

统计推断在数模转换系统中的应用

杨一凡 5140309202 周秉炆 5140309460

2015 年 12 月 2 日

摘要 本课题运用统计推断的方法为某种产品的校准提供优化解决方案。以降低成本及提高精度为原则，综合多项式拟合、埃尔米特插值、遗传算法、随机函数等进行特征点的选择，并对提供的某批数据提出了解决方案。

关键词 MATLAB、Origin、多项式拟合、埃尔米特插值、遗传算法

Statistic Application On The AD-DA Converting System

ABSTRACT In this paper, we use the method of statistical inference to provide a solution for the calibration of a product. Take reducing cost and improving accuracy as principle, comprehensively using polynomial fitting, Hermite Interpolation, Genetic Algorithm, random function, and so on to select the feature points, and propose the solution of the given datas.

KEY WORDS MATLAB、Origin、Polynomial fitting、Hermite Interpolation、Genetic Algorithm

1 引言

假定将有某投入批量生产的电子产品，其内部有一个模块，功能是监测某项与外部环境有关的物理量（可能是温度、压力、光强等）。该监测模块中传感器部件的输入输出特性呈明显的非线性。本课题要求为该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。

由于提供的数据极多（400 组数据，每组 51 个点），因此我们不可能对每个数据来拟合从而寻找最佳的对应曲线。所以在我们选择了合适的拟合方法后，将采用遗传算法来筛选出较为合适的数据点进行定标。

2 建模

2.1 原理

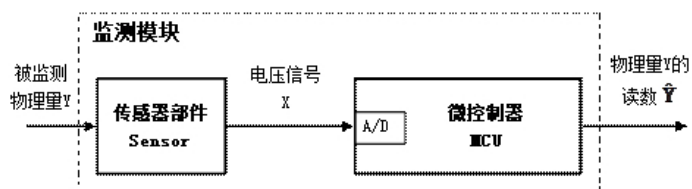


图 1: 监测模块的组成框图

监测模块的组成框图如图 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）。

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。

——引用自本课程讲义

2.2 整体数据观察

首先将数据通过 Origin 作图，如图 2。可以观察到，在 X 坐标大约 6 至 8 的位置，函数的曲线有明显的变化。接下来选择合适的拟合方式。

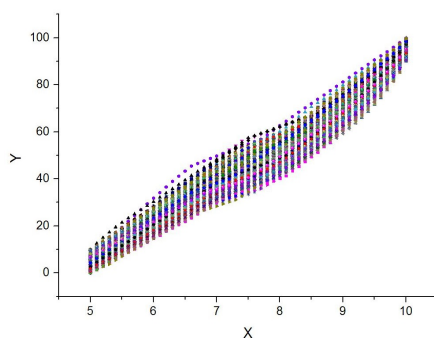


图 2: 原始数据

3 拟合方式的确定

3.1 多项式拟合

由于需求得的 $\hat{y} = f(x)$ 需要尽可能光滑，因此我们选择多项式拟合。而考虑到龙格现象，我们选择对所有数据的平均数进行 3 次拟合，拟合结果如图 3。观察到，X 坐标 6 至 8 处有曲线与数据点有明显的偏离。因此我们需要对函数的拟合进行分段。

