统计推断在数模转换系统中的应用

组号55　　　　谢宗哲　　　学号：5137119003　　　　　饶津瑞　　　学号：5140309384

参考申明：

统计推断课程，2015年秋季学期第55组，成员：谢宗哲 学号5137119003, 饶津瑞 学号5140309384, 在报告编写过程中，以下方面参考了往届报告，现列表说明：

主要参考项目	说明
代码方面	《统计推断在数模模数转换中的应用》，陈丹旸，2014 年秋季学期，组号 66 在该组报告附录提供的程序代码基础上，进行了少量修改。
取点分析	《统计推断在数模转换系统中的应用》，沈扬，2014 年秋季学期，组号 61
数学模型，算法描述方面，包含流程图	《统计推断在数模转换系统中的应用》，方斯瑞，2014 年秋季学期，组号 59 参考了该组报告的算法描述文字，及流程图，数学模型

除了以上注明的参考内容，和报告中列出的引用文献注解，本报告其他部分都不包含任何其他个人或集体已经发表或撰写过的作品成果。

摘要：本报告介绍了统计推断在数模转换系统中的应用：在研究为某电子产品内部传感器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）方案时，首先通过对取点数目的分析，决定使用五个特征点，由此将曲线分成三段分别拟合，根据三段曲线的不同特征进行取点作为母本，再通过模拟退火算法得到五个特征点，最后用三次样条插值拟合，并用成本函数进行评估。

关键词：分段拟合，五特称点法，模拟退火算法，三次样条插值法

1.研究对象

1.1 研究对象

监测模块的组成框图如图1.1。传感器部件监测的对象物理量以符号Y 表示；传感部件的输出电压信号用符号X 表示，该电压经模数转换器（ADC）成为数字编码，并能被微处理器程序所读取和处理，获得信号^Y作为Y的读数（监测模块对Y的估测值）[1]。

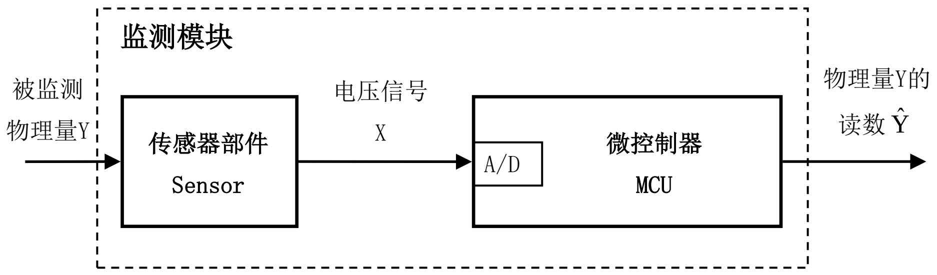

图 1.1 监测模块组成框图

1.2 取点数目的分析

就该系统而言，测定51组数据后，系统特性已经被充分定标，若对每个样本都进行51组数据的完整测定，既费时间又耗成本，没有直接的工程实用意义。我们需要得到一种根据较少的点数就能得到较好的Y-X函数关系的一种实现方法。

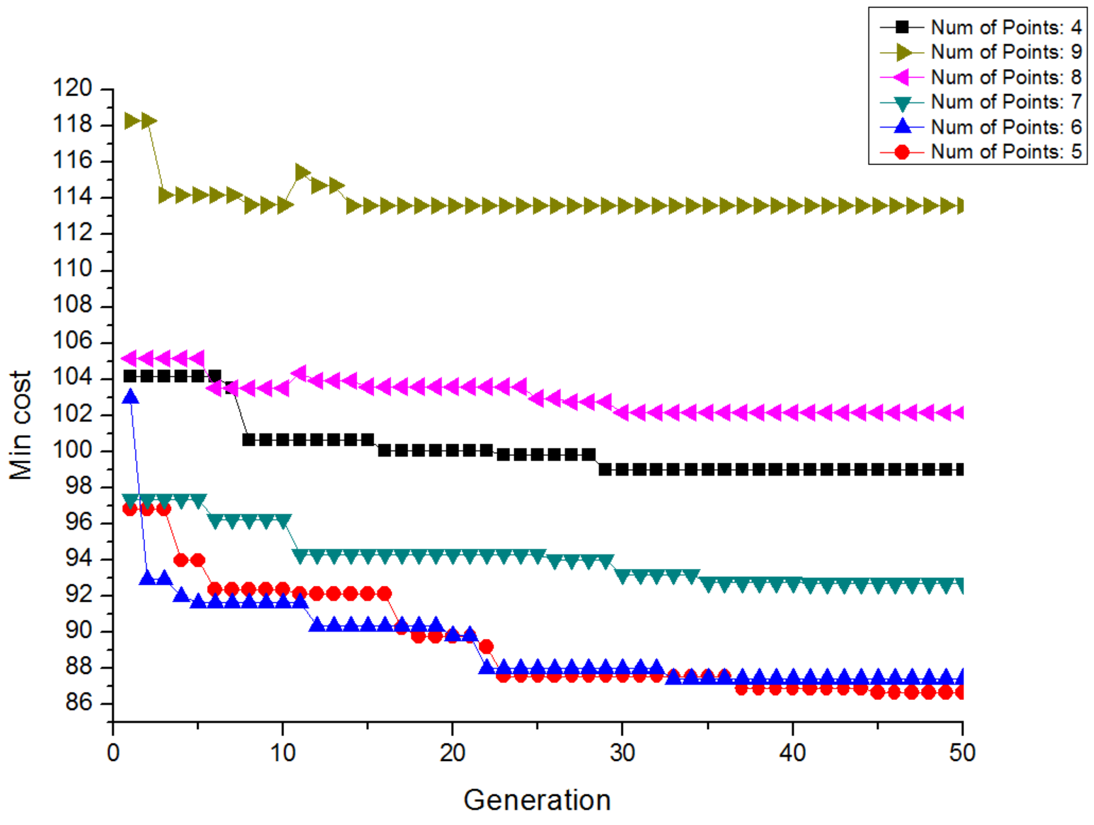

图1.2 取点数目分析曲线图

经过后期多次拟合尝试的结果确定，我们在这里认为取5个点为能达到预期效果，且计算量合适。

1.3 取点区间的划分

为了大致确定这7个点应该存在的范围，故需要对Y-X特性做大致的划分，以便使7个点的选取不至于过于集中，使得7个点在相应的范围内变动，这样也可以大大减少接下来的计算量。

区间划分应该具有普适性，即能够对469组实验数据同时具有划分的意义。由于X的坐标

一样，故以X为横坐标，绘制Y-X图，并通过对Y求二阶差分来判断电压X划分点。

由图1.2可以看出，图像大致分为三段，即1-12点为一段，13-26点为一段，26-51点为一段.

