统计推断在数模转换系统中的应用

组号：28　　汪浩洋（组长）5140309355　　张晟嘉 5140309345

摘要：本文以传感器批量生产为背景，寻找校准的优化方案。在确保精度的前提下，运用统计方法结合 MATLAB 对样本中的数据进行分析，采用遗传算法寻找最优选点方案，并通过不断优化变异概率和变异点数，大大丰富初始种群的多样性，得到一个初步的解决方案。
关键词：遗传算法，MATLAB，适应度，曲线拟合，模拟退火

Application of Statistical Inference in DA Inverting System

ABSTRACT: Based on sensor batch production as the background, we are searching an optimized plan for the calibration. In ensuring the precision of the premise, we use the statistical method combined with MATLAB to analyze the sample data, and genetic algorithm to find the most optimal solution. Through continuous optimization of the mutation probability and location, we greatly enrich the diversity of initial population, in order to find an elementary solution.
Key words: Genetic algorithm, MATLAB, sufficiency, curve fitting, Simulation Annealing Algorithm

1、引言

随着科技的发展，传感器得到广泛的应用。要在确保质量的前提下大批量生产传感器，就需要对样品进行单点测定。我们研究的问题是寻找校准工序的优化方案。通过预先抽样研究多个样品的曲线特性，我们经过分析发现样品的曲线有如下特点：

（1） 个体产品的特性曲线形态相似但各不相同。

（2） 特性曲线都是单调递增的。

（3） 每条特性曲线大致可以分为首（左），中，尾（右）三段，三段都不是完全线性的，有一定的弯曲度（且呈现随机的特点）。

（4） 中段的斜率小于首段和尾段的斜率，且中段的起点位置和长度都带有随机性。

因此我们需要选择一种拟合（或插值）的方法，并且选择一种取点方案，依据从样本库获取的单点测定数值，按照选择的拟合方法对数据进行拟合并逐一完成定标，再根据定标结果，结合原始数据测算出成本记录，最后经过方案对比将成本最低的方案定为最终结果。考虑到拟合效果和运行时间，我们决定先将多项式函数作为拟合函数。同时为了能够高效而准确地筛选出测试点，我们先选用遗传算法作为程序算法。之后再选用模拟退火算法，评价其对整个过程以及最终成本的优化程度。

2、拟合与插值方法

2.1 拟合方法

结合此题的实际背景，我们以成本作为评判一个拟合函数优劣的标准，建立如下的评分规则[1]：

（1） 单点定标误差成本按照下式（2-1）评估。

（2） 单点测定成本即实施一次单点测定的成本以符号 q 记。（本课题指定 q=12）

（3） 某一样本的定标成本按照下式（2-2）评估。

（4） 按下式（2-3）计算评估校准方案的总体成本，即使用该校准方案对标准样本数据库中每个样本个体逐一定标，取所有样本个体的定标成本的统计平均。

总成本最低的校准方案则为最优方案。

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

$$S_i=\sum_{j=1}^{51}s_{i,j}+q\cdot n_i \tag{2-2}$$

$$C=\frac{1}{M}\sum_{i=1}^{M}S_i \tag{2-3}$$

在程序实现过程中，数据为浮点数，且标准样本数据库中数据实测值精度只达到 0.01，即数据在更低位上的差别没有比较意义，考虑到数据精度问题，在式（2-1）中，当不等号两端数之差在 1×10^{-4} 以内时，即认为两者相等。由此方式计算出的最终成本，与老师给出的标准成本计算函数（即严格比较两数大小）得出的最终成本相比，在千分之一位上有一定差别。在本文中，均采取如此的计算方法。

在给定测定点的情况下，我们以多项函数作为拟合函数模型，对测定点进行多项式拟合，以拟合结果作为传感器的特性曲线，并计算此时的成本。分别考虑不同次数的多项式，观察拟合结果，得出较优点的范围。

2.2 插值方法

2.2.1 三次样条插值法

样条插值使用一种特殊分段的分段多项式（称为“样条”）进行插值，用三次函数连接相邻的已知数据点（x_i，y_i），插值函数在已知点处连续，并且它的一阶导数和二阶导数也连续。三次样条插值使用三阶多项式样条来实现较小的插值误差，这样就很好地规避了使用高阶多项式所出现的“龙格”现象。

2.2.2 三次插值法

三次插值法即 Hermite 插值法，在给定的节点处插值，得到的函数及其一阶导数都是连续的。

3、用遗传算法寻找最优解

3.1 遗传算法（Genetic Algorithm）

3.1.1 遗传算法概述

遗传算法是借鉴生物界“适者生存，优胜劣汰”原则的启发式搜索算法，模拟基于达尔文生物进化论的自然选择和遗传学原理的生物进化过程。在自然界中，生物通过染色体的遗传、交叉以及变异来保留、产生优秀性状，创造更加适应环境的子代[2]。该算法的主要特点：

（1）遗传算法从问题解的串集开始搜索，而不是从单个解开始，覆盖面大，利于全局择优。

（2）具有内在的隐并性和很好的全局寻优能力，可以同时处理群体中的多个个体，即对搜索空间中的多个解进行评估，减少了陷入局部最优解的风险，同时使算法本身易于实现并行化。

