We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

5

Educational Simulator for Particle Swarm Optimization and Economic Dispatch Applications

Woo Nam Lee and Jong Bae Park
Konkuk University
Korea

1. Introduction

Optimization problems are widely encountered in various fields in science and technology. Sometimes such problems can be very complex due to the actual and practical nature of the objective function or the model constraints. Most of power system optimization problems have complex and nonlinear characteristics with heavy equality and inequality constraints. Recently, as an alternative to the conventional mathematical approaches, the heuristic optimization techniques such as genetic algorithms (GAs), Tabu search, simulated annealing, and particle swarm optimization (PSO) are considered as realistic and powerful solution schemes to obtain the global or quasi-global optimums (K. Y. Lee et al., 2002).

In 1995, Eberhart and Kennedy suggested a PSO based on the analogy of swarm of bird and school of fish (J. Kennedy et al., 1995). The PSO mimics the behavior of individuals in a swarm to maximize the survival of the species. In PSO, each individual makes his decision using his own experience together with other individuals' experiences (H. Yoshida et al., 2000). The algorithm, which is based on a metaphor of social interaction, searches a space by adjusting the trajectories of moving points in a multidimensional space. The individual particles are drawn stochastically toward the present velocity of each individual, their own previous best performance, and the best previous performance of their neighbours (M. Clerc et al., 2002).

The practical economic dispatch (ED) problems with valve-point and multi-fuel effects are represented as a non-smooth optimization problem with equality and inequality constraints, and this makes the problem of finding the global optimum difficult. Over the past few decades, in order to solve this problem, many salient methods have been proposed such as a hierarchical numerical method (C. E. Lin et al., 1984), dynamic programming (A. J. Wood et al., 1984), evolutionary programming (Y. M. Park et al., 1998; H. T. Yang et al., 1996; N. Sinba et al., 2003), Tabu search (W. M. Lin et al., 2002), neural network approaches (J. H. Park et al., 1993; K. Y. Lee et al., 1998), differential evolution (L. S. Coelho et al., 2006), particle swarm optimization (J. B. Park et al., 2005; T. A. A. Victoire et al., 2004; T. A. A. Victoire et al., 2005), and genetic algorithm (D. C. Walters et al., 1993).

This chapter would introduce an educational simulator for the PSO algorithm. The purpose of this simulator is to provide the undergraduate students with a simple and useable tool for

gaining an intuitive feel for PSO algorithm, mathematical optimization problems, and power system optimization problems. To aid the understanding of PSO, the simulator has been developed under the user-friendly graphic user interface (GUI) environment using MATLAB. In this simulator, instructors and students can set parameters related to the performance of PSO and can observe the impact of the parameters to the solution quality. This simulator also displays the movements of each particle and convergence process of a group. In addition, the simulator can consider other mathematical or power system optimization problems with simple additional MATLAB coding.

2. Overview of particle swarm optimization

Kennedy and Eberhart (J. Kennedy et al., 1995) developed a PSO algorithm based on the behavior of individuals of a swarm. Its roots are in zoologist's modeling of the movement of individuals (e.g., fishes, birds, or insects) within a group. It has been noticed that members within a group seem to share information among them, a fact that leads to increased efficiency of the group (J. Kennedy et al., 2001). The PSO algorithm searches in parallel using a group of individuals similar to other AI-based heuristic optimization techniques.

In a physical n-dimensional search space, the position and velocity of individual i are represented as the vectors $X_i = (x_{i1}, \cdots, x_{in})$ and $V_i = (v_{i1}, \cdots v_{in})$ in the PSO algorithm. Let $Pbest_i = \left(x_{i1}^{Pbest}, \cdots, x_{in}^{Pbest}\right)$ and $Gbest = \left(x_1^{Gbest}, \cdots, x_n^{Gbest}\right)$ be the best position of individual i and its neighbors' best position so far, respectively. The modified velocity and position of each individual can be calculated using the current velocity and the distance from $Pbest_i$ to $Gbest$ as follows:

$$V_i^{k+1} = \omega V_i^k + c_1 rand_1 \times (Pbest_i^k - X_i^k) + c_2 rand_2 \times (Gbest^k - X_i^k) \quad (1)$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \quad (2)$$

where,

V_i^k velocity of individual i at iteration k,

ω weight parameter,

c_1, c_2 acceleration coefficients,

$rand_1, rand_2$ random numbers between 0 and 1,

X_i^k position of individual i at iteration k,

$Pbest_i^k$ best position of individual i until iteration k,

$Gbest^k$ best position of the group until iteration k.

The constants c_1 and c_2 represent the weighting of the stochastic acceleration terms that pull each particle toward the Pbest and Gbest positions. Suitable selection of inertia weight provides a balance between global and local explorations, thus requiring less iteration on average to find a sufficiently optimal solution. In general, the inertia weight ω has a linearly

decreasing dynamic parameter framework descending from ω_{max} to ω_{min} as follows (K. Y. Lee et al., 2002; H. Yoshida et al., 2000; J. B. Park et al., 2005).

$$\omega = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{Iter_{max}} \times Iter \tag{3}$$

Were, $Iter_{max}$ is maximum iteration number and $Iter$ is current iteration number.

3. Structure of educational PSO simulator

3.1 Purpose and motivation of simulator

As a result of the rapid advances in computer hardware and software, computer-based power system educational tools have grown from very simple implementations, providing the user with little more than a stream of numerical output, to very detailed representations of the power system with an extensive GUI. Overbye, et al. had developed a user-friendly simulation program, PowerWorld Simulator, for teaching power system operation and control (T. J. Overbye et al., 2003). They applied visualization to power system information to draw user's attention and effectively display the simulation results. Through these works, they expected that animation, contouring, data aggregation and virtual environments would be quite useful techniques that are able to provide efficient learning experience to users. Also they presented experimental results associated with human factors aspects of using this visualization (D. A. Wiegmann et al., 2005; D. A. Wiegmann et al., 2006; N. Sinba et al., 2003).

Therefore, like other previous simulators, the motivation for the development of this simulator is to provide the students with a simple and useable tool for gaining an intuitive feel for the PSO algorithm, mathematical and power system optimization problems.

3.2 Functions of simulator

The basic objectives of this simulator were to make it easy to use and to provide effective visualization capability suitable for presentations as well as individual studies. This educational simulator was developed by MATLAB 2009b. MATLAB is a scientific computing language developed by The Mathworks, Inc. that is run in an interpreter mode on a wide variety of operating systems. It is extremely powerful, simple to use, and can be found in most research and engineering environments.

The structure and data flow of the developed PSO simulator is shown in Fig. 1. The simulator consists of 3-parts, that is, i) user setting of optimization function as well as parameters, ii) output result, and iii) visualized output variations, as shown in Figs. 2, 3, and 4, respectively. Since the main interaction between user and the simulator is performed through the GUI, it presents novice users with the information they need, and provides easy access for advanced users to additional detailed information. Thus, the GUI is instrumental in allowing users to gain an intuitive feel of the PSO algorithm, rather than just learning how to use this simulator.

In this simulator, parameters (i.e., maximum number of the iteration, maximum and minimum number of inertia weight, acceleration factors c_1 and c_2, and number of particles) that have the influence of PSO performance can be directly inputted by users. In addition,

