一个测量装置在大规模制造中的标定问题

魏思哲 李泓録

[摘　要]对于大规模制造的电子产品测量模块，由于各个传感器之间存在差异，在制造时需要进行标定。但是随着选取的特征点数目的增加，成本也会大大增加。因此再选取特征点进行拟合时，效果与选取的数目之间存在某种制约关系。为适合大规模高效率制造，本文探究如何运用三次样条插值法进行数据拟合以及运用遗传算法来对特征点进行选取，并对得到的结果进行分析和讨论。

[关键词]标定问题，三次样条插值法，遗传算法，特征点选取，拟合，

1. 课题内容和目标

1.1 课题内容的提出

如图 1.1，对不同传感器个体间进行测试，发现存在个体差异，并且输入-输出间存在明显的非线性，因此需要进行标定。而在大规模生产中，标定数目在不影响效果的条件下越少越好，因此提出本课题内容。

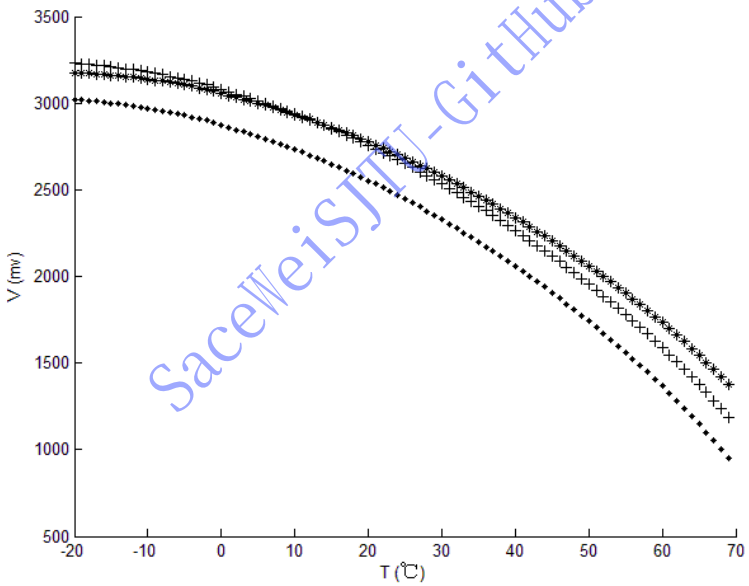

图 1.1 传感器特性的非线性与个体差异性

1.2 课题目标

本课题中，假定对一个大规模的测量装置进行标定，且被测物理量为温度，可测范围是-20℃至 70℃。所用核心传感器的输入-输出特性有明显非线性，且个体差异性比较大。将给出的 500 件传感器个体样品的精确数据作为原始数据，来对特征点进行选取，从而得到高效率低成本且拟合效果好的解决方案。

2. 研究方法

2.1 三次样条插值法

三次样条插值（Cubic Spline Interpolation）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[1]。运用到编程

语言中就是对于中间部分的相邻的 4 个离散点使用三次曲线进行拟合，从而将拟合曲线作用到中间两个点之间；对于边界处的点，选取其旁边三个点同样进行三次曲线拟合，从而在靠近边界的两个点处做出曲线。

使用三次样条插值法，可以保证在断点处斜率和曲率都连续。

2.2 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[2]。

如图 2.2，遗传算法的基本运算过程如下：

1. 初始化：确定目标方案编码机制，并随机生成一些方案，构成初始种群 P（0）。
2. 建立评价机制:对当前种群中每个个体计算适应度。
3. 选择：在每个个体的适应度评估基础上，对种群中的个体进行自然选择，总体上筛选出适应度更高的个体。
4. 交叉运算：对胜出的个体作用交叉算子，进行染色体交叉。
5. 变异运算：依据变异率随机选择个体作用变异算子，模拟染色体变异。
6. 循环：本轮得到的优胜个体组成的种群 P（t）进入新一轮选择。
7. 终止：达到最大进化代数则终止，并以进化过程中所得到的具有最大适应度个体作为最优解输出，得到遗传算法搜索结果。

