统计推断在数模转换系统中的应用

组号:40　姓名：罗锦鉴　学号:5130309369

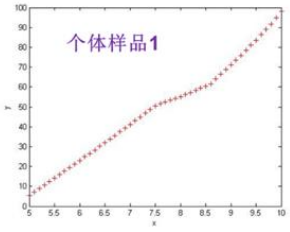
摘要：本次课题通过对具有非线性关系的样本数据的研究，采用三次样条插值并通过改进简化遗传算法来寻求使成本最低的样点。

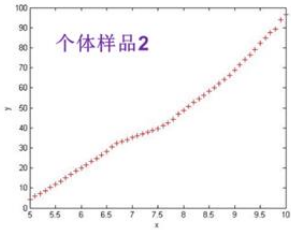
关键词：多项式拟合、三次样条插值、遗传算法

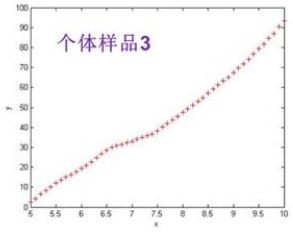
1 引言

假定有某型投入批量试生产的电子产品，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。课题中的程序均使用 Matlab 语言。

2 拟合方法







由上图可以看出样本数据得到的曲线相互之间存在相似性，故可采用统一的方法得到曲线的函数表达式，并用如下的误差成本函数来对拟合曲线的精确性进行评判。拟合曲线的精确性随着误差成本的增大而降低。

$$s_{i,j}=\begin{cases}0 & \text{if } \left|\hat{y}_{i,j}-y_{i,j}\right|\le 1\\ 1 & \text{if } 1<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 2\\ 2 & \text{if } 2<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 3\\ 4 & \text{if } 3<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 5\\ 10 & \text{if } \left|\hat{y}_{i,j}-y_{i,j}\right|>5\end{cases}$$

由于样本数据曲线的函数表达式可以泰勒展开为 n 次多项式，在本次课题中采用多项式拟合。

为确定取点数和拟合次数，决定先采用均匀间隔的取点使用 matlab 自带的 polyfit 函数进行简单的成本估计。

表 2-1 为在三次多项式拟合时，不同的取点数下的各项成本。

表 2-2 为取 7 个点时，分别在三次、四次、五次的条件下进行多项式拟合得到的各项成本。

表 2-3 对次数和取点数对总成本造成的影响进行了综合比较。

表2-1 取点数与成本关系

取点数	误差成本	测定成本	总成本
4	95.9	48	143.9
5	57.3	60	117.3
6	58.3	72	130.3
7	52.2	84	136.2

表2-2 多项式拟合次数与成本关系（取7个点）

多项式次数	误差成本	测定成本	总成本
3	52.2	84	136.2
4	32.6	84	116.6
5	28.4	84	112.4

表2-3 取点数、多项式拟合次数与总成本关系

取点数	三次	四次	五次
4	143.9	/	/
5	117.3	130.0	/
6	130.3	107.8	127.5
7	136.2	116.6	112.4

由表中数据可知，若取点数恰好满足 polyfit 拟合的最低取点数，其误差会相对较大。除去此类情况，可以看出相同取点数下高次拟合误差较小，相同次数拟合下，取点数在超过6点左右时，误差成本的较少会弱于取点成本的增长，从而导致总成本的增高。

表 2-4 为 Matlab 自带的三次样条插值函数 spline 与次数为 3 的 polyfit 拟合的比较。

表2-4 polyfit与spline的成本比较

取点数	Polyfit 误差成本	Spline 误差成本	spline 总成本
4	95.9	95.9	143.9
5	57.3	50.8	110.8
6	58.3	24.4	96.4
7	52.2	13.4	97.4

由表中数据可以看出，spline 函数在取点数增多时相较 polyfit 函数明显降低了误差成本，并且在取 6 点时的总成本低于表 2-3 中的所有用 polyfit 得到的曲线拟合后的总成本。因此在接下来的算法设计中，将采用取 6 点时的三次样条插值进行拟合。

3 程序设计

3.1 遗传算法

遗传算法（Genetic Algorithm）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，而一个种群则由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（即基因型）是某种基因组合，它决定了个体的形状的外部表现，如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此，在一开始需要实

现从表现型到基因型的映射即编码工作。由于仿照基因编码的工作很复杂，我们往往进行简化，如二进制编码，初代种群产生之后，按照适者生存和优胜劣汰的原理，逐代演化产生出越来越好的近似解，在每一代，根据问题域中个体的适应度大小选择个体，并借助于自然遗传学的遗传算子进行组合交叉和变异，产生出代表新的解集的种群。这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境，末代种群中的最优个体经过解码，可以作为问题近似最优解。

