实验 3 曲线拟合的最小二乘法

201311211914
赵帅帅
2016–5–10

一、实验问题

在某冶炼过程中，通过实验检测得到含碳量与时间关系的数据如下，试求含碳量 y 与时间 t 内在关系的拟合曲线。

t	0	5	10	15	20	25	30	35	40	45	50
y	0	1.27	2.16	2.86	3.44	3.87	4.15	4.37	4.51	4.58	4.02

二、实验要求

1. 用最小二乘法进行三次多项式的曲线拟合。
2. 计算 y_i 与 $y(t_j)$ 的误差，$j = 1, 2, \cdots, 11$。
3. 另取一个拟合函数，进行拟合效果的比较
4. 绘制曲线拟合图形。

三、实验目的与意义

1. 掌握曲线拟合的最小二乘法。
2. 探求拟合函数的选择与拟合精度间的关系。

四、算法

输入 自变量的值序列 $x[N]$，观察函数值序列 $y[N]$，每一组值的权重序列 $\rho[N]$[1]，多项式拟合的最高次幂 M。

输出 拟合曲线的表达式。

(1) 设置函数类 $\Phi = span\{\boldsymbol{\varphi}_i(x) = x^i, 0 \leqslant i \leqslant M\}$

(2) 引入向量

$$\begin{aligned}\boldsymbol{\varphi}_j &= (\boldsymbol{\varphi}_j(x_0), \boldsymbol{\varphi}_j(x_1), \cdots, \boldsymbol{\varphi}_j(x_{N-1}))^T, j = 0, 1, \cdots, M \\ \boldsymbol{f} &= (y_0, y_1, \cdots, y_{N-1})^T\end{aligned}$$

[1]本次实验 $\rho[N] = \{1, 1, \cdots, 1\}$。

计加权内积

$$
\begin{aligned}
(\boldsymbol{\varphi}_k, \boldsymbol{\varphi}_i) &= \sum_{j=0}^{N-1} \rho(x_j)\boldsymbol{\varphi}_k(x_j)\boldsymbol{\varphi}_i(x_j) \\
(\boldsymbol{f}, \boldsymbol{\varphi}_i) &= \sum_{j=0}^{N-1} \rho(x_j) y_j \boldsymbol{\varphi}_i(x_j), k, i = 0, 1, \cdots, M
\end{aligned}
$$

(3) 计算正则方程组

$$
\begin{bmatrix}
(\boldsymbol{\varphi}_0, \boldsymbol{\varphi}_0) & (\boldsymbol{\varphi}_0, \boldsymbol{\varphi}_1) & \cdots & (\boldsymbol{\varphi}_0, \boldsymbol{\varphi}_M) \\
(\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_0) & (\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_1) & \cdots & (\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_M) \\
\vdots & \vdots & \ddots & \vdots \\
(\boldsymbol{\varphi}_M, \boldsymbol{\varphi}_0) & (\boldsymbol{\varphi}_M, \boldsymbol{\varphi}_1) & \cdots & (\boldsymbol{\varphi}_M, \boldsymbol{\varphi}_M)
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_M \end{bmatrix}
=
\begin{bmatrix} (\boldsymbol{f}, \boldsymbol{\varphi}_0) \\ (\boldsymbol{f}, \boldsymbol{\varphi}_0) \\ \vdots \\ (\boldsymbol{f}, \boldsymbol{\varphi}_M) \end{bmatrix}
$$