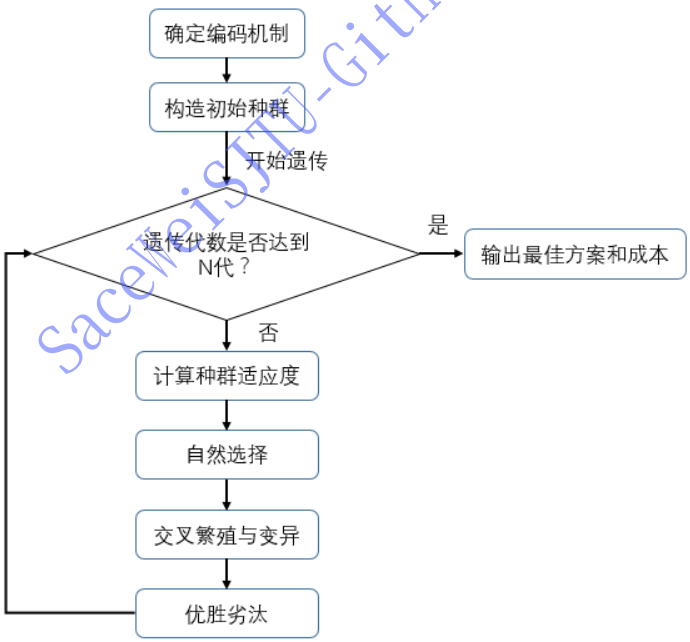

图 2.2 遗传算法流程图

3. 建模与解

3.1 成本模型说明

为评估和比较不同的标定方案，特制定以下成本计算规则。成本较低的标定方案，认定为较优方案。

3.1.1 单点测定成本

实施一次单点测定的成本记为符号 Q。本课题指定 Q=50。

3.1.2 标定误差成本

$$s_{i,j}=\begin{cases}0 & \text{if } \left|\hat{T}_{i,j}-T_{i,j}\right|\le 0.5\\ 1 & \text{if } 0.5<\left|\hat{T}_{i,j}-T_{i,j}\right|\le 1.0\\ 5 & \text{if } 1.0<\left|\hat{T}_{i,j}-T_{i,j}\right|\le 1.5\\ 10 & \text{if } 1.5<\left|\hat{T}_{i,j}-T_{i,j}\right|\le 2.0\\ 10000 & \text{if } \left|\hat{T}_{i,j}-T_{i,j}\right|>2.0\end{cases} \quad (1)$$

单点误差对应的成本值按式（1）计算，以符号 $s_{i,j}$ 记。其中 $T_{i,j}$ 表示第 i 个样本之第 j 点的实际温度值，$\hat{T}_{i,j}$ 表示标定后得到的估计值（读数）。