（3）应用范围大，采用概率化的寻优方法，能自动获得和指导优化的搜索空间，自适应地调整搜索方向，不需要确定的规则。

（4）直接对结构对象操作，不存在求导和函数连续性的限定，也不用搜索空间的知识及其他的辅助信息。仅用适应度函数来评价个体，在此基础上进行遗传操作。适应度函数不仅不受连续可微的约束，而且定义域可以任意设定。

（5）遗传算法在适应度函数选择不当的情况下可能收敛于局部最优而不能达到全局最优

所以，算法实现时可以随机选取指定个数的产品作为初始种群。根据达尔文的“优胜劣汰，适者生存”法则，适应度较强的个体保留的可能性较大。然后在保留下来的个体中，它们的染色体以一定概率进行交叉互换、以一种更小的概率发生变异，而具有新染色体的新个体再经过“优胜劣汰，适者生存”以及交叉变异等，如此循环往复一定次数，最终留存下较适合整个环境的数据（即染色体），即为我们所求的解。值得注意的是，遗传算法在全局寻优时，只能找到较为优化的解，而不能保证一定是最优解（即最终留存下来的适应环境的染色体有多条）。

3.2 算法实现

3.2.1 算法流程框图

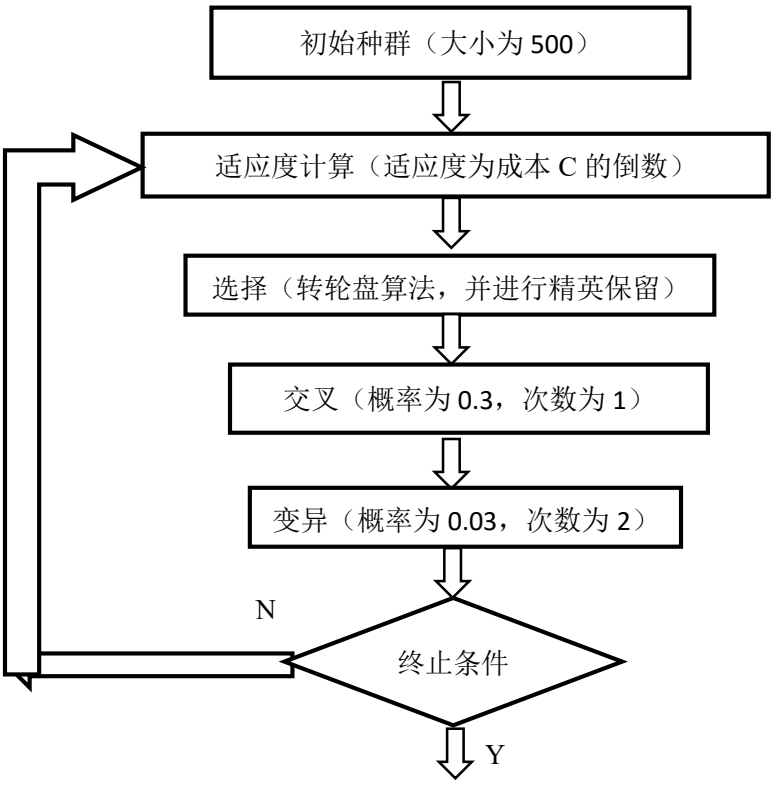

图 3-1 遗传算法流程图

3.2.2 算法分析

（1）初始化：将 51 个数据点按照选（1）与不选（0）进行编码，成为一个 51 位长的染色体。取 pop_size 个个体（pop_size=500），每个个体染色体编码随机给定。

（2）计算适应度：

适应度 f 定义为

$$\frac{1}{f} = C = \frac{1}{400}\sum_{i}^{400} S_i \qquad (3\text{-}1)$$

总成本S_i为

$$S_i = \sum_{j}^{51} S_{i,j} + 12 \times n_i \qquad (3\text{-}2)$$

（$S_{i,j}$ 即单点定标误差成本，n_i为第 i 个染色体中 1 的个数）

为了使不同取点方法的成本差异更明显，我们将成本的倒数设为适应度，作为衡量优劣的标准。适应度的累加值将被计算并保存，作为随机选择个体时进行转轮盘算法（取 0 到适应度总和之间的一个随机数，该随机数落在哪一个适应度累加值区间内，就选择这一个个体）的数据依据，进行选择：将最优个体直接加入下一代（精英保留），再按概率选出其余下一代。

（3）交叉：两两交叉（1 与 2,3 与 4，……），每组染色体进行 n（n=1）次交叉。每次确定交叉概率p_1（p_1=0.3），随机选择交叉位置，然后将交叉点后的基因互换。

（4）变异：对每个个体进行 n（n=2）次单点变异。每次变异时考虑变异概率，确定变异概率p_2（p_2=0.03），随机选定变异位置。

3.3 程序运行情况

表 3-1 用遗传算法获得的初步结果

拟合多项式次数	迭代次数	取点	最低成本（元）
4	400	3 11 23 32 44 51	109.13
5	400	2 8 20 32 44 51	109.70
6	400	2 7 15 27 36 46 51	111.49
6	400	1 8 13 25 37 47 50	114.17

表 3-2 三次样条插值结果

迭代次数	取点	最低成本（元）
200	3 4 17 26 34 43 50	98.63
200	2 8 20 26 36 44 50	96.93
800	3 12 22 31 43 50	94.9

