基于遗传算法的函数最值求解算法

孙国栋

重庆大学无线通信技术实验室

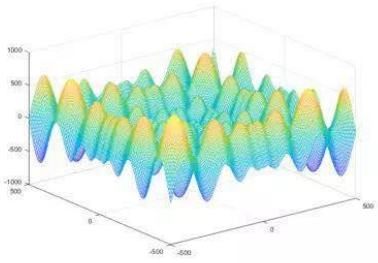
一、问题描述

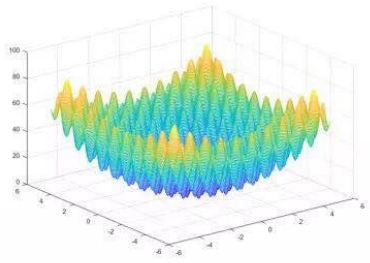
选择如下图中的 6-9 号测试函数在给定的自变量范围内通过遗传算法找出其中的最小值点及最小值结果。

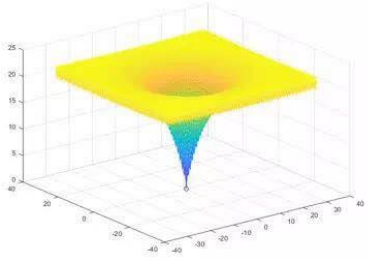
Test functions	SD
$f_1(x)=\sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$
$f_2(x)=\sum_{i=1}^{n} \lvert x_i\rvert+\prod_{i=1}^{n} \lvert x_i\rvert$	$[-10, 10]^n$
$f_3(x)=\sum_{i=1}^{n}(\sum_{j=1}^{n} x_j)^2$	$[-100, 100]^n$
$f_4(x)=\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2]$	$[-30, 30]^n$
$f_5(x)=\sum_{i=1}^{n}(x_i+0.5)^2$	$[-100, 100]^n$
$f_6(x)=\sum_{i=1}^{n}-x_i\sin(\sqrt{\lvert x_i\rvert})$	$[-500, 500]^n$
$f_7(x)=\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)+10]$	$[-5.12, 5.12]^n$
$f_8(x)=-20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i)+20+e$	$[-32, 32]^n$
$f_9(x)=\frac{1}{4000}\sum_{i=1}^{n} x_i^2-\prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})+1$	$[-600, 600]^n$

