统计推断在数模转换系统中的应用

组号：43　　姓名：俞宗伯 5140309072　　孟正凌 5140309278

摘要：通过统计推断，计算传感器 x，y 之间的函数关系，提升生产良率

关键词：统计推断，遗传算法

1　引言

1.1 背景概述

某电子产品内部有一检测模块需要校准，但其输入输出特性呈明显三段且均为非线性。不同个体间虽曲线形态相同但两两相异，且中段的起始位置有随机性差异，如图 1。

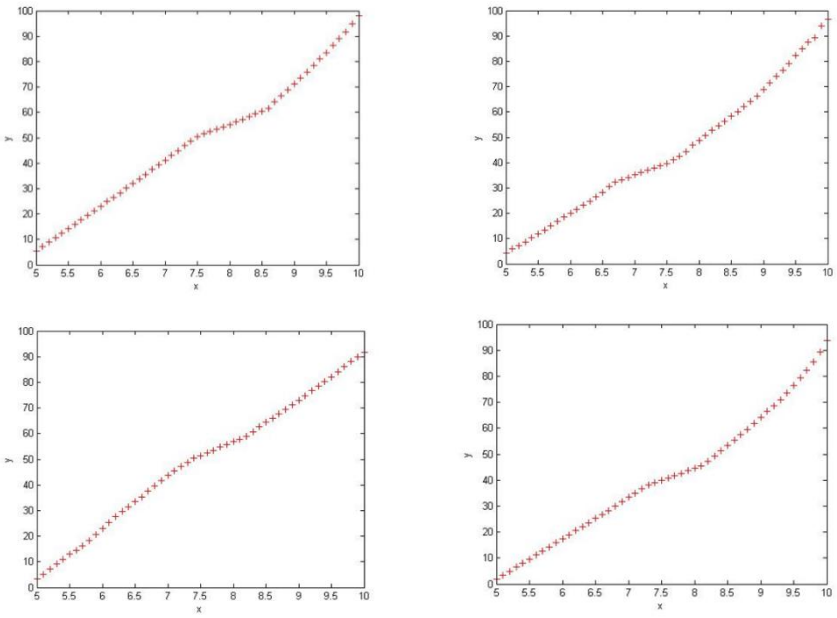

图 1

1.2 任务概述

此课程要求利用我们将过去学习统计的理论应用在工业上，求取个数据间的函数关系。工业上，产品必须要有一定的精度，因此多次测定是必须的，此次课程便是以传感器特性测定作为主题。而要从这些巨量的数据中，利用各种取点、拟合方案互相比较成本，选出较佳的方案便是此次课程的主要目的。（软件工具：matlab）

2 拟合和取点方案的取舍

2.1 拟合

是一种把现有数据透过数学方法来代入一条数式的表示方式。科学和工程 问题可以通过诸如采样、实验等方法获得若干离散的数据，根据这些数据，我们希望得到一个连续的函数（也就是曲线）或者更加密集的离散方程与已知数据相吻合。而其中拟合方法。常见的拟合目标函数形式有多项式(polynomial)、傅里叶函数(Fourier)、Weibull 分布、指数函数、对数函数、幂函数、高斯 函数 (Gauss)，

