APPLICATION OF PARTICLE SWARM OPTIMIZER TO ECONOMIC LOAD DISPATCH PROBLEMS

A Project Report Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology in Electrical Engineering

By

ANUPAMA LAKRA
(Roll No. – 10502027)

ROSALIN DAS
(Roll No. – 10502032)

DEPARTMENT OF
ELECTRICAL ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA

APPLICATION OF PARTICLE SWARM OPTIMIZER TO ECONOMIC LOAD DISPATCH PROBLEMS

A Project Report Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology in Electrical Engineering

By

Anupama Lakra
(Roll No. – 10502027)

Rosalin Das
(Roll No. – 10502032)

Under the guidance of

Prof. S. Rauta

DEPARTMENT OF
ELECTRICAL ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that the project entitled, "**APPLICATION OF PARTICLE SWARM OPTIMIZER TO ECONOMIC LOAD DISPATCH PROBLEMS**" submitted by **Ms. Anupama Lakra** and **Ms. Rosalin Das** is an authentic work carried out by both under my supervision and guidance for the partial fulfillment of the requirements for the award of **Bachelor of Technology Degree** in **Electrical Engineering** at **National Institute of Technology, Rourkela (Deemed University).**

To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Date: 12/05/2009 **(Prof. S.Rauta)**

Rourkela **Dept. of Electrical Engineering,**
National Institute of Technology,
Rourkela,
Orissa-769008

ACKNOWLEDGEMENT

We express our sincere gratitude to our project guide Prof. S.Rauta for his invaluable guidance and support. We are grateful for him for allowing us to do this project with his constant help and support.

Our profound and sincere thanks to Prof. P.C.Panda for his help and support. We also thank our parents and friends for their invariable support. Above all we thank the God Almighty for giving us the will and determination for doing our project.

ANUPAMA LAKRA

Roll No.- 10502027

Date: 12/05/2009

ROSALIN DAS

Roll No.- 10502032

ABSTRACT

This project presents an efficient and reliable Particle Swarm Optimization (PSO) method for the Economic load dispatch (ELD) problems which is considered as one of the complex problems to be tackled. The PSO techniques have drawn much attention from the power system community and been successfully applied in many complex optimization problems in power systems. The PSO method was developed through the simulation of a simplified social system and has been found to be robust in solving continuous nonlinear optimization problems in terms of accuracy of the solution and computation time and it can out perform other algorithms.

In this project, the proposed algorithm is applied for the ELD of three unit thermal plant systems.

CONTENTS

NOMENCLATURE

F_t - Total fuel cost

N - Number of online generators committed to the operating system.

P_i - Power output of ith generator.

F_i (P_i) - Fuel cost characteristics of the ith generator.

P_D - Total power demand

P_L - Total transmission loss

B_{mn} - Co-efficient of Transmission loss formula.

$P_{i,max}$ - Maximum generation capacity of the ith generator.

$P_{i,min}$ - Minimum generation capacity of the ith generator.

n - Number of particles in a group

m - Number of members in a particle

t - Pointer of iterations (generations)

ω - Inertia weight factor

C_1, C_2 - Acceleration constant

rand ( ), Rand ( ) - Uniform random value in the range [0, 1]

$V_{id}^{(t)}$ - velocity of particle i at iteration 't', $V_d^{min} \leq V_{id}^{(t)} \leq V_d^{max}$

$X_{id}^{(t)}$ - current position of particle i at iteration 't'

A_i, B_i, C_i - Fuel cost coefficients

INTRODUCTION

The Economic Load Dispatch (ELD) problem is one of the fundamental issues in power system operation. The ELD problem involves the solution of two different problems. The first of these is the Unit Commitment or predispatch problem wherein it is required to select optimally out of the available generating sources to operate, to meet the expected load and provide a specified margin of operating reserve over a specified period of time. The second aspect of economic dispatch is the on-line economic dispatch wherein it is required to distribute the load among the generating units actually paralleled with the system in such manner as to minimize the total cost of supplying the minute-to-minute requirements of the system. The main objective is to reduce the cost of energy production taking into account the transmission losses. While the problem can be solved easily if the incremental cost curves of the generators are assumed to be monotonically increasing piece-wise linear functions, such an approach will not be workable for nonlinear functions in practical systems. In the past decade, conventional optimization techniques such as lambda iterative method, linear programming and quadratic programming have been successfully used to solve power system optimization problems such as Unit commitment and Economic load dispatch. For highly non-linear and combinatorial optimization problems, the conventional methods are facing difficulties to locate the global optimal solution. Recently there is an upsurge in the use of modern evolutionary computing techniques in the field of power system optimization. Particle Swarm Optimization (PSO), first introduced by Kennedy and Eberhart, is one of the modern heuristic algorithms. It was developed through simulation of a simplified social system,

and has been found to be robust in solving continuous non-linear optimization problems. The PSO technique can generate high-quality solutions within shorter calculation time and stable convergence characteristics than other stochastic methods like Genetic Algorithm (GA). Unlike in GA method, in PSO the selection operation is not performed. All the particles in PSO are kept as members of the population through the course of a run (a run is defined as the total number of generation of the evolutionary algorithms prior to termination). It is the velocity of the particle which is updated according to its previous best position of its companions. The particles fly with the updated velocities. This paper proposes the application of PSO method for solving the economic load dispatch problems.

FORMULATION OF ELD PROBLEM

The Economic Load Dispatch (ELD) is generating adequate electricity to meet the continuously varying consumer load demand at the least possible cost under a number of constraints. Practically, while the scheduled combination of units at each specific period of operation are listed, the ELD planning must perform the optimal generation dispatch among the operating units to satisfy the load demand, spinning reserve capacity, and practical operation constraints of generators.

The objective of the ELD problem is to minimize the total fuel cost. Mathematically it can be represented as

Minimize $F_t = \sum_{i=1}^{n} F_i(P_i)$

where $F_T = \sum_{i=1}^{n} A_i P_i^2 + B_i P_i + C_i$

The ELD problem is subjected to the following constraints,

The power balance equation,

$$\sum_{i=1}^{n_g} P_i = P_D + P_L$$

The total Transmission loss,

$$P_L = \sum\sum P_m B_{mn} P_n$$

In addition, power output of each generator has to fall within the operation limits of the generators as shown below,

$$P_{gi}^{min} \leq P_{gi} \leq P_{gi}^{max} \quad \text{, for } i=1,2\ldots\ldots n.$$

ECONOMIC LOAD DISPATCH INCLUDING LOSSES:

When transmission distances are very small and load density is very high, transmission losses may be neglected and the optimal dispatch of generation is achieved with all plants operating at equal incremental production cost. However in a large interconnected network where power is transmitted over long distances with low load density areas, transmission losses are a major factor and affect the optimum dispatch of generation. One common practice for including the effect of transmission losses is to express the total transmission losses as a quadratic function of the generator power outputs.

The simplest quadratic form is:

$$P_L = \sum_{i=1}^{n_g} \sum_{j=1}^{n_g} P_i B_{ij} P_j$$

A more general formula containing a linear term and a constant term , referred to as Kron's loss formula, is

$$P_L = \sum_{i=1}^{n_g} \sum_{j=1}^{n_g} P_i B_{ij} P_j + \sum_{i=1}^{n_g} B_{0i} P_i + B_{00}$$

The coefficients B_{ij} are called loss coefficients or B-coefficients. These are considered constants and reasonable accuracy can be expected provided the actual operating conditions are close to the base case where the B constants were computed. There are various ways of arriving at a loss equation.

The economic dispatching problem is to minimize the overall generating cost F_T , which is the function of plant output

$$F_T = \sum_{i=1}^{n_g} F_i = \sum_{i=1}^{n_g} C_i + B_i P_i + A_i P_i^2$$

subject to the constraint that generation should equal total demands plus losses, i.e.,

$$\sum_{i=1}^{n_g} P_i = P_D + P_L$$

satisfying the inequality constraints , expressed as follows:

$$P_{gi}^{min} \leq P_{gi} \leq P_{gi}^{max} \quad \text{, for i=1,2…….n.}$$

The B-coefficients obtained are based on the generation in per unit. When generation are expressed in MW , the loss coefficients are

$B_{ij} = B_{ij\,pu} / S_B$, $B_{0i} = B_{0i\,pu}$, and $B_{00} = B_{00\,pu} * S_B$

Where S_B is the base MVA.