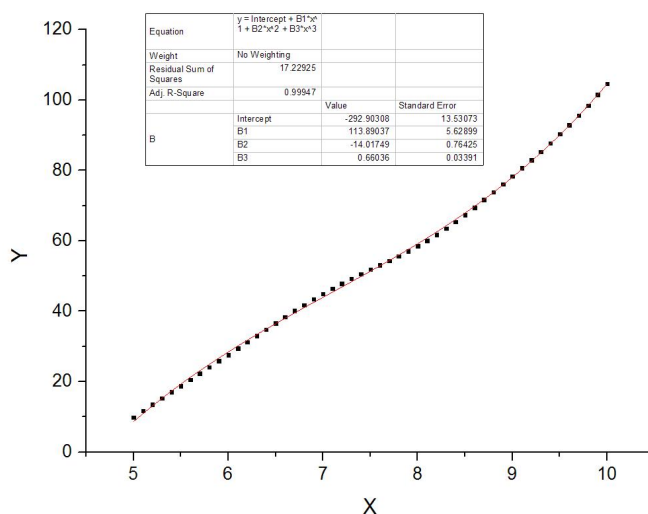


图 3: 多项式拟合

然而，若进行分段，则必须选择大量的数据点（每一段的三次拟合需要至少 4 个点）。在该产品的定标过程中，单点测定的成本较高，因此分段拟合也不适用于本方案。

3.2 Hermite 插值

3.2.1 原理

插值是指在离散数据的基础上补上连续函数，使得这条曲线经过所有给定的数据点。利用它可通过函数在有限个点处的取值状况，估算出函数在其他点处的近似值。

许多实际插值问题中，为使插值函数能更好地和原来的函数重合，不但要求二者在节点上函数值相等，而且还要求相切，对应的导数值也相等，甚至要求高阶导数也相等。这类插值称作 Hermite 插值。满足这种要求的插

值多项式就是 Hermite 多项式。由于 MATLAB 中有关于 Hermite 插值的相关函数，在此不再复述插值的计算方式。

由于插值的所得的函数必定经过每个数据点，因此能较好地降低原数据点定标的成本。

3.2.2 MATLAB 实现

如图 6, 利用 MATLAB 中的 pchip 函数, 得到如下的结果。此外 Hermite 插值相比较多项式拟合的优点在于, 若选出的数据点较少, 则无需如多项式拟合那样考虑分段。

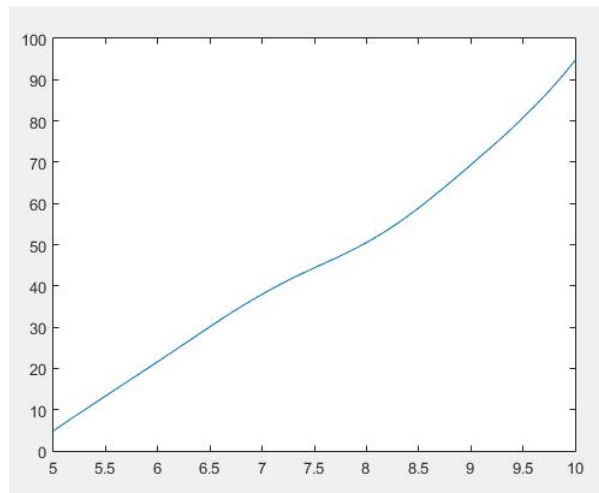


图 4: Hermite 插值

因此, 本方案将采用 Hermite 插值进行拟合, 并使用遗传算法来选出合适的数据点。

4 遗传算法

4.1 介绍及优点

遗传算法是模仿自然界生物进化机制发展起来的随机全局搜索和优化方法, 它借鉴了达尔文的进化论和孟德尔的遗传学说。其本质是一种高效、并行、全局搜索的方法, 它能在搜索过程中自动获取和积累有关搜索空间的知识, 并自适应的控制搜索过程以求得最优解。遗传算法操作使用适者生存的原则, 在潜在的解决方案种群中逐次产生一个近似最优解的方案, 在遗传算法的每一代中, 根据个体在问题域中的适应度值和从自然遗传学中借鉴

来的再造方法进行个体选择，产生一个新的近似解。这个过程导致种群中个体的进化，得到的新个体比原来个体更能适应环境，就像自然界中的改造一样。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。进化算法最初是借鉴了进化生物学中的一些现象而发展起来的，这些现象包括遗传、突变、自然选择以及杂交等。