四个不同样本个体特性图示对比

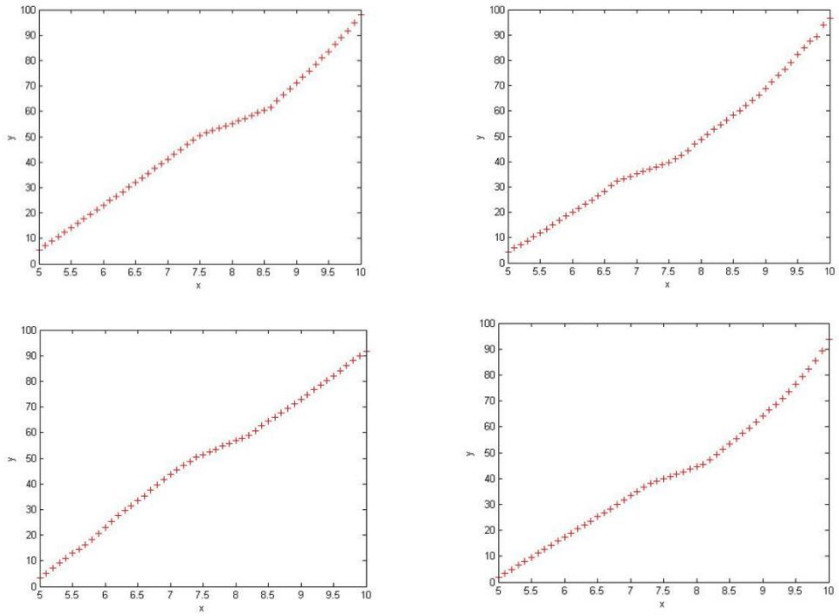

图 2

由图 2 中可以看出，其曲线特征为单调上升，大致可分为三段且有一定的弯曲度和两个较为明显的拐点，因此我们决定先用三次多项式拟合，当然这只是第一步而已，未来会再利用其他拟合方法如五次多项式拟合或是三次样条插值以比较出成本更低的方案。

2.2 取点

如何寻找最优化取点方式是有一定难度的，若是用穷举法将会浪费过多的时间且不切实际，因此我们将采用搜索性质的算法。我们在退火算法和遗传算法中做了不少比较:

2.2.1 模拟退火算法（Simulate Anneal Arithmetic，SAA）

SAA 是一种通用概率演算法，用来在一个大的搜寻空间内找寻命题的最优解。模拟退火算法是解决 TSP(Travelling Salesman Problem，即旅行商问题，其问题为一旅行商须经过 n 个城市，而限制他每个城市只许经过一次且必须回到原点城市，在这些限制下选择最佳路径)问题的有效方法之一。

模拟退火算法得名于金属退火的过程：将热力学的理论套用到统计学上，将搜寻空间内每一点想像成空气内的分子；分子的能量，就是它本身的动能；而搜寻空间内的每一点，也像空气分子一样带有“能量”，以表示该点对命题的合适程度。演算法先以搜寻空间内一个任意点作起始：每一步先选择一个“邻居”，然后再计算从现有位置到达“邻居”的概率。算法

而其优点如贪心法(专注于眼前最方便得到答案的方案，未综合评估整体效益，能够得出在局部的最佳解。) 一样，思想简单且可以求解复杂的非线性优化问题，但其缺点为收敛速度过慢，且参数难控制。

2.2.2 遗传算法(Genetic Algorithm，GA)

GA 的实现过程便如同大自然运作一般。以袋鼠为例，首先寻找一种对问题潜在解进行“数字化”编码的方案。(找出表现和基因的映射关系。)然后用随机数初始化一个种群（那么第一批袋鼠就随机地分散在山脉上。)，种群里面的个体就是这些数字化的编码。接下来，通过解码过程之后，(得到袋鼠的位置坐标。)用适应性函数对每一个基因个体作一次适应度评估。(袋鼠爬得越高，越是受我们的喜爱，所以适应度相应越高。)用选择函数按照某种规定择优选择。(每隔一段时间，位置座标踏低的袋鼠会被干掉，以保证袋鼠总体数目持平。)让个体基因交叉变异。(让袋鼠随机地跳一跳）然后产生子代。(希望存活下来的袋鼠是多产的，并在那里生儿育女。）遗传算法并不保证你能获得问题的最优解，但是使用遗传算法的最大优点在于你不必去了解和操心如何 去“找”最优解。(你不必去指导袋鼠向那边跳，跳多远。）而只要简单的“否定”一些表现不好的个体就行了。请看流程图（图 3）

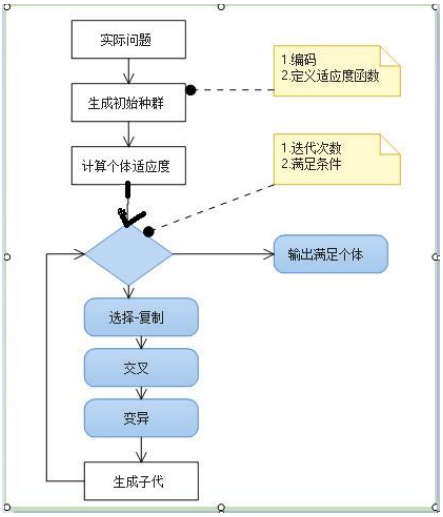

图 3

遗传算法种种优点如:

一,其搜索能力与问题领域无关且快速随机

二,从群体里搜索，具有并行性且可以一次比较多个个体

三,使用机率制迭代，具随机性

遗传算法种种缺点如:

参数的偏差可能带来较大的影响，这是退火算法也具有的；过早收敛，是可以采用一些方法例如增加种群多样性来尽可能避免的。

综合比较两种算法的优劣后我们决定采用遗传算法。

3.遗传算法

使用遗传算法时应注意尽可能避免过早收敛的情况。图 4 给出了可能导致过早收敛的分布示意图。

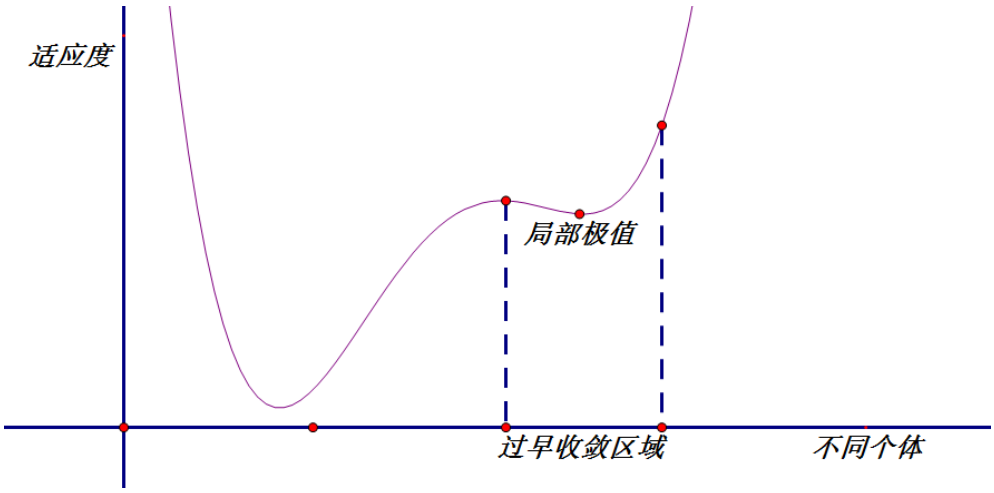

图 4
可能迭代时导致发生过早收敛的函数分布举例

3.1 初始设定

首先定义了一个 pop_size×individual_size 的矩阵。其中 pop_size 表示种群大小，即种群个体数，individual_size 表示个体大小，即个体长度。由于我们采用 0,1 矩阵表示是否取某个点，因此在本题中，个体长度是确定的，为 51，即给定的每个器件的测量点数，而种群个体数 pop_size 我们设定为 100.

3.2 种群的初始化

若实际情况具前文提到的函数极值分布情况，且基因多样性不足导致种群个体集中分布于较小的收敛域中，则可能造成过早收敛，最后只能求得局部最优解而无法取得全局最优解。

若一开始种群初始化时直接对每个点使用 randi()函数，会导致每个个体取点个数在一半左右，就可能产生基因多样性不足，从而产生过早收敛的风险。因此为避免这种情况，在种群初始化时，我们设定种群初始大小为 100，并将这 100 个个体分为 8 个等级，每个等级对于每个点取或不取的概率从 10%至 80%不等，这样可增加种群的基因丰富度。同时注意到三次多项式拟合至少需要取 4 个点，因此当取点不足 4 个时，重新取点。

3.3 结束判定

由于并没有一个标准来确定个体是否是我们想要的最优解，因此我们没有采用条件判断来停止遗传，而是设定一个遗传的代数，当代数达到时停止遗传，再在所有代中找到出现过的最优个体记录下来。

3.4 生存判定

我们尝试了两种种群更替的方式：

3.4.1 根据每一个个体的适应情况赋予其生存概率

由于适应度（即定标成本）越高，则个体越差，因此生存概率的函数应是随适应度增加而递减的。区别于直接使用反比例函数，为了使优劣个体的差别更明显，我们将生存概率函数设定如下（即反比例函数向右平移一定距离）：