单个样本个体的标定误差成本用和式（2）计算。

$$S_i=\sum_{j=1}^{90}s_{i,j} \quad (2)$$

3.1.3 样本个体标定成本

$$c_i=S_i+Q\cdot n_i \quad (3)$$

样本个体的标定成本是测定成本与误差成本之和，见式（3）。式中 n_i 表示对该样本个体标定过程中的测定点数目。

3.1.4 方案成本

标定方案的成本值按式（4）计算。对标准样本库中每个样本个体逐一开展标定，取所有样本个体标定成本的统计平均。其中 M 是样本总数，M=500。

$$C=\frac{1}{M}\sum_{i=1}^{M}c_i \quad (4)$$

3.2 基于遗传算法的建模与解

3.2.1 模型建立

3.2.1.1 基因的建立与特征点的选取

将本问题的样本中的 90 个测试点抽象为基因，并且定义 90 个基因中特征点的取法：

用x_i表示第 i 号测试点是否被选为特征值点，当x_i=1 时表示第 i 号测试点被入选为特征值点，当x_i=0 时表示第 i 号测试点不入选为特征值点。

这样一来，由每个个体中不同的基因可得到不同的结果。

3.2.1.2 适应度函数的定义

根据遗传算法，随着种群的进化，其适应度应不断提高。又依据本此建模的问题，应将成本降至最小，则我们组对适应度取为成本的倒数，及得到如（5）的适应度函数。

$$\text{fitness}=\frac{1}{cost} \quad (5)$$

3.2.1.3 交叉与变异比率的选取

为模拟遗传过程，并且使优秀个体能够最大限度保留下来，经反复实验，我们将交叉互换比率设置为 0.95，变异率设置为 0.05。

3.2.2 算法描述

在该问题的实现过程中，遗传算法作为一项很重要的算法，起到了很关键的作用，因此，在本文中，我们将重点介绍遗传算法的实现过程。

首先，将每一个温度值作为一个基因，而改基因的基因型仅有 0 和 1 两种，其中，0 对应着在进行温度的测定过程中，我们选择该点进行测定；反之，1 对应着不选取该点进行温度测定。

通过随机数函数创建矩阵，矩阵每行即代表一个个体的基因，在创建随机数矩阵时，在实验后期，我们可以适当地调节随机数矩阵中基因 1 出现地概率来缩短实验时间，尽快获得最优解（见后文）。

由于要求选择地温度测试点具有普适性，我们对 500 组真实的电压-温度数据进行误差标定成本计算，运用三次样条拟合，拟合出选择测定点后得到的温度-电压曲线，在运用 interp1 函数进行插值，找到问测定的温度值其电压值对应的实际温度值，通过给定的误差标定成本函数得到每个个体对应的温度选择方式所需要的标定误差成本，对 500 组数据重复上述过程，在除以数据组数，从而得到对应的平均标定误差成本，总成本为标定误差成本与测量温度成本之和，根据总成本，运用成本的倒数或平方倒数，作为适应度函数，适应度函数值越高，意味着在进化过程中，这样的个体存活概率更高。

将适应度函数归一化，然后通过随机数函数产生一个随机数，通过一定的线性关系，对应个体则被选择，通过选择一定对数个体，保持子代个体数不变，进行交叉操作，即，产生随机位点，一定概率的条件下，二者在位点处交换各自位点以后的基因，产生子代个体。

对于新产生的子代个体，通过随机数函数产生一个随机数，通过一定的线性关系，对应个体基因上的一个位点，一定概率情况下，该位点发生变异操作，即 0 变为 1，1 变为 0。

通过上述操作，即进化，约一定代数后，找到该代对应的个体中的成本最低的测定方案，即可作为最优方案。

3.2.3 性能分析，表现和评价

3.2.3.1 参考组的选取

选取种群数量为 200，遗传代数为 100 作为参考组

见表 1

表 1 参考组

入选点	成本	时间（s）
-17 -5 4 51 66	264.6080	1091.545988

3.2.3.2 遗传代数对结果的影响

见表 2

表 2 遗传代数的影响（种群数量=200）

遗传代数	入选点	成本	时间（s）
20	-18 -7 0 21 42 46 68	357.1920	230.324670
30	-18 -7 0 46 55 68	310.8980	358.365470
40	-19 -5 4 53 66	271.1260	467.228967
50	-19 -4 4 38 66	289.8960	582.234918
80	-16 -6 4 45 67	275.2180	884.342594

见表 2 数据可知，当遗传代数达到 40 代以上，入选点稳定在 5 个，同时成本已经降到

300 以下。我们认为数据已经到达相对稳定的状态。但是随着遗传代数的增加，很可能会找到更加优质的解，且对于此类问题，其选用 500 组数据作为其普适性能测试的标准已可以验证其具有良好的普适性，同时，代数为 10~100 代所需的时间均在 20 分钟以内，可以考虑采用更长的时间来尝试得到更优的解。因此我们在程序中取遗传代数为 100。

3.2.3.3 种群数量对结果的影响

见表 3

表 3 种群数量对结果的影响（遗传代数=40）

种群数量	入选点	成本	所花时间（秒）
100	-19 -10 0 15 38 62 64	357.9600	222.422535
150	-16 -11 -4 10 45 60 65	354.5120	339.788252
250	-16 -6 5 59 63	274.9960	563.228084