表 3-3 三次插值结果

迭代次数	取点	最低成本（元）
150	4 17 28 37 47	87.55
200	4 16 26 35 48	84.74

可见，我们所选的拟合、插值方法比价令人满意，予以保留。我们需要改进的主要是算法模块。

4、算法优化

4.1 遗传算法的优化

4.1.1 当前遗传算法的问题

在实际运行时，我们发现当前的遗传算法需要非常长时间的运行而且结果不甚理想，比如得到成本 109.13 元所对应的迭代次数为 400 次，运行时间大约在八个小时；然而利用三次插值法得到的成本可以达到 84.75 元而且只需要迭代 200 次。由此我们不得不对自己所写

的遗传算法产生疑惑。经过与助教老师的沟通，我们将问题根源确定在参数设定的不合理上。比如交叉的概率仅设置为 0.3，与实际自然选择中每次产生后代时染色体均会交叉的现实不符。这直接导致了样本点变化的幅度过小，大大增加了找到合适取点方法的步骤数和时间。

4.1.2 遗传算法的改良

我们将遗传算法中的参数做的改动如下表所示：

表 4-1 遗传算法参数改动前后对比

参数	改动前	改动后
交叉概率	0.3	1
交叉次数	1	1
变异概率	0.03	0.015
变异次数	2	2

为观察最低成本的变化情况，以确定程序合适的迭代次数，我们再一次运行了程序。将程序运行 30 次，每一次运行 80 代，并输出每一次运行中每一代的最小成本，得到了一份数据量十分庞大的表格，最后的结果已经十分接近用三次插值法得到的最优成本。经过数据可视化处理，我们得到如下两张图表。

