统计推断在数模转换系统中的应用

45 组　　黄博天 5130309605 翟斯颖 5130309602

摘要：本文通过对拟合算法和遗传算法进行分析，并把它们和本次统计推断的具体内容进行结合，最后提出适于本次课题的具体遗传算法及其实现。最后对提出的算法进行分析，评估结果。本课题实验所涉及到的知识主要是利用 matlab 实现大量数据的拟合处理，完成起来比较耗时耗力。这次实验最大的收获就是对遗传算法的深入了解，并掌握了基于遗传算法的改进方法。

关键字：三次样条插值法，三次多项式拟合，遗传算法。

1. 引言

本课题研究对象为传感特性的校准，即通过有限次的测定，估计出被检测物理量 X 和其读数 Y 的关系。而由一批传感部件样品测定的每个样品的特性数值，为本课题的统计学研究样本，为 Excel 中所给出的数据，共 469 组数据，每组 51 个。

假设实验曲线在连续域中的函数表达式为 f，则 f 均可以泰勒展开为一个多项式函数。在对本课题分析过程中，由讲义中提示，我们只尝试了三次多项式拟合。根据要求，在拟合前不得“偷看”数据，则对其分布一无所知的情况下，分段、穷举来寻找次优解的算法缺乏分段的依据，无从下手，不分段的暴力穷举则没有实现的可能性，因此只得采取启发式算法。

2. 拟合或差值方法

为建立传感器的数学模型，对传感部件个体的输入输出特性进行初步研究后，可采用三次多项式来描述其输入输出性质。通过有限次取样测定，估计其 Y 值与 X 值间一一对应的特性关系，并对整个过程进行成本计算，来评估拟合方案。对于每个点，根据其实验值与计算值，按如下规则进行评估：

$$s_{i,j}=\begin{cases}0 & \text{if } \left|\hat{y}_{i,j}-y_{i,j}\right|\le 0.5\\ 0.5 & \text{if } 0.5<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 1\\ 1.5 & \text{if } 1<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 2\\ 6 & \text{if } 2<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 3\\ 12 & \text{if } 3<\left|\hat{y}_{i,j}-y_{i,j}\right|\le 5\\ 25 & \text{if } \left|\hat{y}_{i,j}-y_{i,j}\right|>5\end{cases}$$

总成本 $S_i=\sum_{j=1}^{51}s_{i,j}+q\cdot n_i$

实施一次单点测定的成本以符号 q 记。本课题指定 q=12。

2.1. 三次样条插值函数（interp1）

假设在[a,b]上测量有 n-1 个点，在[a,b]上划分。△:$a=x_0<x_1<\ldots.<x_{n-1}<x_n=b$，则三次样条插值法所得的函数 S(x)具有如下特征：

1、 在每个小区间$[x_j,x_{j+1}]$上是三次多项式。

2、 在每个节点 $S(x_j)=y_j$。

3、 S(x)在[a，b]二阶导数连续。

调用格式为： yi= interp1(x,y,xi,'method')其中 x，y 为插值点，yi 为在被插值点 xi 处的插值结果；x,y 为向量， 'method'表示采用的插值方法，本文中采用三次样条插值'spline'。

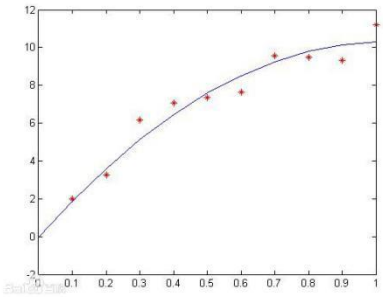
需要注意的是，所有的插值方法都要求 x 是单调的，并且 xi 不能够超过 x 的范围。因此，在使用三次样条差值时，取样测定点应包含首尾两点。

2.2. 曲线拟合 polyfit 函数

曲线拟合：已知离散点上的数据集，即已知在点集上的函数值，构造一个解析函数（其图形为一曲线）使在原离散点上尽可能接近给定的值。polyfit 函数的数学基础是最小二乘法曲线拟合原理，常用于数据拟合。

用法：a=polyfit(xdata,ydata,n)，

其中 n 表示多项式的最高阶数，xdata，ydata 为将要拟合的数据，它是用数组的方式输入。输出参数 a 为拟合多项式 y=a1xˆn+...+anx+a,共 n+1 个系数。