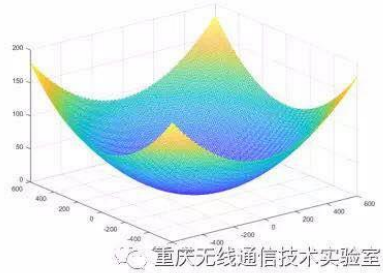
重庆无线通信技术实验室

二、函数图三维展示（函数 6-9）



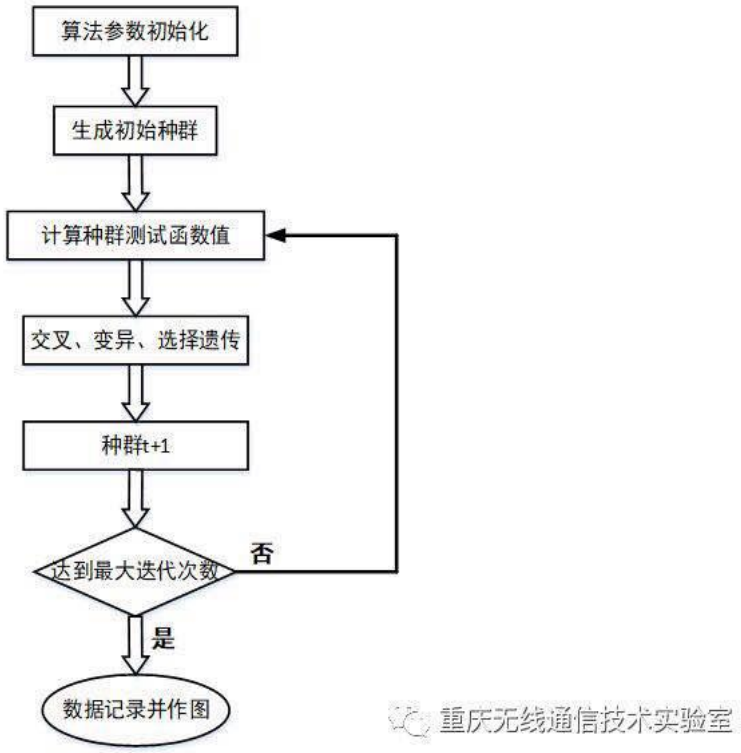






重庆无线通信技术实验室

三、遗传算法程序编写流程及说明



1、算法参数初始化

算法参数初始化是先定义好遗传算法所需要调节的各参数，包括有测试函数选择（6-9 号测试函数选择某一个），对于编码的遗传算法需要先定义好二进制编码的位数，另外对于遗传算法还需要定义种群内的个体数，染色体节点数（是指测试函数自变量的维度），最大迭代次数，突变概率，交叉概率等，需要说明的是遗传算法的交叉概率应在 0.3-0.9 区间内，变异概率应小于 0.1 比较合适。

2、生成初始种群

生成初始种群前要定义好染色体节点的值得变化范围（自变量范围），另外对于 MATLAB 仿真先要开辟好各种群个体染色体存放矩阵，个体适应度函数（测试函数值）存放矩阵，平均适应度和最优适应度存放矩阵以及最优个体染色体存放矩阵等，最后可以生成初始种群，初始种群个体的染色体要满足染色体自变量参数的变化范围，在定义域内随机产生。

3、计算种群测试函数值

根据当前种群个体，计算每个个体的测试函数值，并选择函数值最小的作为当前最优适应度的个体，保存最优适应度及对应的染色体，并计算当前的平均适应度函数值。

4、交叉、变异、遗传选择

对当前种群的染色体部分基因进行交叉、变异操作，交叉及变异概率在参数初始化时已经定义好，最后通过一定的选择方式得到下一代的种群，进而种群代数 t+1。

5、循环迭代

重复上面的 3、4 过程，直到达到预先设置的最大迭代次数，跳出迭代循环。

6、数据记录并做图

记录迭代过程产生的每一代种群的最优适应度和平均适应度函数值，并作图。

四、经典算法说明

1、实数编码与二进制编码

实数编码：根据函数自变量的定义域范围，随机产生种群个体，个体的自变量不经过编码，直接产生随机实数，适合于遗传算法中表示范围较大的数，使遗传算法更接近问题空间，避免了编码和解码的过程，便于较大空间的遗传搜索，提高了 GA 的精度要求。

二进制编码：随机生成一定长度的二进制数，并对应到自变量实数变化范围内进行编码，是遗传算法中最常用的一种编码方法，它采用最小字符编码原则，编码、解码操作简单易行，利于交叉、变异操作实现。

2、最优个体保存选择和轮盘赌法选择

最优个体保存选择：在遗传算法选择过程中，将适应度最劣的部分个体直接

用最优个体取代，其他个体直接遗传，该方法可保证在遗传过程中所得到的个体不会被交叉和变异操作所破坏，能有效保证遗传算法的收敛性。

轮盘赌法选择：基于测试函数值，通过一定方法得到适应度函数，适应度值越大，遗传时被选择的概率越大，根据轮盘概率选择原理进行选择遗传。由于此方法是基于概率选择的，存在统计误差，因此可以结合最优保存策略，以保证当前适应度最好的个体能够进化到下一代而不被遗传操作的随机性破坏掉，保证算法的收敛性。

3、单点交叉、均匀交叉及染色体交叉

单点交叉：在个体编码串中随机设置一个交叉点，然后在该点相互交换两个配对个体的部分基因。

均匀交叉：两个相互配对个体的每一位基因都以相同的概率进行交换，从而形成两个新个体。

染色体交叉：直接随机选择两个对应染色体进行交叉。（由于该问题是多维自变量选择问题，并且显然在测试函数最小值处，各个自变量的取值是相同的，因此才能保证该操作有效性。）

