北京航空航天大學
BEIHANG UNIVERSITY

数值分析大作业

计算实习题（3）

院（系）名称　宇航学院

学　　号　ZY2115107

学生姓名　段诗阳

2021 年 12 月 6 日

一、题目要求

关于x, y, t, u, v, w的方程组如下

$$\begin{cases} 0.5cost + u + v + w - x = 2.67 \\ t + 0.5sinu + v + w - y = 1.07 \\ 0.5t + u + cosv + w - x = 3.74 \\ t + 0.5u + v + sinw - y = 0.79 \end{cases}$$

关于$\{t, u, z\}$的二维数表已给出。试求：

1、使用数值方法求出$f(x, y)$在区域$D = \{(x, y)|0 \le x \le 0.8, 0.5 \le y \le 1.5\}$上的近似表达式

$$p(x, y) = \sum_{r=0}^{k} \sum_{s=0}^{k} c_{rs} x^r y^s$$

要求$p(x, y)$以最小的k值达到以下精度

$$\sigma = \sum_{i=0}^{10} \sum_{j=0}^{20} [f(x_i, y_i) - p(x_i, y_i)]^2 \le 10^{-7}$$

其中$x_i = 0.08i, y_j = 0.5 + 0.05j$。

2、计算$f(x_i^*, y_i^*), p(x_i^*, y_i^*)$ $(i = 1,2, \dots, 8; j = 1,2, \dots 5)$的值，以观察$p(x, y)$逼近$f(x, y)$的效果，其中$x^* = 0.1i, y^* = 0.5 + 0.2j$。

题中要求在作二元插值时使用分片二次代数插值，并打印出$\{x, y, f\}$的二位数表。

二、算法流程

题中给出了$\{t, u\} \to z$的二维数表，所要求的是$\{x, y\} \to f$的二维数表，其中$z = f$。而$\{t, u\}$和$\{x, y\}$的映射关系由非线性方程组确定，因此需要先用牛顿迭代法求解非线性方程组。在对二维函数$z = z(t, u)$进行插值计算时，需要使用分片二次插值和乘积型最小二乘法进行插值和曲面拟合。

1、牛顿法求解非线性方程组

$$\begin{cases} 0.5cost + u + v + w - x = 2.67 \\ t + 0.5sinu + v + w - y = 1.07 \\ 0.5t + u + cosv + w - x = 3.74 \\ t + 0.5u + v + sinw - y = 0.79 \end{cases}$$

对于上述方程组来说，向量x和向量y已知，因此可令向量$Var = \{t, u, v, w\}^T$，设定精度水平$\epsilon = 10^{-12}$和最大迭代次数 2000，迭代步骤：

设定初值$Var^{(0)} = \{1.00,1.00,1.00,1.00\}^T$,

求解线性方程组

$$F'\left(Var^{(k)}\right)\delta = -F$$

$$Var^{(k+1)} = Var^{(k)} + \delta$$

迭代终止条件为$\frac{||delta||_\infty}{\left||Var^{(k)}|\right|_\infty} < \epsilon.$

其中：

$$F = \begin{bmatrix} 0.5cost + u + v + w - x - 2.67 \\ t + 0.5sinu + v + w - y - 1.07 \\ 0.5t + u + cosv + w - x - 3.74 \\ t + 0.5u + v + sinw - y - 0.79 \end{bmatrix}$$

$$F' = \begin{bmatrix} -0.5sint & 1 & 1 & 1 \\ 1 & 0.5cosu & 1 & 1 \\ 0.5 & 1 & -sinv & 1 \\ 1 & 0.5 & 1 & cosw \end{bmatrix}$$

在求解线性方程组时用到了列主元的高斯消元法，可以直接调用之前作业中的函数。

2、分片二次代数插值

二元函数$z = z(t,u)$的代数插值与一元函数类似，设

$$t_i = t_0 + ih \quad (i = 0,1,\dots,n)$$

$$u_j = u_0 + j\tau \quad (j = 0,1,\dots,m)$$

由已知数表可知$t_0 = u_0 = 0$，$h = 0.2$，$\tau = 0.4$，$n = m = 5$。

首先选定插值节点：

对于(t_i, u_j)来说，当满足

$$\begin{cases} t_k - \frac{h}{2} < t \le t_k + \frac{h}{2}, & 2 \le k \le n-2 \\ u_r - \frac{\tau}{2} < u \le u_r + \frac{\tau}{2}, & 2 \le r \le m-2 \end{cases}$$

选择$(t_a, u_b)(a = k-1, k, k+1; b = r-1, r, r+1)$为插值节点，如果$t \le t_1 + \frac{h}{2}$或者$t > t_{n-1} - \frac{h}{2}$，则取$k = 1$或$k = n-1$；如果$u \le u_1 + \frac{\tau}{2}$或者$u > u_{n-1} - \frac{\tau}{2}$，则取$r = 1$或$r = m-1$。