2.3. 两种拟合方式的比较

在初稿中，我们采用了 polyfit 函数进行拟合，认为通过该拟合方式可以直接得到最终的三次表达式。初稿提交后，通过与其他小组的交流，并阅读了老师下发的测试函数，为了使得我们的结论有验证依据，且便于相互比较，最终我们舍弃了 polyfit 函数，改用了 interp1。

通过运行对比，发现按照成本计算标准，对于同一取点方案，polyfit 函数拟合后所得到的总成本要高于 interp1。在对其原理进行研究后不难发现，由于 polyfit 侧重于整个图像的整体状态，而 interp1 研究其局部性质，其成本自然要小于 polyfit。

但若对于这个课题本身而言，我们更倾向于采用 polyfit 来得到最终的表达式。

3. 遗传算法

3.1. 概述

遗传算法(Genetic Algorithm，GA 算法)是由 Holland 教授于 20 世纪 60 年代提出一种模拟达尔文生物进化论中的生物遗传进化过程和自然选择的计算模型.其原理是通过模拟生物进化的过程，使得在每一代中通过对种群进行杂交运算以及变异算子来产生新的个体，通过达尔文生物进化论中的自然选择原则来选择个体得到新的种群，使得每代中最好的个体的适应值越来越好，而个体的平均适应值也会越来越好。

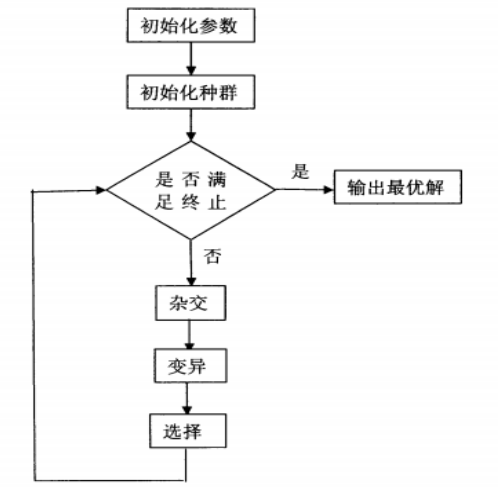
可以把遗传算法的过程看作是一个在多元函数里面求最优解的过程。在这个多维曲

面里面也有数不清的“山峰”，而这些最优解所对应的就是局部最优解。

3.2. 过程

遗传算法的实现过程实际上就像自然界的进化过程那样。首先寻找一种对问题潜在解进行“数字化”编码的方案。然后用随机 数初始化一个种群，种群里面的个体就是这些数字化的编码。接下来，通过适当的解码过程之后，用适应性函数对每一个基因个体作一次适应度评估。用选择函数按照某种规定择优选择。让个体基因交叉变异。然后产生子 代。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不必去了解和操心如何去“找”最优解，只要简单的“否定”一些表现不好的个体就行了。

下图是遗传算法的流程示意图。



3.2.1. 初始化

对于本课题，我们首先需要解决的是如何将取点方案“数字化”的编码为各个“基因”，即如何初始化这些基因。

由我们的生物学知识，人类染色体的编码符号集，由 4 种碱基的两种配合组成。共有 4 种情况，相当于 2 bit 的信息量。与之相类似的，我们选取一个仅由 0、1 构成的编码，来作为我们的“基因”。具体到本课题，由于给出的数据库每行有 51 个点，即可将这个二进制字符串的长度设置为 51,第 n 位代表第 n 个点，0 代表该组取点不包括改点，反之 1 代表包括。

构造上述“基因”后，则初始化为随机设置 0-1 序列。

需要额外注意的是，第 1 位和第 51 位必须为 1。

3.2.2. 杂交

染色体联会的过程中，非姐妹染色单体（分别来自父母双方）之间常常发生交叉，并且相互交换一部分染色体。二进制编码的基因交换过程也非常类似这个过程——随机把其中几个位于同一位置的编码进行交换，产生新的个体。

