遗传算法程序（一）:

说明: fga.m 为遗传算法的主程序; 采用二进制Gray编码,采用基于轮盘赌法的非线性排名选择, 均匀交叉,变异操作,而且还引入了倒位操作!

function [BestPop,Trace]=fga(FUN,LB,UB,eranum,popsize,pCross,pMutation,pInversion,options)

% [BestPop,Trace]=fmaxga(FUN,LB,UB,eranum,popsize,pcross,pmutation)

% Finds a maximum of a function of several variables.

% fmaxga solves problems of the form:

% max F(X) subject to: LB <= X <= UB

% BestPop - 最优的群体即为最优的染色体群

% Trace - 最佳染色体所对应的目标函数值

% FUN - 目标函数

% LB - 自变量下限

% UB - 自变量上限

% eranum - 种群的代数,取100--1000(默认200)

% popsize - 每一代种群的规模；此可取50--200(默认100)

% pcross - 交叉概率,一般取0.5--0.85之间较好(默认0.8)

% pmutation - 初始变异概率,一般取0.05-0.2之间较好(默认0.1)

% pInversion - 倒位概率,一般取0.05－0.3之间较好(默认0.2)

% options - 1\*2矩阵,options(1)=0二进制编码(默认0),option(1)~=0十进制编

%码,option(2)设定求解精度(默认1e-4)

%

% ------------------------------------------------------------------------

T1=clock;

if nargin<3, error('FMAXGA requires at least three input arguments'); end

if nargin==3, eranum=200;popsize=100;pCross=0.8;pMutation=0.1;pInversion=0.15;options=[0 1e-4];end

if nargin==4, popsize=100;pCross=0.8;pMutation=0.1;pInversion=0.15;options=[0 1e-4];end

if nargin==5, pCross=0.8;pMutation=0.1;pInversion=0.15;options=[0 1e-4];end

if nargin==6, pMutation=0.1;pInversion=0.15;options=[0 1e-4];end

if nargin==7, pInversion=0.15;options=[0 1e-4];end

if find((LB-UB)>0)

error('数据输入错误,请重新输入(LB<UB):');

end

s=sprintf('程序运行需要约%.4f 秒钟时间,请稍等......',(eranum\*popsize/1000));

disp(s);

global m n NewPop children1 children2 VarNum

bounds=[LB;UB]';bits=[];VarNum=size(bounds,1);

precision=options(2);%由求解精度确定二进制编码长度

bits=ceil(log2((bounds(:,2)-bounds(:,1))' ./ precision));%由设定精度划分区间

[Pop]=InitPopGray(popsize,bits);%初始化种群

[m,n]=size(Pop);

NewPop=zeros(m,n);

children1=zeros(1,n);

children2=zeros(1,n);

pm0=pMutation;

BestPop=zeros(eranum,n);%分配初始解空间BestPop,Trace

Trace=zeros(eranum,length(bits)+1);

i=1;

while i<=eranum

for j=1:m

value(j)=feval(FUN(1,:),(b2f(Pop(j,:),bounds,bits)));%计算适应度

end

[MaxValue,Index]=max(value);

BestPop(i,:)=Pop(Index,:);

Trace(i,1)=MaxValue;

Trace(i,(2:length(bits)+1))=b2f(BestPop(i,:),bounds,bits);

[selectpop]=NonlinearRankSelect(FUN,Pop,bounds,bits);%非线性排名选择

[CrossOverPop]=CrossOver(selectpop,pCross,round(unidrnd(eranum-i)/eranum));

%采用多点交叉和均匀交叉，且逐步增大均匀交叉的概率

%round(unidrnd(eranum-i)/eranum)

[MutationPop]=Mutation(CrossOverPop,pMutation,VarNum);%变异

[InversionPop]=Inversion(MutationPop,pInversion);%倒位

Pop=InversionPop;%更新

pMutation=pm0+(i^4)\*(pCross/3-pm0)/(eranum^4);

%随着种群向前进化，逐步增大变异率至1/2交叉率

p(i)=pMutation;

i=i+1;

end

t=1:eranum;

plot(t,Trace(:,1)');

title('函数优化的遗传算法');xlabel('进化世代数(eranum)');ylabel('每一代最优适应度(maxfitness)');