——引用自中文维基 <https://zh.wikipedia.org/wiki/遗传算法>

4.2 流程图

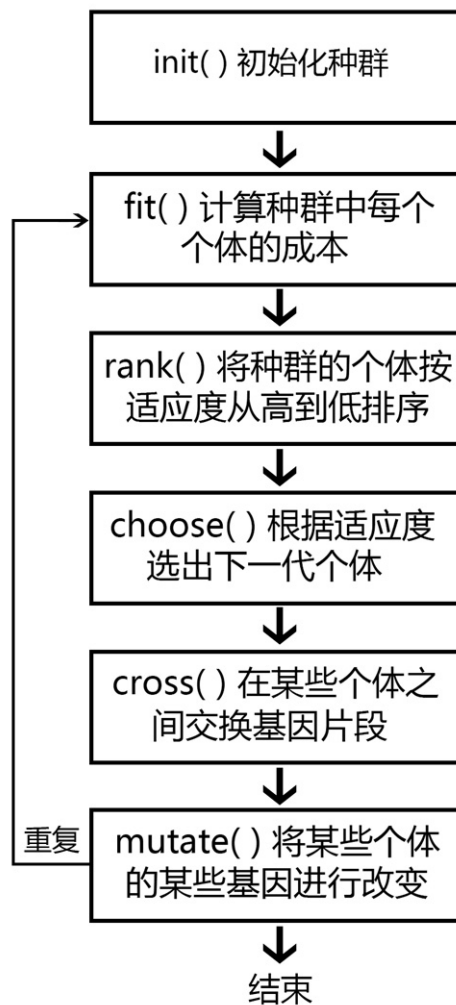


图 5: 流程图

4.3 具体算法

4.3.1 编码与解码

编码 要实现遗传算法首先要弄清楚如何对求解问题的过程进行编码和解码。本方案将采用二进制编码，二进制编码具有稳定性高、种群多样性大等优点，但是需要的存储空间大，需要解码过程并且难以理解。对于一个个体的基因，我们采用 51 位二进制数，分别代表对应位的数据点是否使用，0 表示不使用，1 表示使用。

解码 根据输入的二进制代码，通过某些运算，将数据选出，进行拟合，计算成本。

4.3.2 初始化种群

生成 51 位全为 0 的向量，之后将其中随机的位变为 1。考虑到数据的规律性较强，我们初始点的选取个数为随机 4 至 16 个。

4.3.3 选择操作

该操作即从前代种群中选择个体到下一代种群的过程。一般是用个体适应度的分布来选择个体，每个个体按照其适应度占据一个轮盘的部分面积。随机转动一下转盘，当转盘停止转动时，指针所指向的个体即为被选中。由于这个过程中可能将适应度最高的个体排除，因此我们采用精英优势，即每一代适应度最高的个体以及第二高的个体将强制保留至下一代。

4.3.4 交叉操作

标准的交叉操作是对相邻两个个体进行的。在本方案中是对任意两个个体进行的，因为我们仅仅需要成本最低的那个个体，而不需要整个种群的适应度高（即成本低），因此，对任意两个个体进行交叉可的的以有效提高种群的活力（即更高的获得更好个体的可能性）。同时，为了防止最优个体的基因被污染（即与较差的个体进行了交换），该算法中不对适应度最高的个体进行交叉操作。随机选择两个个体，然后按交叉概率对这两个个体进行交叉。如果需要进行交叉，再随机选择交叉位置，将交叉位置以后的二进制串进行对换。

4.3.5 变异操作

根据生物学，遗传时不但有交换染色体，还有变异的过程。标准的变异操作是仅改变一个基因，然而我们可以遇见到：最终成本较低的个体，其选

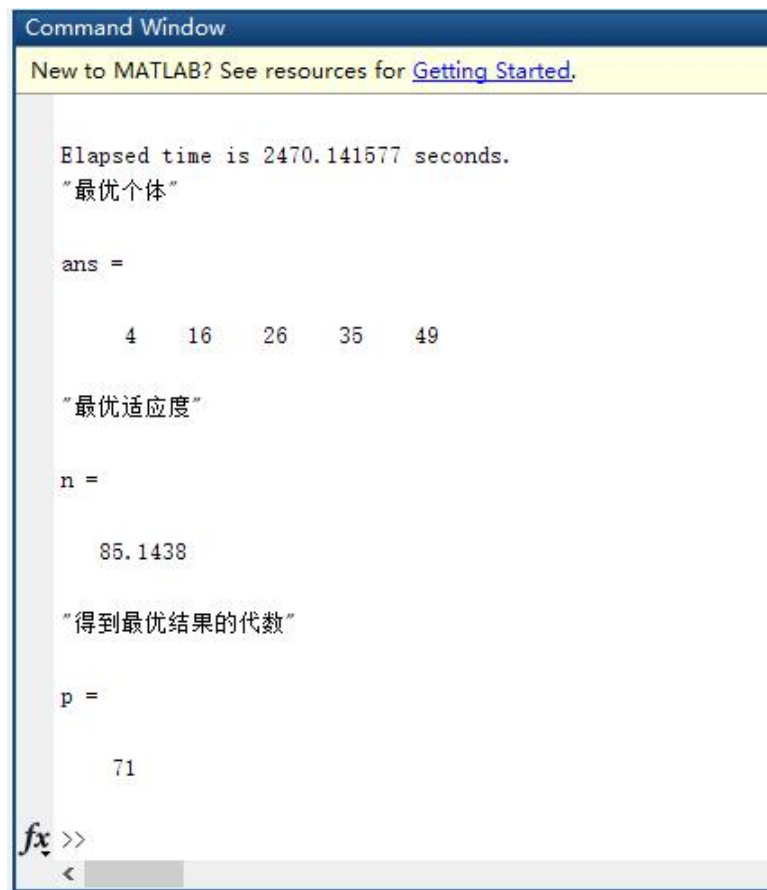
择数据点的数量是一定的，因此若只是改变一个基因，难以起到可能的有益变异的效果。所以在本方案中，变异操作是选择相邻的两个基因同时进行改变。取一个较小的变异概率，若一个个体需要进行变异。则在一个随机位置对其及部分相邻的二进制数字进行改变。

