统计推断在数模转换系统中的应用

组号：49　组长姓名：王思捷 学号：5130309146，组员姓名：郝浚智 学号：5130309136

摘要：

此文是本组成员在了解检测模块工作原理[1]的基础上之，对传感部件的输出电压 X 和经过模数转换器（ADC）及微控制器处理获取的信号 Y 之间的函数关系进行的推断分析。本组成员在了解测量过程以及对所给模块的物理量测量的大量数据进行绘图观察与分析后，根据样本特性以及传感性校准理论，建立恰当数学模型，通过遗传算法进行传感特性校准（定标工序）方案，即通过测量少数特定数据点推断 X 与 Y 之间的近似关系。

关键词：

拟合，插值，搜索算法，启发式搜索，单点测定

1 引言

在现代工业生产中，一个产品往往被分割为许多部件分别生产，这样的生产模式虽然提高了生产效率，但同时带来了整合困难的问题，因此对模块的特性检测至关重要。一种常见的检测模块是通过传感器部件（包含传感原件及必要的放大电路、调理电路等）将被检测物理量转化为输出电压信号，此电压经过模数转换器（ADC）成为数字编码，并能被微处理器程序所读取和处理，最终获得 Y 的读数。在测定相关特性曲线时，需要测量大量数据，但是在工业生产中，对每个样品的测定都有一定的测定成本称为单点测定成本，并且在确定特性曲线后，对曲线错误的错误估计会产生单点定标成本，所以需要寻找一种测定方法，可以在测量少数特定数据点的情况下保证测量精度，以达到保证质量检测和降低检测成本的目的。直观上讲就是尽量减少测量点的同时维护检测精度。

本组成员在此次统计推断课程中通过分析数据特性曲线，建立了恰当的数学模型，运用软件技术得出了给定样本的定标方案。以下是本组的课程设计报告。

2 检测模块函数关系

2.1 问题分析

物理元件具有独一无二的元件特性曲线，是一条客观存在的连续曲线，实验者可以通过大量测量数据点，在误差允许的情况下可以确定这条曲线的近似曲线。通过对数据的分析和观察，我们发现不同组的数据都有相似之处，因此推测该元件的特性曲线上有一些“特征点”，即有部分数据点是确定曲线的关键点。由此把问题转化为：寻找一种取点方案，使得选取的点尽量少，而得出的曲线误差尽量小，即可认为选取的点为“特征点”，从而的到曲线的最优解。

2.2 问题的解决

我们随机选取了十组数据通过 Origin 作图观察后发现，结果适用于多项式拟合或线性插值。

对于样本的数据点选取有多种方案（2^{51}种），经过分析和研究，我们决定初步采取遗传算法来进行取点的方案的优化，并且将采用遗传算法所得的结果和用一定量的穷举法所得的结果进行比较。

下文将对解决问题的具体思路及过程进行说明。

2.2.1 多项式拟合法

(1) 所用软件：

Origin 9.0， CodeBlocks 10.5

(2) 多项式拟合的原理[2]：

曲线拟合又称作函数逼近，是求近似函数的一类数值方法. 它不要求近似函数在每个节点处与函数值相同，只要求其尽可能的反映给定数据点的基本趋势以及某种意义上的无限“逼近”. 在需要对一组数据进行处理、筛选时，我们往往会选择合理的数值方法，而曲线拟合在实际应用中也倍受青睐. 采用曲线拟合处理数据时，一般会考虑到误差的影响，于是我们往往基于残差的平方和最小的准则选取拟合曲线的方法，这便是经常所说的曲线拟合的最小二乘法. 通过对一些文献的分析和整理，不仅有数据处理（数据采集），还有模型建立，可以了解到曲线拟合的最小二乘法的应用领域较为广泛。

最小二乘法多项式曲线拟合，根据给定的 m 个点，并不要求这条曲线精确地经过这些点，而是曲线 y=f(x)的近似曲线 y= φ(x)。给定数据点 pi(x_i,y_i)，其中 i=1,2,…,m。求近似曲线 y= φ(x)。并且使得近似曲线与 y=f(x)的偏差最小。近似曲线在点 pi 处的偏差 δ_i= φ(x_i)-y，i=1,2,...,m。推导过程见下文。

设拟合多项式为：

$$R^2 \equiv \sum_{i=1}^{n} \left[y_i - \left(a_0 + a_1 x_i + \ldots + a_k x_i^k \right) \right]^2 . \tag{2-1}$$

各点到这条曲线的距离之和，即偏差平方和如下：

$$R^2 \equiv \sum_{i=1}^{n} \left[y_i - \left(a_0 + a_1 x_i + \ldots + a_k x_i^k \right) \right]^2 . \tag{2-2}$$

为了求得符合条件的 a 值，对等式右边求 a_i 偏导数，因而我们得到了：

$$-2 \sum_{i=1}^{n} \left[y - \left(a_0 + a_1 x + \ldots + a_k x^k \right) \right] x = 0$$

$$-2 \sum_{i=1}^{n} \left[y - \left(a_0 + a_1 x + \ldots + a_k x^k \right) \right] = 0$$

.....

$$-2 \sum_{i=1}^{n} \left[y - \left(a_0 + a_1 x + \ldots + a_k x^k \right) \right] x^k = 0. \tag{2-3}$$

将等式左边进行一下化简，然后应该可以得到下面的等式：

$$a_0 \sum_{i=1}^{n} x_i^k + a_1 \sum_{i=1}^{n} x_i^{k+1} + \ldots + a_k \sum_{i=1}^{n} x_i^{2k} \tag{2-4}$$

把这些等式表示成矩阵的形式，就可以得到下面的矩阵：

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^k \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_i^k & \sum_{i=1}^{n} x_i^{k+1} & \cdots & \sum_{i=1}^{n} x_i^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \vdots \\ \sum_{i=1}^{n} x_i^k y_i \end{bmatrix}. \tag{2-5}$$

将这个范德蒙得矩阵化简后可得到：

$$\begin{bmatrix} 1 & x_1 & \cdots & x_1^k \\ 1 & x_2 & \cdots & x_2^k \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^k \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}. \tag{2-6}$$

也就是说 X*A=Y，那么 A = (X'*X)-1*X'*Y，便得到了系数矩阵 A，同时，我们也就得到了拟合曲线。

(3) 拟合算法的程序实现：

这里的程序函数实现分为以下几步：传入主函数所给的用于拟合的数据点，将初始 X[I]的值的各幂次方存储在一个二维数组里面，计算正规方程组的系数矩阵，列主元 LU 分解，解系数方程，将所得四个系数传出。具体代码参见附录 2。

2.2.2 线性插值法

(1) 所用软件：

Origin 9.0， CodeBlocks 10.5， MatLab 2012b

(2) 线性插值的原理[3]：

线性插值是数学、计算机图形学等领域广泛使用的一种简单插值方法假设我们已知坐标(x0,y0)与(x1,y1),要得到[x0,x1]区间内某一位置 x 在直线上的 y 值。