本文选择单点交换的方法：即随机选取两条染色体，在 1-51 中随机选取一个点作为交叉点，将交叉点后的部分互相交换。

3.2.3. 变异

二进制编码的遗传操作过程和生物学中的过程非常相类似，基因串上的“ 0”或“ 1”有一定几率变成与之相反的“ 1”或“ 0”。

需要额外注意的是，首尾两位不能变异。

3.2.4. 选择

选择对于整个过程颇为关键。这个过程包含适应度函数和选择函数两部分。

3.2.4.1. 适应度函数

自然界生物竞争过程往往包含两个方面：生物相互间的搏斗与及生物与客观环境的搏斗过程。同样，为了使得各个取点方案之间的“搏斗”能有一个依据，即为了衡量这些点的“适应度”，需要一个衡量标准。这就是适应度函数——计算每一个方案的适应度。

在具体实现中，适应度可以直接的取为拟合成本的倒数——因为本题需要寻找拟合成本最小的取点方案。

3.2.4.2. 选择函数

自然界中，越适应的个体就越有可能繁殖后代。但是也不能说适应度越高的就肯定后代越多，只能是从概率上来说更多。为了么来建立这种概率关系呢，我们选用“轮盘赌”的方法。

假设种群数目，某个个体其适应度为 f_i，则其被选中的概率为：

$$P_i = \frac{f_i}{\sum_{i=1}^{n} f_i}$$

与“轮盘赌”游戏很相似，转到其的概率（即轮盘上该区域的角）与其适应度成正比。

3.2.5. 终止条件

为了能结束整个过程，需提前设置杂交代数，即其终止条件。

4. 遗传算法的具体实现

基于上述原理与分析，我们通过 matlab 编程，实现了整个过程。过程包含了拟合计算成本过程 fitnessfun，基因交叉过程 crossfun，基因突变过程 mutationfun，以及整体循环 mainfun。

4.1. 拟合、成本计算

我们选取了三次样条差值法，结合课程要求中所给出的计算标准，取出数据库中的数据进行计算。输入为一个“基因”，即一个一维二进制数组。自然，该函数包含了二进制编码向具体取点方案的转换（类似于基因的翻译）。

4.2. 基因交叉

为了实现随机的两个基因之间随机的进行单点交换，我们采用了 randperm 函数对所有取点方案进行随机编号，再随机选取交叉点。只需注意交叉点选为 1 或 51 是无意义的。

4.3. 基因突变

依据一个较小的突变概率 pm，再随机选取一个点，0 变为 1,1 变为 0。还需注意的是 1 和 51 不能突变。

4.4. 主函数

主函数中包含了初始化过程与循环中的选择过程。

初始化过程即在 1-51 中随机设置 1，共包含 30 组取点方案。为了控制 1 的数量，除了首尾两点外，其余的点依概率设定，尝试发现选为 0.1 较为合适——即平均每个取点方案有 49*0.1+2≈6 个点。

循环过程中，每次记录下本次循环中最佳取点方案与 30 组取点方案的平均成本，输出后便于比较。计算适应度即为 fitnessfun 函数的返回值的倒数。

每次循环都需按照适应度函数选择，以“轮盘赌”的方式选择出下一代。

5. 对于初稿中算法的一点说明

初稿中的 fitnessfun 函数选用的拟合方式为 polyfit，与所给出的测试函数有计算出的结果有所差别。

初稿中的基因直接选用了一组数字，而非一组二进制数组，导致了每次运行都只能在固定的点数的情况下进行，则需尝试多个取点数目，才能得出最后答案。这也是在终稿中改进之处。

此外，初稿中对于选择函数的处理，也与终稿中的处理不同。

终稿中采取通用的“轮盘赌”的方式，而在初稿中，我按照我的想法，在父代中直接随机交叉（以父代为 30 组点为例），但是要选取若干个父代不经交叉直接进入子代（以 8 个为例），这样子代就成为了随机交叉产生的子代 30+不经交叉进入子代 8=38 组，选择过程中，将这 38 组点拟合计算成本，将成本最高的八组删去，剩余的 30 组作为子代，进行下一次循环。

这种方法缺点在于并不是按适应度选择，则收敛速度较慢。我们根据终稿程序的结果检验了一下初稿函数（当然把拟合方式改为了样条插值），发现运行的结果很接近，只是收敛速度很慢。