五、针对几种经典算法的仿真结果对比

根据编码方式和遗传选择方式不同，这里给出了三种组合方式：

实数编码+最优个体保存选择；

二进制编码+最优个体保存选择；

二进制编码+轮盘赌法选择。

仿真参数：

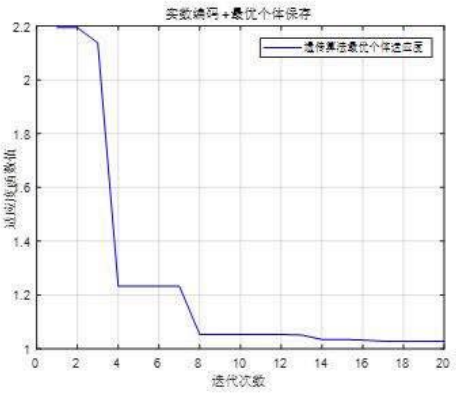
参数名称	设值
测试函数选择	9（对应 6-9 号测试函数）
交叉方式	1（1 染色体交叉，2 单点交叉，3 均匀交叉）
二进制编码位数	16
种群个体数目	100
染色体节点数	2（以 2 为例，便于作图）
迭代次数	20
突变概率	0.01
交叉概率	5

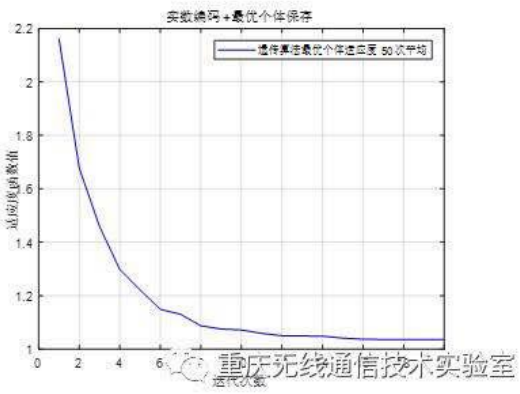
重庆无线通信技术实验室

仿真结果：

（由于遗传算法每次循环迭代的随机性大，因此这里通过完成 50 次平均展示遗传算法的性能）

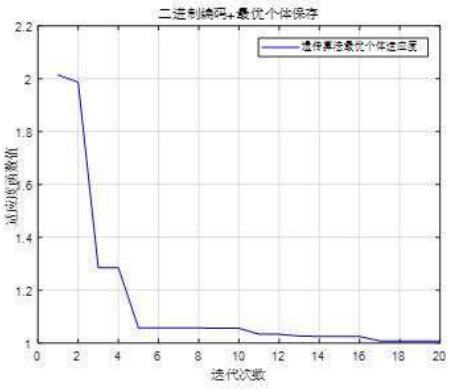
实数编码+最优个体保存选择

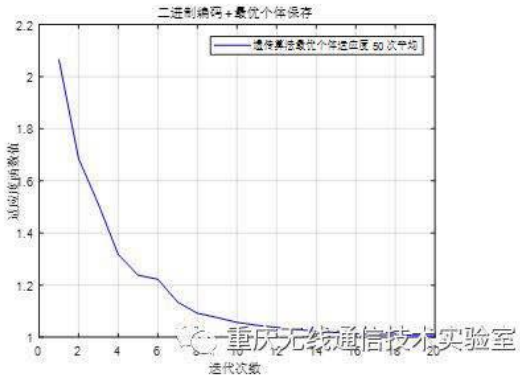




重庆无线通信技术实验室

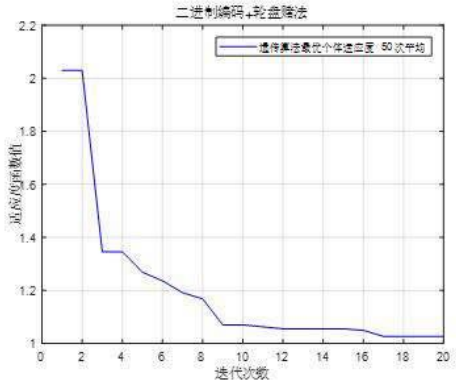
二进制编码+最优个体保存选择

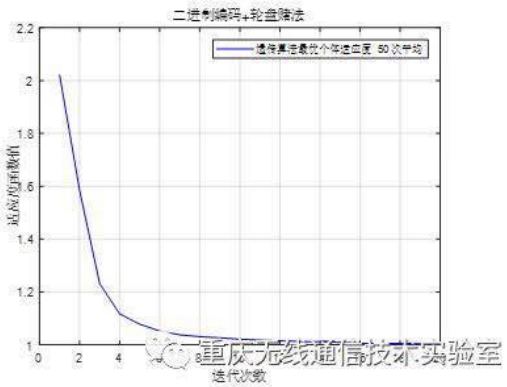