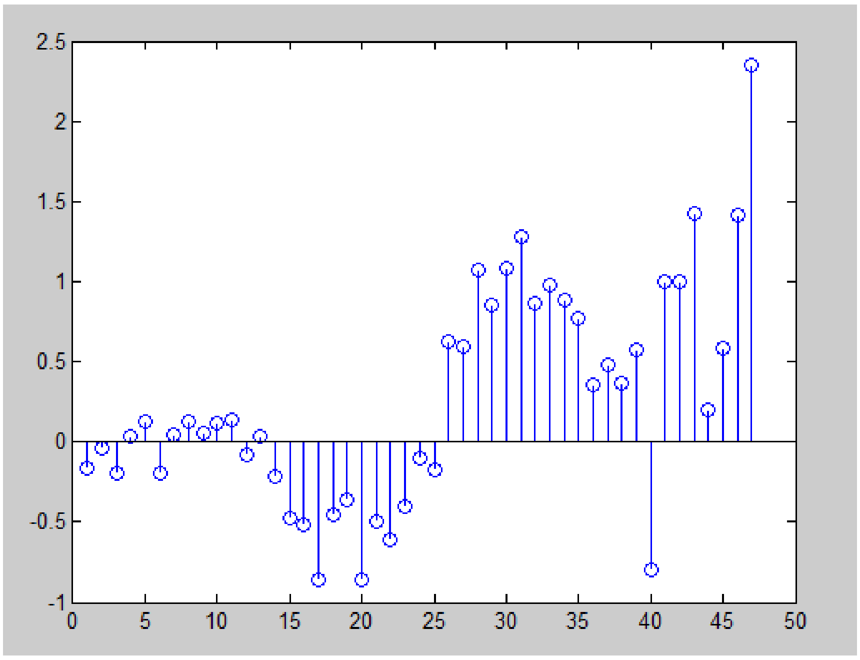

图1.2 Y-X的二阶差分离散图

由图1.2可以看出，图像大致分为三段，即1-12点为一段，13-26点为一段，26-51点为一段。

由此可确定本次实验5个点取点方法：第一段2个点，第二段1个点，第三段2个点。

2. 数学模型

2.1 多项式拟合

2.1.1 概述

实验所提供的数据曲线如下图所示绝大部分都很接近一个多项式曲线。所以我们尝试用多项式拟合曲线。

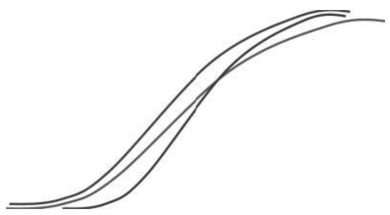

图2.1 实验数据拟合曲线

2.1.2 高次拟合结果

图2.2 为18点17次拟合图,经过每一个点,但曲线已经特别扭曲,而实验中有51个数据,扭曲程度会更高，这种拟合虽然拟合误差最小，但是很值得怀疑。故放弃了这一方法。

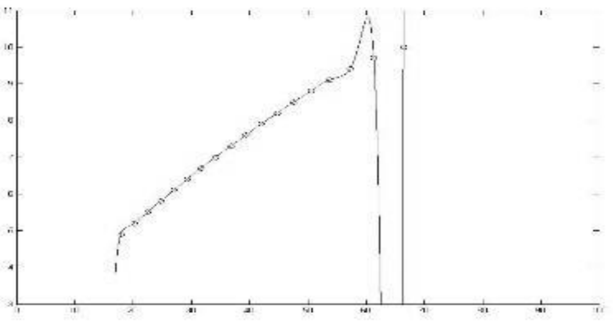

图2.2 多项式拟合函数曲线

2.2 线性插值

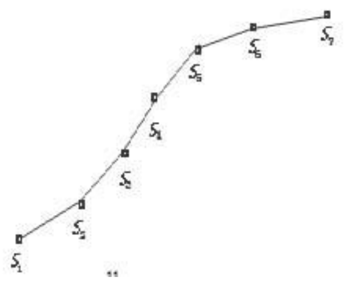

图2.3 线性插值方法拟合曲线

假定观测了7组数值（7个数据点）

假设我们已知坐标(x0,y0)与(x1,y1)，要得到[x0,x1]区间内某一位置x在直线上的y值。

我们得到（y-y0）(x1-x0)=(y1-y0)(x-x0)。假设方程两边的值为α，那么这个值就是插值系数—从x0到x的距离与从x0到x1距离的比值。由于x值已知，所以可以从公式得到α的值：α=(x-x0)/(x1-x0)，同样α=(y-y0)/(y1-y0)。这样，在代数上就可以表示成为：y = (1- α)y0 + αy1，或者y = y0 + α(y1 - y0)，这样通过α就可以直接得到y。

根据《课程设计课题和要求》的传感器特性来看，线性插值对于可能是不完全线性的实验数据而言，线性插值对其会造成较大误差，故同样放弃了这一方法、

2.3 三次样条插值法

2.3.1 概述

假设在[a,b]上测量有n-1个点，在[a,b]上划分。△:a=x0<x1<….<xn-1<xn=b，则三次样条插值法所得的函数S(x)具有如下特征[1]：

1、在每个小区间[xj,xj+1]上是三次多项式。

2、在每个节点S(xj)=yj。

3、S(x)在[a，b]二阶导数连续。

因为S(x)在[a，b]上二阶导数连续，所以在每个节点，由连续性得S(xj-0)=S(xj+0); S’(xj-0)=S’(xj+0); S’’(xj-0)=S’’(xj+0)。共3n-3个条件；除此之外，由插值可得n+1个条件，仍需两个，这两个条件有边值得出：

