智能与优化算法大作业

张锦程 材84 2018012082

考核作业—I

利用至少一种智能算法求解多极小函数

- 采用 SA、GA、TS、PSO、DE 或混合算法等
- 函数可自我选择
- 给出 20 次随机实验的统计结果（平均性能、最佳性能、最差性能、方差等）
- 给出典型的某次仿真过程中目标函数的变化曲线
- 讨论所用算法在函数优化中的特点
- 阐述实验体会

优化函数

$$f(x) = 11sin(7x) - 7cos(3x) + 5sin(6x),\ x \in [0, 11]$$

随机实验的统计结果

1. 遗传算法

次	1	2	3	4	5	6	7	8	9	10
值	-16.160	-19.954	-19.760	-13.310	-19.954	-21.358	-19.760	-16.999	-18.354	-16.594
次	11	12	13	14	15	16	17	18	19	20
值	-21.358	-21.371	-11.830	-15.651	-20.703	-21.311	-20.205	-18.667	-14.596	-19.760

平均性能

-18.38266

最佳性能

-21.3706

最差性能

-11.8296

方差

-8.16579611

2. 粒子群算法

试验次数	1	2	3	4	5
最小值	-21.37444247	-21.37443954	-21.37444247	-21.37443822	-21.37444193
试验次数	6	7	8	9	10
最小值	-21.37437896	-21.37444247	-21.37443875	-21.37444202	-21.37444226
试验次数	11	12	13	14	15
最小值	-21.37444239	-21.37443665	-21.3744423	-21.37440165	-21.37440886
试验次数	16	17	18	19	20
最小值	-21.37443714	-21.37443868	-21.37444247	-21.37444155	-21.37444247

平均性能

-21.37443416

最佳性能

-21.37444247

最差性能

-21.37437896

方差

2.93065×10^{-10}

3. 基于遗传算法的粒子群混合算法

试验次数	1	2	3	4	5
最小值	-21.37440351	-21.37443688	-21.37425442	-21.37442998	-21.37443776
试验次数	6	7	8	9	10
最小值	-21.37443063	-21.37441113	-21.37443923	-21.37443746	-21.37441958
试验次数	11	12	13	14	15
最小值	-21.37437858	-21.37430816	-21.37439186	-21.37443441	-21.37443366

试验次数	1	2	3	4	5
试验次数	16	17	18	19	20
最小值	-21.37443979	-21.37441198	-21.37418368	-21.37442474	-21.37444179