然后确定插值多项式

$$p_{22}(t,u)=\sum_{a=k-1}^{k+1}\sum_{b=r-1}^{r+1}\lambda_a(t)*\lambda_b(u)*z$$

其中

$$\lambda_a(t)=\prod_{\substack{c=k-1\\c!=k}}^{k+1}\frac{t-t_c}{t_a-t_c}\quad(a=k-1,k,k+1)$$

$$\lambda_b(u)=\prod_{\substack{d=k-1\\d!=k}}^{r+1}\frac{u-u_d}{t_b-t_d}\quad(b=r-1,r,r+1)$$

带入(t_i,u_j)即得到z_{ij}，可以直接打印二维数表$\{x_i,y_j,z_{ij}=f(x_i,y_j)\}$

3、使用最小二乘法进行曲面拟合

题目中给定拟合函数的表达式为

$$p(x,y)=\sum_{r=0}^{k}\sum_{s=0}^{k}c_{rs}x^r y^s$$

给定精度水平10^{-7}和最大K_{max}.=8，迭代步骤如下：

计算矩阵B和矩阵G

$$B=\begin{bmatrix}1 & x_0 & x_0^2 & \dots & x_0^k\\ 1 & x_1 & x_1^2 & \dots & x_1^k\\ 1 & \dots & \dots & \dots & \dots\\ 1 & x_n & x_n^2 & \dots & x_n^k\end{bmatrix}$$

$$G=\begin{bmatrix}1 & y_0 & y_0^2 & \dots & y_0^k\\ 1 & y_1 & y_1^2 & \dots & y_1^k\\ 1 & \dots & \dots & \dots & \dots\\ 1 & y_n & y_n^2 & \dots & y_n^k\end{bmatrix}$$

计算系数矩阵C

$$C=(B^TB)^{-1}B^TZG(G^TG)^{-1}$$

计算拟合值$p(x,y)$

拟合精度使用误差平方和

$$\sigma=\sum_{j=0}^{n}\sum_{i=0}^{m}[p(x,y)-z]^2$$

当$\sigma \leq 10^{-7}$时停止迭代，否则$k = k + 1$，进入下一轮计算。

4、使用(x^*, y^*)进行验证

使用牛顿迭代法和分片二次代数插值，得到一组新的$\{t^*, u^*, z^*\}$，即可得到新的二维数表$\{x^*, y^*, z^* = f(x^*, y^*)\}$。