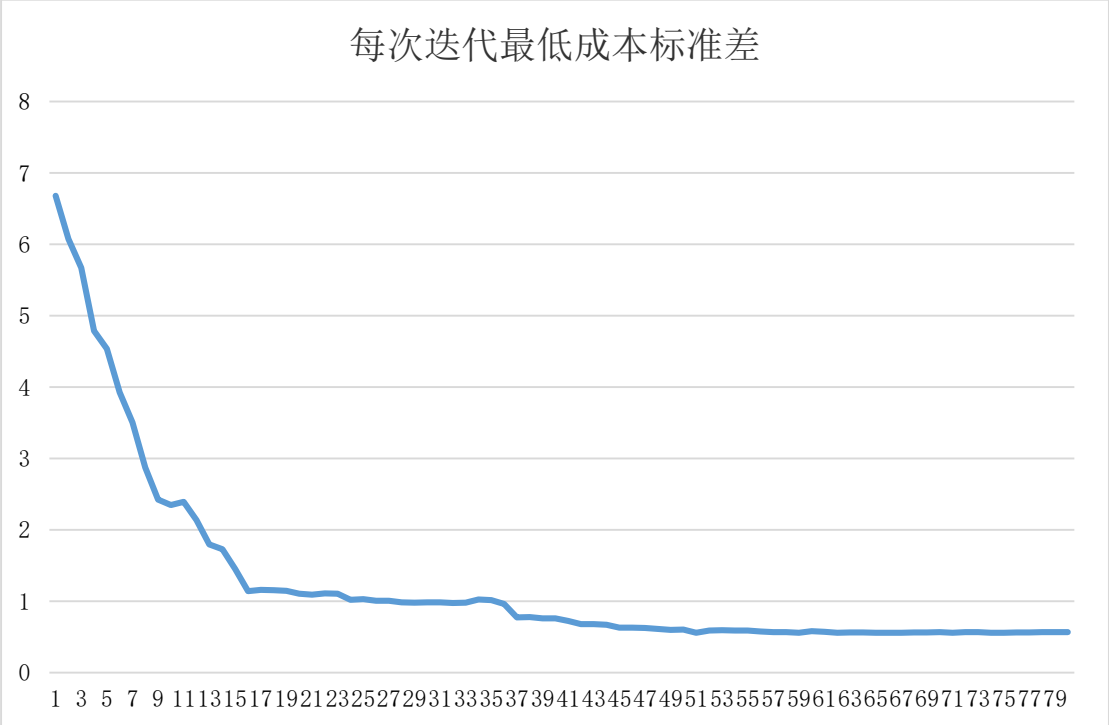

图 4-1 修改后遗传算法每一次迭代最低成本的标准差变化

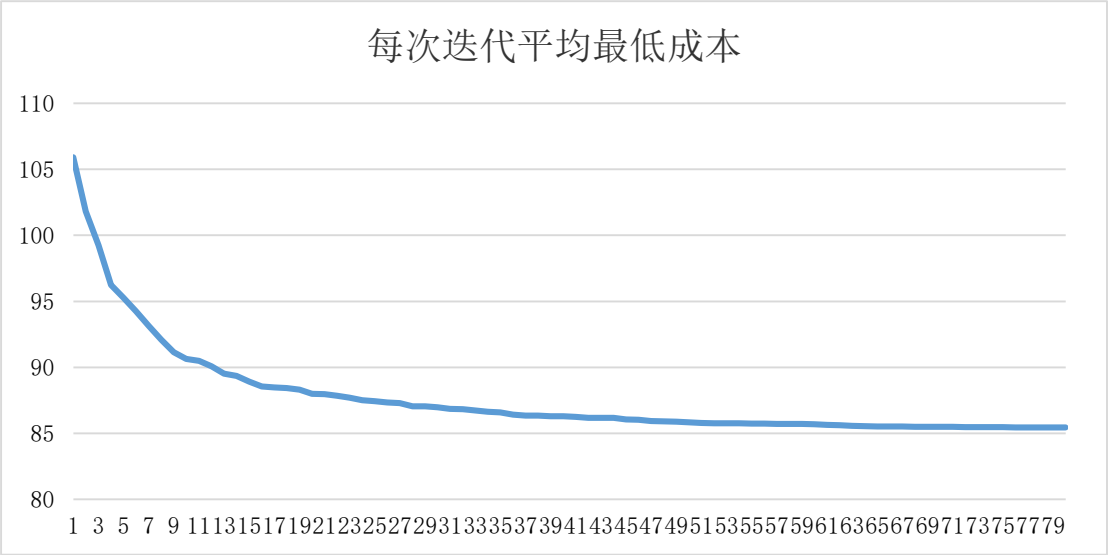

图 4-2 修改后遗传算法每次迭代的平均最低成本

考虑到时间成本，我们认为在实际生产操作中只需运行 50-60 次即可，且 50-60 次后最低成本的标准差较低且趋于稳定，可以认为此时修改后的遗传算法有较高的可靠性（即每次运行的输出结果差别不大，均接近最优情况）。

4.2 模拟退火算法（Simulation Annealing Algorithm）

4.2.1 模拟退火算法综述

模拟退火算法用固体退火模拟组合优化问题，将内能 E 模拟为目标函数 f，温度 T 演化成为控制参数，即得到解决组合优化问题的模拟退火算法：由初始解 S 和控制参数初值 T 开始，对当前解重复地以“产生新解→计算目标函数差→接受或舍弃”进行迭代，并逐步衰减 T 值，以连续 L 个新解都没有被接受为终止条件，算法终止时的当前解即为所得近似最优解[3]。

4.2.2 算法实现流程图

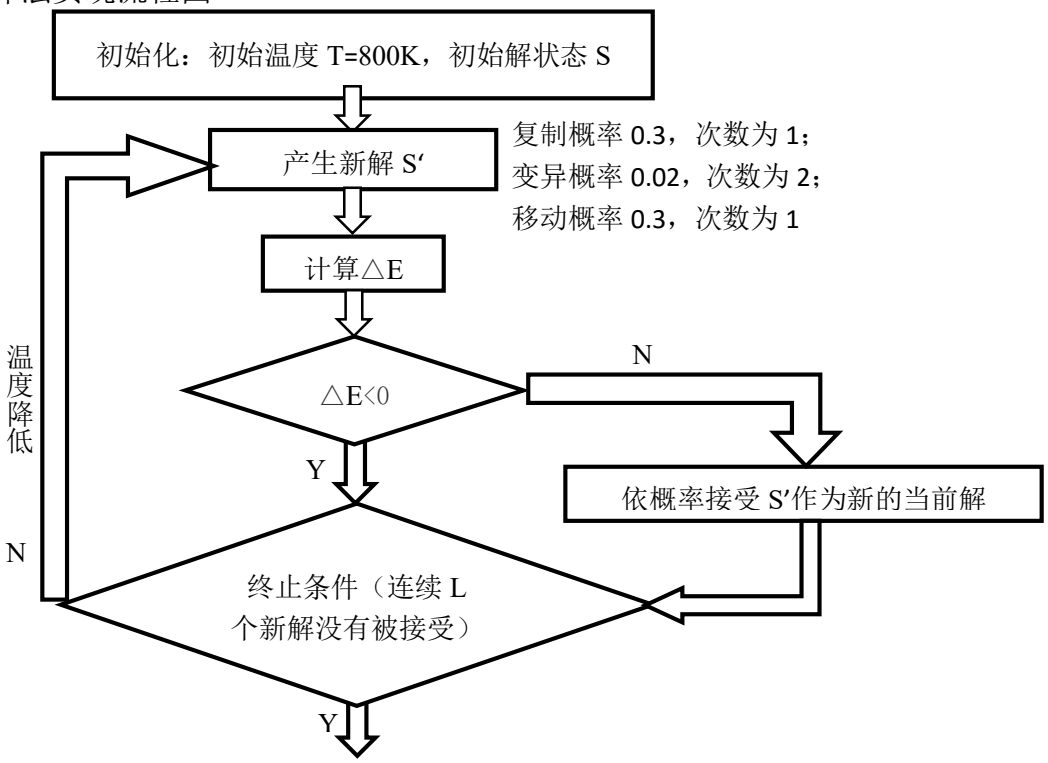

图 4-3 模拟退火算法流程图

需要说明的参数设置：

（1）初始温度 T 需要充分大（T=800K），初始解状态 S 是迭代的起点。

（2）增量△E 即成本的变化。若△E<0 则直接接受 S′为当前解；若△E>0，则以概率 $e^{-\frac{\Delta E}{kT}}$接受 S‘为新的当前解。（k 为玻尔兹曼常量，由于量纲可以任意，因此我们人为给定数值 0.001）

（3）终止条件设定为连续 L（L=6 或 L=8）个新解都没有被接受。

（4）每循环一次，T 乘上一个退火系数 r（r=0.9995），以表明温度降低。

4.2.3 算法分析

我们原先对初始解 S 采用了两种与遗传算法相同的变化方式：变异和交换。然而结果并不稳定：运行时间很长且不稳定，最后得到的成本变化幅度也很大。而且无论我们如何改变参数值都不能使过程和结果变得理想。

考虑到在现实过程中，物体每次状态的变化（即产生新解 S′）幅度不会很大，而我们使用遗传算法中产生新解的方式时，每次新解与原解对应的特性曲线差别都较大，导致每次的能量变化（即成本之差）很大，不符合实际情况，因此造成上述不稳定。于是我们对变化方式做了改进。

