学院：数据科学与计算机学院　　专业：计算机科学与技术
学号：郑 康 泽　　学号：17341213

# 智能控制与计算智能

## 第十章作业

---

10-3　在7.2.4节的BP神经网络逼近算法仿真实例中，试采用遗传算法进行BP神经网络学习参数及权值的优化设计，并进行Matlab仿真。

PS　7.2.4节没有BP网络逼近仿真实例，7.2.5节才有，题目应该是搞错了。

一开始我的思路是，先利用遗传算法搜索出BP神经网络合适的权重初值$\mathbf{w}_1$、$\mathbf{w}_2$，然后再利用这些初值进行BP神经网络的训练，然后比较在不用遗传算法和用遗传算法选择初值两种情况下，BP神经网络逼近误差收敛的速度有什么区别。结果发现，遗传算法直接找到一个合适的权重，让BP神经网络成功逼近对象。
个体的适应度定义为BP神经网络的逼近误差$e$的绝对值的倒数，目标函数就是个体适应度的倒数。代码如下：

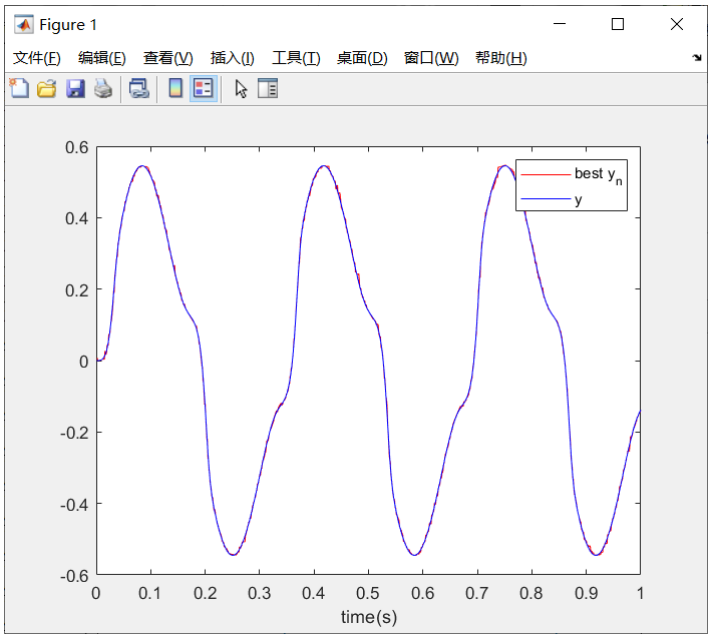
```
% 遗传算法进行BP神经网络学习参数及权值的优化设计
clear;
clc;
close;

population = 80;                                % 种群数量
generation = 1000;                              % 繁衍代数
code_len = 10;                                  % 一个变量的编码长度

w_max = 1.0;                                    % 权值的最大值
w_min = -1.0;                                   % 权值的最小值
gene = round(rand(population, (6 * 2 + 6 * 1) * code_len)); % 所有权值的编码

x = [0, 0];                                     % 网络的输入
y_1 = 0;                                        % 上一步的y
ts = 0.001;                                     % 采样时间
```

```
for k = 1:generation
    time(k) = k * ts;                    % x轴

    u(k) = 0.50 * sin(3 * 2 * pi * k * ts);
    y(k) = u(k)^ 3  + y_1 / (1 + y_1^2);

    for p = 1:population
        % 解码获得w1，w2
        w1 = zeros(2, 6);
        w2 = zeros(6, 1);
        g = gene(p, :);

        % w1
        w1_g = g(1: 12 * code_len);
        for i = 1: 12
            tmp = 0;
            for j = 1: code_len
                tmp = tmp + w1_g((i - 1)* code_len + j) * 2^(j-1);
            end
            w1(i) = (w_max - w_min) * tmp /(2^code_len - 1) + w_min;
        end

        % w2
        w2_g = g(12 * code_len + 1: 18 * code_len);
        for i = 1: 6
            tmp = 0;
            for j = 1: code_len
                tmp = tmp + w2_g((i - 1) * code_len + j) * 2^(j-1);
            end
            w2(i) = (w_max - w_min) * tmp / (2^code_len - 1) + w_min;
        end

        % 网络输出及误差
        h_in = x * w1;
        h_out = 1 ./ (1 + exp(-h_in));
        yn(p) = h_out * w2;
        e(p) = y(k) - yn(p);

        F(p) = 1 / abs(e(p));            % 个体适应度
    end

    J = 1 ./ F;                          % 目标函数
    best_J(k) = min(J);

    f = F;
    [order_f, index_f] = sort(f);        % 按适应度从小到大排序
    best_f(k) = order_f(population);     % 最大适应度
    best_yn(k) = yn(index_f(population));   % 最好的逼近输出
    best_g = gene(index_f(population), :);  % 最好的基因

    % 复制
    sum_f = sum(f);
    f_p = floor((order_f / sum_f) * population);
    kk = 1;
    for i = 1: population
        for j = 1: f_p(i)
            tmp_gene(kk, :) = gene(index_f(i), :);
```