平均性能

$$-21.37439746$$

最佳性能

$$-21.37444179$$

最差性能

$$-21.37418368$$

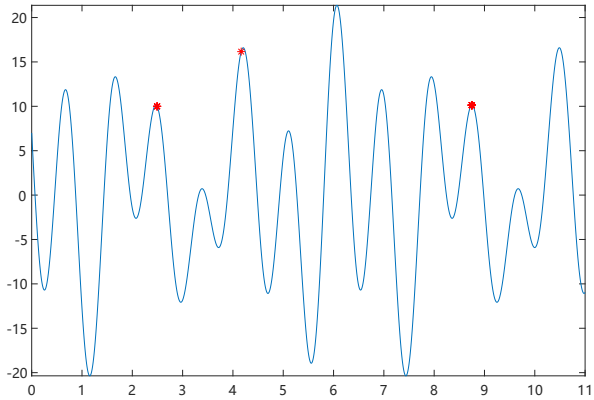
方差

$$-4.80573 \times 10^{-9}$$

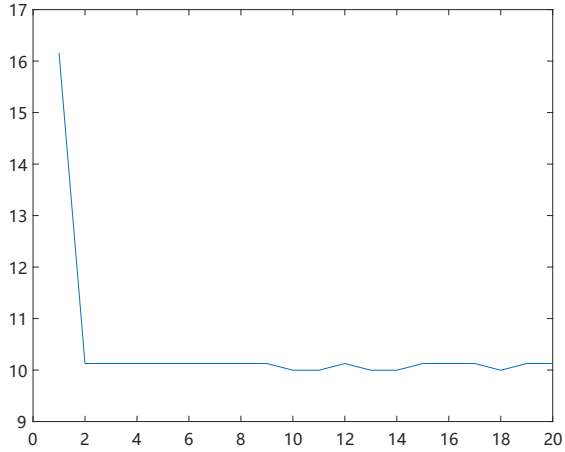
目标函数变化曲线

1. 遗传算法

某次典型的优化结果*



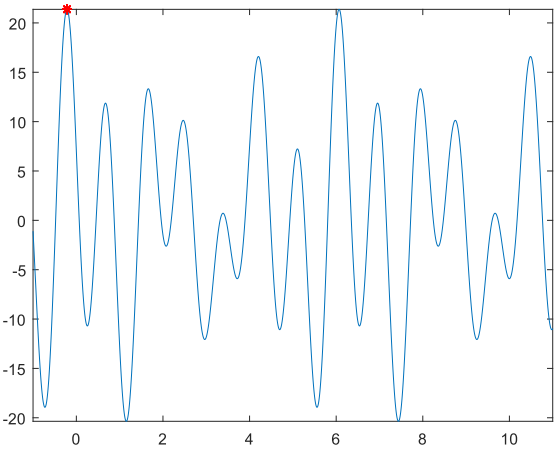
某次典型的最优值变化曲线



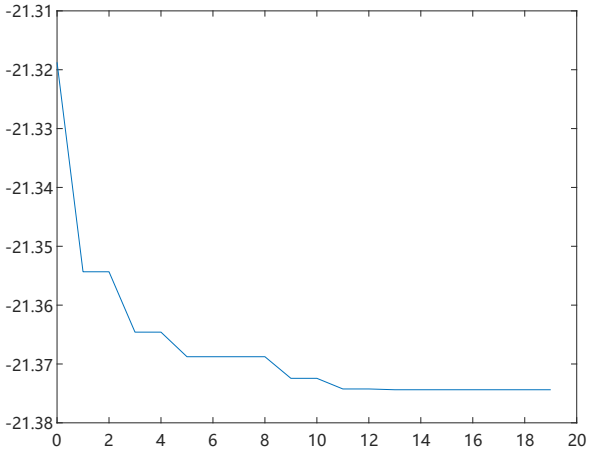
*注：遗传算法中选择适应值最大的个体，这里的适应值可以是原优化函数的函数值，所以在程序求解的过程中首先将函数取反，从而将求极小问题转化为求极大问题；为方便起见，PSO算法中也同理。

2. 粒子群算法

某次典型的优化结果

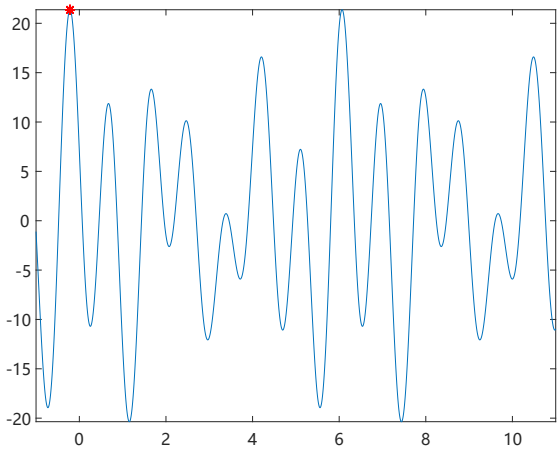


某次典型的最优值变化曲线

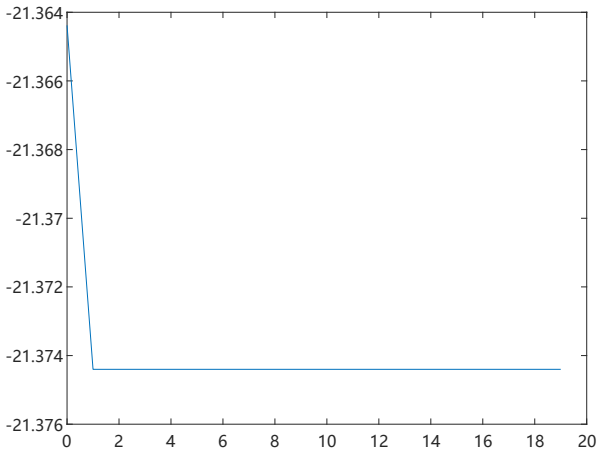


3. 基于遗传算法的粒子群混合算法

某次典型的优化结果



某次典型的最优值变化曲线



####

算法在函数优化中的特点

GA 算法

算法具有随机性，收敛很快，甚至有一定的早熟性，仅仅靠单次实验很难得到全局最优解，需要多次反复实验，算法有很小的概率偏离全局极值点很大，造成单次实验的较大误差。

PSO 算法

算法收敛较快，一般在 10 次迭代内均能收敛，最大的优点是能稳定地收敛到全局的极值点，爬坡能力很强，难以局限在局部的极值点，单次的实验中几乎不会出现偏离全局极值点很大的情况，方差在 2.93065×10^{-10} 量级。个人推断这是因为 PSO 算法显含的并行搜索特性。