3.2 简化设计

遗传算法主要运算过程分为：初始化、个体评价、选择运算、交叉运算、变异运算、终止条件判断。由于过多的循环语句会大大降低程序运行的速度，在程序设计中将交叉运算取消，通过其他方式在变异运算中保留交叉运算所具有的部分性质。

（1） 初始化

在本次课题中，为简化程序，将 5.1-10.0 对应至 1-51，直接以每一种取点方式作为个体，其中所取点的值为基因。种群个体数为固定值。第一代种群通过全随机得到。为防止随机到相同数值而引起的检验重复的费时步骤，本次课题中均通过 Matlab 自带的 randperm 函数得到 1-51 的随机排序，并截取前 6 位得到新的全随机个体。

（2） 个体评价

由于在接下来的选择运算中采用直接淘汰制，因此无需进行总成本与适应度之间的变换，直接以总成本来评价个体。

（3） 选择运算

在得到所有个体的总成本后，将种群按成本大小排序。为了在保持取优的条件下同时不能将除最优个体外的所有基因淘汰，分别取排在 1、2、4…即排在二次幂位置的个体，这样令排在前十的个体得到了最大比例的保留，同时也没有淘汰所有基因。

（4） 变异运算

在选择运算中取出的个体即生存个体，同时也是进行变异的个体。为了在变异的同时也能维持与原个体的相似性，类似交叉运算中保有原父母的基因，变异通过设定基因允许的最大偏离值而非直接随机得到新基因来产生新个体。为防止基因偏离时产生重复或越界，在程序设计中限定基因偏离时不得越过其相邻的基因值。由于共有六位基因，各基因间平均间隔 10 个值，考虑到基因偏离过大会导致程序会浪费时间在判断越界和重新生成偏离值，以及基因偏离过小导致的最优解收敛过慢，最后设定最大偏离值为 3。

生成新的种群时将选择运算中得到的个体进行变异，每个个体变异出同样数量的新的个体，若此时未补足种群限定的数量，则利用全随机生成补足剩余的空位。同时为防止前一代的最优个体被淘汰，在每个个体所变异得到的新的个体中替换一个为原个体。

（5） 终止条件

当设定遗传代数计数器达到初始设定的进化代数时，终止遗传运算，并将最后一代个体的基因转化得到的取点值作为解。

经过较小种群和较少代数的试运行后，耗时统计中 75%左右的时间花费在拟合上，20%的时间花费在读取庞大的样本数据上。除去这些必要的时间花费，其余时间成本可以忽略。

4 结果分析

表 4-1 为程序在最大代数为 30，种群个体数为 100 时得到的运行结果，总运行时间为

614.463s，其中已经省略了最优个体不变时的代数的数据。

表4-1 程序运行结果（100个体，30代）

代数	当前最小成本	当前最优个体	与上一代的成本差
0	111.1514	4 14 25 34 46 47	/
1	104.2047	4 12 23 36 44 47	6.9467
2	101.2420	3 14 24 36 44 49	2.9627
3	97.8561	4 12 24 31 44 49	3.3859
4	95.8827	4 13 22 31 42 49	1.9734
5	94.1557	3 11 22 31 44 49	1.7270
7	93.9510	3 13 22 31 42 50	0.2047
9	93.6482	3 11 22 30 40 49	0.3082
11	93.2431	3 11 22 31 41 49	0.4051
30	93.2431	3 11 22 31 41 49	0

由表中数据可以看出，在程序运行的前十代中，最优个体的成本以极快的速度收敛，在11 代后，最优个体已经保持不变。

表 4-2 为程序在最大代数为 10，种群个体数位 50 时得到的运行结果。

表4-2 程序运行结果（50个体，10代）

代数	当前最小成本	当前最优个体	与上一代的成本差
0	125.5832	6 9 17 26 44 46	/
1	110.4701	4 9 19 27 44 48	15.1131
2	101.4424	2 11 21 29 44 46	9.0277
3	98.3721	2 10 19 29 43 48	3.0703
4	95.2335	3 10 21 31 42 48	3.1386
5	93.5437	3 11 22 31 42 49	1.6898
6	93.3806	3 11 21 31 41 49	0
10	93.2559	3 12 22 32 43 50	0.1247

