统计推断在数模转换系统中的应用

名字：黄灏　学号：5140309306

摘要：采用插值与拟合的方法定量描绘 Y-X 曲线的形态，并且通过分析给定的标准样本数据库， 使用启发式搜索算法给这个数模转换系统进行高效定标，由此得到样本的最优解,从而提高所取点的拟合程度。达到，又快（高效率）又好（低成本）的效果

关键词：matlab，启发式算法，多项式拟合，曲线拟合

1 引言

单传感器提取的信息往往是待识别目标的不完全描述，而利用多个传感器提取的独立 、互补的信息，进行多传感器信息融合 ，可以消除多传感器信息之间可能存在的冗余和矛盾，降低不确定性，并产生新的有意义的信息。

1.1 问题的提出：

在老师的讲解和课后看 ppt 得知，本课程的对象系统框图 1-1 所示，要对 X 和 Y 的关系进行定标。

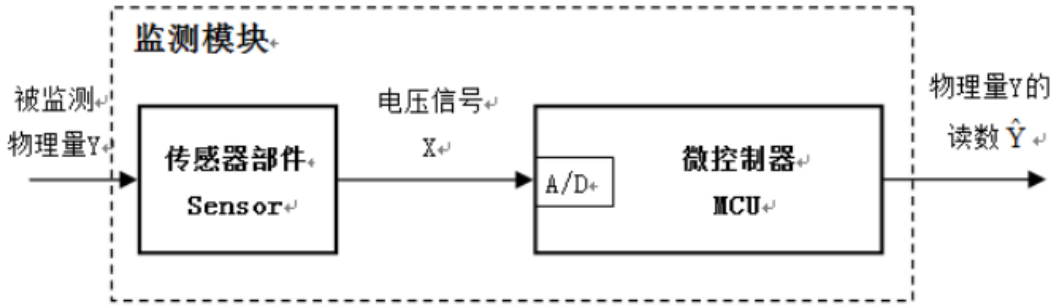
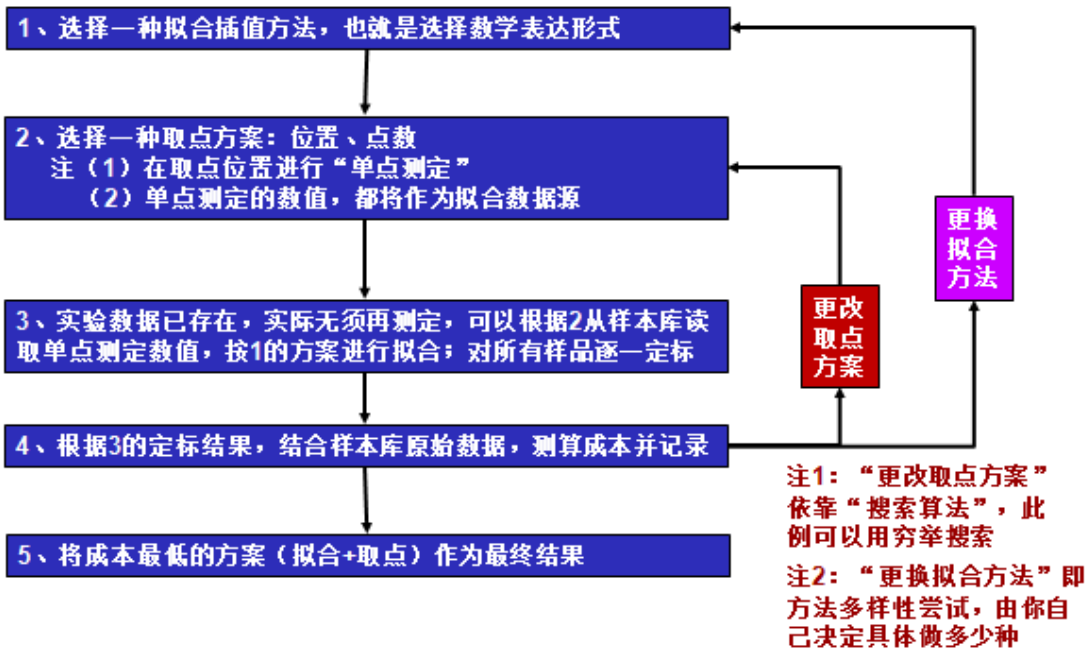


图 1-1 监测模块组成框图[1]