我们设定了一个生存判定函数：$p = \frac{k_1}{x - k_2}$ k1，k2 为控制种群总体存活率的参数。如图 5。

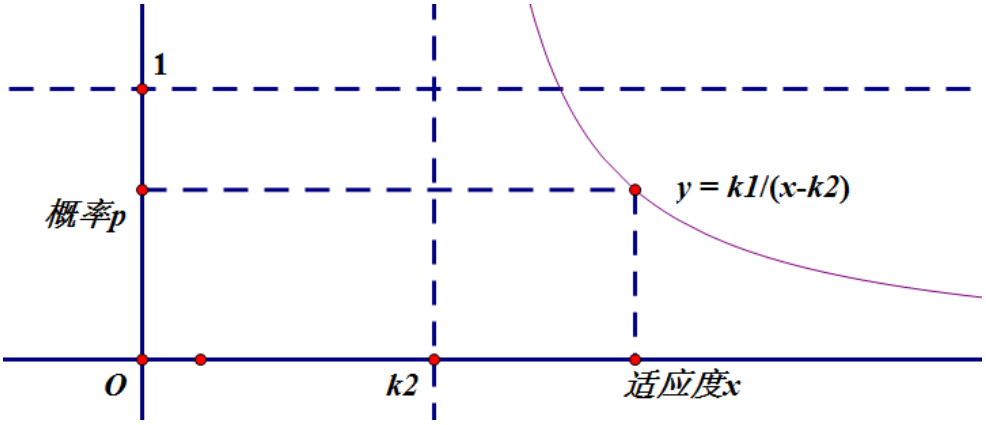

图 5

优点：由于生存概率的设定，使得即使是一代中最差的个体也有概率存活下来，基因多样性得以保留。

缺点：难以控制每一代中存活个体的数目，若该数目过低，容易导致种群个体基因趋于相同，进而产生过早收敛的情况。

3.4.2 对于每一代的个体，记录每一个个体的适应度，直接淘汰适应度低的一半个体，保留剩下一半适应度高的个体。

优点：每一次选择保留个体的比例易控制。

缺点：可能在初期淘汰掉一些“潜力股”（即那些携带优良基因但由于整体表现较差而被淘汰的个体）。

综合比较二者的优劣后我们决定采用第二种方式。

3.5 基因重组

我们定义了一个表示当前种群的个体是否存活的 0,1 矩阵，存活的个体参与杂交，原先淘汰掉的个体的位置由杂交生成的新个体取代，成为新的种群，但种群大小仍不变。

3.6 变异

在新一代的种群产生时，遍历每一个个体的每一个基因，随机数小于设定的变异概率时，变异过程发生，产生一些新的取点方式，为可能已经收敛了的种群注入新鲜活力，为得到更优解带来可能性。

