# 统计推断在数模转换系统中的应用

第 07 组：刘伟鹏 满毅

**摘要**：本课程设计论文基于科创[3A]的实验结果数据，利用统计推断的方法，探究某输入输出具有非线性关系的传感器的校准方案。由于取点方案数量较大，若采取穷举搜索法时间复杂度过高，无可操作性能，所以采用启发式搜索的拟合方案。遗传算法是一种模仿自然界生物遗传机制的启发式搜索方法，在解决优化问题上具有很高的效率，能够得到一个最佳方案的近似解，本文以遗传算法为优化方法，采用多项式曲线拟合以及三次样条拟合的方式来寻求最优方案。

**关键词**：数模转换系统，统计推断，遗传算法，多项式拟合，三次样条拟合

**ABSTRACT**: This thesis is based on the experiment data of the course EI310(3A). We rely on the statistical inference method to explore whether there exists a plan that certain inputs and outputs can be adjusted in a nonlinear fitting. Because of the vast amounts of data and the complexity of time in exhaustion method, we use heuristic search algorithm. Genetic algorithm is a method stimulating the nature that are efficient and can lead to close results. This thesis takes genetic algorithm as optimization, which adopts polynomial curve fitting and three times spline fitting for results.

**Key words**：Statistical Inference; Genetic Algorithm, Polynomial fitting, Polynomial fitting

## 1 课程设计内容与目标

在生产中，经常需要对产品进行检测校准。由于每次检测具有一定的检测成本，取点数目过少又会影响到检测效果。为了寻找一个兼顾成本和检测效果的检测方案，常需要建立一个量化的评价方案，对各检测方案进行评估，最终找到较优方案。

### 1.1 问题的提出

本次课题的探究对象为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。

### 1.2 标准样本数据库

本次课程设计的数据库来自科创 3A 数据，数据样本总数为 469 组，每组样本包含 51 组输入输出值。本次课程设计基于该标准样本数据库进行拟合方案的选取。

### 1.3 评价函数

为了对各方案的效果、成本进行比较，我们选择以下评价函数：

- 单点定标误差成本

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

说明：公式 1-1 中 $s_{i,j}$ 为单点误差成本，$y_{i,j}$ 表示第 i 个样本之第 j 点 Y 的实测值，$\hat{y}_{i,j}$ 表示定标后得到的估测值（读数）。

- 单点测定成本

实施一次单点测定的成本以符号 q 记。本课题指定 q=12。

- 某一样本个体的定标成本

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

说明：样本 i 总的定标成本按式 1-2 计算，式中 $n_i$ 表示对该样本个体定标过程中的单点测定次数。

- 校准方案总体成本

按式（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。

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

说明：式 1-3 中，C 为基于标准样本数据库评价校准方案所算得的该方案总体成本，M 为标准样本数据库中的样本总数。

总体成本较低的校准方案，认定为较优方案。

## 1.4 遗传算法

遗传算法是计算机科学人工智能领域中用于解决最优化的一种搜索启发式算法，是进化算法的一种。这种启发式通常用来生成有用的解决方案来优化和搜索问题。进化算法最初是借鉴了进化生物学中的一些现象而发展起来的，这些现象包括遗传、突变、自然选择以及杂交等。[1]

在遗传算法执行过程中，主要准备步骤有对目标进行编码，产生初始化种群。准备步骤结束后开始进行循环，主要有适应度计算，选择，变异（包括交叉、突变）。达到迭代代数或者满足收敛条件时循环结束并输出结果。图 1-1 为遗传算法的具体流程：

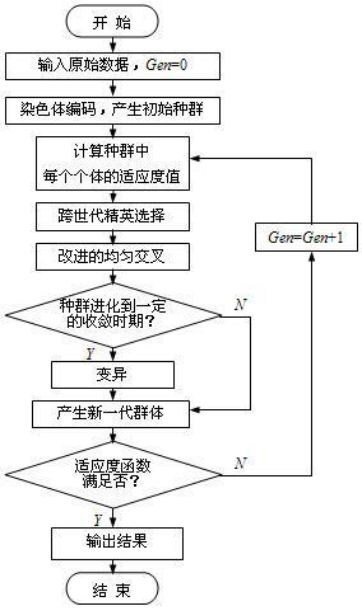


图 1-1 遗传算法流程图

# 2. 寻找拟合模型并尝试优化

## 2.1 通过作图法观测数据关系大致情况

根据课程说明，本课程所探究的输入输出量呈非线性，故首先我们对标准数据库内的输入输出进行了求平均值，并将平均值做出散点图，借此希望观测出输入输出关系的一些基本特征。图 2-1 是通过 Excel 做出的散点图。