[MaxFval,I]=max(Trace(:,1));

X=Trace(I,(2:length(bits)+1));

hold on; plot(I,MaxFval,'\*');

text(I+5,MaxFval,['FMAX=' num2str(MaxFval)]);

str1=sprintf ('进化到 %d 代 ,自变量为 %s 时,得本次求解的最优值 %f\n对应染色体是：%s',I,num2str(X),MaxFval,num2str(BestPop(I,:)));

disp(str1);

%figure(2);plot(t,p);%绘制变异值增大过程

T2=clock;

elapsed\_time=T2-T1;

if elapsed\_time(6)<0

elapsed\_time(6)=elapsed\_time(6)+60; elapsed\_time(5)=elapsed\_time(5)-1;

end

if elapsed\_time(5)<0

elapsed\_time(5)=elapsed\_time(5)+60;elapsed\_time(4)=elapsed\_time(4)-1;

end %像这种程序当然不考虑运行上小时啦

str2=sprintf('程序运行耗时 %d 小时 %d 分钟 %.4f 秒',elapsed\_time(4),elapsed\_time(5),elapsed\_time(6));

disp(str2);

%初始化种群

%采用二进制Gray编码,其目的是为了克服二进制编码的Hamming悬崖缺点

function [initpop]=InitPopGray(popsize,bits)

len=sum(bits);

initpop=zeros(popsize,len);%The whole zero encoding individual

for i=2:popsize-1

pop=round(rand(1,len));

pop=mod(([0 pop]+[pop 0]),2);

%i=1时,b(1)=a(1);i>1时,b(i)=mod(a(i-1)+a(i),2)

%其中原二进制串:a(1)a(2)...a(n),Gray串:b(1)b(2)...b(n)

initpop(i,:)=pop(1:end-1);

end

initpop(popsize,:)=ones(1,len);%The whole one encoding individual

%解码

function [fval] = b2f(bval,bounds,bits)

% fval - 表征各变量的十进制数

% bval - 表征各变量的二进制编码串

% bounds - 各变量的取值范围

% bits - 各变量的二进制编码长度

scale=(bounds(:,2)-bounds(:,1))'./(2.^bits-1); %The range of the variables

numV=size(bounds,1);

cs=[0 cumsum(bits)];

for i=1:numV

a=bval((cs(i)+1):cs(i+1));

fval(i)=sum(2.^(size(a,2)-1:-1:0).\*a)\*scale(i)+bounds(i,1);

end

%选择操作

%采用基于轮盘赌法的非线性排名选择

%各个体成员按适应值从大到小分配选择概率：

%P(i)=(q/1-(1-q)^n)\*(1-q)^i, 其中 P(0)>P(1)>...>P(n), sum(P(i))=1

function [selectpop]=NonlinearRankSelect(FUN,pop,bounds,bits)

global m n

selectpop=zeros(m,n);

fit=zeros(m,1);

for i=1:m

fit(i)=feval(FUN(1,:),(b2f(pop(i,:),bounds,bits)));%以函数值为适应值做排名依据

end

selectprob=fit/sum(fit);%计算各个体相对适应度(0,1)

q=max(selectprob);%选择最优的概率

x=zeros(m,2);

x(:,1)=[m:-1:1]';

[y x(:,2)]=sort(selectprob);

r=q/(1-(1-q)^m);%标准分布基值

newfit(x(:,2))=r\*(1-q).^(x(:,1)-1);%生成选择概率

newfit=cumsum(newfit);%计算各选择概率之和

rNums=sort(rand(m,1));

fitIn=1;newIn=1;

while newIn<=m

if rNums(newIn)<newfit(fitIn)

selectpop(newIn,:)=pop(fitIn,:);

newIn=newIn+1;

else

fitIn=fitIn+1;

end

end

%交叉操作

function [NewPop]=CrossOver(OldPop,pCross,opts)

