### 1.main.m文件

tic

clear;

clc;

load data.mat

inputnum=2;

hiddennum=5;

outputnum=1;

input\_train=input(1:1500,:)';

input\_test=input(1501:2000,:)';

output\_train=output(1:1500)';

output\_test=output(1501:2000)';

[inputn,inputps]=mapminmax(input\_train);

[outputn,outputps]=mapminmax(output\_train);

net=newff(inputn,outputn,hiddennum,{'tansig','purelin'},'trainlm'); %%{'tansig','purelin'}为默认的激活函数（没记错的话，有兴趣的话可以试着进行调整，trainlm为默认的训练算法，Levenberg-Marquart算法)

%% 遗传算法参数初始化

maxgen=10; %进化代数，即迭代次数

sizepop=30; %种群规模

pcross=0.3; %交叉概率选择，0和1之间

pmutation=0.1; %变异概率选择，0和1之间

numsum=inputnum\*hiddennum+hiddennum+hiddennum\*outputnum+outputnum;

lenchrom=ones(1,numsum);

bound=[-3\*ones(numsum,1) 3\*ones(numsum,1)]; %数据范围

individuals=struct('fitness',zeros(1,sizepop), 'chrom',[]); %将种群信息定义为一个结构体

avgfitness=[]; %每一代种群的平均适应度

bestfitness=[]; %每一代种群的最佳适应度

bestchrom=[]; %适应度最好的染色体

for i=1:sizepop %随机产生一个种群

individuals.chrom(i,:)=Code(lenchrom,bound); %编码

x=individuals.chrom(i,:); %计算适应度

individuals.fitness(i)=fun(x,inputnum,hiddennum,outputnum,net,inputn,outputn); %染色体的适应度

end

[bestfitness bestindex]=min(individuals.fitness);

bestchrom=individuals.chrom(bestindex,:); %最好的染色体

avgfitness=sum(individuals.fitness)/sizepop; %染色体的平均适应度

trace=[avgfitness bestfitness]; % 记录每一代进化中最好的适应度和平均适应度

for num=1:maxgen

% 选择

individuals=select(individuals,sizepop);

avgfitness=sum(individuals.fitness)/sizepop;

%交叉

individuals.chrom=Cross(pcross,lenchrom,individuals,sizepop,bound);

% 变异

individuals.chrom=Mutation(pmutation,lenchrom,individuals,sizepop,num,maxgen,bound);

% 计算适应度

for j=1:sizepop

x=individuals.chrom(j,:); %个体

individuals.fitness(j)=fun(x,inputnum,hiddennum,outputnum,net,inputn,outputn);

end

%找到最小和最大适应度的染色体及它们在种群中的位置

[newbestfitness,newbestindex]=min(individuals.fitness);

[worestfitness,worestindex]=max(individuals.fitness);

% 代替上一次进化中最好的染色体

if bestfitness>newbestfitness

bestfitness=newbestfitness;

bestchrom=individuals.chrom(newbestindex,:);

end

individuals.chrom(worestindex,:)=bestchrom;

individuals.fitness(worestindex)=bestfitness;

avgfitness=sum(individuals.fitness)/sizepop;

trace=[trace;avgfitness bestfitness]; %记录每一代进化中最好的适应度和平均适应度

end

figure(1)

[r c]=size(trace);

plot([1:r]',trace(:,2),'b--');

title(['适应度曲线 ' '终止代数＝' num2str(maxgen)]);

xlabel('进化代数');ylabel('适应度');

legend('平均适应度','最佳适应度');

disp('适应度 变量');

%% 把最优初始阀值权值赋予网络预测

% %用遗传算法优化的BP网络进行值预测

x=bestchrom;

w1=x(1:inputnum\*hiddennum);

B1=x(inputnum\*hiddennum+1:inputnum\*hiddennum+hiddennum);

w2=x(inputnum\*hiddennum+hiddennum+1:inputnum\*hiddennum+hiddennum+hiddennum\*outputnum);

B2=x(inputnum\*hiddennum+hiddennum+hiddennum\*outputnum+1:inputnum\*hiddennum+hiddennum+hiddennum\*outputnum+outputnum);

net.iw{1,1}=reshape(w1,hiddennum,inputnum);

net.lw{2,1}=reshape(w2,outputnum,hiddennum);

net.b{1}=reshape(B1,hiddennum,1);

net.b{2}=reshape(B2,outputnum,1);

%% BP网络训练

%网络参数

net.trainParam.epochs=100;

net.trainParam.lr=0.1;

net.trainParam.goal=0.00001;

net.divideParam.trainRatio = 75/100; %默认训练集占比

net.divideParam.valRatio = 15/100; %默认验证集占比

net.divideParam.testRatio = 15/100; %默认测试集占比

%网络训练

[net,per2]=train(net,inputn,outputn);

%% BP网络预测

%数据归一化

inputn\_test=mapminmax('apply',input\_test,inputps);

an=sim(net,inputn\_test);

test\_simu=mapminmax('reverse',an,outputps);

error=test\_simu-output\_test;

figure(2)

plot(test\_simu,':og','LineWidth',1.5)

hold on

plot(output\_test,'-\*','LineWidth',1.5);

legend('预测输出','期望输出')

grid on

set(gca,'linewidth',1.0);

xlabel('X 样本','FontSize',15);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ylabel('Y 输出','FontSize',15);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

set(gcf,'color','w')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

title('GA-BP Network','Color','k','FontSize',15);

toc

### 2.select.m文件

function ret=select(individuals,sizepop)

