Appendix
Codes of Jaya Algorithm and Its Variants

The codes for unconstrained and constrained optimization problems for sample single- and multi-objective optimization functions are given below. The user has to create separate MATLAB files but the files are to be saved in a single folder. These codes may be used for reference and the user may define the objective function(s), design variables and their ranges as per his or her own requirements.

A.1 Jaya Code for Unconstrained Rosenbrock Function

This is a complete program for solving Rosenbrock function using Jaya algorithm. In order to run the program, the following code may be copied and pasted, as it is, into the MATLAB editor file and the same may be executed. This program is only for demonstration purpose. The numbers assigned to the population size, generations, design variables and maximum function evaluations in this program need not be taken as default values.

```
%% Jaya algorithm
%% Rosenbrock function
function Jaya()
clc;
clear all;
RUNS=30;
runs=0;
while(runs<RUNS)
pop=25; % population size
var=30; % no. of design variables
maxFes=500000;
maxGen=floor(maxFes/pop);
mini=-30*ones(1,var);
maxi=30*ones(1,var);
```


R. Venkata Rao, *Jaya: An Advanced Optimization Algorithm and its Engineering Applications*, https://doi.org/10.1007/978-3-319-78922-4

```
[row,var]=size(mini);
x=zeros(pop,var);
for i=1:var
x(:,i)=mini(i)+(maxi(i)-mini(i))*rand(pop,1);
end
ch=1;
gen=0;
f=myobj(x);
while(gen<maxGen)
xnew=updatepopulation(x,f);
xnew=trimr(mini,maxi,xnew);
fnew=myobj(xnew);
for i=1:pop
if(fnew(i)<f(i))
x(i,:)=xnew(i,:);
f(i)=fnew(i);
end
end
disp('%%%%%%%% Final population %%%%%%%%%');
disp([x,f]);
fnew=[];xnew=[];
gen=gen+1;
fopt(gen)=min(f);
end
runs=runs+1;
[val,ind]=min(fopt);
Fes(runs)=pop*ind;
best(runs)=val;
end
bbest=min(best);
mbest=mean(best);
wbest=max(best);
stdbest=std(best);
mFes=mean(Fes);
stdFes=std(Fes);
fprintf('\n best=%f',bbest);
fprintf('\n mean=%f',mbest);
fprintf('\n worst=%f',wbest);
fprintf('\n std. dev.=%f',stdbest);
fprintf('\n mean function evaluations=%f',mFes);
end
function[z]=trimr(mini,maxi,x)
[row,col]=size(x);
for i=1:col
x(x(:,i)<mini(i),i)=mini(i);
```

```
x(x(:,i)>maxi(i),i)=maxi(i);
end
z=x;
end
function [xnew]=updatepopulation(x,f)
[row,col]=size(x);
[t,tindex]=min(f);
Best=x(tindex,:);
[w,windex]=max(f);
worst=x(windex,:);
xnew=zeros(row,col);
for i=1:row
for j=1:col
r=rand(1,2);
xnew(i,j)=x(i,j)+r(1)*(Best(j)-abs(x(i,j)))-r(2)*(worst(j)-abs(x(i,j)));
end
end
end
function [f]=myobj(x)
[r,c]=size(x);
for i=1:r
y=0;
for j=1:c-1
y=y+(100*(x(i,j)^2-x(i,j+1))^2+(1-x(i,j))^2);
end
z(i)=y;
end
f=z';
end
```