4. 遗传结果

4.1 选用拟合方式为三次多项式插值

4.1.1 逐代最优个体适应度变化折线图, 如图 6

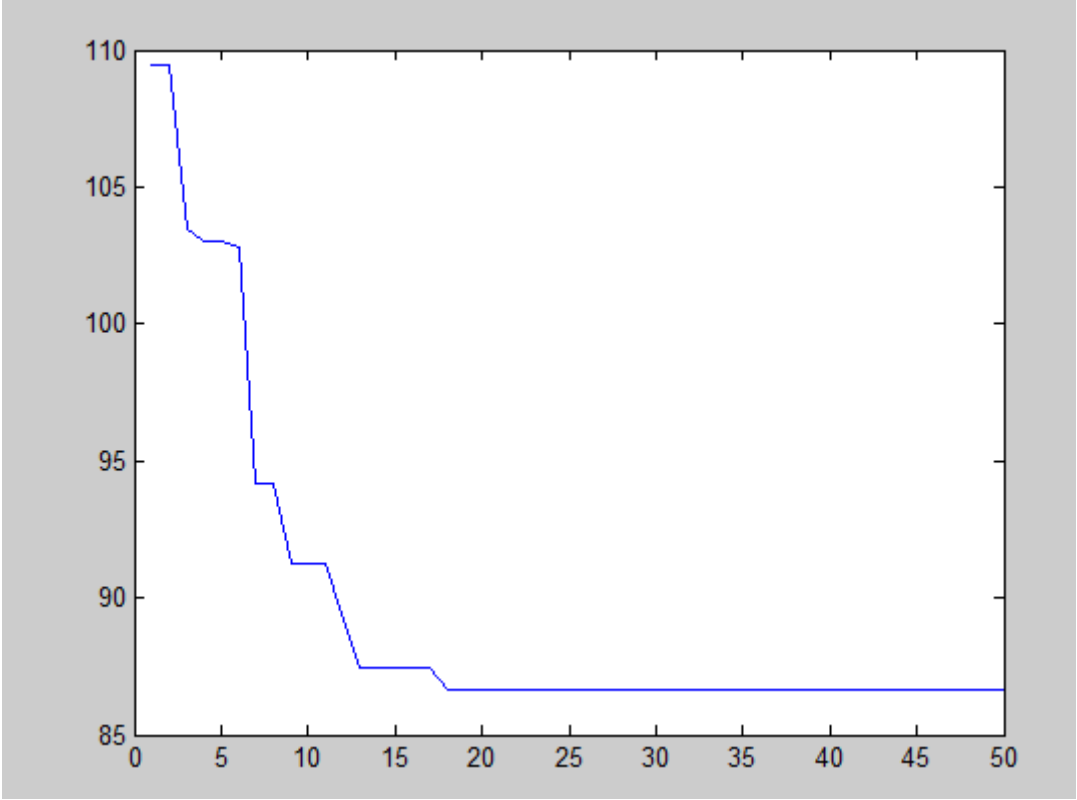

图 6

4.1.2 最终得到的最优取点方式及其成本

取点方式为[2, 14, 25, 35, 48]