GA-PSO 混合算法

从实验结果来看，混合算法继承了 PSO 算法的各项优点，如稳定收敛，爬坡力强，收敛快等等，由于该项优化比较简单，属于连续可微的目标函数，所以暂时还无法看出混合算法相对于 PSO 有了哪些改进。

实验体会

不同的算法有其各自的特点，在实际问题的求解过程中可以注意将不同的算法组合使用以得到更好的结果；GA、PSO 等启发式算法的求解过程具有一定的随机性，每次的实验结果都会有一定的差别，一次的运行结果往往不能保证得到最优解，可以从不同的初始条件出发 (如本实验中 GA 算法的初始染色体为一随机 0-1 数组)，独立运行这些启发式算法，多次重复后往往能够得到最优解；实验过程中发现一些算法(如 GA)收敛较快，具有一定的早熟倾向，出现这种现象不利于超越局部极值点而到达全局极值点(也就是算法的爬坡能力)，这是本算法未来要尽量去克服的。

查阅相关资料可以发现：遗传算法适用于目标函数不连续、不可微、高度非线性，或者优化问题是困难的组合问题。

一个通用而又较少依赖于目标函数值与其他辅助信息的算法不可能比专用且充分利用目标函数值与相关辅助信息的算法更为有效，而当一个问题有某些辅助信息可供使用时，舍弃应用本来可供应用的信息而去应用于这些信息无关的算法也不是一个聪明的选择。所以，遗传算法一般来说并不适宜应用于通常的数值优化问题（例如连续可微的数学规划问题），或者说，当应用于这样的问题时，遗传算法并不总能显示其优越性。这部分解决了我的疑惑。

在实验过程中的另一大收获是一个算法的固有缺点可以通过引入另外一个算法从而构成混合算法来克服，这为我们改进现有的一些算法提供了强有力的思路。

考核作业—II

利用至少一种智能优化算法求解 TSP

- 采用 SA、GA、TS、PSO、DE、HNN 或混合算法等
- 规模可自我选择
- 给出 20 次随机实验的统计结果（平均性能、最佳性能、最差性能、方差等）
- 给出典型的某次仿真过程中目标函数的变化曲线
- 给出最优结果的图形
- 讨论所用算法在组合优化中的特点
- 阐述实验体会

优化 TSP 问题

城市	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
X	54	87	74	71	25	35	45	13	18	24
Y	18	76	78	71	38	56	3	40	40	42
城市	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20
X	71	64	68	83	58	54	51	37	41	2
Y	44	60	58	69	69	62	67	84	94	99
城市	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30
X	7	22	25	62	87	38	83	26	45	44
Y	64	60	62	32	7	91	46	41	21	35

1. PSO 算法

随机实验的统计结果

试验次数	1	2	3	4	5	6	7	8	9	10
最小值	613.8	615.1	583.1	690.2	627.9	627.2	590.7	564.5	647.9	633.7
时间/s	9.828	9.602	9.412	9.317	9.583	9.387	9.483	9.818	9.287	9.595
试验次数	11	12	13	14	15	16	17	18	19	20
最小值	548.4	565.5	553.2	539.2	633.2	678.1	627.7	633.8	608.8	579.1
时间/s	9.335	9.707	9.913	9.952	9.722	10.049	9.904	9.782	9.832	9.823