1.2 问题的求解路径：



2 拟合方法的选取：可供选择的拟合方法

2.1 多项式拟合

实验所提供的数据曲线大部分很接近一个多项式曲线。在本实验中最终经过比较决定采用分段三次 Hermite 多项式插值。

通过在 matlab 中 command 界面输入 help interp1（一维数据插值）得到版本可用的插值法有 nearest、spline、linear 等，分别运行后容易发现用 pchip(即某些版本中的 cubic)得到的最低成本更优，低于 90，由此可知该结果更优。

3 点的选取：

3.1 成本的计算：

单点定标误差成本

$$s_{i,j}=\begin{cases}0 & \text{if } |\hat{y}_{i,j}-y_{i,j}|\le 0.4\\ 0.1 & \text{if } 0.4<|\hat{y}_{i,j}-y_{i,j}|\le 0.6\\ 0.7 & \text{if } 0.6<|\hat{y}_{i,j}-y_{i,j}|\le 0.8\\ 0.9 & \text{if } 0.8<|\hat{y}_{i,j}-y_{i,j}|\le 1\\ 1.5 & \text{if } 1<|\hat{y}_{i,j}-y_{i,j}|\le 2\\ 6 & \text{if } 2<|\hat{y}_{i,j}-y_{i,j}|\le 3\\ 12 & \text{if } 3<|\hat{y}_{i,j}-y_{i,j}|\le 5\\ 25 & \text{if } |\hat{y}_{i,j}-y_{i,j}|>5\end{cases}$$

下标i代表样品序号

下标j代表观测点序号

对过大偏差的“惩罚性”

对某一样品i的定标成本

$$S_i=\sum_{j=1}^{51}s_{i,j}+12N_i$$

N_i是该样品定标时所测定的点数

第一项：误差成本

第二项：测定成本

定标方案总成本

$$C=\frac{1}{M}\sum_{i=1}^{M}S_i$$

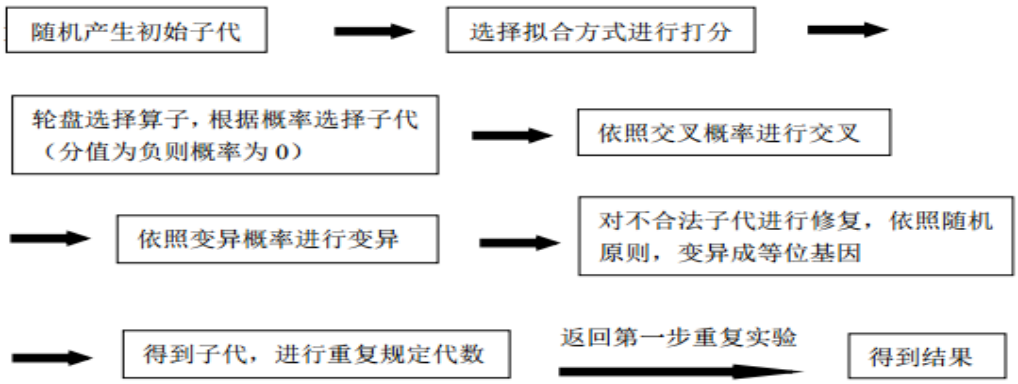
3.2 遗传算法的实现

1.暴力穷举

首先可以排除的就是穷举法，因为数据太多，组合的方法能达到上亿种。显然不切实际。

2.遗传算法

遗传算法是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。遗传算法是从代表问题可能潜在的解集的一个种群开始的，而一个种群则有经过基因编码的一定数目的个体组成。每个个体实际上是染色体带有特征的实体。染色体作为遗传物质的主要载体，即多个基因的集合，其内部表现是某种基因组合。初代种群产生后，按照适者生存和优胜劣汰的原理，逐步演化产生出越来越好的近似解，所以，把它用在解决实际问题的时候，则可以利用问题中所要达到的某个标准作为评估种群的适应度，从而选择个体，作为下一代，这样一代代的每一代比上一代好，则可逐步得到问题近似最优解。



遗传算法程序实现：

本组程序中，选用的迭代代数为 50，每个种群染色体为 51，种群大小为 100，还有交叉，变异。

初始化：对于数据每一组有 51，程序中 1 代表选取该点，0 则不选，初代是进行随机 51 中的 n 个点，把其对应点赋值为 1，而这里的 n 是 6 是 7 还是别的都不重要，由于经过足够多的遗传代数，最后得到的结果是相差无几的。

适应度的计算（fitness）：其实就是对当前代各个个体的成本进行计算，这也是题目所要求的。但题目要求的是成本最低，而在遗传中适应度越大越好，所以在每个个体的成本计算出来后，对其取倒数运算，并记录在 fitnessvalue 中，这样成本越低，适应度便越高。虽然在程序方面来看的话，只需要在选取下代和排序的时候，按从低到高选取即可，这些都是可以通过改变选择的方式可以达到同样的效果，但遗传算法是一个整体，按照其描述写的话，思路更清晰。