所需成本为 86.6700

4.2 选用拟合方式为三次样条插值

4.1.1 逐代最优个体适应度变化折线图,如图 7。

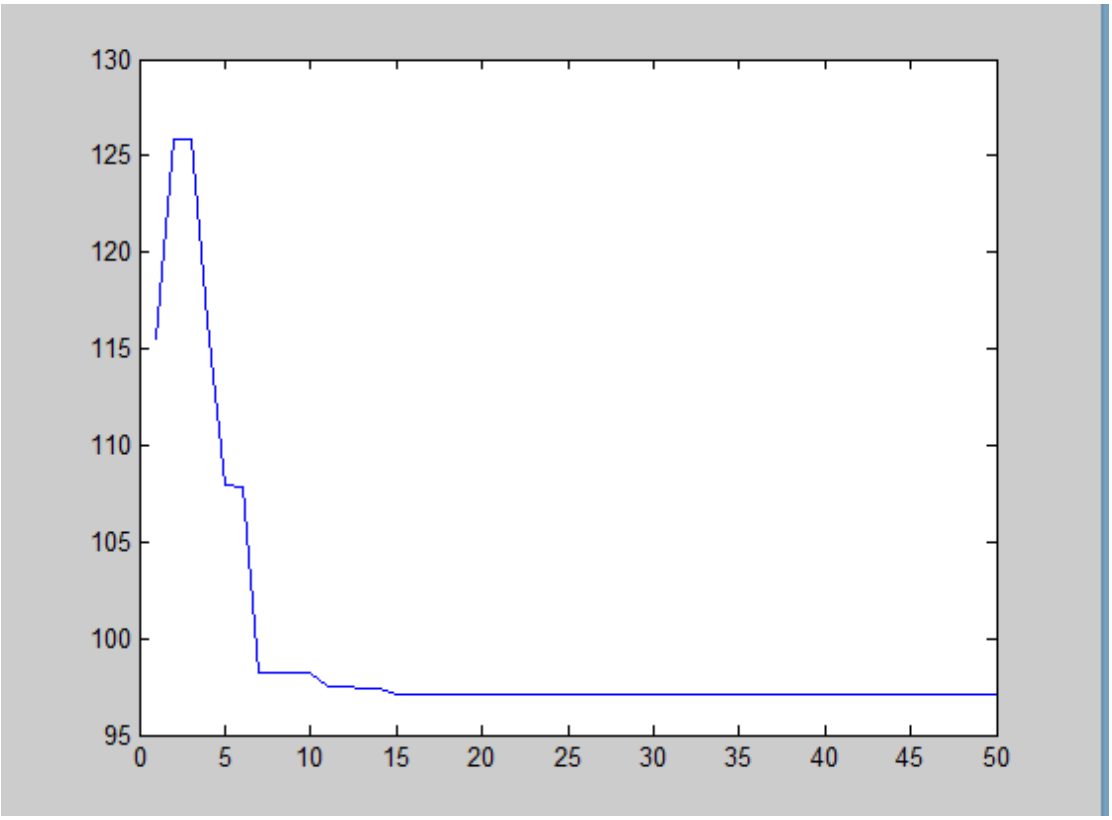

图 7

4.2.2 最终得到的最优取点方式及其成本

取点方式为[1,9,20,26,34,45,51]

所需成本为 97.1010

4.3 比较两种插值方式的结果

最后采用三次多项式插值作为拟合方式

5 结论

本课题是以某投入批量生产的电子产品的检测模块进行成本合理的传感气校准为导向。因数量的庞大和每个传感器个体之差异，使得穷举法变的不切实际。因此我们借助启发式搜索算法——遗传算法。通过遗传算法的拟合，我们有以下结论：在对传感器定标时选取点 的位置为 2,14,25,35,48 并采用三次多项式插值作为拟合方法时可以得到平均最低定标成本 86.6700。

附录：遗传算法 matlab 代码(前面为 test_ur_answer 函数)

test_ur_answer.m

```
%%%%%%% 答案检验程序 2015-11-04 %%%%%%%%

my_answer=[2,14,25,35,48];%把你的选点组合填写在此
my_answer_n=size(my_answer,2);

% 标准样本原始数据读入
minput=dlmread('20150915dataform.csv');
[M,N]=size(minput);
nsample=M/2; npoint=N;
x=zeros(nsample,npoint);
y0=zeros(nsample,npoint);
y1=zeros(nsample,npoint);
for i=1:nsample
    x(i,:)=minput(2*i-1,:);
    y0(i,:)=minput(2*i,:);
end
my_answer_gene=zeros(1,npoint);
my_answer_gene(my_answer)=1;


% 定标计算
index_temp=logical(my_answer_gene);
x_optimal=x(:,index_temp);
y0_optimal=y0(:,index_temp);
for j=1:nsample
    % 请把你的定标计算方法写入函数 mycurvefitting
    y1(j,:)=mycurvefitting(x_optimal(j,:),y0_optimal(j,:));
end

% 成本计算
Q=12;
errabs=abs(y0-y1);

le0_4=(errabs<=0.4);
le0_6=(errabs<=0.6);
```

```
le0_8=(errabs<=0.8);
le1_0=(errabs<=1);
le2_0=(errabs<=2);
le3_0=(errabs<=3);
le5_0=(errabs<=5);
g5_0=(errabs>5);

sij=0.1*(le0_6-le0_4)+0.7*(le0_8-le0_6)+0.9*(le1_0-le0_8)+1.5*(le2_0-
le1_0)+6*(le3_0-le2_0)+12*(le5_0-le3_0)+25*g5_0;
si=sum(sij,2)+Q*ones(nsample,1)*my_answer_n;
cost=sum(si)/nsample;

% 显示结果
fprintf('\n 经计算，你的答案对应的总体成本为%5.2f\n',cost);
```

mycurvefitting.m

```
function y1 = mycurvefitting( x_premea,y0_premea )

x=[5.0:0.1:10.0];

% 将你的定标计算方法写成指令代码，以下样式仅供参考
y1=interp1(x_premea,y0_premea,x,'pchip');



end
```