%OldPop为父代种群，pcross为交叉概率

global m n NewPop

r=rand(1,m);

y1=find(r<pCross);

y2=find(r>=pCross);

len=length(y1);

if len>2&mod(len,2)==1%如果用来进行交叉的染色体的条数为奇数，将其调整为偶数

y2(length(y2)+1)=y1(len);

y1(len)=[];

end

if length(y1)>=2

for i=0:2:length(y1)-2

if opts==0

[NewPop(y1(i+1),:),NewPop(y1(i+2),:)]=EqualCrossOver(OldPop(y1(i+1),:),OldPop(y1(i+2),:));

else

[NewPop(y1(i+1),:),NewPop(y1(i+2),:)]=MultiPointCross(OldPop(y1(i+1),:),OldPop(y1(i+2),:));

end

end

end

NewPop(y2,:)=OldPop(y2,:);

%采用均匀交叉

function [children1,children2]=EqualCrossOver(parent1,parent2)

global n children1 children2

hidecode=round(rand(1,n));%随机生成掩码

crossposition=find(hidecode==1);

holdposition=find(hidecode==0);

children1(crossposition)=parent1(crossposition);%掩码为1，父1为子1提供基因

children1(holdposition)=parent2(holdposition);%掩码为0，父2为子1提供基因

children2(crossposition)=parent2(crossposition);%掩码为1，父2为子2提供基因

children2(holdposition)=parent1(holdposition);%掩码为0，父1为子2提供基因

%采用多点交叉，交叉点数由变量数决定

function [Children1,Children2]=MultiPointCross(Parent1,Parent2)

global n Children1 Children2 VarNum

Children1=Parent1;

Children2=Parent2;

Points=sort(unidrnd(n,1,2\*VarNum));

for i=1:VarNum

Children1(Points(2\*i-1):Points(2\*i))=Parent2(Points(2\*i-1):Points(2\*i));

Children2(Points(2\*i-1):Points(2\*i))=Parent1(Points(2\*i-1):Points(2\*i));

end

%变异操作

function [NewPop]=Mutation(OldPop,pMutation,VarNum)

global m n NewPop

r=rand(1,m);

position=find(r<=pMutation);

len=length(position);

if len>=1

for i=1:len

k=unidrnd(n,1,VarNum); %设置变异点数，一般设置1点

for j=1:length(k)

if OldPop(position(i),k(j))==1

OldPop(position(i),k(j))=0;

else

OldPop(position(i),k(j))=1;

end

end

end

end

NewPop=OldPop;

%倒位操作

function [NewPop]=Inversion(OldPop,pInversion)

global m n NewPop

NewPop=OldPop;

r=rand(1,m);

PopIn=find(r<=pInversion);

len=length(PopIn);

if len>=1

for i=1:len

d=sort(unidrnd(n,1,2));

if d(1)~=1&d(2)~=n

NewPop(PopIn(i),1:d(1)-1)=OldPop(PopIn(i),1:d(1)-1);

NewPop(PopIn(i),d(1):d(2))=OldPop(PopIn(i),d(2):-1:d(1));

NewPop(PopIn(i),d(2)+1:n)=OldPop(PopIn(i),d(2)+1:n);

end

end

end

遗传算法程序（二）:

function youhuafun

D=code;

N=50; % Tunable

maxgen=50; % Tunable

crossrate=0.5; %Tunable

muterate=0.08; %Tunable

generation=1;

num = length(D);

fatherrand=randint(num,N,3);

score = zeros(maxgen,N);

while generation<=maxgen

ind=randperm(N-2)+2; % 随机配对交叉

A=fatherrand(:,ind(1:(N-2)/2));

B=fatherrand(:,ind((N-2)/2+1:end));

% 多点交叉

rnd=rand(num,(N-2)/2);

ind=rnd tmp=A(ind);

A(ind)=B(ind);

B(ind)=tmp;

% % 两点交叉

% for kk=1:(N-2)/2

% rndtmp=randint(1,1,num)+1;

% tmp=A(1:rndtmp,kk);

% A(1:rndtmp,kk)=B(1:rndtmp,kk);

% B(1:rndtmp,kk)=tmp;

% end

fatherrand=[fatherrand(:,1:2),A,B];

% 变异

rnd=rand(num,N);

ind=rnd [m,n]=size(ind);

tmp=randint(m,n,2)+1;

tmp(:,1:2)=0;

fatherrand=tmp+fatherrand;

fatherrand=mod(fatherrand,3);

% fatherrand(ind)=tmp;

