统计推断在数模转换系统中的应用

组号：38 姜蔚涛 5130309144 任杰 5130309310

摘要：本课题要求为某模块的批量生产设计一种成本合理的传感特性校准（定标工序）方案。而监测模块中传感器部件的输入输出特性呈明显的非线性。为了了解实际的特性曲线信息、研究对象系统的输入输出函数关系，唯一的办法就是通过采样进行拟合曲线，然而但采样值和采样总体达到一个极限值时，这项工艺的耗时会十分庞大，而为了精确绘制特性曲线，我们不得不增大采样点。就该系统而言，在工程实际中，若对每个样品都进行完整测定，则既费时又高成本。若能减少需要观测的数值组数量，则可以提高工效。针对这一矛盾，就必须通过一个优化采样的算法来解决。

关键字： 三次多项式拟合，三次样条插值，遗传算法

1 引言

本课题研究的内容为被监测物理量 Y 与 X 之间的函数关系，由于两者并不存在确定的函数关系，因而可以通过研究大量的数据寻找适当的拟合函数。由于样本存在个体差异，因而对单个样本的研究不能代表整体。在对整体研究时，如果所用所有点进行函数拟合，运算量大，成本太高。因而，该课题的研究内容为寻求一个最优化方法，在成本最低的前提下尽量减小误差。

2 课题基本问题的分析

为了对本课题展开有效讨论，需建立一个数学模型，对问题的某些方面进行必要的描述和限定。

监测模块的组成框图如图 2-1。其中，传感器部件（包含传感器元件及必要的放大电路、调理电路等）的特性是我们关注的重点。传感器部件监测的对象物理量以符号 Y 表示；传感部件的输出电压信号用符号 X 表示，该电压经模数转换器（ADC）成为数字编码，并能被微处理器程序所读取和处理，获得信号 $\hat{Y}$ 作为 Y 的读数（监测模块对 Y 的估测值）。

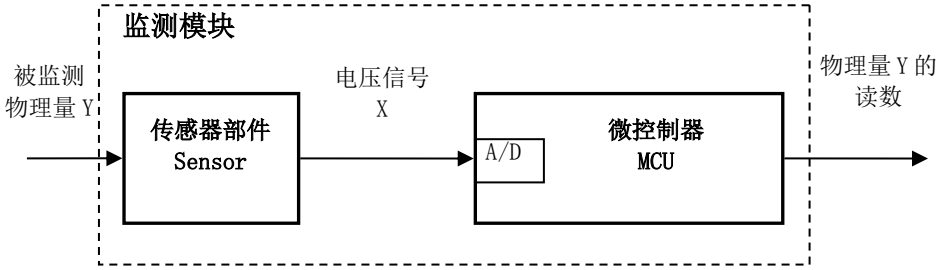

图 2-1 监测模块组成框图

所谓传感特性校准，就是针对某一特定传感部件个体，通过有限次测定，估计其 Y 值与 X 值间一一对应的特性关系的过程。数学上可认为是确定适用于该个体的估测函数 $\hat{y} = f(x)$ 的过程，其中 x 是 X 的取值，$\hat{y}$ 是对应 Y 的估测值。

考虑实际工程中该监测模块的应用需求，同时为便于在本课题中开展讨论，我们将问题限于 X 为离散取值的情况，规定

$$X \in \{x_1, x_2, x_3, ..., x_{50}, x_{51}\} = \{5.0, 5.1, 5.2, ..., 9.9, 10.0\}$$

相应的 Y 估测值记为 $\hat{y}_i = f(x_i)$，Y 实测值记为 y_i，$i = 1, 2, 3, ..., 50, 51$。

对于曲线的拟合，肯定是拟合点数越多，拟合的越为精确，然而取点需要有成本，为了使成本最低，我们需要找到一个合适的拟合取点方案，平衡两者，使总的成本最小。

为评估和比较不同的校准方案，有以下成本计算规则：

（1）单点定标误差成本

$$s_{i,j}=\begin{cases}0 & \text{if } \left|\hat{y}_{i,j}-y_{i,j}\right|\le 0.5\\ 0.5 & \text{if } 0.5<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 1\\ 1.5 & \text{if } 1<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 2\\ 6 & \text{if } 2<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 3\\ 12 & \text{if } 3<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 5\\ 25 & \text{if } \left|\hat{y}_{i,j}-y_{i,j}\right|>5\end{cases} \qquad (2\text{-}1)$$