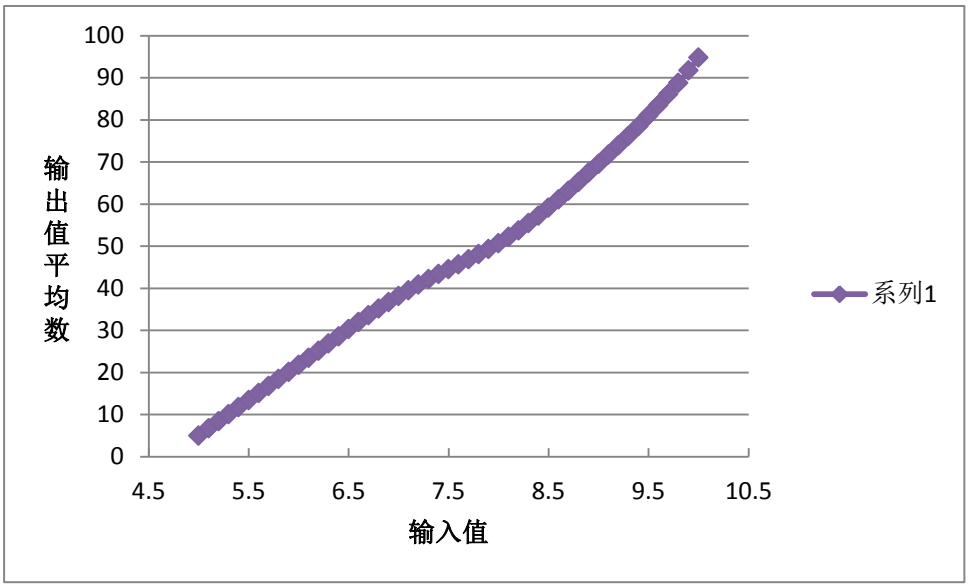


图 2-1 标准数据库样本平均值散点图

## 2.2 拟合方法的选择

通过对平均值散点图进行观察，我们可以看出曲线有两次明显的凹凸性的变化，所以很自然的，我们首先想到了三次多项式拟合。

## 2.3 三次多项式拟合

### 2.3.1 拟合原理及三次多项式拟合在 MATLAB 中的使用方法

所谓曲线拟合是指设法找出某条光滑的曲线,它能最佳地拟合数据。在曲线拟合时,并不要求拟合曲线一定要经过每一个数

据点。其思想是使它能反映这些离散数据的变化趋势，使数据点的误差平方和最小。也就是已知一组测定的数据(例如 N 个点(xi,yi))去求得自变量 x 和因变量 y 的一个近似解析表达式 y=φ(x)。若记误差 δi=φ(xi)- yi，i=1，2,…，N ，则要使误差的平方和最小,即要求公式 2-1 和最小

$$Q=\sum_{i=1}^{n}\delta_i^2 \qquad (2\text{-}1)$$

这就是常用的最小二乘法原理。[2]

我们采用 MATLAB 提供的内置函数 Polyfit 进行曲线拟合。

### 2.3.2 三次多项式拟合的基本参数设置

首先，我们采用了表 2-1 参数进行三次多项式的拟合：

| | |
|---|---|
| **编码方法** | 二进制编码 |
| **初始化方法** | 随机取点，随机数 rand 小于 0.5 取该点 |
| **种群数** | 100 |
| **适应度函数** | 成本取倒数，即 1/C |
| **迭代次数** | 50 |

| | |
|---|---|
| **交换频率及方法** | 0.8，单点交换 |
| **突变频率** | 0.02 |
| **采用精英机制？** | 是 |