使用上述曲面拟合函数及之前求得的系数矩阵C和k，得到一组新的拟合值$p(x^*, y^*)$，输出至控制台窗口。

三、计算程序

```
#include <iostream>
#include <math.h>
#include <vector>
#include <iomanip>
#include <fstream>

using namespace std;

constexpr auto M = 11;
constexpr auto N = 21;
constexpr auto epsilon = 1e-12;

void MatMUL(vector<vector<double> > A, vector<vector<double> > B, 
vector<vector<double> >& C);
void MatT(vector<vector<double> > A, vector<vector<double> >& B);
void MatINV(vector<vector<double> > A, vector<vector<double> >& B);
void MatStar(vector<vector<double> > A, vector<vector<double> >& B);
double MatValue(vector<vector<double> > A);
void GetCrs(vector<vector<double> >& B, vector<vector<double> >& G, 
vector<vector<double> >& Z, vector<vector<double> >& C);
void MatTINV(vector<vector<double> >& A, vector<vector<double> >& B);

int Kk = 0; // 拟合多项式的系数

// 矩阵乘法
void MatMUL(vector<vector<double> > A, vector<vector<double> > B, 
vector<vector<double> >& C) {
    int i, j, m;
    vector<vector<double> >temp(A.size(), vector<double>(B[0].size()));
    C.swap(temp);   //容量够用时，resize 不会改变，可以用 swap 重新调整大小

    for (i = 0; i < C.size(); i++) {
        for (j = 0; j < C[0].size(); j++) {
```

```
            C[i][j] = 0;
            for (m = 0; m < A[0].size(); m++)
                C[i][j] += A[i][m] * B[m][j];
        }
    }
}

// 矩阵转置
void MatT(vector<vector<double> > A, vector<vector<double> >& B) {
    int i, j;
    int m = A.size();
    int n = A[0].size();
    vector<vector<double> >temp(n, vector<double>(m));
    B.swap(temp);

    for (i = 0; i < B.size(); i++)
        for (j = 0; j < B[0].size(); j++)
            B[i][j] = A[j][i];
}

// 矩阵求逆
void MatINV(vector<vector<double> > A, vector<vector<double> >& B) {
    int i, j;
    int n = A.size();
    vector<vector<double> >t(n, vector<double>(n));
    if (n != A[0].size()) {
        cout << "矩阵必须为方阵" << endl;
        return;
    }
    B.swap(t);

    double MatVal = MatValue(A);    // 求矩阵的行列式
    vector<vector<double> >temp(n, vector<double>(n));  // 余子式矩阵

    if (MatVal == 0) {
        cout << "矩阵的行列式为0" << endl;
        return;
    }
    else {
        MatStar(A, temp);    // 求 A*
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                B[i][j] = temp[i][j] / MatVal;
    }
```

```
}

// 矩阵的余子式
void MatStar(vector<vector<double> > A, vector<vector<double> >& B) {
    int i, j, k, t;
    int n = A.size();
    vector<vector<double> >temp(n - 1, vector<double>(n - 1));
    if (n == 1)
        B[0][0] = 1;
    else {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++){
                // 构造余子式
                for (k = 0; k < n - 1; k++)
                    for (t = 0; t < n - 1; t++)
                        temp[k][t] = A[k >= j ? k + 1 : k][t >= i? t + 1 :
t];
                if ((i + j) % 2 == 1)
                    B[i][j] = -MatValue(temp);
                else
                    B[i][j] = MatValue(temp);
            }
        }
    }

}

// 矩阵的值 |A|
double MatValue(vector<vector<double> > A)
{
    int n = A.size();

    if (n == 1)
    {
        return A[0][0];
    }

    double ans = 0;
    vector<vector<double> >temp(n - 1, vector<double>(n - 1));
    int i, j, k;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - 1; j++)
```

```
        {
            for (k = 0; k < n - 1; k++)
            {
                temp[j][k] = A[j + 1][(k >= i) ? k + 1 : k];

            }
        }
        double t = MatValue(temp);
        if (i % 2 == 0)
        {
            ans += A[0][i] * t;
        }
        else
        {
            ans -= A[0][i] * t;
        }
    }
    return ans;
}

// 矩阵求逆 Gauss
void MatInvGauss(vector<vector<double> >A, vector<vector<double> >& Ainv)
{
    Ainv = A;
    int n = Ainv.size();
    vector<int>is(n, 0);
    vector<int>js(n, 0);

    int i, j, k;
    double d, p;
    for (k = 0; k < n; k++)
    {
        d = 0.0;
        for (i = k; i <= n - 1; i++)
            for (j = k; j <= n - 1; j++)
            {
                p = fabs(Ainv[i][j]);
                if (p > d) { d = p; is[k] = i; js[k] = j; }
            }
        if (0.0 == d)
        {
            printf("error, not inv!\n");
            return;
        }
```

```
        if (is[k] != k)
            for (j = 0; j <= n - 1; j++)
            {
                p = Ainv[k][j];
                Ainv[k][j] = Ainv[is[k]][j];
                Ainv[is[k]][j] = p;
            }
        if (js[k] != k)
            for (i = 0; i <= n - 1; i++)
            {
                p = Ainv[i][k];
                Ainv[i][k] = Ainv[i][js[k]];
                Ainv[i][js[k]] = p;
            }
        Ainv[k][k] = 1.0 / Ainv[k][k];
        for (j = 0; j <= n - 1; j++)
            if (j != k)
            {
                Ainv[k][j] *= Ainv[k][k];
            }
        for (i = 0; i <= n - 1; i++)
            if (i != k)
                for (j = 0; j <= n - 1; j++)
                    if (j != k)
                    {
                        Ainv[i][j] -= Ainv[i][k] * Ainv[k][j];
                    }
        for (i = 0; i <= n - 1; i++)
            if (i != k)
            {
                Ainv[i][k] = -Ainv[i][k] * Ainv[k][k];
            }
    }
    for (k = n - 1; k >= 0; k--)
    {
        if (js[k] != k)
            for (j = 0; j <= n - 1; j++)
            {
                p = Ainv[k][j];
                Ainv[k][j] = Ainv[js[k]][j];
                Ainv[js[k]][j] = p;
            }
        if (is[k] != k)
            for (i = 0; i <= n - 1; i++)
```

```
            {
                p = Ainv[i][k];
                Ainv[i][k] = Ainv[i][is[k]];
                Ainv[i][is[k]] = p;
            }
    }
    return;
}

// 打印矩阵
void MatrixPri(vector<vector<double> > A) {
    for (int i = 0; i < A.size(); i++) {
        for (int j = 0; j < A[0].size(); j++) {
            cout << setprecision(4);
            cout << A[i][j] << " ";