初始解状态 S 为随机确定的一组 51 位二进制码，这一步同遗传算法中的初始化一样。

然后对初始解进行改变：

（1）变异：同遗传算法中的变异做法相同。

（2）复制：随机选取原解中的一段二进制码和复制的起始点，将从起始点开始的对应长度的一段二进制码覆盖为选取的二进制码。

（3）移动：随机选定一个二进制码为 1 的位置，先将其变为 0，在随机将其相邻的不为 1 的一个二进制码变为 1（即“移动”该二进制码的位置）。若其左右都为 1 则不再改变。举例说明：“010”改变后为“100”或“001”，“110”改变后为“101”，“111”改变后为“101”。这样做可以使得测量点的位置左右偏移，使得每次状态改变对应的能量改变较小，且避免算法结果落入局部最优解。

以上三种方法都改变完成后，计算新得到的成本，若比原来的小则取定为新的当前解，否则依概率$e^{-\frac{\Delta T'}{kT}}$接受它为新的当前解。然后判断终止条件，判定是继续循环还是退出。

最后，我们只需要设置终止条件中新解连续不被接受的次数 L 即可运行程序。

4.2.4 运行结果

表 4-2 用模拟退火算法获得的结果（L=6）

最后温度（℃）	大致时间（s）	取点	最低成本（元）
498.69	180	4 17 26 36 48	85.7195
534.06	120	4 16 25 35 49	85.4110
560.33	120	4 16 26 35 48	84.7442
753.40	18	5 9 21 27 36 50	90.9862
387.39	240	5 16 25 35 48	85.5377
491.76	180	5 17 26 35 49	85.6372
665.19	60	5 16 26 35 48	85.0102
496.95	180	4 17 26 35 48	84.9110
623.31	60	4 18 26 36 48	86.0462
371.27	240	3 16 26 35 48	85.1767

表 4-3 用模拟退火算法获得的结果（L=8）

最后温度（℃）	大致时间（s）	取点	最低成本（元）
169.22	540	4 16 26 35 48	84.7442
284.39	360	3 16 26 35 48	85.1767
168.63	540	4 16 26 35 48	84.7442
177.28	540	4 16 26 35 48	84.7442
59.86	960	4 16 26 35 48	84.7442
129.69	660	4 16 26 35 48	84.7442
294.97	360	3 16 26 35 48	85.1767
168.63	540	4 16 26 35 48	84.7442
177.28	540	4 16 26 35 48	84.7442
59.86	960	4 16 26 35 48	84.7442

可见，通过模拟退火算法我们也成功得到了理想的取点方案。

5、结论

在通过两种不同的算法成功获得了理想的最优成本取点方案之后，我们对这两种方法进行比较，以取在实际情况下的更优者。我们主要从时间和稳定度上对两种方法进行评价。

首先，我们对遗传算法进行分析，给定迭代次数的遗传算法运行时间大致相同，所以我们归纳整理出迭代次数与运行时间之间的关系。

表 5-1 遗传算法的迭代次数与时间数据

迭代次数	大致运行时间（min）
10	5.17
20	8.82
30	12.92
40	17.23
50	22.98
60	29.5
70	34.6
80	37.0

通过分析数据，我们作出了遗传算法迭代次数与运行时间的图像，并进行了线性拟合。

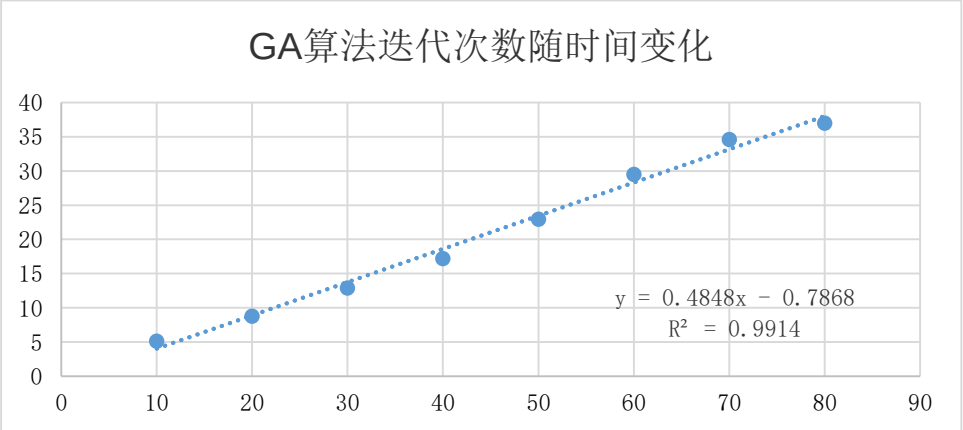

图 5-1 遗传算法迭代次数与运行时间的关系图像

随后，我们整理了模拟退火算法所得成本与运行时间的数据。

表 5-2 模拟退火算法所得成本与时间（L=6）

成本（元）	大致运行时间（s）
85.7195	180
85.411	120
84.7442	120
90.9862	18
85.5377	240
85.6372	180
85.0102	60
84.911	180
86.0462	60
85.1767	240

表 5-3 模拟退火算法所得成本与时间（L=8）

成本（元）	大致运行时间（s）
84.7442	540
85.1767	360
84.7442	540
84.7442	540
84.7442	960
84.7442	660
85.1767	360
84.7442	540
84.7442	540
84.7442	960

