x0):
    if x[0]<= x0 <= x[1]:
        return C[0] * x0 + C[1]
    if x[1]< x0 <= x[2]:
        return C[2] * x0**2 + C[3] * x0 + C[4]
    if x[2]< x0 <= x[3]:
        return C[5] * x0**2 + C[6] * x0 + C[7]
    if x[3]< x0 <= x[4]:
        return C[8] * x0**2 + C[9] * x0 + C[10]
```

结果与图形如下：

	x = 3	y = 25
二次样条	y = 20.7719	x = 3.1445



二次样条在相邻节点处的函数是连续的，但是第一段区间是直线段，节点处的一阶导不一定连续，光滑程度可以进一步提高。

五、三次样条

三次样条理论上有 4n 个未知系数，通过三弯矩法可以将未知数的个数缩减为 n+1 个，但方程个数只有 n-1 个，因此还需要两个边界条件。本例中采用了自然边界条件，即首节点和尾节点的二阶导为 0。

$$h_i = x_i - x_{i-1} \qquad \mu_i = \frac{h_i}{h_i + h_{i+1}} \qquad \lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}}$$

$$g_i = \frac{6}{h_i + h_{i+1}} \left(f[x_i, x_{i+1}] - f[x_{i-1}, x_i] \right) = 6 f[x_{i-1}, x_i, x_{i+1}]$$

$$\Downarrow$$

采用自然边界条件，$\lambda_0 = \mu_n = 0$，$g_0 = g_n = 0$，$M_0 = M_n = 0$

$$\Downarrow$$

$$\begin{bmatrix} 2 & \lambda_1 & \\ \mu_2 & 2 & \lambda_2 \\ & \mu_3 & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix}$$

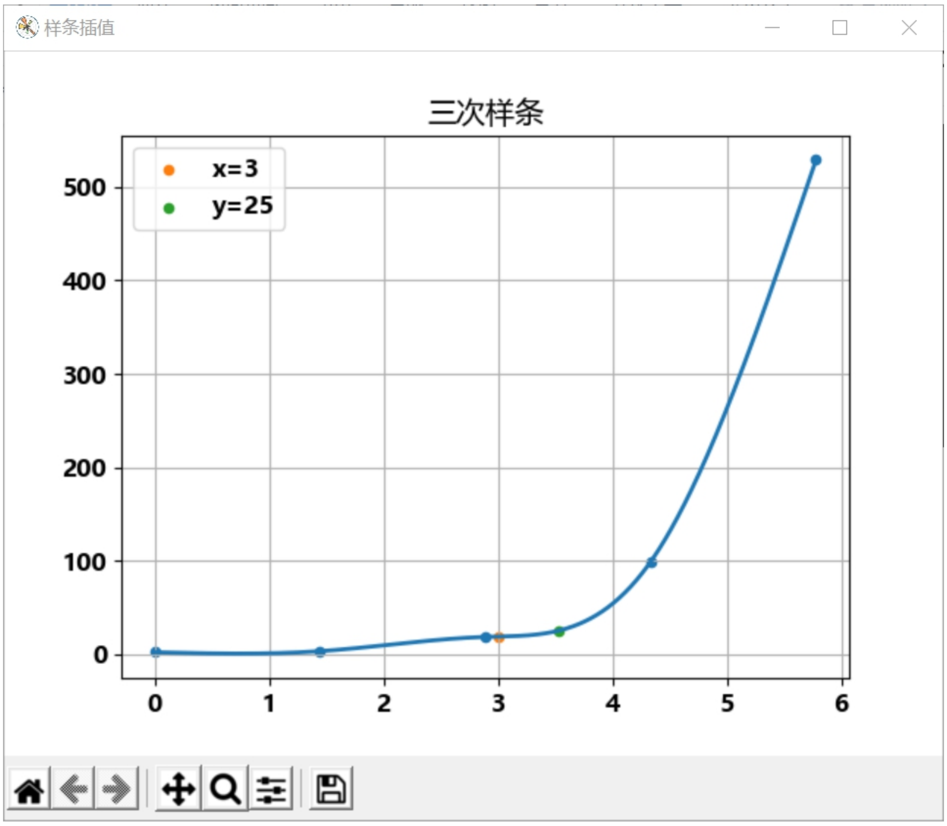
代码主体部分如下：

```python
# 三弯矩法的系数
h = np.zeros(n+1)
for i in range(1, n+1):
    h[i] = x[i] - x[i-1]
miu = np.zeros(n)
lam = np.zeros(n)
for i in range(1, n):
    miu[i] = h[i] / (h[i] + h[i+1])
    lam[i] = h[i+1] / (h[i] + h[i+1])

# 差商表
table = np.zeros((n+1, n-1))
for i in range(n + 1):
    table[i][0] = y[i]
for i in range(1, n-1):
    for j in range(i, n + 1):
        table[j][i] = (table[j][i-1] - table[j-1][i-1]) / (x[j] - x[j-i])
```