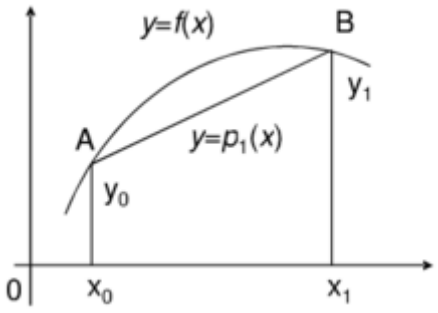

图 2-1 线性插值原理图

根据图中所示，假设 AB 上有一点（x,y），可作出两个相似三角形，我们得到：

（y-y0)/(x-x0)=(y1-y0)/(x1-x0) (2-7)

假设方程两边的值为 α，那么这个值就是插值系数—从 *x*0 到 x 的距离与从 *x*0 到 *x*1 距离的比值。由于 x 值已知，所以可以从公式得到 α 的值：

α=(x-x0)/(x1-x0) (2-8)

这样，在代数上就可以表示成为：

y = (1- α)*y*0 + α*y*1 (2-9)

或者：

y = *y*0 + α(*y*1 - *y*0) (2-10)

这样通过 α 就可以直接得到 y。实际上，即使 x 不在 *x*0 到 *x*1 之间并且 α 也不是介于 0 到 1 之间，这个公式也是成立的。在这种情况下，这种方法叫做线性外插。

(3) 线性插值法的程序实现：

此处程序按照传入的取点方案，将所取各点作为各线段的两端点，分别计算出线段的函数表达式。其中，如果取点方案的两端点不为所有 51 个点的两端点，则认为超出取点方案范围的函数表达式为已得两端点处的线段的延伸。再按照从 5.0 到 10.0 的顺序代入函数表达式求出插值法所得各纵坐标。具体代码参见附录 3。

2.2.2 搜索算法（遗传算法）

(1) 所用软件：

CodeBlocks 10.5

(2) 遗传算法基本原理[4]：

遗传算法是计算机科学人工智能领域中用于解决最优化的一种搜索启发式算法，是进化算法的 一种。这种启发式通常用来生成有用的解决方案来优化和搜索问题。

遗传算法通常实现方式为一种计算机模拟。对于一个最优化问题，一定数量的候选解（称为个 体）的抽象表示（称为染色体）的种群向更好的解进化。传统上，解用二进制表示（即 0 和 1 的 串），但也可以用其他表示方法。进化从完全随机个体的种群开始，之后一代一代发生。在每 一代中，整个种群的适应度被评价，从当前种群中随机地选择多个个体（基于它们的适应 度），通过自然选择产生新的生命种群，该种群在算法的下一次迭代中成为当前种群。

基本算法：

1. 选择初始生命种群
2. 循环：
    - 评价种群中的个体适应度
    - 以比例原则（分数高的挑中机率也较高）选择产生下一个种群（轮盘法 en:roulette wheel selection、竞争法 en:tournament selection 及等级轮盘法 Rank Based Wheel Selection）。不仅仅挑分数最高的原因是这么做可能收敛 到局部的最佳点，而非整体的。
    - 改变该种群（交叉和变异）
3. 直到停止循环的条件满足

(3) 遗传算法在本课题的应用思路：

1. 个体编码：使用二进制串来表示取点方法，按顺序，0 为不选，1 为选取。如，00110110 代表取 3，4，6，7 四个点。因此，这里使用一个长为 51 的二进制串作为遗传 算法中的个体基因（染色体）。个体的表现型由编码解码后得到，如基因型为 ［11111000....00］的个体，表现型为［1，2，3，4，5］，即取点方案为取出第 1， 2，3，4，5 的总共 5 个点。

2. 初始群体：这里设置群体规模的大小为 500，即初始群体中，含有 500 个个体，也即为 500 个比特串。其中，初始个体随机产生，如以下 8 个个体：

[0100111100100101100001000111010110010001010010100001]
[0001011000011000001101110111000101000001011011100000]
[1110110100010101000111011011011111011000000001110111]
[1111010111110111110110100000011101111111100100111110]
[1001001111100010111100100001110100011000001110110001]
[0011111101101010000000000100111001110001101110110000]
[1100110011001111000100100011100010110110110110011101]
[1101111100110001111101110100011000001001000000001011]

3. 适应度计算：将个体解码后，带入 2.2.1 和 2.2.2 中的成本求取函数，得到一个整数，并且得到的 数越小，表示该个体越优，从而遗传机会越大。然而，为了方便依概率进行选

择运算， 需要适应度的数值越大越优。于是，对每个个体计算出成本后，进行重新排序，得到适 应度。比如，计算后 8 个个体的成本值为［1］1，［2］3，［3］5，［4］2，［5］8，［6］7，［7］6，［8］4 ，将得到的 8 个成本值递增排序，然后将 8 个个体按照各自的 成本值递减排序，之后依次赋给对应序号的成本值作为适应度。他们对应的适应度为 ［1］8，［2］6，［3］4，［4］7，［5］1，［6］2，［7］3，［8］5

4. 选择运算：将每个个体的适应度相加，得到适应度之和 S，并且每个个体适应度表示成 数轴上的区间，依次排序，然后产生一个 0 到 S 的随机数，随机数将落在某一个体对应的 区间内，此时计该个体被选择一次。总共产生 500 次随机数，选择出 500 个待选个体。

5. 交叉运算：为便于说明，假定群组有 8 个个体，随机产生 4 组配对组合，如 1-2，3-7，4- 8，5-6。然后每个组分别产生随机的交叉点位置，每个配对组中，前者取交叉点位置之 前的基因，后者取交叉点位置之后的基因（这里以一个交叉点举例，本例实际交叉点个 数为 1 到 51 之间的随机数），然后连接产生一个新个体。每一对产生两个子代个体，经过一轮交叉运算后产生 8 个新个体，原来的 8 个个体作为祖辈被抛弃。注：这里配对概率 为 100%。

6. 变异运算：设置每个基因的编译概率为百分之 0.5，即对于一次变异，长度为 51 的二进制串中，每个点都有百分之 0.5 的概率发生变异，即取反，如 0 变为 1。

7. 重复 3-6 步骤，进行一代又一代的选择，重复一定次数后停止（暂定 1000 次），然后从 最终的群体中，取出适应度最强的个体，将该个体解码后，即得到近似最优取点方案。

3 结论

3.1 程序运行结果：

使用附录 2 以及附录 3 中的代码进行计算。在使用多项式拟合的方法中，程序运行了 64 代后达到稳定，说明已经找到了最优解（可能为局部最优），结果为 136.671，取点方案为第 3, 20, 29, 43, 46, 49 个取值点。而在使用线性插值的方法中，程序运行了 57 代后达到稳定，说明已经找到了最优解（可能为局部最优），结果为 86.0597，取点方案为第 4, 18, 26, 34, 43, 50 个取值点。可以看出，线性插值法的结果明显优于多项式拟合法。