% 该函数用于进行选择操作

% individuals input 种群信息

% sizepop input 种群规模

% ret output 选择后的新种群

%求适应度值倒数

fitness1=10./individuals.fitness; %individuals.fitness为个体适应度值

%个体选择概率

sumfitness=sum(fitness1);

sumf=fitness1./sumfitness;

%采用轮盘赌法选择新个体

index=[];

for i=1:sizepop %sizepop为种群数

pick=rand;

while pick==0

pick=rand;

end

for j=1:sizepop

pick=pick-sumf(j);

if pick<0

index=[index j];

break;

end

end

end

%新种群

individuals.chrom=individuals.chrom(index,:); %individuals.chrom为种群中个体

individuals.fitness=individuals.fitness(index);

ret=individuals;

### 3.Corss.m文件

function ret=Cross(pcross,lenchrom,individuals,sizepop,bound)

%本函数完成交叉操作

% pcorss input : 交叉概率

% lenchrom input : 染色体的长度

% individuals.chrom input : 染色体群

% sizepop input : 种群规模

% ret output : 交叉后的染色体

for i=1:sizepop %每一轮for循环中，可能会进行一次交叉操作，染色体是随机选择的，交叉位置也是随机选择的，

%但该轮for循环中是否进行交叉操作则由交叉概率决定（continue控制）

pick=rand(1,2); % 随机选择两个染色体进行交叉

while prod(pick)==0 %连乘

pick=rand(1,2);

end

index=ceil(pick.\*sizepop); % 交叉概率决定是否进行交叉

pick=rand;

while pick==0

pick=rand;

end

if pick>pcross

continue;

end

% 随机选择交叉位

pick=rand;

while pick==0

pick=rand;

end

flag=0;

while flag==0

pos=ceil(pick\*length(lenchrom)); %随机选择进行交叉的位置，即选择第几个变量进行交叉，注意：两个染色体交叉的位置相同

pick=rand; %交叉开始

v1=individuals.chrom(index(1),pos);

v2=individuals.chrom(index(2),pos);

individuals.chrom(index(1),pos)=pick\*v2+(1-pick)\*v1;

individuals.chrom(index(2),pos)=pick\*v1+(1-pick)\*v2; %交叉结束

flag1=test(individuals.chrom(index(1),:)); %检验染色体1的可行性

flag2=test(individuals.chrom(index(2),:)); %检验染色体2的可行性

if flag1\*flag2==0

flag=0;

else flag=1;

end %如果两个染色体不是都可行，则重新交叉

end

end

ret=individuals.chrom;

### 4.Mutation.m文件

function ret=Mutation(pmutation,lenchrom,individuals,sizepop,num,maxgen,bound)

% 本函数完成变异操作

% pcorss input : 变异概率

% lenchrom input : 染色体长度

% individuals.chrom input : 染色体

% sizepop input : 种群规模

% opts input : 变异方法的选择

% pop input : 当前种群的进化代数和最大的进化代数信息

% bound input : 每个个体的上届和下届

% maxgen input ：最大迭代次数

% num input : 当前迭代次数

% ret output : 变异后的染色体

for i=1:sizepop %每一轮for循环中，可能会进行一次变异操作，染色体是随机选择的，变异位置也是随机选择的，

%但该轮for循环中是否进行变异操作则由变异概率决定（continue控制）

% 随机选择一个染色体进行变异

pick=rand;

while pick==0

pick=rand;

end

index=ceil(pick\*sizepop);

% 变异概率决定该轮循环是否进行变异

pick=rand;

if pick>pmutation

continue;

end

flag=0;

while flag==0

% 变异位置

pick=rand;

while pick==0

pick=rand;

end

pos=ceil(pick\*sum(lenchrom)); %随机选择了染色体变异的位置，即选择了第pos个变量进行变异

pick=rand; %变异开始

fg=(pick\*(1-num/maxgen))^2;

if pick>0.5

individuals.chrom(index,pos)=individuals.chrom(index,pos)+(bound(pos,2)-individuals.chrom(index,pos))\*fg;

else

individuals.chrom(index,pos)=individuals.chrom(index,pos)-(individuals.chrom(index,pos)-bound(pos,1))\*fg;

end %变异结束

flag=test(individuals.chrom(index,:)); %检验染色体的可行性

end

end

ret=individuals.chrom;

### 5.test文件

function flag=test(chrom)

%此函数用来判断individuals.chrom里数值是否超过边界bound

%bound在main里定义为（-3:3）

%flag output 染色体可行（未超界）output为1 ，不可行为0

f1=isempty(find(chrom>3));

f2=isempty(find(chrom<-3));

if f1\*f2==0

flag=0;

else

flag=1;

end

### 6.Code.m文件

function ret=Code(lenchrom,bound)

%本函数将变量编码成染色体，用于随机初始化一个种群

% lenchrom input : 染色体长度

% bound input : 变量的取值范围

% ret output: 染色体的编码值

pick=rand(1,length(lenchrom));

ret=bound(:,1)'+(bound(:,2)-bound(:,1))'.\*pick; %线性插值，编码结果以实数向量存入ret中