```python
# 三弯矩法的系数矩阵
A = np.array([[2, lam[1], 0], [miu[2], 2, lam[2]], [0, miu[3], 2]])
g = np.zeros(n-1)
for i in range(n-1):
    g[i] = 6 * table[i+2][2]
M = np.linalg.inv(A) @ g

# 三次样条函数
def f(x0):
    if x[0]<= x0 <= x[1]:
        return M[0]*(x0-x[0])**3/(6*h[1]) + y[0]*(x[1]-x0)/h[1] + \
               (y[1]-M[0]*(h[1]**2)/6)*(x0-x[0])/h[1]
    if x[1]< x0 <= x[2]:
        return (M[1]*(x0-x[1])**3 + M[0]*(x[2]-x0)**3) / (6*h[2]) + \
               (y[1]-M[0]*h[2]**2/6)*(x[2]-x0)/h[2] + (y[2]-M[1]*(h[2]**2)/6)*(x0-x[1])/h[2]
    if x[2]< x0 <= x[3]:
        return (M[2]*(x0-x[2])**3 + M[1]*(x[3]-x0)**3)/(6*h[3]) + \
               (y[2]-M[1]*h[3]**2/6)*(x[3]-x0)/h[3] + (y[3]-M[2]*(h[3]**2)/6)*(x0-x[2])/h[3]
    if x[3]< x0 <= x[4]:
        return (M[2]*(x[4]-x0)**3)/(6*h[4]) + (y[3]-M[2]*h[4]**2/6)*(x[4]-x0)/h[4] + \
               y[4]*(x0-x[3])/h[4]
```

结果与图形如下：

	x = 3	y = 25
三次样条	y = 18.60051	x = 3.5328



可以明显看到光滑度有了显著的提升。

六、指数函数拟合

由散点图可以看出，该曲线一阶导和二阶导均大于 0，所以考虑指数函数 y=a*e^bx 进行拟合。

$$\boxed{y = ae^{bx}}$$

$$\Downarrow$$

$$\boxed{\ln y = \ln a + bx}$$

$$\Downarrow \quad y' = \ln y,\ A = \ln a$$

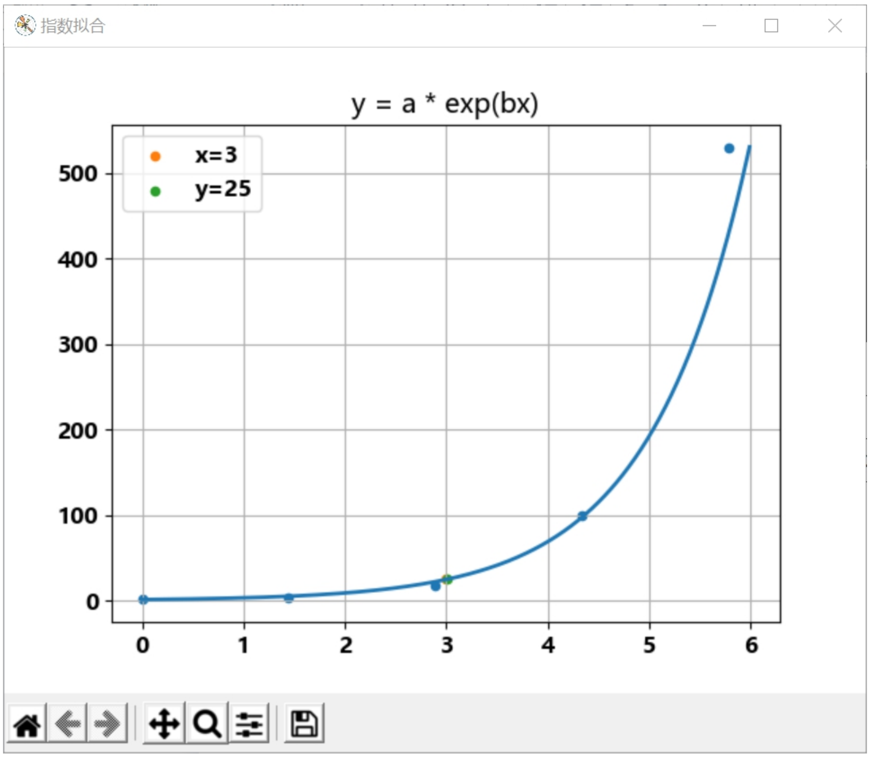
$$\boxed{y' = A + bx}$$

代码主体部分如下：

```
# 指数拟合
lny = []
x_square = []
x_lny = []
for i in range(n + 1):
    lny.append(math.log(y[i]))
    x_square.append(x[i] ** 2)
    x_lny.append(x[i] * lny[i])

x_sum = sum(x)
lny_sum = sum(lny)
x_square_sum = sum(x_square)
x_lny_sum = sum(x_lny)