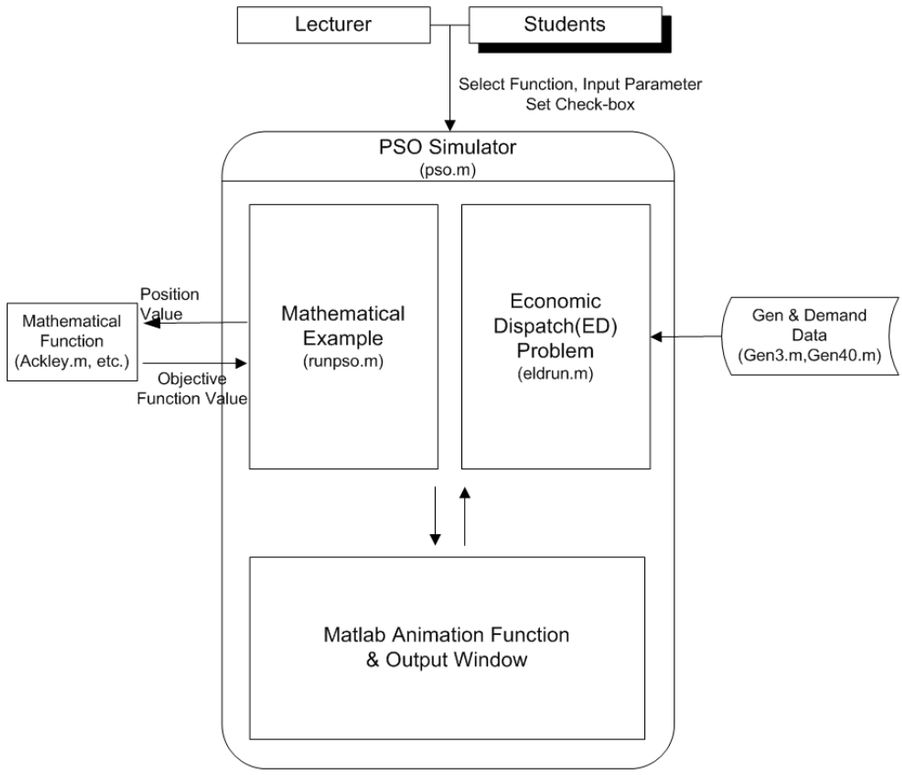


Fig. 1. Structure of the developed PSO simulator

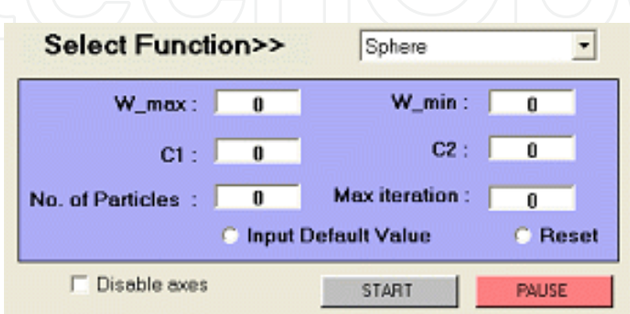


Fig. 2. A window for user setting

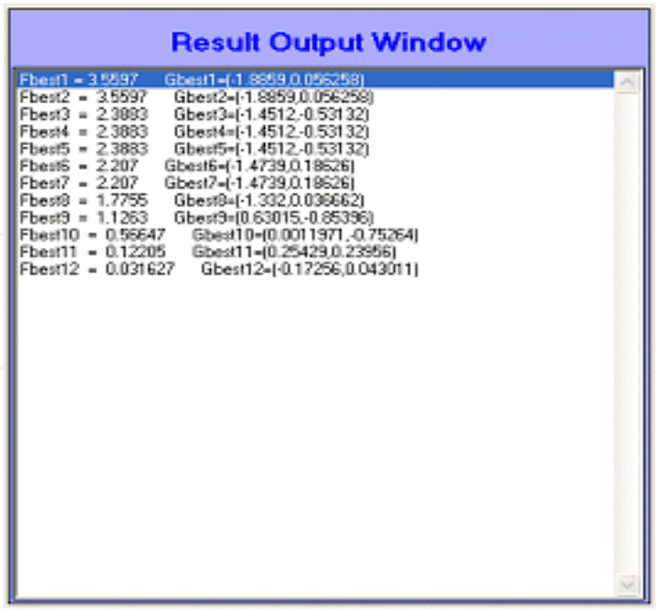


Fig. 3. Output result window

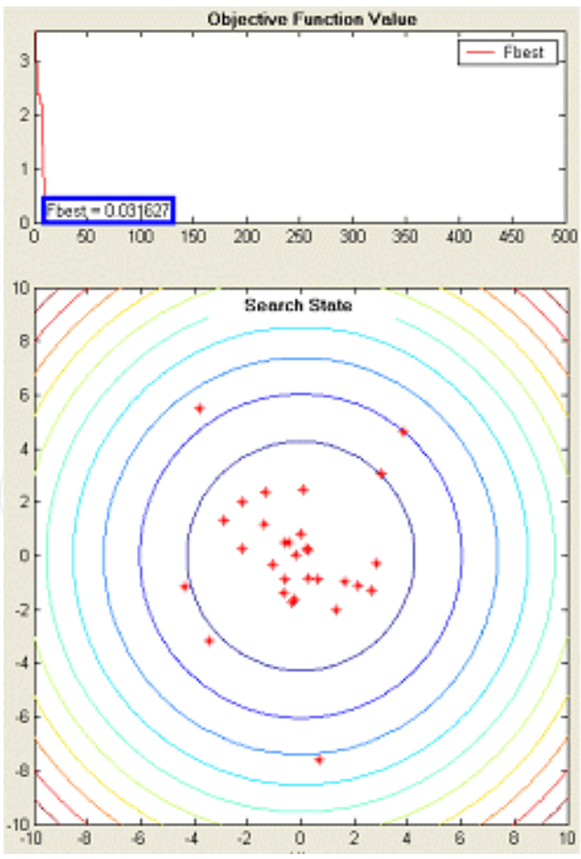


Fig. 4. Viewing parts of output variation

"Input Default Value" check-box was added for users who don't know the proper parameter values of the PSO. If the users push the "START" button finally, then the users can observe the evolution process of the particles on contours of the objective function (in case of a mathematical example) or the output histogram of each generator through MATLAB animation functions and check the changes of the values of the objective function and control variables at each iteration. The "Disable axes" check-box is used when the users want to show only the values of the final result fast. When the check-box is checked, only the final results (i.e., the value of the objective function and control variables) are expressed in the "Result Output Window". At any point in time in the simulation, the user can pause or restart the simulation by pushing the "PAUSE" button. As shown in Figs. 5 and 6, user can observe movements of each particle as well as the trend of the value of the objective function.