1、给定断点x0,xn处的一阶导数值

2、给定断点x0,xn处的二阶导数值（特别的S’’(x0+0)=0, S’’(xn+0)=0称为自然边值条件）

3、周期性便捷条件：即以b-a为周期，S’’(x0+0)= S’’(xn+0)，S’(x0+0) =S’(xn+0)。

这样求出的三次曲线比2.1中的多项式拟合应该要精确许多。

2.3.2三次样条插值法相比于七点法的优点

三次样条插值法相比于七点法对所测量得到的信息的应用足够充分，最后所得的曲线必

过每个测量点，在每两个测量点之间都有自己的三次多项式。七点法最后所得的式子不一定过测量点。即用三次样条插值法所得的拟合方程，更加贴近真实的测量结果，同时也没有丧失多阶导数连续性。不过时间长效率低是一个不足。

2.3.3 Matlab程序实现

类似于线性，Matlab中也提供了三次样条插值函数spline，只需要调用语句

FitY = interp1（DataX，DataY，FitX，‘spline’）即可，其中 DataX和DataY分别是待拟合数据，FitY是根据拟合出的函数代入FitX值而得到的函数值。

3 取特征点

为了减少数据量，我们可以通过缩小范围的方法找到拟合点的大致位置，只在拟合点附近进行嵌套循环，最终找到最优解。

3.1 选择遗传算法

首先我们考虑用“暴力穷举法”，但是总共要尝试 A 次不同组合，求解十分复杂，哪怕对于计算机来说运算量也十分巨大，这种“暴力穷举”法被放弃了。其中，

$$A=C_{51}^{7} \qquad (3\text{-}1)$$

我们还考虑了将 53 个点划分为 1-11,12-22,23-32,33-42,43-51 五个区间，每个区间内随机取一个点，其他两个点由其他 46 个点随机选取产生，即需要计算的次数 B 为

$$B=11^2\times10^2\times9\times C_{46}^{2}=1.13\times10^8 \qquad (3\text{-}2)$$

仍旧毫无改观。我们决定使用遗传算法解决该“N-P 问题”。

遗传算法是计算数学中用于解决最佳化的搜索算法，是进化算法的一种。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。遗传算法通常实现方式为一种计算机模拟。对于一个最优化问题，一定数量的候选解（称为个体）的抽象表示（称为染色体）的种群向更好的解进化。传统上，解用二进制表示（即0 和1 的串）。进化从完全随机个体的种群开始，之后一代一代发生。在每一代中，整个种群的适应度被评价，从当前种群中随机地选择多个个体（基于它们的适应度），通过自然选择和突变产生新的生命种群，该种群在算法的下一次迭代中成为当前种群[1]。

3.1.1 具体思想与实现方法

随机产生50个初始子代

选择拟合方式进行打分

轮盘选择算子，根据概率选出子代

依照交叉概率cr=0.7进行交叉

依照变异概率mr=0.02进行变异

对不合法子代进行修复，依照随机原则，变异成等位基因

得到子代，进行重复规定代数

得到结果。

3.2 模拟退火算法

模拟退火算法来源于固体退火原理，是一种基于概率的算法，将固体加温至充分高，再让其徐徐冷却，加温时，固体内部粒子随温升变为无序状，内能增大，而徐徐冷却时粒子渐趋有序，在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。

模拟退火算法通过赋予搜索过程一种时变且最终趋于零的概率突跳性，从而可有效避免陷入局部极小。但其参数设置比较困难，可能会导致收敛速度慢、执行时间长的问题。[2]

所以最终放弃使用该算法。

4 结论

我们用三次样条插值拟合，用遗传算法选点，所得结果的成本是86.64，得到的最优选点方案为[16,47,27,36 ,5]。

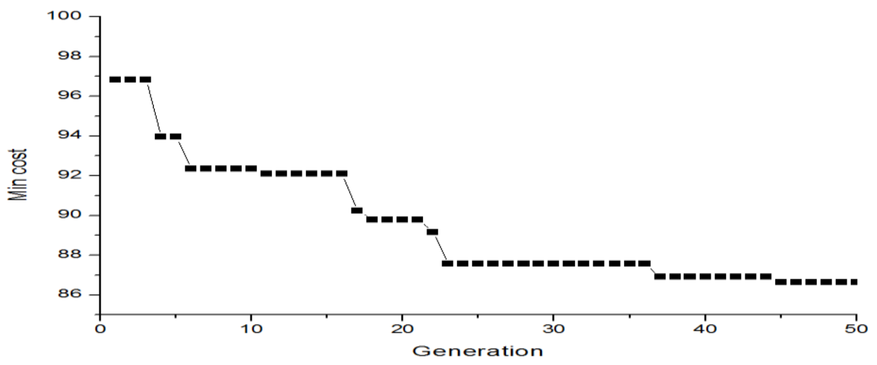

图 4.1 五特征点法最终结果呈现

需要注意的是，以上算出的最低成本不一定就是真正的最低成本。如果要得到最优成本，首先需要遍历每种选点方案，再确定拟合方式，由于总的方案数量庞大，无法在短时间内得出，并且，拟合方式有无穷多种，不能确定到底哪一种是最优的。

5 参考文献

[1]遗传算法 百度百科 网址：http://baike.baidu.com/link?url=6dF3xDAYqTTMq_AT-rkYaqgSD5Bs_9VlB7ICblga1Y2Vw_pYozlc7TNzz3H_Als4Wl8dr95hENFKodkOnVIUr_
[2]模拟退火算法 百度百科 网址：http://baike.baidu.com/view/18185.htm