表 2-1：三次多项式拟合的初始化条件

### 2.3.3 三次多项式拟合的结果

50 代过后，我们得到的拟合结果数据见表 2-2。图 2-2 是每一代平均成本变化曲线。

| | |
|---|---|
| **迭代次数** | 50 |
| **运算时间** | 3474.719 秒 |
| **最佳方案成本** | 136.7313　　（第 50 代出现） |
| **最佳取点方案** | 2,12,18,28,39,43,49 |
| **最后一代平均成本** | 181.8667 |

表 2-2：三次多项式拟合结果

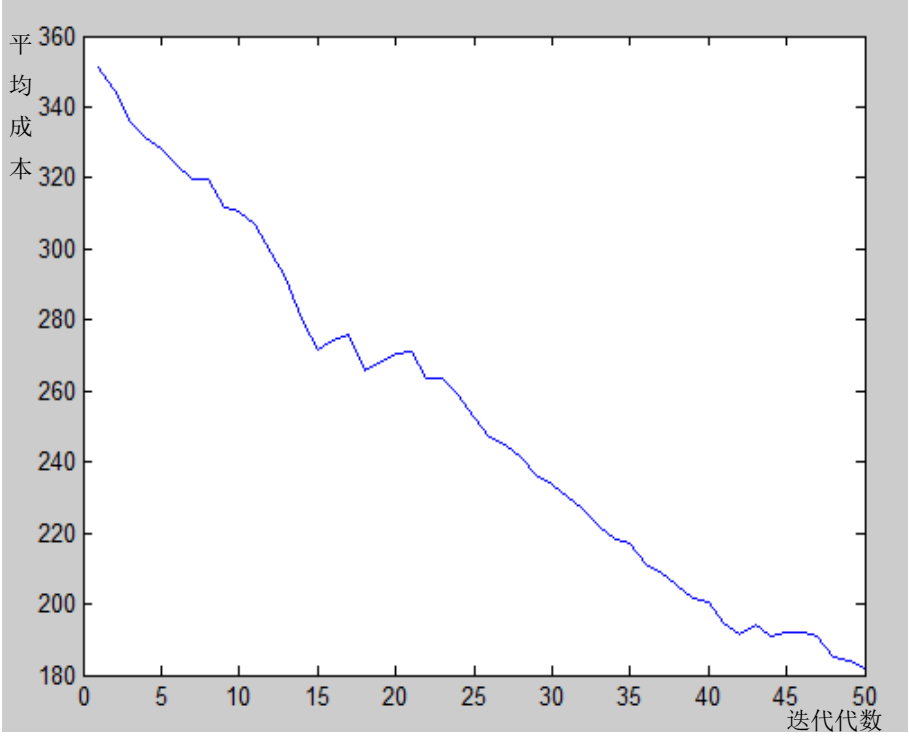


图 2-2：平均成本变化曲线

### 2.3.4 结果分析及优化方案的提出

通过观察我们可以看出，经过 50 代迭代后最佳成本仍然不是很理想。通过观察平均成本变化曲线可以看出，此时种群成本变化率依然很大，说明远没有到达收敛的条件。通过对此时最优取点情况进行分析，我们可以看出最佳取点数目大概在 7 点左右，而通过我们选取的初始化方案初始种群的取点数的数学期望是 25 个，明显过多。所以我们先想到的了一种较为简化的取点方案，即降低阈值到 0.1 左右，在取 4 个初始点 1，15,30,51。

其它函数不变的情况下我们得到了表 2-3 的结果：

| | |
|---|---|
| **迭代次数** | 50 |
| **运算时间** | 2760.511 秒 |
| **最佳方案成本** | 126.4360　　（第 8 代出现） |
| **最佳取点方案** | 1,15,30,41，51 |