限于篇幅，我们没有列出修改后的初稿代码（也没有很大的意义），只是将其作为终稿代码的一个对比与参考。

6. 运行结果

由于算法本身的特性，在实际运行中，经常会收敛于一个次优解，而非最优解。因此，需要多次运行，才能得到相对准确的结果。

我们在多次运行、记录后，得到了一组目前已知的最优解：

1 9 21 28 35 44 51

总成本 95.6354

每次运行用时在 990 秒-1000 秒之间。实践中发现适当增加突变频率会增大跳出局部最优解的几率，更易获得更优的解。

附录

mainfun 函数

```
function mainfun()
tic;

loopnum=60; %代数
number=30;  %个体数
pc=0.6;     %杂交概率
pm=0.05;    %变异概率
pop=zeros(number,51);  %繁殖池
temp=zeros(number,51);
fitness=zeros(1,number);  %适应度
for i=1:number      %初始化
    for j=2:50
        if rand<0.1
            pop(i,j)=1;
        end
        pop(i,1)=1;
        pop(i,51)=1;
    end
end

for q=1:loopnum
ave=0;    %平均值
for i=1:number
    if length(find(pop(i,:)))<=2    %点数小于等于 2 时无效
        fitness(i)=0;
    else t=fitnessfun(pop(i,:));
         fitness(i)=1000/t;     %适应度函数
         ave=ave+t;
    end
end
ave=ave/number;
[~,s]=max(fitness);     %最优个体
fitness=fitness/sum(fitness);  %"轮盘赌"过程
best=pop(s,:);
fff=cumsum(fitness);
for i=1:number
     t=find(rand<=fff);
     temp(i,:)=pop(t(1),:);
```

```
end
temp(number,:)=best;
pop=temp;
pop=crossfun(pc,pop);     %交叉
pop=mutationfun(pm,pop);  %变异
best_=zeros(1,1);
j=1;
for i=1:51
    if best(i)==1
        best_(j)=i;
        j=j+1;
    end
end
fprintf('代数:%d\n',q);
fprintf('最优个体: %d\n');
disp(best_);
fprintf('最佳成本:%d\n');
disp(fitnessfun(best));
fprintf('平均成本:%d\n');
disp(ave);
end
toc;
```

fitnessfun 函数

```
function x=fitnessfun(a)
data=csvread('20141010dataform.csv');
x=zeros(1,51);   %数据库中的 x，即 5，5.1，……10.0
y=zeros(469,51);   %数据库中的 y
Y=zeros(469,1);    %取样点的 y
fits=zeros(469,51);   %拟合值

n=0;
b=zeros(1,1);
c=zeros(1,1);
for i=1:51
    if a(i)==1
        n=n+1;
        b(1,n)=i;
    end
```

```
end
for j=1:n
  c(1,j)=data(1,b(1,j));  %取样点的 x
end
x=data(1,:);
for i=1:469
  y(i,:)=data(2*i,:);
  for j=1:n
      Y(i,j)=data(2*i,b(1,j));
  end
end
for j=1:469
  fits(j,:)=interp1(c(1,:),Y(j,:),x(1,:),'spline'); %三次样条差值拟合
end
  err=abs(fits-y);        %计算成本
  err1=(err<=1 & err>0.5);
  err2=(err<=2 & err>1);
  err3=(err<=3 & err>2);
  err4=(err<=5 & err>3);
  err5=(err>5);
  ERR=err1*0.5+err2*1.5+err3*6+err4*12+err5*25;
  x=sum(sum(ERR))/469+12*n;
```

crossfun 函数

```
function [ newpop ] = crossfun(pc,pop)  %% 个体之间进行交叉。
[px,py]=size(pop);
newpop=zeros(px,py);
a=randperm(px);
for i=1:2:(px-1)
    x=a(i);y=a(i+1);
    if(rand<pc)  %% 以 pc 概率交叉。
        cpoint=ceil(rand*49);  %随机交叉点
        newpop(x,:)=[pop(x,1:cpoint),pop(y,cpoint+1:py)];
        newpop(y,:)=[pop(y,1:cpoint),pop(x,cpoint+1:py)];
    else
        newpop(x,:)=pop(x,:);
        newpop(y,:)=pop(y,:);
    end
end
end
```

mutation 函数

```
function [ newpop ]= mutationfun( pm,pop )
px=size(pop,1);
newpop=zeros(size(pop));
for i=1:px
    newpop(i,:)=pop(i,:);
    if(rand<pm)
        mpoint=round(rand*50+1);    %突变点
        newpop(i,mpoint)=1-pop(i,mpoint);
    end
end
end
```