单点定标误差的成本按式（2-1）计算，其中 $y_{i,j}$ 表示第 i 个样本之第 j 点 Y 的实测值，$\hat{y}_{i,j}$ 表示定标后得到的估测值（读数），该点的相应误差成本以符号 $s_{i,j}$ 记。

（2）单点测定成本

实施一次单点测定的成本以符号 q 记。本课题指定 q=12。

（3）某一样本个体的定标成本

$$S_i=\sum_{j=1}^{51}s_{i,j}+q\cdot n_i \qquad (2\text{-}2)$$

对样本 i 总的定标成本按式（2-2）计算，式中 n_i 表示对该样本个体定标过程中的单点测定次数。

（4）校准方案总体成本

按式（2-3）计算评估校准方案的总体成本，即使用该校准方案对标准样本库中每个样本个体逐一定标，取所有样本个体的定标成本的统计平均。

$$C=\frac{1}{M}\sum_{i=1}^{M}S_i \qquad (2\text{-}3)$$

总体成本较低的校准方案，认定为较优方案。[1]

3 寻找最优的拟合方案

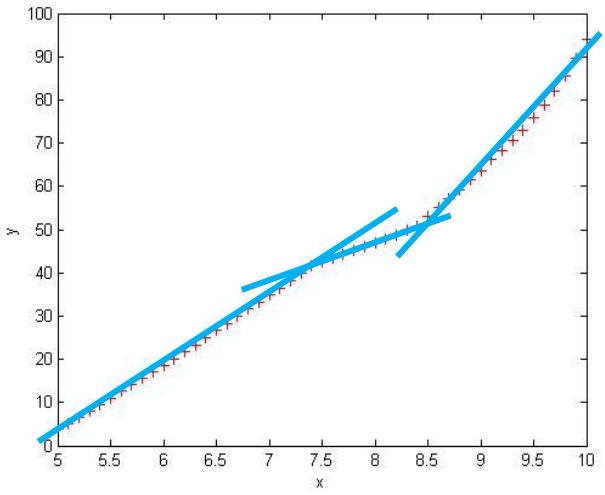

图 3-1　　X-Y 数据大致分布曲线

通过对原始数据的查看与分析，发现每组数据的 X-Y 曲线有这很大的相似性，大致可以看成由三段可近似看为直线段的曲线组成。曲线特性如图 3-1 所示。

由曲线的分布特性，我们联想到了三次函数，三次函数的曲线特性就大致分为三段走，于是我们首先采用的就是三次多项式的拟合这一方案。

3.1 拟合函数的选取

由于取点函数的多样性，我们并不知道针对课题的数据，选用何种拟合函数，可得成本最优，于是我们选取了两个拟合函数进行对比和探索。

3.1.1 三次多项式的拟合

由于前面已经讨论到，取点个数越多，则拟合曲线越为精准，而取点次数会影响成本总值，取点次数越多，成本越高，所以我们在此要寻找一个最优的取点方案，首先要找到一个最优的取点个数 n，能让总成本达到最低。

基于此目的，我们编写了相关三次多项式拟合的可以更改取点数目 n 的 matlab 程序，通过 1000 次的穷举，我们可以大致地认为在这个取点数目 n 下，已经找到最优的取点方案了，通过对不同取点数目 n 下的最优成本的分析，我们可以寻找得到最优的取点数目 n。