%评价、选择

scoreN=scorefun(fatherrand,D);% 求得N个个体的评价函数

score(generation,:)=scoreN;

[scoreSort,scoreind]=sort(scoreN);

sumscore=cumsum(scoreSort);

sumscore=sumscore./sumscore(end);

childind(1:2)=scoreind(end-1:end);

for k=3:N

tmprnd=rand;

tmpind=tmprnd difind=[0,diff(tmpind)];

if ~any(difind)

difind(1)=1;

end

childind(k)=scoreind(logical(difind));

end

fatherrand=fatherrand(:,childind);

generation=generation+1;

end

% score

maxV=max(score,[],2);

minV=11\*300-maxV;

plot(minV,'\*');title('各代的目标函数值');

F4=D(:,4);

FF4=F4-fatherrand(:,1);

FF4=max(FF4,1);

D(:,5)=FF4;

save DData D

function D=code

load youhua.mat

% properties F2 and F3

F1=A(:,1);

F2=A(:,2);

F3=A(:,3);

if (max(F2)>1450)||(min(F2)<=900)

error('DATA property F2 exceed it''s range (900,1450]')

end

% get group property F1 of data, according to F2 value

F4=zeros(size(F1));

for ite=11:-1:1

index=find(F2<=900+ite\*50);

F4(index)=ite;

end

D=[F1,F2,F3,F4];

function ScoreN=scorefun(fatherrand,D)

F3=D(:,3);

F4=D(:,4);

N=size(fatherrand,2);

FF4=F4\*ones(1,N);

FF4rnd=FF4-fatherrand;

FF4rnd=max(FF4rnd,1);

ScoreN=ones(1,N)\*300\*11;

% 这里有待优化

for k=1:N

FF4k=FF4rnd(:,k);

for ite=1:11

F0index=find(FF4k==ite);

if ~isempty(F0index)

tmpMat=F3(F0index);

tmpSco=sum(tmpMat);

ScoreBin(ite)=mod(tmpSco,300);

end

end

Scorek(k)=sum(ScoreBin);

end

ScoreN=ScoreN-Scorek;

遗传算法程序（三）：

%IAGA

function best=ga

clear

MAX\_gen=200; %最大迭代步数

best.max\_f=0; %当前最大的适应度

STOP\_f=14.5; %停止循环的适应度

RANGE=[0 255]; %初始取值范围[0 255]

SPEEDUP\_INTER=5; %进入加速迭代的间隔

advance\_k=0; %优化的次数

popus=init; %初始化

for gen=1:MAX\_gen

fitness=fit(popus,RANGE); %求适应度

f=fitness.f;

picked=choose(popus,fitness); %选择

popus=intercross(popus,picked); %杂交

popus=aberrance(popus,picked); %变异

if max(f)>best.max\_f

advance\_k=advance\_k+1;

x\_better(advance\_k)=fitness.x;

best.max\_f=max(f);

best.popus=popus;

best.x=fitness.x;

end

if mod(advance\_k,SPEEDUP\_INTER)==0

RANGE=minmax(x\_better);

RANGE

advance=0;

end

end

return;

function popus=init%初始化

M=50;%种群个体数目

N=30;%编码长度

popus=round(rand(M,N));

return;

function fitness=fit(popus,RANGE)%求适应度

[M,N]=size(popus);

fitness=zeros(M,1);%适应度

f=zeros(M,1);%函数值

A=RANGE(1);B=RANGE(2);%初始取值范围[0 255]

for m=1:M

x=0;

for n=1:N

x=x+popus(m,n)\*(2^(n-1));

end

x=x\*((B-A)/(2^N))+A;

for k=1:5

f(m,1)=f(m,1)-(k\*sin((k+1)\*x+k));

end

end

f\_std=(f-min(f))./(max(f)-min(f));%函数值标准化

fitness.f=f;fitness.f\_std=f\_std;fitness.x=x;

return;

function picked=choose(popus,fitness)%选择

f=fitness.f;f\_std=fitness.f\_std;

[M,N]=size(popus);

choose\_N=3; %选择choose\_N对双亲

picked=zeros(choose\_N,2); %记录选择好的双亲

p=zeros(M,1); %选择概率

d\_order=zeros(M,1);