(4) 用平方根法或 **SOR**迭代法解方程组得到 $a = (a_0^*, a_1^*, \cdots, a_M^*)^T$

(5) 拟合曲线表达式 $\boldsymbol{\varphi}^*(x) = a_0^* + a_1^* x + \cdots + a_M^* x^M$

五、实验代码

```
#include<stdio.h>
#include<iostream>
#include<math.h>
#include<stdbool.h>
// M = 多项式最高次幂+ 1
// N 为离散数据组数
const int M = 4, N = 11;
using namespace std;
double t[N] = {0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50};
double y[N] = {0, 1.27, 2.16, 2.86, 3.44, 3.87, 4.15, 4.37, 4.51, 4.58,
    4.02};

// 点乘
double dot_product(double a[N], double b[N])
{
    double result = 0;
    for(int i = 0 ; i < N; i ++)
    {
        result += a[i] * b[i];
    }
    return result;
}

// 解正定线性方程组from 实验1
bool Choleskey(double a[M][M], double b[M], double x[M])
{
    double sum = 0;
    double y[M] = {0};
    for(int k = 0; k <= M - 1; k ++)
    {
        sum = 0;
```

```
        for(int m = 0; m <= k - 1; m ++)
        {
            sum += a[k][m] * a[k][m];
        }
        a[k][k] = sqrt(a[k][k] - sum);
        if(a[k][k] == 0)
        {
            printf("Divided by zero!");
            return false;
        }
        for(int i = k + 1; i <= M - 1; i ++)
        {
            sum = 0;
            for(int m = 0; m <= k - 1; m ++)
            {
                sum += a[i][m] * a[k][m];
            }
            a[i][k] = (a[i][k] - sum) / a[k][k];
        }
        sum = 0;
        for(int m = 0; m <= k - 1; m ++)
        {
            sum += a[k][m] * y[m];
        }
        y[k] = (b[k] - sum) / a[k][k];
    }
    x[M - 1] = y[M - 1] / a[M - 1][M - 1];
    for(int k = M -2; k >= 0; k --)
    {
        sum = 0;
        for(int m = k + 1; m <= M - 1; m ++)
        {
            sum += a[m][k] * x[m];
        }
        x[k] = (y[k] - sum) / a[k][k];
    }
    return true;
}

int main(void)
{
    //
    double phi[M][N] = {0};
    for(int i = 0; i < M; i ++)
    {
        for(int j = 0; j < N; j ++)
        {
            phi[i][j] = pow(t[j], i);
        }
    }
    // 系数矩阵
    double a[M][M] = {0};
    // 值向量
    double b[M] = {0};
    // 待求解系数向量
    double x[M] = {0};
    for(int i = 0; i < M; i ++)
    {
```

```
        for(int j = 0; j < M; j ++)
        {
            a[i][j] = dot_product(phi[i], phi[j]);
        }
        b[i] = dot_product(y, phi[i]);
    }
    Choleskey(a, b, x);
    // 输出表达式
    cout<<"y = "<<x[0];
    for(int i = 1 ; i < M; i ++)
    {
        if(x[i] < 0)
            cout<<" - "<<fabs(x[i])<<"x^"<<i;
        else
            cout<<" - "<<x[i]<<"x^"<<i;
    }
    cout<<endl;
    // 输出误差
    for(int i = 0 ; i < N; i ++)
    {
        double result = 0;
        for(int j = 0; j < M; j ++)
        {
            result += x[j] * pow(t[i], j);
        }
        cout<<"y"<<i<<" - y"<<i<<"' = "<<y[i] - result<<endl;
    }
    return 0;
}
```

六、实验结果

操作系统： Windows 10 企业版

GCC 版本： 4.7.1

G++ 版本： 4.7.1

1. 三次多项式拟合结果。

```
y = 0.0887413 - 0.233659x^1 - 0.00338392x^2 - 6.79099e-006x^3
y0 - y0' = -0.0887413
y1 - y1' = 0.0967133
y2 - y2' = 0.0662704
y3 - y3' = 0.00483683
y4 - y4' = -0.0226807
y5 - y5' = -0.0513753
y6 - y6' = -0.0863403
y7 - y7' = -0.042669
y8 - y8' = 0.0545455
y9 - y9' = 0.21021
y10 - y10' = -0.140769
```