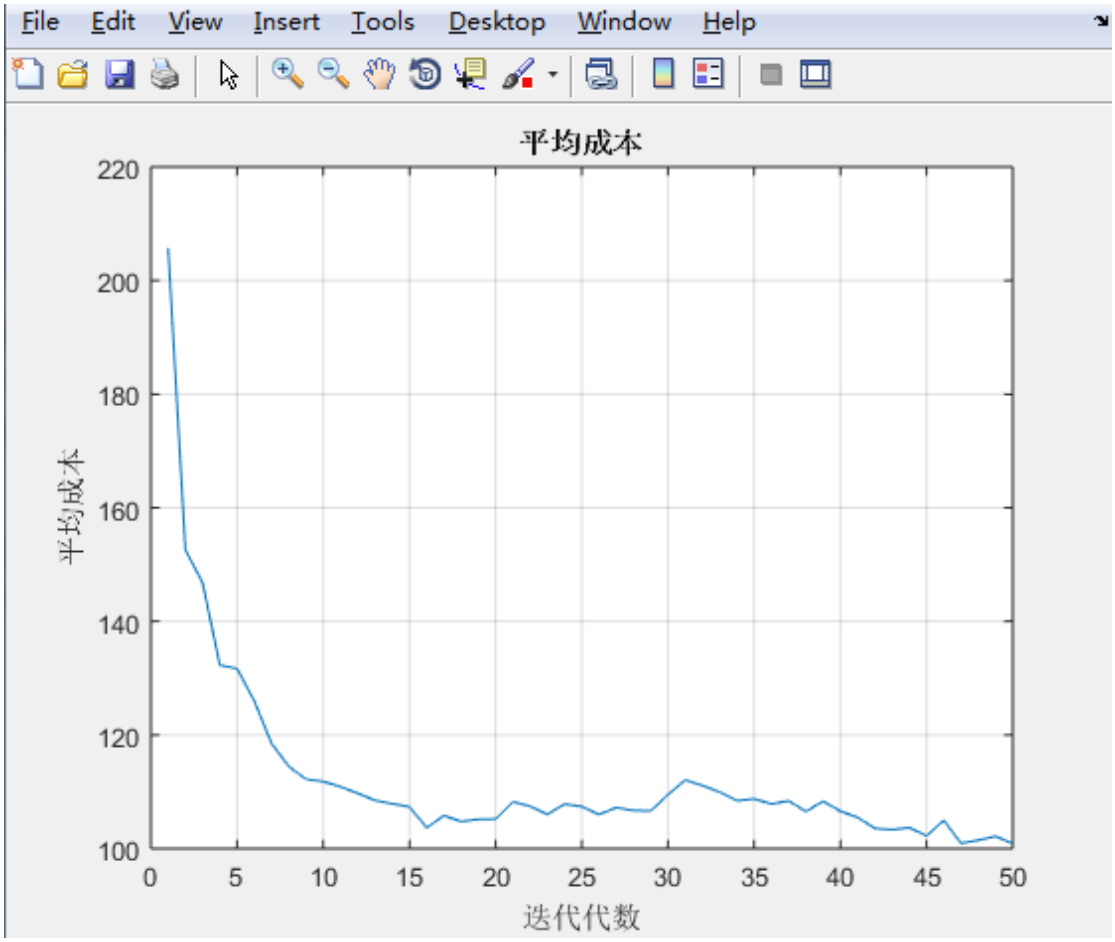
排序：对当代各个个体的适应度进行排序，本组程序中，按照适应度从小到大排序，即成本越低的在后面，第一个个体则是成本最高的。

选择（selection）：由当前代得出下一代，既然是优质遗传，则对更优的解，他的染色体应更容易遗传到下一代。我采用的是普遍都采用的赌轮盘选择法，进行 popsize 次循环选出 popsize 个后代。该方法使得越后的，即适应度越高的更容易被选择，符合遗传算法的思想。

交叉（cross）：本来的写了一次是按照交叉的几率 0.8 计算，但后来发现平均成本降的慢，为了加快种群的“进化”，交叉概率由(个体适应度/总体适应度)决定，则越优秀的个体越容易与别人染色体交叉。除此之外，还有一种交叉方法，是只用最优解与别的个体交叉，但担心这样会导致得到局部收敛的结果，所以并没有采用。

变异：（mutate）没什么特别的，参考生物自身基因变异，有一定概率，把 0 变成 1，1 变成 0，也是促进一种进化一种可能。但为了防止局部收敛，基因多样性，变异又是一个必要的。