%把父代个体按适应度从大到小排序

f\_t=sort(f,'descend');%将适应度按降序排列

for k=1:M

x=find(f==f\_t(k));%降序排列的个体序号

d\_order(k)=x(1);

end

for m=1:M

popus\_t(m,:)=popus(d\_order(m),:);

end

popus=popus\_t;

f=f\_t;

p=f\_std./sum(f\_std); %选择概率

c\_p=cumsum(p)'; %累积概率

for cn=1:choose\_N

picked(cn,1)=roulette(c\_p); %轮盘赌

picked(cn,2)=roulette(c\_p); %轮盘赌

popus=intercross(popus,picked(cn,:));%杂交

end

popus=aberrance(popus,picked);%变异

return;

function popus=intercross(popus,picked) %杂交

[M\_p,N\_p]=size(picked);

[M,N]=size(popus);

for cn=1:M\_p

p(1)=ceil(rand\*N);%生成杂交位置

p(2)=ceil(rand\*N);

p=sort(p);

t=popus(picked(cn,1),p(1):p(2));

popus(picked(cn,1),p(1):p(2))=popus(picked(cn,2),p(1):p(2));

popus(picked(cn,2),p(1):p(2))=t;

end

return;

function popus=aberrance(popus,picked) %变异

P\_a=0.05;%变异概率

[M,N]=size(popus);

[M\_p,N\_p]=size(picked);

U=rand(1,2);

for kp=1:M\_p

if U(2)>=P\_a %如果大于变异概率，就不变异

continue;

end

if U(1)>=0.5

a=picked(kp,1);

else

a=picked(kp,2);

end

p(1)=ceil(rand\*N);%生成变异位置

p(2)=ceil(rand\*N);

if popus(a,p(1))==1%0 1变换

popus(a,p(1))=0;

else

popus(a,p(1))=1;

end

if popus(a,p(2))==1

popus(a,p(2))=0;

else

popus(a,p(2))=1;

end

end

return;

function picked=roulette(c\_p) %轮盘赌

[M,N]=size(c\_p);

M=max([M N]);

U=rand;

if U<c\_p(1)

picked=1;

return;

end

for m=1:(M-1)

if U>c\_p(m) & U<c\_p(m+1)

picked=m+1;

break;

end

end

全方位的两点杂交、两点变异的改进的加速遗传算法（IAGA）

**遗传算法优化pid参数matlab程序**

chap5\_4m

%GA(Generic Algorithm) program to optimize Parameters of PID

clear all;

clear all;

global rin yout timef

G=100;

Size=30;

CodeL=10;

MinX(1)=zeros(1);

MaxX(1)=20\*ones(1);

MinX(2)=zeros(1);

MaxX(2)=1.0\*ones(1);

MinX(3)=zeros(1);

MaxX(3)=1.0\*ones(1);

E=round(rand(Size,3\*CodeL));%Initian Code!

BsJ=0;

for kg=1:1:G

time(kg)=kg;

for s=1:1:Size

m=E(s,:);

y1=0;y2=0;y3=0;

m1=m(1:1:CodeL);

for i=1:1:CodeL

y1=y1+m1(i)\*2^(i-1);

end

Kpid(s,1)=(MaxX(1)-MinX(1))\*y1/1023+MinX(1);

m2=m(CodeL+1:1:2\*CodeL);

for i=1:1:CodeL

y2=y2+m2(i)\*2^(i-1);

end

Kpid(s,2)=(MaxX(2)-MinX(2))\*y2/1023+MinX(2);

m3=m(2\*CodeL+1:1:3\*CodeL);

for i=1:1:CodeL

y3=y3+m3(i)\*2^(i-1);

end

Kpid(s,3)=(MaxX(3)-MinX(3))\*y3/1023+MinX(3);

%\*\*\*\*\*\*\*Step 1:Evaluate Best J\*\*\*\*\*\*\*

Kpidi=Kpid(s,:);

[Kpidi,BsJ]=chap5\_3f(Kpidi,BsJ);

BsJi(s)=BsJ;

end

[OderJi,IndexJi]=sort(BsJi);

BestJ(kg)=OderJi(1);

BJ=BestJ(kg);