根据程序运行结果，最终确定采用线性插值法，取点方案为第 4, 18, 26, 34, 43, 50 个取值点，即 x=5.3， x=6.7， x=7.5， x=8.3， x=9.2， x=9.9 的 6 个点。对 469 组数据使用该取点方案进行线性插值取平均后，得到的表达式为：

$$y=\begin{cases}16.7651x-78.7608 & (5.0\le x\le 6.7)\\ 13.6813x-58.0992 & (6.7<x\le 7.5)\\ 13.6534x-57.8901 & (7.5<x\le 8.3)\\ 20.5128x-144.868 & (8.3<x\le 9.1)\\ 25.4441x-160.187 & (9.1<x\le 10)\end{cases}$$

图像为：

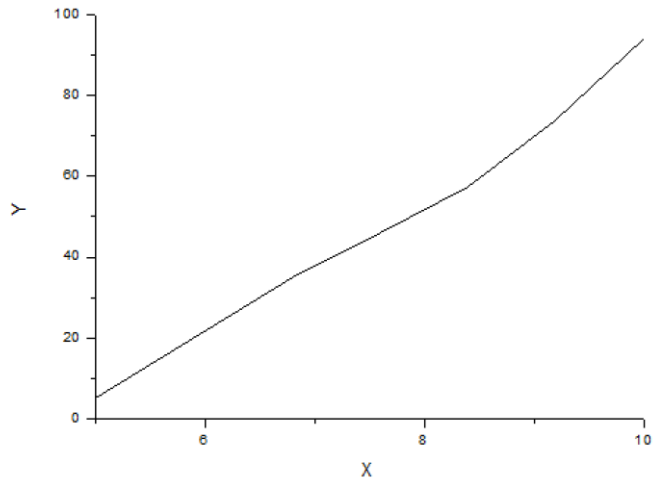

图 3-1 输出电压 X 与读取值 Y 的线性插值结果图像

将该图像与随机选取的 10 组数据的散点图进行比较，发现十分接近，该结果可靠性得到验证。

3.2 课题结论

该模块的传感特性校准方案为:

取点方案{x=5.3, x=6.7, x=7.5, x=8.3, x=9.2, x=9.9},使用线性插值(包含线性外推)。

校准方案总体成本为 86.0597

3.3 问题探讨与反思

本次课题研究中，我们只使用了两种拟合（插值）方法，如果比较更多种拟合方法（或插值），结果将会更加准确与可靠。在启发式搜索算法方面，遗传算法的各项指标仍有调整空间，如变异率、交叉率等。另外，遗传算法本身存在一定的局限性，容易得到局部最优解，而非全局最优解。

4 参考文献

[1] 上海交大电子工程系. 统计推断在数模转换系统中的应用课程讲义 [EB/OL].ftp://202.120.39.248.

[2] Wang Guorong; Wei Yimin, Qiao SanZheng. Equation Solving Generalized Inverses. Generalized Inverses:Theory and Computations. Beijing: Science Press， 2004: 第 6 页. ISBN 7-03-012437-5.

[3] E. Meijering; 插值年表：从古代天文学到现代信号与图像处理. Proceedings of the IEEE 9 (3), 2002, 319 – 342.

[4] Mitchell, Melanie. 遗传算法概论. MIT Press, Cambridge, MA，1996.

附页

1 适用于 MatLab 的结果检验函数

```
function y1 = mycurvefitting( x_premea,y0_premea )
x=[5.0:0.1:10.0];
y1=interp1(x_premea,y0_premea,x,'linear','extrap');
end
```

2 使用多项式拟合的 C++代码

以下代码在 CodeBlocks 10.05 中测试通过，运行结果保存将会打印在屏幕上并且保存在 output.txt 文件中：

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <fstream>

using namespace std;

void fittedCurve(int, double*, double*, double&, double&, double&, double&);
void f1(int, double *,double *);
void f2(int, double *,double *,double [ ]);
void DirectLU(double a[4][5],double [ ]);
void swap(double &,double &);

void f1(int n, double *a,double *x)
{
    int i,j,k;
    double temp;
    for(i=0;i<4;i++)
        for(j=0;j<n;j++)
        {
            temp=1;
            for(k=0;k<i;k++) temp*=x[j];
            *(a+i*n+j)=temp;
        }
}

void f2(int n, double *a,double *b,double y[])
{
    int i,j,k;
    double temp2;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
```

```
            {
                temp2=0;
                for(k=0;k<n;k++) temp2+=*(a+i*n+k)*(*(a+j*n+k));
                *(b+i*5+j)=temp2;
            }
            temp2=0;
            for(k=0;k<n;k++)
            {
                temp2+=y[k]*(*(a+i*n+k));
                *(b+i*5+4)=temp2;
            }
        }
}

void swap(double &a,double &b)
{
        a=a+b;
        b=a-b;
        a=a-b;
}

void DirectLU(double a[4][5],double x[])
{
        int i,r,k,j;
        double s[4],t[4];
        double max;
        for(r=0;r<4;r++)
        {
            max=0;
            j=r;
            for(i=r;i<4;i++)
            {
                s[i]=a[i][r];
                for(k=0;k<r;k++) s[i]-=a[i][k]*a[k][r];
                s[i]=s[i]>0?s[i]:-s[i];
                if(s[i]>max)
                {
                    j=i;
                    max=s[i];
                }
            }
            if(j!=r)
            {
                for(i=0;i<5;i++) swap(a[r][i],a[j][i]);
```

```
            }
            for(i=r;i<5;i++)
                for(k=0;k<r;k++)
                {
                    a[r][i]-=a[r][k]*a[k][i];
                }
            for(i=r+1;i<4;i++)
            {
                for(k=0;k<r;k++) a[i][r]-=a[i][k]*a[k][r];
                a[i][r]/=a[r][r];
            }
        }
        for(i=0;i<4;i++) t[i]=a[i][4];
        for(i=3;i>=0;i--)
        {
            for(r=3;r>i;r--) t[i]-=a[i][r]*x[r];
            x[i]=t[i]/a[i][i];
        }
}

void fittedCurve(int n, double* x, double* y, double& a0, double& a1, double& a2, double& a3)
{
        int i,j;

        double a[4][n];
        double b[4][5];
        double c[4];

        f1(n,a[0],x);
        f2(n,a[0],b[0],y);
        DirectLU(b,c);

        a0 = c[0];
        a1 = c[1];
        a2 = c[2];
        a3 = c[3];
}

double getFitness(int* p, int L)
{
        ifstream f("20141010dataform.csv");
        double costs = 0.0;
        double a0, a1, a2, a3;
        double X[51];
```

