计算智能程序设计

遗传算法和群智能算法程序设计及实例应用

天天向尚磊　lei_tech@qq.com

摘要

本文主要包含以下内容：

遗传算法和粒子群算法的程序设计的一般结构。主要介绍两类算法的程序设计中的主要思想。

介绍在 Matlab 编程中一些需要注意的细节。在实际编程实现中，结合 Matlab 语言的特色，可以将程序效率发挥到极致。

一个遗传算法实例和一个粒子群算法实例。

目录

一、遗传算法和粒子群算法

1.1 遗传算法

对于遗传算法，本文主要是介绍简单遗传算法的主体思想。

1.1.1 算法

1．种群初始化
2．开始循环（循环 N 次后终止）
  2.1．从上一次的种群中选取父体
    （即形成新的种群，被选择的概率根据每个染色体的适应度值的不同而不同）
  2.2．选取若干染色体杂交
    （染色体的选择是根据既定的概率）
  2.3．选取染色体进行变异
    （染色体的变异仍然是根据预先设置的概率）
  2.4．记录每条染色体的适应度值
    （记录的目的有两个：1 为下一次父体选择服务；2 保留最终的最优结果）
3．输出结果

1.1.2 注意事项

针对遗传算法下面作几点说明：

1. 适应度与编码

针对具体问题，遗传算法首要面对的是：编码方式的选择和适应度函数的选择。两者的影响主要有两方面：一是对结果好坏的影响；二是对计算复杂度的影响。

2. 概率常数设置

接下来就是概率常数的设置。一是染色体杂交时所设置的概率，二是染色体变异时所设置的概率。概率设置的不同对算法的收敛快慢影响比较大，针对一类问题，可以根据经验获取经验参数。

3. 迭代终止条件的选取

上述算法时采取的既定迭代次数停止作为终止，另外还可以设置迭代多少次适应度值改变不大而跳出循环，也可设置达到预定使用度值即可跳出循环。即一类是既定次数停止，一类是根据适应度值停止，根据具体问题可自定义设置迭代终止条件，以避免死循环。

1.1.3Matlab 编程注意事项

1．适应度函数选取

适应度函数的选取要作到在对结果影响预估不会太大的情况下，尽量选择简单的函数，即计算量小的函数。其原因在于，Matlab 在执行遗传算法时，函数的计算量占据了很大一部分时间。

1． 分多文档保存各部分程序

分多个文档可提高程序运行效率，另外也可将各部分职能更加清晰的呈现，也使得程序的

调试简化。

2． 编码

若是用遗传算法求解函数优化问题，Matlab 中有自带的二进制字符串和十进制之间的转换函数。编码多用向量或矩阵计算，因此特别注意尽量多的采用向量或者矩阵的方式计算，而不是采用循环，Matlab 中循环结构效率并不高。即便是采用循环也尽量做到列优先。

1.2 粒子群算法

粒子群算法相对于上节的遗传算法在编程实现上简单。在某些注意事项上和遗传算法相同之处。

1.2.1 算法

1．初始化粒子群
2．开始循环（迭代 N 次停止）
    2.1．粒子之间互换信息（即获取相互的适应度值信息）
    2.2．各粒子根据获取的信息调整位置和速度
3．输出结果

1.2.2 注意事项

1． 领域拓扑结构选取

即粒子之间互换信息的单位或结构，思路最简单的即以整个群为一个互换信息单位，互换信息主要涉及到适应度函数值的计算，另外，结构的选取，直接关系到每一步粒子位置和速度的更新情况。因此，它一方面影响收敛速度，一方面对计算复杂度和计算量有很大影响。

2． 在参数设置和适应度函数选择上和遗传算法类似。

1.2.3Matlab 编程注意事项

1． 适应度函数的计算

在粒子群间信息互换时，主要涉及到的计算量就是适应度函数值的计算，可采取向量化计算。即每个粒子信息交换的最小单位的适应度函数值可实现一次性计算，这就极大的提高了程序的效率。