如此循环 generationsize 代以后，输出得到的最优解。



最低成本为 87.2320 在迭代的 40 代就出现了，最优解对应选点方案为五个点 [5,16,28,37,48]

4、结论

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。为了更好地提高效率，我们选择 7 个“事先观测点”对曲线进行拟合，并得到了在评价函数下得分为 96.8787 分的 1 组“事先观测点”。它们的对应组所选取的样点为 [3,10,20,28,33,45,48] ,计算时间为 30min。在该课题中我们采用了遗传算法得到的最优解在第 40 代出现，最低成本为 87.2320，最优解为五个点[5,16,28,37,48]，如果继续重新运行，由于初始点的选取，会影响到最终所得结果，不一定仍为 87.2320 和以上五点，所以所得的 87.2320 也可能并不是真的最优解。

五、参考文献：

1、DS 证据理论在雷达体制识别中的应用

2、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

附录：Matlab 程序：

```
function main()
global m;
global n;
global p;
global q;
global bestnum;
global pingjun;

chromosize=51;
crossrate=0.8;
mutaterate=0.1;
popsize=100;
xuan=true;
generationsize=100;
[m,n,p,q]=
GA(popsize,chromosize,generationsize,crossrate,mutaterate,xuan);
display(p);
display(n);
display(m);
for i=1:m
    disp(bestnum(i+1))
end

x=1:1:generationsize;
y=pingjun(x);
plot(x,y);
grid on;%¼ÓÍø¸ñÏß
xlabel('µü´ú´úÊý');
ylabel('Æ½¾ù³É±¾');
title('Æ½¾ù³É±¾')
end

function fitness(popsize,chromosize)
global G;
global X;
global Y;
global thecost;
global samplesize;
global pop;
global fitnessvalue;
global pingjun;

fitnessvalue=zeros(1,popsize);
for i=1:popsize
```

```
    counter=0;
    for n=1:51
        if pop(i,n)==1
            counter=counter+1;
        end
    end
    x=zeros(1,counter);
    y=zeros(1,counter);
    cost=zeros(samplesize,51);
    allcost=zeros(samplesize);
    costall=0;
for j=1:samplesize
        c=1;
        for n=1:51
            if pop(i,n)==1
                x(c)=X(j,n);
                y(c)=Y(j,n);
                c=c+1;
            end
        end
        f=interp1(x,y,X(j,:),'pchip');

        Q=12;
        errabs=abs(Y(j,:)-f);
        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);
        cost(j,:)=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;

        allcost(j)=(c-1)*Q;
        for k=1:51
            allcost(j)=cost(j,k)+allcost(j);
        end
        costall=costall+allcost(j);
    end


    costave=costall/samplesize;
    thecost(G,i)=costave;
    pingjun(G)=pingjun(G)+costave;

fitnessvalue(i)=1/costave;%ËùËµµÄ³É±¾´úÌæÊÊÓ¦¶È£¬È¡µ¹Êý¡£³É±¾Ô½µÍ£¬ÊÊ
Ó¦¶ÈÔ½¸ß¡£
end
    pingjun(G)=pingjun(G)/popsize;

end



function
[m,n,p,q]=GA(popsize,chromosize,generationsize,crossrate,mutaterate,x
uan)
global G;%µ±Ç°µÚ¼¸´ú
global fitnessavg;
```

```
global bestfitness;
global bestgeneration;
global samplesize;
global X;
global Y;
global fitnessvalue;
global thecost;
global pingjun;
global bestnum;

bestnum(chromosize)=0;
pingjun(generationsize)=0;
thecost=zeros(generationsize,popsize);
samplesize=400;
fitnessavg=zeros(generationsize,1);
fitnessvalue(popsize)=0;
bestfitness=100000;
point=7;
data=csvread('20150915dataform.csv');
for i=1:samplesize
    X(i,:)=data(2*i-1,:);
    Y(i,:)=data(2*i,:);
end
chushi(popsize,chromosize,point);
for G=1:generationsize
   fitness(popsize,chromosize)
   rank(popsize,chromosize);
   selection(popsize,chromosize,xuan);
   cross(popsize,chromosize,crossrate);
   mutation(popsize,chromosize,mutaterate);

end
p=bestgeneration;
n=bestfitness;
m=bestnum(1);
q=0;
end

function chushi(popsize,chromosize,point)
global pop;
  pop=zeros(popsize,chromosize);
  for i=1:popsize
      for j=1:point
          place=round(rand*chromosize);
          while place==0
              place=round(rand*chromosize);
          end
          while (place==0)||(pop(i,place)==1)
              place=round(rand*chromosize);