| | |
|---|---|
| **最后一代平均成本** | 126.4360 |

表 2-3：改进过后三次多项式拟合结果

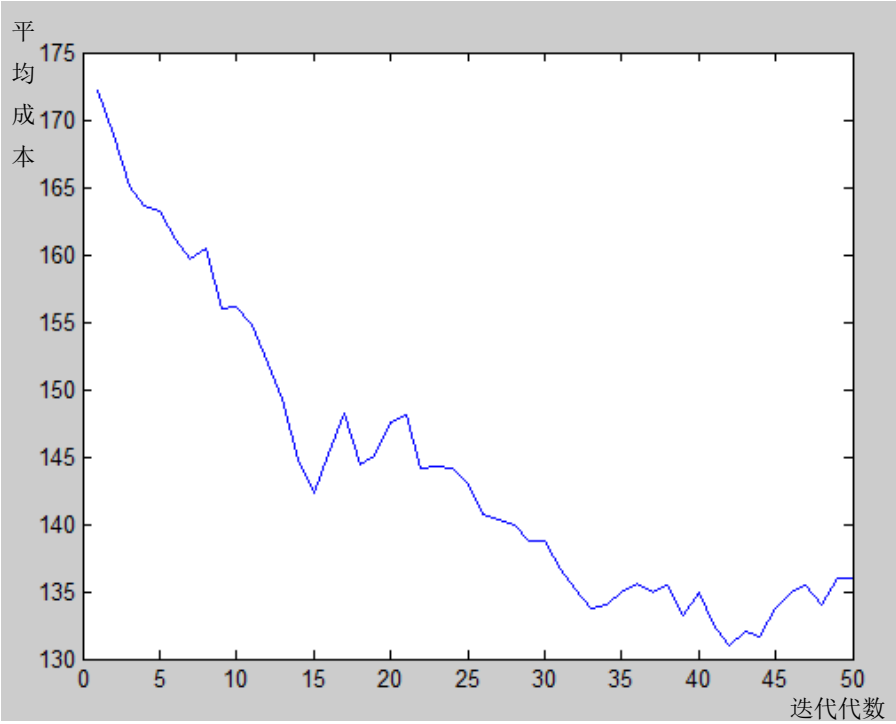


图 2-3：改进过后的平均成本变化曲线

对初始化方法进行改进过后，我们明显看出初始化平均成本大幅降低，且收敛速度明显加快，至迭代至 30 代之后，平均成本函数下降速度已经明显减缓，说明已经趋于收敛。但是这是否说明我们的改动可以称为一种改动呢？我们观察到最优取点方案里竟然有 4 个点就是我们初始化设定好的点，这就不得不让我们产生怀疑，是否我们为了加快收敛速度所采取的方案造成了遗传算法的早熟，进入了局部最优。

更重要的是，我们发现其实三次多项式拟合其实不是一种特别好的拟合方式，于是我们选择了重新选择一种更好的拟合模型进行探究。

## 2.4 三次样条拟合

### 2.4.1 样条拟合及三次样条拟合在 MATLAB 中的使用方法

由于三次多项式进行优化后拟合效果也不是很好，我们采取了三次样条插值进行拟合。样条插值是使用一种名为样条的特殊分段多项式进行插值的形式。由于样条插值可以使用低阶多项式样条实现较小的插值误差，这样可以避免了使用高阶多项式所出现的龙格现象。[3]

在 MATLAB 中，我们可以使用 Spline 函数进行三次样条拟合。

### 2.4.2 三次样条拟合的基本参数设置

针对前文探究，我们发现若采用事先设点的方案有最后得到局部最优的风险，所以在进行三次样条插值的时候，我们试图通过实验去探究我们的这个猜想。为了最终能够得到理想的插值方案，我们进行了两组不同初始化设置的测试（除初始化方案外其余参数同 2-1），分别为：

A 组：纯随机取点，且阈值为 0.2。

B 组：预先设定四点（同 2.3.4 中的取点方案）。

另外，通过我们的初步尝试，我们发现在种群总平均成本达到 100 时，种群就趋于收敛，于是我们选取平均成本达到 100 作为循环结束条件。

### 2.4.3 结果分析

经过分别对 A，B，C 组分别测试，我们得到了如表 2-4 的结果

| | A | B |
|---|---|---|
| **达到条件终止的代数** | 93 | Fail |
| **运行时间** | 3693.009 | Fail |
| **最佳取点方案** | 3,11,20,27,35,43,50 | Fail |
| **最佳方案成本** | 94.2687 | Fail |

表 2-4

实验的结果还是比较超乎我们的预期的，首先是未作初始化设置点的三次样条在进行了 93 代迭代之后，总平均成本就下降到了 100 以下，并且最佳成本为 94.2687，我们认为这个结果已经比较让我们满意。然而我们事先设置好点的方案在初始几代成本非常低，但是到 20 代之后最优已经停留在 100 以上且多代不变化，平均成本则在 105 左右波动，在运行了 100 代时，收敛在 101 左右，不符合我们预先设计的收敛条件。这说明虽然预先设点看上去能加快收敛速度，但是有很大的可能性导致遗传算法进入局部最优而与更好的结果想去甚远，所以当你想得到更好的结果的时候，这种优化方法是不可取的。