重庆无线通信技术实验室

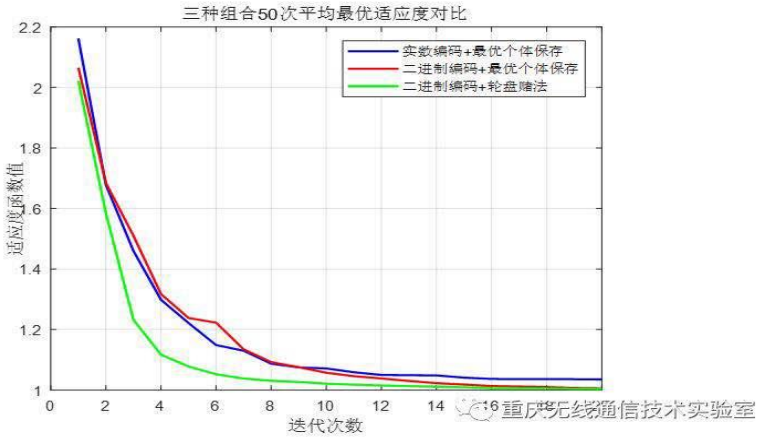
二进制编码+轮盘赌法选择





重庆无线通信技术实验室

为了方便三种组合遗传算法的对比，将三种组合算法的最有个体适应度 50 次平均的结果在一张结果图上展示如下：



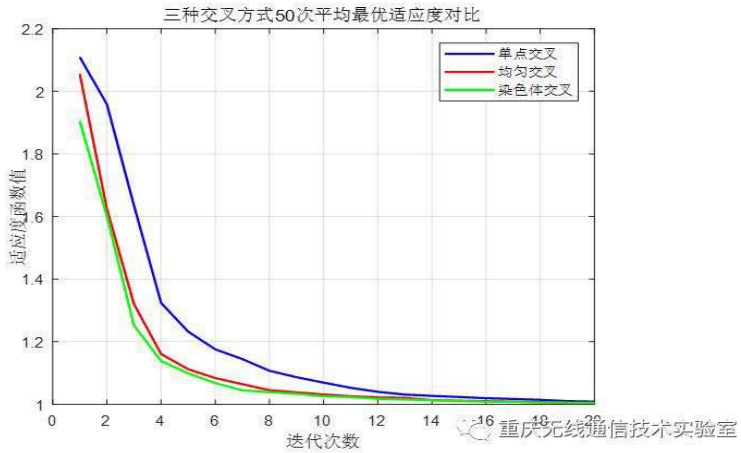
结果分析：

这是三种组合算法在做完 50 次平均后的最优适应度随迭代次数的对比曲线图。由图中结果可知：实数编码对应蓝色曲线在 20 次迭代结束时的收敛函数值性能要比二进制编码对应的红色、绿色曲线在 20 次迭代结束时差，所以对于该测试函数问题二进制编码性能要优于实数编码的算法性能。

主要原因个人猜测是由于实数编码的随机性过大，而二进制编码有固定二进制位数，在一定精度要求下，算法性能是优于实数编码的。

最优个体保存的遗传选择方法对应的红色、蓝色曲线在 10 次迭代之前的收敛速度要比轮盘赌法对应的绿色曲线差，所以认为该问题采用轮盘赌法要比最有个体保存方法更优。

产生这样的结果主要原因是由于最优个体保存是直接将最劣的一部分个体直接用最有个体进行取代，降低了种群的染色体多样性；而轮盘赌法通过概率遗传，则遗传选择后的子代保存了所有上一代较优个体的染色体，因此经过染色体交叉后收敛速度更快。



结果分析：

这是三种交叉方式在完成 50 次平均后的最优适应度随迭代次数的对比曲线图

由图可知，染色体交叉和均匀交叉的性能相近，染色体交叉略优于均匀交叉，而单点交叉对比染色体交叉和均匀交叉方式的收敛性能要差一些。

原因分析，单点交叉对于二进制编码来说，只交叉一位二进制码对于遗传进化来说效果不大，而染色体交叉和均匀交叉则是考虑到整个染色体都可能发生交叉，交叉的效果更佳显著，性能也更优。

附 MATLAB 仿真代码

问题 1：（本文对比了多种经典算法，但大体相同，这里不重复贴程序了）
主函数：

```
clc, clear, close all
%% 基本参数定义
mode = 9;   %6-9 号函数选择
code = 16;   %二进制编码位数
N = 100;    %种群内个体数目
N_chrom = 2; %染色体节点数
iter = 20;   %迭代次数
mut = 0.01; %突变概率
acr = 0.5; %交叉概率
M = 50;   %平均次数
fitness_best_M = zeros(1, iter);
for m = 1:M
%% 开辟数据存放区间
chrom_range = range(N_chrom, mode);%每个节点的值的区间
chrom = zeros(N, N_chrom);%存放染色体的矩阵
fitness = zeros(N, 1);%存放染色体的适应度
fitness_ave = zeros(1, iter);%存放每一代的平均适应度
fitness_best = zeros(1, iter);%存放每一代的最优适应度
chrom_best = zeros(1, N_chrom+1);%存放当前代的最优染色体（2 维）与适应度（1 维）

%% 初始化
chrom = Initialize(N, N_chrom, code); %初始化染色体
fitness = CalFitness(chrom, N, N_chrom,chrom_range, mode); %计算适应度
[chrom_best,best_value]   = FindBest(chrom, fitness, N_chrom, code); %寻找最优染色体
fitness_best(1) = best_value; %将当前最优存入矩阵当中
fitness_ave(1) = CalAveFitness(fitness); %将当前平均适应度存入矩阵当中
fitness_best_M(1) = best_value;
```

```
for t = 2:iter
      chrom =MutChrom(chrom, mut, N, N_chrom); %变异
      chrom =AcrChrom(chrom, acr, N, N_chrom); %交叉
      fitness =CalFitness(chrom, N, N_chrom, chrom_range, mode); %计算适应度
    %chrom_best_temp = FindBest(chrom, fitness, N_chrom); %寻找最优染色体
    [chrom_best_temp,best_value_temp] = FindBest(chrom, fitness, N_chrom,code); %寻找最优染色体
      ifbest_value_temp < best_value %替换掉当前储存的最优
          best_value = best_value_temp;
          chrom_best = chrom_best_temp;
      end
      [chrom,fitness] = ReplaceWorse(chrom, chrom_best, best_value, fitness, code);%替换掉最劣
    fitness_best(t) = best_value; %将当前最优存入矩阵当中
    fitness_ave(t) = CalAveFitness(fitness); %将当前平均适应度存入矩阵当中
    fitness_best_M(m,:) = fitness_best(1,:);
end
end
for i = 1:iter
    fitness_best_average(i) = sum(fitness_best_M(:,i))/M;
end

%% 作图
figure(1)
plot(1:iter, fitness_ave, 'r')
grid on
xlabel('迭代次数');
ylabel('适应度函数值');
title('实数编码+最优个体保存');
legend('遗传算法平均适应度');
figure_FontSize=10;
set(get(gca,'XLabel'),'FontSize',figure_FontSize,'Vertical','top');
set(get(gca,'YLabel'),'FontSize',figure_FontSize,'Vertical','middle');
set(findobj('FontSize',10),'FontSize',figure_FontSize);
set(findobj(get(gca,'Children'),'LineWidth',0.5),'LineWidth',1.5);

figure(3)
plot(1:iter, fitness_best_average, 'b')
grid on
xlabel('迭代次数');
ylabel('适应度函数值');
title('实数编码+最优个体保存');
legend('遗传算法最优个体适应度 50 次平均');
figure_FontSize=10;
```

```
set(get(gca,'XLabel'),'FontSize',figure_FontSize,'Vertical','top');
set(get(gca,'YLabel'),'FontSize',figure_FontSize,'Vertical','middle');
set(findobj('FontSize',10),'FontSize',figure_FontSize);
set(findobj(get(gca,'Children'),'LineWidth',0.5),'LineWidth',1.5);
PlotModel(chrom_best,mode);

%% 输出结果
disp(['最优染色体为  ',num2str(chrom_best(1:end))])
disp(['最优适应度为  ',num2str(best_value)])
```