平均性能

TSP 最小距离：608.05739

程序用时：9.66655 s

最佳性能

539.2225

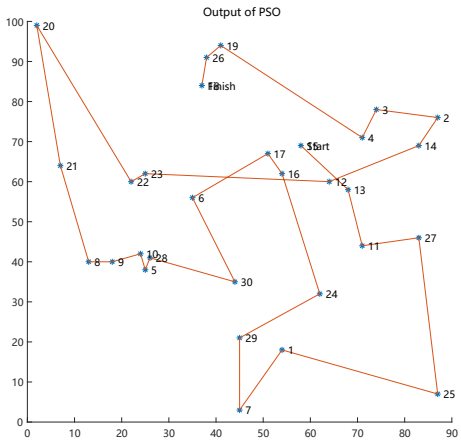
最差性能

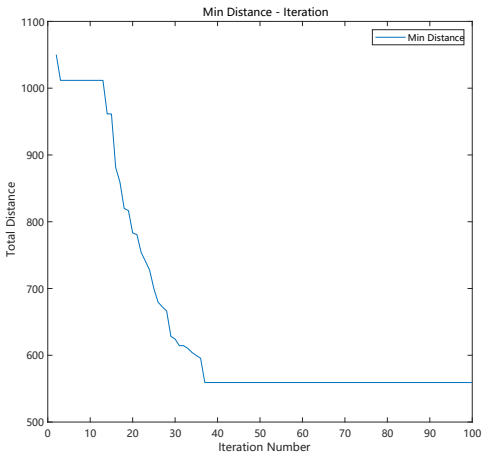
690.2447

方差

1748.239529

最优结果的图形&目标函数变化曲线

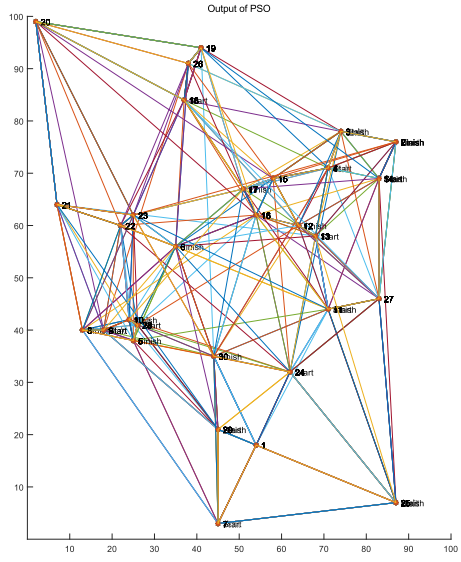


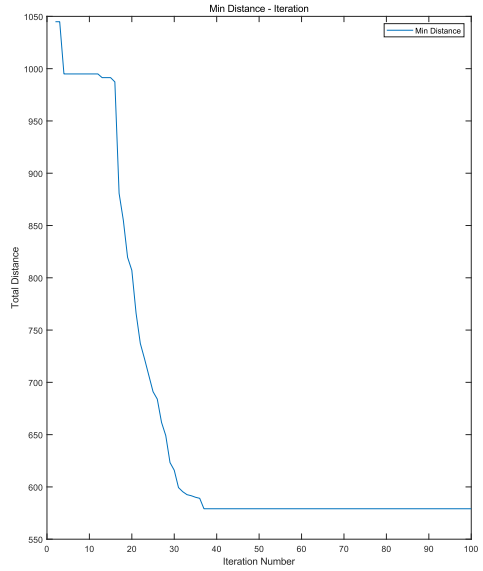


算法在函数优化中的特点

PSO 在 TSP 问题中保持了与多极小函数优化中一贯的稳定性，而且大多在 40 代之内收敛，具有计算上的节约性，结果中较差的解决方案很少，具有一定的可靠性。然而要想得到尽可能接近最优解的方案，还需要多次运行该算法。

将 20 次实验的所有路径绘制在下图：





可以看到 PSO 算法在每次的优化后都会给出多种多样的解决方案，这样的特性有利于充分探索问题的解空间，避免陷入局域的最小值点，这也是启发式算法在解 TSP 问题中的独到之处。

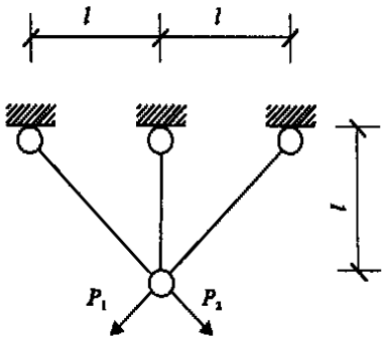
实验体会

将 PSO 算法应用到了一个更加接近于实际问题的领域 (TSP)，提升了自己解决实际问题的能力，培养了将实际问题量化，并转化为特定算法所能求解的形式的能力。

考核作业—III

将所学智能优化算法应用于实践 (感兴趣的某一实际问题、SRT等)

遗传算法在工程结构优化中得到大量的应用, 如桁架结构优化设计，桁架结构拓扑优化、抗震结构智能优化设计等。下面举一个应用的实例。



三杆超静定桁架，其几何尺寸如图所示。材料假定为低碳钢，其材料常数为：$[\sigma_+]=200MPa,\ [\sigma_-]=150MPa,\ \rho=7.85g/cm^3,\ l=1m$。结构受到的载荷有两种工况：