```
        double Y[51];
        char ch1[7];

        for(int n = 0; n < 469; n++)
        {
            for(int i = 0; i < 50; i++) { f.getline(ch1, 7, ','); X[i] = atof(ch1);}
            f.getline(ch1, 7); X[50] = atof(ch1);
            //cout << X[50] << ' ';
            for(int i = 0; i < 50; i++) { f.getline(ch1, 7, ','); Y[i] = atof(ch1);}
            f.getline(ch1, 7); Y[50] = atof(ch1);
            //cout << Y[50] << ' ';

            double *x = new double[L];
            double *y = new double[L];
            for(int l = 0; l < L; l++)
            {
                x[l] = X[(p[l]) - 1];
                y[l] = Y[(p[l]) - 1];
            }
            fittedCurve(L, x, y, a0, a1, a2, a3);

            for(int i = 0; i < 51; i++)
            {
                double deviation = abs((a0 + a1*X[i] + a2*pow(X[i],2) + a3*pow(X[i],3)) - Y[i]);
                if(deviation > 0.5 && deviation <= 1) costs += 0.5;
                else if(deviation > 1 && deviation <= 2) costs += 1.5;
                else if(deviation > 2 && deviation <= 3) costs += 6;
                else if(deviation > 3 && deviation <= 5) costs += 12;
                else if(deviation > 5) costs += 25;
            }
            costs += L*12;
            delete [] x;

            delete [] y;
        }

        return costs/469;
}


class indivisual {
        char genotype[51]; //  基因型
        int phenotype[51]; //  表现型
        int pheLength;
```

```
    double fitness; //  适应度
public:
    indivisual() {
        for (int i = 0; i < 51; ++i) {
            int tmp = rand() % 2;
            if (tmp == 0)
                genotype[i] = '0';
            else if (tmp == 1)
                genotype[i] = '1';
        }
        deCode();
        fitness = 0;
    }
    indivisual(const char * s) {
        genCopy(s);
        deCode();
        fitness = 0;
    }
    indivisual(const indivisual * x) {
        char code[51];
        x->getGenCode(code);
        genCopy(code);
        deCode();
        fitness = x->getFitness();
    }
    // 配对产生新个体的构造函数
    indivisual(const indivisual * x1, const indivisual * x2, char * child2) {
        // 随机产生交叉点个数（0-50）
        int crossNum = rand() % 51;
        int cross[50];
        for (int i = 0; i < crossNum; ++i) {
            int pre;
            if ( i > 0 )
                pre = cross[i-1];
            else
                pre = 0;
            cross[i] = rand() % (52 - pre);
        }
        char code1[51];
        char code2[51];
        x1->getGenCode(code1);
        x2->getGenCode(code2);

        char code[51];
```

```
        int index = 0;
        int order = 0;
        for (int i = 0; i < crossNum; ++i) {

                for (int j = index; j < index + cross[i]; ++j) {
                        if ( order % 2 == 0 ) {
                                code[j] = code1[j];
                                child2[j] = code2[j];
                        }
                        else {
                                code[j] = code2[j];
                                child2[j] = code1[j];
                        }
                        ++order;
                }
                index = cross[i];
        }
        for (int i = index; i < 52; ++i) {
                if ( order % 2 == 0 ) {
                        code[i] = code1[i];
                        child2[i] = code2[i];
                }
                else {
                        code[i] = code2[i];
                        child2[i] = code1[i];
                }
                ++order;
        }
        genCopy(code);
        deCode();
        fitness = 0;
}
void getGenCode(char * s) const {
        for (int i = 0; i < 51; i++)
                s[i] = genotype[i];
}
void getPhe(int * s) {
        for (int i = 0; i < pheLength; i++){
                s[i] = phenotype[i];
        }
}
int getPheLength() {
        return pheLength;
```

```
	}
	void updateFitness(double f) {
		fitness = f;
	}
	double getFitness() const {
		return fitness;
	}
	void mutation() {
		for (int i = 0; i < 51; ++i) {
			int index = rand() % 200;
			if (index == 0) {
				if (genotype[index] == '1')
					genotype[index] = '0';
				else if (genotype[index] == '0')
					genotype[index] = '1';
			}
		}

		deCode();
	}
private:
	void genCopy(const char * s) {
		for (int i = 0; i < 51; i++) {
			genotype[i] = s[i];
		}
	}
	void deCode() {
		int j = 0;
		for (int i = 0; i < 51; i++) {
			if (genotype[i] == '1') {
				phenotype[j++] = i+1;
			}
		}
		pheLength = j;
	}
};

void quickSort(double a[], int low, int high);
int divide(double a[], int low, int high);
int main(int argc, const char * argv[]) {
	srand(time(NULL));

	const int SIZE = 500;
	ofstream f("ouput.txt");
```

```
// 产生初始群体
indivisual * inds[SIZE];
indivisual * tmpInds[SIZE];
for (int i = 0; i < SIZE; ++i)
	inds[i] = new indivisual();

// 计算并统计适应度
int p[51];
int pLength = 0;
double fitness;
for (int i = 0; i < SIZE; ++i) {
	inds[i]->getPhe(p);
	pLength = inds[i]->getPheLength();
	fitness = getFitness(p, pLength);
	inds[i]->updateFitness(fitness);
}
double best = 10000;
for (int i = 0; i < SIZE; ++i) {
	if (inds[i]->getFitness() < best)
		best = inds[i]->getFitness();
}

for (int generation = 0; generation < 1000; generation++) {
	// 计算并统计适应度
	int p[51];
	int pLength = 0;
	double fitness;
	double fits[SIZE];
	for (int i = 0; i < SIZE; ++i) {
		inds[i]->getPhe(p);
		pLength = inds[i]->getPheLength();
		fitness = getFitness(p, pLength);
		inds[i]->updateFitness(fitness); // 此时不是真实的适应度
		fits[i] = fitness;
	}

	// 输出本次最优个体
	// 计算并统计适应度
	for (int i = 0; i < SIZE; ++i) {
		inds[i]->getPhe(p);
		pLength = inds[i]->getPheLength();
		fitness = getFitness(p, pLength);
		inds[i]->updateFitness(fitness);
```

```
}
double best = 10000;
int bestPLength = 0;
int bestP[51];
char tmp[51];
for (int i = 0; i < SIZE; ++i) {
	if (inds[i]->getFitness() < best) {
		best = inds[i]->getFitness();
		inds[i]->getGenCode(tmp);
		inds[i]->getPhe(bestP);
		bestPLength = inds[i]->getPheLength();
	}
}
cout << "generation:" << generation << " best:" << best << " detail:";
for (int i = 0; i < bestPLength; ++i)
	cout << bestP[i] << ",";
cout << endl;

f << "generation:" << generation << " best:" << best << " detail:";
for (int i = 0; i < bestPLength; ++i)
	f << bestP[i] << ",";
f << endl;