6 附录（代码）

```
A = csvread('20150915dataform.csv');
% 设定参数
Nind = 50;%个体数目
MAXGEN = 50; %最大遗传代数
GGAP = 0.7; %代沟
min_cost = 1000;
min_sche = zeros;
min_points = 4;
%min_cost_table 用于记录各采样点下最小成本
min_cost_table = zeros(6,1);
% 外重循环枚举采样点个数
for num_of_ans = 4:9
```

```
BaseV = crtbase(num_of_ans,50);
[Chrom,Lind,BaseV] = crtbp(Nind,BaseV);%初始种群
l = length(Chrom(:,1));
for i = 1:l
        Chrom(i,:) = sort(Chrom(i,:));
end
ObjV = fitness(A,Chrom);
tmpbest = Chrom(1,:);
tmpmincost= ObjV(1,1);
gen = 0;

ObjV = ObjV';
while gen < MAXGEN
        FitnV = ranking(ObjV);
        SelCh = select('sus',Chrom,FitnV,GGAP);
        SelCh = recombin('xovsp',SelCh,0.7);
        SelCh = mut (SelCh,0.3,BaseV);
        ObjVSel = fitness(A,SelCh);
        ObjVSel = ObjVSel';
        [Chrom,ObjV] = reins(Chrom,SelCh,1,1,ObjV,ObjVSel);
        [Y,I] = min(ObjV);
        % 用上一代最优解替换当前这一代非最优解的个体
        if I ~= 1
                Chrom(1,:) = tmpbest;
                ObjV(1,:) = tmpmincost;
        end
        if I == 1
        Chrom(2,:) = tmpbest;
        ObjV(2,:) = tmpmincost;
        end
        gen = gen + 1;
        fprintf('Num of Points: %d\n',num_of_ans);
        fprintf('Generation: %d\n',gen);
        fprintf('Min cost : %5.2f\n',min(ObjV));
        % 记录这一代的最优解， 保留之后用于下一代的替换
        [Y,I] = min(ObjV);
        tmpbest = Chrom(I,:);
        tmpmincost = Y;
end
[Y,I] = min(ObjV);
sche = Chrom(I,:) + 1;
if Y < min_cost
        min_sche = sche;
        min_cost = Y;
```

```
            min_points = num_of_ans;
        end
        min_cost_table(num_of_ans - 3,1) = Y;
        fprintf('The final scheme is(%d points):\n',num_of_ans);
        disp(sche);
end
fprintf('The final scheme is:\n');
disp(min_sche);
fprintf('The final cost is :%5.2f\n',min_cost);

function BaseVec = crtbase(Lind, Base)

[ml LenL] = size(Lind) ;
if nargin < 2
        Base = 2 * ones(LenL,1) ; % default to base 2
end
[mb LenB] = size(Base) ;

% check parameter consistency
if ml > 1 | mb > 1
        error( 'Lind or Base is not a vector') ;
elseif (LenL > 1 & LenB > 1 & LenL ~= LenB) | (LenL == 1 & LenB > 1 )
        error( 'Vector dimensions must agree' ) ;
elseif LenB == 1 & LenL > 1
        Base = Base * ones(LenL,1) ;

end

BaseVec = [] ;
for i = 1:LenL
        BaseVec = [BaseVec, Base(i)*ones(Lind(i),1)'];
end
function [Chrom, Lind, BaseV] = crtbp(Nind, Lind, Base)
nargs = nargin ;

% Check parameter consistency

if nargs >= 1, [mN, nN] = size(Nind) ; end
if nargs >= 2, [mL, nL] = size(Lind) ; end
if nargs == 3, [mB, nB] = size(Base) ; end

if nN == 2
      if (nargs == 1)
            Lind = Nind(2) ; Nind = Nind(1) ; BaseV = crtbase(Lind) ;
```

```
    elseif (nargs == 2 & nL == 1)
        BaseV = crtbase(Nind(2),Lind) ; Lind = Nind(2) ; Nind = Nind(1) ;
    elseif (nargs == 2 & nL > 1)
        if Lind ~= length(Lind), error('Lind and Base disagree'); end
        BaseV = Lind ; Lind = Nind(2) ; Nind = Nind(1) ;
    end
elseif nN == 1
    if nargs == 2
        if nL == 1, BaseV = crtbase(Lind) ;
        else, BaseV = Lind ; Lind = nL ; end
    elseif nargs == 3
        if nB == 1, BaseV = crtbase(Lind,Base) ;
        elseif nB ~= Lind, error('Lind and Base disagree') ;
        else BaseV = Base ; end
    end
else
    error('Input parameters inconsistent') ;
end

% Create a structure of random chromosomes in row wise order, dimensions
% Nind by Lind. The base of each chromosomes loci is given by the value
% of the corresponding element of the row vector base.

Chrom = floor(rand(Nind,Lind).*BaseV(ones(Nind,1),:)) ;
%绘图函数
function draw()
A=csvread('20150915dataform.csv');%读入数据
%先定选样本1,32,55,122,123
t=[1 32 55 122 123];
for i=t(1,:)
     x=A(2*i,:);
     y=A(2*i+1,:);
     figure,polt(x,y,'rp-.');
end
A=csvread('20150915dataform.csv');%读入数据
n=5;
Chrom=[1 2 3 4 47];
syd=fitness(A,Chrom,n);
function syd=fitness(A,Chrom)
%取点方式为Chrom, n表示对该样本个体定标过程中的单点测定次数。
     %循环所有的样本。

     L=length(Chrom(:,1));
     n=length(Chrom(1,:));
```

```
        syd=zeros(1,L);

    for k=1:L

        for i=1:400    %800个样本
                x = A(i * 2 - 1,:); %    x=5:0.1:10;
                y = A(i * 2 ,:);
                xdata = zeros(1,n);
                ydata = zeros(1,n);
                sort(Chrom(k,:));
                b = unique(Chrom(k,:));
                if length(b) < n
                        syd(1,k) = 469000;
                        break;
                end
                for j = 1:n
                        xdata(1,j) = x(1,Chrom(k,j)+1 );
                        ydata(1,j) = y(1,Chrom(k,j)+1 );
                end
                %获得得出xdata,ydata  即取点的x和y的一行矩阵。
                %采用三次多项式插值
                y1 = interp1(xdata,ydata,x,'pchip');
                %调用fun1函数  计算成本。
                syd(1,k)=syd(1,k)+fun1(y1,y,n);
        end

        syd(1,k)=syd(1,k)/400;
    end
end
function f=fun1(y1,y,n)%y为实测值，y1为poly3的拟合值。成本计算c.
s=0;
for i=1:51%每行有51个数据点
        cha=abs(y1(1,i)-y(1,i));
        if ((0.4<cha)&&(cha<=0.6))
                s=s+0.1;
        end
        if ((0.6<cha)&&(cha<=0.8))
                s=s+0.7;
        end
        if ((0.8<cha)&&(cha<=1))
                s=s+0.9;
        end
        if ((1<cha)&&(cha<=2))
                s=s+1.5;
        end
        if ((2<cha)&&(cha<=3))
                s=s+6;
        end
        if ((3<cha)&&(cha<=5))