4.4 MATLAB 实现

代码见附录

4.5 计算结果

执行的主函数为：GeneticAlgorithm(250,51,100,0.5,0.01)，即种群大小 250，基因长度 51，遗传 100 代，0.5 的交叉概率，0.01 的变异概率。结果如图 6，平均成本变化如图 7。



```
Command Window
New to MATLAB? See resources for Getting Started.

Elapsed time is 2470.141577 seconds.
"最优个体"

ans =

    4    16    26    35    49

"最优适应度"

n =

    85.1438

"得到最优结果的代数"

p =

    71

fx >>
<
```

图 6: 计算结果

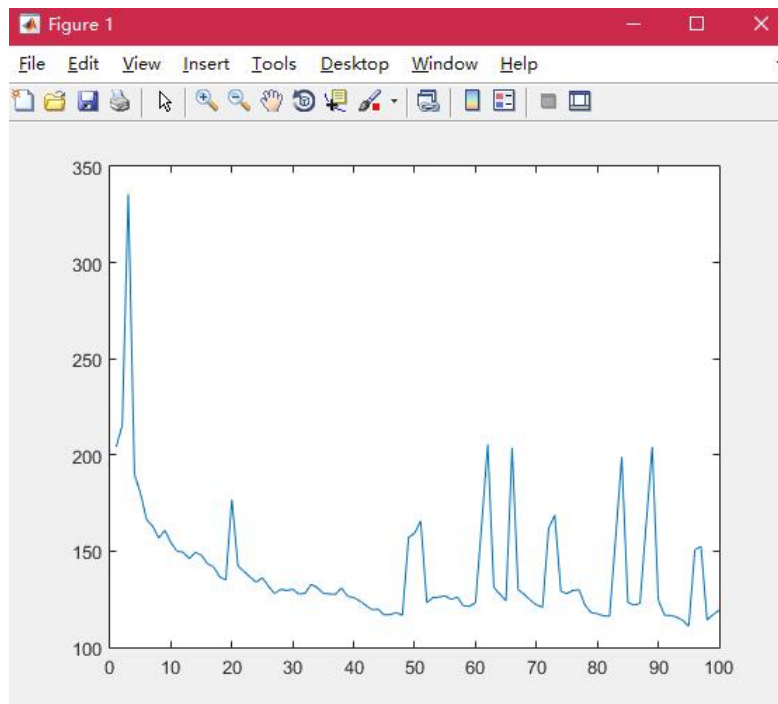


图 7: 每一代平均成本

可以发现,开始时,最优成本下降较快,当达到 50 代左右时,平均值开始在小范围变化。说明此时成本已经达到相对最优解,之后若想获得更低的成本,只能依靠较差个体的变异来获得。由于为了反应真实的遗传状况,变异概率必须取得相对较低。这也是遗传算法的不足之一,只能取得在某个范围内的相对最优解,而不是整体的绝对最优解。不过相对于精确计算的时间成本,遗传算法已经是综合考虑了时间成本的情况下的较好方案了。

4.6 结论

最终,本方案选取的数据点为第 4、16、26、35、49 个,即 X 取 5.3、6.5、7.5、8.4、9.8。所得的成本为 85.1438。

5 感想

在接触这门课之前，我们从未接触过 MATLAB。在有些赶工的学习中，我们还是将 MATLAB 的代码部分解决，尽管某些语句可能有些累赘、重复，不过这些已经是我们尽自己的全力一行行码下来，一遍遍测试、修改的了。同时也要感谢各位老师，老师在上课时的讲解以及面谈时和我们一对一的讨论都启发了我们很多，让我们少走了许多弯路。