## 3 初步的结论及感想

经过之前的探究我们发现，在拟合模型方面，三次样条模型较三次多项式拟合对于本课题标准数据库的拟合效果更好，在初始值设定方面，事先选点虽然有利于加速收敛，但是不利于得到全局最优，所以总体来说不如随机取点。并且，我们得到了一个成本为 94.2687 的取点方案（详见表 2-4）。

通过本次统计推断课程，我们首次参与这种授课自学探究为主模式，在最初参与讲座学习时简直是一头雾水，到最终解题时已经掌握了一些统计推断和遗传算法的基本概念和使用方法，在调试代码和尝试修改参数的过程中也锻炼了我们解决问题的能力，在论文撰写中也习得了一些论文书写的基本要求。不过限于时间限制我们有很多优化的想法并未真正实施，难免遗憾。另外特别感谢袁焱老师耐心的指导讲解。

## 参考文献

[1] 维基百科：遗传算法词条 [OL]
http://zh.wikipedia.org/wiki/%E9%81%97%E4%BC%A0%E7%AE%97%E6%B3%95
[2] 唐家德, 基于 MATLAB 的非线性曲线拟合[J] 计算机与现代化，2008 年第 6 期
[3] 维基百科：样条拟合词条[OL]
http://zh.wikipedia.org/wiki/%E6%A0%B7%E6%9D%A1%E6%8F%92%E5%80%BC

## 【附录】遗传算法代码

### 1. 入口函数 run_ga

```
function run_ga()
elitism = true;%选择精英操作
pop_size = 100;%种群大小
chromo_size = 51;%染色体大小
generation_size =50;%迭代次数
cross_rate = 0.8;%交叉概率
mutate_rate = 0.02;%变异概率

m=zeros(1,51);

[m,n,p,q] = GeneticAlgorithm(pop_size, chromo_size, generation_size, cross_rate,
mutate_rate,elitism);
disp(m);
disp(n);
disp(p);

clear;
```

### 2. 主循环函数 GeneticAlgorithm

```
 %遗传算法主函数
%pop_size: 输入种群大小
%chromo_size: 输入染色体长度
%generation_size: 输入迭代次数
%cross_rate: 输入交叉概率
%cross_rate: 输入变异概率
%elitism: 输入是否精英选择
%m: 输出最佳个体
%n: 输出最佳适应度
%p: 输出最佳个体出现代
%q: 输出最佳个体自变量值

function [m,n,p,q] = GeneticAlgorithm(pop_size, chromo_size, generation_size, cross_rate,
mutate_rate, elitism)

global G ; %当前代
global the_cost;
global fitness_value;%当前代适应度矩阵
global best_fitness;%历代最佳适应值
global fitness_avg;%历代平均适应值矩阵
global best_individual;%历代最佳个体
global best_generation;%最佳个体出现代
global data;%从 EXCEL 中导出的矩阵
global Xdata;
global Ydata;
global sampleSize;

sampleSize=10;
best_costave=zeros(generation_size);
data=xlsread('data.xlsx');
for i=1:sampleSize
```

```
    Xdata(i,:)=data(2*i-1,:);
    Ydata(i,:)=data(2*i,:);
end
the_cost=zeros(generation_size,pop_size);
best_individual=zeros(1,51);
fitness_avg = zeros(generation_size,1);
counter=zeros(pop_size,sampleSize);
fitness_value(pop_size) = 0.;
best_fitness = 0.;
best_generation = 0;


initilize(pop_size, chromo_size);%初始化     %初始化成功

for G=1:generation_size
    fitness(pop_size, chromo_size);  %计算适应度，计算出本代适应度，存放在
fitness_value 中
    rank(pop_size, chromo_size);  %对个体按适应度大小进行排序
    selection(pop_size, chromo_size, elitism);%选择操作
    if 1/fitness_avg<100
        break;
    end
    crossover(pop_size, chromo_size, cross_rate);%交叉操作
    mutation(pop_size, chromo_size, mutate_rate);%变异操作
end

plotGA(generation_size);%打印算法迭代过程
m = best_individual;%获得最佳个体
n = (1/best_fitness)^(1/2);%获得最佳适应度
p = best_generation;%获得最佳个体出现代

tmpmax = 0;
q=0;
for j=1:chromo_size
    if best_individual(j) >tmpmax
            tmpmax=best_individual(j);
            q=j;
    end
end

clear i;
clear j;
```