通过以上数据我们得到以下结论：

终止条件为 L=6 时，模拟退火算法的成本平均值相当于用遗传算法迭代 48 次时的平均成本，但此时模拟退火的成本标准差比遗传算法高出一倍（意味着有概率得出不理想的结果），同时由于模拟退火算法自身的原因，运行时间波动很大。

终止条件为 L=8 时，得到的平均成本低于用遗传算法迭代 80 次时的平均成本，同时标准差也低很多（这两项意味着此时模拟退火算法的可靠性非常高），虽然它仍然有着运行时间波动很大的问题，但此时出现过的最大运行时间 16 分钟仍小于用遗传算法迭代 40 次的运行时间。

但是另外值得注意的是，我们所用的模拟退火算法的解的改变方式已经针对该问题的实际情况进行过专门调整，以使得其改变方式很符合模拟退火的物理模型要求，因此算法的性能较高。但是在某些其他实际情况下，若数据之间的关系与评判好坏的关键值之间的关系非常复杂、晦涩或不是很明了，就不容易为之构造专门的解的改变方式，此时，遗传算法更好的通用性就能体现出来，变得更加适用。

6、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]简单遗传算法 Matlab 实现 http://www.cnblogs.com/biaoyu/archive/2011/12/03/2274604.html
[3]模拟退火算法
http://blog.csdn.net/acdreamers/article/details/10019849

附录 1：遗传算法程序代码文本清单(GA 对应的 test_ur_answer 代码从第 16 页开始)

GA.m

```
pop_size = 500; %每一代的个体数
iterate_time = 800; %迭代次数
poly_n = 3; %拟合多项式的次数
cross_rate = 1; %交叉概率
cross_time = 1; %交叉次数
variation_rate = 0.015; %变异概率
variation_time = 2; %变异次数

chromo_size = 51; %染色体长度
sample_size = 400; %样本个数

datas = load('datas.txt');

fid = fopen('val.txt', 'wt');


cost_change = [];

for q = 1:1

max_fit = 0;
cost = 0;

pop = init(pop_size, chromo_size); %pop 为种群中每个个体的染色体

pop_new = pop;

%计算适应度
fit_v = fit_calc(pop_size, chromo_size, sample_size, pop_new, datas, poly_n); %fit_v 为个体适应度的列向量

for k = 1:iterate_time
%选择
pop_new = choose(pop_size, chromo_size, pop_new, fit_v);

%交叉
pop_new = cross(pop_size, chromo_size, pop_new, cross_rate, cross_time);

%变异
pop_new = variation(pop_size, chromo_size, pop_new, variation_rate, variation_time);
```

```
%计算适应度
fit_v = fit_calc(pop_size, chromo_size, sample_size, pop_new, datas, poly_n); %fit_v 为个体适应度的列向量

max_a = 1; %寻找最大值
max_v = fit_v(1);
for m = 1:pop_size
    if fit_v(m) > max_v
        max_v = fit_v(m);
        max_a = m;
    end
end

if max_v > max_fit
    max_fit = max_v;
    cost = 1/max_fit;
    my_answer = [];
    for n = 1:chromo_size
        if pop_new(max_a, n) == 1
            my_answer = [my_answer, n];
        end
    end
end

cost_change(k, q) = cost;

end

fprintf(fid, '%d: ', q);
fprintf(fid, '%d ', my_answer);
fprintf(fid, '\n');

end
```

init.m

```
function pop = init(pop_size, chromo_size)
%初始化种群染色体

pop = round(rand(pop_size, chromo_size) * 0.7);

end
```

fit_calc.m

```
function fit_v = fit_calc(pop_size, chromo_size, sample_size, pop_new, datas, poly_n)
%计算种群每个个体的适应度并返回

costs = zeros(pop_size,1); %每个个体的成本
x = zeros(sample_size, chromo_size); %记录每个个体需要拟合的 x
y = zeros(sample_size, chromo_size); %记录每个个体需要拟合的 y

sample_costs = zeros(sample_size, chromo_size); %个体每个样本每个测试点的误差成本
count = 0; %个体所取样本数
points = 5:0.1:10;
delta = [];

%函数拟合
for k = 1:pop_size %对每个个体
    count = 0;
    for m = 1:chromo_size %读取该个体需要记录的样本数据
        if abs(pop_new(k, m) - 1) < 1e-4
            count = count + 1;
            for n = 1:sample_size
                x(n,count) = datas(2 * n - 1,m);
                y(n,count) = datas(2 * n,m);
            end
        end
    end
    for m = 1:sample_size %对个体的每个样本进行拟合,计算每个样本的误差成本
        delta = abs(interp1(x(m,1:count), y(m,1:count), points, 'pchip') - datas(2 * m,:)); %该样
本每个数据点的误差
        for n = 1:chromo_size
            if delta(n) < 0.4 || (delta(n) - 0.4) < 1e-4 %浮点数比较大小
                sample_costs(m,n) = 0;
            elseif delta(n) < 0.6 || (delta(n) - 0.6) < 1e-4
                sample_costs(m,n) = 0.1;
            elseif delta(n) < 0.8 || (delta(n) - 0.8) < 1e-4
                sample_costs(m,n) = 0.7;
            elseif delta(n) < 1 || (delta(n) - 1) < 1e-4
                sample_costs(m,n) = 0.9;
            elseif delta(n) < 2 || (delta(n) - 2) < 1e-4
                sample_costs(m,n) = 1.5;
            elseif delta(n) < 3 || (delta(n) - 3) < 1e-4
                sample_costs(m,n) = 6;
            elseif delta(n) < 5 || (delta(n) - 5) < 1e-4
                sample_costs(m,n) = 12;
            else sample_costs(m,n) = 25;
```

```
            end
        end
    end
    costs(k,1) = sum(sum(sample_costs)) / sample_size + 12 * count; %该个体的最终成本
end

fit_v = zeros(pop_size,1);
for k = 1:pop_size
    fit_v(k,1) = 1 / costs(k,1);
end
```