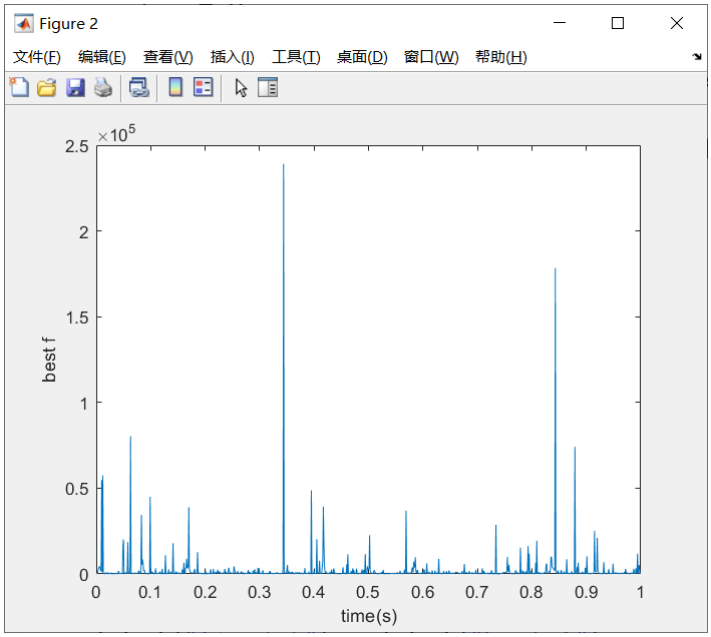
```
            kk = kk + 1;
        end
    end

    % 交叉
    pc = 0.60;
    n = ceil((6 * 2 + 6* 1) * code_len * rand);
    for i = 1: 2: (population - 1)
        tmp = rand;
        if pc > tmp
            for j = n: (6 * 2 + 6* 1) * code_len
                tmp_gene(i, j) = gene(i + 1, j);
                tmp_gene(i + 1, j) = gene(i, j);
            end
        end
    end
    tmp_gene(population, :) = best_g;

    % 变异
    pm = 0.10;
    for i = 1: population
        for j = 1: (6 * 2 + 6* 1) * code_len
            tmp = rand;
            if pm > tmp
                if tmp_gene(i, j) == 0
                    tmp_gene(i, j) = 1;
                else
                    tmp_gene(i, j) = 0;
                end
            end
        end
    end
    tmp_gene(population, :) = best_g;

    % 下一代基因
    gene = tmp_gene;

    % 更新输入和上一步数据
    x(1) = u(k);
    x(2) = y(k);
    y_1 = y(k);
end

% 解码获得最优基因的w1, w2
w1 = zeros(2, 6);
w2 = zeros(6, 1);
g = best_g;

% w1
w1_g = g(1: 12 * code_len);
for i = 1: 12
    tmp = 0;
    for j = 1: code_len
        tmp = tmp + w1_g((i - 1)* code_len + j) * 2^(j-1);
    end
    w1(i) = (w_max - w_min) * tmp /(2^code_len - 1) + w_min;
end
```

```
% w2
w2_g = g(12 * code_len + 1: 18 * code_len);
for i = 1: 6
    tmp = 0;
    for j = 1: code_len
        tmp = tmp + w2_g((i - 1) * code_len + j) * 2^(j-1);
    end
    w2(i) = (w_max - w_min) * tmp / (2^code_len - 1) + w_min;
end

disp("w1");
disp(w1);
disp("w2");
disp(w2);

figure(1);
plot(time, best_yn, 'r', time, y, 'b');
xlabel("time(s)");
legend("best y_n", "y")
figure(2);
plot(time, best_f);
xlabel("time(s)"); ylabel("best f");
```