图 1: 运行结果

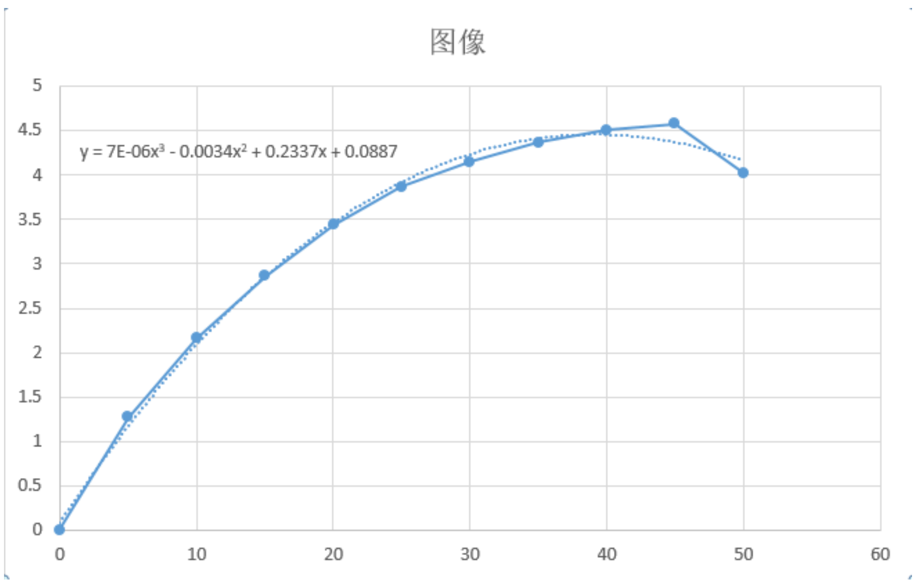

图 2: 曲线图形

2. 另取一拟合函数拟合结果。
   取曲线拟合方程 $y = ax^{\frac{1}{2}} + c$，变换关系为 $\overline{x} = x^{\frac{1}{2}}$，得到变换的结果为 $y = a\overline{x} + c$。
   程序运行结果如下:

```
y = 0.677088x^(1/2) + 0.110245
y0 - y0' = -0.110245
y1 - y1' = -0.35426
y2 - y2' = -0.0913852
y3 - y3' = 0.127405
y4 - y4' = 0.301725
y5 - y5' = 0.374315
y6 - y6' = 0.331191
y7 - y7' = 0.254048
y8 - y8' = 0.117475
y9 - y9' = -0.0722893
y10 - y10' = -0.87798
```

图 3: 运行结果

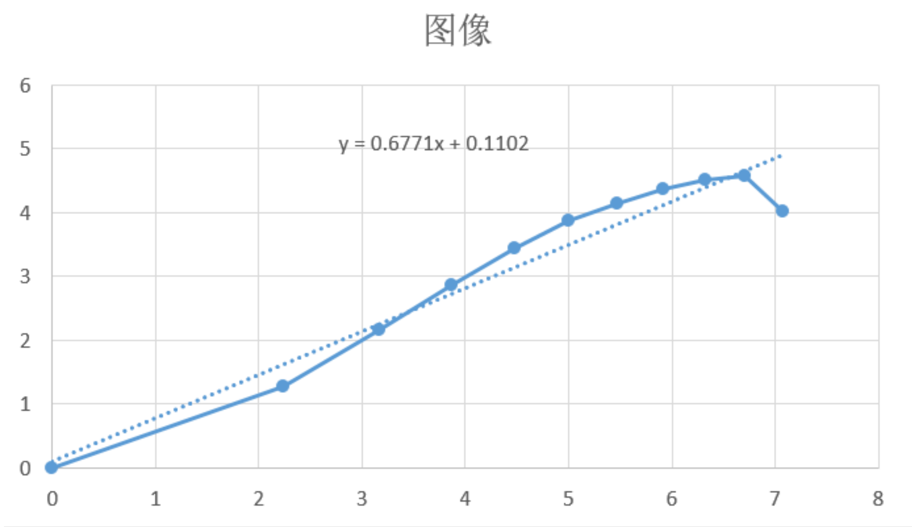

图 4: 曲线图形 (横轴为变量的平方根)

3. 两个拟合函数比较。
   直接观看两个拟合函数的拟合误差来看，多项式拟合效果较好。

七、实验感受

1. 本次实验需要我们自己抽象出代码所需要的算法，花了一点时间。

2. 代码写起来还是很简单的。