(以下为遗传算法代码)

main.m

```
clear;
clc;
global data;
data=csvread('20150915dataform.csv');
global exponent;
exponent=3;          %拟合方式次数
global pop_size;
pop_size=100;         %种群大小，即个体数
global individual_size;
individual_size=51;   %个体基因长度
```

```
global population;          %当前种群
population=zeros(pop_size,individual_size);
global survive;      %记录当前种群的个体是否存活
survive=zeros(pop_size,1);
global best_indi;           %整个过程中的最优个体
global best_indi_generation; %最优个体出现在哪一代
global best_fitness;        %最佳适应度
best_indi=zeros(1,individual_size);
best_indi_generation=0;
best_fitness=1e+5;
generation=50;     %遗传总代数
global generation_num; %当前代数

for generation_num=1:generation
   initialize();
   select();
   cross();
end
best_way=find(best_indi);
```

initialize.m

```
function [] = initialize1()
%INITIALIZE 种群的初始化
   global exponent;
   global population;
   global pop_size;
   global individual_size;
   tmp_pop=randi([0 9],pop_size,individual_size);
   for m=1:pop_size
          while length(find(tmp_pop(m,:)==9))<(exponent+1)    %避免出现取
点过少无法拟合的情况
                tmp_pop(m,:)=randi([0 9],1,individual_size);
          end
   end
   for k1=1:pop_size
       index=find(tmp_pop(k1,:)>(k1/13+1));    %取点的概率从 80%递减至 10%
       population(k1,:)=0;
       population(k1,index)=1;
```

```
   end
end
```

evaluate.m

```
function [ cost ] = evaluate(x)
%DEVIATION 误差成本
%   x 是取点方式向量
global individual_size;
global pop_size;
global data;
global best_fitness
global best_indi;
global best_indi_generation;
global generation_num;
cost=0;
for k1=1:400
    devicost=0;
    value=interp1(data(2*k1-1,x),data(2*k1,x),data(1,:),'pchip');
    for k2=1:individual_size
        devi=abs(value(k2)-data(2*k1,k2));
        if devi<=0.4 devicost=devicost+0;
        elseif devi<=0.6 devicost=devicost+0.1;
        elseif devi<=0.8 devicost=devicost+0.7;
        elseif devi<=1 devicost=devicost+0.9;
        elseif devi<=2 devicost=devicost+1.5;
        elseif devi<=3 devicost=devicost+6;
        elseif devi<=5 devicost=devicost+12;
        else  devicost=devicost+25;
        end
    end
    cost=cost+12*length(x)+devicost;
end
cost=cost/400;
if cost<best_fitness;
     best_fitness=cost;
     indi=zeros(1,individual_size);
     indi(x)=1;
```

```
        best_indi=indi;
        best_indi_generation=generation_num;
    end
end
```

p_survive.m

```
function [ p_s ] = p_survive( x )
%P_SURVIVE 生存下来的概率
%   参数为某种取点方式
    global exponent;
    para=40;  %控制整体存活率的一个系数
    global data;
    %a=evaluate(x)-12*(exponent+1);    %平均成本肯定在 12*(exponent+1)以上，
因为至少取 exponent+1 个点拟合
    a=evaluate(x)-50;
    p_s=para/a;
    %这里可以增加一个精英选择的判断，即某一代最优个体存活概率为 1
end
```

select.m

```
function [] = select()
    global population;
    global pop_size;
    global survive;
    fitness=zeros(pop_size,1);

    for k=1:pop_size
        tmp1=find(population(k,:));
        fitness(k,1)=evaluate(tmp1);
    end
    tmp2=fitness;
    tmp2=sort(tmp2);
    threshold=tmp2(pop_size/2);
    survive=zeros(pop_size,1);
    for k=1:pop_size
        if fitness(k)<=threshold
            survive(k)=1;
        else
```

```
            survive(k)=0;
        end
    end
end
```

cross.m

```
function [] = cross()
%CROSS 杂交函数
    global survive;
    global individual_size;
    global population;
    index_new_indi=find(survive==0);          %新的个体替换原来被淘汰的个体
    index_old_indi=find(survive==1);
    for m=1:length(index_new_indi)
        k=index_new_indi(m);
        a=index_old_indi(randperm(length(index_old_indi)));
        b=a(1:2);
        parent1=population(b(1),:);
        parent2=population(b(2),:);
        cut=randi([10 41]);         %基因重组时的断点

population(k,:)=[parent1(1:cut),parent2((cut+1):individual_size)];
    end
end
```

best_display.m

```
function [ output_args ] = best_display( input_args )
global displ;
figure,plot(displ);

end
```