此处涉及的向量化计算优势将在后面的实例中得到淋漓精致的体现。

2． 另外，适应度函数可单独编写 function 或者采用匿名函数。其余事项和遗传算法类似。

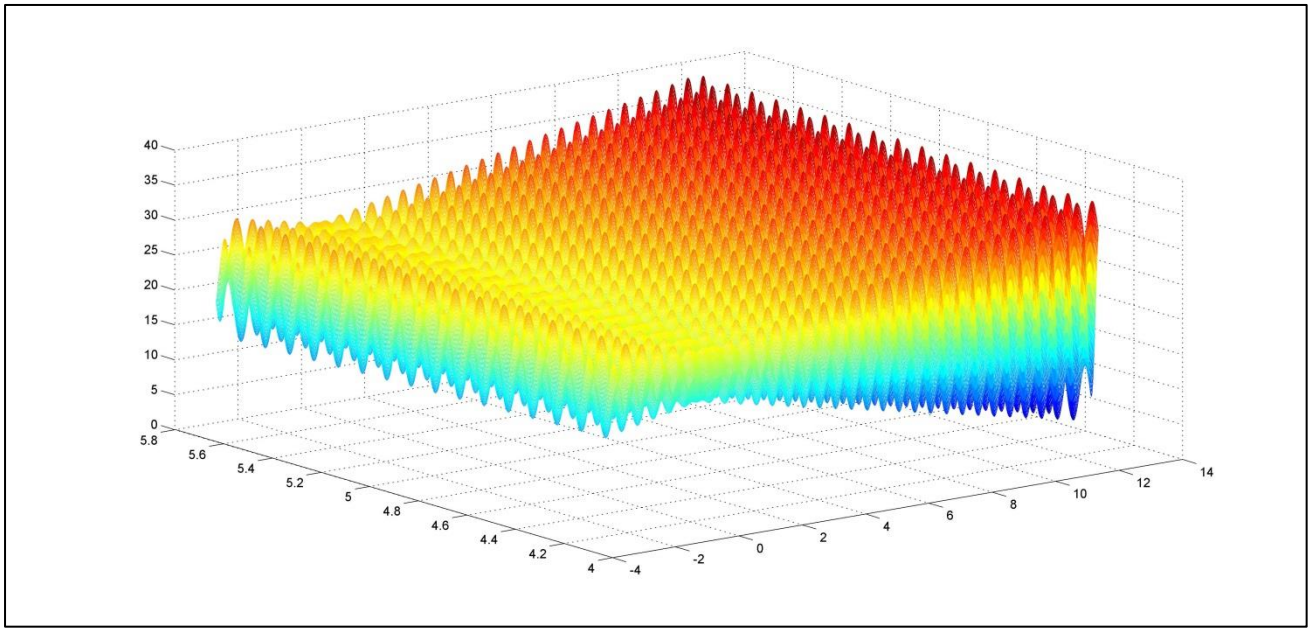
二、遗传算法和粒子群算法编程实例

2.1 遗传算法实例

本例选自《计算智能》2.2 的案例作为编程实现对象。

2.1.1 解决的问题

$$maxf(x_1,x_2) = 21.5 + x_1\sin(4\pi x_1) + x_2\sin(20\pi x_2),$$
$$-3.0 \le x_1 \le 12.1,$$
$$4.1 \le x_2 \le 5.8.$$



程序设计中选择的二进制编码，种群规模二 20，染色体杂交时的概率为 0.25，变异时概率为 0.01，适应度函数为函数本身，终止条件为达到预定最大迭代次数。

2.1.2Matlab 程序主体构成


```
function here(n)
```

```
%HERE - heredity - 《计算智能》 黄竞伟 2.2 例
%   n - 繁衍 n 代种群，不加上父代（初始第一代）
%   ChenLei
%   2012/9/15  21:44:7
```

Objective function

```
f = @(x1,x2)21.5 + x1*sin(4*pi*x1) + x2*sin(20*pi*x2);
```

Initial group

```
V = cell(1,20);
for i=1:20
    temp1 = round(rand(1,33));
    temp2 = vec2str(temp1);
    V{i}  = temp2;
end
clear temp1 temp2
```

```
% adapt value of initial group
record = adapt(V,f);        % adaptive value
maxrec = maxrecord(record,V); % keep info
```

Loop for better result

```
tic
for i=1:n
    % choose father
    V = chfather(V,record);

    % genetic operator
    V = opcrossover(V); % cross operator
    V = variation(V);   % variant operator

    record = adapt(V,f);
    temp = maxrecord(record,V);

    if (temp(1,1) > maxrec(1,1)) % keep the best value from now on
        maxrec = temp;
        mark = i;
    end
end
time = toc;
```