虽然这门课花费了我们很多时间，但现在回看起来，这都是值得的。数学方法在工程方面的应用是有很大的现实意义，有时候绝对的最优解并不适合实际，需要综合考虑时间和成本两方面。同时，在 MATLAB 的编写方面我们还有待改进，我们并没有充分利用 MATLAB 矩阵运算的优点，造成了一定的时间上的浪费。

最后，在此感谢每一位对我们提供了帮助和启迪的同学和老师。

6 附录

6.1 MATLAB 代码

6.1.1 函数入口

```
global x;
global y;

%数据准备
data=xlsread('20150915dataform.csv','A1:AY800');
y=zeros(400,51);
x=data(1,:);
for i=1:400
    y(i,:)=data(2*i,:);
end

tic
[m,n,p]=GeneticAlgorithm(250,51,100,0.6,0.01);
toc
disp "最优个体"
find(m)
disp "最优适应度"
n
disp "得到最优结果的代数"
p
```

6.1.2 主函数

```
%主函数
%主函数
function [m,n,p]=GeneticAlgorithm(pop_size,chromo_size
    ,generation_size,cross_rate,mutate_rate)

global G ; %遗传代数
global fit_value;%当前代适应度矩阵
global best_fit;%最佳适应度
```

```

global fit_plot;%各代最佳适应值
global best_individual;%最佳个体
global best_generation;%最佳代数

G=0;
fit_plot=zeros(generation_size,1);

fit_value(pop_size)=0.;
best_fit=99999.9;
best_generation=0;
init(pop_size,chromo_size);%初始化
for G=1:generation_size
    fit(pop_size,chromo_size); %计算适应度
    rank(pop_size,chromo_size); %对个体按适应度大小进行排序
    choose(pop_size,chromo_size);%选择操作
    cross(pop_size,chromo_size,cross_rate);%交叉操作
    mutate(pop_size,chromo_size,mutate_rate);%变异操作

    %显示每一代过程中的最优适应度
    G
    best_fit
end
plotG(generation_size); %打印算法迭代过程
m=best_individual; %获得最佳个体
n=best_fit; %获得最佳适应度
p=best_generation; %获得最佳个体出现代

clear i;
clear j;

```

6.1.3 初始化函数

```

%初始化种群
%pop_size: 种群大小
%chromo_size: 染色体长度

```

```
function init (pop_size , chromo_size)
global pop;

pop=zeros (pop_size , chromo_size);
for i=1:pop_size
    r=randperm ( chromo_size );
    for j=1:round (12*rand+4)
        %由于数据规律性，初始点选取4至16个
        pop(i , r(j))=1;%随机进行初始化二进制数
    end
end

clear i;
clear j;
clear r;
```

6.1.4 适应度函数

```
%计算种群个体适应度
%pop_size: 种群大小
%chromo_size: 染色体长度

function fit (pop_size , chromo_size)
global fit_value;
global pop;

%解码由成本计算函数完成
for i=1:pop_size
    fit_value(i)=mycost (pop(i , :) , chromo_size);
end

clear i;
clear j;
```

6.1.5 成本计算函数

%根据输入的二进制向量解码并计算成本

```
function [p]=mycost(t, chromo_size)
global x;
global y;
```

```
p=0;
m=find(t);
n=length(m);
```

%若取的数据点不多于3个则判定为无效

```
if n<=3
    p=999;
    return;
end
```

```
u=zeros(1,n);
v=zeros(400,n);
vv=zeros(400,chromo_size);
```

%解码过程

```
for i=1:n
    u(1,i)=4.9+0.1*m(1,i); %对X的解码
    for j=1:400
        v(j,i)=y(j,m(1,i)); %对Y的解码
    end
end
```

%拟合并判断各点成本

```
for i=1:400
    vv(i,:)=pchip(u,v(i,:),x); %Hermite插值
    for j=1:chromo_size
        tmp=abs(y(i,j)-vv(i,j));
        p=p+0.1*(tmp>0.4)+0.6*(tmp>0.6)+0.2*(tmp>0.8)
            +0.6*(tmp>1)+4.5*(tmp>2)+6*(tmp>3)+13*(tmp
```

```
                >5);  
            end  
end  
p=p/400+12*n;  
  
clear i;  
clear j;  
clear n;  
clear m;
```