工况一：$P_1=20kN,\ P_2=0;$

工况二：$P_1=0,\ P_2=20kN;$

试进行优化设计。

应用工程力学中的知识：$A_1=A_3=X_1,\ A_2=X_2$，$\min$：$f(x)=2\sqrt{2}X_1+X_2$

$$s.t.:\qquad f(x)=\begin{cases}\dfrac{P_1(X_2+\sqrt{2}X_1)}{\sqrt{2}X_1^2+2X_1X_2}\leq[\sigma_+]\\ \dfrac{\sqrt{2}P_1X_1}{\sqrt{2}X_1^2+2X_1X_2}\leq[\sigma_+]\\ \dfrac{P_1X_2}{\sqrt{2}X_1^2+2X_1X_2}\leq[\sigma_-]\\ 0.1\leq X_1\leq 1\\ 0.1\leq X_2\leq 1\end{cases}$$

本例的遗传算法求解过程中保留八位有效数字，即采用十进制时每个变量的编码长度为八。按违反约束条件的程度成比例地降低个体的适度值。比如，当违反第一个约束时，适度值的降低采用下式：

$$fitness=fitness-\lambda^*abs(\frac{P_1(X_2+\sqrt{2}X_1)}{\sqrt{2}X_1^2+2X_1X_2})-[\sigma_+])/[\sigma_+]$$

λ^* 是常数，这里不妨取 1 。 程序的种群数为 80，最大进化代数为 100，初始种群在变量的范围内均匀随机生成。在进化的过程中，交叉概率 0.5, 变异概率为 0.01，无保优操作。程序运行会出现下列结果：

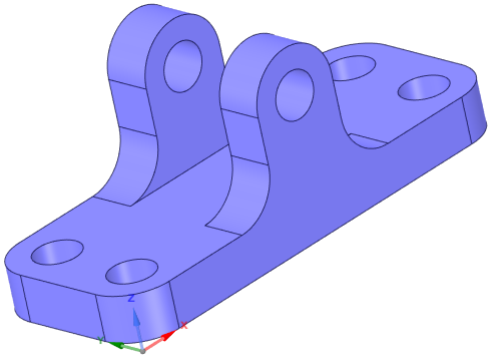
试验次数	1	2	3	4	5	6	7	8	9	10
最小值	2.695	2.643	2.686	2.686	2.730	2.713	2.740	2.714	2.664	2.677
时间/s	14.33	14.56	14.57	14.35	14.20	14.30	14.29	14.63	14.16	14.39
试验次数	11	12	13	14	15	16	17	18	19	20
最小值	2.678	2.682	2.643	2.643	2.640	2.726	2.723	2.650	2.723	2.729
时间/s	14.98	14.99	14.58	14.38	14.70	14.81	14.97	14.44	14.93	14.39

得到的最优解为：$X_1 = 0.8025,\ X_2 = 0.3997$

经验算，程序结果满足约束条件。可以看出，遗传算法是一种有效的工程优化方法。

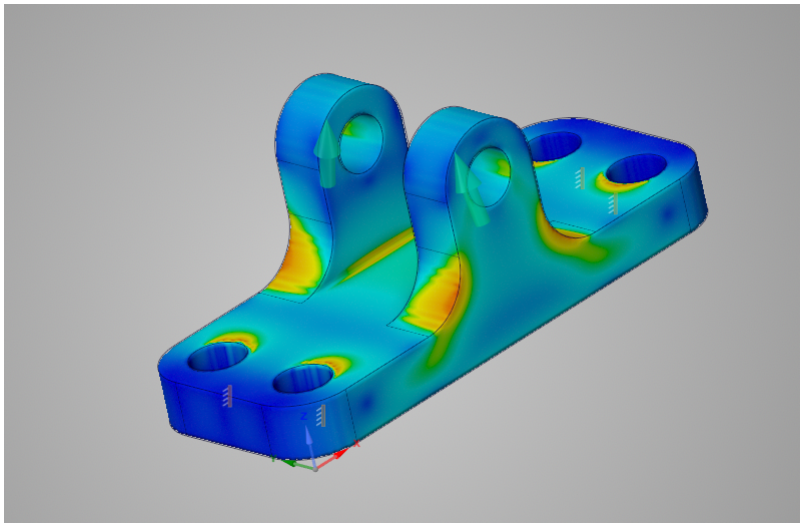
以上仅仅列举了 GA 在实际工程应用中的一个简单例子。在实际的应用中，复杂结构的受力难以通过简单的工程力学方程描述，而常常采用有限元计算的方法来进行数值计算。如使用 ANSYS 软件，它能与多数 CAD 软件接口，实现数据的共享和交换。在一些结构优化过程中，可以使用 ANSYS 自带的优化器，而该软件也提供了优化器的外部接口，这也为 GA 在基于有限元模拟的结构优化中的应用提供了便利。