Output

```
format long
fprintf('     x1: %f\n',maxrec(2));
fprintf('     x2: %f\n',maxrec(3));
fprintf('f(x1,x2): %f\n',maxrec(1));
fprintf('    from: %1.0f(th) generation \n',mark);
fprintf('    time: %f s\n',time);
end
```

上述程序中所调用的函数以附件形式列出。

2.1.3 程序运行示例

- 迭代 1000 次

```
>>here(1000)
     x1: 11.628123
     x2: 5.623759
f(x1,x2): 38.725830
    from: 979(th) generation
    time: 4.559407 s
```

上面的运行结果显示了最优函数值及其对应的变量值，最优值出现在 979 代，计算时间为 4.559407s。

- 迭代 5000 次

```
>>here(5000)
     x1: 11.625128
     x2: 5.725498
f(x1,x2): 38.847806
    from: 331(th) generation
    time: 22.815248 s
```

迭代 5000 次时，实际结果比迭代 1000 次要好，但实际上，最优值在第 331 代就已经出现，后面几乎是浪费时间。由此看来，并不是迭代次数越多越好，同时也体现了遗传算法的随机性。因此，选择好的终止条件还是需要的，以避免无谓的计算量。

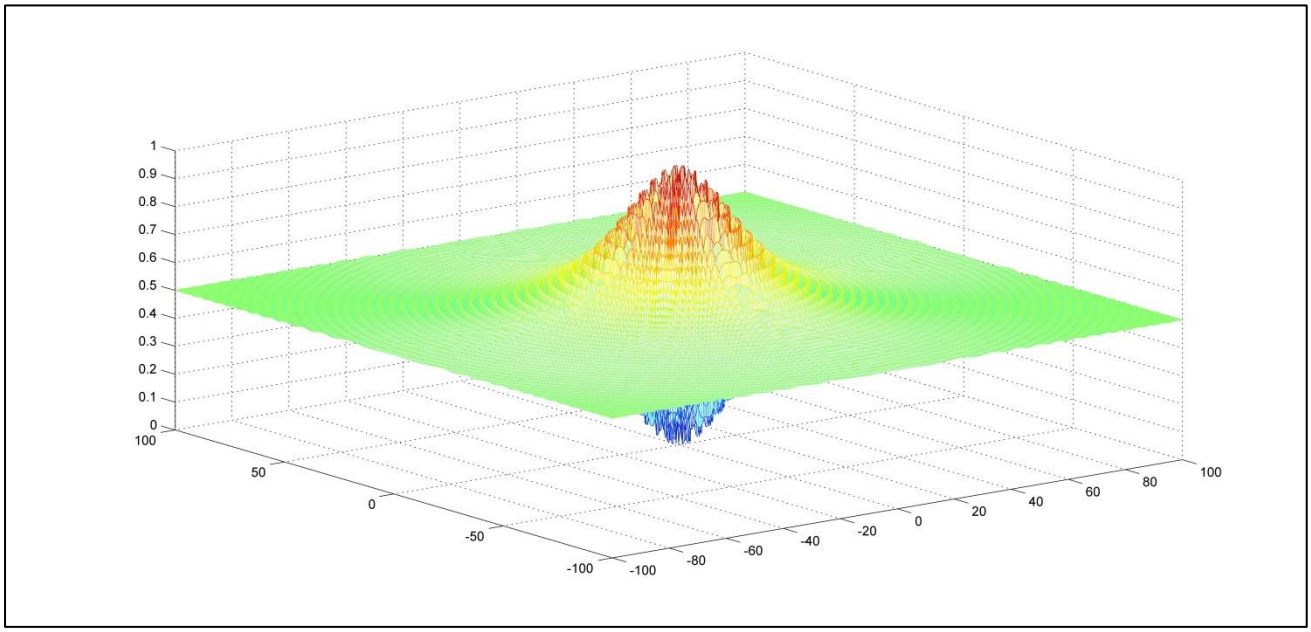
2.2 粒子群算法实例

本例选自《计算智能》7.3 实例。

2.2.1 解决的问题

$$minf(x_1,x_2)=0.5+\frac{sin^2\sqrt{x_1^2+x_2^2}-0.5}{(1+0.001(x_1^2+x_2^2))^2},$$