A.2 Jaya Code for Constrained Himmelblau Function

This is a complete program for solving constrained Himmelblau function using Jaya algorithm. In order to run the program, the following code may be copied and pasted, as it is, into the Matlab editor file and the same may be executed. This program is only for demonstration purpose. The numbers assigned to the population size, generations, design variables and maximum function evaluations in this program need not be taken as default values.

```
%% Jaya algorithm
%% Constrained optimization
%% Himmelblau function
function Jaya()
```

```
clc;
clear all;
RUNS=10;
runs=0;
while(runs<RUNS)
pop=5; % population size
var=2; % no. of design variables
maxFes=150000;
maxGen=floor(maxFes/pop);
mini=[-5 -5];
maxi=[5 5];
[row,var]=size(mini);
x=zeros(pop,var);
for i=1:var
x(:,i)=mini(i)+(maxi(i)-mini(i))*rand(pop,1);
end
ch=1;
gen=0;
f=myobj(x);
while(gen<maxGen)
xnew=updatepopulation(x,f);
xnew=trimr(mini,maxi,xnew);
fnew=myobj(xnew);
for i=1:pop
if(fnew(i)<f(i))
x(i,:)=xnew(i,:);
f(i)=fnew(i);
end
end
disp('%%%%%% Final population%%%%%%%');
disp([x,f]);
fnew=[];xnew=[];
gen=gen+1;
fopt(gen)=min(f);
end
runs=runs+1;
[val,ind]=min(fopt);
Fes(runs)=pop*ind;
best(runs)=val;
end
bbest=min(best);
mbest=mean(best);
wbest=max(best);
stdbest=std(best);
mFes=mean(Fes);
```

```
fprintf('\n best=%f',bbest);
fprintf('\n mean=%f',mbest);
fprintf('\n worst=%f',wbest);
fprintf('\n std. dev.=%f',stdbest);
fprintf('\n mean Fes=%f',mFes);
end
function[z]=trimr(mini,maxi,x)
[row,col]=size(x);
for i=1:col
x(x(:,i)<mini(i),i)=mini(i);
x(x(:,i)>maxi(i),i)=maxi(i);
end
z=x;
end
function [xnew]=updatepopulation(x,f)
[row,col]=size(x);
[t,tindex]=min(f);
Best=x(tindex,:);
[w,windex]=max(f);
worst=x(windex,:);
xnew=zeros(row,col);
for i=1:row
for j=1:col
r=rand(1,2);
xnew(i,j)=x(i,j)+r(1)*(Best(j)-abs(x(i,j)))-r(2)*(worst(j)-abs(x(i,j)));
end
end
end
function [f]=myobj(x)
[r,c]=size(x);
Z=zeros(r,1);
for i=1:r
x1=x(i,1);
x2=x(i,2);
z=(((x1^2)+x2-11)^2)+((x1+x2^2-7)^2);
g1=26-((x1-5)^2)-((x2)^2);
g2=20-(4*x1)-x2;
p1=10*((min(0,g1))^2); % penalty if constraint 1 is violated
p2=10*((min(0,g2))^2); % penalty if constraint 2 is violated
Z(i)=z+p1+p2; % penalized objective function value
end
f=Z;
end
```