3. 初始化函数 initialize

```
%初始化种群
%pop_size: 种群大小
%chromo_size: 染色体长度

function initilize(pop_size, chromo_size)
global pop;

for i=1:pop_size
    key=rand;
    pop(i,1)=1;
    pop(i,15)=1;
```

```
        pop(i,51)=1;
        pop(i,30)=1;
        for j=1:51
            if rand<0.1
                pop(i,j)=1;
            end
        end
    end
    clear k;
    clear i;
    clear j;
    clear c;
```

4. 适应度函数 fitness

```
    function    fitness(pop_size, chromo_size)
    global fitness_value;
    global pop;
    global G;
    global Xdata;
    global Ydata;
    global sampleSize;
    global the_cost;


    fitness_value=zeros(1,pop_size);
    cost=zeros(1,sampleSize);

    for i=1:pop_size
        counter=0;
        for n=1:51
            if(pop(i,n)==1)
                counter=counter+1;
            end;
        end
        temp_x=zeros(1,counter);
        temp_y=zeros(1,counter);
        f=zeros(1,51);

        for j=1:sampleSize
            c=1;
            x=[0 0 0 0];
            for n=1:51
                if(pop(i,n)==1)
                    temp_x(c)=Xdata(j,n);
                    temp_y(c)=Ydata(j,n);
                    c=c+1;
                end
            end
            %三次多项式用这个
            %p=polyfit(temp_x,temp_y,3);
            %f=polyval(p,Xdata(j,:));

            %三次样条插值法用下面这个
            f=spline(temp_x,temp_y,Xdata(j,:));
            cost(j)=costfun(f,Ydata(j,:))+(c-1)*12;
        end
```

```
        costall=0;
        for j=1:sampleSize;
            costall=costall+cost(j);
        end
        costave=costall/sampleSize;
        the_cost(G,i)=costave;
        fitness_value(i)=1/costave^2;

    end

    clear i;
    end
```

5. 选择函数 selection

```
    %轮盘赌选择操作
    %pop_size: 种群大小
    %chromo_size: 染色体长度
    %cross_rate: 是否精英选择

    function selection(pop_size, chromo_size, elitism)
    global pop;
    global fitness_table;%fitness_table 为当前代 前 i 个染色体适应度的和

    for i=1:pop_size
        r = rand * fitness_table(pop_size);
        first = 1;
        last = pop_size;
        mid = round((last+first)/2);%round 是四舍五入函数
        idx = -1;
        while (first <= last) && (idx == -1)
            if r > fitness_table(mid)
                first = mid;
            elseif r < fitness_table(mid)
                last = mid;
            else
                idx = mid;
                break;
            end
            mid = round((last+first)/2);
            if (last - first) == 1
                idx = last;
                break;
            end
        end

        for j=1:chromo_size
            pop_new(i,j)=pop(idx,j);
        end
    end
    if elitism
        a=2;
    else
        a=1;
    end
    for i=a:pop_size
```

```
    for j=1:chromo_size
        pop(i,j) = pop_new(i,j);
    end
end

clear i;
clear j;
clear pop_new;
clear first;
clear last;
clear idx;
clear mid;
```