逼近结果如下：




每一代最优的适应度如下：




---

10-4　参考2.3节专家PID控制的鉴定方法，对PID调节参数进行二进制编码，采用遗传算法实现PID调节参数的在线整定，试给出遗传算法设计过程，并进行Matlab仿真。

遗传算法实现PID调节参数的在线整定的设计过程如下：

1. 确定决策变量和约束条件：决策变量是PID三个调节参数：比例单元P、积分单元I和微分单元D，约束比例单元P的范围是0.3～1.0，约束积分单元I的范围是0.01～0.10，约束微分单元的范围是0.01～0.05。
2. 确定编码方法：用长度为10的二进制编码串来分别表示三个决策变量P、I、D，从而将决策变量的定义域离散为1023个均等的区域，包括两个断点在内共有1024个不同的离散点。再将分别表示三个决策变量的三个10位长的二进制编码串连接在一起，组成一个30位长的二进制编码串。
3. 确定解码方法：解码时需要将30位长的二进制编码串切断为三个10位长的二进制编码串，然后分别将它们转换为对应的十进制整数，然后将十进制整数转为对应的决策变量的值，例如决策变量P，假设得到的十进制整数为$z$, 那么P的值应该为$p=(1.0-0.3)\times\frac{z}{1023}+0.3$。
4. 确定个体评价方法：个体的适应度取为期望轨迹$r$与基于该个体得到的被控对象的输出$y$之差的绝对值，即个体的适应度为$F=|r-y|$。选个体适应度的倒数作为目标函数$J=\frac{1}{F}$。

5. 设计遗传算子：选择运算使用比例算子，交叉运算使用单点交叉算子，变异运算使用基本为变异算子。
6. 确定遗传算法的运行参数：群体大小$M=80$，终止进化代数$G=500$，交叉概率$P_c=0.6$，变异概率$P_m=0.1$。