子函数 1：

```
function chrom_range = range(N_chrom, Mode)
switch Mode
    case 6
        chrom_range = [-500*ones(1,N_chrom); 500*ones(1,N_chrom)];
    case 7
        chrom_range = [-5.12*ones(1,N_chrom); 5.12*ones(1,N_chrom)];
    case 8
        chrom_range = [-32*ones(1,N_chrom); 32*ones(1,N_chrom)];
    case 9
        chrom_range = [-600*ones(1,N_chrom); 600*ones(1,N_chrom)];
    otherwise
        error('Invalid Modulation Mode. Use {6,7,8,or 9}');
end
```

子函数 2：

```
function chrom_new = Initialize(N, N_chrom, code)
chrom_new = randi([0,1],N,code,N_chrom);
```

子函数 3：

```
function fitness = CalFitness(chrom, N, N_chrom,chrom_range, Mode)
fitness = zeros(N, 1);
y1 = zeros(N, 1);
y2 = zeros(N, 1);
[~,code,~] = size(chrom);
chrom1 = zeros(N, N_chrom);
for n = 1:N
    for m =1:N_chrom
        for c= 1:code
            chrom1(n,m) = 2^((16-c)*chrom(n,c,m))+chrom1(n,m);%二进制转十进制
        end
    end
end
chrom_new = rand(N, N_chrom);    %对应到区间内
for i = 1:N_chrom    %每一列乘上范围
    chrom_new(:,    i)    =    (chrom1(:,    i)/(2^code-1))*(chrom_range(2,i)-chrom_range(1,
```

```
i))+chrom_range(1, i);
end
%计算测试函数值
switch Mode
    case 6
        for i= 1:N
            for j = 1:N_chrom
                x(1,j) = chrom_new(i, j);
                fitness(i) = -x(1,j)*sin(sqrt(abs(x(1,j))))+fitness(i);
            end
        end
    case 7
        for i= 1:N
            for j = 1:N_chrom
                x(1,j) = chrom_new(i, j);
                fitness(i) = x(1,j)^2-10*cos(2*pi*x(1,j))+10+fitness(i);
            end
        end
    case 8
        for i= 1:N
            for j = 1:N_chrom
                x(1,j) = chrom_new(i, j);
                y1(i) = x(1,j)^2+y1(i);
                y2(i) = cos(2*pi*x(1,j))+y2(i);
            end
            fitness(i) =-20*exp(-0.2*sqrt(y1(i)/N_chrom))-exp(y2(i)/N_chrom)+20+exp(1);
        end
    case 9
        for i= 1:N
            for j = 1:N_chrom
                x(1,j) = chrom_new(i, j);
                y1(i) = x(1,j)^2+y1(i);
                y2(i) = cos(x(1,j)/sqrt(i))*y2(i);
            end
            fitness(i) = y1(i)/4000-y2(i)+1;
        end
    otherwise
       error('Invalid Modulation Mode. Use {6,7,8,or 9}');
end
```