        }
        cout << endl;
    }
}

// 测试函数：矩阵的操作
void MatTest() {
    vector<vector<double> > A = { {2.1,1.4,4.7},
                                  {8.4,1.6,7.0},
                                  {9.5,4.1,6.8} };
    vector<vector<double> > B = { {4.5,4.8,1.4},
                                  {6.5,4.1,8.7},
                                  {5.8,1.0,2.4} };
    vector<vector<double> >C(3, vector<double>(3));
    cout << "原矩阵 A" << endl;
    MatrixPri(A);
    cout << "原矩阵 B" << endl;
    MatrixPri(B);

    MatMUL(A, B, C);
    cout << "矩阵乘法" << endl;
    MatrixPri(C);

    MatT(A, C);
    cout << "矩阵转置" << endl;
    MatrixPri(C);

    cout << "矩阵的行列式 = " << MatValue(A) << endl;
```

```
    MatINV(B, C);
    cout << "矩阵逆" << endl;
    MatrixPri(C);

    cout << "矩阵 B" << endl;
    MatrixPri(B);
    MatInvGauss(B, C);
    cout << "高斯法求矩阵逆" << endl;
    MatrixPri(C);
    cout << "矩阵 B" << endl;
    MatrixPri(B);

    //GetCrs(A, B, B, C);
    //cout << "Test" << endl;
    //MatrixPri(C);
}

// 求取(B^T*B)^(-1)
void MatTINV(vector<vector<double> >& A, vector<vector<double> >& B) {
    vector<vector<double> >temp1;
    vector<vector<double> >temp2;

    MatT(A, temp1);
    MatMUL(temp1, A, temp2);
    //MatINV(temp2, B);   // 定义法求逆矩阵，精度未达要求
    MatInvGauss(temp2, B);  // Gauss 求逆矩阵，精度达到要求
}

// 求取 系数矩阵 C
void GetCrs(vector<vector<double> >& B, vector<vector<double> >& G,
vector<vector<double> >& Z, vector<vector<double> >& C) {
    vector<vector<double> >temp1;
    vector<vector<double> >temp2;
    vector<vector<double> >temp3;
    vector<vector<double> >temp4;

    MatTINV(B, temp1);  //temp1=(B^T*B)^(-1)
    //cout << "测试 1" << endl;
    //MatrixPri(temp1);
    MatTINV(G, temp2);  //temp2=(G^T*G)^(-1)
    //cout << "测试 2" << endl;
    //MatrixPri(temp2);
    MatT(B, temp3);
    MatMUL(temp3, Z, temp4);
```

```
    MatMUL(temp4, G, temp3);
    //cout << "测试 3" << endl;
    //MatrixPri(temp3);
    MatMUL(temp1, temp3, temp4);
    //cout << "测试 4" << endl;
    //MatrixPri(temp4);
    MatMUL(temp4, temp2, C);
}

// 求取向量的无穷范数
double Max(vector<double> A) {
    int i;
    double max = fabs(A[0]);
    for (i = 0; i < A.size(); i++) {
        if (fabs(A[i]) > max)
            max = fabs(A[i]);
    }
    return max;
}

// 列主元的 Gauss 消去法
void Gauss(vector<vector<double> >& A, vector<double>& B, vector<double>&
x) {
    int k, i, j;
    double temp;

    // 消元过程
    for (k = 0; k < A.size() - 1; k++) {
        int flag = k;
        temp = fabs(A[k][k]);
        for (j = k + 1; j < A.size(); j++)
            if (fabs(A[j][k]) > temp) {
                flag = j;
                temp = fabs(A[j][k]);   // 找出主元
            }

        if (flag != k) {
            for (j = k; j < A[0].size(); j++) {// 交换主元所在行全部元素
                swap(A[k][j], A[flag][j]);
            }
            swap(B[k], B[flag]);
        }

        for (i = k + 1; i < A.size(); i++) {// 消元
```

```
            temp = A[i][k] / A[k][k];
            for (j = k + 1; j < A[0].size(); j++)
                A[i][j] -= temp * A[k][j];
            B[i] -= temp * B[k];
        }
    }
    // 回代过程
    x[x.size() - 1] = B[B.size() - 1] / A[A.size() - 1][A[0].size() - 1];
    for (k = x.size() - 2; k >= 0; k--) {
        temp = 0;
        for (j = k + 1; j < x.size(); j++)
            temp += A[k][j] * x[j];
        x[k] = (B[k] - temp) / A[k][k];
    }
}

// Newton 法，求解非线性方程组
void Newton(vector<double>& x, vector<double>& y, vector<vector<double> >&
T, vector<vector<double> >& U) {
    int i, j, k;

    vector<double>Var(4);   // 非线性方程组的解向量，保存 t,u,v,w
    vector<double>F(4);
    vector<vector<double> >dF(4, vector<double>(4));
    vector<double>delta(4); // 线性方程组的解

    for (i = 0; i < x.size(); i++) {
        for (j = 0; j < y.size(); j++) {
            Var = { 1.000,1.000,1.000,1.000 }; // 初始化解向量
            for (int num = 1; num < 2000; num++) {
                // 清空向量
                dF.clear();
                F.clear();
                // 给 F(Var)赋值
                F = {-(0.500 * cos(Var[0]) + Var[1] + Var[2] + Var[3] - x[i]
- 2.670),
                        -(Var[0] + 0.500 * sin(Var[1]) + Var[2] + Var[3] -
y[j] - 1.070),
                        -(0.500 * Var[0] + Var[1] + cos(Var[2]) + Var[3] -
x[i] - 3.740),
                        -(Var[0] + 0.500 * Var[1] + Var[2] + sin(Var[3]) -
y[j] - 0.790) };
                // 给 F'(Var)赋值
                dF = { {-0.500 * sin(Var[0]),1.000,1.000,1.000},
```

```
                        {1.000,0.500 * cos(Var[1]),1.000,1.000},
                        {0.500,1.000,-sin(Var[2]),1.000},
                        {1.00,0.500,1.000,cos(Var[3])} };
                // 列主元的 Gauss 消元法
                Gauss(dF, F, delta);
                for (k = 0; k < 4; k++)
                    Var[k] += delta[k];
                T[i][j] = Var[0];
                U[i][j] = Var[1];
                if (Max(delta) / Max(Var) < epsilon)
                    break;
            }
        }
    }
}

// 分片二次代数插值
void xyInter(vector<vector<double> >& T, vector<vector<double> >& U,
vector<vector<double> >& Z) {
    int i, j, k, r;
    double temp;
    vector<double>Tlist = { 0.000,0.200,0.400,0.600,0.800,1.000 };
    vector<double>Ulist = { 0.000,0.400,0.800,1.200,1.600,2.00 };
    vector<vector<double>>Zlist =
{ {-0.500,-0.340,0.140,0.940,2.060,3.50},
                                    {-0.420,-0.500,-0.260,0.300,1.180,2.3
80},
                                    {-0.180,-0.500,-0.500,-0.180,0.460,1.
420},
                                    {0.220,-0.340,-0.580,-0.500,-0.100,0.
620},
                                    {0.780,-0.020,-0.500,-0.660,-0.500,-0
.020},
                                    {1.500,0.460,-0.260,-0.660,-0.740,-0.
500} };

    for (i = 0; i < T.size(); i++) {
        for (j = 0; j < T[0].size(); j++) {
            // 给 t 选插值点