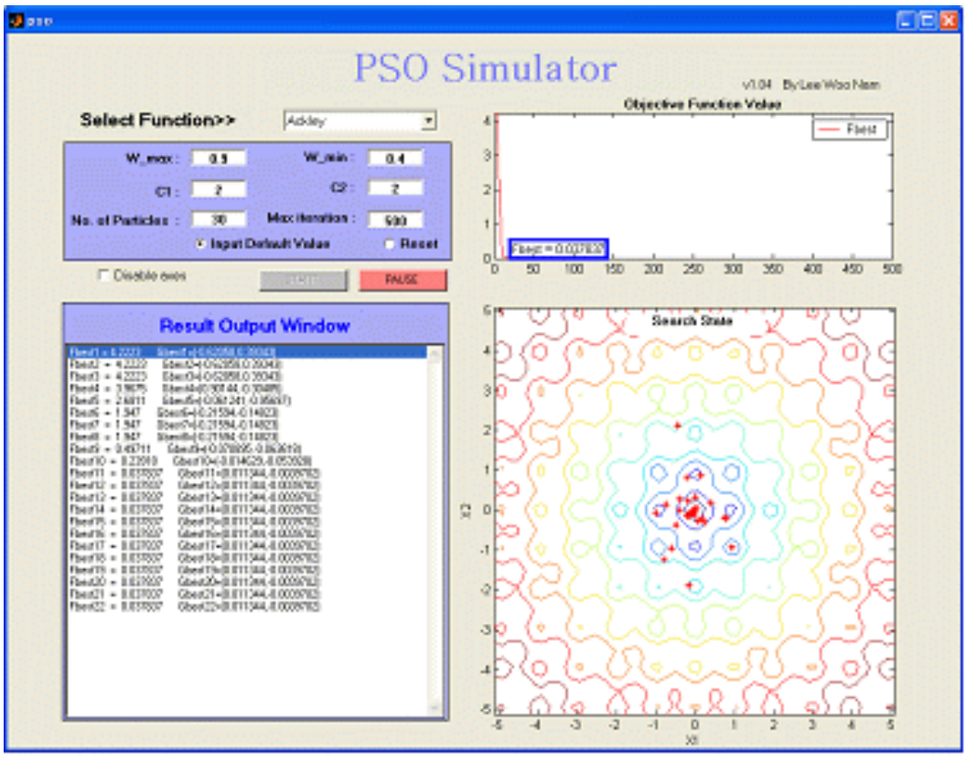


Fig. 5. Simulation for a mathematical example

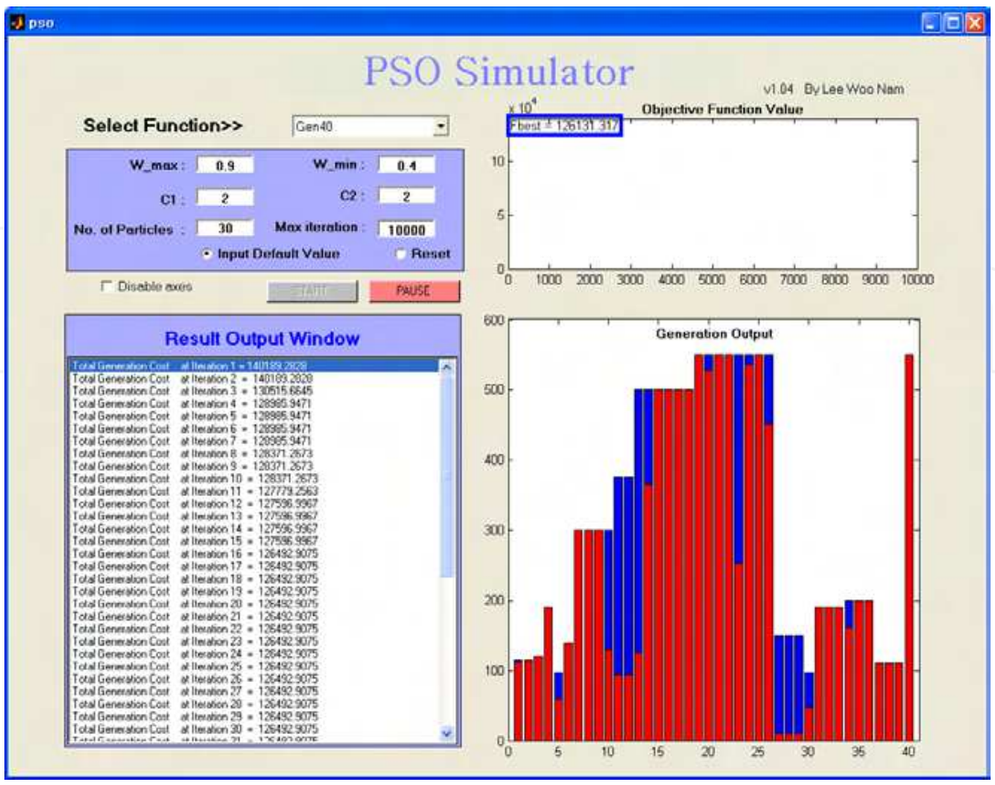


Fig. 6. Simulation for an economic dispatch problem.

4. Economic dispatch(ED) problem

4.1 Basic ED problem formulation

The ED problem is one of the basic optimization problems for the students who meet the power system engineering. The objective is to find the optimal combination of power generations that minimizes the total generation cost while satisfying an equality constraint and a set of inequality constraints. The most simplified cost function can be represented in a quadratic form as following (A. J. Wood et al., 1984):

$$C = \sum_{j \in J} F_j(P_j) \tag{4}$$

$$F_j(P_j) = a_j + b_j P_j + c_j P_j^2 \tag{5}$$

where,

C total generation cost;

F_j cost function of generator j;

a_j, b_j, c_j cost coefficients of generator j;

P_j electrical output of generator j;

J set for all generators.

While minimizing the total generation cost, the total generated power should be the same as the total load demand plus the total line loss. However, the transmission loss is not considered in this paper for simplicity. In addition, the generation output of each generator should be laid between minimum and maximum limits as follows:

$$P_{j\min} \le P_j \le P_{j\max} \tag{6}$$

where $P_{j\min}$ and $P_{j\max}$ are the minimum and maximum output of generator j, respectively.

4.2 Valve-point effects

The generating units with multi-valve steam turbines exhibit a greater variation in the fuel-cost functions. Since the valve point results in ripples, a cost function contains high order nonlinearities (H. T. Yang et al., 1996; N. Sinba et al., 2003; D. C. Walters et al. 1993). Therefore, the cost function (5) should be replaced by the following to consider the valve-point effects:

$$F_j(P_j) = a_j + b_j P_j + c_j P_j^2 + \left| e_j \times \sin(f_j \times (P_{j\min} - P_j)) \right| \tag{7}$$

where e_j and f_j are the cost coefficients of generator j reflecting valve-point effects. Here, the sinusoidal functions are added to the quadratic cost functions.

5. Case studies

This simulator can choose and run five different mathematical examples and two different ED problems: (i) The Sphere function, (ii) The Rosenbrock (or banana-valley) function, (iii) Ackley's function, (iv) The generalized Rastrigin function, (v) The generalized Griewank function, (vi) 3-unit system with valve-point effects, and (vii) 40-unit system with valve-point effects. In the case of each mathematical example (functions (i)-(v)), two input variables (i.e., 2-dimensional space) have been set in order to show the movement of particles on contour. For the case study, 30 independent trials are conducted to observe the variation during the evolutionary processes and compare the solution quality and convergence characteristics.

To successfully implement the PSO, some parameters must be assigned in advance. The population size NP is set to 30. Since the performance of PSO depends on its parameters such as inertia weight ω and two acceleration coefficients (i.e., c_1 and c_2), it is very important to determine the suitable values of parameters. The inertia weight is varied from 0.9 (i.e., $\omega_{\max}$) to 0.4 (i.e., $\omega_{\min}$), as these values are accepted as typical for solving wide varieties of problems. Two acceleration coefficients are determined through the experiments for each problem so as to find the optimal combination.

5.1 Mathematical examples

For development of user's understanding of the PSO algorithm, five non-linear mathematical examples are used here. In each case, the maximum number of iterations (i.e., $iter_{\max}$) was set to 500. The acceleration coefficients (i.e., c_1 and c_2) was equally set to 2.0

from the experimental results for each case using the typical PSO algorithm. And all of the global minimum value of each function is known as 0. The global minimum value was successfully verified by the simulator.

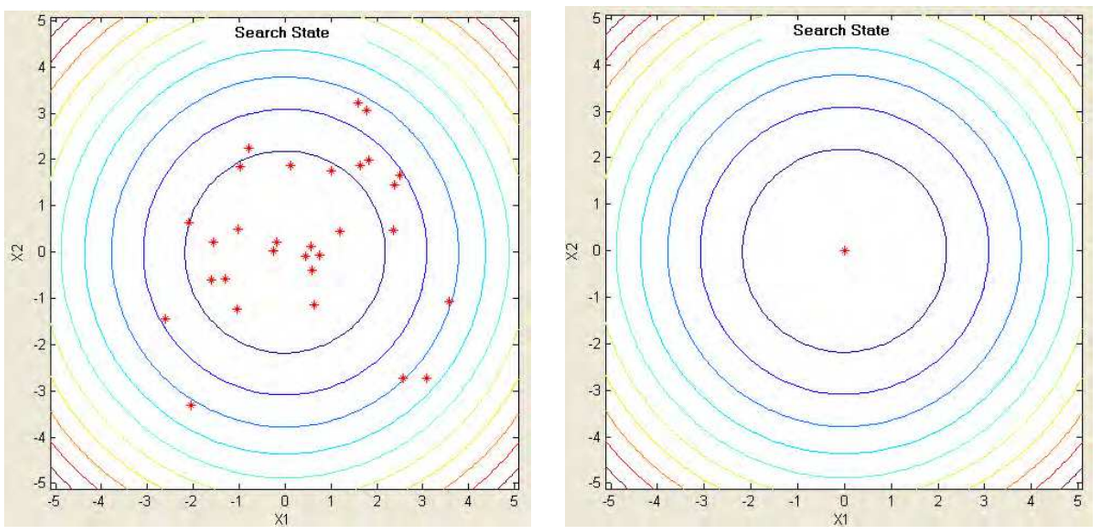
5.1.1 The sphere function

The function and the initial position range of input variables (i.e., x_i) are as follows:

$$f_0(x) = \sum_{i=1}^{n} x_i^2 \tag{8}$$

$$-5.12 \le x_i \le 5.12$$

Initial and final stages of the optimization process for the Sphere function are shown in Fig. 7.



(a) Initial stage (b) Final stage

Fig. 7. Optimization process for the sphere function.

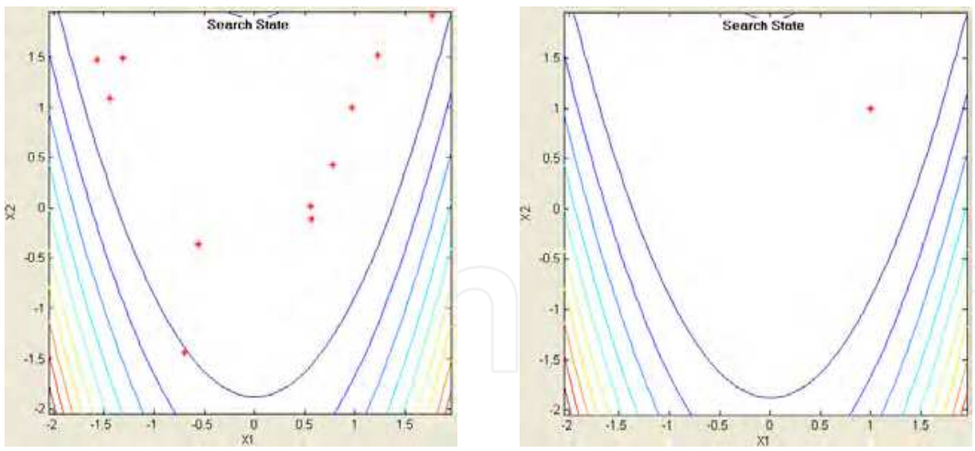
5.1.2 The rosenbrock (or banana-valley) function

The function and the initial position range of input variables (i.e., x_i) are as follows:

$$f_1(x) = \sum_{i=1}^{n/2} (100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2) \tag{9}$$

$$-2.048 \le x_i \le 2.048$$

Initial and final stages of the optimization process for the Rosenbrock function are shown in Fig. 8.