由遗传代数对结果的影响可知，结果中可取代数为 100。但是在研究种群数量对结果的影响时，为节省时间我们取代数为 0。由上表及参考组可知当种群数量达到 200 后入选点数量即维持在 5 个，因此，在结果中我们选择种群数量为 200。

3.2.4 模型得到结果

由上述分析知，使用遗传算法的模型，采用种群数量为 200，遗传代数为 100，可以得到最优结果在 260 左右。具体结果还会随着实验的变动有所浮动，但是基本稳定。

3.3 代码说明

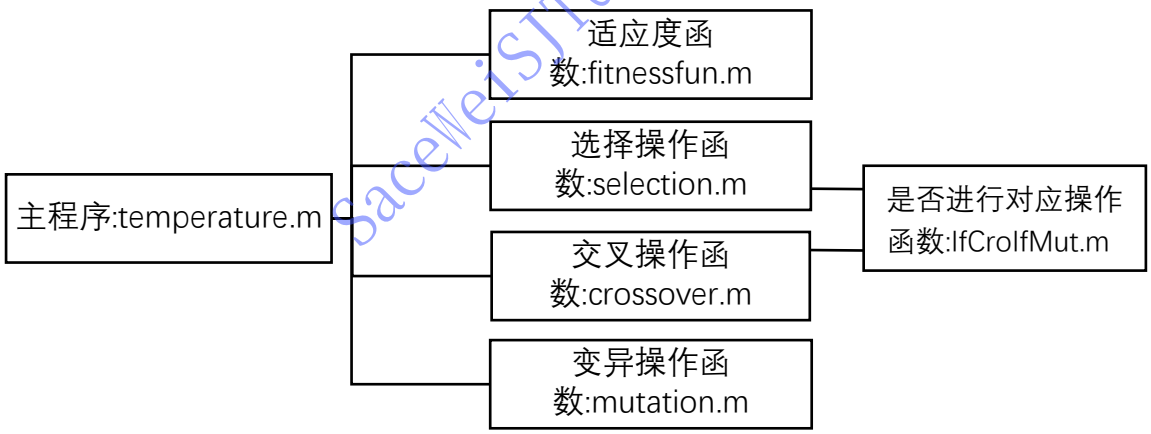

图 3.3 1 代码调用关系示意图

代码各部分的调用关系如上图所示，主程序中随机数函数构建初始种群 population[]为：个体数*90 的一个矩阵，对矩阵整体操作，利用适应度函数 fitnessfun()，返回每个个体的适应度函数值构建的 1*90 的行阵 fitvalue[]以及含有将适应度函数归一化后的到每个个体的累计概率 1*90 行阵 cumsump[]；调用 selection()函数，参数为 cumsump[]，返回值为选择的两个个体编号；对应该个体编号传入 crossover()函数中，调用 IfCroIfMut 函数决定是否进行交叉操作，返回得到新的个体行阵 scnew[]；将子代传入 mutation()函数中，调用 IfCroIfMut 函数决定是否进行变异操作，返回得到新的子代个体行阵 smnew[],同时，将所有子代组建为新的种群 population[],重复操作。

4.自主拓展

在此问题中，可以发现，其主要成本来源与测量温度点所需要的成本，因此，我们在通过以上的探究也通过计算，例如：当选择的温度测量点为 6 个时，对应的成本为 357.9600、354.5120；当测量点为 5 个时，对应的成本为 274.9960、264.6080，成本基本上接近选择的测试点数量乘以每个测试点需要花费的价格（50），因此，我们在后期的实验中，产生初始种群即适当地降低选中为测试点的比例，利用随机函数 population=round(rand(popsize,Bitlength)-0.2)，从而加快实验速度，也可以更好地找到最优解。

5.结论

本文通过使用遗传算法对温度测量装置的标定中特征点的选取进行分析，得到的最优结果为选取 5 个特征点，从而将成本控制在 260 左右。相对来说，使用遗传算法速度较慢，但是得到结果更好。