// 对临时适应度进行排序
quickSort(fits, 0, SIZE-1);
// 分配真实适应度
for (int i = 0; i < SIZE; ++i) {
	double tmp = inds[i]->getFitness();
	for (int j = 0; j < SIZE; ++j) {
		if (fits[j] == tmp) {
			inds[i]->updateFitness(fits[SIZE-1-j]);
			break;
		}
	}
}

// 进行选择操作
double range[SIZE-1];
double fitSum = 0;
for (int i = 0; i < SIZE-1; ++i) {
	fitSum += inds[i]->getFitness();
	range[i] = fitSum;
}
fitSum += inds[SIZE-1]->getFitness();
```

```
for (int i = 0; i < SIZE; ++i) {
	int randomrange = fitSum;
	int seed = rand() % (randomrange+1);
	int chosen = 0;
	if (seed <= range[0])
		chosen = 0;
	for (int j = 1; j < SIZE - 1; ++j) {
		if (seed > range[j-1] && seed <= range[j]) {
			chosen = j;
			break;
		}
	}
	if (seed > range[SIZE - 2])
		chosen = SIZE - 1;

	tmpInds[i] = new indivisual(inds[chosen]);
}
for (int i = 0; i < SIZE; ++i) {
	delete inds[i];
	inds[i] = tmpInds[i];
}

// 进行交叉运算
int matingRule[SIZE]; // 配对规则
for (int i = 0; i < SIZE; ++i)
	matingRule[i] = i;
for (int i = 0; i < SIZE; ++i) {
	int matingSeed = ( rand() % (SIZE-i) ) + i;
	int temp;
	temp = matingRule[i];
	matingRule[i] = matingRule[matingSeed];
	matingRule[matingSeed] = temp;
}
for (int i = 0; i < SIZE; i+=2) {
	char child2[51];
	tmpInds[i] = new indivisual(inds[matingRule[i]], inds[matingRule[i+1]], child2);
	tmpInds[i+1] = new indivisual(child2);
	//tmpInds[i] = new indivisual(inds[matingRule[i]]);
	//tmpInds[i+1] = new indivisual(inds[matingRule[i+1]]);
}
for (int i = 0; i < SIZE; ++i) {
	delete inds[i];
	inds[i] = tmpInds[i];
}
```

```
            // 进行变异
            for (int i = 0; i < SIZE; ++i) {
                inds[i]->mutation();
            }

        }


        // 回收空间
        for (int i = 0; i < SIZE; ++i) {
            delete inds[i];
        }

        return 0;
}

void quickSort(double a[], int low, int high)
{
        int mid;
        if (low >= high)
            return;
        mid = divide(a, low, high);
        quickSort(a, low, mid-1);
        quickSort(a, mid+1, high);
}

int divide(double a[], int low, int high)
{
        double k = a[low];
        do {
            while (low < high && a[high] >= k)
                --high;
            if (low < high) {
                a[low] = a[high];
                ++low;
            }
            while (low < high && a[low] <= k)
                ++ low;
            if (low < high) {
                a[high] = a[low];
                --high;
            }
        } while (low != high);
```