                s=s+12;
        end
        if (5<cha)
                s=s+25;
        end
end
s=s+12*n;% n表示对该样本个体定标过程中的单点测定次数。
f=s;
end
% MUT.m
%
% This function takes the representation of the current population,
% mutates each element with given probability and returns the resulting
% population.
%
% Syntax:       NewChrom = mut(OldChrom,Pm,BaseV)
%
% Input parameters:
%
%               OldChrom - A matrix containing the chromosomes of the
%                          current population. Each row corresponds to
%                          an individuals string representation.
%
%               Pm      - Mutation probability (scalar). Default value
%                          of Pm = 0.7/Lind, where Lind is the chromosome
%                          length is assumed if omitted.
%
%               BaseV       - Optional row vector of the same length as the
%                          chromosome structure defining the base of the
%                          individual elements of the chromosome. Binary
%                          representation is assumed if omitted.
%
% Output parameter:
%
%               NewChrom - A Matrix containing a mutated version of
%                          OldChrom.
%

% Author: Andrew Chipperfield
% Date: 25-Jan-94

function NewChrom = mut(OldChrom,Pm,BaseV)

% get population size (Nind) and chromosome length (Lind)
[Nind, Lind] = size(OldChrom) ;

% check input parameters
if nargin < 2, Pm = 0.7/Lind ; end
if isnan(Pm), Pm = 0.7/Lind; end

if (nargin < 3), BaseV = crtbase(Lind);    end
if (isnan(BaseV)), BaseV = crtbase(Lind);    end
if (isempty(BaseV)), BaseV = crtbase(Lind);    end

if (nargin == 3) & (Lind ~= length(BaseV))
    error('OldChrom and BaseV are incompatible'), end

% create mutation mask matrix
BaseM = BaseV(ones(Nind,1),:) ;

% perform mutation on chromosome structure
NewChrom =
rem(OldChrom+(rand(Nind,Lind)<Pm).*ceil(rand(Nind,Lind).*(BaseM-1)),BaseM);
function y1 = mycurvefitting( x_premea,y0_premea )

x=[5.0:0.1:10.0];

%  将你的定标计算方法写成指令代码，以下样式仅供参考
y1=interp1(x_premea,y0_premea,x,'cubic');



end
function FitnV = ranking(ObjV, RFun, SUBPOP);

% Identify the vector size (Nind)
    [Nind,ans] = size(ObjV);

    if nargin < 2, RFun = []; end
    if nargin > 1, if isnan(RFun), RFun = []; end, end
    if prod(size(RFun)) == 2,
        if RFun(2) == 1, NonLin = 1;
        elseif RFun(2) == 0, NonLin = 0;
        else error('Parameter for ranking method must be 0 or 1'); end
        RFun = RFun(1);
        if isnan(RFun), RFun = 2; end
    elseif prod(size(RFun)) > 2,
        if prod(size(RFun)) ~= Nind, error('ObjV and RFun disagree'); end
    end

    if nargin < 3, SUBPOP = 1; end
    if nargin > 2,
        if isempty(SUBPOP), SUBPOP = 1;
        elseif isnan(SUBPOP), SUBPOP = 1;
        elseif length(SUBPOP) ~= 1, error('SUBPOP must be a scalar'); end
    end

    if (Nind/SUBPOP) ~= fix(Nind/SUBPOP), error('ObjV and SUBPOP disagree'); end
    Nind = Nind/SUBPOP;    % Compute number of individuals per subpopulation

% Check ranking function and use default values if necessary
    if isempty(RFun),
        % linear ranking with selective pressure 2
            RFun = 2*[0:Nind-1]'/(Nind-1);
    elseif prod(size(RFun)) == 1
        if NonLin == 1,
            % non-linear ranking
            if RFun(1) < 1, error('Selective pressure must be greater than 1');
            elseif RFun(1) > Nind-2, error('Selective pressure too big'); end
            Root1 = roots([RFun(1)-Nind [RFun(1)*ones(1,Nind-1)]]);
            RFun = (abs(Root1(1)) * ones(Nind,1)) .^ [(0:Nind-1)'];
            RFun = RFun / sum(RFun) * Nind;
        else
            % linear ranking with SP between 1 and 2
            if (RFun(1) < 1 | RFun(1) > 2),
                error('Selective pressure for linear ranking must be between 1 and 2');
            end
            RFun = 2-RFun + 2*(RFun-1)*[0:Nind-1]'/(Nind-1);
        end
    end;

    FitnV = [];

% loop over all subpopulations
for irun = 1:SUBPOP,
    % Copy objective values of actual subpopulation
        ObjVSub = ObjV((irun-1)*Nind+1:irun*Nind);
    % Sort does not handle NaN values as required. So, find those...
```

```
            NaNix = isnan(ObjVSub);
            Validix = find(~NaNix);
        % ... and sort only numeric values (smaller is better).
            [ans,ix] = sort(-ObjVSub(Validix));

        % Now build indexing vector assuming NaN are worse than numbers,
        % (including Inf!)...
            ix = [find(NaNix) ; Validix(ix)];
        % ... and obtain a sorted version of ObjV
            Sorted = ObjVSub(ix);

        % Assign fitness according to RFun.
            i = 1;
            FitnVSub = zeros(Nind,1);
            for j = [find(Sorted(1:Nind-1) ~= Sorted(2:Nind)); Nind]',
                FitnVSub(i:j) = sum(RFun(i:j)) * ones(j-i+1,1) / (j-i+1);
                i =j+1;
            end

        % Finally, return unsorted vector.
            [ans,uix] = sort(ix);
            FitnVSub = FitnVSub(uix);

        % Add FitnVSub to FitnV
            FitnV = [FitnV; FitnVSub];
end
% RECOMBIN.M             (RECOMBINation high-level function)
%
% This function performs recombination between pairs of individuals
% and returns the new individuals after mating. The function handles
% multiple populations and calls the low-level recombination function