子函数 4：

```
function [chrom_best,best_value]    = FindBest(chrom, fitness, N_chrom, code)
chrom_best = zeros(code, N_chrom);
best_value = 0;
[minNum, minCorr] = min(fitness);
```

```
chrom_best(:,:) = chrom(minCorr,:,:);
best_value = minNum;
```

子函数 5：

```
function fitness_ave = CalAveFitness(fitness)
[N ,~] = size(fitness);
fitness_ave = sum(fitness)/N;
```

子函数 6：

```
function chrom_new = MutChrom(chrom, mut, N,N_chrom)
[~,code,~] = size(chrom);
for i = 1:N
    for j =1:N_chrom
        for c= 1:code
            mut_rand = rand; %是否变异
            if mut_rand <=mut
                chrom(i,c,j) = 1-chrom(i,c,j);
            end
        end
    end
end
chrom_new = chrom;
```

子函数 7：

```
function chrom_new = AcrChrom(chrom, acr, N,N_chrom)
for i = 1:N
    acr_rand= rand;
    ifacr_rand<acr %如果交叉
        acr_chrom = floor((N-1)*rand+1); %要交叉的染色体
        acr_node = floor((N_chrom-1)*rand+1); %要交叉的节点
         %交叉开始
         temp= chrom(i,:,acr_node);
        chrom(i,:,acr_node) = chrom(acr_chrom,:,acr_node);
        chrom(acr_chrom,:,acr_node) = temp;
    end
end
chrom_new = chrom;
```

子函数 8：

```
function [chrom_new, fitness_new] =ReplaceWorse(chrom, chrom_best, best_value, fitness,
code)
max_num = max(fitness);
min_num = min(fitness);
limit = (max_num-min_num)*0.8+min_num;
replace_corr = fitness>limit;
replace_num = sum(replace_corr);
[~,~,N_chorm] = size(chrom);
chrom_best1 = zeros(replace_num, code, N_chorm);
```

```
for r = 1:replace_num
    chrom_best1(r,:,:) = chrom_best(:,:);
end
chrom(replace_corr,:,:) = chrom_best1;
fitness(replace_corr) =ones(replace_num,1)*best_value;
chrom_new = chrom;
fitness_new = fitness;
end
```

子函数 9：

```
function PlotModel(chrom,Mode)
x = chrom(1);
y = chrom(2);
z = chrom(3);
figure(2)
scatter3(x, y, z, 'ko')
hold on

switch Mode
    case 6
        [X,Y] = meshgrid(-500:1:500);
        fori = 1:1001
            for j = 1:1001
                Z(i,j) = -X(i,j)*sin(sqrt(abs(X(i,j))))-Y(i,j)*sin(sqrt(abs(Y(i,j))));
            end
        end
       mesh(X, Y, Z)
    case 7
        [X,Y] = meshgrid(-5.12:0.01:5.12);
        fori = 1:1025
            for j = 1:1025
                Z(i,j) =X(i,j)^2-10*cos(2*pi*X(i,j))+10+Y(i,j)^2-10*cos(2*pi*Y(i,j))+10;
            end
        end
       mesh(X, Y, Z)
    case 8
        [X,Y] = meshgrid(-32:0.1:32);
        fori = 1:641
            for j = 1:641
                Z(i,j)
=-20*exp(-0.2*sqrt(X(i,j)^2/2+Y(i,j)^2/2))-exp(cos(2*pi*X(i,j))/2+cos(2*pi*Y(i,j))/2)+20+exp(1);
             end
        end
       mesh(X, Y, Z)
    case 9
```

```
        [X,Y] = meshgrid(-600:01:600);
        fori = 1:1201
            for j = 1:1201
                Z(i,j) = (X(i,j)^2+Y(i,j)^2)/4000-cos(X(i,j))*cos(Y(i,j)/sqrt(2))+1;
            end
        end
       mesh(X, Y, Z)
otherwise
    error('Invalid Modulation Mode. Use {6,7,8,or 9}');
```

本文原载微信公众号“重庆无线通信技术实验室”。

作者简介：孙国栋，重庆大学无线通信技术实验室硕士研究生，主研方向为高速移动通信场景中的信道估计。

重庆无线通信技术实验室