下面是一个简单机械零件的拓扑优化过程：

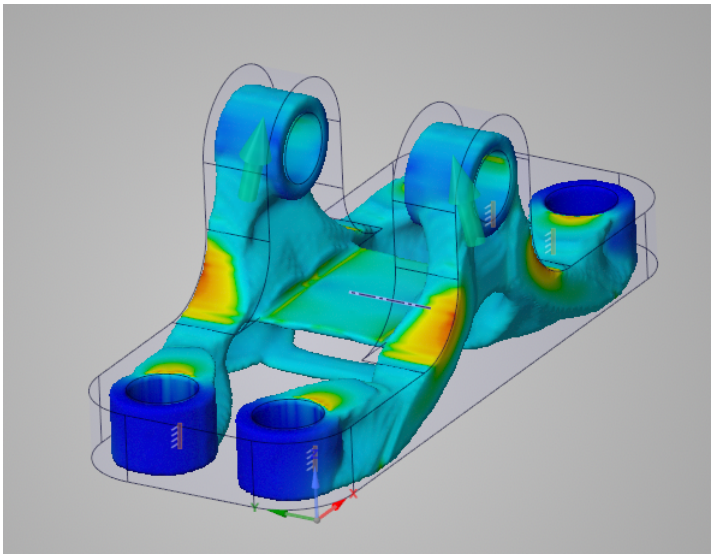


某工程固定铰支零件结构图

通过拓扑优化，可以在所用原材料大大减少的情况下保持原有结构的稳定性，从而达到工程或经济上的最优性。

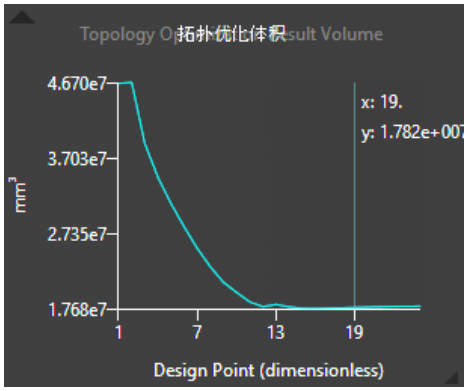


施加边界条件和工作应力后的应力状态有限元分析示意图



拓扑优化后的形变分析图

其中使用了 ANSYS 自带的 GA 算法。经过优化后的结构用料大大减少：



值得一提的是，借助于日渐成熟的 3D 打印技术，上述工程结构优化不再是纸上谈兵，而变成了具有实际意义的优化问题。

附录

1. 遗传算法代码

```
clear all
clc
popsize=40;              %群体大小
chromlength=10;          %字符串长度（个体长度）
pc=0.8;                  %交配概率
pm=0.006;                                    %变异概率
pop=initpop(popsize,chromlength);              %随机产生初始群体

% fitvalue即为函数值，倾向于选择fitvalue大的解
for i=1:20                                     %40为迭代次数
[objvalue]=calobjvalue(pop);                     %计算目标函数
fitvalue=calfitvalue(objvalue);                    %计算群体中每个个体的适应度
[newpop]=selection(pop,fitvalue);                  %复制
[newpop]=crossover(pop,pc);                      %交叉
[newpop]=mutation(pop,pc);                       %变异
[bestindividual,bestfit]=best(pop,fitvalue);       %求出群体中适应值最大的个体及
其适应值
y(i)=max(bestfit);
n(i)=i;
pop5=bestindividual;
x(i)=decodechrom(pop5,1,chromlength)*10/1023;
pop=newpop;
end

%给出计算结果
fm=max(y)
disp('**************************************************');
disp('目标函数取最小值时的自变量：');
xm
disp('目标函数的最小值为：')
fm
disp('**************************************************');

figure(1) % 绘制函数曲线及所拟合的最优解、最优值
fplot('-11*sin(7*x)+7*cos(3*x)-5*sin(6*x)',[0 11])
hold on
plot(x,y,'r*')
hold off
```

```
38
39  figure(2) % 绘制各世代的最优值
40  plot(n,y)
41
42  %初始化
43  function pop=initpop(popsize,chromlength)
44  pop=round(rand(popsize,chromlength));
45  end
46  %计算个体的适应值
47  function [objvalue]=calobjvalue(pop)
48  temp1=decodechrom(pop,1,10); %将pop每行转化成十进制数
49  x=temp1*10/1023; %将二值域 中的数转化为变量域 的数
50  objvalue=-11*sin(7*x)+7*cos(3*x)-5*sin(6*x); %计算目标函数值
51  end
52  function fitvalue=calfitvalue(objvalue)
53  global Cmin;
54  Cmin=0;
55  [px,py]=size(objvalue);
56  for i=1:px
57      if objvalue(i)+Cmin>0
58          temp=Cmin+objvalue(i);
59      else
60          temp=0.0;
61      end
62      fitvalue(i)=temp;
63  end
64  fitvalue=fitvalue'
65  %选择复制
66  function [newpop]=selection(pop,fitvalue)
67  totalfit=sum(fitvalue); %求适应值之和
68  fitvalue=fitvalue/totalfit; %单个个体被选择的概率
69  fitvalue=cumsum(fitvalue); %如 fitvalue=[1 2 3 4]，则 cumsum(fitvalue)=[1 3 6
    10]
70  [px,py]=size(pop);
71  ms=sort(rand(px,1)); %从小到大排列
72  fitin=1;
73  newin=1;
74  while newin<=px
75      if(ms(newin))<fitvalue(fitin)
76          newpop(newin)=pop(fitin);
77          newin=newin+1;
78      else
79          fitin=fitin+1;
```

```
    end
end
%交叉
function [newpop]=crossover(pop,pc)
[px,py]=size(pop);
newpop=ones(size(pop));
for i=1:2:px-1
    if(rand<pc)
        cpoint=round(rand*py);
        newpop(i,:)=[pop(i,1:cpoint),pop(i+1,cpoint+1:py)];
        newpop(i+1,:)=[pop(i+1,1:cpoint),pop(i,cpoint+1:py)];
    else
        newpop(i,:)=pop(i);
        newpop(i+1,:)=pop(i+1);
    end
end
%变异
function [newpop]=mutation(pop,pm)
[px,py]=size(pop);
newpop=ones(size(pop));
for i=1:px
    if(rand<pm)
        mpoint=round(rand*py);
        if mpoint<=0
            mpoint=1;
        end
        newpop(i)=pop(i);
        if any(newpop(i,mpoint))==0
            newpop(i,mpoint)=1;
        else
            newpop(i,mpoint)=0;
        end
    else
        newpop(i)=pop(i);
    end
end

```

2. 粒子群算法代码

```
clear all
clc
N = 40% N初始化群体个体数目
c1= 1.2 % c1学习因子1
c2= 2.2 % c2学习因子6
w = 0.6 % w惯性权重
M = 100 % M最大迭代次数
D = 1 % D搜索空间维数

x = unifrnd (0,10,1,1) %zeros(1,10);
[xm,fm,xn,yn]=PSO(@fitness,40,1.2,2.2,0.6,20,1);%fitness,N,c1,c2,w,M,D

%给出计算结果
disp('**************************************************');
disp('目标函数取最小值时的自变量：');
xm
disp('目标函数的最小值为：')
fm
disp('**************************************************');
n = 0:(size(yn,2)-1);
figure(1) % 绘制函数曲线及所拟合的最优解、最优值
fplot(@(x)-11.*sin(7.*x)+7.*cos(3.*x)-5.*sin(6.*x),[-1 11])
hold on
plot(xn,-yn,'r*')
hold off
figure(2) % 绘制各世代的最优值
plot(n,yn)

% 求函数的全局最小值
function[xm,fm,xn,yn]=PSO(fitness,N,c1,c2,w,M,D) %xn,yn为分步的值
%初始化种群的个体(可以在这里限定位置和速度的范围)
format long;
for i=1:N
    for j=1:D
        x(i,j)=randn; %随机初始化位置
        v(i,j)=randn; %随机初始化速度
    end
end
%计算各个粒子的适应度，并初始化Pi和Pg
for i=1:N
    p(i)=fitness(x(i,:));
```

```
    y(i,:)=x(i,:);
end
pg=x(N,:);              %Pg为全局最优
for i=1:(N-1)
    if fitness(x(i,:)) < fitness(pg)
        pg=x(i,:);
    end
end
%进入主要循环，按照公式依次迭代，直到满足精度要求
for t=1:M
    for i=1:N     %更新速度、位移
        v(i,:)=w*v(i,:)+c1*rand*(y(i,:)-x(i,:))+c2*rand*(pg-x(i,:));
        x(i,:)=x(i,:)+v(i,:);
        if fitness(x(i,:)) < p(i)
            p(i)=fitness(x(i,:));
            y(i,:)=x(i,:);
        end
        if p(i)<fitness(pg)
            pg=y(i,:);
        end
    end
    yn(t)=fitness(pg);
    xn(t)=pg;
end
xm=pg';
fm=fitness(pg);
%待优化的目标函数（求极小值）
function F=fitness(x)
F=0;
for i=1:1
    F = F+11*sin(7*x(i))-7*cos(3*x(i))+5*sin(6*x(i));
end