% for the actual recombination process.
%
% Syntax:    NewChrom = recombin(REC_F, OldChrom, RecOpt, SUBPOP)
%
% Input parameters:
%       REC_F         - String containing the name of the recombination or
%                       crossover function
%       Chrom         - Matrix containing the chromosomes of the old
%                       population. Each line corresponds to one individual
%       RecOpt        - (optional) Scalar containing the probability of
%                       recombination/crossover occurring between pairs
%                       of individuals.
%                       if omitted or NaN, 1 is assumed
%        SUBPOP        - (optional) Number of subpopulations
%                       if omitted or NaN, 1 subpopulation is assumed
%
% Output parameter:
%        NewChrom    - Matrix containing the chromosomes of the population
%                       after recombination in the same format as OldChrom.

%    Author:       Hartmut Pohlheim
%    History:     18.03.94        file created


function NewChrom = recombin(REC_F, Chrom, RecOpt, SUBPOP)


% Check parameter consistency
   if nargin < 2, error('Not enough input parameter'); end

   % Identify the population size (Nind)
   [Nind,Nvar] = size(Chrom);

   if nargin < 4, SUBPOP = 1; end
   if nargin > 3,
      if isempty(SUBPOP), SUBPOP = 1;
      elseif isnan(SUBPOP), SUBPOP = 1;
      elseif length(SUBPOP) ~= 1, error('SUBPOP must be a scalar'); end
   end

   if (Nind/SUBPOP) ~= fix(Nind/SUBPOP), error('Chrom and SUBPOP disagree'); end
   Nind = Nind/SUBPOP;    % Compute number of individuals per subpopulation

   if nargin < 3, RecOpt = 0.7; end
   if nargin > 2,
      if isempty(RecOpt), RecOpt = 0.7;
      elseif isnan(RecOpt), RecOpt = 0.7;
      elseif length(RecOpt) ~= 1, error('RecOpt must be a scalar');
      elseif (RecOpt < 0 || RecOpt > 1), error('RecOpt must be a scalar in [0, 1]'); end
   end


% Select individuals of one subpopulation and call low level function
   NewChrom = [];
   for irun = 1:SUBPOP,
      ChromSub = Chrom((irun-1)*Nind+1:irun*Nind,:);
      NewChromSub = feval(REC_F, ChromSub, RecOpt);
```

```
          NewChrom=[NewChrom; NewChromSub];
      end


% End of function
% REINS.M          (RE-INSertion of offspring in population replacing parents)
%
% This function reinserts offspring in the population.
%
% Syntax: [Chrom, ObjVCh] = reins(Chrom, SelCh, SUBPOP, InsOpt, ObjVCh, ObjVSel)
%
% Input parameters:
%      Chrom        - Matrix containing the individuals (parents) of the current
%                     population. Each row corresponds to one individual.
%      SelCh        - Matrix containing the offspring of the current
%                     population. Each row corresponds to one individual.
%      SUBPOP       - (optional) Number of subpopulations
%                     if omitted or NaN, 1 subpopulation is assumed
%      InsOpt       - (optional) Vector containing the insertion method parameters
%                     ExOpt(1): Select - number indicating kind of insertion
%                                0 - uniform insertion
%                                1 - fitness-based insertion
%                                if omitted or NaN, 0 is assumed
%                     ExOpt(2): INSR - Rate of offspring to be inserted per
%                                subpopulation (% of subpopulation)
%                                if omitted or NaN, 1.0 (100%) is assumed
%      ObjVCh       - (optional) Column vector containing the objective values
%                     of the individuals (parents - Chrom) in the current
%                     population, needed for fitness-based insertion
%                     saves recalculation of objective values for population
%      ObjVSel      - (optional) Column vector containing the objective values
%                     of the offspring (SelCh) in the current population, needed for
%                     partial insertion of offspring,
%                     saves recalculation of objective values for population
%
% Output parameters:
%      Chrom        - Matrix containing the individuals of the current
%                     population after reinsertion.
%      ObjVCh       - if ObjVCh and ObjVSel are input parameter, than column
%                     vector containing the objective values of the individuals
%                     of the current generation after reinsertion.

% Author:     Hartmut Pohlheim
% History:    10.03.94     file created
%                    19.03.94        parameter checking improved

function [Chrom, ObjVCh] = reins(Chrom, SelCh, SUBPOP, InsOpt, ObjVCh, ObjVSel);


% Check parameter consistency
      if nargin < 2, error('Not enough input parameter'); end
      if (nargout == 2 && nargin < 6), error('Input parameter missing: ObjVCh and/or ObjVSel');
end

      [NindP, NvarP] = size(Chrom);
      [NindO, NvarO] = size(SelCh);

      if nargin == 2, SUBPOP = 1; end
      if nargin > 2,
          if isempty(SUBPOP), SUBPOP = 1;
          elseif isnan(SUBPOP), SUBPOP = 1;
          elseif length(SUBPOP) ~= 1, error('SUBPOP must be a scalar'); end
      end

      if (NindP/SUBPOP) ~= fix(NindP/SUBPOP), error('Chrom and SUBPOP disagree'); end
      if (NindO/SUBPOP) ~= fix(NindO/SUBPOP), error('SelCh and SUBPOP disagree'); end