(a) Initial stage b) Final stage

Fig. 8. Optimization process for the Rosenbrock function.

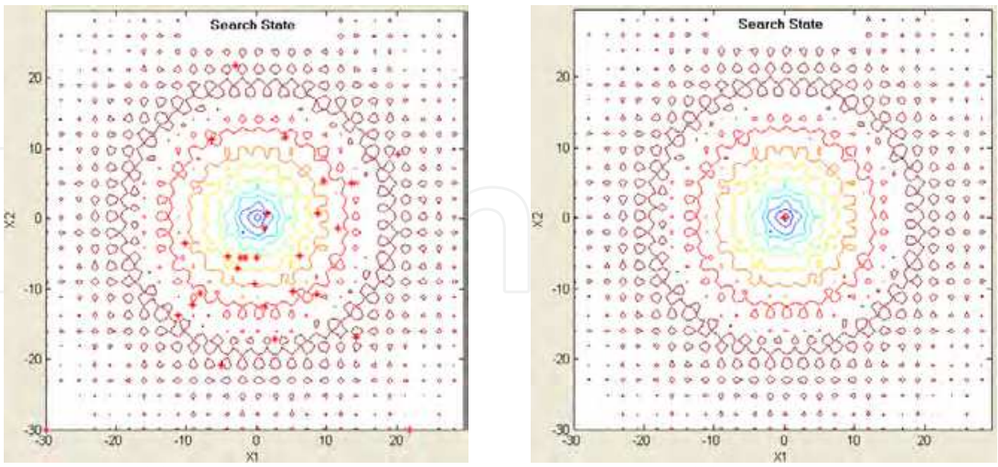
5.1.3 The ackley's function

The function and the initial position range of input variables (i.e., x_i) is as follows:

$$f_2(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \tag{10}$$

$$-30 \le x_i \le 30$$

Initial and final stages of the optimization process for the Ackley's function are shown in Fig. 9.



(a) Initial stage (b) Final stage

Fig. 9. Optimization process for Ackley's function.

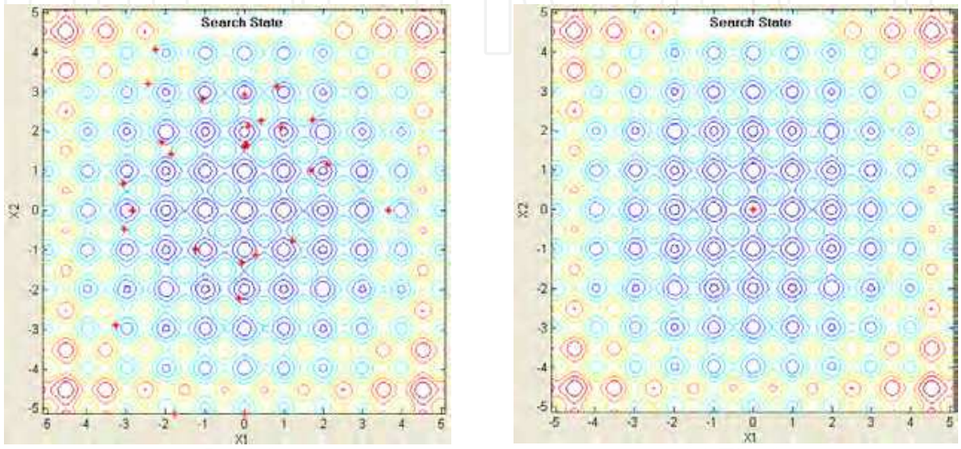
5.1.4 The generalized rastrigin function

The function and the initial position range of input variables (i.e., x_i) is as follows:

$$f_3(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10) \tag{11}$$

$$-5.12 \le x_i \le 5.12$$

Initial and final stages of the optimization process for the generalized Rastrigin function are shown in Fig. 10.



(a) Initial stage (b) Final stage

Fig. 10. Optimization process for Rastrigin function.

5.1.5 The generalized griewank function

The function and the initial position range of input variables (i.e., x_i) is as follows:

$$f_4(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{12}$$

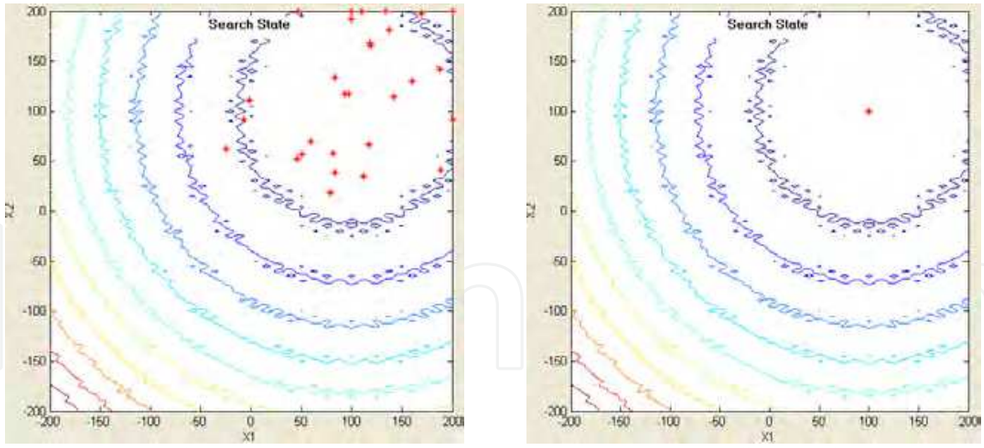
$$-200 \le x_i \le 200$$

Initial and final stages of the optimization process for the generalized Griewank function are shown in Fig. 11.

Table 1 shows the average values of objective functions and two input variables for each function achieved by the PSO simulator.

Function Name	Objective Function Value	x_1	x_2
Sphere	0	0	0
Rosenbrock	0	1	1
Ackley	-8.8818e-16	-2.9595e-16	1.6273e-16
Rastrigin	0	9.7733e-10	-7.9493e-10
Griewank	0	100	100

Table 1. Results for Each Test Function



(a) Initial stage (b) Final stage

Fig. 11. Optimization process for Griewank function.

5.2 Economic dispatch(ED) problems with valve-point effects

This simulator also offers examples to solve ED problem for two different power systems: a 3-unit system with valve-point effects, and a 40-unit system with valve-point effects. The total demands of the 3-unit and the 40-unit systems are set to 850MW and 10,500MW, respectively. All the system data and related constraints of the test systems are given in (N. Sinba et al., 2003). Because these systems have more than 3 input variables, the simulator shows a histogram for the generation output instead of the contour and particles. Since the global minimum for the total generation cost is unknown in the case of the 40-unit system, the maximum number of iterations (i.e., $iter_{\max}$) is set to 10,000 in order to sufficiently search for the minimum value.

Table 2 shows the minimum, mean, maximum, and standard deviation for the 3-unit system obtained from the simulator. The generation outputs and the corresponding costs of the best solution for 3-unit system are described in Table 3.