```
    a[low] = k;
    return low;
}
```

3 使用线性插值的 C++代码

以下代码在 CodeBlocks 10.05 中测试通过，运行结果保存将会打印在屏幕上并且保存在 output.txt 文件中：

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <ctime>
#include <fstream>

using namespace std;

int findNearestNeighbourIndex( double value, vector< double > &x )
{
    double dist = 99999;
    int idx = 0;
    for ( int i = 0; i < x.size(); ++i ) {
        double newDist = value - x[i];
        if ( newDist > 0 && newDist < dist ) {
            dist = newDist;
            idx = i;
        }
    }

    return idx;
}
vector< double > interp1( vector< double > &x, vector< double > &y, vector< double > &x_new )
{
    vector< double > y_new;
    y_new.reserve( x_new.size() );

    std::vector< double > dx, dy, slope, intercept;
    dx.reserve( x.size() );
    dy.reserve( x.size() );
    slope.reserve( x.size() );
    intercept.reserve( x.size() );
    for( int i = 0; i < x.size(); ++i ){
        if( i < x.size()-1 )
        {
            dx.push_back( x[i+1] - x[i] );
```

```
            dy.push_back( y[i+1] - y[i] );
            slope.push_back( dy[i] / dx[i] );
            intercept.push_back( y[i] - x[i] * slope[i] );
        }
        else
        {
            dx.push_back( dx[i-1] );
            dy.push_back( dy[i-1] );
            slope.push_back( slope[i-1] );
            intercept.push_back( intercept[i-1] );
        }
    }

    for ( int i = 0; i < x_new.size(); ++i )
    {
        int idx = findNearestNeighbourIndex( x_new[i], x );
        y_new.push_back( slope[idx] * x_new[i] + intercept[idx] );

    }

    return y_new;
}

double getFitness(int *p, int L){
    ifstream f("20141010dataform.csv");
    double costs = 0.0;
    double Y[51];
    char ch1[7];

    for(int n = 0; n < 469; n++)
    {
        double tmp = 0;

        for(int i = 0; i < 50; i++) f.getline(ch1, 7, ',');
        f.getline(ch1, 7);
        for(int i = 0; i < 50; i++) { f.getline(ch1, 7, ','); Y[i] = atof(ch1);}
        f.getline(ch1, 7); Y[50] = atof(ch1);

        vector<double> x(L);
        vector<double> y(L);

        vector<double> X(51);
        for (int i = 0; i < 51; ++i){
            X[i] = 5.00 + 0.10 * i;
```

```
            }

            for (int i = 0; i < L; ++i){
                x[i] = X[(p[i]) - 1];
                y[i] = Y[(p[i]) - 1];
            }

            vector<double> result(51);
            result = interp1(x, y, X);

            for(int i = 0; i < 51; i++)
            {
                double deviation = result[i] - Y[i];
                if (deviation < 0)
                    deviation = 0 - deviation;
                if(deviation > 0.5 && deviation <= 1) costs += 0.5;
                else if(deviation > 1 && deviation <= 2) costs += 1.5;
                else if(deviation > 2 && deviation <= 3) costs += 6;
                else if(deviation > 3 && deviation <= 5) costs += 12;
                else if(deviation > 5) costs += 25;
            }
            costs += L*12;
        }

        return costs/469;
}


class indivisual {
        char genotype[51]; //  基因型
        int phenotype[51]; //  表现型
        int pheLength;
        double fitness; //  适应度
public:
        indivisual() {
            for (int i = 0; i < 51; ++i) {
                int tmp = rand() % 2;
                if (tmp == 0)
                    genotype[i] = '0';
                else if (tmp == 1)
                    genotype[i] = '1';
            }
            deCode();
            fitness = 0;
```

```
}
indivisual(const char * s) {
	genCopy(s);
	deCode();
	fitness = 0;
}
indivisual(const indivisual * x) {
	char code[51];
	x->getGenCode(code);
	genCopy(code);
	deCode();
	fitness = x->getFitness();
}
// 配对产生新个体的构造函数
indivisual(const indivisual * x1, const indivisual * x2, char * child2) {
	// 随机产生交叉点个数（0-50）
	int crossNum = rand() % 51;
	int cross[50];
	for (int i = 0; i < crossNum; ++i) {
		int pre;
		if ( i > 0 )
			pre = cross[i-1];
		else
			pre = 0;
		cross[i] = rand() % (52 - pre);
	}
	char code1[51];
	char code2[51];
	x1->getGenCode(code1);
	x2->getGenCode(code2);

	char code[51];

	int index = 0;
	int order = 0;
	for (int i = 0; i < crossNum; ++i) {

		for (int j = index; j < index + cross[i]; ++j) {
			if ( order % 2 == 0 ) {
				code[j] = code1[j];
				child2[j] = code2[j];
			}
			else {
				code[j] = code2[j];
```

```
                        child2[j] = code1[j];
                }
                ++order;
        }
        index = cross[i];
    }
    for (int i = index; i < 52; ++i) {
        if ( order % 2 == 0 ) {
            code[i] = code1[i];
            child2[i] = code2[i];
        }
        else {
            code[i] = code2[i];
            child2[i] = code1[i];
        }
        ++order;
    }
    genCopy(code);
    deCode();
    fitness = 0;
}
void getGenCode(char * s) const {
    for (int i = 0; i < 51; i++)
        s[i] = genotype[i];
}
void getPhe(int * s) {
    for (int i = 0; i < pheLength; i++){
        s[i] = phenotype[i];
    }
}
int getPheLength() {
    return pheLength;
}
void updateFitness(double f) {
    fitness = f;
}
double getFitness() const {
    return fitness;
}
void mutation() {
    for (int i = 0; i < 51; ++i) {
        int index = rand() % 200;
        if (index == 0) {
            if (genotype[index] == '1')
```

```
                    genotype[index] = '0';
                else if (genotype[index] == '0')
                    genotype[index] = '1';
            }
        }

        deCode();
    }
private:
    void genCopy(const char * s) {
        for (int i = 0; i < 51; i++) {
            genotype[i] = s[i];
        }
    }
    void deCode() {
        int j = 0;
        for (int i = 0; i < 51; i++) {
            if (genotype[i] == '1') {
                phenotype[j++] = i+1;
            }
        }
        pheLength = j;
    }
};

void quickSort(double a[], int low, int high);
int divide(double a[], int low, int high);
int main(int argc, const char * argv[]) {
    srand(time(NULL));

    const int SIZE = 500;
    ofstream f("ouput.txt");

    // 产生初始群体
    indivisual * inds[SIZE];
    indivisual * tmpInds[SIZE];
    for (int i = 0; i < SIZE; ++i)
        inds[i] = new indivisual();

    // 计算并统计适应度
    int p[51];
    int pLength = 0;
    double fitness;
    for (int i = 0; i < SIZE; ++i) {
```

```
        inds[i]->getPhe(p);
        pLength = inds[i]->getPheLength();
        fitness = getFitness(p, pLength);
        inds[i]->updateFitness(fitness);
}
double best = 10000;
for (int i = 0; i < SIZE; ++i) {
        if (inds[i]->getFitness() < best)
                best = inds[i]->getFitness();
}

for (int generation = 0; generation < 1000; generation++) {
        // 计算并统计适应度
        int p[51];
        int pLength = 0;
        double fitness;
        double fits[SIZE];
        for (int i = 0; i < SIZE; ++i) {
                inds[i]->getPhe(p);
                pLength = inds[i]->getPheLength();
                fitness = getFitness(p, pLength);
                inds[i]->updateFitness(fitness); // 此时不是真实的适应度
                fits[i] = fitness;
        }

        // 输出本次最优个体
        // 计算并统计适应度
        for (int i = 0; i < SIZE; ++i) {
                inds[i]->getPhe(p);
                pLength = inds[i]->getPheLength();
                fitness = getFitness(p, pLength);
                inds[i]->updateFitness(fitness);
        }
        double best = 10000;
        int bestPLength = 0;
        int bestP[51];
        char tmp[51];
        for (int i = 0; i < SIZE; ++i) {
                if (inds[i]->getFitness() < best) {
                        best = inds[i]->getFitness();
                        inds[i]->getGenCode(tmp);
                        inds[i]->getPhe(bestP);
                        bestPLength = inds[i]->getPheLength();
                }
```

```
}
cout << "generation:" << generation << " best:" << best << " detail:";
for (int i = 0; i < bestPLength; ++i)
	cout << bestP[i] << ",";
cout << endl;

f << "generation:" << generation << " best:" << best << " detail:";
for (int i = 0; i < bestPLength; ++i)
	f << bestP[i] << ",";
f << endl;

// 对临时适应度进行排序
quickSort(fits, 0, SIZE-1);
// 分配真实适应度
for (int i = 0; i < SIZE; ++i) {
	double tmp = inds[i]->getFitness();
	for (int j = 0; j < SIZE; ++j) {
		if (fits[j] == tmp) {
			inds[i]->updateFitness(fits[SIZE-1-j]);
			break;
		}
	}
}

// 进行选择操作
double range[SIZE-1];
double fitSum = 0;
for (int i = 0; i < SIZE-1; ++i) {
	fitSum += inds[i]->getFitness();
	range[i] = fitSum;
}
fitSum += inds[SIZE-1]->getFitness();
for (int i = 0; i < SIZE; ++i) {
	int randomrange = fitSum;
	int seed = rand() % (randomrange+1);
	int chosen = 0;
	if (seed <= range[0])
		chosen = 0;
	for (int j = 1; j < SIZE - 1; ++j) {
		if (seed > range[j-1] && seed <= range[j]) {
			chosen = j;
			break;
		}
	}
```

```
            if (seed > range[SIZE - 2])
                chosen = SIZE - 1;

            tmpInds[i] = new indivisual(inds[chosen]);
        }
        for (int i = 0; i < SIZE; ++i) {
            delete inds[i];
            inds[i] = tmpInds[i];
        }

        // 进行交叉运算
        int matingRule[SIZE]; // 配对规则
        for (int i = 0; i < SIZE; ++i)
            matingRule[i] = i;
        for (int i = 0; i < SIZE; ++i) {
            int matingSeed = ( rand() % (SIZE-i) ) + i;
            int temp;
            temp = matingRule[i];
            matingRule[i] = matingRule[matingSeed];
            matingRule[matingSeed] = temp;
        }
        for (int i = 0; i < SIZE; i+=2) {
            char child2[51];
            tmpInds[i] = new indivisual(inds[matingRule[i]], inds[matingRule[i+1]], child2);
            tmpInds[i+1] = new indivisual(child2);
            //tmpInds[i] = new indivisual(inds[matingRule[i]]);
            //tmpInds[i+1] = new indivisual(inds[matingRule[i+1]]);
        }
        for (int i = 0; i < SIZE; ++i) {
            delete inds[i];
            inds[i] = tmpInds[i];
        }

        // 进行变异
        for (int i = 0; i < SIZE; ++i) {
            inds[i]->mutation();
        }

    }


    // 回收空间
    for (int i = 0; i < SIZE; ++i) {
        delete inds[i];
```

```
	}

	return 0;
}

void quickSort(double a[], int low, int high)
{
	int mid;
	if (low >= high)
		return;
	mid = divide(a, low, high);
	quickSort(a, low, mid-1);
	quickSort(a, mid+1, high);
}

int divide(double a[], int low, int high)
{
	double k = a[low];
	do {
		while (low < high && a[high] >= k)
			--high;
		if (low < high) {
			a[low] = a[high];
			++low;
		}
		while (low < high && a[low] <= k)
			++ low;
		if (low < high) {
			a[high] = a[low];
			--high;
		}
	} while (low != high);
	a[low] = k;
	return low;
}
```