A.3 SAMPE Jaya Code for Unconstrained Himmelblau Function

This is a complete program for solving unconstrained Himmelblau functionusing self-adaptive multi-population elitist-Jaya algorithm. The user has to create separate MATLAB files (all saved in a single folder). He or she can define the design variables, ranges and the objective function(s) as per his or her requirements.

```
%%%%%%%%%%%%%%%% objective%%%%%%%%%%%%%%%%%%
function [ Z ] = objective( x )
% Unconstrained optimization
% Himmelblau function
% Detailed explanation goes here
x1=x(1);
x2=x(2);
Z=(((x1^2)+x2-11)^2)+((x1+x2^2-7)^2);
end
%%%%%%%%Algorithm_MP_ES%%%%%%%%%%%%%%
clc
clearall
funev=150000; % Maximum functionevaluations
IDR=1;
formatshortg
%NP = input('Enter the population size:');
NP=25; % Population size
ES=2; % Elite size
NMP=2 ;% Number of multi population initial
NG=funev/NP; % Maximum number of generations
DV=2; % Number of design variables
% NG = input('Enter the no of generation :');
% DV = input('Enter the no of Decision variable:');
for m=1:IDR % No. of Independent run
for j = 1 : DV
```

```
DV_min(j)=[-5];
DV_max(j)=[5];
end
% Initial Population Generation
for i=1:NP
for j = 1 : DV
x(i,j) = DV_min(j) + (DV_max(j) - DV_min(j))*rand(1);
end
end
for i = 1 : NP
x(i,DV + 1) = objective(x(i,:));
end
% start of generation
for k=1:NG
x=unique(x, 'rows');
[r,c]= size(x);
if r<NP  % If current population size is less than NP add NP-r     %popula-
tions
for i=r+1:NP
for j = 1 : DV
x(i,j) = DV_min(j) + (DV_max(j) - DV_min(j))*rand(1);
end
end
for i = r+1:NP
x(i,DV + 1) = objective(x(i,:));
end
else
x(:,:)=x(:,:);
end
%% Replace the worst solution with elite solutions
x=sortrows(x,DV+1); % sort the solutions in ascending order
x(NP-ES+1:NP,:)= x(1:ES,:);
[f_min,ind_min] = min(x(:,DV+1));
[f_max,ind_max] = max(x(:,DV+1));
%% Devide population and modify the solutions
x1=[];
for n=1:NMP
  a=round((n-1)*NP/NMP)+1;
  b=round(NP/NMP*n);
if b>NP
for i=NP+1:b
for j=1:DV
x(i,j)= randsample(x(:,j),1);
end
x(i,DV + 1) = objective(x(i,:));
```

```
end
end
for i=a:b
for j=1:DV
     x1(i,j) = x(i,j) +rand*(x(a,j)- abs(x(i,j)))-rand*(x(b,j)- abs(x(i,
j)));
end
end
end
% Check for the bounds of decision variable
For i=1:NP
for j=1:DV
if x1(i,j)<DV_min(j)
x1(i,j)= DV_min(j);
elseif x1(i,j)>DV_max(j)
x1(i,j) =  DV_max(j);
else
x1(i,j) =  x1(i,j);
end
end
end
for i = 1 : NP
x1(i,DV + 1) = objective(x1(i,:));
end
for i=1:NP
if  x(i,DV + 1)< x1(i,DV + 1)
x(i,DV + 1)= x(i,DV + 1);
for j=1:DV+1
x(i,j) = x(i,j);
end
else
x(i,DV + 1)= x1(i,DV + 1);
for j=1:DV+1
x(i,j) = x1(i,j);
end
end
end
[f_best(k),l2] = min(x(:,DV+1));
for j=1:DV
x_fi(j)= x(l2,j);
end
```

```
%%%%%%%self-adaptive population concept starts over here
if (f_min<f_best(k))
     NMP=NMP+1;
if (NP/NMP)>=3
        NMP=NMP;
else
     NMP=NMP-1;
end
else
if NMP>1
  NMP= NMP-1;
else
  NMP=1;
end
end
np(k)=NMP;
% disp(NMP)
%%%%%%%%%%%%%%%% self-adaptive populations concept ends over
end
disp(['————————[ ','Run No.= ',num2str(m), ' ]——————————————————'])
disp(['Optimum value =',num2str(min(f_best),30)])
disp(['DV: ',num2str(x_fi,7)])
fprintf('\n \n');
 [f_min1(m),ind1(m)]= min(f_best);
 f_max1(m)= max(f_best);
end
  Best= min(f_min1);
  Worst=max(f_min1);
  Mean = mean(f_min1);
  SD  = std(f_min1);
  MFE= mean(ind1)*NP;
fprintf('\n Best=%f',Best);
fprintf('\n Worst=%f',Worst);
fprintf('\n Mean=%f',Mean);
fprintf('\n SD=%f', SD);
fprintf('\n MFE=%f', MFE);
fprintf('\n FE=%f',funev);
fprintf('\n \n');
```