Case	Minimum Cost ($)	Average Cost ($)	Maximum Cost ($)	Standard Deviation
3-Unit System	8234.0717	8234.0717	8234.0717	0

* Global value of the 3-unit system was known as 8234.0717.T

Table 2. Convergence Results for 3-Unit System

Unit	Generation	Cost
1	300.2669	3087.5099
2	400.0000	3767.1246
3	149.7331	1379.4372
TP/TC	850.0000	8234.0717

*TP: total power [MW], TC: total generation cost [$]

Table 3. Generation Output of Each Generator and The Corresponding Cost in 3-Unit System

In order to find the optimal combination of parameters (i.e., ω_{max}, ω_{min}, c_1, and c_2 B), six cases are considered as given in Table 4. The parameters are determined through the experiments for 40-unit system using the simulator. In Table 4, the effects of parameters are illustrated

Cases	ω_{max}	ω_{min}	c1,c2	Minimum Cost ($)	Average Cost ($)	Maximum Cost ($)	Standard Deviation
1	1.0	0.5	1	121755.49	122221.90	122624.07	156.97
2	0.9	0.4	1	121761.40	122343.32	123087.16	303.62
3	0.8	0.3	1	121949.15	122842.59	124363.11	602.06
4	1.0	0.5	2	121865.23	122285.12	122658.29	175.19
5	0.9	0.4	2	121768.69	122140.32	122608.27	187.74
6	0.8	0.3	2	121757.09	122158.00	122615.71	212.36

Table 4. Influence of Acceleration Coefficients for 40-Unit System

The result screens for 3-unit and 40-unit system are shown in Figs. 12 and 13, respectively. Each histogram expresses the result of generation output for each generator.

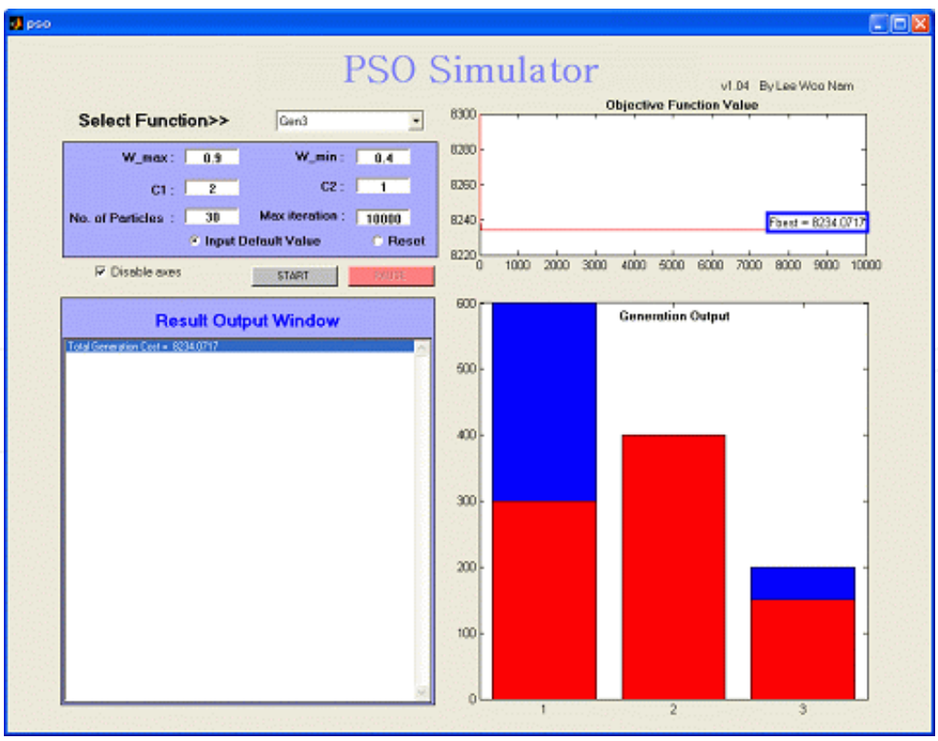


Fig. 12. Result screen for the 3-units system.

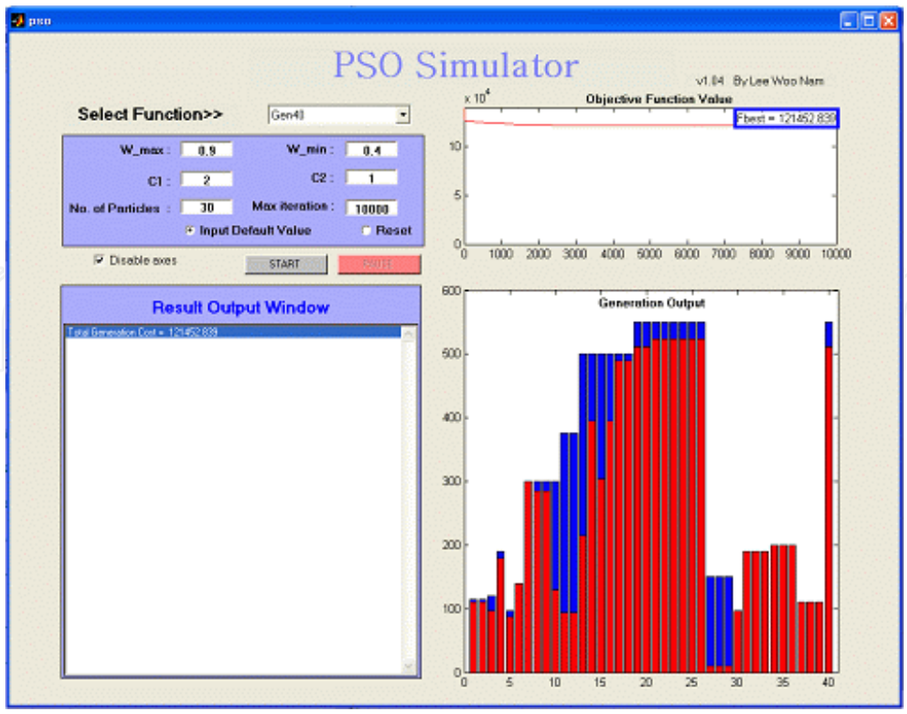


Fig. 13. Result screen for the 40-units system.

6. Conclusion

This chapter presents an educational simulator for particle swarm optimization (PSO) and application for solving mathematical test functions as well as ED problems with non-smooth cost functions. Using this simulator, instructors and students can select the test functions for simulation and set the parameters that have an influence on the PSO performance. Through visualization process of each particle and variation of the value of objective function, the simulator is particularly effective in providing users with an intuitive feel for the PSO algorithm. This simulator is expected to be an useful tool for students who study electrical engineering and optimization techniques.

7. Appendix 1: pso.m

```
function varargout = pso(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @pso_OpeningFcn, ...
    'gui_OutputFcn',  @pso_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
function pso_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = pso_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function select_func_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function select_func_Callback(hObject, eventdata, handles)
function default_value_Callback(hObject, eventdata, handles)
set(handles.default_value, 'Value', 1);
set(handles.reset, 'Value', 0);
set(handles.wmax, 'String', 0.9);
set(handles.wmin, 'String', 0.4);
set(handles.X_max, 'String', 5.12);
set(handles.X_min, 'String', -5.12);
set(handles.c1, 'String', 2);
set(handles.c2, 'String', 2);
set(handles.N, 'String', 30);
set(handles.itmax, 'String', 500);
function reset_Callback(hObject, eventdata, handles)
set(handles.default_value, 'Value', 0);
set(handles.reset, 'Value', 1);
set(handles.wmax, 'String', 0);
set(handles.wmin, 'String', 0);
set(handles.X_max, 'String', 0);
set(handles.X_min, 'String', 0);
set(handles.c1, 'String', 0);
set(handles.c2, 'String', 0);
set(handles.N, 'String', 0);
set(handles.itmax, 'String', 0);
function wmax_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function wmax_Callback(hObject, eventdata, handles)
wmax = str2double(get(hObject,'String'));
if isnan(wmax)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
pso_para = getappdata(gcbf, 'metricdata');
pso_para.wmax = wmax;
setappdata(gcbf, 'metricdata', pso_para);
function wmin_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function wmin_Callback(hObject, eventdata, handles)
wmin = str2double(get(hObject,'String'));
if isnan(wmin)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
pso_para = getappdata(gcbf, 'metricdata');
pso_para.wmin = wmin;
setappdata(gcbf, 'metricdata', pso_para);
function c1_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function c1_Callback(hObject, eventdata, handles)
c1 = str2double(get(hObject,'String'));
if isnan(c1)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
pso_para = getappdata(gcbf, 'metricdata');
pso_para.c1 = c1;
setappdata(gcbf, 'metricdata', pso_para);
function c2_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function c2_Callback(hObject, eventdata, handles)
c2 = str2double(get(hObject,'String'));
if isnan(c2)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
pso_para = getappdata(gcbf, 'metricdata');
pso_para.c2 = c2;
setappdata(gcbf, 'metricdata', pso_para);
function N_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function N_Callback(hObject, eventdata, handles)
N = str2double(get(hObject,'String'));
if isnan(N)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
pso_para = getappdata(gcbf, 'metricdata');
pso_para.N = N;
setappdata(gcbf, 'metricdata', pso_para);
function itmax_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
```

```
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function itmax_Callback(hObject, eventdata, handles)
itmax = str2double(get(hObject,'String'));
if isnan(itmax)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
function start_Callback(hObject, eventdata, handles)
if get(handles.select_func,'value')>=7
    eldrun
else
    runpso
end
function Result_window_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function Result_window_Callback(hObject, eventdata, handles)
function pause_Callback(hObject, eventdata, handles)
if isequal(get(handles.pause,'String'),'PAUSE')
    set(handles.start,'Enable','on');
    set(handles.pause,'String','RESUME');
    uiwait;
else
    set(handles.start,'Enable','off');
    set(handles.pause,'String','PAUSE');
    uiresume;
end
function disable_Callback(hObject, eventdata, handles)
function close_Callback(hObject, eventdata, handles)
delete(get(0,'CurrentFigure'));
function X_max_Callback(hObject, eventdata, handles)
X_max = str2double(get(hObject,'String'));
if isnan(X_max)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
pso_para = getappdata(gcbf, 'metricdata');
pso_para.X_max = X_max;
setappdata(gcbf, 'metricdata', pso_para);
function X_max_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function X_min_Callback(hObject, eventdata, handles)
X_min = str2double(get(hObject,'String'));
if isnan(X_min)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
pso_para = getappdata(gcbf, 'metricdata');
```

```
pso_para.X_min = X_min;
setappdata(gcbf, 'metricdata', pso_para);
function X_min_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

8. Appendix 2: runpso.m