IMPLEMENTATION OF PSO FOR ELD PROBLEMS

A. OVERVIEW OF PARTICLE SWARM OPTIMIZER

Kennedy and Eberhart developed a particle swarm optimization **(PSO)** algorithm based on the behavior of individuals (i.e., particles or agents) of a swarm. Its roots are in zoologist's modeling of the movement of individuals (i.e., fishes, birds, insects) within a group. It has been noticed that members of the group seem to share information among them, a fact that leads to increased efficiency of the group**.** PSO, as an optimization tool, provides a population-based search procedure in which individuals called particles change their position (states) with time. In a PSO system particles fly around in a multi-dimensional search space.

During flight, each particle adjusts its position according to its own experience and the experience of neighboring particles, making use of the best position encountered by it and neighbors. The swarm direction of a particle is defined by the set of particles neighboring the particle and its history experience. Instead of using evolutionary operation to manipulate the individuals, like in other evolutionary computational algorithms, each individual in PSO flies in the search space with a velocity which is dynamically adjusted according to its own flying experience and its companions flying experience.

Let x and v denote a particle co-ordinate (position) and its corresponding flight speed (velocity) in a search space respectively. Therefore, each i^{th} particle is treated as a volume less particle, represented as $xi= (x_{i1}, x_{i2} \ldots x_{id})$ in the d - dimensional space. The best

previous position of the ith particle is recorded and represented as $pbest_i$= ($pbest_{i1}$, $pbest_{i2}$,…….. $pbest_{id}$).The index of the best particle among all

the particles is treated as global best particle, is represented as $gbest_d$. The rate of velocity for particle 'i' is represented as v_i= (v_{i1}, v_{i2}……...v_{id}).The modified velocity and position of each particle can be calculated using the current velocity and the distance from $pbest_{id}$ to $gbest_d$ as shown in the following formulas,

$$V_{id}^{(t+1)} = \omega V_i^{(t)} + C_1 \, rand(\,)\,(\,pbest_{id} - P_{gid}^{(t)}\,) + C_2 \, Rand(\,)\,(\,gbest_{id} - P_{gid}^{(t)}\,)$$

$$P_{gid}^{(t+1)} = P_{gid}^{(t)} + V_{id}^{(t+1)}$$

In the above equation, C1 has a range (1.5, 2), which is called self-confidence range; C2 has a range (2, 2.5), which is called swarm range.

The term rand ( )*($pbest_{id}$-$P_{gid}^{(t)}$) is called particle memory influence The term Rand ( )*($gbest_d$-$P_{gid}^{(t)}$ is called swarm influence.

$V_i^{(t)}$ which is the velocity of i^{th} particle at iteration't' must lie in the range $V_d^{min} \le v_{id}^{(t)} \le V_d^{max}$. The parameter V_d^{max} determines the resolution, or fitness, with which regions are to be searched between the present position and the target position .If V_d^{max} is too high, particles may fly past good solutions. If V_d^{max} is too small, particles may not explore sufficiently beyond local solutions.

The constants C1and C2 pull each particle towards pbest and *gbest* positions. Low values allow particles to roam far from the target regions before being tugged back. On the other hand, high values result in abrupt movement towards, or past, target regions. Hence, the acceleration constants C1 and C2 are often set to be 2.0 according to past

experiences. Suitable selection of inertia weight ' ω ' provides a balance between global and local explorations, thus requiring less iteration on

average to find a sufficiently optimal solution. As originally developed, ω often decreases linearly from about 0.9 to 0.4 during a run. In general, the inertia weight *w* is set according to the following equation,

$\omega = \omega_{max} - [(\omega_{max} - \omega_{min}) / iter_{max}] * iter$

where ω - inertia weight factor

ω_{max} - maximum value of weighting factor

ω_{min} - minimum value of weighting factor

$iter_{max}$ - maximum number of iterations

iter - current number of iteration

Each individual moves from the current position to the next one by the modified velocity using the following equation:

$$P_{gid}^{(t+1)} = P_{gid}^{(t)} + V_{id}^{(t+1)}$$

The search mechanism of PSO using the modified velocity and position of individual ***i*** is illustrated in fig.

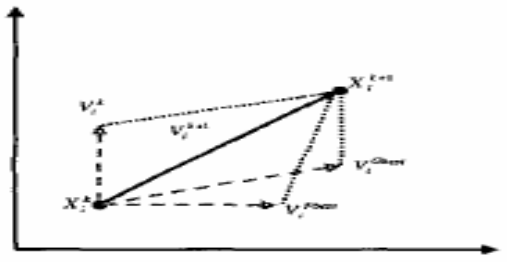

Fig. 2. The searching mechanism of the particle swarm optimization.

B. APPLICATION OF PSO METHOD TO ECONOMIC LOAD DISPATCH

An algorithm is used to solve a constrained ELD problem using PSO was developed to obtain a high quality solution. The PSO algorithm was utilized mainly to determine the optimal allocation of power among the units, which were scheduled to operate at the specific period, thus minimizing the total generation cost.

CALCULATION PROCESS OF THE PROPOSED METHOD

This paper presents a quick solution to the constrained ELD problem using the PSO algorithm to search optimal or near optimal generation of each unit. The sequential steps of the proposed PSO method are given below.

Step 1: Initialize randomly the individuals of the population according to the limit of each unit including individual dimensions, searching points, and velocities. These initial individuals must be feasible candidate solutions that satisfy the practical operation constraints.

Step 2**:** To each chromosome of the population the dependent unit output Pd will be calculated from the power balance equation and Bmn coefficient matrix.

Step 3: Calculate the evaluation value of each individual Pgi, in the population using the evaluation function f given by:

$$\text{Minimize } F_T = \sum_{i=1}^{n} F_i(P_i)$$

Step 4: Compare each individual's evaluation value with its *pbest.* The best evaluation value among the pbests is denoted as *gbest.*

Step 5: Modify the member velocity *v* of each individual P_g,

$V_{id}^{(t+1)} = \omega V_i^{(t)} + C_1 \, rand( ) \, (pbest_{id} - P_{gid}^{(t)}) + C_2 \, Rand( ) \, (gbest_{id} - P_{gid}^{(t)})$

Where i=1, 2 ….n. d=1, 2… m

Step 6: Check the velocity components constraint occurring in the limits from the following conditions,

$V_{id}^{(t+1)} > V_d^{max}$, then $V_{id}^{(t+1)} = V_d^{max}$,

$V_{id}^{(t+1)} > V_d^{min}$, then $V_{id}^{(t+1)} = V_d^{min}$,

Where

$V_i^{min} = -0.5 \, P_g^{min}$ and

$V_i^{max} = +0.5 \, P_g^{max}$

Step 7: Modify the member position of each individual P_g

$P_{gid}^{(t+1)} = P_{gid}^{(t)} + V_{id}^{(t+1)}$

$P_{gid}^{(t+1)}$ must be modified toward the near margin of the feasible solution.

Step 8: If the evaluation value of each individual is better than previous *pbest,* the current value is set to be *pbest*. If the best *pbest* is better than *gbest*, the value is set to be *gbest*.

Step 9: If the number of iterations reaches the maximum, then go to step 10.Otherwise, go to step 2.

Step 10: The individual that generates the latest *gbest* is the optimal generation power of each unit with the minimum total generation cost.

EXAMPLE PROBLEM AND SIMULATION RESULTS:

To verify the feasibility of the PSO method, three thermal plants of a power system are tested. A reasonable B_{mn} loss coefficients matrix of power system network is employed. The program is developed in MATLAB and executed.

EXAMPLE : THREE-UNIT THERMAL SYSTEM:

The fuel cost in $/h of three thermal plants of a power system are given as

$C_1 = 200 + 7.0P_1 + 0.008P_1^2$ $/h

$C_2 = 180 + 6.3P_2 + 0.009P_2^2$ $/h

$C_3 = 140 + 6.8P_3 + 0.007P_3^2$ $/h

The unit operating ranges are:

$10 \text{ MW} \leq P_1 \leq 85 \text{ MW}$

$10 \text{ MW} \leq P_2 \leq 80 \text{ MW}$

$10 \text{ MW} \leq P_3 \leq 70 \text{ MW}$

For this problem, assume the real power loss is given by the simplified expression

$$P_{L(pu)} = 0.0218P_1^{2}{}_{(pu)} + 0.0228P_2^{2}{}_{(pu)} + 0.0179P_3^{2}{}_{(pu)}$$

The B matrices of the loss formula for this system are given in per unit on a 100 MVA base as follows:

$$B = \begin{bmatrix} 0.0218 & 0.0093 & 0.0028 \\ 0.0093 & 0.0228 & 0.0017 \\ 0.0028 & 0.0017 & 0.0179 \end{bmatrix}$$

$$B_0 = \begin{bmatrix} 0.0003 & 0.0031 & 0.0015 \end{bmatrix}$$

$B_{00} = 0.00030523$

For the system load of 150 MW, the PSO method is applied.

