APPLICATION OF PARTICLE SWARM OPTIMIZATION FOR GENERATION SCHEDULING

A

Major project Report Submitted to

Veer Surendra Sai University of Technology, Burla
in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ELECTRICAL ENGINEERING

Submitted By:

PRAKHAR SHARMA (1702050124)

BIJAYINI DALAI (1704050001)

SUBHA BISWAL (1704050016)

Under Guidance of:

Dr. RAJAT KANTI SAMAL (Assistant professor)

DEPARTMENT OF ELECTRICAL ENGINEERING

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, BURLA

JUNE 2021

Dept. of Electrical Engineering

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY

CERTIFICATE

This is to certify that the major project entitled ***"APPLICATION OF PARTICLE SWARM OPTIMIZATION FOR GENERATION SCHEDULING"*** Submitted by- **Prakhar Sharma (1702050124), Bijayini Dalai (1704050001), Subha Biswal (1704050016)** towards the partial fulfilment of the requirement for the award of degree in Electrical Engineering of VSSUT, Burla. An authentic work carried out by them under the guidance of **Dr. Rajat Kanti Samal.** This is in partial fulfilment of requirement of project report for Bachelor of Technology degree in Electrical Engineering of VSSUT, Burla. No part of this report has been submitted to any other University or Institution for any award of degree or otherwise to the best of my knowledge.

Dr. Rajat Kanti Samal
Project Guide
Department of Electrical Engg.
VSSUT, Burla, Odisha

Dr. Papia Ray
Head of Department
Department of Electrical Engg.
VSSUT, Burla, Odisha.

ACKNOWLEDGEMENT

We express our deepest gratitude to our project guide Dr. Rajat Kanti Samal whose encouragement, guidance and support from the initial to the ultimate level enabled us to develop an understanding of the topic.

Besides, we would like to thank to Dr. Papia Ray, Head of the Electrical Engineering Department, Veer Surendra Sai University of Technology, Burla, for providing her invaluable advice and for providing us with an environment to complete our project successfully.

We are deeply indebted to all faculty members of Electrical Engineering Department, Veer Surendra Sai University of Technology, Burla, for their help in making the project a successful one.

Finally, we take this chance to extend our deep appreciation to our friends, for all their cooperation during the crucial times of the completion of our project.

Prakhar Sharma (1702050124)

Bijayini Dalai (1704050001)

Subha Biswal (1704050016)

ABSTRACT

This report presents the analysis and implementation of the concept and algorithm of **Single Objective Particle Swarm Optimization**. Particle Swarm Optimization (PSO) may be a computational method that optimizes a problem by iteratively trying to enhance a candidate solution with regard to a given measure of quality. It solves an issue by having a population of candidate solutions, here dubbed particles, and moving these particles around within the search-space in line with simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best-known position, but is additionally guided toward the best-known positions within the search-space, which are updated as better positions are found by other particles. This is often expected to manoeuvre the swarm toward the most effective solutions. During this report the algorithm has been used for getting cost and emission minimization for IEEE 30 bus 6 generator system and generation cost of ten generator system considering the valve point effect for fossil fuel fired generator in single objective function PSO algorithm.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Introduction to Optimization: -

Optimization is a vital tool in making decisions and in analysing physical systems. In mathematical words, an optimization problem is that the problem of finding the simplest solution from the set of all feasible solutions. A mathematical optimization algorithm consists of an objective function and a collection of constraints expressed within the variety of a system of equations. Optimization models are used extensively in most areas of decision-making like engineering design, and financial portfolio selection.

Optimization problems are classified in line with the mathematical characteristics of the target function, the constraints, and also the controllable decision variables:

- An objective function that we wish to attenuate or maximize.
- The controllable inputs are the set of decision variables which affect the worth of the target function.
- Constraints are relations between decision variables and also the parameters. a group of constraints allows a number of the choice variables to require on certain values, and exclude others.

The basic aim of the optimization process is to seek out values of the variables that to minimize or maximize the target function while satisfying the constraints. This result's called an optimal solution
[4]

1.1.1 Applications

1. Time tabling
2. Site selection for an industry
3. Production planning, controlling and scheduling
4. Optimal tariff design
5. Design of aircraft and aerospace structures for minimum cost
6. Design of pumps, turbines and heat exchangers for maximum efficiency

1.2 Classification of Optimization Techniques

1.2.1 Mathematical Programming Techniques

- Based on geometrical properties of the problem
  - Simplex Algorithm, Interior Point Algorithm
  - Steepest Decent, Newton’s method, Quasi-Newton
  - Branch & bound, Cutting Planes

1.2.2 Metaheuristic Techniques

- Genetic Algorithm
- Particle Swarm Optimization
- Differential Evolution
- Teaching Learning Based Optimization
- Artificial Bee Colony
- Grey Wolf Optimization
- Simulated Annealing

1.2.3 Other Techniques

- Nadler Mead
- Fibonacci method
- Golden Section Method
- Hooke-Jeeves’ Pattern Search Method
- Powell’s Conjugate Direction method

1.3 Mathematical Programming Techniques

1.3.1 Advantages

- Guaranteed optimal solutions for well-structured problems (LP, MILP, QP)
- Helpful in debottlenecking as it provides reasonable insight into the behaviour of solutions.
- Does not require multiple runs.
- Requires lower computational resources for certain classes of problems.

1.3.2 Drawbacks

- Rigid modelling framework as it requires an explicit model in a definite form.
- Not naturally amenable to multi-objective optimization problems.
- Computational load usually increases significantly with increase in the population size.[3]

1.4 Metaheuristic Ttechniques

- Most of metaheuristic techniques are nature inspired
- Don’t need any information about the physics of the problem
- Problem need not be postulated in conventional equality & inequality form
- Suitable to solve problems which are multimodal, large dimension or discontinuous
- Can solve black-box optimization problems
- Works with a population of potential solutions
- Do not guarantee global optima

1.5 Objectives

a) To study the optimization of Sphere function using PSO algorithm and derive bestfitness and bestsolution of objective function by execution of code.

b) To analyse the four optimization techniques to solve & plot the following benchmark functions: Sphere function, Rosenbrock function, Rastrigin function & Griewank function

c) To run the optimization several times and obtain statistical measures such as mean, median & standard deviation.

d) To minimize the cost of generation and emission satisfying power balance & generation capacity constraints in an IEEE 30 bus 6 generator systems by using **Single Objective PSO algorithm**.

e) To minimize the cost of generation satisfying power balance, generation capacity constraints and also the valve point effect for fossil fuel fired generator in ten generator system.

2. OVERVIEW OF OPTIMIZATION TECHNIQUE

2.1 Particle Swarm Optimization: -

Particle Swarm Optimization may be a stochastic, population-based optimization algorithm. Rather than competition/selection, like say in Evolutionary Computation, PSO makes use of cooperation, consistent with a paradigm sometimes called "swarm intelligence". Such systems are typically made from a population of easy interacting agents with non-centralized control, and inspired by cases which will be found in nature, like ant colonies, bird flocking, fish schooling, animal herding, bacteria moulding, etc. There are many types of PSO including multi objective, constrained and discrete or combinatorial versions and applications have been developed using PSO in many fields.

2.2 Swarm Intelligence

Biologists studied the behaviour of social insects for an extended time. After voluminous years of evolution of these species have developed incredible solutions for a good range of problems. The intelligent solutions to problems naturally emerge from the self-organization and indirect communication of those individuals. Indirect interactions happen between two individuals when one amongest them modifies the environment and therefore the other responds to the new environment at a later time.[3]

2.3 Algorithmic Structure of Normal PSO

PSO performs searching via a swarm of particles that updates from iteration to iteration. To hunt the optimal solution, each particle moves within the direction to its previously best (*pbest*) position and also the global best (*gbest*) position within the swarm. One has

pbest(i,t) = arg min[f(Pi(k))],   i belongs to {1,2,...,Np} & k=1,..,t

gbest(t) = arg min[f(Pi(k))]

where i denotes the particle index, Np the overall number of particles, t is the iteration number, f the fitness function, and P the position. The velocity V and position P of particles are updated by the subsequent equations:

Vi (t+1) = w*Vi(t) + c1*r1*(pbest(i,t)-Pi(t)) + c2*r2*(gbest(t)-Pi(t)),

Pi (t+1) = Pi (t) + Vi (t+1)

where V denotes the velocity vector, w is inertia weight, used for balancing the global exploration and local exploitation, r1 and r2 are uniformly distributed random variables within range [0,1], and c1 and c2 are positive constant parameters called acceleration coefficients.[3]

W*v(t): Momentum part

- Serves as the memory of previous flight.
- Prevents the particle from drastically changing the directions.

C1*r1*(pbest(1, t)-Pi(t)):Cognitive part

- Quantifies the performance of ith particle relative to its past performance.
- Particles are drawn back to its best position.

C2*r2(gbest(t)-Pi(t)): Social part

- Quantifies the performance of ith particle relative to its neighbour.
- Particles are drawn back to its best position determined group.

Significance of Acceleration coefficient (c1 and c2):

This coefficient has been found to extend the efficiency of PSO and also improve its convergence speed

Significance of Inertia Weight ‘w’ (w1 and w2):

It controls the impact of previous velocity in new direction and also helps in balancing exploration and exploitation.

2.4 Benchmark Functions

Benchmark numerical test functions are widely used as performance evaluation tools in metaheuristic literature. This can also be considered that metaheuristic algorithms that perform well on these functions are ready to solve real-life hard optimization problems. Commonly used benchmark functions are:

2.4.1 Sphere Function

$$\boldsymbol{f(x) = \sum_{i=1}^{d} x_i^2}$$

Description:

Dimensions:d

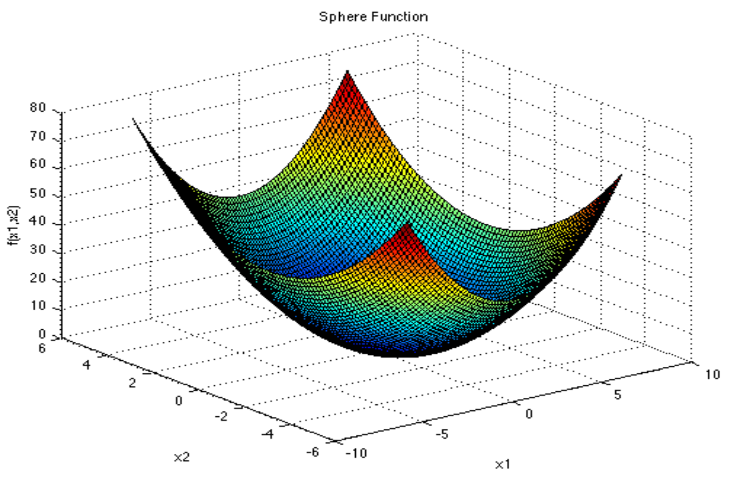
The Sphere function has d local minima except for the global one. It's continuous, convex and unimodal. The Figure 1 shows its two-dimensional form.[2]

Input Domain:

The function is typically evaluated on the hypercube $x_i \in [-5.12, 5.12]$,

for all i = 1, …, d.