```
cla;
set(handles.start,'Enable','off');
set(handles.pause,'String','PAUSE','Enable','on');
set(handles.text14,'String','Search State ');
functnames = get(handles.select_func,'String');
functname = functnames{get(handles.select_func,'Value')};
wmax = str2double(get(handles.wmax, 'String'));
wmin = str2double(get(handles.wmin, 'String'));
X_max = str2double(get(handles.X_max, 'String'));
X_min = str2double(get(handles.X_min, 'String'));
c1 = str2double(get(handles.c1, 'String'));
c2 = str2double(get(handles.c2, 'String'));
N = str2double(get(handles.N, 'String'));
itmax = str2double(get(handles.itmax, 'String'));
pso_para = getappdata(gcbf, 'metricdata');
pso_para.wmax = wmax;
pso_para.wmin = wmin;
pso_para.X_max = X_max;
pso_para.X_min = X_min;
pso_para.c1 = c1;
pso_para.c2 = c2;
pso_para.N = N;
pso_para.itmax = itmax;
setappdata(gcbf, 'metricdata', pso_para);
D=2; % Dimension
% Weight Parameter
for iter=1:pso_para.itmax
    W(iter)= pso_para.wmax-((pso_para.wmax-
pso_para.wmin)/pso_para.itmax)*iter;
end
%Initialization of positions of agents
% agents are initialized between -5.12,+5.12 randomly
a= X_min;  %min
b= X_max;   %max
x=a+(b-a)*rand(pso_para.N,D,1);
%Initialization of velocities of agents
%Between -5.12 , +5.12, (which can also be started from zero)
m=X_min;
n=X_max;
V=m+(n-m)*rand(pso_para.N,D,1);
%Function to be minimized.
```

```
F = feval(functname,x(:,:,1));
% Saving address and value; C:Value of E, I: The Number of Particle
[C,I]=min(abs(F(:,1,1))); B(1,1,1)=C;
XX(1,1,1)=I;
gbest(1,:,1)=x(I,:,1);
%Matrix composed of gbest vector
for p=1:pso_para.N
    for r=1:D
        G(p,r,1)=gbest(1,r,1);
    end
end
Fbest(1,1,1) = feval(functname,G(1,:,1));
pbest=x;
% Calculating Velocity
V(:,:,2) = W(1) * V(:,:,1) + pso_para.c1*rand*(pbest(:,:,1)-
x(:,:,1)) + pso_para.c2*rand*(G(:,:,1)-x(:,:,1));
x(:,:,2)=x(:,:,1) + V(:,:,2);
for i=1:pso_para.N
    for j=1:D
        if x(i,j,2)<a
            x(i,j,2)=a;
        else
            if x(i,j,2)>b
                x(i,j,2)=b;
            else
            end
        end
    end
end
Fb(1,1,1) = feval(functname,gbest(1,:,1));
if get(handles.disable,'Value')==0
    %%%%%%%%%%%%%%%%%%%%%% Contour Plot %%%%%%%%%%%%%%%%%%%%%%
    axes(handles.axes2);
    axis([a b a b])
    con_m=a:0.1:b;
    con_n=con_m;
    [con_m,con_n]=meshgrid(con_m,con_n);
    for q=1:length(con_m(1,:))
        for z=1:length(con_n(1,:))
            r(q,z)= feval(functname,[con_m(q,z),con_n(q,z)]);
        end
    end
    r_save=r;
    [c,h]=contour(con_m,con_n,r_save,10);
    xlabel('X1')
    ylabel('X2')
    title('Search State')
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end

for j=2:pso_para.itmax
```

```
    % Calculation of new positions
    F(:,1,j) = feval(functname,x(:,:,j));
    [C,I]=min(abs(F(:,1,j)));
    B(1,1,j)=C;
    for i=1:D
        gbest(1,i,j)=x(I,i,j);
    end
    Fb(1,1,j) = feval(functname,gbest(1,:,j));
    [C,I]=min(Fb(1,1,:));
    if Fb(1,1,j)<=C
        for k=1:D
            gbest(1,k,j)=gbest(1,k,j);
        end
    else
        for m=1:D
            gbest(1,m,j)=gbest(1,m,I);
        end
    end
    %Matrix composed of gbest vector
    for p=1:pso_para.N
        for r=1:D
            G(p,r,j)=gbest(1,r,j);
        end
    end
    Fbest(1,1,j) = feval(functname,G(1,:,j));
    for i=1:pso_para.N;
        [C,I]=min(F(i,1,:));
        if  F(i,1,j)<=C
            pbest(i,:,j)=x(i,:,j);
        else
            pbest(i,:,j)=x(i,:,I);
        end
    end
    V(:,:,j+1)= W(j)*V(:,:,j) + pso_para.c1*rand*(pbest(:,:,j)-
x(:,:,j)) + pso_para.c2*rand*(G(:,:,j)-x(:,:,j));
    x(:,:,j+1)=x(:,:,j)+V(:,:,j+1);
    for k=1:pso_para.N
        for m=1:D
            if x(k,m,j+1)<a
                x(k,m,j+1)=a;
            else
                if x(k,m,j+1)>b
                    x(k,m,j+1)=b;
                else
                end
            end
        end
    end
    if get(handles.disable,'Value')==0
        set(gcf,'Doublebuffer','on');
```

```
        %%%%%%%%% Display to the ListBox%%%%%%%%%%
        ResultStr(1) = [{['Fbest','1',' ','=',' ',
num2str(Fbest(1,1,1)),'
','Gbest','1','=','(',num2str(gbest(1,1,1)),',',num2str(gbest(1,2,1)
),')']}];
        ResultStr(j) = [{['Fbest',num2str(j),'  ','=','  ',
num2str(Fbest(1,1,end)),'
','Gbest',num2str(j),'=','(',num2str(gbest(1,1,end)),',',num2str(gbe
st(1,2,end)),')']}];
        set(handles.Result_window, 'String', ResultStr);
        %%%%%%%%% end of Display %%%%%%%%%%%%%%%%%

        %%%%%%%%%%%%%%%%  AXE-1  %%%%%%%%%%%%%%%%
        axes(handles.axes1);
        cla;
        set(gca,'xlim',[0 pso_para.itmax],'ylim',[0 Fbest(1,1,1)]),
        plot(Fbest(:),'r-')
        if j<=pso_para.itmax/2
            text(j,Fbest(1,1,end),['Fbest = ',
num2str(Fbest(1,1,end))],'HorizontalAlignment','Left','VerticalAlign
ment','bottom','EdgeColor','blue','LineWidth',3);
        else
            text(j,Fbest(1,1,end),['Fbest = ',
num2str(Fbest(1,1,end))],'HorizontalAlignment','Right','VerticalAlig
nment','bottom','EdgeColor','blue','LineWidth',3);
        end
        legend('Fbest');
        hold on
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%%%%%%%%%%%%%%%  AXE-2  %%%%%%%%%%%%%%%%
        axes(handles.axes2);
        axis([a b a b])
        [c,h]=contour(con_m,con_n,r_save,10);
        hold on
        plot(pbest(:,1,j),pbest(:,2,j),'r*')
        xlabel('X1')
        ylabel('X2')
        drawnow
        hold off
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
end
if get(handles.disable,'Value')==1
    ResultStr = [{['Fbest  ','=','  ', num2str(Fbest(1,1,end)),'
','Gbest','=','(',num2str(gbest(1,1,end)),',',num2str(gbest(1,2,end)
),')']}];
    set(handles.Result_window, 'String', ResultStr);
end
set(handles.start,'Enable','on');
set(handles.pause,'String','PAUSE','Enable','off');
```

9. Appendix 3: eldrun.m