Ji=BsJi+1e-10;

fi=1./Ji;

%Cm=max(Ji);

%fi=Cm-Ji; %Avoiding deviding zero

[Oderfi,Indexfi]=sort(fi);%Arranging fi small to bigger

%Bestfi=Oderfi(Size); %Let Bestfi=max(fi)

%BestS=Kpid(Indexfi(Size),:); %Let BestS=E(m),m is the Indexfi belong to

%max(fi)

Bestfi=Oderfi(Size);%Let Bestfi=max(fi)

BestS=E(Indexfi(Size),:);%Let BestS=E(m),m is the Indexfi belong to max(fi)

kg

BJ

BestS;

%\*\*\*\*Step 2:Select and Reproduct Operation\*\*\*

fi\_sum=sum(fi);

fi\_Size=(Oderfi/fi\_sum)\*Size;

fi\_S=floor(fi\_Size); %Selecting Bigger fi value

kk=1;

for i=1:1:Size

for j=1:1:fi\_S(i) %Select and Reproduce

TempE(kk,:)=E(Indexfi(i),:);

kk=kk+1; %kk is used to reproduce

end

end

%\*\*\*\*\*\*\*\*\*\*Step 3:Crossover Operation\*\*\*\*\*\*

pc=0.06;

n=ceil(20\*rand);

for i=1:2:(Size-1)

temp=rand;

if pc>temp

for j=n:1:20

TempE(i,j)=E(i+1,j);

TempE(i+1,j)=E(i,j);

end

end

end

TempE(Size,:)=BestS;

E=TempE;

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Step 4: Mutation Operation\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%pm=0.001;

pm=0.001-[1:1:Size]\*(0.001)/Size;%Bigger fi,smaller pm

%pm=0.0; %No mutation

%pm=0.1; %Big mutation

for i=1:1:Size

for j=1:1:3\*CodeL

temp=rand;

if pm>temp %Mutation Condition

if TempE(i,j)==0

TempE(i,j)=1;

else

TempE(i,j)=0;

end

end

end

end

%Guarantee TempE(Size,:)belong to the best individual

TempE(Size,:)=BestS;

E=TempE;

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

end

Bestfi

BestS

Kpidi

Best\_J=BestJ(G)

figure(1);

plot(time,BestJ);

xlabel('Time');ylabel('Best J');

figure(2);

plot(timef,rin,'r',timef,yout,'b');

xlabel('Time(s)');ylabel('ran,yout');

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

chap5\_3f.m

function [Kpidi,BsJ]=pid\_gaf(Kpidi,BsJ)

global rin yout timef

ts=0.001;

sys=tf(400,[1,50,0]);

dsys=c2d(sys,ts,'z');

[num,den]=tfdata(dsys,'v');

rin=1.0;

u\_1=0.0;u\_2=0.0;

y\_1=0.0;y\_2=0.0;

x=[0,0,0]';

B=0;

error\_1=0;

tu=1;

s=0;

P=100;

for k=1:1:P

timef(k)=k\*ts;

r(k)=rin;

u(k)=Kpidi(1)\*x(1)+Kpidi(2)\*x(2)+Kpidi(3)\*x(3);

if u(k)>=10

u(k)=10;

end

if u(k)<=-10

u(k)=-10;

end

yout(k)=-den(2)\*y\_1-den(3)\*y\_2+num(2)\*u\_1+num(3)\*u\_2;

error(k)=r(k)-yout(k);

%-------------------Return of PID parameters----------------

u\_2=u\_1;u\_1=u(k);

y\_2=y\_1;y\_1=yout(k);

x(1)=error(k); % Calculating P

x(2)=(error(k)-error\_1)/ts; % D

x(3)=x(3)+error(k)\*ts; % I

error\_2=error\_1;

error\_1=error(k);

if s==0

if yout(k)>0.95&yout(k)<1.05

tu=timef(k);

s=1;

end

end

end

for i=1:1:P

Ji(i)=0.999\*abs(error(i))+0.01\*u(i)^2\*0.1;

B=B+Ji(i);

if i>1

erry(i)=yout(i)-yout(i-1);

if erry(i)<0

B=B+100\*abs(erry(i));

end

end

end

Bsj=B+0.2\*tu\*10