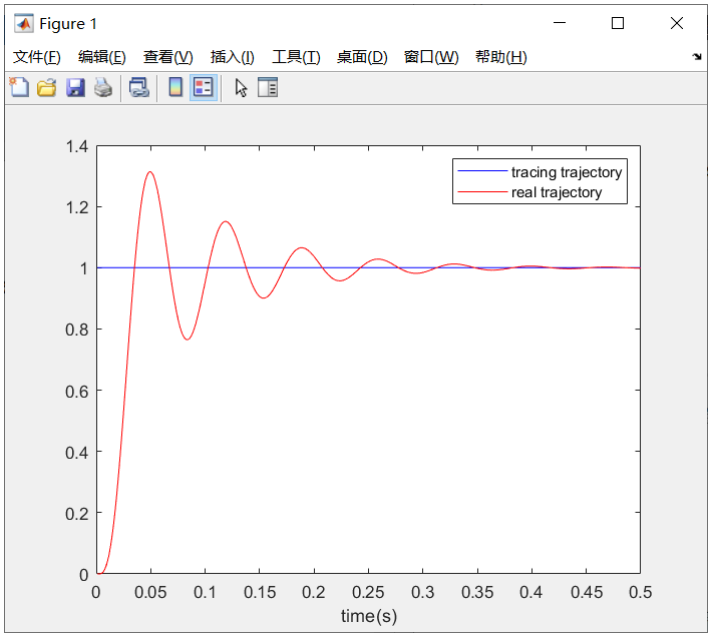
设计完遗传算法，要确定控制对象的输入$u$，这里我们选择每一代中适应度最高的个体来确定$u$，同理，每代的误差也是取适应度最高的个体得到的误差。
代码如下：

```
% 遗传算法调节PID参数
clear;
clc;
close;

ts = 0.001;              % 采样间隔
sys = tf(523500, [1, 87.35, 10470, 0]); % 建立传递函数模型
dsys = c2d(sys, ts, 'z');               % 离散化模型
[num, den] = tfdata(dsys, 'v');         % 离散模型的分子分母

x = [0, 0, 0]';                         % 设计u用到
u_3 = 0; u_2 = 0; u_1 = 0;              % 前三部的u
y_3 = 0; y_2 = 0; y_1 = 0;              % 前三步的y

generation = 500;                       % 种群代数
population = 80;                        % 种群个数
code_len = 10;                          % 一个变量的编码长度

kp_max = 1.0;                           % kp最大值
kp_min = 0.3;                           % kp最小值
ki_max = 0.1;                           % ki最大值
ki_min = 0.01;                          % ki最小值
kd_max = 0.05;                          % kd最大值
kd_min = 0.01;                          % kd最小值

gene = round(rand(population, 3 * code_len));   % 三个变量的编码

for k = 1: generation
    time(k) = k * ts;                   % x轴
    r(k) = 1.0;                         % 期望轨迹

    for p = 1: population
        % 解码获得三个参数
        kp = 0;
        ki = 0;
        kd = 0;
        g = gene(p, :);

        % kp
        kp_g = g(1: code_len);
        tmp = 0;
        for i = 1:code_len
            tmp = tmp + kp_g(i) * 2^(i-1);
```

```
        end
        kp = (kp_max - kp_min) * tmp / (2^code_len - 1) + kp_min;

        % ki
        ki_g = g(code_len + 1: 2 * code_len);
        tmp = 0;
        for i = 1:code_len
            tmp = tmp + ki_g(i) * 2^(i-1);
        end
        ki = (ki_max - ki_min) * tmp / (2^code_len - 1) + ki_min;

        % kd
        kd_g = g(2 * code_len + 1: 3 * code_len);
        tmp = 0;
        for i = 1:code_len
            tmp = tmp + kd_g(i) * 2^(i-1);
        end
        kd = (kd_max - kd_min) * tmp / (2^code_len - 1) + kd_min;

        u(p) = kp * x(1) + ki * x(2) * kd * x(3);   % 个体得到的u
        % 个体得到的y
        y(p) = -den(2) * y_1 - den(3) * y_2 - den(4) * y_3 +...
            num(1) * u(p) + num(2) * u_1 + num(3) * u_2 + num(4) * u_3;
        e(p) = r(k) - y(p);              % 跟踪误差

        F(p) = 1 / abs(e(p));            % 个体适应度
    end

    J = 1 ./ F;
    best_J(k) = min(J);                  % 目标函数

    f = F;
    [order_f, index_f] = sort(f);        % 按适应度从小到大排序
    best_f(k) = order_f(population);     % 最大适应度
    best_g = gene(index_f(population)); % 最好的跟踪输出
    best_u(k) = u(index_f(population)); % 最好的对象输入
    best_y(k) = y(index_f(population)); % 最好的对象输出
    best_e(k) = e(index_f(population)); % 最小的跟踪误差


    % 复制
    sum_f = sum(f);
    f_p = floor((order_f / sum(f)) * population);
    kk = 1;
    for i = 1: population
        for j = 1: f_p(i)
            tmp_gene(kk, :) = gene(index_f(i), :);
            kk = kk + 1;
        end
    end

    % 交叉
    pc = 0.60;
    n = ceil(3 * code_len * rand);
    for i = 1: 2: (population - 1)
        tmp = rand;
        if pc > tmp
            for j = n: 3 * code_len
```

```
                tmp_gene(i, j) = gene(i + 1, j);
                tmp_gene(i + 1, j) = gene(i, j);
            end
        end
    end
    tmp_gene(population, :) = best_g;

    % 变异
    pm = 0.10;
    for i = 1: population
        for j = 1: 3 * code_len
            tmp = rand;
            if pm > tmp
                if tmp_gene(i, j) == 0
                    tmp_gene(i, j) = 1;
                else
                    tmp_gene(i, j) = 0;
                end
            end
        end
    end
    tmp_gene(population, :) = best_g;

    % 下一代基因
    gene = tmp_gene;

    % 更新记录
    u_3 = u_2; u_2 = u_1; u_1 = best_u(k);
    y_3 = y_2; y_2 = y_1; y_1 = best_y(k);

    x(3) = (best_e(k) - x(1)) / ts;
    x(1) = best_e(k);
    x(2) = x(2) + best_e(k) * ts;
end

figure(1);
plot(time, r, 'b', time, best_y, 'r');
xlabel("time(s)");
legend("tracing trajectory", "real trajectory");
```

跟踪结果如下：




---

10-5　分别利用粒子群算法和差分进化算法辨识如下非线性动态模型参数并进行比较分析。

$$G(s)=\frac{K}{(T_1 s+1)(T_2 s+1)}\mathrm{e}^{-Ts}$$

其中，参数真实值为$K=2$，$T_1=1$，$T_2=20$，$T=0.8$。

试给出差分进化算法设计过程，并进行Matlab仿真。

1. 利用粒子群算法辨识非线性动态模型参数的代码如下：

```
clear;
close;
% clc;

start_time = clock;                    % 开始时间

% 获取样本
[N, s, G] = make_data;

param_min = [0, 0, 0, 0];              % 参数的最小值
param_max = [5, 5, 50, 5];             % 参数的最大值
v_max = 1;                             % 速度的最大值
v_min = -1;                            % 速度的最小值
population = 80;                       % 种群大小
param_num = 4;                         % 参数的数量
generation = 500;                      % 种群代数
c1 = 1.3; c2 = 1.7;                    % 学习因子
w_max = 0.9; w_min = 0.1;              % 惯性权重的最大值和最小值

% 线性递减的惯性权重
```

```
for i = 1: generation
    w(i) = w_max - ((w_max - w_min) / generation) * i;
end

% 初始化个体及速度
for i = 1: param_num
    param(:, i) = param_min(i)  +(param_max(i) - param_min(i)) *
rand(population, 1);
    v(:, i) = v_min + (v_max - v_min) * rand(population, 1);
end

% 找到全局最优个体及其目标函数
global_best_param = param(1, :);          % 全局最优个体
global_best_J = 1e10;                     % 全局最优个体的目标函数值
for i = 1: population
    J(i) = cal_J(param(i, :), N, s, G);      % 个体的目标函数值
    local_best_param(i, :) = param(i, :);    % 个体历史最优
    if J(i) < global_best_J
        global_best_param = param(i, :);
        global_best_J = J(i);
    end
end

% 开始搜索
for k = 1: generation
    time(k) = k;                          % x轴

    for i = 1: population
        % 更新速度
        v(i, :) = w(k) * v(i, :) + c1 * rand *...
            (local_best_param(i, :) - param(i, :)) + c2 * rand *...
            (global_best_param - param(i, :));

        % 限定边界
        for j = 1: param_num
            if v(i, j) < v_min
                v(i, j) = v_min;
            elseif v(i, j) > v_max
                v(i, j) = v_max;
            end
        end

        % 更新个体
        param(i, :) = param(i, :) + v(i, :);

        % 限定边界
        for j = 1: param_num
            if param(i, j) < param_min(j)
                param(i, j) = param_min(j);
            elseif param(i, j) > param_max(j)
                param(i, j) = param_max(j);
            end
        end

        % 变异
        if rand > 0.8
            n = ceil(param_num * rand);
            param(i, n) = param_max(n) * rand;
```