For finding local best(pbest) and global best(gbest)

Particle swarm optimization toolbox for matlab developed by Prof. Brian Birge implementing Common, Clerc 1", and Trelea types along with an alpha version of tracking changing environments. It can search for min, max, or 'distance' of user developed cost function. It uses similar syntax to Matlab's optimization toolbox.

% Usage:

% [optOUT]=PSO(functname,D)

% or:

```
%  [optOUT,tr,te]=...


%PSO(functname,D,mv,VarRange,minmax,PSOparams,plotfcn,PSOseedValue)
% Inputs:
%    functname - string of matlab function to optimize
%    D - # of inputs to the function (dimension of problem)
%
% Optional Inputs:
%    mv - max particle velocity, either a scalar or a vector of length D
%          (this allows each component to have it's own max velocity),
%          default = 4, set if not input or input as NaN
%
%    VarRange - matrix of ranges for each input variable,
%     default -100 to 100, of form:
%      [ min1 max1
%        min2 max2
%          ...
%        minD maxD ]
%
%    minmax = 0, funct minimized (default)
%          = 1, funct maximized
%          = 2, funct is targeted to P(12) (minimizes distance to errgoal)
%    PSOparams - PSO parameters
```

```
%     P(1) - Epochs between updating display, default = 100. if 0,
%          no display
%     P(2) - Maximum number of iterations (epochs) to train, default = 2000.
%     P(3) - population size, default = 24
%
%     P(4) - acceleration const 1 (local best influence), default = 2
%     P(5) - acceleration const 2 (global best influence), default = 2
%     P(6) - Initial inertia weight, default = 0.9
%     P(7) - Final inertia weight, default = 0.4
%     P(8) - Epoch when inertial weight at final value, default = 1500
%     P(9)- minimum global error gradient,
%               if abs(Gbest(i+1)-Gbest(i)) < gradient over
%               certain length of epochs, terminate run, default = 1e-25
%     P(10)- epochs before error gradient criterion terminates run,
%               default = 150, if the SSE does not change over 250 epochs
%                         then exit
%     P(11)- error goal, if NaN then unconstrained min or max, default=NaN
%     P(12)- type flag (which kind of PSO to use)
%               0 = Common PSO w/intertia (default)
%               1,2 = Trelea types 1,2
%               3   = Clerc's Constricted PSO, Type 1"
%     P(13)- PSOseed, default=0
%            = 0 for initial positions all random
```

```
%             = 1 for initial particles as user input
%
%    plotfcn - optional name of plotting function, default 'goplotpso',
%             make your own and put here
%
%    PSOseedValue - initial particle position, depends on P(13), must be
%                   set if P(13) is 1 or 2, not used for P(13)=0, needs to
%                   be nXm where n<=ps, and m<=D
%                   If n<ps and/or m<D then remaining values are set random
%                   on Varrange
% Outputs:
%    optOUT - optimal inputs and associated min/max output of function, of form:
%        [ bestin1
%          bestin2
%            ...
%          bestinD
%          bestOUT ]
%
% Optional Outputs:
%    tr    - Gbest at every iteration, traces flight of swarm
%    te    - epochs to train, returned as a vector 1:endepoch
```

SOURCE CODE:

```
function [OUT,varargout]=pso_Trelea_vectorized(functname,D,varargin)


rand('state',sum(100*clock));
if nargin < 2
   error('Not enough arguments.');
end


% PSO PARAMETERS
if nargin == 2      % only specified functname and D
   VRmin=ones(D,1)*-100;
   VRmax=ones(D,1)*100;
   VR=[VRmin,VRmax];
   minmax = 0;
   P = [];
   mv = 4;
   plotfcn='goplotpso';
elseif nargin == 3  % specified functname, D, and mv
   VRmin=ones(D,1)*-100;
   VRmax=ones(D,1)*100;
   VR=[VRmin,VRmax];
   minmax = 0;
```

```
    mv=varargin{1};
    if isnan(mv)
        mv=4;
    end
    P = [];
    plotfcn='goplotpso';
elseif nargin == 4  % specified functname, D, mv, Varrange
    mv=varargin{1};
    if isnan(mv)
        mv=4;
    end
    VR=varargin{2};
    minmax = 0;
    P = [];
    plotfcn='goplotpso';
elseif nargin == 5  % Functname, D, mv, Varrange, and minmax
    mv=varargin{1};
    if isnan(mv)
        mv=4;
    end
    VR=varargin{2};
    minmax=varargin{3};
    P = [];
```

```
   plotfcn='goplotpso';
elseif nargin == 6  % Functname, D, mv, Varrange, minmax, and psoparams
   mv=varargin{1};
   if isnan(mv)
      mv=4;
   end
   VR=varargin{2};
   minmax=varargin{3};
   P = varargin{4}; % psoparams
   plotfcn='goplotpso';
elseif nargin == 7  % Functname, D, mv, Varrange, minmax, and psoparams, plotfcn
   mv=varargin{1};
   if isnan(mv)
      mv=4;
   end
   VR=varargin{2};
   minmax=varargin{3};
   P = varargin{4}; % psoparams
   plotfcn = varargin{5};
elseif nargin == 8  % Functname, D, mv, Varrange, minmax, and psoparams, plotfcn,
PSOseedValue
   mv=varargin{1};
   if isnan(mv)
```

```
      mv=4;
  end
  VR=varargin{2};
  minmax=varargin{3};
  P = varargin{4}; % psoparams
  plotfcn = varargin{5};
  PSOseedValue = varargin{6};
else
  error('Wrong # of input arguments.');
end

% sets up default pso params
Pdef = [100 2000 24 2 2 0.9 0.4 1500 1e-25 250 NaN 0 0];
Plen = length(P);
P    = [P,Pdef(Plen+1:end)];

df      = P(1);
me      = P(2);
ps      = P(3);
ac1     = P(4);
ac2     = P(5);
iw1     = P(6);
iw2     = P(7);
```

```
iwe     = P(8);
ergrd   = P(9);
ergrdep = P(10);
errgoal = P(11);
trelea  = P(12);
PSOseed = P(13);

% used with trainpso, for neural net training
if strcmp(functname,'pso_neteval')
   net = evalin('caller','net');
    Pd = evalin('caller','Pd');
    Tl = evalin('caller','Tl');
    Ai = evalin('caller','Ai');
     Q = evalin('caller','Q');
    TS = evalin('caller','TS');
end


% error checking
 if ((minmax==2) & isnan(errgoal))
     error('minmax= 2, errgoal= NaN: choose an error goal or set minmax to 0 or 1');
 end
```

```
if ( (PSOseed==1) & ~exist('PSOseedValue') )
    error('PSOseed flag set but no PSOseedValue was input');
end

if exist('PSOseedValue')
    tmpsz=size(PSOseedValue);
    if D < tmpsz(2)
        error('PSOseedValue column size must be D or less');
    end
    if ps < tmpsz(1)
        error('PSOseedValue row length must be # of particles or less');
    end
end

% set plotting flag
if (P(1))~=0
  plotflg=1;
else
  plotflg=0;
end

% preallocate variables for speed up
 tr = ones(1,me)*NaN;
```

```
% take care of setting max velocity and position params here
if length(mv)==1
 velmaskmin = -mv*ones(ps,D);     % min vel, psXD matrix
 velmaskmax = mv*ones(ps,D);      % max vel
elseif length(mv)==D
 velmaskmin = repmat(forcerow(-mv),ps,1); % min vel
 velmaskmax = repmat(forcerow( mv),ps,1); % max vel
else
 error('Max vel must be either a scalar or same length as prob dimension D');
end
posmaskmin  = repmat(VR(1:D,1)',ps,1);  % min pos, psXD matrix
posmaskmax  = repmat(VR(1:D,2)',ps,1);  % max pos
posmaskmeth = 3; % 3=bounce method (see comments below inside epoch loop)

% PLOTTING
 message = sprintf('PSO: %%g/%g iterations, GBest = %%20.20g.\n',me);

% INITIALIZE INITIALIZE INITIALIZE INITIALIZE INITIALIZE INITIALIZE

% initialize population of particles and their velocities at time zero,
% format of pos= (particle#, dimension)
 % construct random population positions bounded by VR
```

```
 pos(1:ps,1:D) = normmat(rand([ps,D]),VR',1);


 if PSOseed == 1         % initial positions user input, see comments above
   tmpsz                    = size(PSOseedValue);
   pos(1:tmpsz(1),1:tmpsz(2)) = PSOseedValue;
 end


 % construct initial random velocities between -mv,mv
  vel(1:ps,1:D) = normmat(rand([ps,D]),...
      [forcecol(-mv),forcecol(mv)]',1);


% initial pbest positions vals
 pbest = pos;


% VECTORIZE THIS, or at least vectorize cost funct call
 out = feval(functname,pos);  % returns column of cost values (1 for each particle)
%---------------------------


 pbestval=out;   % initially, pbest is same as pos


% assign initial gbest here also (gbest and gbestval)
 if minmax==1
   % this picks gbestval when we want to maximize the function
```

```
   [gbestval,idx1] = max(pbestval);
elseif minmax==0
  % this works for straight minimization
   [gbestval,idx1] = min(pbestval);
elseif minmax==2
  % this works when you know target but not direction you need to go
  % good for a cost function that returns distance to target that can be either
  % negative or positive (direction info)
   [temp,idx1] = min((pbestval-ones(size(pbestval))*errgoal).^2);
   gbestval    = pbestval(idx1);
end

% preallocate a variable to keep track of gbest for all iters
bestpos        = zeros(me,D+1)*NaN;
gbest          = pbest(idx1,:);  % this is gbest position
  % used with trainpso, for neural net training
  % assign gbest to net at each iteration, these interim assignments
  % are for plotting mostly
   if strcmp(functname,'pso_neteval')
      net=setx(net,gbest);
   end
%tr(1)          = gbestval;       % save for output
bestpos(1,1:D) = gbest;
```

```
% this part used for implementing Carlisle and Dozier's APSO idea
% slightly modified, this tracks the global best as the sentry whereas
% their's chooses a different point to act as sentry
% see "Tracking Changing Extremea with Adaptive Particle Swarm Optimizer",
% part of the WAC 2002 Proceedings, June 9-13, http://wacong.com
 sentryval = gbestval;
 sentry    = gbest;

if (trelea == 3)
% calculate Clerc's constriction coefficient chi to use in his form
 kappa   = 1; % standard val = 1, change for more or less constriction
 if ( (ac1+ac2) <=4 )
    chi = kappa;
 else
    psi     = ac1 + ac2;
    chi_den = abs(2-psi-sqrt(psi^2 - 4*psi));
    chi_num = 2*kappa;
    chi     = chi_num/chi_den;
 end
end

% INITIALIZE END INITIALIZE END INITIALIZE END INITIALIZE END
```

```
rstflg = 0; % for dynamic environment checking
% start PSO iterative procedures
 cnt    = 0; % counter used for updating display according to df in the options
 cnt2   = 0; % counter used for the stopping subroutine based on error convergence
 iwt(1) = iw1;
for i=1:me  % start epoch loop (iterations)

    out        = feval(functname,[pos;gbest]);
    outbestval = out(end,:);
    out        = out(1:end-1,:);

    tr(i+1)          = gbestval; % keep track of global best val
    te               = i; % returns epoch number to calling program when done
    bestpos(i,1:D+1) = [gbest,gbestval];

    %assignin('base','bestpos',bestpos(i,1:D+1));
  %-----------------------------------------------------------------------
  % this section does the plots during iterations
   if plotflg==1
     if (rem(i,df) == 0 ) | (i==me) | (i==1)
        fprintf(message,i,gbestval);
        cnt = cnt+1; % count how many times we display (useful for movies)
```

```
        eval(plotfcn); % defined at top of script


      end  % end update display every df if statement
    end % end plotflg if statement


    % check for an error space that changes wrt time/iter
    % threshold value that determines dynamic environment
    % sees if the value of gbest changes more than some threshold value
    % for the same location
    chkdyn = 1;
    rstflg = 0; % for dynamic environment checking


    if chkdyn==1
     threshld = 0.05;  % percent current best is allowed to change, .05 = 5% etc
     letiter  = 5; % # of iterations before checking environment, leave at least 3 so PSO has
time to converge
     outorng  = abs( 1- (outbestval/gbestval) ) >= threshld;
     samepos  = (max( sentry == gbest ));


     if (outorng & samepos) & rem(i,letiter)==0
        rstflg=1;
       % disp('New Environment: reset pbest, gbest, and vel');
       %% reset pbest and pbestval if warranted
```

```
%       outpbestval = feval( functname,[pbest] );
%       Poutorng    = abs( 1-(outpbestval./pbestval) ) > threshld;
%       pbestval    = pbestval.*~Poutorng + outpbestval.*Poutorng;
%       pbest       = pbest.*repmat(~Poutorng,1,D) + pos.*repmat(Poutorng,1,D);

      pbest     = pos; % reset personal bests to current positions
      pbestval  = out;
      vel       = vel*10; % agitate particles a little (or a lot)

     % recalculate best vals
      if minmax == 1
        [gbestval,idx1] = max(pbestval);
      elseif minmax==0
        [gbestval,idx1] = min(pbestval);
      elseif minmax==2 % this section needs work
        [temp,idx1] = min((pbestval-ones(size(pbestval))*errgoal).^2);
        gbestval    = pbestval(idx1);
      end

      gbest  = pbest(idx1,:);

      % used with trainpso, for neural net training
      % assign gbest to net at each iteration, these interim assignments
```

```
   % are for plotting mostly
   if strcmp(functname,'pso_neteval')
      net=setx(net,gbest);
   end
 end  % end if outorng

 sentryval = gbestval;
 sentry    = gbest;

end % end if chkdyn

% find particles where we have new pbest, depending on minmax choice
% then find gbest and gbestval
 %[size(out),size(pbestval)]
if rstflg == 0
 if minmax == 0
   [tempi]           = find(pbestval>=out); % new min pbestvals
   pbestval(tempi,1)  = out(tempi);   % update pbestvals
   pbest(tempi,:)     = pos(tempi,:); % update pbest positions

   [iterbestval,idx1] = min(pbestval);

   if gbestval >= iterbestval
```

```
      gbestval = iterbestval;
      gbest    = pbest(idx1,:);
      % used with trainpso, for neural net training
      % assign gbest to net at each iteration, these interim assignments
      % are for plotting mostly
       if strcmp(functname,'pso_neteval')
         net=setx(net,gbest);
       end
   end
elseif minmax == 1
   [tempi,dum]        = find(pbestval<=out); % new max pbestvals
   pbestval(tempi,1)  = out(tempi,1); % update pbestvals
   pbest(tempi,:)     = pos(tempi,:); % update pbest positions

   [iterbestval,idx1] = max(pbestval);
   if gbestval <= iterbestval
      gbestval = iterbestval;
      gbest    = pbest(idx1,:);
      % used with trainpso, for neural net training
      % assign gbest to net at each iteration, these interim assignments
      % are for plotting mostly
       if strcmp(functname,'pso_neteval')
         net=setx(net,gbest);
```

```
    end
  end
elseif minmax == 2  % this won't work as it is, fix it later
  egones           = errgoal*ones(ps,1); % vector of errgoals
  sqrerr2          = ((pbestval-egones).^2);
  sqrerr1          = ((out-egones).^2);
  [tempi,dum]      = find(sqerr1 <= sqrerr2); % find particles closest to targ
  pbestval(tempi,1) = out(tempi,1); % update pbestvals
  pbest(tempi,:)   = pos(tempi,:); % update pbest positions

  sqrerr           = ((pbestval-egones).^2); % need to do this to reflect new pbests
  [temp,idx1]      = min(sqrerr);
  iterbestval      = pbestval(idx1);

  if (iterbestval-errgoal)^2 <= (gbestval-errgoal)^2
    gbestval = iterbestval;
    gbest    = pbest(idx1,:);
    % used with trainpso, for neural net training
     % assign gbest to net at each iteration, these interim assignments
     % are for plotting mostly
      if strcmp(functname,'pso_neteval')
        net=setx(net,gbest);
      end
```

```
      end
    end
   end


%   % build a simple predictor 10th order, for gbest trajectory
%   if i>500
%    for dimcnt=1:D
%      pred_coef  = polyfit(i-250:i,(bestpos(i-250:i,dimcnt))',20);
%     % pred_coef  = polyfit(200:i,(bestpos(200:i,dimcnt))',20);
%      gbest_pred(i,dimcnt) = polyval(pred_coef,i+1);
%    end
%    else
%      gbest_pred(i,:) = zeros(size(gbest));
%    end

  %gbest_pred(i,:)=gbest;
  %assignin('base','gbest_pred',gbest_pred);

%   % convert to non-inertial frame
%    gbestoffset = gbest - gbest_pred(i,:);
%    gbest = gbest - gbestoffset;
%    pos   = pos + repmat(gbestoffset,ps,1);
```

```
%    pbest = pbest + repmat(gbestoffset,ps,1);



%PSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSO
PSOPSOPSOPSOPSOPSO


    % get new velocities, positions (this is the heart of the PSO algorithm)
    % each epoch get new set of random numbers
     rannum1 = rand([ps,D]); % for Trelea and Clerc types
     rannum2 = rand([ps,D]);
     if     trelea == 2
      % from Trelea's paper, parameter set 2
       vel = 0.729.*vel...                          % prev vel
            +1.494.*rannum1.*(pbest-pos)...         % independent
            +1.494.*rannum2.*(repmat(gbest,ps,1)-pos); % social
     elseif trelea == 1
      % from Trelea's paper, parameter set 1
       vel = 0.600.*vel...                          % prev vel
            +1.700.*rannum1.*(pbest-pos)...         % independent
            +1.700.*rannum2.*(repmat(gbest,ps,1)-pos); % social
     elseif trelea ==3
      % Clerc's Type 1" PSO
       vel = chi*(vel...                            % prev vel
```

```
      +ac1.*rannum1.*(pbest-pos)...              % independent
      +ac2.*rannum2.*(repmat(gbest,ps,1)-pos)) ; % social
else
 % common PSO algo with inertia wt
 % get inertia weight, just a linear funct w.r.t. epoch parameter iwe
  if i<=iwe
    iwt(i) = ((iw2-iw1)/(iwe-1))*(i-1)+iw1;
  else
    iwt(i) = iw2;
  end
 % random number including acceleration constants
  ac11 = rannum1.*ac1;    % for common PSO w/inertia
  ac22 = rannum2.*ac2;

  vel = iwt(i).*vel...                        % prev vel
      +ac11.*(pbest-pos)...                   % independent
      +ac22.*(repmat(gbest,ps,1)-pos);        % social
end

% limit velocities here using masking
 vel = ( (vel <= velmaskmin).*velmaskmin ) + ( (vel > velmaskmin).*vel );
 vel = ( (vel >= velmaskmax).*velmaskmax ) + ( (vel < velmaskmax).*vel );
```

```
% update new position (PSO algo)
 pos = pos + vel;

% position masking, limits positions to desired search space
% method: 0) no position limiting, 1) saturation at limit,
%         2) wraparound at limit , 3) bounce off limit
 minposmask_throwaway = pos <= posmaskmin;  % these are psXD matrices
 minposmask_keep      = pos >  posmaskmin;
 maxposmask_throwaway = pos >= posmaskmax;
 maxposmask_keep      = pos <  posmaskmax;

 if     posmaskmeth == 1
  % this is the saturation method
   pos = ( minposmask_throwaway.*posmaskmin ) + ( minposmask_keep.*pos );
   pos = ( maxposmask_throwaway.*posmaskmax ) + ( maxposmask_keep.*pos );
 elseif posmaskmeth == 2
  % this is the wraparound method
   pos = ( minposmask_throwaway.*posmaskmax ) + ( minposmask_keep.*pos );
   pos = ( maxposmask_throwaway.*posmaskmin ) + ( maxposmask_keep.*pos );
 elseif posmaskmeth == 3
  % this is the bounce method, particles bounce off the boundaries with -vel
   pos = ( minposmask_throwaway.*posmaskmin ) + ( minposmask_keep.*pos );
   pos = ( maxposmask_throwaway.*posmaskmax ) + ( maxposmask_keep.*pos );
```

```
        vel = (vel.*minposmask_keep) + (-vel.*minposmask_throwaway);
        vel = (vel.*maxposmask_keep) + (-vel.*maxposmask_throwaway);
      else
       % no change, this is the original Eberhart, Kennedy method,
       % it lets the particles grow beyond bounds if psoparams (P)
       % especially Vmax, aren't set correctly, see the literature
      end
```

```
%PSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSO
PSOPSOPSOPSOPSOPSO
% check for stopping criterion based on speed of convergence to desired
  % error
   tmp1 = abs(tr(i) - gbestval);
   if tmp1 > ergrd
     cnt2 = 0;
   elseif tmp1 <= ergrd
     cnt2 = cnt2+1;
     if cnt2 >= ergrdep
      if plotflg == 1
       fprintf(message,i,gbestval);
       disp(' ');
```

```
      disp(['--> Solution likely, GBest hasn''t changed by at least ',...
          num2str(ergrd),' for ',...
              num2str(cnt2),' epochs.']);
      eval(plotfcn);
     end
     break
   end
 end

% this stops if using constrained optimization and goal is reached
 if ~isnan(errgoal)
  if ((gbestval<=errgoal) & (minmax==0)) | ((gbestval>=errgoal) & (minmax==1))

     if plotflg == 1
        fprintf(message,i,gbestval);
        disp(' ');
        disp(['--> Error Goal reached, successful termination!']);

        eval(plotfcn);
     end
     break
  end
```

```
   % this is stopping criterion for constrained from both sides
    if minmax == 2
      if ((tr(i)<errgoal) & (gbestval>=errgoal)) | ((tr(i)>errgoal) ...
            & (gbestval <= errgoal))
        if plotflg == 1
            fprintf(message,i,gbestval);
            disp(' ');
            disp(['--> Error Goal reached, successful termination!']);

            eval(plotfcn);
        end
        break
      end
    end % end if minmax==2
   end  % end ~isnan if

 %    % convert back to inertial frame
 %     pos = pos - repmat(gbestoffset,ps,1);
 %     pbest = pbest - repmat(gbestoffset,ps,1);
 %     gbest = gbest + gbestoffset;


end  % end epoch loop
```

```
%% clear temp outputs
% evalin('base','clear temp_pso_out temp_te temp_tr;');

% output & return
 OUT=[gbest';gbestval];
 varargout{1}=[1:te];
 varargout{2}=[tr(find(~isnan(tr)))];

 return
```

```
% forcerow.m
% function to force a vector to be a single row

function[out]=forcerow(in)
len=prod(size(in));
out=reshape(in,[1,len]);
```

```
% goplotpso.m
% default plotting script used in PSO functions

 clf
 set(gcf,'Position',[651    31   626   474]); % this is the computer dependent part
 %set(gcf,'Position',[743    33   853   492]);
 set(gcf,'Doublebuffer','on');

% particle plot, upper right
 subplot('position',[.7,.6,.27,.32]);
 set(gcf,'color','k')

 plot3(pos(:,1),pos(:,D),out,'b.','Markersize',7)

 hold on
 plot3(pbest(:,1),pbest(:,D),pbestval,'g.','Markersize',7);
 plot3(gbest(1),gbest(D),gbestval,'r.','Markersize',25);

 % crosshairs
 offx = max(abs(min(min(pbest(:,1)),min(pos(:,1)))),...
          abs(max(max(pbest(:,1)),max(pos(:,1)))));
```

```
offy = max(abs(min(min(pbest(:,D)),min(pos(:,D)))),...
        abs(min(max(pbest(:,D)),max(pos(:,D)))));
plot3([gbest(1)-offx;gbest(1)+offx],...
    [gbest(D);gbest(D)],...
    [gbestval;gbestval],...
    'r-.');
plot3([gbest(1);gbest(1)],...
    [gbest(D)-offy;gbest(D)+offy],...
    [gbestval;gbestval],...
    'r-.');

hold off

xlabel('Dimension 1','color','y')
ylabel(['Dimension ',num2str(D)],'color','y')
zlabel('Cost','color','y')

title('Particle Dynamics','color','w','fontweight','bold')

set(gca,'Xcolor','y')
set(gca,'Ycolor','y')
set(gca,'Zcolor','y')
set(gca,'color','k')
```

```
% camera control
view(2)
try
  axis([gbest(1)-offx,gbest(1)+offx,gbest(D)-offy,gbest(D)+offy]);
catch
  axis([VR(1,1),VR(1,2),VR(D,1),VR(D,2)]);
end

% error plot, left side
subplot('position',[0.1,0.1,.475,.825]);
 semilogy(tr(find(~isnan(tr))),'color','m','linewidth',2)
 %plot(tr(find(~isnan(tr))),'color','m','linewidth',2)
 xlabel('epoch','color','y')
 ylabel('gbest val.','color','y')

 if D==1
    titstr1=sprintf(['%11.6g = %s( [ %9.6g ] )'],...
            gbestval,strrep(functname,'_','\_'),gbest(1));
 elseif D==2
    titstr1=sprintf(['%11.6g = %s( [ %9.6g, %9.6g ] )'],...
            gbestval,strrep(functname,'_','\_'),gbest(1),gbest(2));
 elseif D==3
```

```
    titstr1=sprintf(['%11.6g = %s( [ %9.6g, %9.6g, %9.6g ] )'],...
            gbestval,strrep(functname,'_','\_'),gbest(1),gbest(2),gbest(3));
  else
    titstr1=sprintf(['%11.6g = %s( [ %g inputs ] )'],...
            gbestval,strrep(functname,'_','\_'),D);
  end
  title(titstr1,'color','m','fontweight','bold');

  grid on
%  axis tight

  set(gca,'Xcolor','y')
  set(gca,'Ycolor','y')
  set(gca,'Zcolor','y')
  set(gca,'color','k')

  set(gca,'YMinorGrid','off')

% text box in lower right
% doing it this way so I can format each line any way I want
subplot('position',[.62,.1,.29,.4]);
  clear titstr
  if trelea==0
```

```
    PSOtype  = 'Common PSO';
    xtraname = 'Inertia Weight : ';
    xtraval  = num2str(iwt(length(iwt)));

  elseif trelea==2 | trelea==1

    PSOtype  = (['Trelea Type ',num2str(trelea)]);
    xtraname = ' ';
    xtraval  = ' ';

  elseif trelea==3
    PSOtype  = (['Clerc Type 1"']);
    xtraname = '\chi value : ';
    xtraval  = num2str(chi);

end
if isnan(errgoal)
  errgoalstr='Unconstrained';
else
  errgoalstr=num2str(errgoal);
end
if minmax==1
   minmaxstr = ['Maximize to : '];
```

```
elseif minmax==0
   minmaxstr = ['Minimize to : '];
else
   minmaxstr = ['Target   to : '];
end

if rstflg==1
   rststat1 = 'Environment Change';
   rststat2 = ' ';
else
   rststat1 = ' ';
   rststat2 = ' ';
end

titstr={'PSO Model: '      ,PSOtype;...
        'Dimensions : '    ,num2str(D);...
        '# of particles : ',num2str(ps);...
        minmaxstr          ,errgoalstr;...
        'Function : '      ,strrep(functname,'_','\_');...
        xtraname           ,xtraval;...
        rststat1           ,rststat2};

text(.1,1,[titstr{1,1},titstr{1,2}],'color','g','fontweight','bold');
```

```
hold on
text(.1,.9,[titstr{2,1},titstr{2,2}],'color','m');
text(.1,.8,[titstr{3,1},titstr{3,2}],'color','m');
text(.1,.7,[titstr{4,1}],'color','w');
text(.55,.7,[titstr{4,2}],'color','m');
text(.1,.6,[titstr{5,1},titstr{5,2}],'color','m');
text(.1,.5,[titstr{6,1},titstr{6,2}],'color','w','fontweight','bold');
text(.1,.4,[titstr{7,1},titstr{7,2}],'color','r','fontweight','bold');

% if we are training a neural net, show a few more parameters
if strcmp('pso_neteval',functname)
  % net is passed from trainpso to pso_Trelea_vectorized in case you are
  % wondering where that structure comes from
  hiddlyrstr = [];
  for lyrcnt=1:length(net.layers)
    TF{lyrcnt} = net.layers{lyrcnt}.transferFcn;
    Sn(lyrcnt) = net.layers{lyrcnt}.dimensions;
    hiddlyrstr = [hiddlyrstr,', ',TF{lyrcnt}];
  end
  hiddlyrstr = hiddlyrstr(3:end);

  text(0.1,.35,['#neur/lyr = [ ',num2str(net.inputs{1}.size),'  ',...
          num2str(Sn),' ']],'color','c','fontweight','normal',...
```

```
            'fontsize',10);
  text(0.1,.275,['Lyr Fcn: ',hiddlyrstr],...
     'color','c','fontweight','normal','fontsize',9);

end



legstr = {'Green = Personal Bests';...
          'Blue  = Current Positions';...
          'Red   = Global Best'};
text(.1,0.025,legstr{1},'color','g');
text(.1,-.05,legstr{2},'color','b');
text(.1,-.125,legstr{3},'color','r');

hold off

set(gca,'color','k');
set(gca,'visible','off');

drawnow
```

```
 function [y  Pl]=f6(in)
 in=abs(in);
global data B B0  Pd
a=data(:,1);
b=data(:,2);
c=data(:,3);
 y1=in.*in*diag(a)+in*diag(b);
 Pl1=(in*B).*in+in*diag(B0);
 Pl=sum(Pl1')';
 lam=abs(Pd+Pl'-sum(in'))';
 y=(sum(y1')+sum(c))'+100*lam;
```

```
% the data matrix should have 5 columns of fuel cost coefficients and plant  limits.
% 1.a ($/MW^2) 2. b $/MW 3. c ($) 4.lower lomit(MW) 5.Upper limit(MW)
%no of rows denote the no of plants(n)
clear
clc;
format long;
global data B B0 B00 Pd
data=[0.008     7         200      10        85
0.009   6.3       180      10        80
0.007   6.8       140 10 70];
B=.01*[.0218 .0093 .0028;.0093 .0228 .0017;.0028 .0017 .0179];
 B0=0*[.0003 .0031 .0015];
 B00=100*.00030523;
 Pd=150;
 Pd=Pd+B00;
l=data(:,4)';
u=data(:,5)';
ran=[l' u'];
n=length(data(:,1));
Pdef = [100 100000 100 2 2 0.9 0.4 1500 1e-6 5000 NaN 0 0];
```

```
[OUT]=pso_Trelea_vectorized('f6',n,1,ran,0,Pdef);
out=abs(OUT)
P=out(1:n)
 [F Pl]=f6(P')
```