choose.m

```
function pop_new = choose(pop_size, chromo_size, pop, fit_v)
%  依概率选择个体

pop_new = zeros(pop_size, chromo_size);

max = 1; %适应度最大的个体
fit_v_sum = zeros(pop_size,1); %适应度的累加值
fit_v_sum(1) = fit_v(1);
for k = 2:pop_size
    if fit_v(max) < fit_v(k)
        max = k;
    end
    fit_v_sum(k) = fit_v(k) + fit_v_sum(k - 1);
end

pop_new(1,:) = pop(max,:); %精英保留

r = rand(pop_size,1) * fit_v_sum(pop_size); %选择的位置
for k = 2:pop_size
    if r(k) < fit_v_sum(1) || (r(k) - fit_v_sum(1)) < 1e-4
        pop_new(k,:) = pop(1,:);
    else
        first = 1; last = pop_size;
        while (last - first - 1) > 1e-4 %二分法找选择的个体
            middle = round((first + last) / 2);
            if r(k) < fit_v_sum(middle) || (r(k) - fit_v_sum(middle)) < 1e-4
                last = middle;
            else
                first = middle;
            end
        end
```

```
            pop_new(k,:) = pop(last,:);
        end
    end
```

cross.m

```
function pop_new = cross(pop_size, chromo_size, pop_new, cross_rate, cross_time)
%两两交叉

for k = 1:2:pop_size %第 k 个和第 k+1 个交叉
    for m = 1:cross_time %进行 cross_time 次交叉
        if rand < cross_rate
            cross_pos = round(rand * chromo_size);
            if cross_pos == 0 || cross_pos == 1
                continue;
            end
            for n = cross_pos:chromo_size
                temp = pop_new(k,n);
                pop_new(k,n) = pop_new(k + 1,n);
                pop_new(k + 1,n) = temp;
            end
        end
    end
end
```

variation.m

```
function pop_new = variation(pop_size, chromo_size, pop_new, variation_rate, variation_time)
%变异

for k = 1:pop_size %对每一个个体考虑变异
    for m = 1:variation_time %多点变异
        if rand < variation_rate
            variation_pos = round(rand * chromo_size);
            if variation_pos == 0
                continue;
            end
            pop_new(k,variation_pos) = 1 - pop_new(k,variation_pos);
        end
    end
```

```
end
```

test_ur_answer.m

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

my_answer=[2,15,24,35,49];%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);
```

mycurvefitting.m

```
function y1 = mycurvefitting( x_premea,y0_premea )

x=[5.0:0.1:10.0];

%三次样条插值
%y1=interp1(x_premea,y0_premea,x,'spline');

%三次插值
y1=interp1(x_premea,y0_premea,x,'pchip');

%n 次多项式拟合
%poly_n = 3;
%poly = polyfit(x_premea, y0_premea, poly_n);
%y1 = polyval(poly, x);

end
```

附录 2：模拟退火算法代码文本清单(SA 对应的 test_ur_answer 代码从第 23 页开始)
SA.m

```
copy_rate = 0.3; %复制概率
copy_time = 1; %复制次数
variation_rate = 0.02; %变异概率
variation_time = 2; %变异次数
move_time = 1; %移动次数
move_rate = 0.3; %移动概率

pop_size = 1; %每一代的个体数
chromo_size = 51; %染色体长度
sample_size = 400; %样本个数

count_end = 8; %终止条件
count = 0; %退火连续失败次数

k = 0.001; %波尔兹曼常数
r = 0.9995; %每次降温系数
T = 800; %初始温度

datas = load('datas.txt');


%初始化
pop = init(pop_size, chromo_size); %pop 为种群中每个个体的染色体

pop_new = pop;

%计算适应度
fit_v = fit_calc(pop_size, chromo_size, sample_size, pop_new, datas); %fit_v 为个体适应度的列向量

while count <= count_end

    %变异
    pop_new_temp = change(pop_new, chromo_size, copy_rate, copy_time, variation_rate, variation_time, move_time, move_rate);

    %计算适应度
    fit_v_temp = fit_calc(pop_size, chromo_size, sample_size, pop_new_temp, datas); %fit_v 为个体适应度的列向量

    %退火
```

```
        answer = annealing(fit_v, fit_v_temp, k, T);

        if answer == 1
            T = r * T;
            pop_new = pop_new_temp;
            count = 0;
            fit_v = fit_v_temp;
        else
            count = count + 1;
        end

end

my_answer = [];
for k = 1:chromo_size
        if pop_new(k) == 1
            my_answer = [my_answer, k];
        end
end
```

init.m