```
cla;
set(handles.start,'Enable','off');
set(handles.pause,'String','PAUSE','Enable','on');
set(handles.text14,'String','Generation Output');
functnames = get(handles.select_func,'String');
functname = functnames{get(handles.select_func,'Value')};
wmax = str2double(get(handles.wmax, 'String'));
wmin = str2double(get(handles.wmin, 'String'));
c1 = str2double(get(handles.c1, 'String'));
c2 = str2double(get(handles.c2, 'String'));
N = str2double(get(handles.N, 'String'));
itmax = str2double(get(handles.itmax, 'String'));
pso_para = getappdata(gcbf, 'metricdata');
pso_para.wmax = wmax;
pso_para.wmin = wmin;
pso_para.c1 = c1;
pso_para.c2 = c2;
pso_para.N = N;
pso_para.itmax = itmax;
setappdata(gcbf, 'metricdata', pso_para);

[Gen,Demand]=feval(functname);

%Initialization of PSO parameters
D=size(Gen,1); % Dimension (Number of Generator)
CR = 0.5;
for iter=1:pso_para.itmax
    W(iter)= pso_para.wmax-((pso_para.wmax-
pso_para.wmin)/pso_para.itmax)*iter;
end
%Initialization of positions of agents
%agents are initialized between P_min,P_max randomly
for i=1:D
    P_min(i) = Gen(i,6);     % P_min
    P_max(i) = Gen(i,7);     % P_max
end
% Constraints handling
for i=1:pso_para.N
    yes=1;
    while yes==1
        p=randperm(D);

        for n=1:D-1
            g = p(n);
            x(i,g) = P_min(g) +  (P_max(g)-P_min(g)) * rand;
            A(n) = x(i,g);
        end
        SUM=0;
        for f=1:D-1
```

```
            SUM = SUM + A(f);
        end
        A(D) = Demand - SUM;
        g=p(D);
        if A(D) < P_min(g)
            A(D) = P_min(g);
            ok=0;
        else
            if A(D) > P_max(g)
                A(D) = P_max(g);
                ok=0;
            else
                ok=1;
                yes=0;
            end
        end

        L=1;
        while ok==0
            A(L) = Demand -(sum(A(:))-A(L));
            if A(L) < P_min(p(L))
                A(L) = P_min(p(L));
                ok=0;
                L = L+1;
                if L==D+1
                    ok=1;
                    yes=1;
                else
                end
            else
                if A(L) > P_max(p(L))
                    A(L) = P_max(p(L));
                    ok=0;
                    L= L+1;
                    if L==D+1
                        ok=1;
                        yes =1;
                    else
                    end
                else
                    ok=1;
                    yes=0;
                end
            end
        end
    end
    for k=1:D
        x(i,p(k))=A(k);
    end
end
%Initialization of velocities of agents
```

```
%Between V_min , V_max, (which can also be started from zero)
for i=1:pso_para.N
    for j=1:D
        m(j) = Gen(j,6) - x(i,j);  %V_min
        n(j) = Gen(j,7) - x(i,j);  %V_max
        V(i,j) = m(j) + (n(j)-m(j)) * rand;
    end
end
% End of Initialization
% Function to be minimized.
for i=1:pso_para.N;
    for j=1:D;
        Cost(i,j) = Gen(j,1) + Gen(j,2)*x(i,j) + Gen(j,3)*x(i,j).^2
+ abs(Gen(j,4)*sin(Gen(j,5)*(Gen(j,6)-x(i,j))));
    end
    F(i,1) = sum(Cost(i,:)); % Total Cost
end
pbest=x;
[C,I]=min(abs(F(:,1)));
B(1,1)=C;
XX(1,1)=I;
gbest(1,:)=x(I,:);
Gen_sum(1,1) = sum(gbest(1,:));
%Matrix composed of gbest vector
for j=1:D;
    Cost_Best(1,j) =
Gen(j,1)+Gen(j,2)*gbest(1,j)+Gen(j,3)*gbest(1,j).^2
+abs(Gen(j,4)*sin(Gen(j,5)*(Gen(j,6)-gbest(1,j))));
end
Fbest(1,1) = sum(Cost_Best(1,:)); % Total Cost
% Constraints handling
for i=1:pso_para.N
    yes=1;
    while yes==1
        p=randperm(D);
        for n=1:D-1
        g = p(n);
        V(i,g) = W(1) * V(i,g) + c1*rand*(pbest(i,g)-x(i,g)) +
c2*rand*(gbest(1,g)-x(i,g));
        x(i,g)=x(i,g) + V(i,g);
        if rand<=CR
            x_adj(i,g) = x(i,g);
        else
            x_adj(i,g) = pbest(i,g);
        end
        A(n) = x_adj(i,g);
        if A(n) < P_min(g)
            A(n) = P_min(g);
        else
            if A(n) > P_max(g)
                A(n) = P_max(g);
```

```
            else
            end
        end
    end
    SUM=0;
    for u=1:D-1
        SUM = SUM + A(u);
    end
    A(D) = Demand - SUM;

    g=p(D);
    if A(D) < P_min(g)
        A(D) = P_min(g);
        ok=0;
    else
        if A(D) > P_max(g)
            A(D) = P_max(g);
            ok=0;
        else
            ok=1;
        end
        yes=0;
    end

    L=1;
    while ok==0
        A(L) = Demand -(sum(A(:))-A(L));
        if A(L) < P_min(p(L))
            A(L) = P_min(p(L));
            ok=0;
            L = L+1;
            if L==D+1
                ok=1;
                yes=1;
            else
            end
        else
            if A(L) > P_max(p(L))
                A(L) = P_max(p(L));
                ok=0;
                L= L+1;
                if L==D+1
                    ok=1;
                    yes =1;
                else
                end
            else
                ok=1;
            end

            yes=0;
```

```
            end
        end
    end
    for k=1:D
        x_adj(i,p(k))=A(k);
    end
end

for j=2:pso_para.itmax
    % Calculation of new positions
    for i=1:pso_para.N
        for k=1:D
            Cost(i,k) =
Gen(k,1)+Gen(k,2)*x_adj(i,k)+Gen(k,3)*x_adj(i,k).^2
+abs(Gen(k,4)*sin(Gen(k,5)*(Gen(k,6)-x_adj(i,k))));
        end
        F(i,j) = sum(Cost(i,:)); % Total Cost
    end

    for i=1:pso_para.N
        [C,I]=min(F(i,:));
        if  F(i,j)<=C
            pbest(i,:)=x_adj(i,:);
        else
        end
    end
    for i=1:pso_para.N
        for k=1:D
            Cost_pbest(i,k) =
Gen(k,1)+Gen(k,2)*pbest(i,k)+Gen(k,3)*pbest(i,k).^2
+abs(Gen(k,4)*sin(Gen(k,5)*(Gen(k,6)-pbest(i,k))));
        end
        F_pbest(i,1) = sum(Cost_pbest(i,:)); % Total Cost
    end
    [C,I]=min(F_pbest(:,1));
    for k=1:D
        gbest(1,k)=pbest(I,k);
    end
    Gen_sum(j,1) = sum(gbest(1,:));
    Fbest(j,1) = C;
% Constraints handling
    for i=1:pso_para.N
        yes=1;
        while yes==1
            p=randperm(D);
            for n=1:D-1
                g = p(n);
                V(i,g) = W(j) * V(i,g) + c1*rand*(pbest(i,g)-x(i,g))
+ c2*rand*(gbest(1,g)-x(i,g));
                x(i,g) = x(i,g) + V(i,g);
                if rand<=CR
```