          end
          pop(i,place)=1;
      end
  end

clear i;
clear j;
end


function rank(popsize,chromosize)
```

```
global fitnesssum;
global fitnessvalue;
global G;
global thecost;
global pop;
global bestgeneration;
global bestfitness;
global bestnum;

fitnesssum=zeros(popsize);
fitnesssum(1)=fitnessvalue(1);
for i=2:popsize
    fitnesssum(i)=fitnesssum(i-1)+fitnessvalue(i);
end
max=1;
tmp=1;
tmp1(chromosize)=0;
for i=1:popsize
    max=i;
   for j=i+1:popsize
       if fitnessvalue(j)<fitnessvalue(max);
           max=j;
       end
   end
   if max~=i
       tmp=fitnessvalue(i);
       fitnessvalue(i)=fitnessvalue(max);
       fitnessvalue(max)=tmp;

       tmp=thecost(G,i);
       thecost(G,i)=thecost(G,max);
       thecost(G,max)=tmp;

       for k=1:chromosize
           tmp1(k)=pop(i,k);
           pop(i,k)=pop(max,k);
           pop(max,k)=tmp1(k);
       end
   end

end

disp(G);
disp(1/fitnessvalue(1));
disp(1/fitnessvalue(popsize));
if 1/fitnessvalue(popsize)<bestfitness
    bestfitness=1/fitnessvalue(popsize);
    bestgeneration=G;
    counter=0;
    for k=1:chromosize
        if pop(popsize,k)==1
           counter=counter+1;
           bestnum(counter+1)=k;
        end
    end
    bestnum(1)=counter;
end

end
```

```
function
selection(popsize,chromosize,xuan)%±»³ÆÎª¶ÄÂÖÅÌÑ¡Ôñ·¨£¬½øÐÐpopsize´ÎÑ
¡³öpopsize¸ö°ó´ú
global pop;
global fitnessvalue;
global fitnesssum;
fitnesssum=zeros(1,popsize);
fitnesssum(1)=fitnessvalue(1);
for i=2:popsize
    fitnesssum(i)=fitnesssum(i-1)+fitnessvalue(i);
end
for i=1:popsize
   r=rand*fitnesssum(popsize);
   first =1;k=-1;
   last=popsize;

mid=round((last+first)/2);%ÕÒ´óÓÚrµÄ×îÐ¡Öµk,¶ÔÕâµÚG´úÇé¿öÑ¡µÚk¸ö¸öÌå£
¬Ô½°óµÄ±»Ñ¡ÔñµÄ¼¸ÂÊ¸ü´ó
   while (first<=last)&&(k==-1)
      if r>fitnesssum(mid)
          first=mid;
      elseif r<fitnesssum(mid)
          last=mid;
      else
          k=mid;
          break;
      end
      mid=round((last+first)/2);
      if (last-first)==1
          k=last;
          break;
      end
   end

   for j=1:chromosize
       popnew(i,j)=pop(k,j);
   end
end

   if xuan
       p=popsize-1;
   else
       p=popsize;
   end
   %Èç¹ûÑ¡£¬ÔñÁôÏÂµÚ¸ö×îÓÅµÄ²»½øÐÐ¸Ä±ä£¬Ö»±äÇ°ÃæµÄpopsize-1¸ö
   for i=1:p
       for j=1:chromosize
           pop(i,j)=popnew(i,j);
       end
   end

clear k;
clear i;
clear j;
end

function cross(popsize,chromosize,crossrate)
global pop;
global fitnessvalue;
global fitnesssum;
```

```
newpop=zeros(popsize,chromosize);
for i=1:popsize
    for j=1:chromosize
        newpop(i,j)=pop(i,j);
    end
end

for i=1:popsize
    if rand<fitnessvalue(i)/fitnesssum(popsize)
         place=round(rand*chromosize);
         while place<1
           place=round(rand*chromosize);
         end
         which=round(rand*popsize);
         while which<1
             which=round(rand*popsize);
         end
         for j=place:chromosize
             newpop(i,j)=pop(which,j);
             newpop(which,j)=pop(i,j);
         end
    end

end

for i=1:popsize
    for j=1:chromosize
        pop(i,j)=newpop(i,j);
    end
end

clear i;
clear j;
clear place;
clear which;
end

function mutation(popsize,chromosize,mutaterate)
global pop;
for i=1:popsize
   if rand<mutaterate
       place=round(rand*chromosize);
       if place==0
           continue;
       end
       if pop(i,place)==1
           pop(i,place)=0;
       else pop(i,place)=1;
       end
   end
end
clear i;
clear place;
end
```