```
function pop = init(pop_size, chromo_size)
%初始化种群染色体

pop = round(rand(pop_size, chromo_size) * 0.7);

end
```

fit_calc.m

```
function fit_v = fit_calc(pop_size, chromo_size, sample_size, pop_new, datas)
%计算种群每个个体的适应度并返回

costs = zeros(pop_size,1); %每个个体的成本
x = zeros(sample_size, chromo_size); %记录每个个体需要拟合的 x
y = zeros(sample_size, chromo_size); %记录每个个体需要拟合的 y

sample_costs = zeros(sample_size, chromo_size); %个体每个样本每个测试点的误差成本
count = 0; %个体所取样本数
```

```
points = 5:0.1:10;
delta = [];


%函数拟合
for k = 1:pop_size %对每个个体
    count = 0;
    for m = 1:chromo_size %读取该个体需要记录的样本数据
        if abs(pop_new(k, m) - 1) < 1e-4
            count = count + 1;
            for n = 1:sample_size
                x(n,count) = datas(2 * n - 1,m);
                y(n,count) = datas(2 * n,m);
            end
        end
    end
    for m = 1:sample_size %对个体的每个样本进行拟合,计算每个样本的误差成本
        delta = abs(interp1(x(m,1:count), y(m,1:count), points, 'pchip') - datas(2 * m,:)); %该样本每个数据点的误差
        for n = 1:chromo_size
            if delta(n) < 0.4 || (delta(n) - 0.4) < 1e-4 %浮点数比较大小
                sample_costs(m,n) = 0;
            elseif delta(n) < 0.6 || (delta(n) - 0.6) < 1e-4
                sample_costs(m,n) = 0.1;
            elseif delta(n) < 0.8 || (delta(n) - 0.8) < 1e-4
                sample_costs(m,n) = 0.7;
            elseif delta(n) < 1 || (delta(n) - 1) < 1e-4
                sample_costs(m,n) = 0.9;
            elseif delta(n) < 2 || (delta(n) - 2) < 1e-4
                sample_costs(m,n) = 1.5;
            elseif delta(n) < 3 || (delta(n) - 3) < 1e-4
                sample_costs(m,n) = 6;
            elseif delta(n) < 5 || (delta(n) - 5) < 1e-4
                sample_costs(m,n) = 12;
            else sample_costs(m,n) = 25;
            end
        end
    end
    costs(k,1) = sum(sum(sample_costs)) / sample_size + 12 * count; %该个体的最终成本
end

fit_v = zeros(pop_size,1);
for k = 1:pop_size
    fit_v(k,1) = costs(k,1);
```

```
end
```

change.m

```
function pop_new_temp = change(pop_new, chromo_size, copy_rate, copy_time, variation_rate, variation_time, move_time, move_rate)

%pop_new 部分交换、变异

pop_new_temp = pop_new;

%复制
for m = 1:copy_time
    if rand < copy_rate
        copy_pos_1 = round(rand * chromo_size);
        copy_pos_2 = round(rand * chromo_size);
        copy_pos_b = min(copy_pos_1, copy_pos_2);
        copy_pos_e = max(copy_pos_1, copy_pos_2);
        if (copy_pos_b ~= copy_pos_e)
            copy_pos_new = round(rand * (copy_pos_b + chromo_size - copy_pos_e));
            pop_new_temp(copy_pos_new + 1:copy_pos_new + copy_pos_e-copy_pos_b) = pop_new(copy_pos_b + 1:copy_pos_e);
        end
    end
end

%变异
for m = 1:variation_time
    if rand < variation_rate
        variation_pos = round(rand * chromo_size);
        if variation_pos ~= 0
            pop_new_temp(variation_pos) = 1 - pop_new_temp(variation_pos);
        end
    end
end

%左右移动 1 的位置，如果左右都是 1，则将选择位置零
for m = 1:move_time
    if rand < move_rate
        move_pos = round(rand * chromo_size);
        while ((move_pos == 0) || (move_pos < chromo_size && pop_new(move_pos) == 0))
            move_pos = move_pos + 1;
```

```
            end

            pop_new_temp(move_pos) = 0;
            if move_pos == 1
                pop_new_temp(2) = 1;
                continue;
            end
            if move_pos == chromo_size
                if pop_new(chromo_size) ~= 0
                    pop_new_temp(move_pos-1) = 1;
                end
                continue;
            end

            if pop_new(move_pos-1) == 1
                pop_new_temp(move_pos+1) = 1;
            elseif pop_new(move_pos+1) == 1
                pop_new_temp(move_pos-1) = 1;
            elseif rand < 0.5
                pop_new_temp(move_pos-1) = 1;
            else
                pop_new_temp(move_pos+1) = 1;
            end
        end

end
```

annealing.m

```
function answer = annealing(fit_v, fit_v_temp, k, T)
%退火
if fit_v_temp < fit_v
    answer = 1;
else
    if exp((fit_v - fit_v_temp) / (k * T)) > rand
        answer = 1;
    else answer = 0;
    end
end
```

test_ur_answer.m

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

my_answer=[4,16,26,35,48];%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);
```

mycurvefitting.m

```
function y1 = mycurvefitting( x_premea,y0_premea )

x=[5.0:0.1:10.0];

%三次样条插值
%y1=interp1(x_premea,y0_premea,x,'spline');

%三次插值
y1=interp1(x_premea,y0_premea,x,'pchip');

%n 次多项式拟合
%poly_n = 3;
%poly = polyfit(x_premea, y0_premea, poly_n);
%y1 = polyval(poly, x);

end
```