      NIND = NindP/SUBPOP;    % Compute number of individuals per subpopulation
      NSEL = NindO/SUBPOP;    % Compute number of offspring per subpopulation

      IsObjVCh = 0; IsObjVSel = 0;
      if nargin > 4,
          [mO, nO] = size(ObjVCh);
          if nO ~= 1, error('ObjVCh must be a column vector'); end
          if NindP ~= mO, error('Chrom and ObjVCh disagree'); end
          IsObjVCh = 1;
      end
      if nargin > 5,
          [mO, nO] = size(ObjVSel);
          if nO ~= 1, error('ObjVSel must be a column vector'); end
          if NindO ~= mO, error('SelCh and ObjVSel disagree'); end
          IsObjVSel = 1;
      end

      if nargin < 4, INSR = 1.0; Select = 0; end
      if nargin >= 4,
          if isempty(InsOpt), INSR = 1.0; Select = 0;
          elseif isnan(InsOpt), INSR = 1.0; Select = 0;
          else
```

```
            INSR = NaN; Select = NaN;
            if (length(InsOpt) > 2), error('Parameter InsOpt too long'); end
            if (length(InsOpt) >= 1), Select = InsOpt(1); end
            if (length(InsOpt) >= 2), INSR = InsOpt(2); end
            if isnan(Select), Select = 0; end
            if isnan(INSR), INSR =1.0; end
        end
    end

    if (INSR < 0 || INSR > 1), error('Parameter for insertion rate must be a scalar in [0, 1]'); end
    if (INSR < 1 && IsObjVSel ~= 1), error('For selection of offspring ObjVSel is needed'); end
    if (Select ~= 0 && Select ~= 1), error('Parameter for selection method must be 0 or 1'); end
    if (Select == 1 && IsObjVCh == 0), error('ObjVCh for fitness-based exchange needed'); end

    if INSR == 0, return; end
    NIns = min(max(floor(INSR*NSEL+.5),1),NIND);     % Number of offspring to insert

% perform insertion for each subpopulation
    for irun = 1:SUBPOP,
        % Calculate positions in old subpopulation, where offspring are inserted
            if Select == 1,       % fitness-based reinsertion
                [Dummy, ChIx] = sort(-ObjVCh((irun-1)*NIND+1:irun*NIND));
            else                          % uniform reinsertion
                [Dummy, ChIx] = sort(rand(NIND,1));
            end
            PopIx = ChIx((1:NIns)')+ (irun-1)*NIND;
        % Calculate position of Nins-% best offspring
            if (NIns < NSEL),    % select best offspring
                [Dummy,OffIx] = sort(ObjVSel((irun-1)*NSEL+1:irun*NSEL));
            else
                OffIx = (1:NIns)';
            end
            SelIx = OffIx((1:NIns)')+(irun-1)*NSEL;
        % Insert offspring in subpopulation -> new subpopulation
            Chrom(PopIx,:) = SelCh(SelIx,:);
            if (IsObjVCh == 1 && IsObjVSel == 1), ObjVCh(PopIx) = ObjVSel(SelIx); end
    end


% End of function
function SelCh = select(SEL_F, Chrom, FitnV, GGAP, SUBPOP);

% Check parameter consistency
    if nargin < 3, error('Not enough input parameter'); end
    % Identify the population size (Nind)
    [NindCh,Nvar] = size(Chrom);
    [NindF,VarF] = size(FitnV);
    if NindCh ~= NindF, error('Chrom and FitnV disagree'); end
    if VarF ~= 1, error('FitnV must be a column vector'); end

    if nargin < 5, SUBPOP = 1; end
    if nargin > 4,
       if isempty(SUBPOP), SUBPOP = 1;
       elseif isnan(SUBPOP), SUBPOP = 1;
       elseif length(SUBPOP) ~= 1, error('SUBPOP must be a scalar'); end
    end

    if (NindCh/SUBPOP) ~= fix(NindCh/SUBPOP), error('Chrom and SUBPOP disagree'); end
    Nind = NindCh/SUBPOP;    % Compute number of individuals per subpopulation

    if nargin < 4, GGAP = 1; end
    if nargin > 3,
       if isempty(GGAP), GGAP = 1;
       elseif isnan(GGAP), GGAP = 1;
       elseif length(GGAP) ~= 1, error('GGAP must be a scalar');
       elseif (GGAP < 0), error('GGAP must be a scalar bigger than 0'); end
    end

% Compute number of new individuals (to select)
    NSel=max(floor(Nind*GGAP+.5),2);

% Select individuals from population
    SelCh = [];
    for irun = 1:SUBPOP,
       FitnVSub = FitnV((irun-1)*Nind+1:irun*Nind);
       ChrIx=feval(SEL_F, FitnVSub, NSel)+(irun-1)*Nind;
       SelCh=[SelCh; Chrom(ChrIx,:)];
    end
A = csvread('20150915dataform.csv');%读入数据
chorm=[1 2 3 4 5 6 7;2 4 6 34 35 36 41;2 3 4 5 6 7 8];
%f=FitnessFunc(A,chorm);
f=fitness(A,chorm);
%L= length(chrom(:,1));
%s=size(chorm);
%f=f';
A = csvread('20150915dataform.csv');
```

```
% 设定参数
Nind = 50;%个体数目
MAXGEN = 10; %最大遗传代数
GGAP = 0.7; %代沟
min_cost = 1000;
min_sche = zeros;
min_points = 4;

%min_cost_table 用于记录各采样点下最小成本
min_cost_table = zeros(6,1);

% 外重循环枚举采样点个数
for num_of_ans = 4:9
    BaseV = crtbase(num_of_ans,50);
    [Chrom,Lind,BaseV] = crtbp(Nind,BaseV);%初始种群
    l = length(Chrom(:,1));
    for i = 1:l
        Chrom(i,:) = sort(Chrom(i,:));
    end
    ObjV = fitness(A,Chrom,num_of_ans);
    tmpbest = Chrom(1,:);
    tmpmincost= ObjV(1,1);
    gen = 0;
    ObjV = ObjV';
    while gen < MAXGEN
        FitnV = ranking(ObjV);
        SelCh = select('sus',Chrom,FitnV,GGAP);
        SelCh = recombin('xovsp',SelCh,0.7);
        SelCh = mut(SelCh,0.3,BaseV);
        ObjVSel = fitness(A,SelCh,num_of_ans);
        ObjVSel = ObjVSel';
        [Chrom,ObjV] = reins(Chrom,SelCh,1,1,ObjV,ObjVSel);
        [Y,I] = min(ObjV);
        % 用上一代最优解替换当前这一代非最优解的个体
        if I ~= 1
            Chrom(1,:) = tmpbest;
            ObjV(1,:) = tmpmincost;
        end
        if I == 1
        Chrom(2,:) = tmpbest;