            if (T[i][j] <= 0.3)
                k = 1;
            else if (T[i][j] > 0.3 && T[i][j] <= 0.5)
                k = 2;
            else if (T[i][j] > 0.5 && T[i][j] <= 0.7)
```

```
                k = 3;
            else k = 4;
            // 给 u 选插值点
            if (U[i][j] <= 0.6)
                r = 1;
            else if (U[i][j] > 0.6 && U[i][j] <= 1.0)
                r = 2;
            else if (U[i][j] > 1.0 && U[i][j] <= 1.4)
                r = 3;
            else r = 4;
            // 计算插值多项式
            for (int a = k - 1; a <= k + 1; a++) {
                for (int b = r - 1; b <= r + 1; b++) {
                    temp = Zlist[a][b];
                    for (int c = k - 1; c <= k + 1; c++)
                        if (c != a)
                            temp *= (T[i][j] - Tlist[c]) / (Tlist[a] -
Tlist[c]);
                    for (int d = r - 1; d <= r + 1; d++)
                        if (d != b)
                            temp *= (U[i][j] - Ulist[d]) / (Ulist[b] -
Ulist[d]);
                    Z[i][j] += temp;
                }
            }
        }
    }
}

// 打印二维数表
void Printxyf(vector<double>& x,vector<double>& y,vector<vector<double> >&
Z) {
    for (int i = 0; i < x.size(); i++) {
        for (int j = 0; j < y.size(); j++) {
            cout << fixed << setprecision(2) << "x = " << x[i];
            cout << fixed << setprecision(2) << "  y = " << y[j];
            cout << scientific << setprecision(11) << "  f(x, y) = " << Z[i][j]
<< endl;
        }
    }
}

// 曲面拟合
```

```cpp
vector<vector<double> > SurfFit(vector<double>x, vector<double>y,
vector<vector<double> >Z) {
    int m = x.size();
    int n = y.size();

    vector<vector<double> >B;
    vector<vector<double> >G;
    vector<vector<double> >C;
    vector<vector<double> >P(m, vector<double>(n));

    ofstream MatB;
    ofstream MatG;
    ofstream MatZ;

    int i, j, k;
    double eps;

    for (k = 0; k < 7;k++) {
        // 重新给矩阵分配空间
        vector<vector<double> >t1(m, vector<double>(k + 1));
        B.swap(t1);
        vector<vector<double> >t2(n, vector<double>(k + 1));
        G.swap(t2);
        vector<vector<double> >t3(k + 1, vector<double>(k + 1));
        C.swap(t3);
        // 构建矩阵 B 和 G
        // 保存数据
        MatB.open("./MatB.csv");
        MatG.open("./MatG.csv");
        MatZ.open("./MatZ.csv");
        for (i = 0; i < m; i++) {
            for (j = 0; j <= k; j++) {
                B[i][j] = pow(x[i], j);
                MatB << B[i][j] << ",";
            }
            MatB << endl;
        }
        for (i = 0; i < n; i++) {
            for (j = 0; j <= k; j++) {
                G[i][j] = pow(y[i], j);
                MatG << G[i][j] << ",";
            }
            MatG << endl;
        }
```

```
        for (i = 0; i < m; i++) {
            for (j = 0; j < n; j++) {
                MatZ << Z[i][j] << ",";
            }
            MatZ << endl;
        }
        MatB.close();
        MatG.close();
        MatZ.close();

        //求系数矩阵 C
        GetCrs(B, G, Z, C);

        for (i = 0; i < m; i++) {
            for (j = 0; j < n; j++) {
                P[i][j] = 0;
                for (int r = 0; r <= k; r++) {
                    for (int s = 0; s <= k; s++) {
                        P[i][j] += C[r][s] * pow(x[i], r) * pow(y[j], s);
                    }
                }
            }
        }

        eps = 0;
        for (i = 0; i < m; i++)
            for (j = 0; j < n; j++)
                eps += pow((P[i][j] - Z[i][j]), 2);
        cout << "选择过程: k = " << k << "  eps = " << eps << endl;

        // debug
        /*Printxyf(x, y, P);*/

        if (eps <= 1e-7) {
            cout << "达到精度要求时, done! " << endl;
            Kk = k;
            return C;
        }
    }
    cout << "达到最大 K 值未满足精度要求" << endl;
}

int main()
{
```

```cpp
std::cout << "Hello World!\n";
int i, j;

vector<double>x(M);
vector<double>y(N);
vector<vector<double>>T(M, vector<double>(N));
vector<vector<double>>U(M, vector<double>(N));
vector<vector<double>>Z(M, vector<double>(N));


for (i = 0; i < M; i++) {
    for (j = 0; j < N; j++) {
        x[i] = 0.0800 * i;
        y[j] = 0.5000 + 0.0500 * j;
    }
}

Newton(x, y, T, U);

xyInter(T, U, Z);
Printxyf(x, y, Z);

//MatTest();

// 曲面拟合
vector<vector<double>>C;    // 系数矩阵
C = SurfFit(x, y, Z);
// debug
cout << "此时的系数矩阵 C" << endl;
MatrixPri(C);

// 观察结果
double eps = 0;
vector<double>xstar(9);
vector<double>ystar(6);
vector<vector<double> >F(9, vector<double>(6));
vector<vector<double> >P(9, vector<double>(6));
vector<vector<double>>Tstar(9, vector<double>(6));
vector<vector<double>>Ustar(9, vector<double>(6));

for (i = 0; i < 9; i++) {
    for (j = 0; j < 6; j++) {
        xstar[i] = 0.1 * i;
        ystar[j] = 0.5 + 0.2 * j;
```

```
        }
    }

    Newton(xstar, ystar, Tstar, Ustar);
    xyInter(Tstar, Ustar, F);

    for (i = 0; i < 9; i++) {
        for (j = 0; j < 6; j++) {
            P[i][j] = 0;
            for (int r = 0; r <= Kk; r++) {
                for (int s = 0; s <= Kk; s++) {
                    P[i][j] += C[r][s] * pow(xstar[i], r) * pow(ystar[j],
s);
                }
            }
        }
    }
    // 打印数表：x,y,f(x,y),p(x,y)
    cout << "打印 x*, y*, f(x*,y*), p(x*,y*)" << endl;
    for (i = 1; i < xstar.size(); i++) {
        for (j = 1; j < ystar.size(); j++) {
            cout << fixed << setprecision(2) << "x = " << xstar[i];
            cout << fixed << setprecision(2) << "  y = " << ystar[j];
            cout << scientific << setprecision(11) << "  f(x, y) = " <<
F[i][j];
            cout << scientific << setprecision(11) << "  p(x, y) = " << P[i][j]
<< endl;

        }
    }
    for (i = 0; i < 9; i++)
        for (j = 0; j < 6; j++)
            eps += pow((P[i][j] - F[i][j]), 2);
    cout << "eps = " << eps << endl;
}
```