4 多项式拟合法的运行结果

```
generation:0 best:257.534 detail:2,3,4,6,11,15,24,26,29,30,36,37,38,42,43,49,50,
generation:1 best:266.181 detail:2,3,10,11,15,16,20,22,28,29,30,32,37,42,43,46,49,50,
generation:2 best:232.665 detail:1,4,7,13,15,22,27,28,29,31,33,46,47,51,
generation:3 best:263.834 detail:1,2,4,7,10,12,13,15,24,28,29,33,39,45,46,47,49,
generation:4 best:209.726 detail:1,5,9,11,22,28,29,41,45,46,47,49,
generation:5 best:179.642 detail:6,16,18,22,28,29,37,43,46,49,
generation:6 best:211.203 detail:5,6,8,13,26,29,30,34,41,45,47,49,
generation:7 best:185.542 detail:3,6,7,9,11,22,28,37,43,46,49,
```

generation:8 best:198.955 detail:2,3,6,7,9,11,22,28,37,43,46,49,
generation:9 best:195.516 detail:3,6,7,9,11,17,22,28,37,43,46,49,
generation:10 best:188.836 detail:1,6,7,9,11,22,28,37,43,46,49,
generation:11 best:211.664 detail:3,6,8,9,19,22,25,28,35,37,43,46,51,
generation:12 best:204.403 detail:5,6,13,20,22,28,29,41,45,46,47,49,
generation:13 best:214.382 detail:5,6,7,16,20,22,29,30,37,38,43,46,49,
generation:14 best:194.963 detail:5,6,8,13,22,28,29,41,46,49,51,
generation:15 best:209.068 detail:3,10,13,22,28,29,33,35,37,39,46,49,51,
generation:16 best:195.844 detail:3,13,18,24,29,30,41,42,46,49,50,51,
generation:17 best:183.542 detail:3,8,13,15,22,28,29,37,43,46,49,
generation:18 best:188.424 detail:3,10,13,16,24,29,30,41,46,49,51,
generation:19 best:178.033 detail:3,10,13,22,28,29,41,46,49,51,
generation:20 best:207.237 detail:3,6,13,16,20,22,29,37,41,45,46,47,49,
generation:21 best:195.396 detail:2,3,6,13,16,20,22,29,37,43,46,49,
generation:22 best:197.638 detail:2,3,12,13,16,20,28,29,37,43,46,49,
generation:23 best:208.098 detail:2,3,13,16,24,29,30,34,41,42,44,46,49,
generation:24 best:188.471 detail:2,6,16,24,25,29,30,34,41,46,49,
generation:25 best:183.812 detail:1,3,5,13,15,22,29,37,43,46,49,
generation:26 best:183.145 detail:3,8,13,16,20,28,29,37,43,46,49,
generation:27 best:175.535 detail:2,3,9,22,29,30,37,43,46,49,
generation:28 best:180.062 detail:2,7,13,22,23,29,30,43,46,49,
generation:29 best:173.041 detail:1,9,15,22,29,30,37,43,46,49,
generation:30 best:173.805 detail:1,3,13,22,29,30,37,43,46,49,
generation:31 best:177.592 detail:5,8,13,20,24,29,37,43,46,49,
generation:32 best:170.276 detail:3,7,13,22,29,30,37,43,46,49,
generation:33 best:172.994 detail:1,3,7,13,20,29,37,43,46,49,
generation:34 best:174.663 detail:1,3,12,13,20,29,37,43,46,49,
generation:35 best:160.586 detail:1,3,12,20,29,37,43,46,49,
generation:36 best:153.507 detail:1,3,12,20,29,43,46,49,
generation:37 best:153.981 detail:2,10,13,20,29,43,46,49,
generation:38 best:142.64 detail:1,10,20,29,43,46,49,
generation:39 best:142.606 detail:1,12,20,29,43,46,49,
generation:40 best:138.11 detail:3,10,20,29,43,46,49,
generation:41 best:149.517 detail:3,12,13,20,29,41,46,49,
generation:42 best:138.11 detail:3,10,20,29,43,46,49,
generation:43 best:142.64 detail:1,10,20,29,43,46,49,
generation:44 best:153.507 detail:1,3,12,20,29,43,46,49,
generation:45 best:138.33 detail:3,12,20,29,43,46,49,
generation:46 best:145.5 detail:3,12,20,29,37,43,46,49,
generation:47 best:138.11 detail:3,10,20,29,43,46,49,
generation:48 best:144.268 detail:1,13,22,29,43,46,49,
generation:49 best:150.563 detail:1,10,20,29,41,43,46,49,
generation:50 best:136.457 detail:3,13,29,43,46,49,
generation:51 best:146.933 detail:6,10,20,29,43,46,49,

generation:52 best:145.34 detail:3,13,15,29,43,46,49,
generation:53 best:145.051 detail:3,10,20,29,37,43,46,49,
generation:54 best:138.11 detail:3,10,20,29,43,46,49,
generation:55 best:130.82 detail:3,15,29,43,46,49,
generation:56 best:139.365 detail:3,15,20,29,43,46,49,
generation:57 best:136.671 detail:3,20,29,43,46,49,
generation:58 best:138.562 detail:3,13,20,29,43,46,49,
generation:59 best:138.11 detail:3,10,20,29,43,46,49,
generation:60 best:136.671 detail:3,20,29,43,46,49,
generation:61 best:136.671 detail:3,20,29,43,46,49,
generation:62 best:136.671 detail:3,20,29,43,46,49,
generation:63 best:138.562 detail:3,13,20,29,43,46,49,
generation:64 best:136.671 detail:3,20,29,43,46,49,
generation:65 best:136.671 detail:3,20,29,43,46,49,
generation:66 best:136.671 detail:3,20,29,43,46,49,
generation:67 best:136.671 detail:3,20,29,43,46,49,
generation:68 best:136.671 detail:3,20,29,43,46,49,
generation:69 best:136.671 detail:3,20,29,43,46,49,
generation:70 best:136.671 detail:3,20,29,43,46,49,
generation:71 best:136.671 detail:3,20,29,43,46,49,
generation:72 best:136.671 detail:3,20,29,43,46,49,
generation:73 best:136.671 detail:3,20,29,43,46,49,
generation:74 best:136.671 detail:3,20,29,43,46,49,
generation:75 best:136.671 detail:3,20,29,43,46,49,

5 线性插值法的运行结果

generation:0 best:178.454 detail:5,7,11,19,20,21,24,26,28,30,33,35,48,
generation:1 best:199.273 detail:3,4,5,6,20,21,23,24,27,33,37,39,42,49,50,51,
generation:2 best:196.659 detail:1,6,10,13,15,16,19,21,28,32,37,44,45,46,49,50,
generation:3 best:195.71 detail:2,5,6,10,15,20,26,29,35,37,40,42,44,46,48,51,
generation:4 best:183.02 detail:5,6,10,12,15,29,34,35,37,38,42,50,51,
generation:5 best:190.186 detail:2,3,8,13,14,15,16,17,26,32,34,36,43,47,48,
generation:6 best:165.501 detail:6,10,21,23,24,28,32,40,41,44,45,50,51,
generation:7 best:177.324 detail:4,8,11,13,14,20,26,32,33,34,35,42,45,47,
generation:8 best:183.919 detail:1,7,13,16,17,19,20,21,24,28,32,36,37,45,49,
generation:9 best:165.864 detail:6,8,10,15,16,18,24,27,28,32,41,45,51,
generation:10 best:164.267 detail:2,7,9,12,14,28,32,34,36,43,47,50,
generation:11 best:131.969 detail:1,2,17,18,21,27,34,41,43,51,
generation:12 best:149.845 detail:5,7,14,17,24,28,32,34,41,45,46,51,
generation:13 best:139.966 detail:7,14,21,28,32,34,36,40,43,48,49,
generation:14 best:144.087 detail:14,15,27,28,34,36,43,46,49,
generation:15 best:142.723 detail:5,9,11,20,21,23,30,38,43,49,50,
generation:16 best:142.807 detail:1,4,20,26,27,30,38,43,44,49,50,
generation:17 best:125.921 detail:3,17,18,21,25,29,33,38,43,50,
generation:18 best:127.813 detail:2,5,17,18,21,30,32,43,50,

generation:19 best:132.247 detail:3,7,15,20,26,29,38,41,43,50,
generation:20 best:125.77 detail:3,16,20,24,28,34,40,43,46,49,
generation:21 best:117.45 detail:3,22,28,34,42,43,48,49,
generation:22 best:114.191 detail:1,14,17,20,30,38,43,50,
generation:23 best:112.568 detail:4,18,27,30,38,39,49,50,
generation:24 best:111.819 detail:12,18,24,30,38,43,49,50,
generation:25 best:119.642 detail:6,20,26,28,32,40,43,48,49,
generation:26 best:118.2 detail:3,14,18,19,24,30,38,43,50,
generation:27 best:107.435 detail:9,19,20,26,32,38,43,49,
generation:28 best:114.913 detail:5,18,28,31,40,41,45,48,
generation:29 best:111.321 detail:12,17,24,32,38,43,49,50,
generation:30 best:112.373 detail:5,13,17,28,34,40,43,49,
generation:31 best:102.946 detail:5,12,20,30,38,43,50,
generation:32 best:111.069 detail:11,17,18,24,30,38,43,49,
generation:33 best:106.824 detail:7,19,24,30,38,43,49,50,
generation:34 best:107.147 detail:1,5,20,26,32,38,43,49,
generation:35 best:98.4552 detail:4,17,27,34,42,49,50,
generation:36 best:102.979 detail:1,17,21,27,30,38,49,
generation:37 best:96.6461 detail:9,19,30,38,43,49,
generation:38 best:95.0672 detail:4,17,27,30,38,49,
generation:39 best:96.1493 detail:4,21,30,38,45,47,
generation:40 best:95.0736 detail:3,17,27,30,38,49,
generation:41 best:95.8443 detail:1,17,24,27,34,43,49,
generation:42 best:96.2292 detail:7,18,24,30,38,43,49,
generation:43 best:96.4286 detail:5,13,24,30,38,49,
generation:44 best:98.0107 detail:1,15,24,30,38,43,49,
generation:45 best:96.4883 detail:1,16,24,30,38,47,50,
generation:46 best:95.291 detail:5,19,30,38,43,49,
generation:47 best:96.6066 detail:3,19,24,30,38,43,49,
generation:48 best:95.8412 detail:1,17,27,30,38,49,
generation:49 best:95.1557 detail:7,22,30,38,43,50,
generation:50 best:86.1279 detail:5,18,26,34,43,50,
generation:51 best:86.1279 detail:5,18,26,34,43,50,
generation:52 best:86.1279 detail:5,18,26,34,43,50,
generation:53 best:86.5554 detail:7,18,26,34,43,50,
generation:54 best:86.2729 detail:5,19,26,34,43,50,
generation:55 best:87.2431 detail:1,18,26,34,43,50,
generation:56 best:86.0597 detail:4,18,26,34,43,50,
generation:57 best:86.0597 detail:4,18,26,34,43,50,
generation:58 best:86.0597 detail:4,18,26,34,43,50,
generation:59 best:86.0597 detail:4,18,26,34,43,50,
generation:60 best:86.0597 detail:4,18,26,34,43,50,
generation:61 best:86.0597 detail:4,18,26,34,43,50,
generation:62 best:86.0597 detail:4,18,26,34,43,50,

generation:63 best:86.0597 detail:4,18,26,34,43,50,
generation:64 best:86.0597 detail:4,18,26,34,43,50,
generation:65 best:86.0597 detail:4,18,26,34,43,50,
generation:66 best:86.0597 detail:4,18,26,34,43,50,
generation:67 best:86.0597 detail:4,18,26,34,43,50,
generation:68 best:86.0597 detail:4,18,26,34,43,50,
generation:69 best:86.0597 detail:4,18,26,34,43,50,
generation:70 best:86.0597 detail:4,18,26,34,43,50,
generation:71 best:86.0597 detail:4,18,26,34,43,50,
generation:72 best:86.0597 detail:4,18,26,34,43,50,
generation:73 best:86.0597 detail:4,18,26,34,43,50,