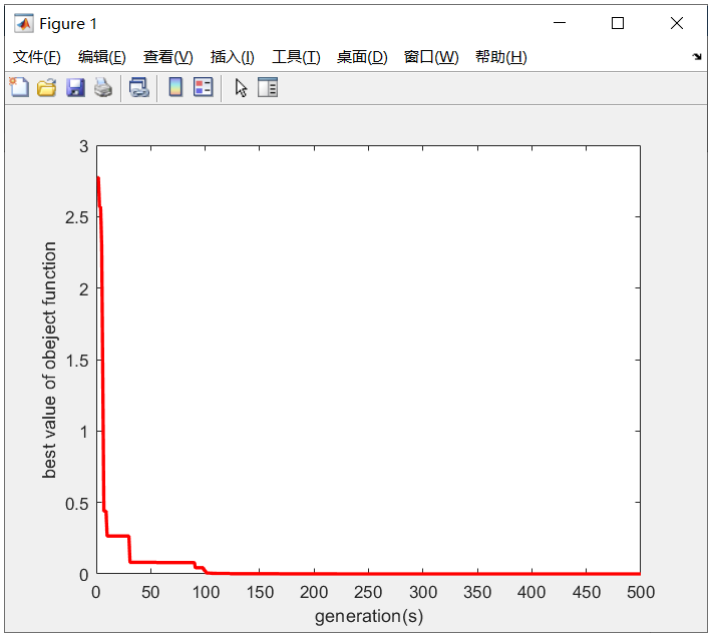
```
        end

        % 更新个体的历史最优
        new_J = cal_J(param(i, :), N, s, G);
        if  new_J < J(i)
            J(i) = new_J;
            local_best_param(i, :) = param(i, :);
        end

        % 更新全局最优
        if J(i) < global_best_J
            global_best_param = param(i, :);
            global_best_J = J(i);
        end
    end

    % 每一代的全局最优的目标函数值
    generation_best_J(k) = global_best_J;

%     if generation_best_J(k) < eps
%         disp(k)
%     end
end

disp("true value: K = 2, T1 = 1, T2 = 20, T = 0.8");
disp("estimated value: ");
disp(global_best_param);
disp("best value of object funtion: ");
disp(global_best_J);

end_time = clock;
disp(["time spent: ", num2str((end_time(5) - start_time(5)) * 60 +
(end_time(6) - start_time(6))), "s"]);

figure(1);
plot(time, generation_best_J, 'r', "linewidth", 2);
xlabel("generation(s)"); ylabel("best value of obeject function");

% 计算目标函数
function J = cal_J(param, N, s, G)
    Kp = param(1);
    T1p = param(2);
    T2p = param(3);
    Tp = param(4);
    J = 0;
    for i = 1: N
        Gp(i) = Kp * exp(-Tp * s(i)) / (T1p * s(i) + 1) / (T2p * s(i) + 1);
        e = Gp(i) - G(i);
        J = J + 0.5 * e^2;
    end
end

% 生成样本
function [N, s, G] = make_data
    K = 2;
    T1 = 1;
    T2 = 20;
    T = 0.8;
```

```
    smin = -3;
    smax = 3;
    N = (smax - smin) / 0.1 + 1;
    index = 0;

    for i = 1: N
        index = index + 1;
        s(index) = smin + (i - 1) * 0.1;
        if T1 * s(index) + 1 == 0 || T2 * s(index) + 1 == 0 % 防止分母为零
            index = index - 1;
            s = s(1: index);
            continue;
        end
        G(index) = K * exp(-T * s(index)) / (T1 * s(index) + 1) / (T2 *
s(index) + 1);
    end
    N = index;
end
```

辨识结果：

```
true value: K = 2, T1 = 1, T2 = 20, T = 0.8
estimated value:
    2.0001    1.0000   20.0004    0.8000

best value of object funtion:
   3.9341e-09
```

每一代最优的目标函数值如下：




2. 差分进化算法设计过程如下：

    1. 生成初始群体：在4维空间里随机产生满足约束条件的$M$个个体，实施措施如下：

$$x_{ij}(0) = \mathrm{rand}_{ij}(0,1)(x_{ij}^U - x_{ij}^L) + x_{ij}^L,$$

其中，$x_{ij}^U$和$x_{ij}^L$分别是第$j$个染色体的上界和下界，$\text{rand}_{ij}(0,1)$是$[0,1]$之间的随机小数。