SIMULATION RESULT:

```
PSO: 1/100000 iterations, GBest =   1632.5074919140593.
PSO: 100/100000 iterations, GBest =    1597.882135116467.
PSO: 200/100000 iterations, GBest =   1597.8093232242156.
PSO: 300/100000 iterations, GBest =   1597.7974910663006.
PSO: 400/100000 iterations, GBest =   1597.7974910663006.
PSO: 500/100000 iterations, GBest =   1597.7864074485133.
PSO: 600/100000 iterations, GBest =    1597.775247783934.
PSO: 700/100000 iterations, GBest =   1597.7659229125113.
PSO: 800/100000 iterations, GBest =   1597.7659228346886.
PSO: 900/100000 iterations, GBest =   1597.7535586579863.
PSO: 1000/100000 iterations, GBest =   1597.7441037441795.
PSO: 1100/100000 iterations, GBest =   1597.7441030611044.
PSO: 1200/100000 iterations, GBest =   1597.7441030610737.
PSO: 1300/100000 iterations, GBest =   1597.7441030610737.
```

PSO: 1400/100000 iterations, GBest = 1597.7441030610737.

PSO: 1500/100000 iterations, GBest = 1597.7441030610737.

PSO: 1600/100000 iterations, GBest = 1597.7417625988853.

PSO: 1700/100000 iterations, GBest = 1597.7417625988728.

PSO: 1800/100000 iterations, GBest = 1597.7417625988728.

PSO: 1900/100000 iterations, GBest = 1597.7417625988728.

PSO: 2000/100000 iterations, GBest = 1597.7400589502358.

PSO: 2100/100000 iterations, GBest = 1597.7398239799877.

PSO: 2200/100000 iterations, GBest = 1597.7384905040958.

PSO: 2300/100000 iterations, GBest = 1597.7361693317653.

PSO: 2400/100000 iterations, GBest = 1597.7361693317653.

PSO: 2500/100000 iterations, GBest = 1597.7358745606207.

PSO: 2600/100000 iterations, GBest = 1597.7338612604658.

PSO: 2700/100000 iterations, GBest = 1597.7329821456585.