A.4 SAMPE Jaya Code for Constrained Himmelblau Function

This is a complete program for solving constrained Himmelblau function using self-adaptive multi-population elitist-Jaya algorithm. The user has to create separate MATLAB files (all saved in a single folder). He or she can define the ranges, design variables and the objective function(s) as per his or her requirements.

```
%%%%%%%%%%%%%%%% objective%%%%%%%%%%%%%%%%%%%

function [ Z ] = objective( x )
% Constrained optimization
% Himmelblau function
% Detailed explanation goes here
x1=x(1);
x2=x(2);
z=(((x1^2)+x2-11)^2)+((x1+x2^2-7)^2);
g1=26-((x1-5)^2)-((x2)^2);
g2=20-(4*x1)-x2;
p1=10*((min(0,g1))^2); % penalty if constraint 1 is violated
p2=10*((min(0,g2))^2); % penalty if constraint 2 is violated
Z=z+p1+p2; % penalized objective function value
end
%%%%%%%%%Algorithm_MP_ES%%%%%%%%%%%%%%
clc
clearall
funev=150000; % Maximum function evaluations
IDR=1;
formatshortg
%NP = input('Enter the population size:');
NP=25; % Population size
ES=2;  % Elite size
NMP=2 ;% Number of multi population initial
NG=funev/NP;  % Maximum number of generations
DV=2; % Number of design variables
%  NG = input('Enter the no of generation :');
%  DV = input('Enter the no of Decision variable:');
for m=1:IDR % No. of Independent run
for j = 1 : DV
DV_min(j)=[-5];
DV_max(j)=[5];
end
% Initial Population Generation
for i=1:NP
for j = 1 : DV
```

```
x(i,j) = DV_min(j) + (DV_max(j) - DV_min(j))*rand(1);
end
end
for i = 1 : NP
x(i,DV + 1) = objective(x(i,:));
end
% start of generation
for k=1:NG
x=unique(x, 'rows');
[r,c]= size(x);
if r<NP  % If current population size is less than NP add NP-r      %popula-
tions
fori=r+1:NP
for j = 1 : DV



x(i,j) = DV_min(j) + (DV_max(j) - DV_min(j))*rand(1);
end
end
for i = r+1:NP
x(i,DV + 1) = objective(x(i,:));
end
else
x(:,:)=x(:,:);
end
%% Replace the worst solution with elite solutions
x=sortrows(x,DV+1); % sort the solutions in ascending order
x(NP-ES+1:NP,:)= x(1:ES,:);
[f_min,ind_min] = min(x(:,DV+1));
[f_max,ind_max] = max(x(:,DV+1));
%% Devide population and modify the solutions
x1=[];
for n=1:NMP
  a=round((n-1)*NP/NMP)+1;
  b=round(NP/NMP*n);
if b>NP
for i=NP+1:b
for j=1:DV
x(i,j)= randsample(x(:,j),1);
end
x(i,DV + 1) = objective(x(i,:));
end
end
for i=a:b
```

```
for j=1:DV
     x1(i,j) = x(i,j) +rand*(x(a,j)- abs(x(i,j)))-rand*(x(b,j)- abs(x(i,
j)));
end
end
end
% Check for the bounds of decision variable
For i=1:NP
for j=1:DV
if x1(i,j)<DV_min(j)
x1(i,j)= DV_min(j);
elseif x1(i,j)>DV_max(j)
x1(i,j) =  DV_max(j);
else
x1(i,j) =  x1(i,j);
end
end
end
for i = 1 : NP
x1(i,DV + 1) = objective(x1(i,:));
end
for i=1:NP
if  x(i,DV + 1)< x1(i,DV + 1)
x(i,DV + 1)= x(i,DV + 1);
for j=1:DV+1
x(i,j) = x(i,j);
end
else
x(i,DV + 1)= x1(i,DV + 1);
for j=1:DV+1
x(i,j) = x1(i,j);
end
end
end
 [f_best(k),l2] = min(x(:,DV+1));
for j=1:DV
x_fi(j)= x(l2,j);
end
%%%%%%% self-adaptive population concept startsover here
if (f_min<f_best(k))
     NMP=NMP+1;
if (NP/NMP)>=3
        NMP=NMP;
else
     NMP=NMP-1;
```

```
end
else
if NMP>1
  NMP= NMP-1;
else
  NMP=1;
end
end
np(k)=NMP;
% disp(NMP)
%%%%%%%%%self-adaptive populations concept ends over here%%%%%
end
disp(['————————[ ','Run No.= ',num2str(m), ' ]————————'])
disp(['Optimum value =',num2str(min(f_best),30)])
disp(['DV: ',num2str(x_fi,7)])
fprintf('\n \n');
 [f_min1(m),ind1(m)]= min(f_best);
 f_max1(m)= max(f_best);
end
  Best= min(f_min1);
  Worst=max(f_min1);
  Mean = mean(f_min1);
  SD  = std(f_min1);
  MFE= mean(ind1)*NP;
fprintf('\n Best=%f',Best);
fprintf('\n Worst=%f',Worst);
fprintf('\n Mean=%f',Mean);
fprintf('\n SD=%f', SD);
fprintf('\n MFE=%f', MFE);
fprintf('\n FE=%f',funev);
fprintf('\n \n');
```

A.5 MOQO Jaya Code for ECM Process

The following is the code for multi-objective quasi-oppositional (MOQO) Jaya algorithm for electro-chemical machining (ECM) process. This is a sample program for reference.

```
function [z]=moqojaya()
RUNS=0;
maxRUNS=1;
while(RUNS<maxRUNS)
clearvars -except maxRUNS RUNS;
```

```
clc;
FEs=0;
maxFEs=3750;
gen=0;
temp_cl=[];
pop=50;
sch=[0 0 1];
mini=[8 300 3];
maxi=[200 5000 21];
dim=numel(mini);
n=numel(sch);
for i=1:dim
cl(:,i)=mini(i)+(maxi(i)-mini(i))*rand(pop,1);
end
cl_new=quasi(mini,maxi,cl);
cl=[cl;cl_new];
cl(:,dim+1:dim+n)=myobj(cl);
temp_cl=cl;
cl=[];
b=[];
cl=NS(n,dim,temp_cl,sch);
cl=cl(1:pop,:);
temp_cl=[];
while(FEs<=maxFEs)
a=[]; b=[]; a= cl(:,dim+n+1)==1; b=find(cl(a,dim+n+2)==100); count=numel(b);
temp_cl=updation(cl(:,1:dim),count);
temp_cl=trimr(mini,maxi,temp_cl);
temp_cl(:,dim+1:dim+n)=myobj(temp_cl);
comb_cl=cat(1,cl(:,1:dim+n),temp_cl);
temp_cl=[];
cl=[];
cl=NS(n,dim,comb_cl,sch);
cl=cl(1:pop,:);
comb_cl=[];
FEs=FEs+pop;
temp_cl=quasi(mini,maxi,cl(:,1:dim));
temp_cl(:,dim+1:dim+n)=myobj(temp_cl);
comb_cl=cat(1,cl(:,1:dim+n),temp_cl);
temp_cl=[];
cl=[];
cl=NS(n,dim,comb_cl,sch);
cl=cl(1:pop,:);
comb_cl=[];
cl=cat(2,cl,constraint(cl(:,1:dim)));
FEs=FEs+pop;
```

```
disp(FEs);
plot3(cl(:,dim+1),cl(:,dim+2),cl(:,dim+3),'*');
 xlabel('Z1');
 ylabel('Z2');
 zlabel('Z3');
gen=gen+1;
disp(gen);
figure(1);
end
RUNS=RUNS+1;
end
end


function [z1]= NS(n,dim,cl,sch)
[pop,col]=size(cl);
x=cl(:,1:dim);
func=cl(:,dim+1:dim+n);
f=func;
srn=[1:pop];
cons=constraint(x);
tempf=f;
[p,c]=size(f);
l=1;
rank=1;
score=[];
tscore=zeros(1,2);
nd=[];
todel=[];
index=[];
t=[];
while(p>1)
for i=1:p
  win=1;
  for j=1:p
    if(i~=j)
      if(cons(i)<cons(j))
        win=win+1;
      end
      if(cons(i)>cons(j))
        break;
      end


     if(cons(i)==cons(j))
```

```
for k=1:n
 switch sch(k)
     case 0
       if(f(i,k)<f(j,k))
       score(1,k)=1;
       score(2,k)=0;
       end
       if(f(i,k)==f(j,k))
       score(1,k)=0;
       score(2,k)=0;
       end
       if(f(i,k)>f(j,k))
        score(1,k)=0;
        score(2,k)=1;
       end
     case 1
       if(f(i,k)>f(j,k))
         score(1,k)=1;
         score(2,k)=0;
       end
       if(f(i,k)==f(j,k))
         score(1,k)=0;
         score(2,k)=0;
       end
       if(f(i,k)<f(j,k))
         score(1,k)=0;
         score(2,k)=1;
       end


 end
end
 tscore(1)=sum(score(1,:));
 tscore(2)=sum(score(2,:));


 if(tscore(1)>0.0 && tscore(1)<=n)
    win=win+1;
 end
      end
     end
   end
```

```
   if(win==p)
     nd(l)=srn(i);
     todel(l)=i;
     l=l+1;
   end


   score=[];
   tscore=zeros(1,2);
end
if(numel(nd)==0)


  tempf(srn,n+1)=rank;
  rank=rank+1;
   f=[];
   srn=[];
   cons=[];
else
tempf(nd,n+1)=rank;
rank=rank+1;
f(todel,:)=[];
srn(todel)=[];
cons(todel)=[];
end
nd=[];
todel=[];
l=1;


[p,c]=size(f);
end
tempf(tempf(:,n+1)==0,n+1)=rank;
x=cat(2,x,tempf);
[t,index]=sort(tempf(:,n+1),'ascend');
x=x(index,:);
%disp(x);


for i=1:n
fmin(i)=min(x(:,dim+i));
fmax(i)=max(x(:,dim+i));
end
r=max(x(:,dim+n+1));
%disp(r);
```

```
for k=1:r
  t=[];
  index=[];
  f2=x(x(:,dim+n+1)==k,dim+1:dim+n);
[row,col]=size(f2);
if(row<2)
  x(x(:,dim+n+1)==k,dim+n+2)=100;
else


  x(x(:,dim+n+1)==k,dim+n+2)=crowdist(fmin,fmax,f2);
  x1=x(x(:,dim+n+1)==k,:);
  [t,index]=sort(x1(:,dim+n+2),'descend');
  x1=x1(index,:);
  x(x(:,dim+n+1)==k,:)=x1;
end


end
z1=x;


end


function [d]=crowdist(fmin,fmax,f)


[row,col]=size(f);
d=zeros(row,1);


for i=1:col
  [I,t]=sort(f(:,i));
  %disp(t);
  d(t(1))=100;
  d(t(row))=100;
  for j=2:row-1
    d(t(j))=d(t(j))+(f(t(j+1),i)-f(t(j-1),i))/(fmax(i)-fmin(i));
  end
end
d(d>100)=100;
end
```

```
function [z2]=updation(temp_cl,count)
[row,col]=size(temp_cl);
best=temp_cl(round(count-(count-1)*rand(1,1)),:);
worst=temp_cl(end,:);
for j=1:row
  for i=1:col
  rn=rand(1,2);
  temp_cl(j,i)=temp_cl(j,i)+rn(1)*(best(i)-abs(temp_cl(j,i)))-rn(2)*
(worst(i)-abs(temp_cl(j,i)));
  end
end
z2=temp_cl;
end


function[z4]=trimr(mini,maxi,temp)
[row,col]=size(temp);
for i=1:col
  temp(temp(:,i)<mini(i),i)=mini(i);
  temp(temp(:,i)>maxi(i),i)=maxi(i);
end
  z4=temp;
end


function [f1]=myobj(x)
[row,col]=size(x);


for i=1:row


f=x(i,1);
U=x(i,2);
V=x(i,3);


Z1=(f^0.381067)*(U^-0.372623)*(V^3.155414)*exp(-3.128926);
Z2 = (f^3.528345)*(U^0.000742)*(V^-2.52255)*exp(0.391436);
Z3=f;
```

```
f1(i,1)=Z1;
f1(i,2)=Z2;
f1(i,3)=Z3;
end
end


function[z5]=constraint(x)
[row,col]=size(x);
for i=1:row


f=x(i,1);
U=x(i,2);
V=x(i,3);


g1=(1-(f^2.133007)*(U^-1.088937)*(V^-0.351436)*exp(0.321968));
g2=((f^-0.844369)*(U^-2.526075)*(V^1.546257)*exp(12.57697))-1;
g3=1-((f^0.075213)*(U^-2.488362)*(V^0.240542)*exp(11.75651));


G1(i,1)=min(0,g1);
G2(i,1)=min(0,g2);
G3(i,1)=min(0,g3);
end
G1str=max(abs(G1));
G2str=max(abs(G2));
G3str=max(abs(G3));


if(max(abs(G1))==0)
  G1str=1;
end


if(max(abs(G2))==0)
  G2str=1;
end
```

```
if(max(abs(G3))==0)
  G3str=1;
end


z5=(abs(G1)/G1str)+(abs(G2)/G2str)+(abs(G3)/G3str);
end


function [Pnew]=quasi(mini,maxi,P)
[row,col]=size(P);
mini=repmat(mini,row,1);
maxi=repmat(maxi,row,1);
A=0.5*(mini+maxi);
B=((mini+maxi)-P);
Pnew=A+(B-A).*rand(row,col);
end
```

In addition to the above, the readers may also refer to https://sites.google.com/site/jayaalgorithm/.

Index

R. Venkata Rao, *Jaya: An Advanced Optimization Algorithm and its Engineering Applications*, https://doi.org/10.1007/978-3-319-78922-4