2. 变异操作：从群体中随机选择2个个体$x_{p_1}$和$x_{p_2}$，且$i \neq p_1 \neq p_2$，变异操作为$h(t+1) = x_b(t) + F(x_{p_1}(t) - x_{p_2}(t))$，其中$x_{p_1}(t) - x_{p_2}(t)$为差异化向量，$F$为缩放因子，$p_1$和$p_2$为随机整数，表示个体在种群中的序号，$x_b(t)$为当前代中种群中最好的个体。

3. 交叉操作：具体操作如下：

$$v_{ij}(t+1) = \begin{cases} h_{ij}(t+1), & \text{rand } l_{ij} \leq \text{CR} \\ x_{ij}(t), & \text{rand } l_{ij} > \text{CR}, \end{cases}$$

其中，rand $l_{ij}$为$[0,1]$之间的随机小数，CR为交叉概率，$\text{CR} \in [0,1]$。

4. 选择操作：为了确认$x_i(t)$是否成为下一代的成员，试验向量$v_i(t+1)$和目标向量$x_i(i)$对评价函数进行比较：

$$x_i(t+1) = \begin{cases} v_i(t+1), & f(v_{i1}(t+1), \ldots, v_{i4}(t+1)) > f(x_{i1}(t), \ldots, x_{i4}(t)) \\ x_i(t), & f(v_{i1}(t+1), \ldots, v_{i4}(t+1)) \leq f(x_{i1}(t), \ldots, x_{i4}(t)). \end{cases}$$

反复执行步骤2至步骤4操作，直至达到最大的进化代数$G$，以上就是差分进化算法的设计过程。

利用差分进化算法辨识非线性动态模型参数的代码如下：

```
clear;
% clc;
close;

start_time = clock;                    % 开始时间

% 获取样本
[N, s, G] = make_data;

param_min = [0, 0, 0, 0];              % 参数的最小值
param_max = [5, 5, 50, 5];             % 参数的最大值
param_num = 4;                         % 参数的个数
population = 80;                       % 种群大小
generation = 500;                      % 种群代数
f = 0.8;                               % 缩放因子
cr = 0.6;                              % 交叉概率

% 随机初始化个体
for i = 1: param_num
    param(:, i) = param_min(i) + (param_max(i) - param_min(i)) *
rand(population, 1);
end

% 找到最优个体及其目标函数值
best_param = param(:, 1);
best_J = 1e10;
for i = 2: population
    J = cal_J(param(i, :), N, s, G);
    if J < best_J
```