四、运行结果

1、打印输出二维数表$\{x_i, y_j, f(x_i, y_j)\}$（部分截图）

```
Hello World!
x = 0.00  y = 0.50  f(x, y) = 4.46504018480e-01
x = 0.00  y = 0.55  f(x, y) = 3.24683262927e-01
x = 0.00  y = 0.60  f(x, y) = 2.10159686683e-01
x = 0.00  y = 0.65  f(x, y) = 1.03043608316e-01
x = 0.00  y = 0.70  f(x, y) = 3.40189556266e-03
x = 0.00  y = 0.75  f(x, y) = -8.87358136380e-02
x = 0.00  y = 0.80  f(x, y) = -1.73371632750e-01
x = 0.00  y = 0.85  f(x, y) = -2.50534611467e-01
x = 0.00  y = 0.90  f(x, y) = -3.20276506388e-01
x = 0.00  y = 0.95  f(x, y) = -3.82668066110e-01
x = 0.00  y = 1.00  f(x, y) = -4.37795766738e-01
x = 0.00  y = 1.05  f(x, y) = -4.85758941444e-01
x = 0.00  y = 1.10  f(x, y) = -5.26667254884e-01
x = 0.00  y = 1.15  f(x, y) = -5.60638479797e-01
x = 0.00  y = 1.20  f(x, y) = -5.87796538768e-01
x = 0.00  y = 1.25  f(x, y) = -6.08269779090e-01
x = 0.00  y = 1.30  f(x, y) = -6.22189452876e-01
x = 0.00  y = 1.35  f(x, y) = -6.29688378186e-01
x = 0.00  y = 1.40  f(x, y) = -6.30899760003e-01
x = 0.00  y = 1.45  f(x, y) = -6.25956152545e-01
x = 0.00  y = 1.50  f(x, y) = -6.14988546609e-01
```