```

3. GA-PSO 混合算法代码

```
clear all
clc
% fitness学习函数
N  = 80  % N初始化群体个体数目
c1 = 1.8 % c1学习因子1
c2 = 1.8 % c2学习因子2
```

```
w  = 0.8 % w惯性权重
bc = 0.8 % bc杂交概率
bs = 0.1 % bs杂交池的大小比例
M  = 20  % M最大迭代次数
D  = 1   % D搜索空间维数

[xm,fm,xn,yn] = PSO_br (@BrF,N,c1,c2,w,bc,bs,M,D);

%得到出计算结果
disp('**************************************************');
disp('目标函数取最小值时的自变量：');
xm
disp('目标函数的最小值为：')
fm
disp('**************************************************');

n = 0:(size(yn,2)-1);
figure(1) % 绘制函数曲线及所拟合的最优解、最优值
fplot(@(x)-11.*sin(7.*x)+7.*cos(3.*x)-5.*sin(6.*x),[-1 11])
hold on
plot(xn,-yn,'r*')
hold off
figure(2) % 绘制各世代的最优值
plot(n,yn)

% 求待优化函数的极小值
function y=BrF (x)
y=0;
for i=1:1
    y=y+11*sin(7*x)-7*cos(3*x)+5*sin(6*x);
end
end

% 求解函数的极小值
function [xm,fv,xn,yn] = PSO_br(fitness,N,c1,c2,w,bc,bs,M,D)
format long;
% 初始化种群的个体
for i=1:N
    for j=1:D
        x(i,j)=randn;  %随机初始化位置
        v(i,j)=randn;  %随机初始化速度
    end
end
```

```
% 先计算各个粒子的适应度，并初始化Pi和Pg
for i=1:N
    p(i)=fitness(x(i,:));
    y(i,:)=x(i,:);
end
pg = x(N,:);              %Pg为全局最优
for i=1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg=x(i,:);
    end
end
% 进入主要循环，按照公式依次迭代
for t=1:M
    for i=1:N
        v(i,:)=w*v(i,:)+c1*rand*(y(i,:)-x(i,:))+c2*rand*(pg-x(i,:));
        x(i,:)=x(i,:)+v(i,:);
        if fitness(x(i,:))<p(i)
            p(i)=fitness(x(i,:));
            y(i,:)=x(i,:);
        end
        if p(i)<fitness(pg)
            pg=y(i,:);
        end
        r1 =rand();
        if r1 < bc
            numPool = round(bs*N);
            PoolX = x(1:numPool,:);
            PoolVX = v(1:numPool,:);
            for i=1:numPool
                seed1 = floor(rand()*(numPool-1)) + 1;
                seed2 = floor(rand()*(numPool-1)) + 1;
                pb = rand();
                childx1(i,:) = pb*PoolX(seed1,:) + (1-pb)*PoolX(seed2,:);
                 childv1(i,:) = (PoolVX(seed1,:) +
PoolVX(seed2,:))*norm(PoolVX(seed1,:))/ ...
                    norm(PoolVX(seed1,:) + PoolVX(seed2,:));
            end
            x(1:numPool,:) = childx1;
            v(1:numPool,:) = childv1;
        end
    end
    xn(t) = pg';
    yn(t) = fitness(pg);
```

```
end
xm = pg';
fv = fitness(pg);

```