****

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

## 题目要求

关于的方程组如下

关于的二维数表已给出。试求：

1、使用数值方法求出在区域上的近似表达式

要求以最小的值达到以下精度

其中,。

2、计算,)的值，以观察逼近的效果，其中。

题中要求在作二元插值时使用分片二次代数插值，并打印出的二位数表。

## 二、算法流程

题中给出了的二维数表，所要求的是的二维数表，其中。而和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

对于上述方程组来说，向量和向量已知，因此可令向量,设定精度水平和最大迭代次数2000，迭代步骤：

设定初值,

求解线性方程组

迭代终止条件为.

其中：

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。

1. 分片二次代数插值

二元函数的代数插值与一元函数类似，设

由已知数表可知，，，。

首先选定插值节点：

对于来说，当满足

选择为插值节点，如果或者，则取或；如果或者，则取或。

然后确定插值多项式

其中

带入即得到，可以直接打印二维数表

1. 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

给定精度水平和最大=8，迭代步骤如下：

计算矩阵和矩阵

计算系数矩阵

计算拟合值

拟合精度使用误差平方和

当时停止迭代，否则，进入下一轮计算。

1. 使用进行验证

使用牛顿迭代法和分片二次代数插值，得到一组新的，即可得到新的二维数表。

使用上述曲面拟合函数及之前求得的系数矩阵和，得到一组新的拟合值，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,-0.260,0.300,1.180,2.380},

                                    {-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

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()

{

    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++)

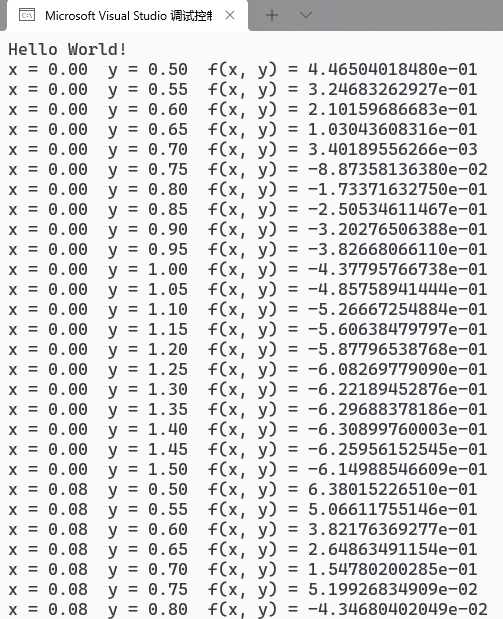
            eps += pow((P[i][j] - F[i][j]), 2);

    cout << "eps = " << eps << endl;

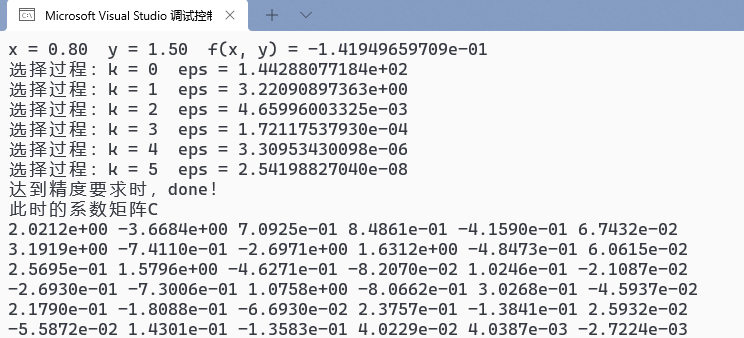
}

## 四、运行结果

1、打印输出二维数表（部分截图）

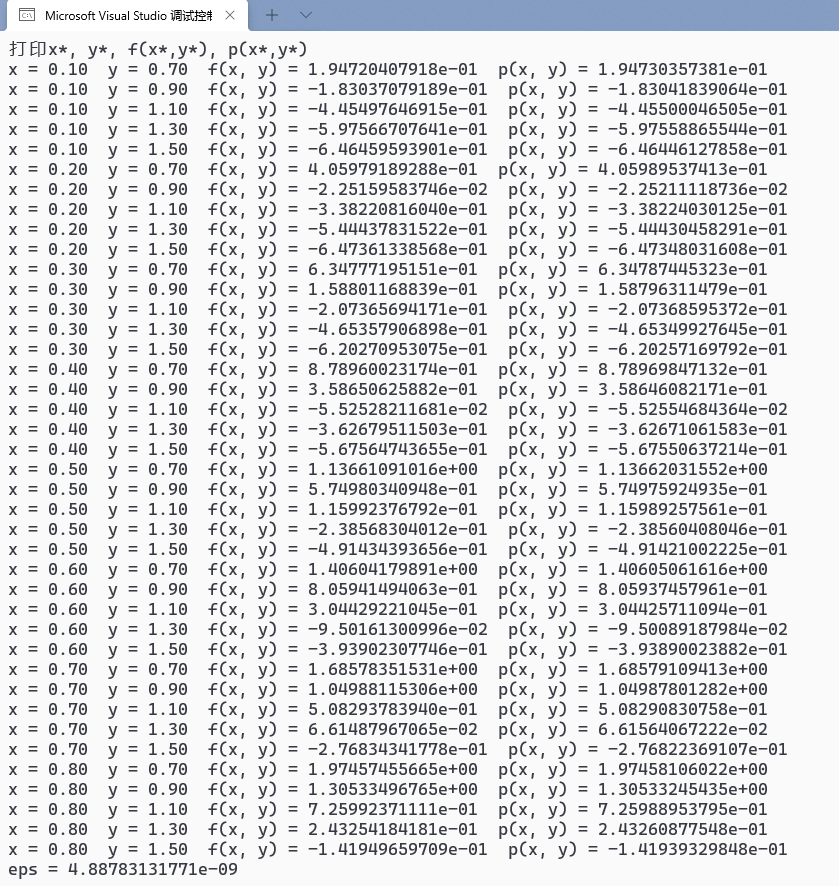


2、选择过程中的和，达到精度要求时的和以及系数矩阵



此时，精度水平

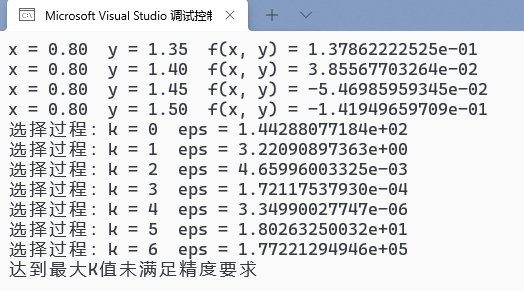
1. 打印数表



此时精度水平

## 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，精度水平本应该越来越小，结果也确实如此，在时，程序计算得到的与matlab计算结果大致相同；而当时，精度误差突然发散且越来越大，最终无法得到正确结果。



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，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，求出矩阵的行列式和伴随矩阵
2. 根据，解一组线性方程组
3. 根据，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，在测试和较小时没有出现问题，但当时，矩阵和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。