6.鸣谢

我们非常感谢袁焱老师和李安琪老师的指导。这门课程的设立，使我们从完全不了解 matlab 到能够掌握这一门语言进行问题建模仿真。同时，在对案例一的学习研究过程中，我们了解了遗传算法和三次样条插值法的应用，能够为我们之后的学习提供很大帮助。

7.参考文献

[1] 百度百科三次样条插值词条：
https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%A0%B7%E6%9D%A1%E6%8F%92%E5%80%BC/3476729?fr=aladdin
[2] 百度百科遗传算法词条：
https://baike.baidu.com/item/%E9%81%97%E4%BC%A0%E7%AE%97%E6%B3%95/838140?fr=aladdin
[3] 工程问题建模与仿真之案例 1_V1.1 20180910
[4] 上海交通大学 电子系 工程问题建模与仿真讲座 3_案例 1 问题和基本数学方法
[5] 上海交通大学 电子系 工程问题建模与仿真讲座 4_案例 1 报告初稿问题讲解和写作要领提示
[6] 卓金武，MATLAB 在数学建模中的应用.北京航空航天大学出版社,2011.04.01

附录：程序源码

temperature.m

```
tic
global Bitlength
global truetablelist
global costofmea %测量一个的成本
global popsize
Bitlength=90; %编码长度
%直接读入文件
truetablelist=csvread('dataform20160902.csv');
popsize=200; %初始种群的大小
costofmea=50;%测量一个温度值的成本为 50
Generationmax=100;     %最大代数
pcrossover=0.95;       %交配概率
pmutation=0.05;        %变异概率
%产生初始种群
population=round(rand(popsize,Bitlength)-0.2);
%计算适应度，返回值为适应度 fitvalue 和积累概率 cumsump
[fitvalue,cumsump]=fitnessfun(population);
Generation=1;
while Generation<Generationmax+1
    for j=1:2:popsize
        %选择操作
        seln=selection(cumsump);
        %交叉操作
        scro=crossover(population,seln,pcrossover);
        scnew(j,:)=scro(1,:);
        scnew(j+1,:)=scro(2,:);
        %变异操作
        smnew(j,:)=mutation(scnew(j,:),pmutation);
        smnew(j+1,:)=mutation(scnew(j+1,:),pmutation);
    end
    population=smnew;    %产生新种群
    %计算新种群的适应度
    [fitvalue,cumsump]=fitnessfun(population);
    %记录当前代最好的适应度和平均适应度
    [fmax,nmax]=max(fitvalue);
    fmean=mean(fitvalue);
    ymax(Generation)=1/fmax;  %此处计算的是该代最优成本
    ymean(Generation)=1/fmean;%计算平均最优成本
    %记录当前代的最佳染色体
    unit=population(nmax,:);
    xmax=zeros(Generationmax,Bitlength);
    xmax(Generation,:)=unit;
```

```
    Generation=Generation+1;
end

toc

Generation=Generation-1;
bestpopulation=unit;
bestvalue=ymax(Generation);
finaltesttable=[];
for i=1:Bitlength
    if unit(i)==1
        finaltesttable(end+1)=i-21;
    end
end
disp('The final data we choose is:');
disp(finaltesttable);
disp('The final cost is:');
disp(bestvalue);

figure(1);
hand1=plot(1:Generation,ymax);
set(hand1,'linestyle','-','linewidth',1.8,'marker','*','markersize',6)
hold on;
hand2=plot(1:Generation,ymean);
set(hand2,'color','r','linestyle','-','linewidth',1.8,'marker','h',...
    'markersize',6)
xlabel('进化代数');ylabel('最优/平均成本值');xlim([1 Generationmax]);
legend('最优成本','平均成本');
hold off;
```

fitnessfun.m

```
function [fitvalue,cumsump]=fitnessfun(population)
global Bitlength
global truetablelist
global costofmea
global popsize
testtable=[];
caltable=[];
fitvalue=zeros(popsize,1);
cumsump=zeros(popsize,1);
numberoftruevalue=size(truetablelist,1)/2;
```

```
sumcostofmea=zeros(popsize,1);%测量点成本  mm
sumcostofcal=zeros(popsize,1);%标定误差成本
sumcost=zeros(popsize,1);%总成本
fitvalue=zeros(popsize,1);%适应度值
for i=1:popsize
    caltable=[];
    testtable=[];
    for k=1:Bitlength
        if population(i,k)==1
            testtable(end+1)=k-21;
        else
            caltable(end+1)=k-21;
        end
    end
    %测定总成本
    sumcostofmea(i)=costofmea*size(testtable,2);
    for j=1:(numberoftruevalue)
        volcaltable=[];
        voltesttable=[];
        for h=1:size(testtable,2)
            voltesttable(end+1)=truetablelist(2*j,(testtable(h)+21));
        end
        for h=1:size(caltable,2)
            volcaltable(end+1)=truetablelist(2*j,(caltable(h)+21));
        end
        fitresult=interp1(voltesttable,testtable,volcaltable,'spline');
        deviationresult=abs(fitresult-caltable);
        %误差标定总成本
        anumberfortest=size(caltable,2);
        for l=1:(anumberfortest)
            if deviationresult(l)<=0.5
                sumcostofcal(i)=sumcostofcal(i)+0;
            elseif deviationresult(l)<=1.0
                sumcostofcal(i)=sumcostofcal(i)+1;
            elseif deviationresult(l)<=1.5
                sumcostofcal(i)=sumcostofcal(i)+5;
            elseif deviationresult(l)<=2.0
                sumcostofcal(i)=sumcostofcal(i)+10;
            else
                sumcostofcal(i)=sumcostofcal(i)+10000;
            end
        end
    end
    sumcostofcal(i)=sumcostofcal(i)/numberoftruevalue;%算出平均值
```

```
    sumcost(i)=sumcostofmea(i)+sumcostofcal(i);%总成本
    fitvalue(i)=1/sumcost(i);%适应度函数值
end
%计算选择概率
fsum=sum(fitvalue);
Pperpopulation=fitvalue/fsum;
Pperpopulation_20=Pperpopulation.^2;
psum=sum(Pperpopulation_20);
Pperpopulation_20=Pperpopulation_20/psum;
%计算累计概率
cumsump(1)=Pperpopulation_20(1);
for i = 2:popsize
    cumsump(i)=cumsump(i-1)+Pperpopulation_20(i);
end
fitvalue=fitvalue';
cumsump=cumsump';
```

selection.m

```
%选择操作
function seln=selection(cumsump)
%从种群中选择两个个体
for i=1:2
    r = rand;
    prand=cumsump-r;
    j=1;
    while prand(j)<0
        j=j+1;
    end
    seln(i)=j;
end
```

mutation.m

```
%新种群变异操作
function smnew=mutation(snew,pmutation)
global Bitlength
smnew=snew;
pmm=IfCroIfMut(pmutation);  %根据变异概率决定是否进行变异操作，1 为是
if pmm==1
```

```
    %在[1,Bitlength-1]中随机产生一个变异位
    chb=round(rand*(Bitlength-1))+1;
    smnew(chb)=abs(snew(chb)-1);
end
```

crossover.m

```
%交叉操作
function scro=crossover(population,seln,pc)
global Bitlength
pcc=IfCroIfMut(pc);   %根据交叉概率决定是否进行交叉操作，1 为是
if pcc==1
    %在[1,Bitlength-1]中随机产生一个交叉位
    chb=round(rand*(Bitlength-2))+1;
    scro(1,:)=[population(seln(1),1:chb) population(seln(2),chb+1:Bitlength)];
    scro(2,:)=[population(seln(2),1:chb) population(seln(1),chb+1:Bitlength)];
else
    scro(1,:)=population(seln(1),:);
    scro(2,:)=population(seln(2),:);
end
```

IfCrolfMut.m

```
%判断遗传运算是否需要进行交叉或变异
function pcc=IfCroIfMut(mutORcro)
test(1:100)=0;
L=round(100*mutORcro);
test(1:L)=1;
n=round(rand*99)+1;
pcc=test(n);
```