Global Minimum:



$\boldsymbol{f(x*) = 0, at\ x* = (0, \dots, 0)}$

Figure 1: Sphere Function Curve

2.4.2 Griewank Function

$$\boldsymbol{f(x) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1}$$

Description:

Dimensions:d

The Griewank function has many widespread local minima, which are regularly distributed. The complexity is shown within the zoomed-in Figure 2.[2]

Input Domain:

The function is sometimes evaluated on the hypercube $x_i \in [-600, 600]$, for all i = 1, …, d.

Global Minimum:

$\boldsymbol{f(x*) = 0, at\ x* = (0, \ldots, 0)}$

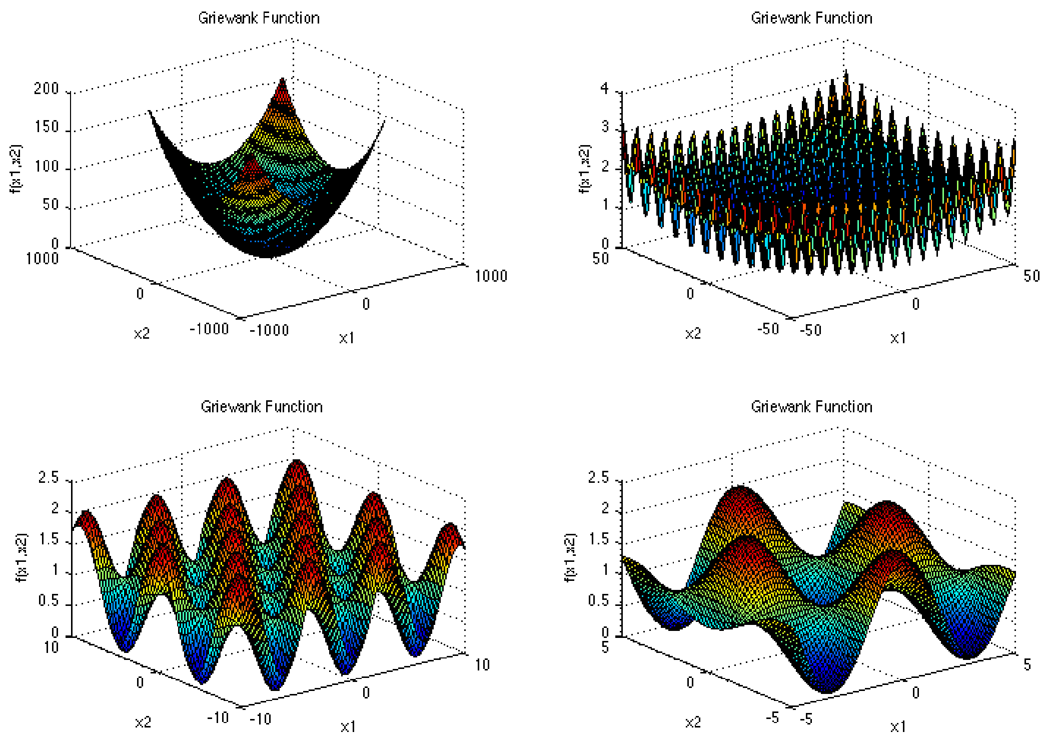


Figure 2: Griewank Function Curve

2.4.3 Rastrigin Function

$$\boldsymbol{f(x) = 10d + \sum_{i=1}^{d}\left[x_i^2 - 10\cos(2\pi x_i)\right]}$$

Description:

Dimensions: d

The Rastrigin function has several local minima. It’s highly multimodal, but locations of the minima are regularly distributed. It’s shown within the in its two-dimensional form in Figure 3.[2]

Input Domain:

The function is sometimes evaluated on the hypercube $x_i \in [-5.12, 5.12]$, for all i = 1, …, d.

Global Minimum:

$\boldsymbol{f(x*) = 0, at\ x* = (0, \dots, 0)}$

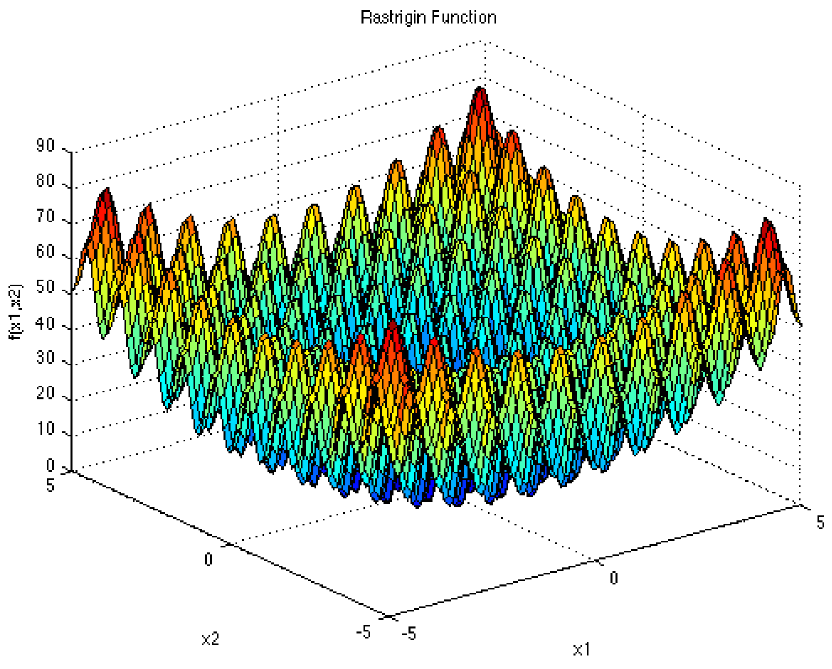


Figure 3: Rastrigin Function Curve

2.4.4 Rosenbrock Function

$$\boldsymbol{f(x) = \sum_{i=1}^{d-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]}$$

Description:

Dimensions: d

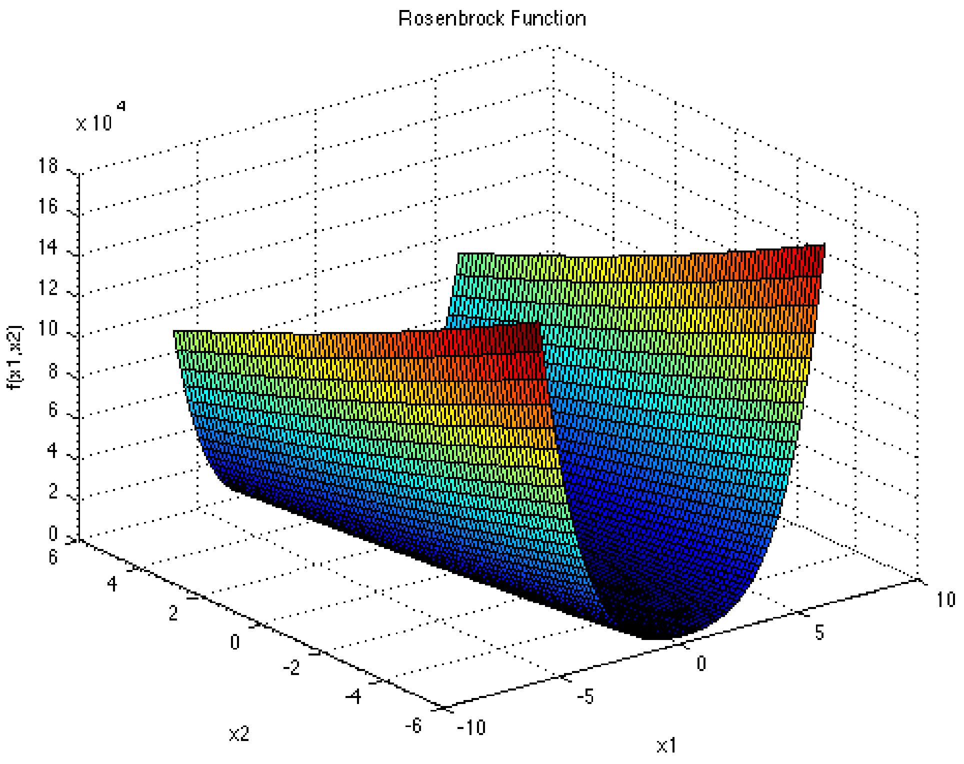
The Rosenbrock function, also cited because the Valley or Banana function, could be a popular test problem for gradient-based optimization algorithms. It’s shown within the Figure 4 in its two-dimensional form.

The function is unimodal, and also the global minimum lies during a narrow, parabolic valley. However, while though this valley is simple to search out, convergence to the minimum is difficult.[2]

Input Domain:

The function is typically evaluated on the hypercube $x_i \in [-5, 10]$, for all i = 1, …, d, although it's going to be restricted to the hypercube $x_i \in [-2.048, 2.048]$, for all i = 1, …, d.

Global Minimum:



$$\boldsymbol{f(x*) = 0, at\ x* = (0, \ldots, 0)}$$

Figure 4: Rosenbrock Function Curve

3. RESULTS AND DISCUSSIONS

3.1 Results for Benchmark Functions

3.1.1 Sphere function

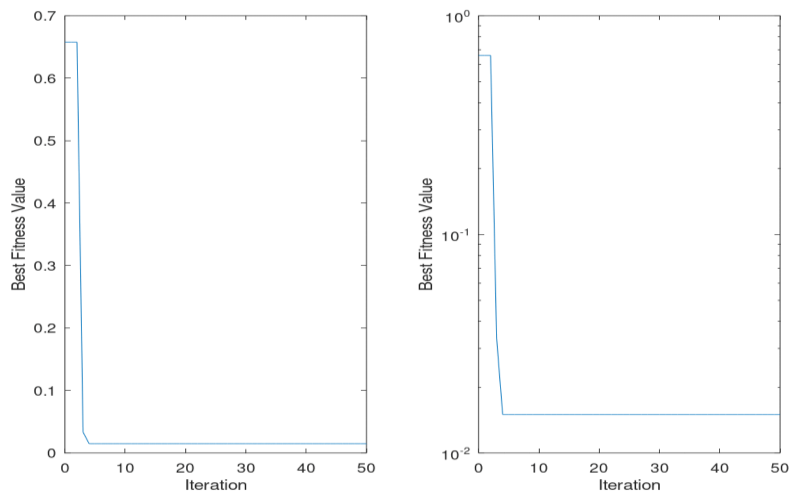


Figure 5: Sphere Function Curve

3.1.2 Rosenbrock function

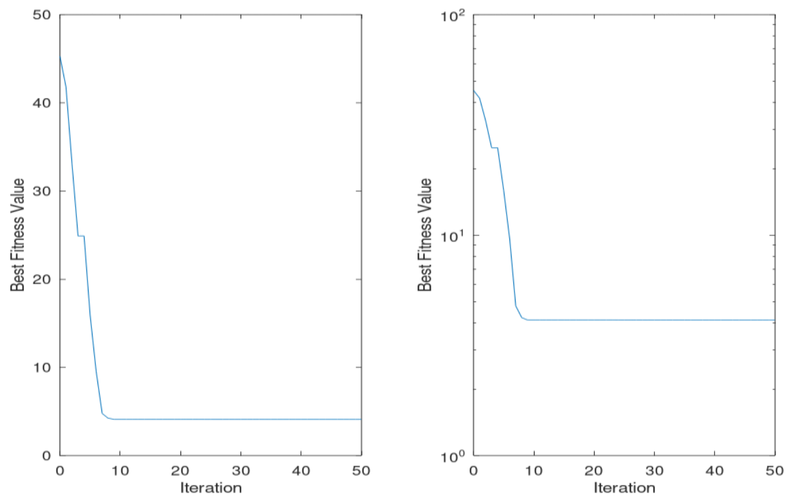


Figure 6: Rosenbrock Function Curve

3.1.3 Rastrigin function

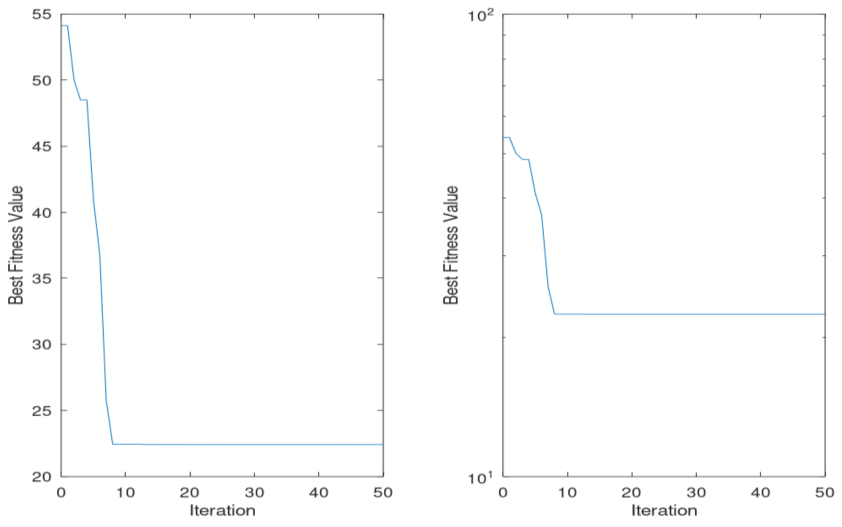


Figure 7: Rastrigin Function Curve

3.1.4 Griewank function

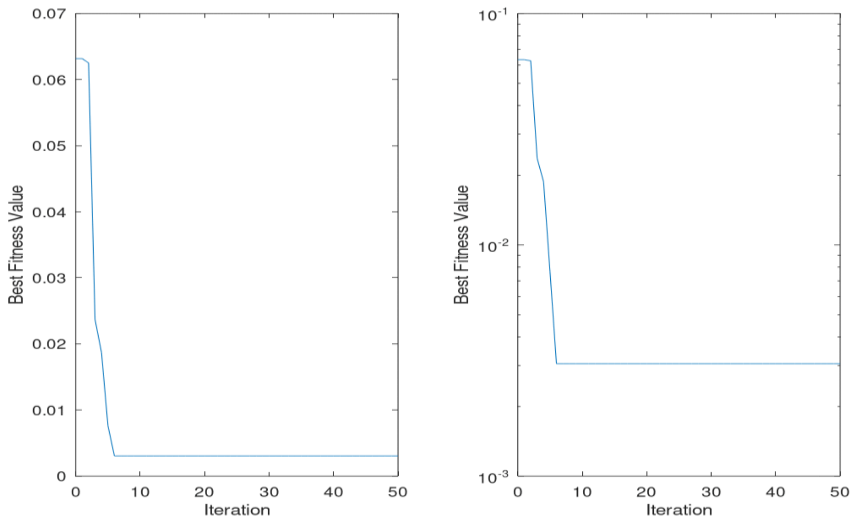


Figure 8: Griewank Function Curve

3.2 Results for Evaluating Statistical Values using PSO

3.2.1 Rastrigin Function

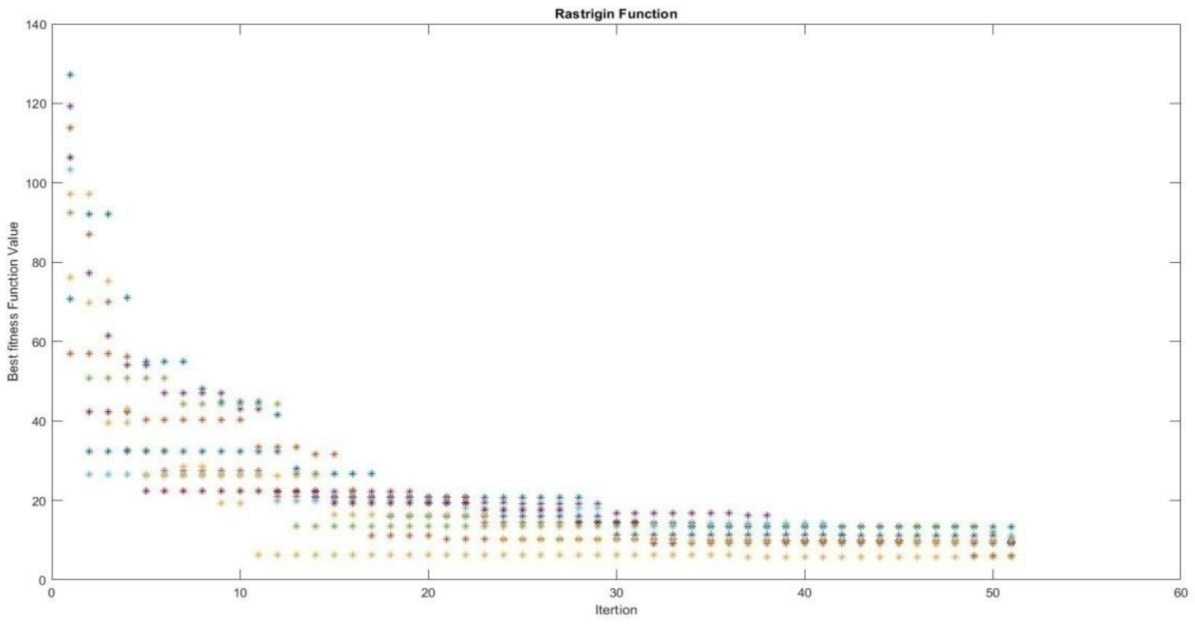


Figure 9: Best Fitness Function Value vs Iteration Curve for Rastrigin Function

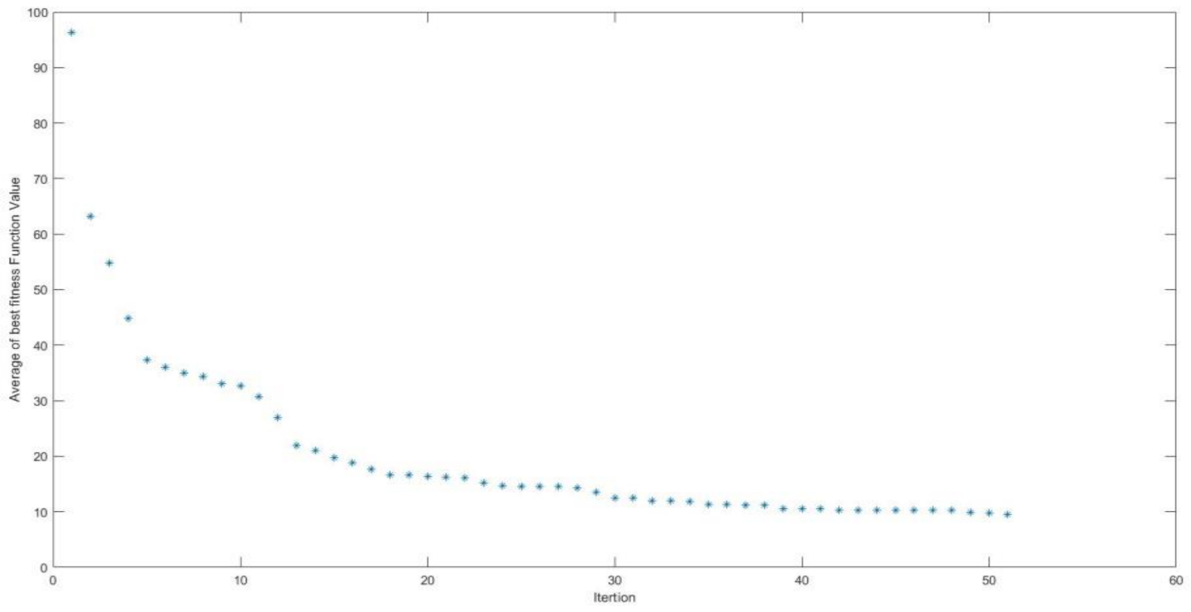


Figure 10: Average of Best Fitness function value for Rastrigin Function

3.2.2 Sphere function

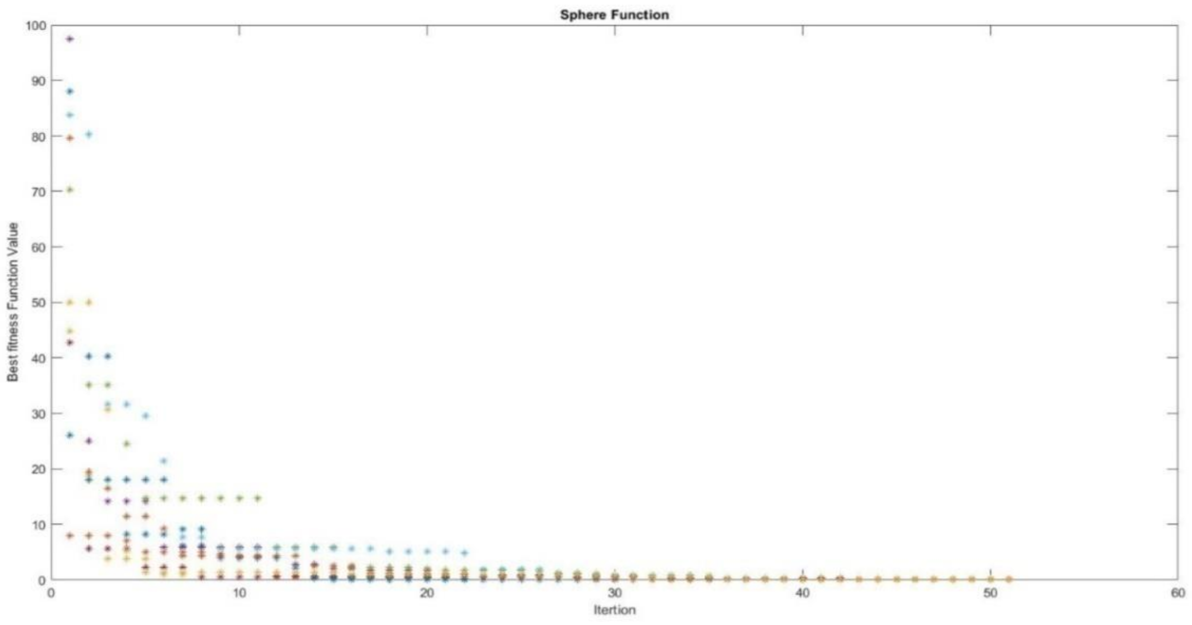


Figure 11: Best Fitness Function Value vs Iteration Curve for Sphere Function

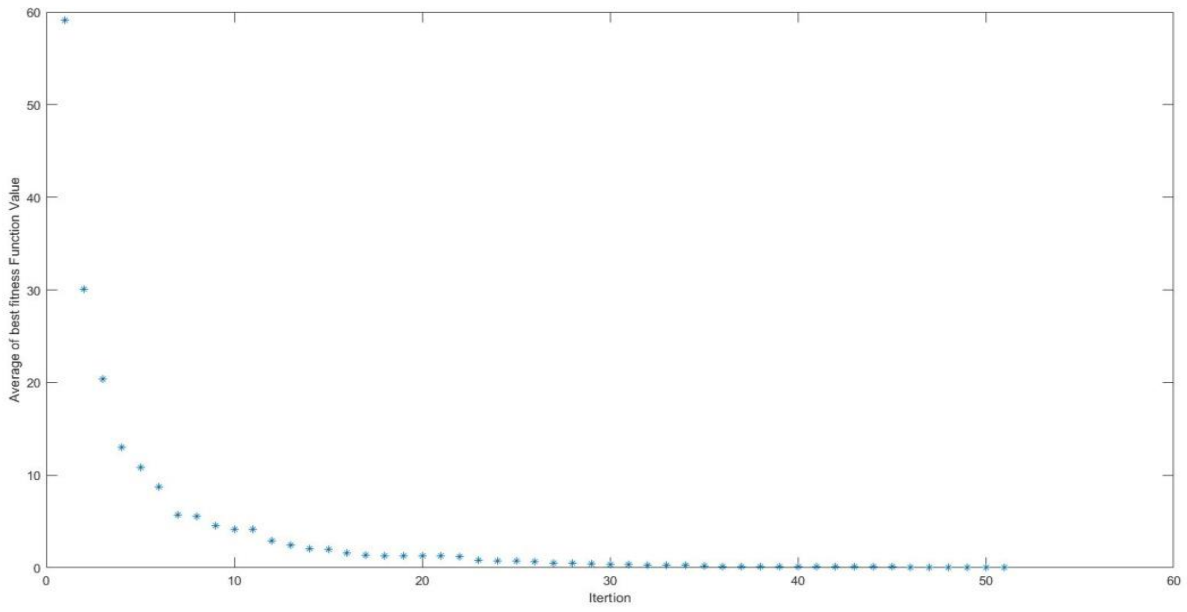


Figure 12: Average of Best Fitness function value for Sphere Function

3.2.3 Rosenbrock function

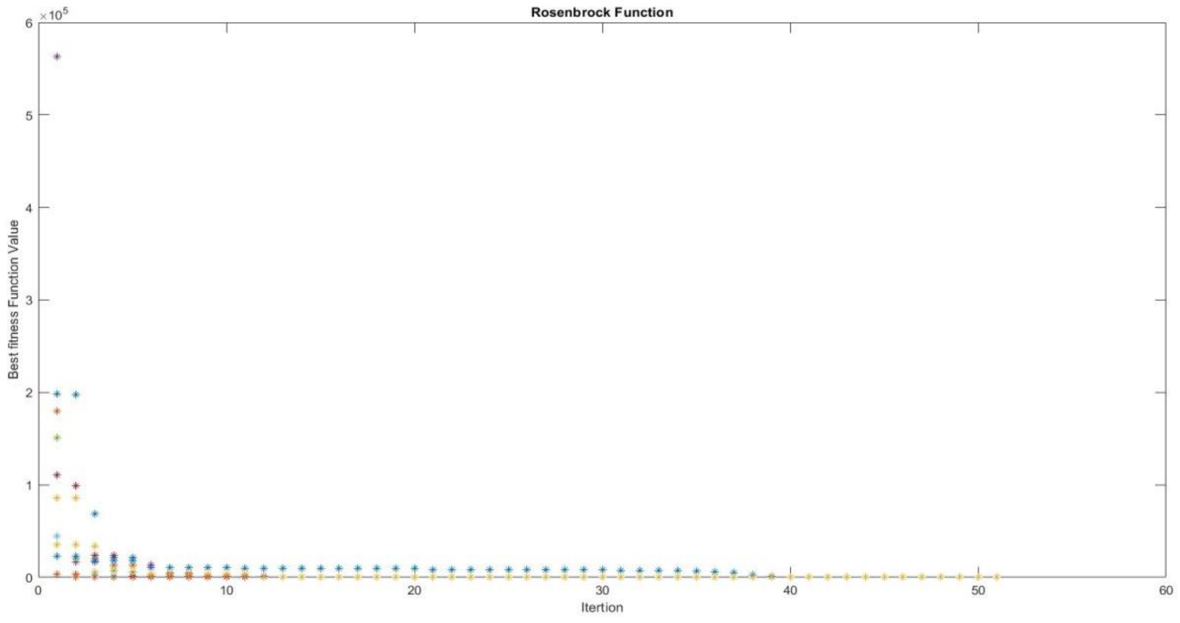


Figure 13: Best Fitness Function Value vs Iteration Curve for Rosenbrock Function