2、选择过程中的k和σ，达到精度要求时的k和σ以及系数矩阵C

```
x = 0.80  y = 1.50  f(x, y) = -1.41949659709e-01
选择过程: k = 0  eps = 1.44288077184e+02
选择过程: k = 1  eps = 3.22090897363e+00
选择过程: k = 2  eps = 4.65996003325e-03
选择过程: k = 3  eps = 1.72117537930e-04
选择过程: k = 4  eps = 3.30953430098e-06
选择过程: k = 5  eps = 2.54198827040e-08
达到精度要求时, done!
此时的系数矩阵C
2.0212e+00 -3.6684e+00 7.0925e-01 8.4861e-01 -4.1590e-01 6.7432e-02
3.1919e+00 -7.4110e-01 -2.6971e+00 1.6312e+00 -4.8473e-01 6.0615e-02
2.5695e-01 1.5796e+00 -4.6271e-01 -8.2070e-02 1.0246e-01 -2.1087e-02
-2.6930e-01 -7.3006e-01 1.0758e+00 -8.0662e-01 3.0268e-01 -4.5937e-02
2.1790e-01 -1.8088e-01 -6.6930e-02 2.3757e-01 -1.3841e-01 2.5932e-02
-5.5872e-02 1.4301e-01 -1.3583e-01 4.0229e-02 4.0387e-03 -2.7224e-03
```

此时$k = 5$，精度水平$\sigma = 2.54198827040e - 08$

5、打印数表$\{x_i^*, y_j^*, f(x_i^*, y_j^*), p(x_i^*, y_j^*)\}$

Microsoft Visual Studio 调试控制台

```
打印x*, y*, f(x*,y*), p(x*,y*)
x = 0.10  y = 0.70  f(x, y) = 1.94720407918e-01  p(x, y) = 1.94730357381e-01
x = 0.10  y = 0.90  f(x, y) = -1.83037079189e-01  p(x, y) = -1.83041839064e-01
x = 0.10  y = 1.10  f(x, y) = -4.45497646915e-01  p(x, y) = -4.45500046505e-01
x = 0.10  y = 1.30  f(x, y) = -5.97566707641e-01  p(x, y) = -5.97558865544e-01
x = 0.10  y = 1.50  f(x, y) = -6.46459593901e-01  p(x, y) = -6.46446127858e-01
x = 0.20  y = 0.70  f(x, y) = 4.05979189288e-01  p(x, y) = 4.05989537413e-01
x = 0.20  y = 0.90  f(x, y) = -2.25159583746e-02  p(x, y) = -2.25211118736e-02
x = 0.20  y = 1.10  f(x, y) = -3.38220816040e-01  p(x, y) = -3.38224030125e-01
x = 0.20  y = 1.30  f(x, y) = -5.44437831522e-01  p(x, y) = -5.44430458291e-01
x = 0.20  y = 1.50  f(x, y) = -6.47361338568e-01  p(x, y) = -6.47348031608e-01
x = 0.30  y = 0.70  f(x, y) = 6.34777195151e-01  p(x, y) = 6.34787445323e-01
x = 0.30  y = 0.90  f(x, y) = 1.58801168839e-01  p(x, y) = 1.58796311479e-01
x = 0.30  y = 1.10  f(x, y) = -2.07365694171e-01  p(x, y) = -2.07368595372e-01
x = 0.30  y = 1.30  f(x, y) = -4.65357906898e-01  p(x, y) = -4.65349927645e-01
x = 0.30  y = 1.50  f(x, y) = -6.20270953075e-01  p(x, y) = -6.20257169792e-01
x = 0.40  y = 0.70  f(x, y) = 8.78960023174e-01  p(x, y) = 8.78969847132e-01
x = 0.40  y = 0.90  f(x, y) = 3.58650625882e-01  p(x, y) = 3.58646082171e-01
x = 0.40  y = 1.10  f(x, y) = -5.52528211681e-02  p(x, y) = -5.52554684364e-02
x = 0.40  y = 1.30  f(x, y) = -3.62679511503e-01  p(x, y) = -3.62671061583e-01
x = 0.40  y = 1.50  f(x, y) = -5.67564743655e-01  p(x, y) = -5.67550637214e-01
x = 0.50  y = 0.70  f(x, y) = 1.13661091016e+00  p(x, y) = 1.13662031552e+00
x = 0.50  y = 0.90  f(x, y) = 5.74980340948e-01  p(x, y) = 5.74975924935e-01
x = 0.50  y = 1.10  f(x, y) = 1.15992376792e-01  p(x, y) = 1.15989257561e-01
x = 0.50  y = 1.30  f(x, y) = -2.38568304012e-01  p(x, y) = -2.38560408046e-01
x = 0.50  y = 1.50  f(x, y) = -4.91434393656e-01  p(x, y) = -4.91421002225e-01
x = 0.60  y = 0.70  f(x, y) = 1.40604179891e+00  p(x, y) = 1.40605061616e+00
x = 0.60  y = 0.90  f(x, y) = 8.05941494063e-01  p(x, y) = 8.05937457961e-01
x = 0.60  y = 1.10  f(x, y) = 3.04429221045e-01  p(x, y) = 3.04425711094e-01
x = 0.60  y = 1.30  f(x, y) = -9.50161300996e-02  p(x, y) = -9.50089187984e-02
x = 0.60  y = 1.50  f(x, y) = -3.93902307746e-01  p(x, y) = -3.93890023882e-01
x = 0.70  y = 0.70  f(x, y) = 1.68578351531e+00  p(x, y) = 1.68579109413e+00
x = 0.70  y = 0.90  f(x, y) = 1.04988115306e+00  p(x, y) = 1.04987801282e+00
x = 0.70  y = 1.10  f(x, y) = 5.08293783940e-01  p(x, y) = 5.08290830758e-01
x = 0.70  y = 1.30  f(x, y) = 6.61487967065e-02  p(x, y) = 6.61564067222e-02
x = 0.70  y = 1.50  f(x, y) = -2.76834341778e-01  p(x, y) = -2.76822369107e-01
x = 0.80  y = 0.70  f(x, y) = 1.97457455665e+00  p(x, y) = 1.97458106022e+00
x = 0.80  y = 0.90  f(x, y) = 1.30533496765e+00  p(x, y) = 1.30533245435e+00
x = 0.80  y = 1.10  f(x, y) = 7.25992371111e-01  p(x, y) = 7.25988953795e-01
x = 0.80  y = 1.30  f(x, y) = 2.43254184181e-01  p(x, y) = 2.43260877548e-01
x = 0.80  y = 1.50  f(x, y) = -1.41949659709e-01  p(x, y) = -1.41939329848e-01
eps = 4.88783131771e-09
```

此时精度水平$\sigma = 4.88783131771e-09$

五、结果总结

在使用最小二乘法进行曲面拟合时，出现了以下离奇的结果：随着 k 的增大，精度水平σ本应该越来越小，结果也确实如此，在$k \leq 4$时，程序计算得到的σ与 matlab 计算结果大致相同；而当$k \geq 5$时，精度误差突然发散且越来越大，最终无法得到正确结果。

```
Microsoft Visual Studio 调试控制
x = 0.80  y = 1.35  f(x, y) = 1.37862222525e-01
x = 0.80  y = 1.40  f(x, y) = 3.85567703264e-02
x = 0.80  y = 1.45  f(x, y) = -5.46985959345e-02
x = 0.80  y = 1.50  f(x, y) = -1.41949659709e-01
选择过程: k = 0  eps = 1.44288077184e+02
选择过程: k = 1  eps = 3.22090897363e+00
选择过程: k = 2  eps = 4.65996003325e-03
选择过程: k = 3  eps = 1.72117537930e-04
选择过程: k = 4  eps = 3.34990027747e-06
选择过程: k = 5  eps = 1.80263250032e+01
选择过程: k = 6  eps = 1.77221294946e+05
达到最大K值未满足精度要求
```

在重新检查程序后，我认为程序的算法设计是没有问题的，原因可能是因为随着k的增大，程序中所计算的数也越来越小，在运算中可能造成误差，主要有浮点数引起。我推测出错的地方可能在以下几个位置：

（1） 列主元的 Gauss 消元法，其中涉及到消元、浮点数相除等运算，可能造成舍入误差

（2） 分片二次插值，二位数表的有效位数较少，可能造成误差

（2）矩阵的基本运算；

通过 VS 调试和将中间变量数据导入 MATLAB 等手段，我将程序求得的二位数表导入 MATLAB 同样得到了正确的收敛结果，因此错误只能存在于矩阵的基本运算中。在曲面拟合时，需要计算系数矩阵C，涉及到矩阵的转置、求逆、乘积运算。其中转置和乘积运算不会出现太大的计算误差，问题在矩阵求逆中。

常用的简单矩阵求逆有几种不同的做法：

（1） 根据$A^{-1} = \frac{A^*}{|A|}$，求出矩阵A的行列式和伴随矩阵

（2） 根据$AA^{-1} = I$，解一组线性方程组

（3） 根据$A = LU$，LU 分解求逆矩阵

起初我根据（1）编写了矩阵求逆的函数，使用递归法求矩阵的行列式，在测试和k较小时没有出现问题，但当$k = 5$时，矩阵B和G中的数彼此相差太大，因此带来了较大的截断误差。因此我重新编写了列主元的 Gauss 消元法的求逆函数，得到了正确的结果。