6. 排序函数 rank

```
%对个体按适应度大小进行排序，并且保存最佳个体
%pop_size: 种群大小
%chromo_size: 染色体长度

function rank(pop_size, chromo_size)
global fitness_value;%当前代 cost
global fitness_table;
global fitness_avg;
global best_fitness;
global best_individual;
global best_generation;
global pop;
global G;
global the_cost;


for i=1:pop_size
    fitness_table(i) = 0.;
end
min = 1;
temp = 1;
temp1(chromo_size)=0;
for i=1:pop_size
    min = i;
    for j = i+1:pop_size
        if fitness_value(j)>fitness_value(min);
            min = j;
        end
    end
    if min~=i
        temp = fitness_value(i);
        fitness_value(i) = fitness_value(min);
        fitness_value(min) = temp;
        tmp=the_cost(G,i);
        the_cost(G,i)=the_cost(G,min);
        the_cost(G,min)=tmp;
        for k = 1:chromo_size
            temp1(k) = pop(i,k);
            pop(i,k) = pop(min,k);
            pop(min,k) = temp1(k);
        end
    end
```

```
end

%for i=1:pop_size;
    %disp (fitness_value(i));
%上面完成的是将 fitness 向量从小到大排序，交换顺序时需要同时交换染色体和染色体的值
%disp "本代最佳";
%disp ( fitness_value(pop_size));
%=测试成功

for i=1:pop_size
    if i==1
        fitness_table(i) = fitness_table(i) + fitness_value(i);
    else
        fitness_table(i) = fitness_table(i-1) + fitness_value(i);
    end
end
fitness_table;
fitness_avg(G) = fitness_table(pop_size)/pop_size;
disp (G);
disp (1/fitness_avg(G))
disp (1/fitness_value(1))
%disp(fitness_value(:))
if fitness_value(1) > best_fitness
    best_fitness = fitness_value(1);
    best_generation = G;
    for j=1:chromo_size
        best_individual(j) = pop(1,j);
    end
end


clear i;
clear j;
clear k;
clear min;
clear temp;
clear temp1;
```

## 7. 选择函数 selection

```
%轮盘赌选择操作
%pop_size: 种群大小
%chromo_size: 染色体长度
%cross_rate: 是否精英选择

function selection(pop_size, chromo_size, elitism)
global pop;
global fitness_table;%fitness_table 为当前代 前 i 个染色体适应度的和

for i=1:pop_size
    r = rand * fitness_table(pop_size);
    first = 1;
    last = pop_size;
```

```
        mid = round((last+first)/2);%round 是四舍五入函数
        idx = -1;
        while (first <= last) && (idx == -1)
            i f r > fitness_table(mid)
                first = mid;
            elseif r < fitness_table(mid)
                last = mid;
            else
                idx = mid;
                break;
          end
          mid = round((last+first)/2);
          if (last - first) == 1
                idx = last;
                break;
          end
       end

    for j=1:chromo_size
        pop_new(i,j)=pop(idx,j);
    end
end
if elitism
    a=2;
else
  a=1;
end
for i=a:pop_size
    for j=1:chromo_size
        pop(i,j) = pop_new(i,j);
end
end

clear i;
clear j;
clear pop_new;
clear first;
clear last;
clear idx;
clear mid;
```

8. 交换函数 crossover

```
function cross(pop_size, chromo_size, cross_rate)
global pop;
for i=1:2:pop_size
       if(rand < cross_rate)
            cross_pos = round(rand * chromo_size);
            if or (cross_pos == 0, cross_pos == 1)
                continue;
            end
            for j=cross_pos:chromo_size
                temp = pop(i,j);
                pop(i,j) = pop(i+1,j);
                pop(i+1,j) = temp;
            end
       end
```

```
end


clear i;
clear j;
clear temp;
clear cross_pos;
```

9. 突变函数 mutation

```
%已经修改过



%单点变异操作
%pop_size: 种群大小
%chromo_size: 染色体长度
%cross_rate: 变异概率
function mutation(pop_size, chromo_size, mutate_rate)
global pop;

for i=1:pop_size
    if rand < mutate_rate
        mutate_pos = round(rand*chromo_size);
        if mutate_pos == 0
            continue;
        end
        if(pop(i,mutate_pos)==1)
            pop(i,mutate_pos)=0;
        else pop(i,mutate_pos)=1;
        end

    end
end

clear i;
clear mutate_pos;
```

10. 绘图函数 plotGA

```
%打印算法迭代过程
%generation_size: 迭代次数
function plotGA(generation_size)
global fitness_avg;
cost_avg=zeros(1,generation_size);
for i=1:generation_size;
    cost_avg(i)=(1/fitness_avg(i))^(1/2);
end

x = 1:1:generation_size;
y = cost_avg;
plot(x,y)
```