AE 551: Introduction to Optimal Control **(Due: 2020/06/12)**

Homework #9 Submission

Instructor: Min-Jea Tahk *Name:* Joshua Julian Damanik, *Student ID:* 20194701

Solve the four problems treated in the paper "Coevolutionary Augmented Lagrangian Methods for Constrained Optimization" by using CEALM and PSO. The paper is included in the ZIP file of cealm_v20. For each problem, run the codes of CEALM and PSO for at least 10 times to find the best and worst results of each method.

- CEALM: Use the CEALM code posted on KLMS.
- PSO:
  - You can use any PSO code but you need to explicitly state the source of the code (for example, URL or the reference papers)
  - Describe in detail the PSO algorithm you use in the homework report. (Do not use any code you don't understand the details.)
  - Attach the code to your report.

(Solution) Code for PSO method is attached at the Appendix section. However, the rest of the code can be accessed from `https://github.com/joshuadamanik/Homework-9`.

For the CEALM method [1], the code used in this homework is the one provided at cealm_v20_ae551.zip. However, the code for the PSO method is created based on proposed algorithm [2] by Zambrano-Bigiarini, et al.

Algorithm 1: Standard Particle Swarm Optimization

for *each particle:* $i \leftarrow 1, \ldots, N$ **do**
  initialize random particle's parameter: $X_i \in U(X_{min}, X_{max})$;
  initialize random particle's velocity: $V_i \in U(-(X_{max} - X_{min}), (X_{max} - X_{min}))$;
  initialize particle's best parameter: $\bar{X}_i \leftarrow x_i$;
  initialize swarm's best parameter: $X^* \leftarrow \arg\min f(\bar{X}_i)$;
end
while *termination criteria is not satisfied* **do**
  for *each particle:* $i \leftarrow 1, \ldots, N$ **do**
    for *each dimension:* $j \leftarrow 1, \ldots, M$ **do**
      initialize random numbers: $r_p, r_g \in U(0, 1)$;
      update particle's velocity: $V_{i,j} \leftarrow \omega V_{i,j} + c_1 r_p(\bar{X}_{i,j} - X_{i,j}) + c_2 r_g(X^*_j - X_{i,j})$;
    end
    update particle's parameter: $X_i \leftarrow X_i + v_i$;
    if $f(X_i) < f(\bar{X}_i)$ **then**
      update particle's best parameter: $\bar{X}_i \leftarrow X_i$;
      if $f(\bar{X}_i) < f(X^*)$ **then**
        update swarm's best parameter: $X^* \leftarrow \bar{X}_i$;
      end
    end
  end
end

However, the algorithm 1 is an unconstrained optimization method. For constrained optimization, the objective function $f(X)$ is replaced by augmented Lagrangian function $L_A(X, Y)$, defined as

$$L(X_i, \lambda_i, \mu_i) = f(X_i) + \rho \sum_{j=1}^{N_{ineq}} \max{}^2 \left\{ g_j(X_i) + \frac{\mu_{i,j}}{2\rho} \right\} + \sum_{j=1}^{N_{ineq}} \frac{\mu_{i,j}^2}{4\rho} + \lambda_i^T h(X_i) + \rho h(X_i)^T h(X_i) \tag{1}$$

where $g(X_i) \leq 0$ is set of inequality constraints and $h(X_i) = 0$ is set of equality constraints with their Lagrange multiplier μ_i and λ_i respectively. The multiplier μ_i and λ_i is then combined into a particle object Y_i with dimension $N_{ineq} + N_{eq}$.

By using augmented Lagrangian function (eq. 1), the optimization is done in an unconstrained fashion by solving both the X and Y particles. But, instead of searching for minimum value, we need to find the maximum value for particle Y. Then, the swarm's best parameter both for X and Y are selected by using security strategy [1].

In addition, because all of the problems defined in this homework have restricted search space, the softwall algorithm [1] is added into the Standard PSO (algorithm 1). While the search space of particle X is defined in each problem, the search space for particle Y is defined as

$$Y_{min} = \begin{cases} 0, & \text{inequality constraint} \\ -10, & \text{equality constraint} \end{cases} \tag{2}$$

$$Y_{max} = 10; \tag{3}$$

Problem 1

$$\min f(X) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{1} 3x_i \tag{4}$$

$$\begin{aligned} \text{subject to} \quad 2x_1 + 2x_2 + x_{10} + x_{11} &\leq 10 \\ 2x_1 + 2x_3 + x_{10} + x_{12} &\leq 10 \\ 2x_1 + 2x_3 + x_{10} + x_{12} &\leq 10 \\ -8x_1 + x_{10} &\leq 0 \\ -8x_2 + x_{11} &\leq 0 \\ -8x_3 + x_{12} &\leq 0 \\ -2x_4 - x_5 + x_{10} &\leq 0 \\ -2x_6 - x_7 + x_{11} &\leq 0 \\ -2x_8 - x_9 + x_{12} &\leq 0 \end{aligned}$$

$$\text{search space:} \quad 0 \leq x_i \leq 1, i = 1, \ldots, 9; \quad 0 \leq x_i \leq 10, i = 10, 11, 12; \quad 0 \leq x_{13} \leq 1$$

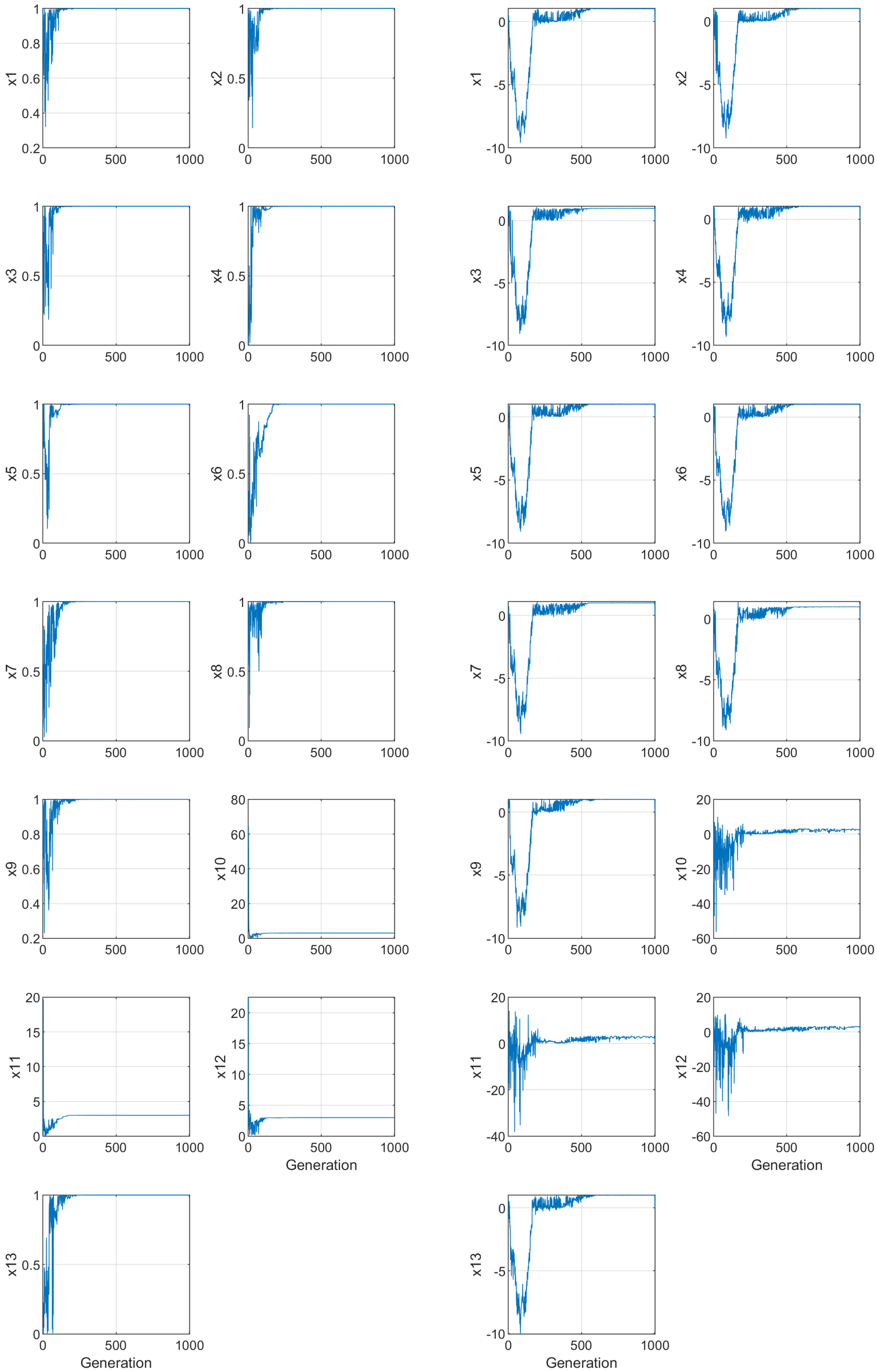
(Solution)

Table 1: Cost comparison of CEALM and PSO. Lowest cost is labelled with bold.

Run	CEALM	PSO	Analytical
1	-15.0000	-13.6296	-15.0000
2	-15.0000	**-14.5906**	
3	-15.0000	-4.7454	
4	-13.8281	-13.0326	
5	-15.0000	-14.1221	
6	-15.0000	-4.5065	
7	-13.8281	-13.1253	
8	-15.0000	-14.4504	
9	-15.0000	-3.8945	
10	**-15.0000**	-14.2518	

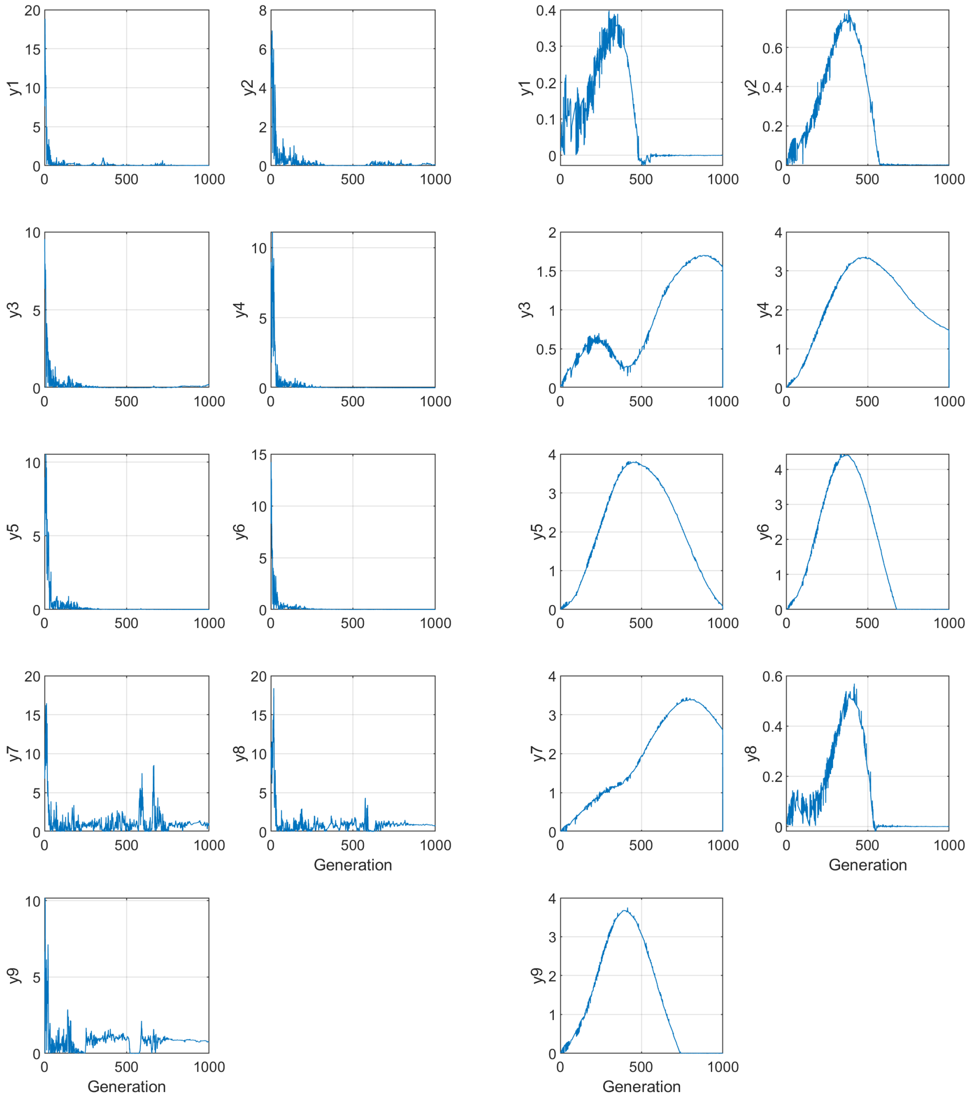
Table 2: Optimal parameter value of problem 1 with lowest cost

	CEALM	PSO		CEALM	PSO
x_1	1.0000	0.9993	y_1	0.0289	-0.0002
x_2	1.0000	0.9982	y_2	0.0754	-0.0004
x_3	1.0000	0.9995	y_3	0.1624	1.5536
x_4	1.0000	0.9999	y_4	0.0000	1.4803
x_5	1.0000	0.9991	y_5	0.0002	0.0763
x_6	1.0000	0.9999	y_6	0.0000	-0.0002
x_7	1.0000	0.9999	y_7	0.8247	2.6092
x_8	1.0000	0.9995	y_8	0.8140	-0.0005
x_9	1.0000	0.9990	y_9	0.7531	0.0017
x_{10}	3.0000	2.8896			
x_{11}	3.0000	2.7007			
x_{12}	3.0000	3.0186			
x_{13}	1.0000	0.9997			



(a) CEALM (b) PSO

Figure 1: Parameter X of problem 1 with lowest cost



(a) CEALM (b) PSO

Figure 2: Parameter Y of problem 1 with lowest cost

Problem 2

$$\begin{aligned}\min f(X) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3-10)^2 + 4(x_4-5)^2 + (x_5-3)^2 + 2(x_6-1)^2 \\ + 5x_7^2 + 7(x_8-11)^2 + 2(x_9-10)^2 + (x_10-7)^2 + 45\end{aligned} \tag{5}$$

$$\begin{aligned}\text{subject to} \quad 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0 \\ -3(x_1-2)^2 - 4(x_2-3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0 \\ -10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0 \\ -x_1^2 - 2(x_2-2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0 \\ 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0 \\ -5x_1^2 - 8x_2 - (x_3-6)^2 + 2x_4 + 40 \geq 0 \\ 3x_1 - 6x_2 - 12(x_9-8)^2 + 7x_{10} \geq 0 \\ -0.5(x_1-8)^2 - 2(x_2-4) - 3x_5^2 + x_6 + 30 \geq 0\end{aligned}$$

$$\text{search space:} \quad -10 \leq x_i \leq 10, i = 1, \ldots, 10$$

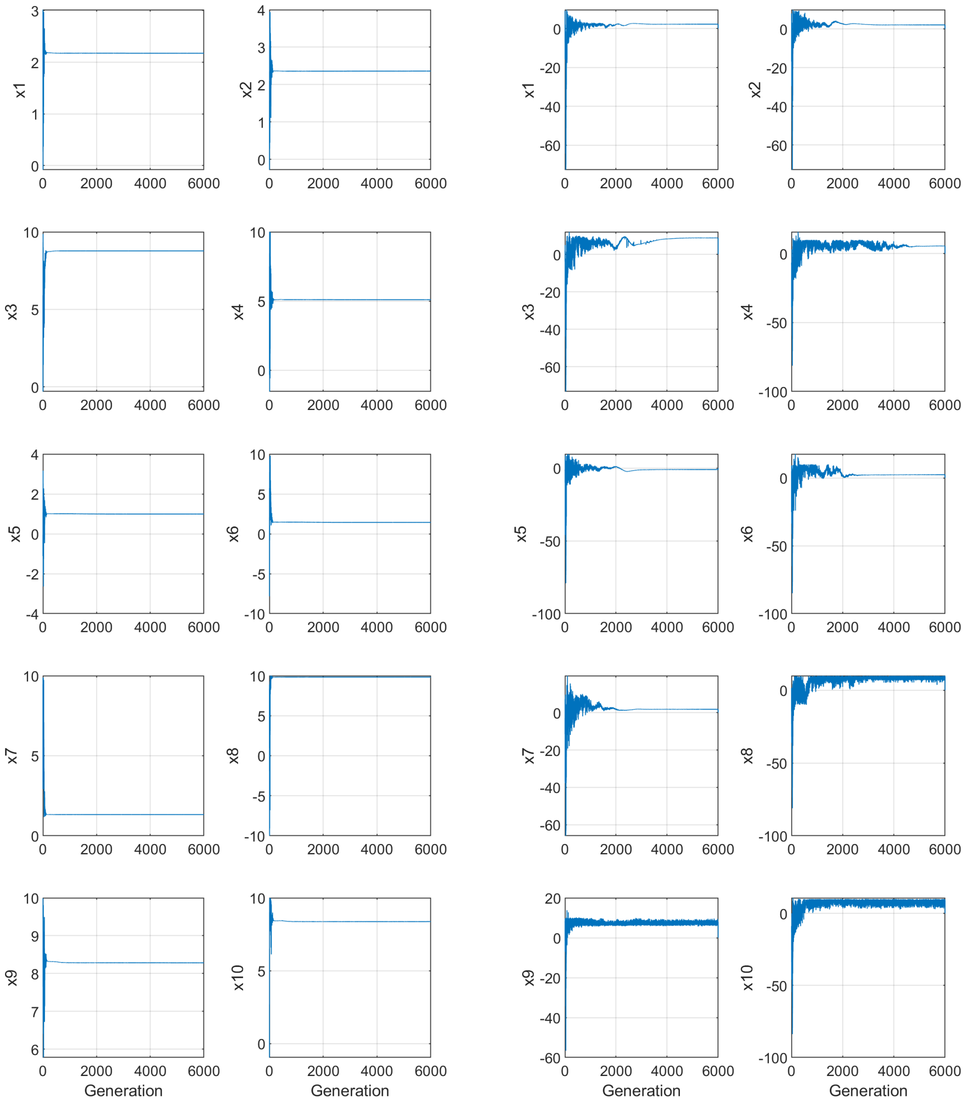
(Solution)

Table 3: Cost comparison of CEALM and PSO. Lowest cost is labelled with bold.

Run	CEALM	PSO	Analytical
1	24.4014	356.8283	24.3060
2	24.3097	107.9832	
3	24.4034	105.9014	
4	24.3591	270.3384	
5	24.3325	63.6385	
6	24.3077	203.3939	
7	24.3547	170.0315	
8	24.3079	93.1403	
9	24.3078	**61.1770**	
10	**24.3065**	70.9870	

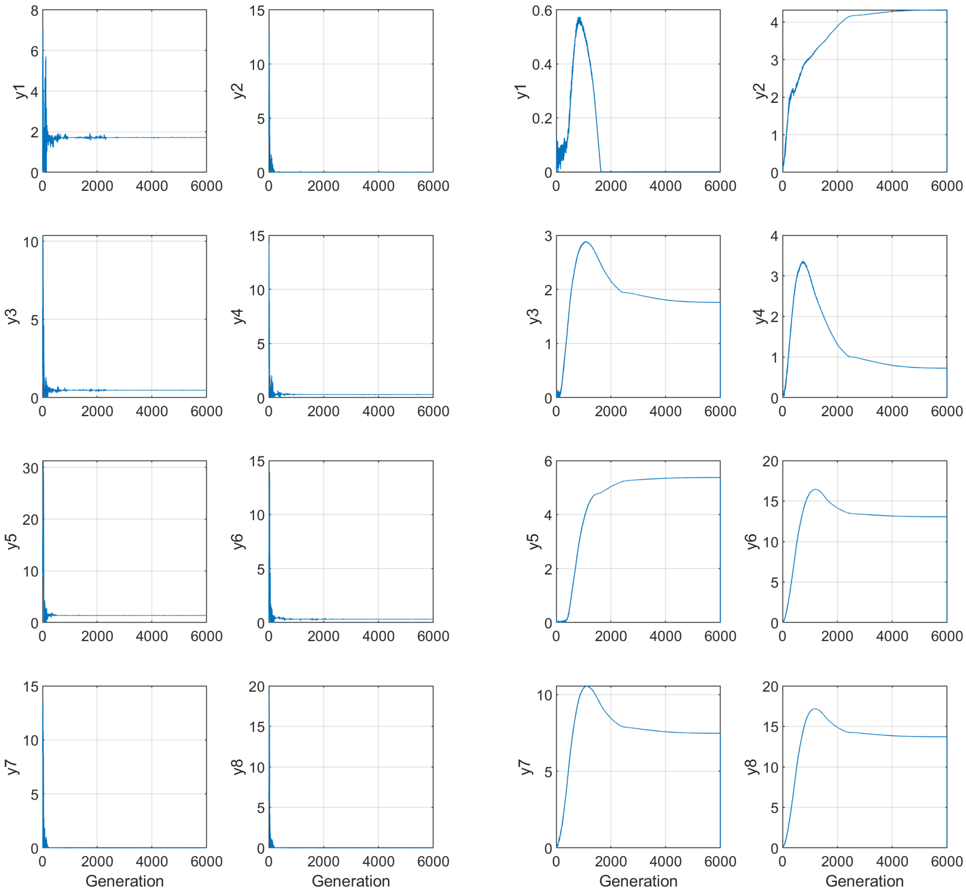
Table 4: Optimal parameter value of problem 2 with lowest cost

	CEALM	PSO		CEALM	PSO
x_1	2.1736	2.2521	y_1	1.7130	0.0000
x_2	2.3598	2.1415	y_2	0.0205	4.3195
x_3	8.7734	8.7517	y_3	0.4761	1.7603
x_4	5.0963	5.5805	y_4	0.2856	0.7250
x_5	0.9904	-0.9687	y_5	1.3778	5.3716
x_6	1.4318	2.3595	y_6	0.3054	13.0650
x_7	1.3247	1.7573	y_7	0.0001	7.4777
x_8	9.8312	9.4168	y_8	0.0004	13.7182
x_9	8.2833	8.0274			
x_{10}	8.3736	9.0898			



(a) CEALM (b) PSO

Figure 3: Parameter X of problem 2 with lowest cost



(a) CEALM (b) PSO

Figure 4: Parameter Y of problem 2 with lowest cost

Problem 3

$$\min f(X) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \quad (6)$$

$$\begin{aligned} \text{subject to} \quad 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 &\geq 0 \\ 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 &\geq 0 \\ 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 &\geq 0 \\ -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 &\geq 0 \end{aligned}$$

$$\text{search space:} \quad -10 \leq x_i \leq 10, i = 1, \ldots, 7$$

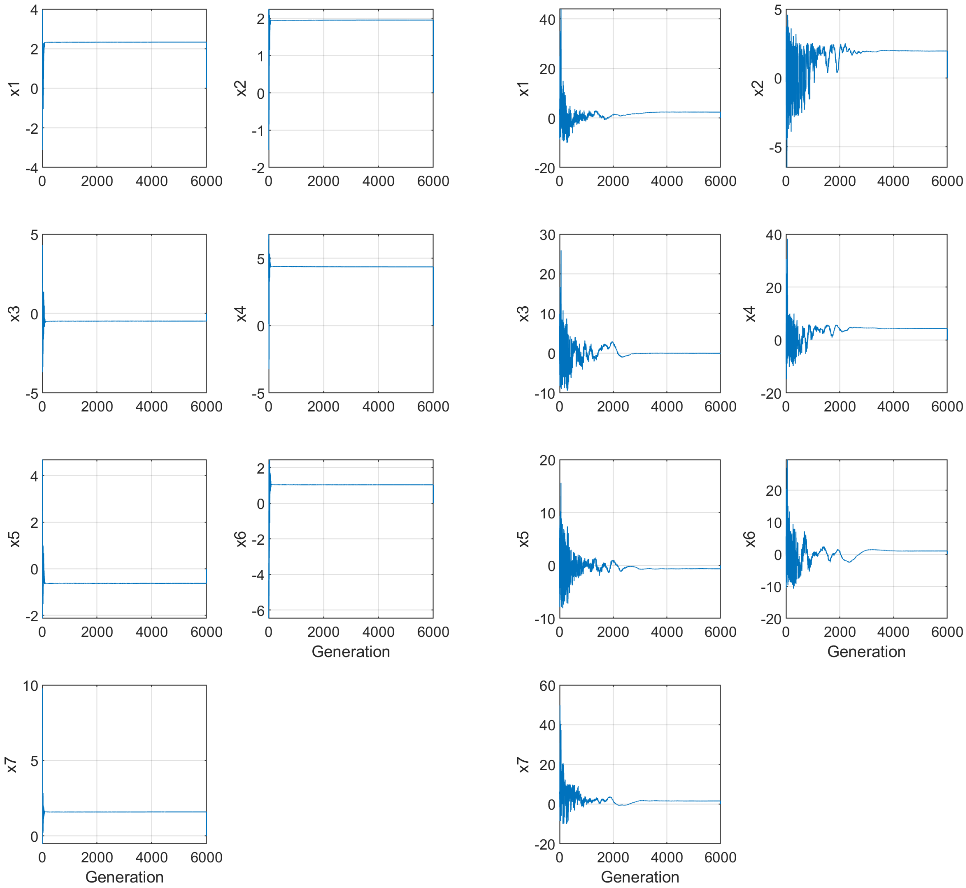
(Solution)

Table 5: Cost comparison of CEALM and PSO. Lowest cost is labelled with bold.

Run	CEALM	PSO	Analytical
1	680.6305	686.4466	680.6300
2	680.6300	693.6093	
3	680.6301	10007452.1252	
4	680.6301	838.4517	
5	680.6303	39618.8533	
6	**680.6300**	**698.7194**	
7	680.6314	681.0175	
8	680.6305	782.3090	
9	680.6300	803.8013	
10	680.6301	1307.3733	

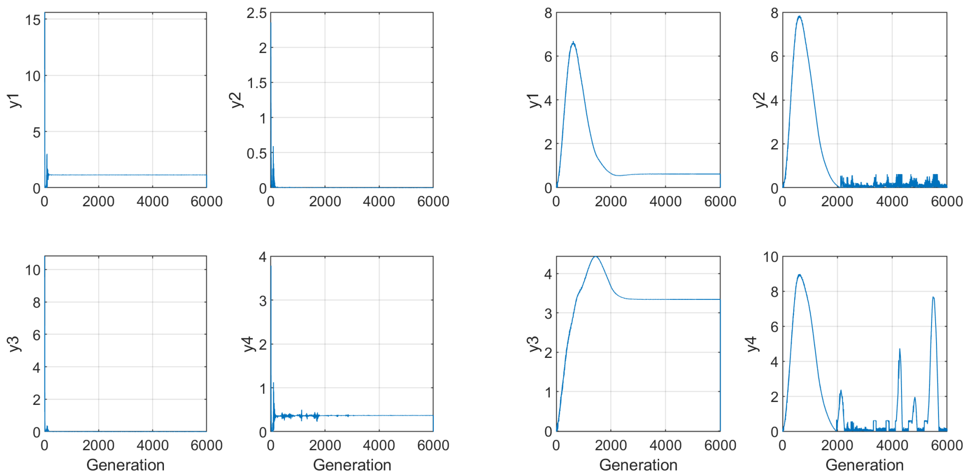
Table 6: Optimal parameter value of problem 3 with lowest cost

	CEALM	PSO		CEALM	PSO
x_1	2.3303	2.3486	y_1	1.1404	0.6142
x_2	1.9511	1.9554	y_2	0.0001	0.0531
x_3	-0.4777	-0.0253	y_3	0.0002	3.3404
x_4	4.3665	4.3385	y_4	0.3685	0.1580
x_5	-0.6245	-0.6265			
x_6	1.0382	1.0181			
x_7	1.5941	1.5908			



(a) CEALM (b) PSO

Figure 5: Parameter X of problem 3 with lowest cost



(a) CEALM (b) PSO

Figure 6: Parameter Y of problem 3 with lowest cost

Problem 4

$$\min f(X) = x_1 + x_2 + x_3 \tag{7}$$

$$\begin{aligned} \text{subject to} \quad 1 - 0.0025(x_4 + x_6) &\geq 0 \\ 1 - 0.0025(x_5 + x_7 - x_4) &\geq 0 \\ 1 - 0.01(x_8 - x_5) &\geq 0 \\ x_1 x_6 - 833.33252 x_4 - 100 x_1 + 83333.333 &\geq 0 \\ x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 &\geq 0 \\ x_3 x_8 - 1250000 - x_3 x_5 + 2500 x_5 &\geq 0 \end{aligned}$$

$$\text{search space:} \quad 100 \leq x_1 \leq 10000; \quad 1000 \leq x_i \leq 10000, i = 2, 3; \quad 10 \leq x_i \leq 1000, i = 4, \ldots, 8$$

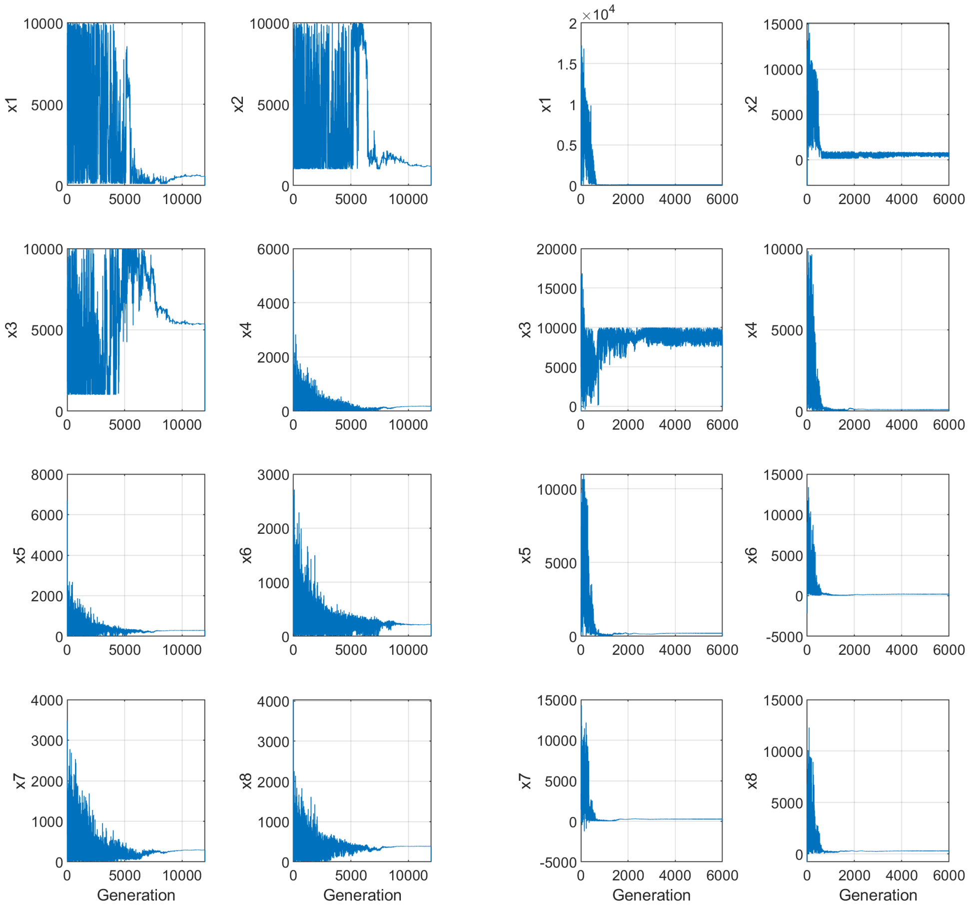
(Solution)

Table 7: Cost comparison of CEALM and PSO. Lowest cost is labelled with bold.

Run	CEALM	PSO	Analytical
1	**7107.6943**	Infeasible	7049.3310
2	7205.8859	Infeasible	
3	7122.9906	Infeasible	
4	7157.6723	Infeasible	
5	7140.7222	Infeasible	
6	7122.7595	Infeasible	
7	7145.1340	Infeasible	
8	7149.9819	Infeasible	
9	7156.5398	**10016.9597**	
10	7216.4679	Infeasible	

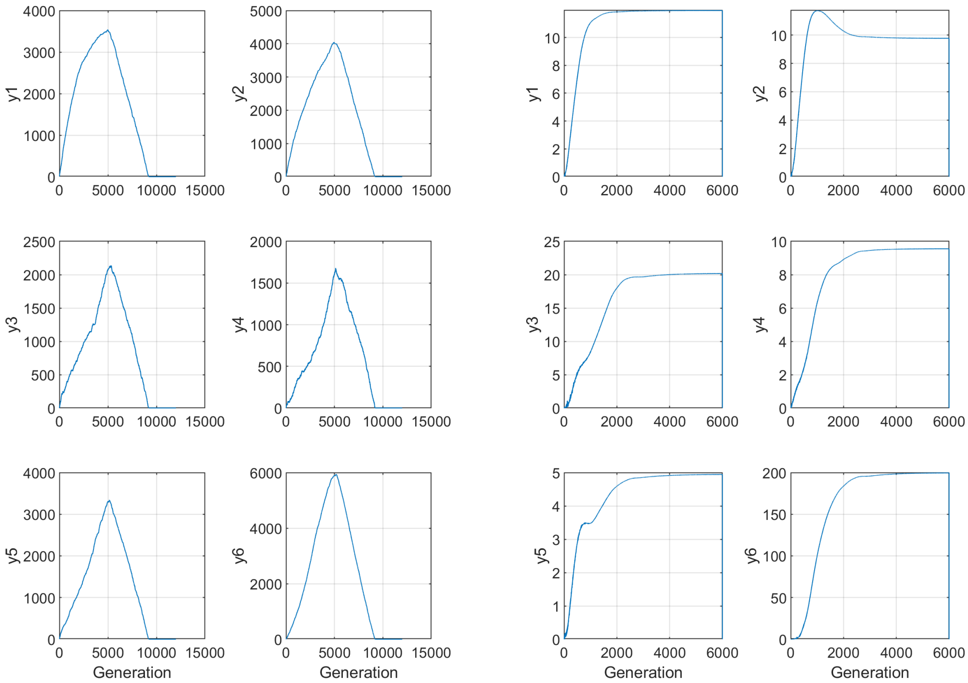
Table 8: Optimal parameter value of problem 4 with lowest cost

	CEALM	PSO		CEALM	PSO
x_1	572.9815	100.0000	y_1	0.0000	11.9362
x_2	1190.8736	836.9181	y_2	0.0002	9.7719
x_3	5343.8391	9080.0416	y_3	0.0000	20.1803
x_4	180.6481	89.9301	y_4	0.0002	9.5554
x_5	287.2824	202.5691	y_5	0.0000	4.9434
x_6	218.6001	194.7662	y_6	0.0000	199.5982
x_7	293.0581	272.1678			
x_8	387.2795	293.0828			



(a) CEALM (b) PSO

Figure 7: Parameter X of problem 4 with lowest cost



(a) CEALM (b) PSO

Figure 8: Parameter Y of problem 4 with lowest cost

Appendix

Listing 1: Main code

```
%
%
%%%  Main Program of CEALM & PSO

  clear; clc; close all;

%------ Problem to be solved --------------------
%  fprintf('\n prob_m01 \n' ) ;
  prob_m01;                                  % defines the problem to be solved1

%---------- Monte-Carlo computation -----------------------------------------------
   nrun           = 10  ;                    % no. of MC computation
   MaxGen         = 6000;                 % Maximum Generation

   nPrint = MaxGen * 10;      % print every nPrint generations

%------ Output control ------------------------------------------
  outfile = fopen ('out_v20.dat','w');


%---------- Strategy ------------------------------------------------------

   method = 'pso';
%  pso    = Particle swarm optimization
%  cealm = Co-Evolution Augmented Lagrangian Method

   istrategy = 0;
%  ONLY FOR CEALM. PSO ALWAYS USE SECURITY STRATEGY
%  0 = security strategy for both players (recommended)
%  1 = man-to-man strategy for the parameter, which is the follower
%

%---------- Populations -----------------------------------------------------

   NumOffspringX   = 40;                 % CEALM Offspring & PSO Population
   NumParentX      = 8;                  % CEALM Parent Population

   NumOffspringY   = 40;                 % CEALM Offspring & PSO Population
   NumParentY      = 8;                  % CEALM Parent Population

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

  done = false;
  for irun=1:nrun
    fprintf(outfile,'\n Run=%3d ', irun) ;
    fprintf(outfile,'\n') ;
    if strcmp(method, 'pso')
        pso_v1
    else
        cealm_v20
    end

  end
  done = true;
  graph_data;

  fclose(outfile);

%---------- end ------------
```

Listing 2: PSO method code

```
%--------------------------ParamMaxX------------------------------------------------------------
% Particle Swarm Optimization
%--------------------------------------------------------------------------------
%
%% Version 1.0
% June 8, 2020
%
% Algorithm from https://en.wikipedia.org/wiki/Particle_swarm_optimization
% Modified from cealm_v20.m
%
% Compiled by:
% Joshua Julian Damanik (20194701)
% Aerospace Enginnering
% Korea Advanced Institute of Science and Technology (KAIST)

%----------- initial bounds of Lagrange multipliers

   NumParamY = NumIneq + NumEq;

   if(NumIneq ~= 0)
       for k=1:NumIneq
          ParamMaxY(k) = 0.01;
          ParamMinY(k) = 0;
       end
   end
   if(NumEq ~= 0)
       for k=1:NumEq
          ParamMaxY(NumIneq+k) = 0.01;
          ParamMinY(NumIneq+k) = -0.01;
       end
   end

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

   Gencount     = 1 ;                          % Generation counter

   InitFlag      = 0 ;

   IntervalX      = ParamMaxX - ParamMinX;
   IntervalY      = ParamMaxY - ParamMinY;

   VelocityMinX   = -IntervalX;
   VelocityMaxX   = IntervalX;

   VelocityMinY   = -IntervalY;
   VelocityMaxY   = IntervalY;

   OffspringPopX     = zeros(NumOffspringX,NumParamX);
   OffspringPopY     = zeros(NumOffspringY,NumParamY);

   VelocityPopX   = zeros(NumOffspringX,NumParamX);
   VelocityPopY   = zeros(NumOffspringY,NumParamY);

   BestPopX       = zeros(NumOffspringX,NumParamX);
   BestPopY       = zeros(NumOffspringY,NumParamY);

   Tol            = Tolerance*ones(1,NumParamY);

   PrevF           = 1000000000000000.;
   PrevV           = 1000000000000000.;

%---------------------------------------------------------------
```

```
% Initial Population:
%    Uniform distrubution within the search space
%------------------------------------------------------

%   randn('seed',sum(100*clock));                  % Used 'seed' instead of 'state' for
%   randn('seed',sum(100*clock));                 % compatability of Matcom3 to Matlab4

  for i=1:NumOffspringX
     for j=1:NumParamX
         OffspringPopX(i,j) = rand*(ParamMaxX(j)-ParamMinX(j)) + ParamMinX(j);
         BestPopX(i,j) = OffspringPopX(i,j);
         VelocityPopX(i,j) = (rand*(VelocityMaxX(j)-VelocityMinX(j)) + VelocityMinX(j));
%          OffspringSigmaX(i,j)= VarIntervalX(j);
     end
  end

  for i=1:NumOffspringY
     for j=1:NumParamY
        OffspringPopY(i,j) = rand*(ParamMaxY(j)-ParamMinY(j)) + ParamMinY(j);
        BestPopY(i,j) = OffspringPopY(i,j);
        VelocityPopY(i,j) = (rand*(VelocityMaxY(j)-VelocityMinY(j)) + VelocityMinY(j));
%         OffspringSigmaY(i,j)= VarIntervalY(j);
     end
  end

SX = ceil(rand*NumOffspringX);
SY = ceil(rand*NumOffspringY);

BestParamX = BestPopX(SX,:);
BestParamY = BestPopY(SY,:);

SigInitX      = 0.000001*ones(1,NumParamX);
SigInitY      = 0.000001*ones(1,NumParamY);

%  SigInitX = VarIntervalX;
%  SigInitY = VarIntervalY;

SigMinX       = 0.000001*ones(1,NumParamX);
SigMinY       = 0.000001*ones(1,NumParamY);

%---------- mutation lower bounds ---------------------------------------------

   alpha   = 10^(-1/DeciGen);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% >>======= Iteration Starts Here =======<< %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

iPrint = 1;
while (Gencount <= MaxGen)

%------------------------------------------------------
% Recombination Process   (evolutionary strategy)
%------------------------------------------------------

for i=1:NumOffspringX
    RP = rand();
    RG = rand();
    for j=1:NumParamX
        VelocityPopX(i,j) = omega * VelocityPopX(i,j)...
            + phiP * RP * (BestPopX(i,j) - OffspringPopX(i,j))...
            + phiG * RG * (BestParamX(j) - OffspringPopX(i,j));
```

```
%          VelocityPopX(i,j) = min(max(VelocityPopX(i,j),VelocityMinX(j)),VelocityMaxX(j));
        OffspringPopX(i,j) = OffspringPopX(i,j) + VelocityPopX(i,j);
    end
end

for i=1:NumOffspringY
    RP = rand();
    RG = rand();
    for j=1:NumParamY
        VelocityPopY(i,j) = omega * VelocityPopY(i,j)...
            + phiP * RP * (BestPopY(i,j) - OffspringPopY(i,j))...
            + phiG * RG * (BestParamY(j) - OffspringPopY(i,j));
%          VelocityPopY(i,j) = min(max(VelocityPopY(i,j),VelocityMinY(j)),VelocityMaxY(j));
        OffspringPopY(i,j) = OffspringPopY(i,j) + VelocityPopY(i,j);
    end
end

%---------------------------------------------
%  search space restriction (softwalls)
%---------------------------------------------

for i=1:NumOffspringX
   for j=1:NumParamX
      violate1 = OffspringPopX(i,j)-ParamMaxX(j);
      violate2 = ParamMinX(j)-OffspringPopX(i,j);
      if violate1 >0
         OffspringPopX(i,j) = OffspringPopX(j) - rand*min(violate1,IntervalX(j)) ;
      end
      if violate2 >0
         OffspringPopX(i,j) = OffspringPopX(j) + rand*min(violate2,IntervalX(j)) ;
      end
   end
end

%-------------- lower bound check only for inequality constraints --------------

for i=1:NumOffspringY
   for j=1:NumIneq
      violate1 = OffspringPopY(i,j)-ParamMaxY(j);
      violate2 = ParamMinY(j)-OffspringPopY(i,j);
%        if violate1 >0
%           OffspringPopY(i,j) = OffspringPopY(j) - rand*min(violate1,IntervalY(j)) ;
%        end
      if violate2 >0
         OffspringPopY(i,j) = OffspringPopY(j)+rand*min(violate2,IntervalY(j)) ;
      end
   end
end



%---------------------------------------------
% Matching Process (full match)
%---------------------------------------------


for i=1:NumOffspringX

  [costf,cnstr] = feval(CostDef,OffspringPopX(i,:));

  for j=1:NumOffspringY
    AL1 = 0;
    AL2 = 0;
```

```
    AL3 = 0;
    if(NumIneq ~= 0)
        for k=1:NumIneq
            AL1 = AL1 + (max(cnstr(k)+0.5*OffspringPopY(j,k)/rho, 0))^2;
            AL2 = AL2 + OffspringPopY(j,k)^2;
        end
        AL1 = rho*AL1;
        AL2 = - 0.25*AL2/rho;
    end
    if(NumEq ~= 0)
        for k=1:NumEq
            AL3 = AL3 + OffspringPopY(j,NumIneq+k)*cnstr(NumIneq+k) + rho*cnstr(NumIneq+k)^2
        end
    end
    ALV = costf + AL1 + AL2 + AL3;
    CostOffspringX(i,j) = ALV;
    CostOffspringY(j,i) = ALV;
  end
end



%-----------------------------------------------------------------
%  Fitness evaluation and ordering
%-----------------------------------------------------------------



if(istrategy==0)     % Y & X = security
    [maxcost,mxcind] = max(CostOffspringX');
    [mincost,mncind] = min(CostOffspringY');
end

if exist('CostBestX','var') == 0
    BestPopX = OffspringPopX;
    BestPopY = OffspringPopY;

    BestCostPopX = maxcost;
    BestCostPopY = mincost;

    [BestCostX,bestindexx] = min(BestCostPopX);
    [BestCostY,bestindexy] = max(BestCostPopY);

    BestParamX = BestPopX(bestindexx,:);
    BestParamY = BestPopY(bestindexy,:);
end

for i=NumOffspringX
    if (maxcost(i) < BestCostPopX(i))
        BestPopX(i,:) = OffspringPopX(i,:);
        BestCostPopX(i) = maxcost(i);
    end
    if BestCostPopX(i) < BestCostX
        BestParamX = BestPopX(i,:);
        BestCostX = BestCostPopX(i);
    end
end

% [BestCostY, bestindexy] = max(CostOffspringY(:,bestindexx));
% BestParamY = OffspringPopY(bestindexy,:);

for i=NumOffspringY
```

```
    if (mincost(i) > BestCostPopY(i))
        BestPopY(i,:) = OffspringPopY(i,:);
        BestCostPopY(i) = mincost(i);
    end
    if BestCostPopY(i) > BestCostY
        BestParamY = BestPopY(i,:);
        BestCostY = BestCostPopY(i);
    end
end

%---------------------------------------------------
% Output to Display & File
%---------------------------------------------------


  [CostF,Cnstr] = feval(CostDef,BestParamX);

  if(NumEq ~= 0)
     for k=1:NumEq
        Cnstr(NumIneq+k)=abs(Cnstr(NumIneq+k));
     end
  end
  Vx   = max(Cnstr,0);
  Vsum = sum(Vx);

  C1 = Cnstr - Tol;
  V1    = max(C1,0);
  CV   = sum(V1);


  if CV <= 0
    CnstrFlag = 1;
  else
    CnstrFlag = 0;
  end

  if InitFlag == 0  & CnstrFlag ==1              % set BestF if constraint is satisfied
    InitFlag = 1;                                % for the first time
    PrevF = CostF;
    PrevV = Vsum;
    PrevPX = BestParamX;
    PrevPY = BestParamY;
  end

  if InitFlag ==1 & CnstrFlag ==1  & CostF < PrevF      % check if cost improved
    ImprvFlag = 1;
  else
    ImprvFlag=0;
  end

  if ImprvFlag == 1
     PrevF = CostF;
     PrevV = Vsum;
     PrevPX = BestParamX;
     PrevPY = BestParamY;
  end

%------  Write to the output file if improved ------------------

  if ImprvFlag == 1 || (InitFlag == 0  & CnstrFlag ==1 )
    fprintf(outfile, '\n Run=%3d   G=%3d', irun, Gencount) ;
    fprintf(outfile, '\n CostF =%10.7f', CostF) ;
    fprintf(outfile,'\n');
```

```
    if NumIneq ~= 0
        for i=1:NumIneq
            fprintf(outfile,' G%d=%10.7f',i, Cnstr(i));
        end
        fprintf(outfile,'\n');
    end
    if NumEq ~= 0
       for i=1:NumEq
           fprintf(outfile,' H%d=%10.7f',i, Cnstr(i+NumIneq));
       end
       fprintf(outfile,'\n');
    end
    for i=1:NumParamX
      fprintf(outfile,' X%d=%10.7f',i, BestParamX(i));
    end
    fprintf(outfile,'\n');
    for i=1:NumParamY
      fprintf(outfile,' Y%d=%10.7f',i, BestParamY(i));
    end
    fprintf(outfile,'\n');
  end

%--------- Print at every nPrint generation ------------------------------------------

   if iPrint ==  nPrint
     fprintf('\n Run=%3d G=%3d ', irun, Gencount);
     fprintf('\n CV     =%10.7f',CV);
     for i=1:NumParamY
       fprintf('   G%d=%10.7f',i, Cnstr(i));
     end
     fprintf('\n CostX =%10.7f', BestCostX) ;
     for i=1:NumParamX
       fprintf('   X%d=%10.7f',i, BestParamX(i));
     end
     fprintf('\n CostY =%10.7f', BestCostY) ;
     for i=1:NumParamY
       fprintf('   Y%d=%10.7f',i, BestParamY(i));
     end
     fprintf('\n');
     iPrint = 0;
   end
   iPrint = iPrint + 1;


%---------Screen output at the end of evolution --------------

  if(Gencount >= MaxGen)
     fprintf('\n Evolution Summary: Run=%3d G=%3d ', irun, Gencount);

    if(InitFlag == 1)
     else
        fprintf('   Unable to find a feasible solution!');
     end

     fprintf('\n CostF=%10.7f Vsum=%10.7f AL_X=%10.7f AL_Y=%10.7f', CostF, Vsum, BestCostX,
     fprintf('\n');
     for i=1:NumParamX
       fprintf('   X%d=%10.7f',i, BestParamX(i));
     end
     fprintf('\n');
     for i=1:NumParamY
       fprintf('   Y%d=%10.7f',i, BestParamY(i));
     end
```

```
      fprintf('\n');

   end
%  check_transient;
 graph_data;
%  if done == true, break, end
 Gencount = Gencount + 1 ;



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

end          %%%%%   End of 'While' Loop   %%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

References

[1] M.-J. Tahk and B.-C. Sun, "Coevolutionary augmented lagrangian methods for constrained optimization," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 2, p. 114–124, 2000.

[2] M. Zambrano-Bigiarini, M. Clerc, and R. Rojas, "Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements," *2013 IEEE Congress on Evolutionary Computation*, 2013.