下面所列的就是 1000 次穷举下，不同取点数目 n 所对应的最优取点方案以及最优成本。

n＝4：

5　18　39　48

129.5544

n=5：

4　13　26　39　50

116.3945

n=6：

2　10　23　34　39　48

126.8124

n=7：

3　9　17　25　36　45　50

134.4360

n=8：

3　13　19　26　34　41　46　50

145.1557

通过对以上数据的分析，我们可以看出，在取点数目取 5 时，其最优取点方案总成本最低。

3.1.2 三次样条插值的拟合

通过对数据曲线的进一步分析，以及了解到了三次样条插值法这一拟合方案，我们预测，由于曲线三段较为线性，那么通过三次样条插值法得出的拟合曲线也许更为优质。

首先先来介绍一下三次样条插值法，三次样条插值（简称 Spline 插值）是通过一系列形值点的一条光滑曲线，数学上通过求解三弯矩方程组得出曲线函数组的过程。定义:函数 $S(x)\in C2[a,b]$，且在每个小区间$[xj,xj+1]$上是三次多项式，其中 $a=x0<x1<...<xn=b$ 是给定节点，则称 S(x)是节点 x0,x1,...xn 上的三次样条函数。若在节点 x_j 上给定函数值 $Yj=f(Xj).(j=0,1,,n)$，并成立 $S(xj)=yj.(j=0,1,,n)$，则称 S(x)为三次样条插值函数。

实际计算时还需要引入边界条件才能完成计算。边界通常有自然边界（边界点的二阶导为 0），夹持边界（边界点导数给定），非扭结边界（使两端点的三阶导与这两端点的邻近点的三阶导相等）。一般的计算方法书上都没有说明非扭结边界的定义，但数值计算软件如 Matlab 都把非扭结边界条件作为默认的边界条件。

我们不必刻意去追究三次插值法的实现，因为 matlab 程序已经为我们很好地将这一插值函数实现

。

于是，和三次多项式最优拟合方案探索思想类似，我们同样也通过编写可以更改取点数目 n 的三次样条插值的拟合函数的 matlab 程序来寻求最优的 n 取值。

下面所列的就是 1000 次穷举下，不同取点数目 n 所对应的最优取点方案以及最优成本。

n=4：

4　18　37　49

127.7857

n=5：

1　12　24　38　49

106.5352

n=6：

1　10　20　31　39　49

97.8443

n=7：

1　9　20　25　34　44　50

95.3049

n=8：

2　10　20　26　31　36　47　51

103.1908

通过对以上数据的分析，我们可以看出，在取点数目取 7 时，其最优取点方案总成本最低。

3.2 遗传算法

3.2.1 遗传算法介绍

遗传算法是具有“生成+检 测 ”的迭代过程的搜索算 法。基本流程如图 3-2 所示。可见,遗传算法是一种群体 型操作,该操作以群体中的所有个体为对象。选择(selection)、交叉(crossover)和变异(mutation)是 遗 传 算 法 的 三个主要操作算子。遗传算法包含如下 6 个基本要素:

(1)参数编码:由于遗传算法不能直接处理解空 间的解数据,因此必须通过编码将它们表示成遗传空 间的基因型串结构数据。

(2)生成初始群体:由于遗传算法的群体型操作 需要,所以必须为遗传操作准备一个由若干初始解组 成的初始群体。初始群体的每个个体都是通过随机方 法产生的。

(3)适 应 度 评 估 检 测 :遗 传 算 法 在 搜 索 进 化 过 程 中 一 般 不 需 要 其 他 外 部 信 息 ,仅 用 适 应 度(fitness)值来评估个体或解的优劣,并作为以后遗传操作的依 据。

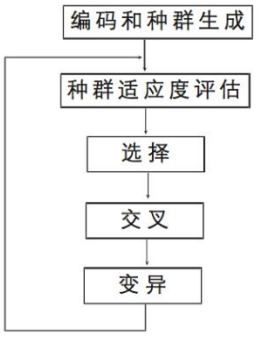

图 3-2 遗传算法流程图

(4)选 择(selection):选 择 或 复 制 操 作 是 为 了 从 当 前群体中选出优良的个体,使它们有机会作为父代为 下一代繁殖子孙。个体适应度越高,其被选择的机会 就越多。本文采用与适应度成比例的概率方法进行选 择。具体地说,就是首先计算群体中所有个体适应度 的 总 和,再 计 算 每 个 个 体 的 适 应 度 所 占 的 比 例,并以此作为相应的选择概率。

(5)交 叉 操 作 :交 叉 操 作 是 遗 传 算 法 中 最 主 要 的 遗传操作。简单的交叉(即一点交叉)可分两步进行: 首先对种群中个体进行随机配对;其次,在配对个体 中随机设定交叉处,配对个体彼此交换部分信息。

(6)变异:变异操作是按位(bit)进行的,即把某一 位的内容进行变异。变异操作同样也是随机进行的。一般而言,变异概率 p_m 都取得较小。变异操作是十分 微妙的遗传操作,它需要和交叉操作配合使用,目的 是挖掘群体中个体的多样性,克服有可能限于局部解 的弊病。[2]

遗传算法就是模拟进化论的优胜劣汰的规则，并加有概率性质的随机遗传的算法。初始群体的每个个体都是通过随机方法产生的。仅用适应度值来评估个体或解的优劣,并作为以后遗传操作的依据。选 择或复制操作是为了从当前群体中选出优良的个体,使它们有机会作为父代为下一代繁殖子孙。个体适应度越高,其被选择的机会就越多。交叉操作是遗传算法中最主要的遗传操作。简单的交叉(即一点交叉)可分两步进行: 首先对种群中个体进行随机配对;其次,在配对个体中随机设定交叉处,配对个体彼此交换部分信息。 变异操作是指每个父代个体都有一定的概率发生随机的基因突变。评价基因好坏的依据就是适应度。在遗传算法的步骤中起着突出作用的其实是变异。正因为有了变异，算法才存在在全局范围内寻找到最优解的可能性。但是遗传算法需要通过反复试验，才有可能给出最优解。

3.2.2 遗传算法思想在三次样条插值的应用

在先前的探索过程中，我们从结果中可以看出使用三次样条插值法并且取点数为 7 进行拟合时，可取得较为优秀的取点组合，故我们首先依据该结论将遗传算法仅用于取点数为 7 的三次样条插值法。我们设定交叉率为 0.8，变异率为 0.008，种群数目为 100，循环代数为 100，选择函数通过删除每一代中成本值大于平均成本值来实现。

在这个条件下，通过运行程序，我们得到的最优的取点方案以及成本是：

2.0000 11.0000 20.0000 27.0000 33.0000 44.0000 50.0000

94.4659

与先前穷举得出的最优解相比，有了一定的优化。

3.2.3 基于全局的遗传算法的应用

先前的算法只是大致应用了遗传算法的思想，而并未完全的应用遗传算法进行。我们只是通过穷举大致的了解了成本关于 n 的分布情况，然后基于这一结论进行的探索。真正的遗传算法的应是随机产生取点数 n 进行函数拟合的算法，而且我们只是大致以为当取点数目为 7 时，成本最优，不排除成本最优解并不是取点数为 7 的可能。所以基于这一想法，我们通过编写随机取点（不规定取点数目）的遗传算法程序进行最优取点方案的探索。

我们通过矩阵的方法来实现遗传算法的取点，先随机生成一个行数种群数，列数为 51 的 0-1 矩阵，每一行代表一个个体，第 i 行第 j 列为 1 代表第 i 个个体取第 j 个数，反之为 0 则表示不取，然后通过适应度的评估，以及选择，交叉以及变异等步骤完成第一代的进化。其中适应度函数我们使用了成本函数的倒数来进行代替。经过一代代的进化，可以获得最优的个体。

我们设置种群数目为 100，交叉率为 0.8，变异率为 0.008，循环代数为 100，最终得出最优的取点方案以及成本是：

1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

99.0981

这意味着最优的取点方案仍是取 7 个点，分别为 1，9，20，30，32，42，51，最低成本可到 99.0981 。

3.3 结果的分析

3.3.1 拟合函数选择结果的分析

就拟合函数的选取带来的成本差异而言，还是很大的，三次样条插值法要明显好于三次多项式拟合。1000 次穷举的三次样条插值法的最好结果已经达到了 95.3049，我们分析了导致这一差异的原因。我们认为首先根据数据的分布情况，已经在上文中有过介绍，就是数据曲线大致分为前段，中段，后段这三段，并且这三段的线性度较高，正好符合三次插值法是取两点之间连线以进行拟合的特性，而三次多项式的拟合还是更偏向于曲线化，所以就这个课题而言，还是三次样条插值法的精确度更高。

3.3.2 遗传算法的分析

在选定了三次样条插值法之后，我们就要通过对取点的优化来更好地进行函数拟合以使成本最小。我们首先通过不同取点数目 n 值下的穷举来探索最优的取点数目，最终得出结论是当取点数目为 7 时，成本最优。 然而暴力穷举固然可以找到一些很好的点，却由于随机性而导致了一定的不确定性，且时间需要要求很高。为了使结果更加优秀，我们选择了遗传算法这一方法。我们首先在此结论基础上，运用遗传算法思想编写程序，通过 100 代的循环，最后得出了最低成本为 94.4659 的取点方案。

由于之前的算法是基于一个事先探索得出的结论，所以算法缺乏智能型，于是编写了基于全局的遗传算法，设定种群数为 100，通过一百代的进化，得出的最小成本为 99.0981。

我们发现遗传算法得出的最后结果没有在先前做过一定铺垫后的结果来的好，这也并不奇怪。因为最后遗传算法得出的点也是取了七个点，而由于种群生成的随机性，种群中还夹杂了其他很多点数的个体，相比于限定在七个点的算法在相同条件下肯定没有那么优秀。

3.3.3 最终结果的分析

我们通过基于先前铺垫工作后编写的遗传算法得出的结论是最优的成本为 94.4659 。然而不管是哪种方法，由于取点的随机性，所以并不能保证这就是最优的取点方案了，而且更不能保证是全局下的最优解。

4 总结

在本课题中，我们通过拟合函数的选择以及通过算法的优化，最终找到的最好的取点方案成本最低为 94.4659 。其实三次多项式拟合和三次插值法都是非常好的拟合函数，对于不同的问题要具体问题具体分析，看哪个函数拟合方法更加符合数据曲线的曲线特性，然后进行选择，只不过在本课题中，三次样条插值法这一拟合函数更为优秀。我们所选择的遗传算法是一种模拟生物进化过程的一种寻优算法，在这个课题中适用，可以为我们进一步地优化最后的结果，通过不断地进化最终找到最优的取点方案，同时在其他很多问题中也能运用遗传算法以帮助我们进行解的优化。

5 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].殷 铭 张兴华 戴先中.基于 MATLAB 的遗传算法实现[D].南京.东南大学自动控制 系.2000

附录：

程序 1（注：可以通过删除备注%符号，进行三次样条插值法与三次多项式拟合法的切换，通过设置 n 值来控制取点数目）

```
data=csvread('20141010dataform.csv');
allcosts = zeros(1,100);
max = 0;
n=7;
points = zeros(1,n);
x_point=zeros(1,n);
y_point=zeros(1,n);
allcost=0;
result=zeros(1,n);
min=1000;
cost=0;
step=ceil(51/n);
for  k = 1:1000
    allcost=0;
    for m=1:n-1
        points(m)=randi([1+(m-1)*step m*step]);
    end;
    points(n)=randi([(n-1)*step+1 51]);
    for i=1:469
    x=data(2*i-1,:);
    y=data(2*i,:);
    for j=1:n
        x_point(j)=x(points(j));
        y_point(j)=y(points(j));
    end;
    %p1=polyfit(x_point,y_point,3);
    %p2=polyval(p1,x);
    p2=interp1(x_point,y_point,[5.0:0.1:10.0],'spline');
    deta=p2(1:51)-y(1:51);
    cost_1=0;
    for i = 1:51
        if abs(deta(i))<=0.5
           cost_1 = cost_1;
        elseif abs(deta(i))<=1
              cost_1 = cost_1+0.5;
        elseif abs(deta(i))<=2
              cost_1 = cost_1+1.5;
        elseif abs(deta(i))<=3
```

```
            cost_1 = cost_1+6;
    elseif abs(deta(i))<=5
            cost_1 = cost_1+12;
        else   cost_1 = cost_1+25;
        end;
    end;
     allcost = allcost + cost_1+12*n;

end;
cost= allcost/469;
if min>cost
    min=cost;
    for l=1:n
        result(l)=points(l);
    end;
end;
end

disp(result);
disp(min);
```

程序 2（基于三次样条插值法的取点数为 7 的遗传算法）

主函数

```
pcross = 0.8;
pmutation = 0.008;
size = 100;
data = csvread('20141010dataform.csv');
loop = 100;
min=1000;
flag=0;

while(flag==0)
    [pop flag]=initialise1(size);
end;
points = zeros(1,8);

for i = 1:loop
    pop1=mutation1(pop,pmutation,data,size);
    pop0=crossover1(pop1,data,pcross);
    all=0;
```

```
    number=0;
    min=1000;
    for j=1:size
        if pop0(j,8)<200
            if pop0(j,8)>10

                all=all+pop0(j,8);
                number=number+1;
                %disp(pop0(j,1:8));
                %disp(number);
            end;
        end;
    end;
    all=all/number;
    for k = 1:size
        if (pop0(k,8) < min)
            if(pop0(k,2) >1)
                min=pop0(k,8);
                points=pop0(k,1:8);
            end;
        end;
    end;
    disp(points);
    for l=1:size
        if(pop0(l,8) >all)
            flag=0;
            while(flag==0)
                [pop0(l,1:8) flag]=initialise1(1);
            end;
        end;
    end;
end;
```

初始设定函数 function [pop flag] =initialise1(size)

```
pop=ones(size ,8)*51;
flag=0;

for i=1:size
    pop(i,1) = randi([2,6]);
    pop(i,2) = randi([7,15]);
    pop(i,3) = randi([16,22]);
```

```
    pop(i,4) = randi([23,29]);
    pop(i,5) = randi([30,38]);
    pop(i,6) = randi([39,46]);
    pop(i,7) = randi([47,51]);
end;

if size==1
    flag=1;
end;

for j=1:size-1
    for k=j+1:size
        for l=1:7
            if(pop(j,l)~=pop(k,l))
                flag=1;
                break;
            end;
        end;
    end;
end;
```

变异函数 function child=mutation1(pop,pmutation,data,size)

```
child=pop;
arr=ones(1,51)+randperm(51);
for i = 1: size
    if(rand < pmutation)
       point= randi([1,7]);
       point1=setdiff(arr,pop(i,1:7));
       child(i,point) = point1(randi([1,44]));
    end;
end;
child= sort(child,2);
for p = 1:size
   child(p,8) = fitness1(child(p,1:7),data);
end;
```

交叉函数 function child =crossover1(pop,data,pcross)

```
[px,py]=size(pop);
child=ones(px,py);
for i=1:2:(px-1)
    if(rand<pcross)
        childpoint= randi([1,6]);
        cost_1 = pop(i,7)+pop(i+1,7);
```

```
        child(i,1:7) = [pop(i,1:childpoint) pop(i+1,childpoint+1:7)];
        child(i+1,1:7) = [pop(i+1,1:childpoint) pop(i,childpoint+1:7)];
        child(i,8) = fitness1(child(i,1:7),data);
        child(i+1,8) = fitness1(child(i+1,1:7),data);
        cost_2 = child(i,8)+child(i+1,8);
        if(cost_2>cost_1)
            child(i,:)=pop(i,:);
            child(i+1,:)=pop(i+1,:);
       end;
    end;
end;
```

成本计算函数 function cost=fitness1(point,data)

```
cost=0;
for i=1:469
    x_point=zeros(1,7);
    y_point=zeros(1,7);
    x=data(2*i-1,1:51);
    y=data(2*i,1:51);

    for j=1:7
        x_point(j)=x(point(j));
        y_point(j)=y(point(j));
    end;

    for k=1:51
        a(k)=data(2*i-1,k);
    end;
    x0=[5.0:0.1:10.0];
    p2=interp1(x_point,y_point,x0,'spline');
    %p1=polyfit(x_point,y_point,3);
    %p2=polyval(p1,x);
    deta=p2-y;
    cost_1=0;
    for i = 1:51
        if abs(deta(i))<=0.5
           cost_1 = cost_1;
        elseif abs(deta(i))<=1
              cost_1 = cost_1+0.5;
        elseif abs(deta(i))<=2
```

```
            cost_1 = cost_1+1.5;
        elseif abs(deta(i))<=3
            cost_1 = cost_1+6;
    elseif abs(deta(i))<=5
            cost_1 = cost_1+12;
        else   cost_1 = cost_1+25;
        end;
    end;
    cost = cost + cost_1+7*12;
end;
cost= cost/469;
```