6.1.6 选择函数

```
%选择操作  
%pop_size: 种群大小  
%chromo_size: 染色体长度  
  
function choose(pop_size,chromo_size)  
global pop;  
global fit_value;  
  
pop_new=zeros(pop_size-1,chromo_size);  
%轮盘操作  
chance=zeros(1,pop_size);  
disk=zeros(1,pop_size);  
uu=fit_value(pop_size);  
  
%将成本最高的值减去个体成本作为每个个体的适应度  
for i=1:pop_size  
    chance(i)=uu-fit_value(i);  
end  
disk(1)=chance(1);  
for i=2:pop_size  
    disk(i)=disk(i-1)+chance(i);  
end  
  
%保留成本最低的两个个体
```

```
for i=1:pop_size-2
    idx=1;
    r=rand*disk(pop_size);
    while disk(idx)<r
        idx=idx+1;
    end;
    for j=1:chromo_size
        pop_new(i,j)=pop(idx,j);
    end
end
```

%将选中的个体代替上一代

```
for i=3:pop_size
    for j=1:chromo_size
        pop(i,j) = pop_new(i-2,j);
    end
end
```

```
clear i;
clear j;
clear k;
clear pop_new;
clear idx;
clear m;
```

6.1.7 排序函数

%对个体按适应度大小进行排序，并且保存最佳个体

%pop_size: 种群大小

%chromo_size: 染色体长度

```
function rank(pop_size, chromo_size)
global fit_value; %个体适应度
global fit_plot; %适应度作图
global best_fit; %最佳适应度
global best_individual; %最佳个体
```

```
global best_generation; %最佳代数
global pop;
global G; %G指遗传代数

tmp(chromo_size)=0;

%按照从小至大排序
for i=1:pop_size
    min = i;
    for j = i+1:pop_size
        if fit_value(j)<fit_value(min);
            min = j;
        end
    end
    if min~=i
        temp = fit_value(i);
        fit_value(i) = fit_value(min);
        fit_value(min) = temp;
        for k = 1:chromo_size
            tmp(k) = pop(i,k);
            pop(i,k) = pop(min,k);
            pop(min,k) = tmp(k);
        end
    end
end

end

%计算最佳适应度，用于作图
fit_plot(G)=0;
for i=1:pop_size
    fit_plot(G)=fit_plot(G)+fit_value(i)/pop_size;
end

%选取精英个体
if fit_value(1) < best_fit
    best_fit = fit_value(1);
```



```
        best_generation = G;
        for j=1:chromo_size
            best_individual(j) = pop(1,j);
        end
    end

    clear i;
    clear j;
    clear k;
    clear min;
    clear temp;
    clear tmp;
```

6.1.8 交叉函数

```
%交叉操作
%pop_size: 种群大小
%chromo_size: 染色体长度
%cross_rate: 交叉概率

function cross(pop_size,chromo_size,cross_rate)
global pop;

i=2;
while i<pop_size
    if(rand<cross_rate)
        u=round(rand*2*(pop_size-i)/3);%防止优秀个体与
            过于差的个体交叉
        if u==0
            continue;
        end
        cp=round(rand*chromo_size);
        if (cp==0||cp==1||cp==pop_size || cp==pop_size
            -1)
            continue;
```

```

        end

        %若交换的基因片段相同则重新进行交叉
        if pop(i,cp:chromo_size)==pop(i+u,cp:
            chromo_size)
            continue;
        end

        %交换基因片段
        for j=cp:chromo_size
            temp=pop(i,j);
            pop(i,j)=pop(i+u,j);
            pop(i+u,j)=temp;
        end
    end
    i=i+1;
end

clear i;
clear j;
clear temp;
clear cp;

```

6.1.9 变异函数

```

%变异操作
%pop_size: 种群大小
%chromo_size: 染色体长度
%mutate_rate: 变异概率
function mutate(pop_size, chromo_size, mutate_rate)
global pop;

for i=2:pop_size
    if rand<mutate_rate

```

```

        %随机选择一个基因
        mutate_pos=randperm(chromo_size);

        for j=1:2
            pop(i,mutate_pos(j))=1-pop(i,mutate_pos(j));
        end
    end
end

clear i;
clear mutate_pos;

```

6.1.10 成本测试函数

```

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

my_answer=[ 4,16,26,35,49 ];%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);

```

6.1.11 mycurvefitting

```
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=pchip(x_premea,y0_premea,x);  
  
end
```