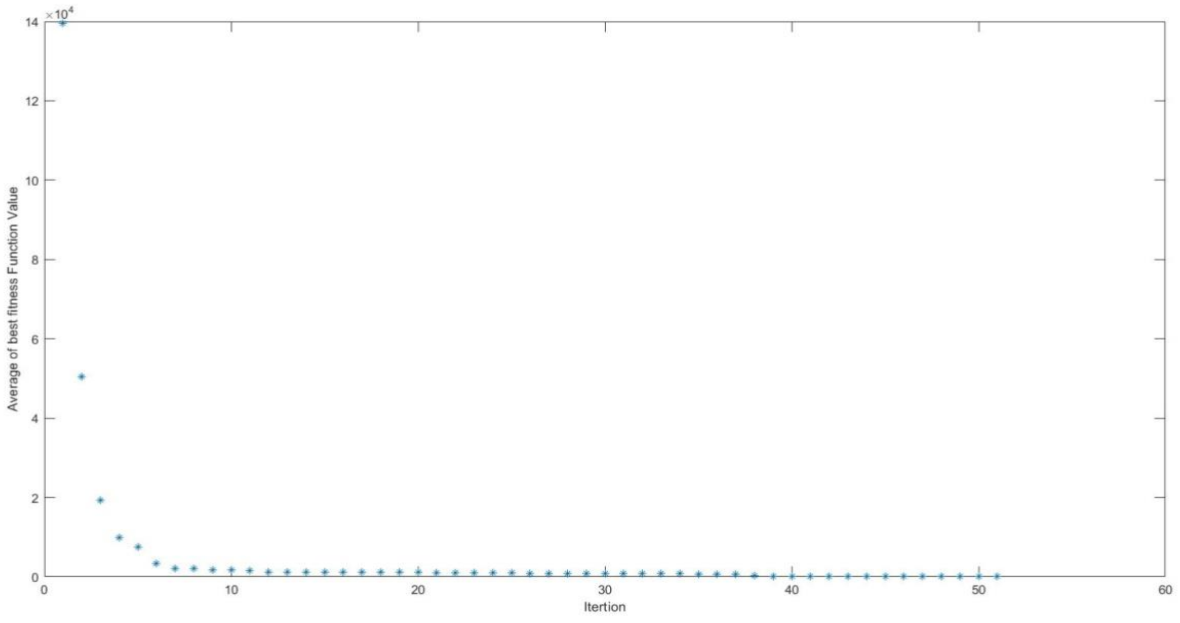


Figure 14: Average of Best Fitness function value for Rosenbrock Function

3.2.4 Griewank function

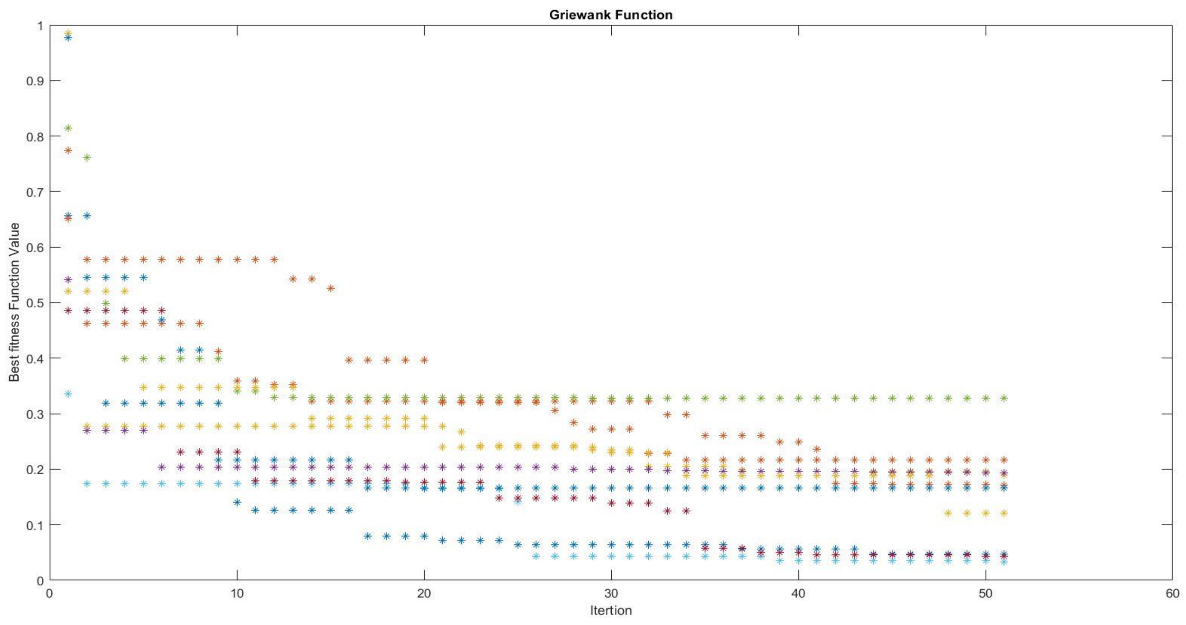


Figure 15: Best Fitness Function Value vs Iteration Curve for Griewank Function

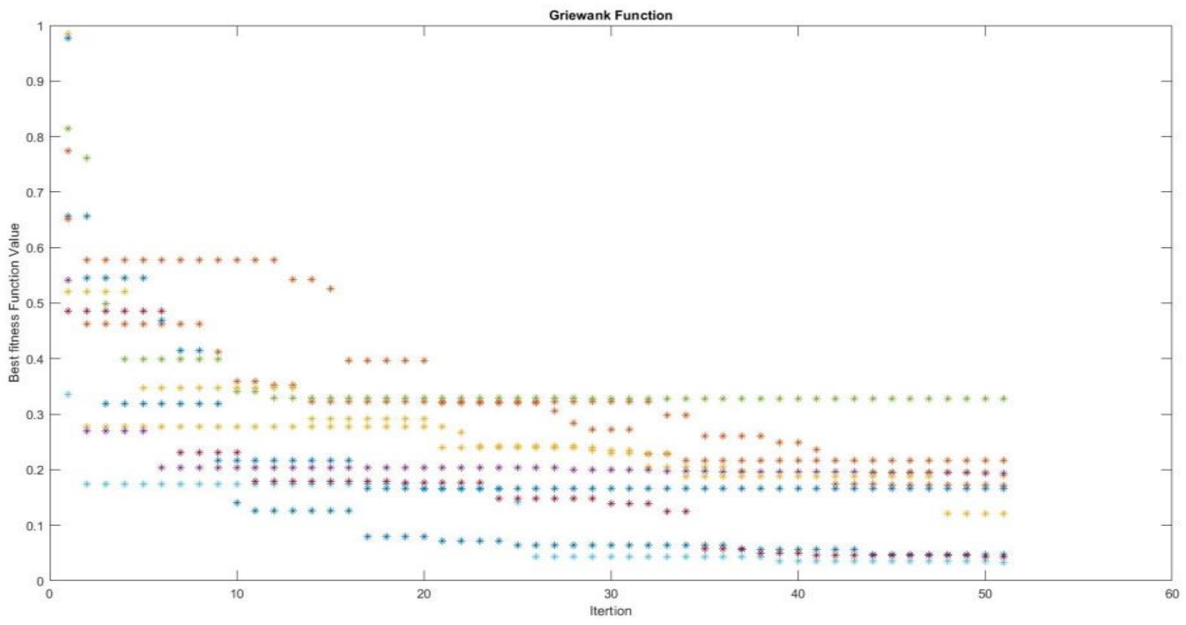


Figure 16: Average of Best Fitness function value for Griewank Function

3.3 Results for IEEE 30 Bus 6 Generator System

3.3.1 Total fuel cost minimization

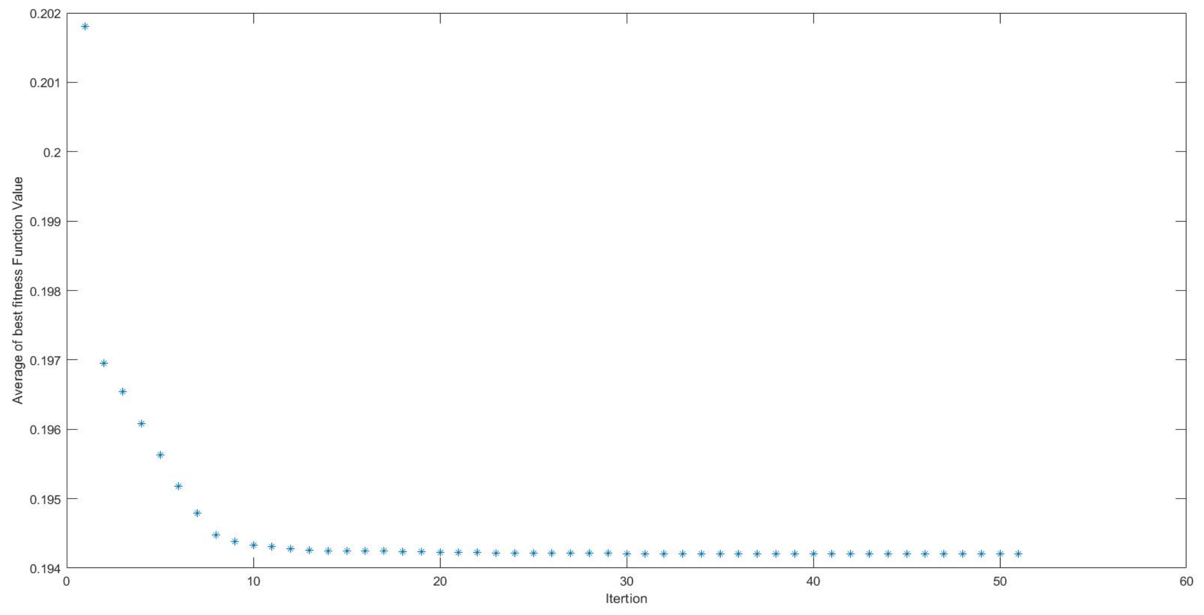


Figure 17: Average of Best Fitness function value for Fuel Cost Minimization

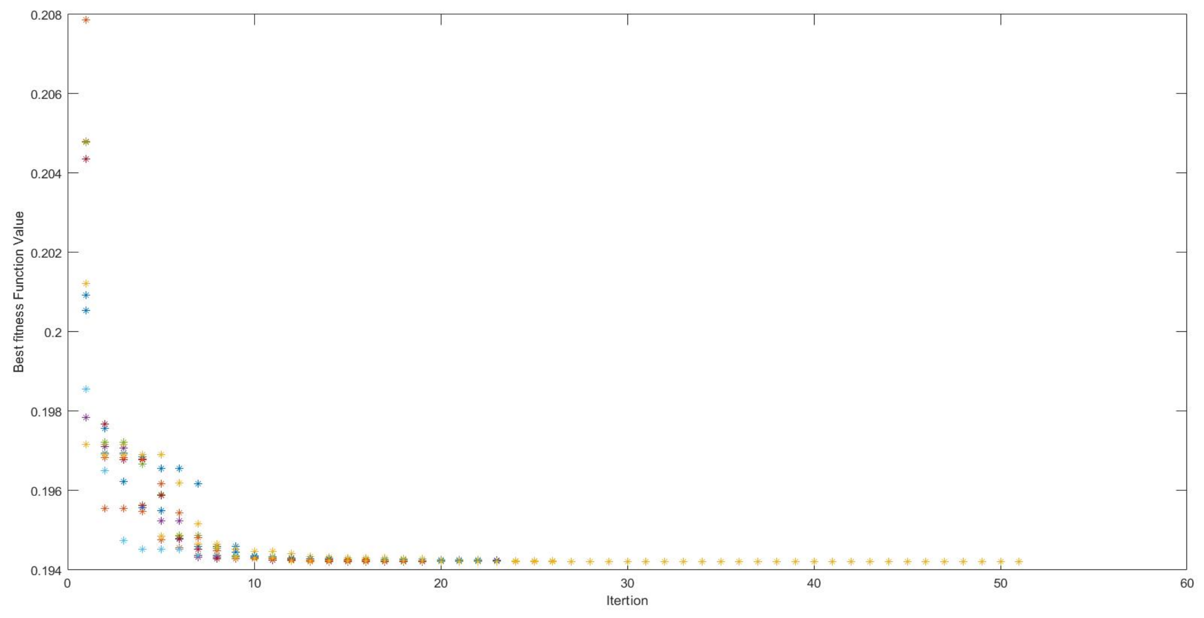


Figure 18: Best Fitness Function Value vs Iteration Curve for Fuel Cost Minimization

3.3.2 Total emission minimization

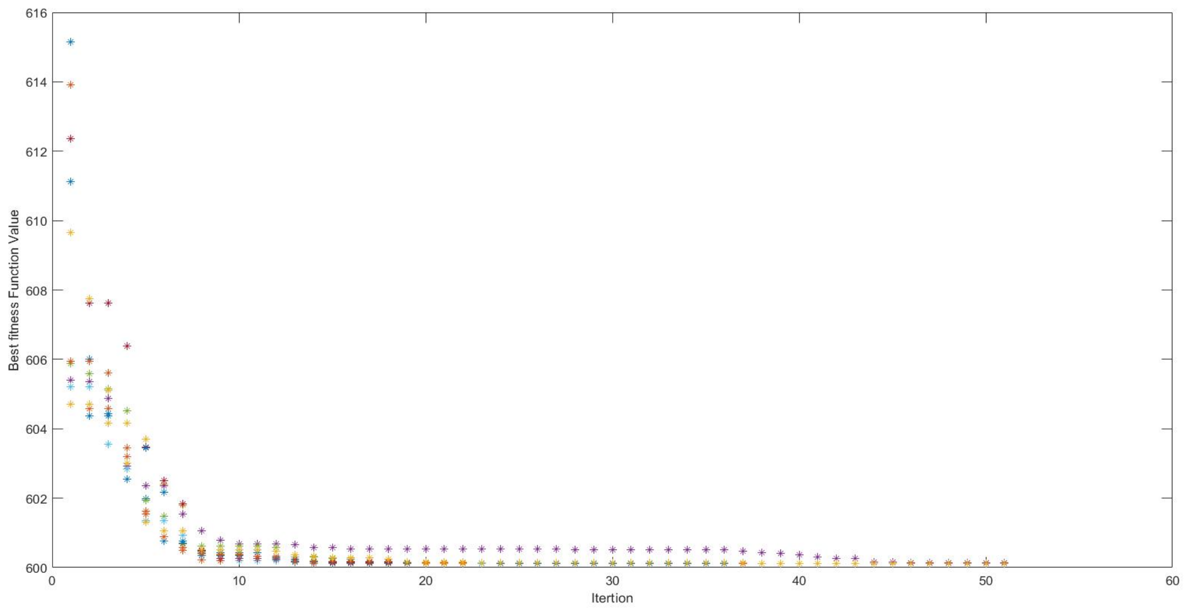


Figure 19: Best Fitness Function Value vs Iteration Curve for Total Emission Minimization

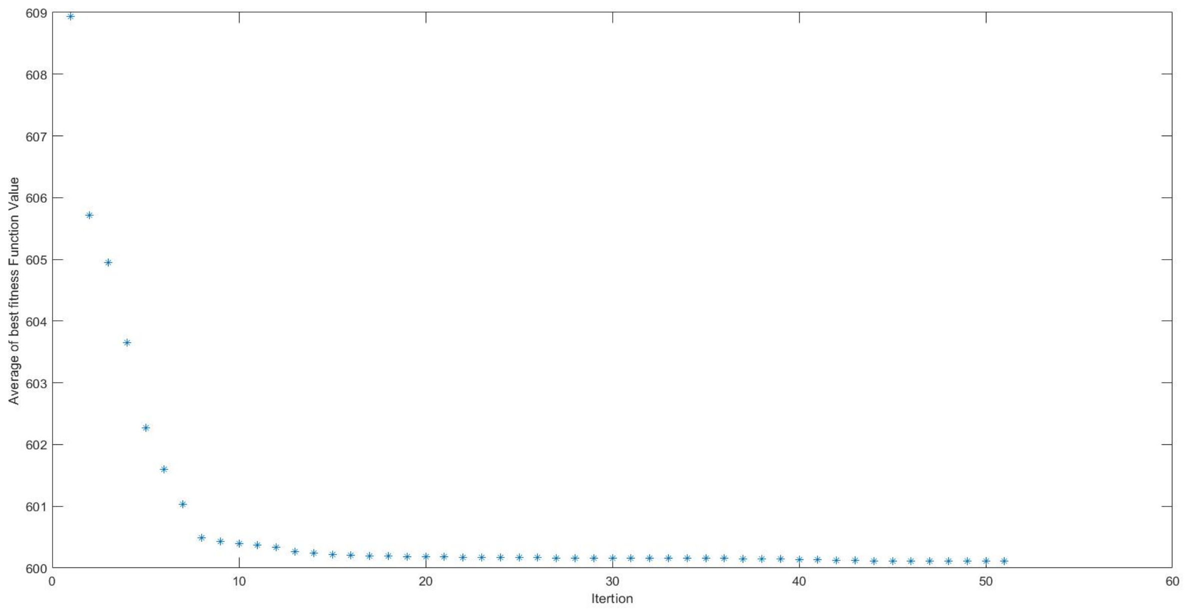


Figure 20: Average of Best Fitness function value for Total Emission Minimization

3.4 Results for 10 Generator System Considering Valve-Point Effect

3.4.1 Total fuel cost minimization

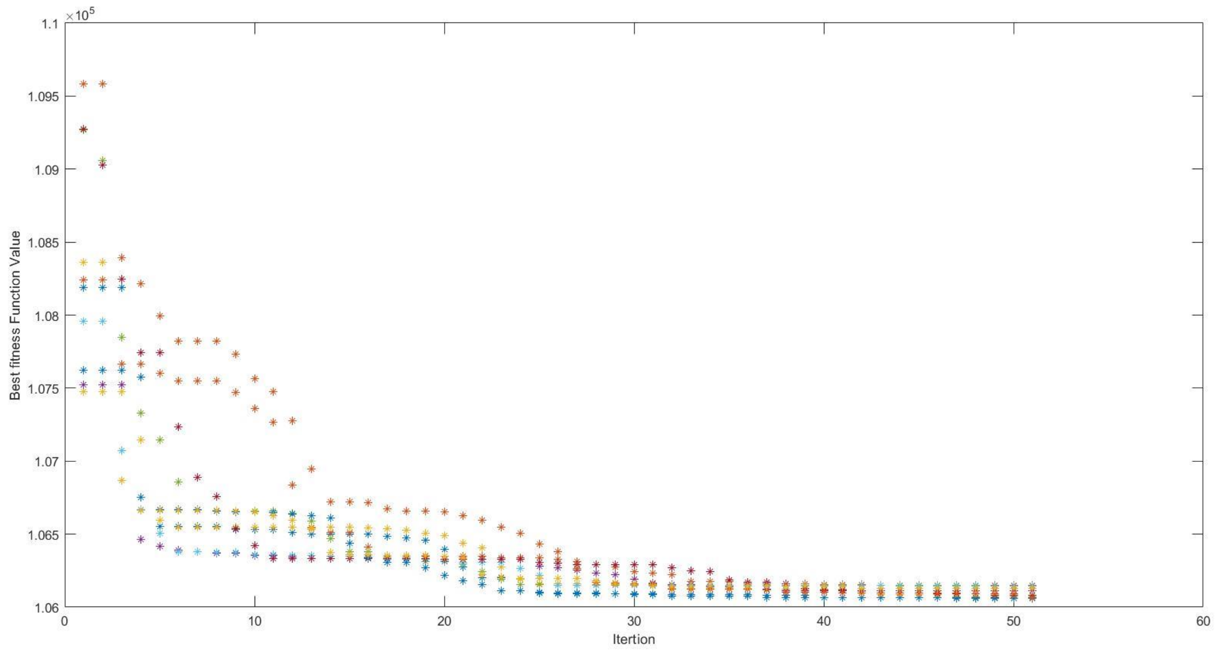


Figure 21: Best Fitness Function Value vs Iteration Curve for Fuel Cost Minimization

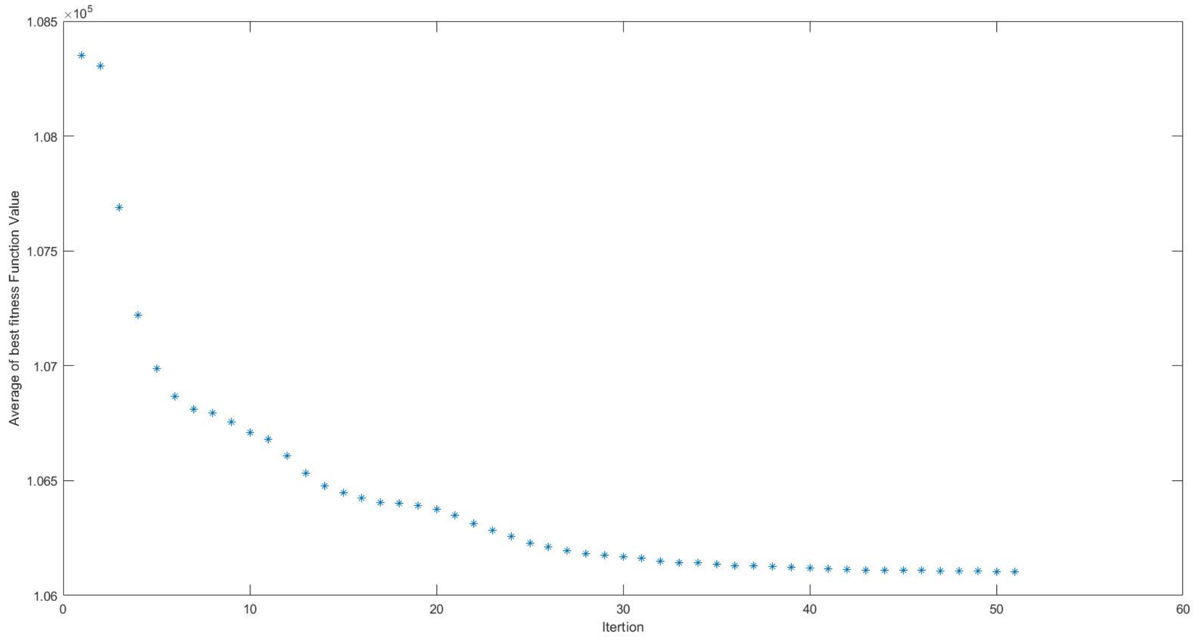


Figure 22: Average of Best Fitness function value for Total Emission Minimization

3.5 Result Summary

3.5.1 Results for the four benchmark functions used are as follows:

Funtion Name	Sphere Function	Rosenbrock Function	Rastrigin function	Griewank Function
Best fitness Value	0.0891	12.7892	8.8751	0.0998

3.5.2 Statistical results for the four benchmark functions used are as follows:

Functions	Best Fitness values				
	Minimum	Maximum	Mean	Median	Standard Deviation
Sphere	0.0103	0.1992	0.0441	0.0255	0.0585
Rosenbrock	1.2058	150.5213	46.8545	11.2098	63.3420
Rastrigin	5.6465	13.2991	9.4504	9.561	2.2370
Griewank Function	0.0331	0.3276	0.1511	0.1692	0.0923

3.5.3 Results for IEEE 30 bus 6 generator system:

Functions	Best Fitness values				
	Minimum	Maximum	Mean	Median	Standard Deviation
Total Fuel Cost	600.11	600.13	600.11	600.11	0.0044013
Total Emission	0.1942	0.19421	0.1942	0.1942	1.327e-06

3.5.4 Results for 10 generator system considering valve-point effect:

Best Fitnesss	Minimum	Maximum	Mean	Median	Standard Deviation
Total Fuel Cost	1.0606e+05	1.0615e+05	1.061e+05	1.061e+05	37.182

4. CONCLUSION AND FUTURE SCOPE

4.1 CONCLUSION

In our study, each cluster unit (each individual unit) is considered to be a swarm particle in a multidimensional potential energy surface (PES) where the stationary points (maxima, minima, and higher order saddle points) are connected. The randomly generated individual particle is governed by a position vector and a velocity vector. Again, each position vector representing a candidate solution in the hyperspace starts searching for the optima of a given function of several variables by updating generations in iterative process without much of any assumption leading to a minimum energy structure. After iteration the particle driven by a velocity vector changes its search direction. The position and velocity vectors together store the information regarding its own best position or the local best position (called p_{best}) seen so far and a global best position (called g_{best}) which is obtained by communicating with its nearest neighbours. Further, the advancement of particles toward the global best position is attained via particle swarm optimizer ideology and they gravitate toward the global best solution with the help of the best variable memory values. Our proposed PSO implementation explores rapidly without being entrapped in local optima and executes extensively, followed by immediate convergence to the desired objective value, the global optima.

4.2 FUTURE SCOPE

PSO can be applied to many types of problems in the most diverse areas of science. As an example, PSO has been used in healthcare in diagnosing problems of a type of leukaemia through microscopic imaging. In the economic sciences, PSO has been used to test restricted and unrestricted risk investment portfolios to achieve optimal risk portfolios. In the engineering field, the applications are as diverse as possible. Optimization problems involving PSO can be found in the literature in order to increase the heat transfer of systems or even in algorithms to predict the heat transfer coefficient. In the field of thermodynamics, one can find papers involving the optimization of thermal systems such as diesel engine–organic Rankine cycle, hybrid diesel-ORC/photovoltaic system, and integrated solar combined cycle power plants (ISCCs).PSO has also been used for geometric optimization problems in order to find the best system configurations that best fit the design constraints.

APPENDIX

A. PSO main function

```
clc                                              % To clear the command window
clear                                            % To clear the workspace
lb = [-10-10-10-10];                             % Lower Bound
ub = [ 10 10 10 10];                             % Upper Bound
prob = @rastr;                                   % Fitness Function
Np = 10;                                         % Population size
T = 50;                                          % No. of Iterations
w = 0.8;                                         % Inertia weight
c1 = 1.5;                                        % Acceleration Coefficient 1
c2 = 1.5;                                        % Acceleration Coefficient 2
f = NaN(Np,1);                   % Vector to store the fitness function value of the
%population size
BestFitIter = NaN(T+1,1);        % Vector to store the best objective fitness function
%value
D = length(lb);                  % Determining the no of decision variables
available
P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);
v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);
for p = 1:Np
   f(p) = prob(P(p,:));                   % Determining the fitness function of the
%initial population
end
pbest = P;                                % Initializing the personal best solution
f_pbest = f;                              % Initializing the fitness of the personal best
%solutions
[f_gbest,ind] = min(f_pbest);             % Determining the best objective fitness
%function value
gbest = P(ind,:);                         % Determining the best solutions
```

```
BestFitIter(1) = f_gbest;                  % Initializing the best fitness value at initial
%iteration

for t = 1:T

  for p = 1:Np

       v(p,:) = w*v(p,:) + c1*rand(1,D).*(pbest(p,:) - P(p,:)) + c2*rand(1,D).*(gbest -
P(p,:));

        P(p,:) = P(p,:) + v(p,:);              % Update the position

        P(p,:) = max(P(p,:),lb);             % Bounding the violating variables to their
%lower bound

        P(p,:) = min(P(p,:),ub);             % Bounding the violating variables to their
%lower bound

        f(p) = prob(P(p,:));                   % Determining the fitness of the new solution

        if f(p) < f_pbest(p)

            f_pbest(p) = f(p);              % Updating the fitness function value of the
%personal best solution

            pbest(p,:) = P(p,:);            % Updating the personal best solution

            if f_pbest(p) < f_gbest

                f_gbest = f_pbest(p);         % Updating the fitness function value of the
%best solution

                gbest = pbest(p,:);             % Updating the best solution

           end

        end

    end

    BestFitIter(t+1) = f_gbest;            % Updating the best fitness value at (t+1)
%iteration

    disp(['Iteration' num2str(t) ': Best Fitness = ' num2str(BestFitIter(t+1))]);

end

BestFitness = f_gbest

BestSol = gbest

subplot(1,2,1)

plot(0:T,BestFitIter);                       % Plotting the best fitness value vs iteration no.
```

```
curve
xlabel('Iteration');                     % Naming the X-axis
ylabel('Best Fitness Value');            % Naming the Y-axis
subplot(1,2,2)
semilogy(0:T,BestFitIter);               % Plotting the best fitness value vs iteration
no. %in semi log plot
xlabel('Iteration');                     % Naming the X-axis
ylabel('Best Fitness Value');            % Naming the Y-axis
                                                                   ……….[1]
```

B. Code for Finding Statistical Values

B.I Main Function File:

```
clc                                      % To clear the command window
clear                                    % To clear the workspace
close all
%% PROBLEM SETTING
lb = [-10-10-10-10];                     % Lower Bound
ub = [10 10 10 10 ];                     % Upper Bound
prob = @spheref;                         % Fitness Function
%% ALGORITHM PARAMETERS
Np = 10;                                 % Population size
T = 50;                                  % No. of Iterations
w = 0.8;                                 % Inertia weight
c1 = 1.5;                                % Acceleration Coefficient 1
c2 = 1.5;                                % Acceleration Coefficient 2
Nruns = 10;
BestSol = NaN(Nruns,length(lb));
```

```
BestFitness = NaN(Nruns,1);
BestFitIter = NaN(Nruns,T+1);
for i = 1:Nruns
    rng(i,'twister')                        % For controlling random numbers generated by
%rand, randi
    [BestSol(i,:),BestFitness(i),BestFitIter(i,:),P,f] = PSOf(prob,lb,ub,Np,T,w,c1,c2);
    plot(BestFitIter(i,:),'*')
    hold on
end
xlabel('Itertion')
ylabel('Best fitness Function Value')
[a,ind] = min(BestFitness);
b = max(BestFitness);
c = mean(BestFitness);
d = median(BestFitness);
e = std(BestFitness);
colnames = {'min' 'max' 'mean' 'maedian' 'std'};
Statistical_Values = table(a, b, c, d, e, 'VariableNames', colnames)
Best_Soln = BestSol(ind,:)
BestSol
figure
plot(mean(BestFitIter),'*');
xlabel('Itertion')
ylabel('Average of best fitness Function Value')
end                                                            ……….[1]
```

** Stat =

[$\text{Minimum}_{\text{BestFitness}}$ $\text{Maximum}_{\text{BestFitness}}$ $\text{Mean}_{\text{BestFitness}}$ $\text{Median}_{\text{BestFitness}}$ $\text{StandardDeviation}_{\text{BestFitness}}$]

B.II PSO Function File:

```
function [BestSol,BestFitness,BestFitIter,P,f] = PSOf(prob,lb,ub,Np,T,w,c1,c2);

f = NaN(Np,1);                          % Vector to store the fitness function value of the
%population size

BestFitIter = NaN(T+1,1);% Vector to store the best objective fitness function
%value at each iteration

D = length(lb);                         % Determining the no of decision variables
%available

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);    % Generation of the
%initial population

v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);    % Generation of the
%initial vector

for p = 1:Np

   f(p) = prob(P(p,:));                 % Determining the fitness function of the initial
%population

end

pbest = P;                              % Initializing the personal best solution

f_pbest = f;                            % Initializing the fitness of the personal best
%solutions

[f_gbest,ind] = min(f_pbest);    % Determining the best objective fitness function
%value

gbest = P(ind,:);                       % Determining the best solutions

BestFitIter(1) = f_gbest;          % Initializing the best fitness value at initial
%iteration

for t = 1:T

   for p = 1:Np

      v(p,:) = w*v(p,:) + c1*rand(1,D).*(pbest(p,:) - P(p,:)) + c2*rand(1,D).*(gbest
- P(p,:));

      P(p,:) = P(p,:) + v(p,:);             % Update the position

      P(p,:) = max(P(p,:),lb);            % Bounding the violating variables to their
%lower bound

      P(p,:) = min(P(p,:),ub);            % Bounding the violating variables to their
%lower bound
```

```
        if f(p) < f_pbest(p)
            f_pbest(p) = f(p);
            pbest(p,:) = P(p,:);                  % Updating the personal best solution
            if f_pbest(p) < f_gbest
                f_gbest = f_pbest(p
             gbest = pbest(p,:);                  % Updating the best solution
            end
        end
    end
    BestFitIter(t+1) = f_gbest;       % Updating the best fitness value at (t+1) iteration
end
    BestFitIter(t+1) = f_gbest;       % Updating the best fitness value at (t+1) iteration
end
BestFitness = f_gbest;
BestSol = gbest;
end
            ……….[1]
```

****This code is used as a function with little modification, by which we can take the help of another code to call it multiple times and run it to store BestFitness value every time and to find statistical values like maximum, minimum, mean, median and standard deviation value of all BestFitness value**

C. CODE FOR BENCHMARK FUNCTIONS

C.I. Sphere Function

```
function f = SphereNew(x)
% SPHERE FUNCTION
% INPUT: xx = [x1, x2, ..., xd]
% f(xx)=[(x1^2)+(x2^2)+(x3^2)+....+(xd^2)]
f = sum(x.^2);
end                                                              ……….[2]
```

C.II Griewank Function

```
function [y] = griewank(xx)
% GRIEWANK FUNCTION
% INPUT: xx = [x1,..xd]
% f(x) = 1 + { [((1/4000)(x1^2))+((1/4000)(x2^2))+
%             ((1/4000)(x3^2))+...+((1/4000)(xd^2))] -
%            [(cos(x1/?1))*(cos(x2/?2))*
%             (cos(x3/?3))*...*(cos(xd/?d))] }
d = length(xx);
sum = 0;
prod = 1;
for ii = 1:d
        xi = xx(ii);
        sum = sum + xi^2/4000;
        prod = prod * cos(xi/sqrt(ii));
end
y = 1 + sum - prod;
end                                                    ……….[2]
```

C.III Rastrigin Function

```
function [y] = rastr(xx)
% RASTRIGIN FUNCTION
% INPUT: xx = [x1, x2, ..., xd]
% f(x)=(10*d) + { [(x1^2)-(10*cos(2?x1))] +
%              [(x2^2)-(10*cos(2?x2))] +
%              [(x3^2)-(10*cos(2?x3))] +....+
%              [(xd^2)-(10*cos(2?xd))] }
```

```
d = length(xx);
sum = 0;
for ii = 1:d
        xi = xx(ii);
        sum = sum + (xi^2 - 10*cos(2*pi*xi));
end
y = 10*d + sum;
end                                                  ……….[2]
```

C.IV Rosenbrock Function

```
function [y] = rosen(xx)
% ROSENBROCK FUNCTION
% INPUT: xx = [x1, x2, ..., xd]
% f(x)= { [100*((x2-(x1^2))^2)+((x1-1)^2)] +
%        [100*((x3-(x2^2))^2)+((x2-1)^2)] +
%        [100*((x4-(x3^2))^2)+((x3-1)^2)] +....+
%        [100*((xd-(x(d-1)^2))^2)+((x(d-1)-1)^2)] }
d = length(xx);
sum = 0;
for ii = 1:(d-1)
        xi = xx(ii);
        xnext = xx(ii+1);
        new = 100*(xnext-xi^2)^2 + (xi-1)^2;
        sum = sum + new;
end
y = sum;
end                                                  ……….[2]
```

D. CODE FOR IEEE 30 BUS 6 GENERATOR SYSTEM

D.I. Function to evaluate power balance constraint

```
function [Pg]=PEvaluation(Pg)
totPd=2.834; ng=6;
Pgsum=sum(Pg);
for i=1:ng
   Pg(i)=Pg(i)/Pgsum*totPd;
end
```

D.II. Total Emission Function

```
function [Temission]=ieee30G6Em(Pg)
% %%This is the objective function for minimum emission schedule
%IEEE 30G6%%%
% %@Author: Rajat Kanti Samal
ng=6;
totPd=2.834;%pu
%0.01*(alpha+beta*Pgi+gamma*Pgi2)+zeta*exp(lambda*Pgi)
d=[4.091 2.543 4.258 5.326 4.258 6.131];%alpha
e=[-5.554 -6.047 -5.094 -3.550 -5.094 -5.555];%beta
f=[6.490 5.638 4.586 3.380 4.586 5.151];%gamma
g=[2.0e-04 5.0e-4 1e-6 2e-3 1e-6 1e-5];%zeta
h=[2.857 3.333 8.000 2.000 8.000 6.667];%phi
E=zeros(1,ng);
Temission=0;
for i=1:ng
   %Pg(i)=Pg(i)/Pgsum*totPd;
   E(i)=0.01*(d(i)+e(i)*Pg(i)+f(i)*(Pg(i)^2))+g(i)*exp(h(i)*Pg(i));
   Temission=Temission+E(i);
end
```

D.III Total cost minimization function

```
function [Tcost]=ieee30G6(Pg)
%Solving ED problem for IEEE 30 bus 6 generator system
totPd=2.834;%pu
ng=6;
%a+bPgi+cPgi2
a=[10 10 20 10 20 10];
b=[200 150 180 100 180 150];
c=[100 120 40 60 40 100];
%0.01*(alpha+beta*Pgi+gamma*Pgi2)+zeta*exp(lambda*Pgi)
d=[4.091 2.543 4.258 5.326 4.258 6.131];%alpha
e=[-5.554 -6.047 -5.094 -3.550 -5.094 -5.555];%beta
f=[6.490 5.638 4.586 3.380 4.586 5.151];%gamma
g=[2.0e-04 5.0e-4 1e-6 2e-3 1e-6 1e-5];%zeta
h=[2.857 3.333 8.000 2.000 8.000 6.667];%phi
Pmin=[0.05 0.05 0.05 0.05 0.05 0.05];
Pmax=[0.50 0.60 1.00 1.20 1.00 0.60];
Tcost=0;
F=zeros(1,ng);
for i=1:ng
   F(i)=a(i)+b(i)*Pg(i)+c(i)*(Pg(i)^2);
   Tcost=Tcost+F(i);
   end                                                  ……….[5]
```

D.IV PSO Function File-

```
function [BestSol,BestFitness,BestFitIter,P,f] = PSOf(prob,lb,ub,Np,T,w,c1,c2);

f = NaN(Np,1);                    % Vector to store the fitness function value of the
%population size

BestFitIter = NaN(T+1,1);         % Vector to store the best objective fitness function
%value at each iteration

D = length(lb);                   % Determining the no of decision variables available

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);    % Generation of the
%initial population

v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);    % Generation of the
%initial vector

for p = 1:Np

   f(p) = prob(P(p,:));           % Determining the fitness function of the initial
%population

end

pbest = P;                        % Initializing the personal best solution

f_pbest = f;                      % Initializing the fitness of the personal best
%solutions

[f_gbest,ind] = min(f_pbest);     % Determining the best objective fitness function
value

gbest = P(ind,:);                 % Determining the best solutions

BestFitIter(1) = f_gbest;         % Initializing the best fitness value at initial iteration

for t = 1:T

   for p = 1:Np

      v(p,:) = w*v(p,:) + c1*rand(1,D).*(pbest(p,:) - P(p,:)) + c2*rand(1,D).*(gbest -
P(p,:));

      P(p,:) = P(p,:) + v(p,:);            % Update the position

      P(p,:) = max(P(p,:),lb);             % Bounding the violating variables to their
%lower bound

      P(p,:) = min(P(p,:),ub);             % Bounding the violating variables to their
%lower bound
```

```
      f(p) = prob(P(p,:));                    % Determining the fitness of the new solution
      if f(p) < f_pbest(p)
         f_pbest(p) = f(p);
         pbest(p,:) = P(p,:);                 % Updating the personal best solution
         if f_pbest(p) < f_gbest
            f_gbest = f_pbest(p
          gbest = pbest(p,:);                 % Updating the best solution
         end
      end
   end
   BestFitIter(t+1) = f_gbest;                % Updating the best fitness value at (t+1)
%iteration
end
   BestFitIter(t+1) = f_gbest;                % Updating the best fitness value at (t+1)
%iteration
end
BestFitness = f_gbest;
BestSol = gbest;
end                                                                      ……….[1]
```