程序 3（基于矩阵方法的遗传算法）

主程序

```
pcross=0.8;
pmutation=0.008;
size=100;
data=csvread('20141010dataform.csv');
loop=100;
min=1000;
flag=0;
pops=zeros(size,51);
fitness=zeros(1,size);
min=1000;
temp=zeros(1,size);
pops=initialise2(size);
cost=zeros(1,size);
for i=1:size
    cost(i)=fitness2(pops(i,:),data);
end;


for i=1:loop
    for j=1:size
        fitness(j)=1/fitness2(pops(j,:),data);
    end;
    allsum=zeros(1,size);
    for j=1:size
        sum=0;
        for k=1:j
            sum=fitness(k)+sum;
        end;
        allsum(j)=sum;
    end;
    pops=mutation2(pops,pmutation,data,size);
```

```
    pops=crossover2(pops,data,pcross);
    for j=1:size
        cost(j)=fitness2(pops(j,:),data);
    end;
    for j=1:size
        if(cost(j)<min)
            min=cost(j);
            number=j;
        end;
    end;


    disp(min);
    disp(pops(number,:));
end;
```

种群生成函数 function pops=initialise2(size)

```
pops=zeros(size,51);
for i=1:size
    pop=zeros(1,51);
    for j=1:51
        temp=rand(1);
        if(temp>=0.5)
            pop(j)=1;
        end;
    end;
    pops(i,:)=pop;
end;
```

变异函数 function child=mutation2(pop,pmutation,data,size)

```
child=pop;
arr=zeros(1,51)+rand;
for i = 1: size
    if(rand < pmutation)
       point=randi([1,51]);
       point1=setdiff(arr,pop(i,1:51));
       child(i,point) = point1(rand);
    end;
end;
child= sort(child,2);
for p = 1:size
   child(p,52) = fitness2(child(p,1:51),data);
end;
```

交叉函数 function child =crossover2(pop,data,pcross)

```
[px,py]=size(pop);
child=ones(px,py);
for i=1:2:(px-1)
    if(rand<pcross)
        childpoint= randi([1,50]);
        cost_1 = pop(i,51)+pop(i+1,51);
        child(i,1:51) = [pop(i,1:childpoint) pop(i+1,childpoint+1:51)];
        child(i+1,1:51) = [pop(i+1,1:childpoint) pop(i,childpoint+1:51)];
        child(i,52) = fitness2(child(i,1:51),data);
        child(i+1,52) = fitness2(child(i+1,1:51),data);
        cost_2 = child(i,52)+child(i+1,52);
        if(cost_2>cost_1)
            child(i,:)=pop(i,:);
            child(i+1,:)=pop(i+1,:);
       end;
    end;
end;
```

成本计算函数 function cost=fitness2(point,data)

```
cost=0;

for i=1:469
    count=0;
    x_point=zeros(1,51);
    y_point=zeros(1,51);
    x=data(2*i-1,1:51);
    y=data(2*i,1:51);
    for j=1:51
        if(point(j)==1)
            x_point(count+1)=x(j);
            y_point(count+1)=y(j);
            count=count+1;
        end;
    end ;
    x0=[5.0:0.1:10.0];
    p2=interp1(x_point(1:count),y_point(1:count),x0,'spline');
    %p1=polyfit(x_point,y_point,3);
    %p2=polyval(p1,x);
    deta=p2-y;
```

```
    cost_1=0;
    for i = 1:51
        if abs(deta(i))<=0.5
           cost_1 = cost_1;
        elseif abs(deta(i))<=1
              cost_1 = cost_1+0.5;
        elseif abs(deta(i))<=2
              cost_1 = cost_1+1.5;
        elseif abs(deta(i))<=3
              cost_1 = cost_1+6;
    elseif abs(deta(i))<=5
              cost_1 = cost_1+12;
        else   cost_1 = cost_1+25;
        end;
    end;
    cost = cost + cost_1+count*12;
end;
cost= cost/469;
```