# 系数矩阵
A = np.array([[n+1, x_sum], [x_sum, x_square_sum]])
B = np.array([lny_sum, x_lny_sum])
a, b = np.linalg.inv(A) @ B

# 指数拟合函数
def f(x0):
    return math.exp(a + b*x0)
```

结果与图形如下：

	x = 3	y = 25
指数拟合	y = 24.6740	x = 3.0128



残差、最大偏差、均方误差：

	x1	x2	x3	x4	x5
残差 xi-f(xi)	0.7060	-2.0402	-3.8045	1.6528	101.5443
最大偏差Δx	101.5443				
均方误差 MSE	2066.62				

前四个点与真实值都很接近，但是最后一个点的偏差非常大，导致均方误差也很大。

七、代数多项式拟合

也可以考虑用代数多项式进行拟合，为了避免方程组的系数矩阵是病态的，将最高次项设为二次。

$$\boxed{y = a_0 + a_1 x + a_2 x^2}$$

$$\Downarrow$$

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

$$\Downarrow$$

$$\boxed{A = X^T \cdot X \qquad b = X^T \cdot Y}$$

$$\Downarrow$$

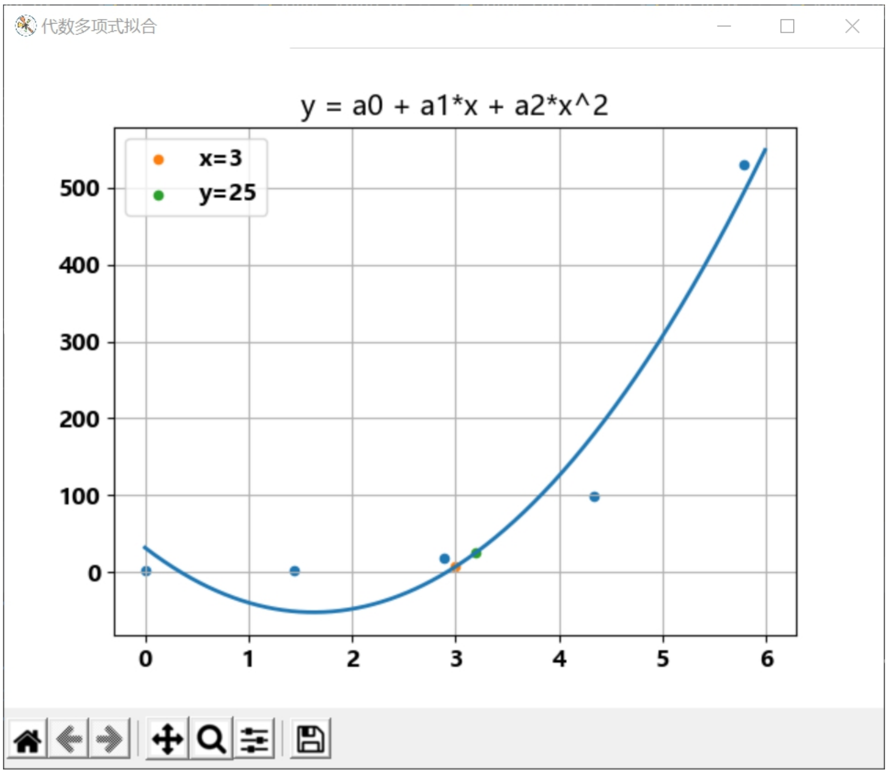
$$\boxed{A \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = b}$$

代码主体部分如下：

```
# 二次函数拟合
X = np.zeros((n+1, 3))
Y = np.zeros(n+1)
# 构造范德蒙行列式并求解系数
for i in range(n + 1):
    X[i] = [1, x[i], x[i]**2]
    Y[i] = y[i]
A = X.T @ X
B = X.T @ Y
C = np.linalg.inv(A) @ B

# 拟合的二次函数
def f(x0):
    return C[0] + C[1]*x0 + C[2]*x0**2
```

结果与图形如下：

	x = 3	y = 25
代数多项式拟合	y = 7.3489	x = 3.1907



残差、最大偏差、均方误差：

	x1	x2	x3	x4	x5
残差 xi-f(xi)	-30.2441	53.8129	20.0279	-80.5160	39.9200
最大偏差Δx	80.52				
均方误差 MSE	2411.5				

二次多项式拟合的残差相对而言数值较为接近，最大偏移要比指数拟合的小，但均方误差更大一些，因此在最小二乘法的前提下，应采用指数拟合的形式。

八、小结

对插值方法来说，拉格朗日多项式和牛顿多项式在次数偏高的情况下很可能会产生龙格现象，而二次样条在光滑程度不是很到位，因此三次样条是极好的选择，它也与节点的个数无关，可以在多个节点上插值。

对拟合方法来说，首先要通过散点图确定所选取的拟合函数，函数的选择往往不是唯一的，所以可以通过最小二乘原则确定拟合效果最好的一个函数。