D.V Code for Finding Statistical Values

```
clc                                               % To clear the command window
clear                                             % To clear the workspace
close all
%% PROBLEM SETTING
lb = [0.05 0.05 0.05 0.05 0.05 0.05];                       % Lower Bound
ub = [0.50 0.60 1.00 1.20 1.00 0.60];                        % Upper Bound
```

```
prob = @ieee30G6Em;                                              % Fitness Function

%% ALGORITHM PARAMETERS

Np = 10;                                                        % Population size

T = 50;                                                         % No. of Iterations

w = 0.8;                                                        % Inertia weight

c1 = 1.5;                                                       % Acceleration Coefficient 1

c2 = 1.5;                                                       % Acceleration Coefficient 2

Pc = 1;                                                         % Population Condition

Nruns = 10;

BestSol = NaN(Nruns,length(lb));

BestFitness = NaN(Nruns,1);

BestFitIter = NaN(Nruns,T+1);



for i = 1:Nruns

   rng(i,'twister')                          % For controlling random numbers generated by
%rand, randi

   [BestSol(i,:),BestFitness(i),BestFitIter(i,:)] = PSOf(prob,lb,ub,Np,T,w,c1,c2,Pc);

   plot(BestFitIter(i,:),'*')

hold on

end

xlabel('Itertion')

ylabel('Best fitness Function Value')

p1=BestSol(:,1);

p2=BestSol(:,2);

p3=BestSol(:,3);

p4=BestSol(:,4);

p5=BestSol(:,5);

p6=BestSol(:,6);
```

```
tc=sum(BestSol,2);
colnames = {'P1' 'P2' 'P3' 'P4' 'P5' 'P6' 'Total'};
Load_Sharing = table(p1,p2,p3,p4,p5,p6,tc, 'VariableNames', colnames)
[a,ind] = min(BestFitness);
b = max(BestFitness);
c = mean(BestFitness);
d = median(BestFitness);
e = std(BestFitness);
colnames = {'Min' 'Max' 'Mean' 'Median' 'Standard_Deviation'};
Statistical_Values = table(a, b, c, d, e, 'VariableNames', colnames)
Best_Soln = BestSol(ind,:)
figure
plot(mean(BestFitIter),'*');
xlabel('Itertion')
ylabel('Average of best fitness Function Value')                    ……….[1]
```

E. CODE FOR 10 GENERATOR SYSTEM CONSIDERING VALVE-POINT EFFECT

E.I. Function to evaluate power balance constraint

```
function [Pg]=PEvaluation(Pg)
totPd=2000;
ng=10;
Pgsum=sum(Pg);
for i=1:ng
   Pg(i)=Pg(i)/Pgsum*totPd;
end
```

E.II. Total Emission Function

```
function [Tcost]=EED(Pg)
%Solving economic environmental dispatch (EED) problem for Micro Grid 6
generator system
%The fuel cost function of each fossil fuel fired generator
%considering the valve-point efffect into consideration

totPd=2000;%MW
ng=10;
%a+bPgi+cPgi2+|dsin{e(pimin-pi)}|
a=[1000.403 950.606 900.705  800.705 756.799 451.325 1243.531 1049.998
1658.569 1356.659]; %($/h)
b=[ 40.5407 39.5804 36.5104 39.5104  38.5390 46.1592 38.3055 40.3965  36.3278
38.2704]; %($/MWh)
c=[0.12951  0.10908  0.12511 0.12111 0.15247  0.10587  0.03546  0.02803
0.02111 0.01799]; %($/h)
d=[33 25 32 30 30 20 20 30 60 40]; %($/h)
e=[0.0174 0.0178 0.0162 0.0168  0.0148 0.0163 0.0152 0.0128 0.0136 0.0141];
%(rad/MW)
Pmin=[10 20 47 20 50 70 60 70 135 150]; %(MW)
Pmax=[55 80 120 130 160 240 300 340 470 470]; %(MW)

Tcost=0;
F=zeros(1,ng);
for i=1:ng
    F(i)=a(i)+b(i)*Pg(i)+c(i)*(Pg(i)^2)+abs(d(i)*sin(e(i)*(Pmin(i)-Pg(i))));
    Tcost=Tcost+F(i);

end                                                          …………[6]
```

E.III PSO Function File-

```
function [BestSol,BestFitness,BestFitIter,P,f] = PSOf(prob,lb,ub,Np,T,w,c1,c2);

f = NaN(Np,1);                    % Vector to store the fitness function value of the
%population size

BestFitIter = NaN(T+1,1);         % Vector to store the best objective fitness function
%value at each iteration

D = length(lb);                   % Determining the no of decision variables available

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);    % Generation of the
%initial population

v = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);    % Generation of the
%initial vector

for p = 1:Np

   f(p) = prob(P(p,:));           % Determining the fitness function of the initial
%population

end

pbest = P;                        % Initializing the personal best solution

f_pbest = f;                      % Initializing the fitness of the personal best
%solutions

[f_gbest,ind] = min(f_pbest);    % Determining the best objective fitness function
value

gbest = P(ind,:);                 % Determining the best solutions

BestFitIter(1) = f_gbest;         % Initializing the best fitness value at initial iteration

for t = 1:T

   for p = 1:Np

      v(p,:) = w*v(p,:) + c1*rand(1,D).*(pbest(p,:) - P(p,:)) + c2*rand(1,D).*(gbest -
P(p,:));

      P(p,:) = P(p,:) + v(p,:);              % Update the position

      P(p,:) = max(P(p,:),lb);               % Bounding the violating variables to their
%lower bound

      P(p,:) = min(P(p,:),ub);               % Bounding the violating variables to their
%lower bound
```

```
        f(p) = prob(P(p,:));                % Determining the fitness of the new solution
        if f(p) < f_pbest(p)
            f_pbest(p) = f(p);
            pbest(p,:) = P(p,:);            % Updating the personal best solution
            if f_pbest(p) < f_gbest
                f_gbest = f_pbest(p
             gbest = pbest(p,:);            % Updating the best solution
            end
        end
    end
    BestFitIter(t+1) = f_gbest;             % Updating the best fitness value at (t+1)
%iteration
end
    BestFitIter(t+1) = f_gbest;             % Updating the best fitness value at (t+1)
%iteration
end
BestFitness = f_gbest;
BestSol = gbest;
end                                                                  ……….[1]
```

E.IV Code for Finding Statistical Values

```
clc                                                % To clear the command window
clear                                              % To clear the workspace
close all
%% PROBLEM SETTING
lb = [0.05 0.05 0.05 0.05 0.05 0.05];                       % Lower Bound
ub = [0.50 0.60 1.00 1.20 1.00 0.60];                       % Upper Bound
```

```
prob = @ieee30G6Em;                                    % Fitness Function
%% ALGORITHM PARAMETERS
Np = 10;                                          % Population size
T = 50;                                          % No. of Iterations
w = 0.8;                                          % Inertia weight
c1 = 1.5;                                         % Acceleration Coefficient 1
c2 = 1.5;                                         % Acceleration Coefficient 2
Pc = 1;                                          % Population Condition
Nruns = 10;
BestSol = NaN(Nruns,length(lb));
BestFitness = NaN(Nruns,1);
BestFitIter = NaN(Nruns,T+1);



for i = 1:Nruns
   rng(i,'twister')                    % For controlling random numbers generated by
%rand, randi
   [BestSol(i,:),BestFitness(i),BestFitIter(i,:)] = PSOf(prob,lb,ub,Np,T,w,c1,c2,Pc);
   plot(BestFitIter(i,:),'*')
hold on
end
xlabel('Itertion')
ylabel('Best fitness Function Value')
p1=BestSol(:,1);
p2=BestSol(:,2);
p3=BestSol(:,3);
p4=BestSol(:,4);
p5=BestSol(:,5);
p6=BestSol(:,6);
```

```
tc=sum(BestSol,2);

colnames = {'P1' 'P2' 'P3' 'P4' 'P5' 'P6' 'Total'};

Load_Sharing = table(p1,p2,p3,p4,p5,p6,tc, 'VariableNames', colnames)

[a,ind] = min(BestFitness);

b = max(BestFitness);

c = mean(BestFitness);

d = median(BestFitness);

e = std(BestFitness);

colnames = {'Min' 'Max' 'Mean' 'Median' 'Standard_Deviation'};

Statistical_Values = table(a, b, c, d, e, 'VariableNames', colnames)

Best_Soln = BestSol(ind,:)

figure

plot(mean(BestFitIter),'*');

xlabel('Itertion')

ylabel('Average of best fitness Function Value')                    ……….[1]
```

REFERENCES

[1] **Dr. Prakash Kotecha**, *Associate Professor (IIT, Guwahati)*, “NOC: Computer Aided Applied Single Objective Optimization”, https://nptel.ac.in/courses/103/103/103103164/

[2] **SonjaSurjanovi c&Derek Bingham**, *Simon Fraser University*, “Virtual Library of Simulation Experiments: Test Functions and Datasets”, https://www.sfu.ca/~ssurjano/optimization.html

[3] **Gourhari Jana**, Department of Chemistry and Centre for Theoretical Studies, Indian Institute of Technology, Kharagpur , **Arka Mitra**, Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur, **Sudip Pan**, FachbereichChemie, Philipps-Universität Marburg, Marburg, Germany, **Shamik Sural**, Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, **Pratim K. Chattaraj**, Department of Chemistry, Indian Institute of Technology Bombay, Mumbai, “Modified Particle Swarm Optimization Algorithms for the Generation of Stable Structures of Carbon Clusters, C_n (n = 3–6, 10)”, https://www.frontiersin.org/articles/10.3389/fchem.2019.00485/full

[4] **Dr. Hossein Arsham**, The Wright Distinguished Research Professor at the University of Baltimore and Visiting Professor at the Johns Hopkins Carey Business School, “Deterministic Modeling: Linear Optimization with Applications”, http://home.ubalt.edu/ntsbarsh/opre640a/partviii.htm

[5] **M. A. Abido**, *Member, IEEE*, “Multiobjective Evolutionary Algorithms for Electric Power Dispatch Problem”, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 10, NO. 3, JUNE 2006.

[6] **M. Basu,** Department of Power Engineering, Jadavpur University, Block LB, Sector-3, Salt Lake City, Kolkata, West Bengal 700098, India, “Economic environmental dispatch using multi-objective differential evolution”, APPLIED SOFT COMPUTING 11 (2011) 2845–2853