%

function [pso F] = PSO\_2D()

% FUNCTION PSO --------USE Particle Swarm Optimization Algorithm

%global present;

% close all;

pop\_size = 10; % pop\_size 种群大小

part\_size = 2; % part\_size 粒子大小, \*\* =n-D

gbest = zeros(1,part\_size+1); % gbest 当前搜索到的最小的值

max\_gen = 80; % max\_gen 最大迭代次数

region=zeros(part\_size,2); % 设定搜索空间范围

region=[-3,3;-3,3]; % \*\*每一维设定不同范围

rand('state',sum(100\*clock)); % 重置随机数发生器状态

arr\_present = ini\_pos(pop\_size,part\_size); % present 当前位置,随机初始化,rand()的范围为0~1

v=ini\_v(pop\_size,part\_size); % 初始化当前速度

pbest = zeros(pop\_size,part\_size+1); % pbest 粒子以前搜索到的最优值，最后一列包括这些值的适应度

w\_max = 0.9; % w\_max 权系数最大值

w\_min = 0.4;

v\_max = 2; % \*\*最大速度,为粒子的范围宽度

c1 = 2; % 学习因子

c2 = 2; % 学习因子

best\_record = zeros(1,max\_gen); % best\_record记录最好的粒子的适应度。

% ————————————————————————

% 计算原始种群的适应度,及初始化

% ————————————————————————

arr\_present(:,end)=ini\_fit(arr\_present,pop\_size,part\_size);

% for k=1:pop\_size

% present(k,end) = fitness(present(k,1:part\_size)); %计算原始种群的适应度

% end

pbest = arr\_present; %初始化各个粒子最优值

[best\_value best\_index] = min(arr\_present(:,end)); %初始化全局最优，即适应度为全局最小的值，根据需要也可以选取为最大值

gbest = arr\_present(best\_index,:);

%v = zeros(pop\_size,1); % v 速度

% ————————————————————————

% 迭代

% ————————————————————————

% global m;

% m = moviein(1000); %生成帧矩阵

x=[-3:0.01:3];

y=[-3:0.01:3];

z=@(x,y) 3\*(1-x).^2.\*exp(-(x.^2) - (y+1).^2) ...

- 10\*(x/5 - x.^3 - y.^5).\*exp(-x.^2-y.^2) ...

- 1/3\*exp(-(x+1).^2 - y.^2);

for i=1:max\_gen

grid on;

plot3(x,y,z);

% subplot(121),ezmesh(z),hold on,grid on,plot3(arr\_present(:,1),arr\_present(:,2),arr\_present(:,3),'\*'),hold off;

% subplot(122),ezmesh(z),view([145,90]),hold on,grid on,plot3(arr\_present(:,1),arr\_present(:,2),arr\_present(:,3),'\*'),hold off;

ezmesh(z),hold on,grid on,plot3(arr\_present(:,1),arr\_present(:,2),arr\_present(:,3),'\*'),hold off;

drawnow

F(i)=getframe;

% ezmesh(z)

% % view([-37,90])

% hold on;

% grid on;

% % plot(-0.0898,0.7126,'ro');

% plot3(arr\_present(:,1),arr\_present(:,2),arr\_present(:,3),'\*'); %改为三维

% axis([-2\*pi,2\*pi,-pi,pi,-50,10]);

% hold off;

pause(0.01);

% m(:,i) = getframe; %添加图形

w = w\_max-(w\_max-w\_min)\*i/max\_gen;

% fprintf('# %i 代开始！\n',i);

% 确定是否对打散已经收敛的粒子群——————————————————————————————

reset = 0; % reset = 1时设置为粒子群过分收敛时将其打散，如果＝1则不打散

if reset==1

bit = 1;

for k=1:part\_size

bit = bit&(range(arr\_present(:,k))<0.1);

end

if bit==1 % bit=1时对粒子位置及速度进行随机重置

arr\_present = ini\_pos(pop\_size,part\_size); % present 当前位置,随机初始化

v = ini\_v(pop\_size,part\_size); % 速度初始化

for k=1:pop\_size % 重新计算适应度

arr\_present(k,end) = fitness(arr\_present(k,1:part\_size));

end

warning('粒子过分集中！重新初始化……'); % 给出信息

display(i);

end

end

for j=1:pop\_size

v(j,:) = w.\*v(j,:)+c1.\*rand.\*(pbest(j,1:part\_size)-arr\_present(j,1:part\_size))...

+c2.\*rand.\*(gbest(1:part\_size)-arr\_present(j,1:part\_size)); % 粒子速度更新 (a)

% 判断v的大小，限制v的绝对值小于5————————————————————————————

c = find(abs(v)>6); %\*\*最大速度设置，粒子的范围宽度

v(c) = sign(v(c))\*6; %如果速度大于3.14则，速度为3.14

arr\_present(j,1:part\_size) = arr\_present(j,1:part\_size)+v(j,1:part\_size); % 粒子位置更新 (b)

arr\_present(j,end) = fitness(arr\_present(j,1:part\_size));

if (arr\_present(j,end)>pbest(j,end))&(Region\_in(arr\_present(j,:),region)) % 根据条件更新pbest,如果是最小的值为小于号,相反则为大于号

pbest(j,:) = arr\_present(j,:);

end

end

[best best\_index] = max(arr\_present(:,end)); % 如果是最小的值为min,相反则为max

if best>gbest(end)&(Region\_in(arr\_present(best\_index,:),region)) % 如果当前最好的结果比以前的好，则更新最优值gbest,如果是最小的值为小于号,相反则为大于号

gbest = arr\_present(best\_index,:);

end

best\_record(i) = gbest(end);

end

pso = gbest;

display(gbest);

% figure;

% plot(best\_record);

% movie2avi(F,'pso\_2D1.avi','compression','MSVC');

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% 计算适应度

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function fit = fitness(present)

fit=3\*(1-present(1)).^2.\*exp(-(present(1).^2) - (present(2)+1).^2) ... %\*\*需要求极值的函数,本例即peaks函数

- 10\*(present(1)/5 - present(1).^3 - present(2).^5).\*exp(-present(1).^2-present(2).^2) ...

- 1/3\*exp(-(present(1)+1).^2 - present(2).^2);

function ini\_present=ini\_pos(pop\_size,part\_size)

ini\_present = 3\*rand(pop\_size,part\_size+1); %初始化当前粒子位置,使其随机的分布在工作空间 %\*\* 6即为自变量范围

function ini\_velocity=ini\_v(pop\_size,part\_size)

ini\_velocity =3/2\*(rand(pop\_size,part\_size)); %初始化当前粒子速度,使其随机的分布在速度范围内

function flag=Region\_in(pos\_present,region)

[m n]=size(pos\_present);

flag=1;

for j=1:n-1

flag=flag&(pos\_present(1:j)>=region(j,1))&(pos\_present(1:j)<=region(j,2));

end

function arr\_fitness=ini\_fit(pos\_present,pop\_size,part\_size)

for k=1:pop\_size

arr\_fitness(k,1) = fitness(pos\_present(k,1:part\_size)); %计算原始种群的适应度

end