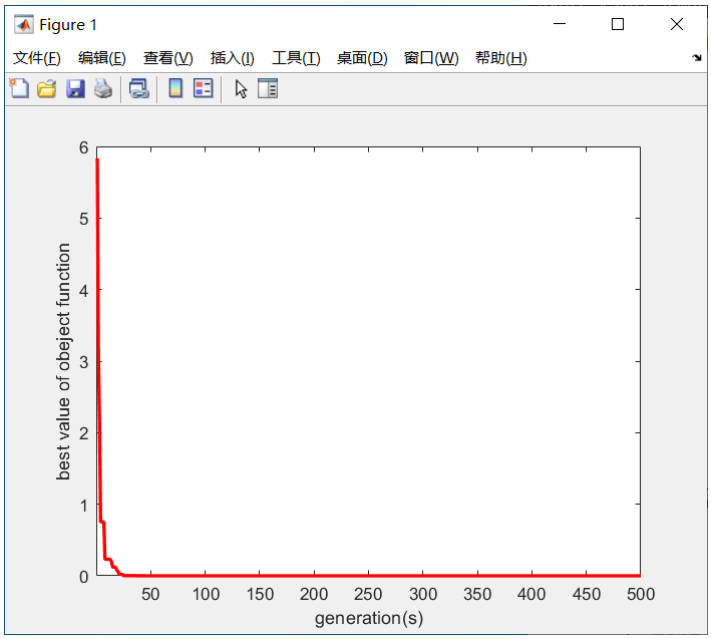
```
        best_param = param(i, :);
        best_J = J;
    end
end

% 开始搜索
for k = 1: generation
    time(k) = k;                % x轴

    for i = 1: population
        % 找到两个随机个体
        r1 = 1; r2 = 1;
        while (r1 == r2 || r1 == i || r2 == i)
            r1 = ceil(population * rand);
            r2 = ceil(population * rand);
        end
        % 变异
        h(i, :) = best_param + f * (param(r1, :) - param(r2, :));

        % 交叉
        for j = 1: param_num
            if h(i, j) < param_min(j)
                h(i, j) = param_min(j);
            elseif h(i, j) > param_max(j)
                h(i, j) = param_max(j);
            end
        end

        % 限定边界
        for j = 1: param_num
            tmp = rand;
            if cr > tmp
                v(i, j) = h(i, j);
            else
                v(i, j) = param(i, j);
            end
        end

        % 选择
        if cal_J(v(i, :), N, s, G) < cal_J(param(i, :), N, s, G)
            param(i, :) = v(i, :);
        end

        % 更新最优个体
        J = cal_J(param(i, :), N, s, G);
        if J < best_J
            best_param = param(i, :);
            best_J = J;
        end
    end

    % 每一代的最优个体的目标函数值
    generation_best_J(k) = best_J;

%     if generation_best_J(k) < eps
%         disp(k)
%     end
end
```

```
disp("true value: K = 2, T1 = 1, T2 = 20, T = 0.8");
disp("estimated value: ");
disp(best_param);
disp("best value of object funtion: ");
disp(best_J);

end_time = clock;                    % 结束时间
disp(["time spent: ", num2str((end_time(5) - start_time(5)) * 60 +
(end_time(6) - start_time(6))), "s"]);

figure(1);
plot(time, generation_best_J, 'r', "linewidth", 2);
xlabel("generation(s)"); ylabel("best value of obeject function");

% 计算目标函数
function J = cal_J(param, N, s, G)
    Kp = param(1);
    T1p = param(2);
    T2p = param(3);
    Tp = param(4);
    J = 0;
    for i = 1: N
        Gp(i) = Kp * exp(-Tp * s(i)) / (T1p * s(i) + 1) / (T2p * s(i) + 1);
        e = Gp(i) - G(i);
        J = J + 0.5 * e^2;
    end
end

% 生成样本
function [N, s, G] = make_data
    K = 2;
    T1 = 1;
    T2 = 20;
    T = 0.8;

    smin = -3;
    smax = 3;
    N = (smax - smin) / 0.1 + 1;
    index = 0;

    for i = 1: N
        index = index + 1;
        s(index) = smin + (i - 1) * 0.1;
        if T1 * s(index) + 1 == 0 || T2 * s(index) + 1 == 0 % 防止分母为零
            index = index - 1;
            s = s(1: index);
            continue;
        end
        G(index) = K * exp(-T * s(index)) / (T1 * s(index) + 1) / (T2 *
s(index) + 1);
    end
    N = index;
end
```

辨识结果如下：

```
true value: K = 2, T1 = 1, T2 = 20, T = 0.8
estimated value:
    2.0000    1.0000   20.0000    0.8000

best value of object funtion:
     0
```

每一代最优的目标函数值如下：




3. 比较分析：

    1. 在相同的最大进化代数$G = 500$下，粒子群算法所花费的时间为1.0s~1.5s，而差分进化算法所花费的时间为2.5s~3.0s，
    2. 粒子群算法最终的最优目标函数值不能降到eps以下，而差分进化算法最终的最优目标函数值为0，并且差分进化算法在进化代数大概为205代时，最优目标函数值已经降到eps以下了，因此差分进化算法的全局收敛能力更强。
    3. 粒子群算法辨识到的参数值与实际值还是有一点误差的，而差分进化算法辨识到的参数值与实际值完全一样。
    4. 差分进化算法的参数比粒子群算法的参数少。