PSO: 2800/100000 iterations, GBest = 1597.7329821367932.

PSO: 2900/100000 iterations, GBest = 1597.7329821367932.

PSO: 3000/100000 iterations, GBest = 1597.7329821367932.

PSO: 3100/100000 iterations, GBest = 1597.7329821367932.

PSO: 3200/100000 iterations, GBest = 1597.7329821367932.

PSO: 3300/100000 iterations, GBest = 1597.7329821367932.

PSO: 3400/100000 iterations, GBest = 1597.7329821367932.

PSO: 3500/100000 iterations, GBest = 1597.7329821367932.

PSO: 3600/100000 iterations, GBest = 1597.7329821367932.

PSO: 3700/100000 iterations, GBest = 1597.7329821367932.

PSO: 3800/100000 iterations, GBest = 1597.7329821367932.

PSO: 3900/100000 iterations, GBest = 1597.7329821367932.

PSO: 4000/100000 iterations, GBest = 1597.7329821367932.

PSO: 4100/100000 iterations, GBest = 1597.7329821367932.

PSO: 4200/100000 iterations, GBest = 1597.7329821367932.

PSO: 4300/100000 iterations, GBest = 1597.7329821367932.

PSO: 4400/100000 iterations, GBest = 1597.7329821367932.

PSO: 4500/100000 iterations, GBest = 1597.7329821367932.

PSO: 4600/100000 iterations, GBest = 1597.7329821367932.

PSO: 4700/100000 iterations, GBest = 1597.7329821367932.

PSO: 4800/100000 iterations, GBest = 1597.7329821367932.

PSO: 4900/100000 iterations, GBest = 1597.7329821367932.

PSO: 5000/100000 iterations, GBest = 1597.7329821367932.

PSO: 5100/100000 iterations, GBest = 1597.7248330782008.

PSO: 5200/100000 iterations, GBest = 1597.7248278205327.

PSO: 5300/100000 iterations, GBest = 1597.7248278205327.

PSO: 5400/100000 iterations, GBest = 1597.7248278205327.

PSO: 5500/100000 iterations, GBest = 1597.7248278205327.

PSO: 5600/100000 iterations, GBest = 1597.7248278205327.

PSO: 5700/100000 iterations, GBest = 1597.7248278205327.

PSO: 5800/100000 iterations, GBest = 1597.7248278205327.

PSO: 5900/100000 iterations, GBest = 1597.7248278205327.

PSO: 6000/100000 iterations, GBest = 1597.7248278205327.

PSO: 6100/100000 iterations, GBest = 1597.7244683219881.

PSO: 6200/100000 iterations, GBest = 1597.7242514442241.

PSO: 6300/100000 iterations, GBest = 1597.7242514442241.

PSO: 6400/100000 iterations, GBest = 1597.7242514442241.

PSO: 6500/100000 iterations, GBest = 1597.7242514442241.

PSO: 6600/100000 iterations, GBest = 1597.7237628788021.

PSO: 6700/100000 iterations, GBest = 1597.7237585283665.

PSO: 6800/100000 iterations, GBest = 1597.7237585283665.

PSO: 6900/100000 iterations, GBest = 1597.7237585283665.

PSO: 7000/100000 iterations, GBest = 1597.7237585283665.

PSO: 7100/100000 iterations, GBest = 1597.7237585283665.

PSO: 7200/100000 iterations, GBest = 1597.7237585283665.

PSO: 7300/100000 iterations, GBest = 1597.7237585283665.

PSO: 7400/100000 iterations, GBest = 1597.7237585283665.

PSO: 7500/100000 iterations, GBest = 1597.7237585283665.

PSO: 7600/100000 iterations, GBest = 1597.7217703636868.

PSO: 7700/100000 iterations, GBest = 1597.7217701186883.

PSO: 7800/100000 iterations, GBest = 1597.7217701186883.

PSO: 7900/100000 iterations, GBest = 1597.7217701186883.

PSO: 8000/100000 iterations, GBest = 1597.7217701186883.

PSO: 8100/100000 iterations, GBest = 1597.7217701186883.

PSO: 8200/100000 iterations, GBest = 1597.7217701186883.

PSO: 8300/100000 iterations, GBest = 1597.7217701186883.

PSO: 8400/100000 iterations, GBest = 1597.7217701186883.

PSO: 8500/100000 iterations, GBest = 1597.7217701186883.

PSO: 8600/100000 iterations, GBest = 1597.7217701186883.

PSO: 8700/100000 iterations, GBest = 1597.7217701186883.

PSO: 8800/100000 iterations, GBest = 1597.7217701186883.

PSO: 8900/100000 iterations, GBest = 1597.7217701186883.

PSO: 9000/100000 iterations, GBest = 1597.7217701186883.

PSO: 9100/100000 iterations, GBest = 1597.7217701186883.

PSO: 9200/100000 iterations, GBest = 1597.7217701186883.

PSO: 9300/100000 iterations, GBest = 1597.7217701186883.

PSO: 9400/100000 iterations, GBest = 1597.7217701186883.

PSO: 9500/100000 iterations, GBest = 1597.7217701186883.

PSO: 9600/100000 iterations, GBest = 1597.7217701186883.

PSO: 9700/100000 iterations, GBest = 1597.7208551318367.

PSO: 9800/100000 iterations, GBest = 1597.7208548636595.

PSO: 9900/100000 iterations, GBest = 1597.7208548636595.

PSO: 10000/100000 iterations, GBest = 1597.7208548636595.

PSO: 10100/100000 iterations, GBest = 1597.7208548636595.

PSO: 10200/100000 iterations, GBest = 1597.7208548636595.

PSO: 10300/100000 iterations, GBest = 1597.7208548636595.

PSO: 10400/100000 iterations, GBest = 1597.7208548636595.

PSO: 10500/100000 iterations, GBest = 1597.7208548636595.

PSO: 10600/100000 iterations, GBest = 1597.7208548636595.

PSO: 10700/100000 iterations, GBest = 1597.7208548636595.

PSO: 10800/100000 iterations, GBest = 1597.7208548636595.

PSO: 10900/100000 iterations, GBest = 1597.7208548636595.

PSO: 11000/100000 iterations, GBest = 1597.7208548636595.

PSO: 11100/100000 iterations, GBest = 1597.7208548636595.

PSO: 11200/100000 iterations, GBest = 1597.7208548636595.

PSO: 11300/100000 iterations, GBest = 1597.7208548636595.

PSO: 11400/100000 iterations, GBest = 1597.7208548636595.

PSO: 11500/100000 iterations, GBest = 1597.7208548636595.

PSO: 11600/100000 iterations, GBest = 1597.7208548636595.

PSO: 11700/100000 iterations, GBest = 1597.7208548636595.

PSO: 11800/100000 iterations, GBest = 1597.7208548636595.

PSO: 11900/100000 iterations, GBest = 1597.7208548636595.

PSO: 12000/100000 iterations, GBest = 1597.7208548636595.

PSO: 12100/100000 iterations, GBest = 1597.7208548636595.

PSO: 12200/100000 iterations, GBest = 1597.7208548636595.

PSO: 12300/100000 iterations, GBest = 1597.7208548636595.

PSO: 12400/100000 iterations, GBest = 1597.7208548636595.

PSO: 12500/100000 iterations, GBest = 1597.7208548636595.

PSO: 12600/100000 iterations, GBest = 1597.7208548636595.

PSO: 12700/100000 iterations, GBest = 1597.720854856927.

PSO: 12800/100000 iterations, GBest = 1597.7208548565961.

PSO: 12900/100000 iterations, GBest = 1597.7208548565961.

PSO: 13000/100000 iterations, GBest = 1597.7208548565961.

PSO: 13100/100000 iterations, GBest = 1597.7208548565961.

PSO: 13200/100000 iterations, GBest = 1597.7208548565961.

PSO: 13300/100000 iterations, GBest = 1597.7208548565961.

PSO: 13400/100000 iterations, GBest = 1597.7208548565961.

PSO: 13500/100000 iterations, GBest = 1597.7208548565961.

PSO: 13600/100000 iterations, GBest = 1597.7208548565961.

PSO: 13700/100000 iterations, GBest = 1597.7208548565961.

PSO: 13800/100000 iterations, GBest = 1597.7208548565961.

PSO: 13900/100000 iterations, GBest = 1597.7208159195.

PSO: 14000/100000 iterations, GBest = 1597.7206926239182.

PSO: 14100/100000 iterations, GBest = 1597.7206926239182.