        ObjV(2,:) = tmpmincost;
        end
        gen = gen + 1;
        fprintf('Num of Points: %d\n',num_of_ans);
```

```
            fprintf('Generation: %d\n',gen);
            fprintf('Min cost : %5.2f\n',min(ObjV));
            % 记录这一代的最优解， 保留之后用于下一代的替换
            [Y,I] = min(ObjV);
            tmpbest = Chrom(I,:);
            tmpmincost = Y;
        end
        [Y,I] = min(ObjV);
        sche = Chrom(I,:) + 1;
        if Y < min_cost
            min_sche = sche;
            min_cost = Y;
            min_points = num_of_ans;
        end
        min_cost_table(num_of_ans - 3,1) = Y;
        fprintf('The final scheme is(%d points):\n',num_of_ans);
        disp(sche);
end
fprintf('The final scheme is:\n');
disp(min_sche);
fprintf('The final cost is :%5.2f\n',min_cost);
function NewChrIx = sus(FitnV,Nsel);

% Identify the population size (Nind)
   [Nind,ans] = size(FitnV);

% Perform stochastic universal sampling
   cumfit = cumsum(FitnV);
   trials = cumfit(Nind) / Nsel * (rand + (0:Nsel-1)');
   Mf = cumfit(:, ones(1, Nsel));
   Mt = trials(:, ones(1, Nind))';
   [NewChrIx, ans] = find(Mt < Mf & [ zeros(1, Nsel); Mf(1:Nind-1, :) ] <= Mt);

% Shuffle new population
   [ans, shuf] = sort(rand(Nsel, 1));
   NewChrIx = NewChrIx(shuf);


% End of function
%%%%%% 答案检验程序 2015-11-04 %%%%%%%

my_answer=[ 5      18      26      33      42      50];%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);
```

```
% XOVMP.m                          Multi-point crossover
%
%           Syntax: NewChrom =    xovmp(OldChrom, Px, Npt, Rs)
%
%           This function takes a matrix OldChrom containing the binary
%           representation of the individuals in the current population,
%           applies crossover to consecutive pairs of individuals with
%           probability Px and returns the resulting population.
%
%           Npt indicates how many crossover points to use (1 or 2, zero
%           indicates shuffle crossover).
%           Rs indicates whether or not to force the production of
%           offspring different from their parents.
%

% Author: Carlos Fonseca,     Updated: Andrew Chipperfield
% Date: 28/09/93,          Date: 27-Jan-94

function NewChrom = xovmp(OldChrom, Px, Npt, Rs);

% Identify the population size (Nind) and the chromosome length (Lind)
[Nind,Lind] = size(OldChrom);

if Lind < 2, NewChrom = OldChrom; return; end

if nargin < 4, Rs = 0; end
if nargin < 3, Npt = 0; Rs = 0; end
if nargin < 2, Px = 0.7; Npt = 0; Rs = 0; end
if isnan(Px), Px = 0.7; end
if isnan(Npt), Npt = 0; end
if isnan(Rs), Rs = 0; end
if isempty(Px), Px = 0.7; end
if isempty(Npt), Npt = 0; end
if isempty(Rs), Rs = 0; end

Xops = floor(Nind/2);
DoCross = rand(Xops,1) < Px;
odd = 1:2:Nind-1;
even = 2:2:Nind;

% Compute the effective length of each chromosome pair
Mask = ~Rs | (OldChrom(odd, :) ~= OldChrom(even, :));
Mask = cumsum(Mask')';
```

```
% Compute cross sites for each pair of individuals, according to their
% effective length and Px (two equal cross sites mean no crossover)
xsites(:, 1) = Mask(:, Lind);
if Npt >= 2,
        xsites(:, 1) = ceil(xsites(:, 1) .* rand(Xops, 1));
end
xsites(:,2) = rem(xsites + ceil((Mask(:, Lind)-1) .* rand(Xops, 1)) ...
                                .* DoCross - 1 , Mask(:, Lind) )+1;

% Express cross sites in terms of a 0-1 mask
Mask = (xsites(:,ones(1,Lind)) < Mask) == ...
                        (xsites(:,2*ones(1,Lind)) < Mask);

if ~Npt,
        shuff = rand(Lind,Xops);
        [ans,shuff] = sort(shuff);
        for i=1:Xops
          OldChrom(odd(i),:)=OldChrom(odd(i),shuff(:,i));
          OldChrom(even(i),:)=OldChrom(even(i),shuff(:,i));
        end
end

% Perform crossover
NewChrom(odd,:) = (OldChrom(odd,:).* Mask) + (OldChrom(even,:).*(~Mask));
NewChrom(even,:) = (OldChrom(odd,:).*(~Mask)) + (OldChrom(even,:).*Mask);

% If the number of individuals is odd, the last individual cannot be mated
% but must be included in the new population
if rem(Nind,2),
  NewChrom(Nind,:)=OldChrom(Nind,:);
end

if ~Npt,
        [ans,unshuff] = sort(shuff);
        for i=1:Xops
          NewChrom(odd(i),:)=NewChrom(odd(i),unshuff(:,i));
          NewChrom(even(i),:)=NewChrom(even(i),unshuff(:,i));
        end
end
% XOVSP.M          (CROSSOVer Single-Point)
%
% This function performs single-point crossover between pairs of
% individuals and returns the current generation after mating.
%
```

```
% Syntax:    NewChrom = xovsp(OldChrom, XOVR)
%
% Input parameters:
%       OldChrom    - Matrix containing the chromosomes of the old
%                      population. Each line corresponds to one individual
%                      (in any form, not necessarily real values).
%       XOVR         - Probability of recombination occurring between pairs
%                      of individuals.
%
% Output parameter:
%       NewChrom    - Matrix containing the chromosomes of the population
%                      after mating, ready to be mutated and/or evaluated,
%                      in the same format as OldChrom.

%    Author:       Hartmut Pohlheim
%    History:     28.03.94         file created

function NewChrom = xovsp(OldChrom, XOVR);

if nargin < 2, XOVR = NaN; end

% call low level function with appropriate parameters
    NewChrom = xovmp(OldChrom, XOVR, 1, 0);


% End of function
```