$$-100\le x_1,x_2\le 100.$$



此问题的最优解是已知的，其解为$f(x_1,x_2)=0$，最优解为$(0,0)$。

种群规模为 20，适应度函数取目标函数，领域结构采取全局领域结构，惯性权重为 0.9，$c_1=c_2=2$，最大速度为 3。

2.2.2Matlab 程序主体构成

下面的程序可设置迭代次数和种群规模，默认迭代为 1000 次，种群为 20。


```
function pso(N,M)
%PSO PSO algorithm
%   《计算智能》黄竞伟 page116 7.3 例
%
%   N = 1000;    % max iterative times
%   M = 20;      % Popsize
%
%   ChenLei
%   2012/9/18  20:34:16
```

Initial Setting

```
if(nargin < 2), M = 20;   end
if(nargin < 1), N = 1000; end

% set constant value
w = 0.9; c1 = 2; c2 = 2; Vmax = 3;
```

Initial Group

```
P(:,1:2) = 200*rand(M,2) - 100; % 1-2 x1 x2
P(:,3:4) = 6*rand(M,2) - 3;     % 3-4 v1 v2
[g,pbest] = pb(P);              % initial pbest and g
```

Loop for Result

```
j = 0;
tic
while j < N
    for i=1:M
        % exchange message between each other
        if(f(P(i,1),P(i,2)) < f(pbest(i,2),pbest(i,3)))
            pbest(i,2:3) = P(i,1:2);
            pbest(i,1) = f(P(i,1),P(i,2));
        end
        g = pbest(i,:);          % update g by Piself
        g = betterNei(pbest,i,g); % update g vs neighbour

        % update position and speed of Pi
        % update Px
        P(i,3:4) = w*P(i,3:4)+c1*rand*(pbest(i,1:2)-P(i,1:2)) ...
                   + c2*rand*(g(2:3)-P(i,1:2));
        P(i,3:4) = P2Vmax(P(i,3:4),Vmax);
        % update Pv
        P(i,1:2) = P(i,1:2) + P(i,3:4);
    end
    j = j + 1;
end
time = toc;
```

Output

```
result(g,time);  % output
end
```

此程序所调用函数将在附录中列出。

2.2.3 程序运行示例

■ 迭代1000次，种群规模20

```
>>pso(1000,20)
     x1: 0.000753207234
     x2: -0.000095197173
f(x1,x2): 0.000000576960
    time: 2.335782358416 s
```

上面的运行结果已经很接近真实解了，计算时间才 2s。而实际上在先前的程序版本是分开计算适应度值的，其计算时间达到了 10s 以上，差距是很大的。所以，注意结合 Matlab 的语言特色提高程序效率是很必要的。

■ 迭代 1000 次，种群规模 50

```
>>pso(1000,50)
     x1: -0.000066685360
     x2: 0.000121684686
f(x1,x2): 0.000000019273
```

```
    time: 6.032649268323 s
```

■ 迭代 5000 次，种群规模 20

```
>>pso(5000,20)
     x1: -0.000074625736
     x2: 0.000013028288
f(x1,x2): 0.000000005744
    time: 11.364202781232 s
```

迭代 5000 次，种群规模 50

```
>>pso(5000,50)
     x1: -0.000017378976
     x2: -0.000004150803
f(x1,x2): 0.000000000320
    time: 30.087009315094 s
```

从上面的几个结果来看，增加迭代次数和种群规模对解的精度提高并不是十分明显。

三、参考文献

黄竞伟等. 计算智能. 北京. 科学出版社. 2010

附录

1 遗传算法 Matlab 程序调用子函数

```
function record = adapt(V,f)
% 计算2进制的适应值
record = zeros(20,1,'double');
for i =1:20
    temp1 = V{i};
    temp2 = temp1(1:18);
    temp3 = temp1(19:end);
    record(i,1) = f(bin_x(temp2,1),bin_x(temp3,2));
end
end
```

```
function num = bin_x(bin,opt)
% 直接将2进制字符串转化为题中x1
switch opt
    case 1
        num = -3.0 + bin2dec(bin)*((12.1 - (-3.0))/(2^18 - 1));
    case 2
        num = 4.1 + bin2dec(bin)*((5.8 - 4.1)/(2^15 - 1));
end
end
```

```
function V = chfather(V,record)
% choose father
F = sum(record);  % 计算所有染色体适应值之和
```

```
pk = record/F;    % 计算每个染色体选择概率
% 计算每个染色体累积概率
qk = zeros(20,1,'double');
for i=1:20
    qk(i) = sum(pk(1:i));
end
% 模拟转动轮盘20次
r = rand(20,1);  % r存储随机数，同时后续保存选择的染色体号
for i=1:20
    k = 1;
    while (r(i)>qk(k))
        k = k + 1;
    end
    r(i) = k;
end
% 保存选择的新序列
temp = V;
for i=1:20
    V{i} = temp{r(i)};
end
end
```

```
function maxrec = maxrecord(record,V)
% keep the max value
[m,i] = max(record);
maxrec(1,1) = m;
temp1 = V{i};
temp2 = temp1(1:18);
temp3 = temp1(19:end);
maxrec(2,1) = bin_x(temp2,1);
maxrec(3,1) = bin_x(temp3,2);
end
```

```
function [g1,g2] = onecross(gene1,gene2,pos)
len = numel(gene1);
g1 = gene1(1:pos);
g2 = gene2(1:pos);
g1(pos+1:len) = gene2(pos+1:end);
g2(pos+1:len) = gene1(pos+1:end);
end
```

```
function V = opcrossover(V)
% one point crossover
pc = 0.25;
l = 1;
while (l == 1)
    r = rand(20,1);
    mk = find(r < pc);
    l = numel(mk);
end
if (mod(numel(mk),2) == 1)
    mk = mk(1:end-1);
end
r1 = randi([1,32],1,numel(mk)/2);
for i=1:numel(mk)/2
    [V{mk(2*(i-1)+1)},V{mk(2*(i-1)+2)}] = onecross(V{mk(2*(i-1)+1)},...
        V{mk(2*(i-1)+2)},r1(i));
end
end
```

```
function gnew = vari(gold,pos)
gnew = gold;
if (gnew(pos) == '1')
    gnew(pos) = '0';
else
    gnew(pos) = '1';
end
end
```

```
function V = variation(V)
% variate operater
pm = 0.01;
for i=1:20
    r = rand(33,1);
    k = find(r < pm);
    for j=1:numel(k)
        V{i} = vari(V{i},k(j));
    end
end
end
```

```
function str = vec2str(vec)
%CHAR2STR convert num vector to string.
len = length(vec);
str = '';
for i=1:len
    temp = num2str(vec(i));
    str = strcat(str,temp);
end
end
```

2 粒子群算法 Matlab 程序调用子函数

```
function g = betterNei(pbest,i,g)
%BETTERNEI 在邻居中找最好的替代g
M = size(pbest,1);
pbe(1:i-1,:) = pbest(1:i-1,2:3);
pbe(i:M-1,:) = pbest(i+1:M,2:3);
temp0 = f(pbe(:,1),pbe(:,2));  % except i
[~,temp] = min(temp0);         % minimum index
temp = min(temp);             % min (minimum index)
g(1) = temp0(temp);
g(2:3) = pbe(temp,:);
end
```

```
function f = f(x1,x2)
f = 0.5+((sin((x1.^2+x2.^2).^0.5).^2) - 0.5)./(1+0.001*(x1.^2+x2.^2)).^2;
```

```
function P = P2Vmax(P,Vmax)
% > Vmax
P(P > Vmax) = Vmax;
% < -Vmax
P(P < -Vmax) = -Vmax;
end
```

```
function [g,pbes] = pb(P)
pbes(:,1) = f(P(:,1),P(:,2)); % first col save f(x1,x2)
```

```
pbes(:,2:3) = P(:,1:2);
[~,temp1] = min(pbes);       % minimum
temp1 = min(temp1);          % avoid repeative value
g(1) = pbes(temp1);
g(2:3) = P(temp1,1:2);
end
```

```
function result(g,time)
fprintf('     x1: %10.12f\n',g(2));
fprintf('     x2: %10.12f\n',g(3));
fprintf('f(x1,x2): %10.12f\n',g(1));
fprintf('    time: %10.12f s\n',time);
end
```