PSO: 14200/100000 iterations, GBest = 1597.7206926239182.

PSO: 14300/100000 iterations, GBest = 1597.7206926239182.

PSO: 14400/100000 iterations, GBest = 1597.7206926239182.

PSO: 14500/100000 iterations, GBest = 1597.7206926239182.

PSO: 14600/100000 iterations, GBest = 1597.7206926239182.

PSO: 14700/100000 iterations, GBest = 1597.7206926239182.

PSO: 14800/100000 iterations, GBest = 1597.7206926239182.

PSO: 14900/100000 iterations, GBest = 1597.7206926239182.

PSO: 15000/100000 iterations, GBest = 1597.7206926239182.

PSO: 15100/100000 iterations, GBest = 1597.7206926239182.

PSO: 15200/100000 iterations, GBest = 1597.7206926239182.

PSO: 15300/100000 iterations, GBest = 1597.7206926239182.

PSO: 15400/100000 iterations, GBest = 1597.7206926239182.

PSO: 15500/100000 iterations, GBest = 1597.7206926239182.

PSO: 15600/100000 iterations, GBest = 1597.7206926239182.

PSO: 15700/100000 iterations, GBest = 1597.7206224012184.

PSO: 15800/100000 iterations, GBest = 1597.7205341873023.

PSO: 15900/100000 iterations, GBest = 1597.7205341873023.

PSO: 16000/100000 iterations, GBest = 1597.7205341873023.

PSO: 16100/100000 iterations, GBest = 1597.7205341873023.

PSO: 16200/100000 iterations, GBest = 1597.7205341873023.

PSO: 16300/100000 iterations, GBest = 1597.7205341873023.

PSO: 16400/100000 iterations, GBest = 1597.7205341873023.

PSO: 16500/100000 iterations, GBest = 1597.7205341873023.

PSO: 16600/100000 iterations, GBest = 1597.7205341873023.

PSO: 16700/100000 iterations, GBest = 1597.7205341873023.

PSO: 16800/100000 iterations, GBest = 1597.7205341873023.

PSO: 16900/100000 iterations, GBest = 1597.7205341873023.

PSO: 17000/100000 iterations, GBest = 1597.7205332369635.

PSO: 17100/100000 iterations, GBest = 1597.7205332369635.

PSO: 17200/100000 iterations, GBest = 1597.7197603509003.

PSO: 17300/100000 iterations, GBest = 1597.7197599611234.

PSO: 17400/100000 iterations, GBest = 1597.7197599611234.

PSO: 17500/100000 iterations, GBest = 1597.7197599611234.

PSO: 17600/100000 iterations, GBest = 1597.7197599611234.

PSO: 17700/100000 iterations, GBest = 1597.7197599611234.

PSO: 17800/100000 iterations, GBest = 1597.7197599611234.

PSO: 17900/100000 iterations, GBest = 1597.7197599611234.

PSO: 18000/100000 iterations, GBest = 1597.7197599611234.

PSO: 18100/100000 iterations, GBest = 1597.7197599611234.

PSO: 18200/100000 iterations, GBest = 1597.7197599611234.

PSO: 18300/100000 iterations, GBest = 1597.7197599611234.

PSO: 18400/100000 iterations, GBest = 1597.7197599611234.

PSO: 18500/100000 iterations, GBest = 1597.7197599611234.

PSO: 18600/100000 iterations, GBest = 1597.7197599611234.

PSO: 18700/100000 iterations, GBest = 1597.7197599611234.

PSO: 18800/100000 iterations, GBest = 1597.7197599611234.

PSO: 18900/100000 iterations, GBest = 1597.7197599611234.

PSO: 19000/100000 iterations, GBest = 1597.7197599611234.

PSO: 19100/100000 iterations, GBest = 1597.7197599611234.

PSO: 19200/100000 iterations, GBest = 1597.7197599611234.

PSO: 19300/100000 iterations, GBest = 1597.7197599611234.

PSO: 19400/100000 iterations, GBest = 1597.7197599611234.

PSO: 19500/100000 iterations, GBest = 1597.7197599611234.

PSO: 19600/100000 iterations, GBest = 1597.7197599611234.

PSO: 19700/100000 iterations, GBest = 1597.7197599611234.

PSO: 19800/100000 iterations, GBest = 1597.7197599611234.

PSO: 19900/100000 iterations, GBest = 1597.7197599611234.

PSO: 20000/100000 iterations, GBest = 1597.7197599611234.

PSO: 20100/100000 iterations, GBest = 1597.7197599611234.

PSO: 20200/100000 iterations, GBest = 1597.7197599611234.

PSO: 20300/100000 iterations, GBest = 1597.7197599611234.

PSO: 20400/100000 iterations, GBest = 1597.7197599611234.

PSO: 20500/100000 iterations, GBest = 1597.7197599611234.

PSO: 20600/100000 iterations, GBest = 1597.7197599611234.

PSO: 20700/100000 iterations, GBest = 1597.7197599611234.

PSO: 20800/100000 iterations, GBest = 1597.7197599611234.

PSO: 20900/100000 iterations, GBest = 1597.7197599611234.

PSO: 21000/100000 iterations, GBest = 1597.7197599611234.

PSO: 21100/100000 iterations, GBest = 1597.7197599611234.

PSO: 21200/100000 iterations, GBest = 1597.7197599611234.

PSO: 21300/100000 iterations, GBest = 1597.7197599611234.

PSO: 21400/100000 iterations, GBest = 1597.7197599611234.

PSO: 21500/100000 iterations, GBest = 1597.7197599611234.

PSO: 21600/100000 iterations, GBest = 1597.7197599611234.

PSO: 21700/100000 iterations, GBest = 1597.7197599611234.

PSO: 21800/100000 iterations, GBest = 1597.7197599611234.

PSO: 21900/100000 iterations, GBest = 1597.7197599611234.

PSO: 22000/100000 iterations, GBest = 1597.7197599611234.

PSO: 22100/100000 iterations, GBest = 1597.7197599611234.

PSO: 22192/100000 iterations, GBest = 1597.7197599611234.

--> Solution likely, GBest hasn't changed by at least 1e-006 for 5000 epochs.

out =

1.0e+003 *

0.03279536615492

0.06432351520447

0.05525055374426

1.59771975996112

P =

32.79536615491545

64.32351520447459

55.25055374426049

F =1.597719759961123e+003

Pl = 2.33891210365054

CONCLUSION

This paper presents a new approach to ELD problems based on the **PSO** algorithm. **A** position adjustment strategy is incorporated in the **PSO** framework in order to provide solutions satisfying the inequality constraints. The equality constraint in the ELD problem **is** resolved by reducing the degree of freedom by one at random. The strategies for handling constraints are devised in order not to intervene the dynamic process of PSO algorithm. The proposed PSO provided the global solution for the ELD problems with smooth cost functions within a reasonable computation time and iteration number.

The PSO algorithm has been demonstrated to have superior features including high quality solution, stable convergence characteristics, and less computation time. The PSO method is employed to solve the ELD problem. The proposed method was indeed capable of obtaining higher quality solutions with better computation efficiency and convergence property.

REFERENCES

1. Zwe-Lee Gaing, "Particle Swarm Optimization to solving the Economic Dispatch Considering the Generator Constraints", *IEEE Trans. On Power Systems,* Vol.18, No.3, pp. 1187-1195, August 2003.
2. P. J. Angeline, "Using selection to improve particle swarm optimization", *Proceedings of IEEE Int. Conference on Evolutionary Computation*, pp. 84-89, May 1998.
3. Park, J.-B., K.-S. Lee, J.-R. Shin, and K. Y. Lee, "A Particle Swarm Optimization for Economic Dispatch with Non-Smooth Cost Functions," IEEE Transactions on Power Systems, Vol. 20, No. 1, pp. 34-42, February 2005.
4. J. Kennedy and R. C. Eberhart, "Particle swarm optimization," *Proceedings of IEEE International Conference on Neural Networks (ICNN'95)*, Vol. IV, pp. 1942-1948, Perth, Australia, 1995.
5. Birge, B.," *PSOt, A Particle Swarm Optimization Toolbox for Matlab"*, **IEEE Swarm Intelligence Symposium Proceedings**, April 24-26,2003.
6. K. Y. Lee and M. A. El-Sharkawi (Editors), Modern ***Heuristic Optimization Techniques with Applications*** to ***Power*** **Systems,** IEEE Power Engineering Society (02TP160), 2002