比较表 4-1 和表 4-2 的数据可以发现，虽然初代最优个体成本相差较大，但在 10 代过后种群中的最优个体成本已十分接近。可以看出在不追求更高的精确度的情况下，增加种群个体数是没有必要的。同时每一代的最优个体偏离最终解的幅度越大，程序的矫正力度也越大，但在极为接近最终解时，往往无法在较少的代数内使得种群个体得到进一步的优化。

综合数据对程序进行分析可以发现，增加种群个体数只会增加在最初代接近最优解区域的可能性。虽然更大的种群可以在随后的选择、变异运算中增多变异个体的种类，但由于程序自身未提供筛选除去相同个体的函数，这导致经过 2 至 3 代后产生的新种群中的大部分个体都与最优解相似。这也解释了为什么程序对偏离最终解的个体矫正力度较强而对极为接近的个体难以进行进一步优化。

通过以上分析能够得知程序所得的解收敛较快，由于没有基因的相互交换以及每代个体被高比率地直接淘汰，程序产生的最终解有可能依赖于最初产生的随机个体而导致局限于局部最优解的情况。

经过多次的程序测试，发现最终解之间的差值小于 0.3，由于程序运行中存在大量随机现象，0.3 的差值不足以说明程序运行的最终解对最初种群的依赖性。

对本次课题给出的最后解是在测试程序运行的最终解对最初种群依赖性的过程中得到的最小解[3　　11　　22　　31　　41　　50]，其成本为 93.0533。

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] 卓金武,魏永生等. Matlab 在数学建模中的应用.北京航空航天大学出版,2010-5

附录：

1 主程序（genetic）

```
clc;
close all;
clear all;

Generationmax=30;

popsize=50;
point=6;
population=randpop(51,popsize,point);
generation=0;

while generation<Generationmax
    disp(generation);
    population=selection(Fitness(population,popsize),population,popsize,point);
    population=repopulate(population,popsize,point);
    generation=generation+1;
end
```

2 全体成本计算函数（Fitness）

```
function fitvalue = Fitness(population,popsize)
for i=1:1:popsize
    fitvalue(i,1)=cost(population(i,:));
end
```

3 单个个体成本计算函数（cost）

```
function cost = cost(xx)

my_answer=xx;
my_answer_n=size(my_answer,2);

minput=dlmread('20141010dataform.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;

index_temp=logical(my_answer_gene);
x_optimal=x(:,index_temp);
y0_optimal=y0(:,index_temp);
for j=1:nsample
    y1(j,:)=mycurvefitting(x_optimal(j,:),y0_optimal(j,:));
end

Q=12;
errabs=abs(y0-y1);
le0_5=(errabs<=0.5);
le1_0=(errabs<=1);
le2_0=(errabs<=2);
le3_0=(errabs<=3);
le5_0=(errabs<=5);
g5_0=(errabs>5);
sij=0.5*(le1_0-le0_5)+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;
end
```

4 拟合函数

```
function y1 = mycurvefitting( x_premea,y0_premea )
x=5.0:0.1:10.0;
y1=interp1(x_premea,y0_premea,x,'spline');
end
```

5 选择运算函数

```
function population = selection(fitvalue,population,popsize,point)
population=sort(population,2);
population(:,point+1)=fitvalue;
population=sortrows(population,point+1);
disp(population(1:5,point+1));
population=population(1:popsize,1:point);
disp(population(1:5,:));
end
```

6 变异运算函数

```
function sample = gerand(mut,point)
flag=0;
list(1,1)=0;
list(1,point+2)=52;
list(1,2:point+1)=mut;
for i=2:1:point+1
    while flag==0
        x=randi(3);
        y=randi(2);
        if y==1&&list(1,i)+x-1<list(1,i+1)
            list(1,i)=list(1,i)+x-1;
            flag=1;
        end
        if y==2&&list(1,i)-x+1>list(1,i-1)
            list(1,i)=list(1,i)-x+1;
            flag=1;
        end
    end
    flag=0;
end
sample=list(1,2:point+1);
end
```

7 新种群生成函数

```
function population = repopulate(population,popsize,point)
for q=1:1:5
    mut(q,:)=population(power(2,q-1),:);
end
k=1;
for i=1:1:50
    population(i,:)=gerand(mut(k,:),point);
    k=ceil(i/10);
```

```
end
for p=51:1:popsize
    population(p,:)=randpop(51,1,point);
end
for j=1:10:41
    population(j,:)=mut(ceil(j/10),:);
end
end
```

8 随机个体生成函数

```
function sample = randpop(bounds,popsize,num)
size=0;
while size<popsize
    list=randperm(bounds);
    sample(size+1,:)=list(1,1:num);
    size=size+1;
end
end
```