```
            x_adj(i,g) = x(i,g);
        else
            x_adj(i,g) = pbest(i,g);
        end
        A(n) = x_adj(i,g);
        if A(n) < P_min(g)
            A(n) = P_min(g);
        else
            if A(n) > P_max(g)
                A(n) = P_max(g);
            else
            end
        end
    end
    SUM=0;
    for f=1:D-1
        SUM = SUM + A(f);
    end
    A(D) = Demand - SUM;

    g=p(D);
    if A(D) < P_min(g)
        A(D) = P_min(g);
        ok=0;
    else
        if A(D) > P_max(g)
            A(D) = P_max(g);
            ok=0;
        else
            ok=1;
            yes=0;
        end

    end
    L=1;
    while ok==0
        A(L) = Demand -(sum(A(:))-A(L));
        if A(L) < P_min(p(L))
            A(L) = P_min(p(L));
            ok=0;
            L = L+1;
            if L==D+1
                ok=1;
                yes=1;
            else
            end
        else
            if A(L) > P_max(p(L))
                A(L) = P_max(p(L));
                ok=0;
                L= L+1;
```

```
                        if L==D+1
                            ok=1;
                            yes=1;
                        else
                        end
                    else
                        ok=1;
                        yes=0;
                    end
                end
            end
        end
        for k=1:D
            x_adj(i,p(k))=A(k);
        end
    end

    if get(handles.disable,'Value')==0
        set(gcf,'Doublebuffer','on');
        %%%%%%%%% Display to the ListBox%%%%%%%%%%
        ResultStr(1) = [{['Total Generation Cost    at Iteration
','1',' ','=',' ', num2str(Fbest(1,1))]}];
        ResultStr(j) = [{['Total Generation Cost    at Iteration
',num2str(j),'  ','=','  ', num2str(Fbest(end,1))]}];
        set(handles.Result_window, 'String', ResultStr);
        %%%%%%%%% end of Display %%%%%%%%%%%%%%%%%

        %%%%%%%%%%%%%%%%  AXE-1  %%%%%%%%%%%%%%%%%
        axes(handles.axes1);
        cla;
        set(gca,'xlim',[0 pso_para.itmax],'ylim',[0 Fbest(1,1)]),
        plot(Fbest(:),'r-')
        if j<=pso_para.itmax/2
            text(j,Fbest(end,1),['Fbest = ',
num2str(Fbest(end,1))],'HorizontalAlignment','Left','VerticalAlignme
nt','bottom','EdgeColor','blue','LineWidth',3);
        else
            text(j,Fbest(end,1),['Fbest = ',
num2str(Fbest(end,1))],'HorizontalAlignment','Right','VerticalAlignm
ent','bottom','EdgeColor','blue','LineWidth',3);
        end
        legend('Fbest');
        hold on
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%%%%%%%%%%%%%%%  AXE-2  %%%%%%%%%%%%%%%%
        axes(handles.axes2);
        axis([0 D+1 0 max(Gen(:,7))+50])
        bar(Gen(:,7),'r')
        hold on
        bar(gbest,'w')
```

```
        drawnow
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    else
    end
end
if get(handles.disable,'Value')==1
    cla;
    ResultStr = [{['Total Generation Cost ','=','  ',
num2str(Fbest(end,1))]}];

    set(handles.Result_window, 'String', ResultStr);
end
%%%%%%%%%%%%%%%%%  AXE-2  %%%%%%%%%%%%%%%%
axes(handles.axes2);
axis([0 D+1 0 max(Gen(:,7))+50])
bar(Gen(:,7),'r')
hold on
bar(gbest(end,:),'w')
drawnow
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(handles.start,'Enable','on');
set(handles.pause,'String','PAUSE','Enable','off');
```

10. References

A. J. Wood and B. F. Wollenbergy (1984). Power Genration, Operation, and Control, *New York: Wiley*.

C. E. Lin and G. L. Viviani (1984). Hierarchical economic dispatch for piecewise quadratic cost functions, *IEEE Trans. Power App. System*, vol. PAS-103, no. 6, pp. 1170-1175.

D. A. Wiegmann, G. R. Essenberg, T. J. Overbye, and Y. Sun (2005). Human Factor Aspects of Power System Flow Animation, *IEEE Trans. Power Syst.*, vol. 20, no. 3, pp. 1233-1240.

D. A. Wiegmann, T. J. Overbye, S. M. Hoppe, G. R. Essenberg, and Y. Sun (2006). Human Factors Aspects of Three-Dimensional Visualization of Power System Information, *IEEE Power Eng. Soci. Genral Meeting*, pp. 7.

D. C. Walters and G. B. Sheble (1993). Genetic algorithm solution of economic dispatch with the valve point loading, *IEEE Trans. Power Syst.*, vol. 8, pp. 1325-1332.

H. T. Yang, P. C. Yang, and C. L. Huang (1996). Evolutionary programming based economic dispatch for units with nonsmooth fuel cost functions, *IEEE Trans. Power Syst.*, vol. 11, no. 1, pp. 112-118.

H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi (2000). A particle swarm optimization for reactive power and voltage control considering voltage security assessment, *IEEE Trans. Power System*, vol. 15, pp. 1232-1239.

J. B. Park, K. S. Lee, J. R. Shin, and K. Y. Lee (2005). A particle swarm optimization for economic dispatch with nonsmooth cost functions, *IEEE Trans. Power Syst.*, vol. 20, no. 1, pp. 34-42.

J. H. Park, Y. S. Kim, I. K. Eom, and K. Y. Lee (1993). Economic load dispatch for piecewise quadratic cost function using Hopfield neural network, *IEEE Trans. Power Syst.*, vol. 8, pp. 1030-1038.

J. Kennedy and R. Eberhart (1995). Particle swarm optimization, *Proc. IEE Int. Conf. Neural Networks (ICNN'95)*, vol. IV, Perth, Australia, pp.1942-1948.

J. Kennedy and R. C. Eberhart (2001). Swarm Intelligence. *San Francisco, CA: Morgan Kaufmann*.

K. Y. Lee, A. Sode-Yome, and J. H. Park (1998). Adaptive Hopfield neural network for economic load dispatch, *IEEE Trans. Power Syst.*, vol. 13, pp. 519-526.

K. Y. Lee and M. A. El-Sharkawi, Eds. (2002). Modern Heuristic Optimization Techniques with Applications to Power Systems: *IEEE Power Engineering Society (02TP160)*.

L. S. Coelho and V. C. Mariani (2006). Combining of chaotic differential evolution and quadratic programming for economic dispatch optimization with valve-point effect, *IEEE Trans. Power Syst.*, vol. 21, No. 2.

M. Clerc and J. Kennedy (2002). The particle swarm-expolsion, stability, and convergence in a multidimensional complex space, *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58-73.

N. Sinha, R. Chakrabarti, and P. K. Chattopadhyay (2003). Evolutionary programming techniques for economic load dispatch, *IEEE Trans. on Evolutionary Computations*, Vol. 7, No. 1, pp. 83-94.

T. A. A. Victoire and A. E. Jeyakumar (2004). Hybrid PSO-SQP for economic dispatch with valve-point effect, *Electric Power Syst. Research*, vol. 71, pp. 51-59.

T. A. A. Victoire and A. E. Jeyakumar (2005). Reserve constrained dynamic dispatch of units with valve-point effects, *IEEE Trans. Power Syst.*, vol. 20, No. 3, pp. 1273-1282.

T. J. Overbye, D. A. Wiegmann, A. M. Rich, and Y. Sun (2003). Human Factor Aspects of Power System Voltage Contour Visualizations, *IEEE Trans. Power Syst.*, vol. 18, no. 1, pp. 76-82.

W. M. Lin, F. S. Cheng, and M. T. Tasy (2002). An improved Tabu search for economic dispatch with multiple minima, *IEEE Trans. Power Syst.*, vol. 17, pp. 108-112.

Y. M. Park, J. R. Won, and J. B. Park (1998). A new approach to economic load dispatch based on improved evolutionary programming, *Eng. Intell. Syst. Elect. Eng. Commun.*, vol. 6, no. 2, pp. 103-110.

MATLAB - A Ubiquitous Tool for the Practical Engineer

Edited by Prof. Clara Ionescu

ISBN 978-953-307-907-3
Hard cover, 564 pages
Publisher InTech
Published online 13, October, 2011
Published in print edition October, 2011

A well-known statement says that the PID controller is the â€œbread and butterâ€ of the control engineer. This is indeed true, from a scientific standpoint. However, nowadays, in the era of computer science, when the paper and pencil have been replaced by the keyboard and the display of computers, one may equally say that MATLAB is the â€œbreadâ€ in the above statement. MATLAB has became a de facto tool for the modern system engineer. This book is written for both engineering students, as well as for practicing engineers. The wide range of applications in which MATLAB is the working framework, shows that it is a powerful, comprehensive and easy-to-use environment for performing technical computations. The book includes various excellent applications in which MATLAB is employed: from pure algebraic computations to data acquisition in real-life experiments, from control strategies to image processing algorithms, from graphical user interface design for educational purposes to Simulink embedded systems.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen