CHAPTER 1:

INTRODUCTION

1.1 **OVERVIEW**

Economic load dispatch problem involves the solution of two different problems. The first of these is the Unit Commitment or pre-dispatch problem wherein it is required to select available generating source to operate so that expected load can be met. The second problem of economic load dispatch is the on-line economic dispatch in such a way to minimize the cost of supplying while fulfilling the minute to minute requirement. In economic load dispatch we find the generation of the generators so that the total fuel cost is minimum, and at the total demand and the losses at any instant must be met by the total generation. Due to energy crisis in the world, and continues rise in prices, it is essential to reduce the fuel consumption to meet the particular load demand. Depending upon the various operational constraint, cost of generation is not fixed for a particular load demand.

In this project work an attempt has been made to solve ELD problem by using different acceleration coefficients in conventional PSO. Various improved PSO algorithms IPSO-1,IPSO-2,IPSO-3 and IPSO-4 has been developed. Comparative analysis suggests that IPSO significantly improves the performance with less no of iteration. All the above algorithms have been implemented on IEEE 5, 14, 30 bus systems.

1.2 OBJECTIVE AND METHODOLOGY

Our objective in this work is to find the solution of economic load dispatch (ELD) problem considering cost of generation. For this IEEE 5, 14, & 30 bus system have been considered.

The work has been carried out in following order:

a. Exploring Particle Swarm Optimization (PSO) and coding its algorithm in MATLAB R2011b.
b. Solution of various mathematical benchmark functions using PSO.
c. Formulation of Economic Load Dispatch (ELD) considering cost of generation for IEEE 5, 14, & 30 Bus Systems.
d. Generation of non-inferior sets of IEEE 5, 14 and 30 bus systems.
e. Achievement of solution for IEEE 5, 14 and 30 bus system with less no of iteration.

1.3 LITERATURE SURVEY

By economic load dispatch problems, the best combination of power outputs of all generating units is to be determined so that total fuel cost is to be minimized, while satisfying the load demand and operational constraints. Recently, PSO technique has been successfully applied to find the solution of economic load dispatch as well as multiobjective economic load dispatch problem. K.E. PARSOPOULOS et al [1] describes PSO in Minimax, Multiobjective, errors-in-variables problems and Integer Programming ,as well as problems in noisy and changing environments. Zhiyu You et al [2] solved premature convergence problem which is possible by generally local optimization in the way of global optimization, an adaptive weight Particle Swarm Optimization (PSO) algorithm with constriction factor. The value of inertia weight is set according to dynamic information about the changes in objective function value, so as to effectively balance the advantages of global optimization against the shortage of local optimization. J. C. Bansal et al [3] proposed variations of Inertia Weight strategy and then compares their performance on different optimization test problems. Russell Eberhart et al [4] proposed two Paradigms of PSO namely

globally oriented (Gbest), and locally oriented (Lbest) and compared for the extremely nonlinear Schaffer f6 function.The authour propose both the paradigm for training of neural network & learning of robot. Ajith Abraham et al [5] implemented Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) algorithms on some mathematical function (Griewank function, Quadric function and Schwefel function), real world applications as travelling sales men problem & data mining. These results are then analysed & are disscused in detail.

Ehsan Shahamatnia et al [6] proposed an approach for Active contour searching by using new PSO. The concepts from active contour model into particle swarm optimization is done by this approach so that a snaxel of the active contour is represented by each particle. Flood control optimization problem is focused by Changming Ji et al [7] and he proposed the CE-PSO model and specific steps to solve it. Steffen Freitag et al [8] introduced a new training strategy for recurrent neural networks considering both ways of computation. It is based on swarm intelligence. To solve the discrete optimization problem, Ismail Ibrahim et al [9] proposed a novel approach and this approach is called multi-state particle swarm optimization (MSPSO). J.J. Jamian et al [10] described rank evolutionary PSO (REPSO) method. The comparisons of REPSO method with evolutionary PSO (EPSO) and traditional PSO. The author has shown that REPSO is superior over PSO and EPSO when optimum size of distributed generation in 69 bus radial system is determining. Chien-Ching Chiu et al [11] described dynamic differential evolution (DDE) and particle swarm optimization technique for invention of shape reconstruction of perfectly conducting cylinder. Liu Jin-yue et al [12] described ideology of mutation in genetic algorithm and introduce the mutation operation in the standard particle swarm algorithm. The improved algorithm is compared with standard PSO for the multi peak function namely Ackley and Rastirgrin-function. It has been shown that improved PSO algorithm can effectively get rid of local optimum to avoid premature convergence. Jun Sun et al [13] described drift particle swarm optimisation algorithm to solve economic dispatch problems. The RDPSO method is evaluated on power systems and compared with other optimization method like GA, ACSA, DE, AIS ETC, in terms of the solution quality, robustness and conveyance performance. It has been shown that RDPSO method performs better in solving ED problems than any other optimization techniques.

In this thesis, improved version of PSO (**IPSO**) has been discussed**. IPSO 1, 2, 3, 4** has been discussed. Various improved PSO algorithms IPSO-1,IPSO-2,IPSO-3 and IPSO-4 has been developed. Comparative analysis suggests that IPSO significantly improves the

performance with less no of iteration. In this thesis, special consideration has been given to equality constraint of the ELD problem. The objective function has been modified by the inclusion of a parameter K.

1.4 PLAN OF THESIS

This dissertation has been arranged in six chapters. The contents of the chapters are briefly outlined as indicated below:

Chapter 1: Discussion the introduction to economic load dispatch and Research objectives of the thesis. Literature survey of the covered topics has also been presented.

Chapter 2: Present the Particle Swarm Optimization

Chapter 3: Presents the Particle Swarm Optimization and its applications.

Chapter 4: Explores the concepts of Particle Swarm Optimization algorithm in MATLAB R2011b. Analysis of various parameters in PSO algorithm has been carried out.

Chapter 5: Discussion to the solution of Economic Load Dispatch for IEEE 5, 14 and 30 bus systems.

Chapter 6: Conclusion and the prospects for Future Directions have been discussed. Appendix and references are at the end of the thesis.

CHAPTER 2:

PARTICLE SWARM OPTIMIZATION

2.1 INTRODUCTION

The optimization technique PSO is population based self-adaptive method developed by Kennedy and Eberhart in 1995, inspired by the collective behaviour of nature to optimize the linear and non-linear functions. Kennedy (a social psychologist) and Eberhart (an electrical engineer) gave the initial idea of PSO, which aimed to produce computational intelligence by social analogous way. Heppner and Grenander's work influenced the first simulation of Kennedy and Eberhart in 1995, which involved alogues of bird flocks searching for corn.

Under the class of swarm intelligence techniques PSO is used to find the solution of extremely difficult optimization problems. Swarm intelligence is the technique used to study of self-organised systems so that it gives the collective behaviour of decentralised system. By PSO we create the mathematical modeling and then simulation of food searching activities of a swarm of birds. By adding a velocity with its position each particle is moved toward optimal position in the multidimentional space.In an n-dimensional search space, position and velocity of particle i are represented by vectors $X_i = (X_{i1,} X_{i2}…X_{ip})$ and $V_i = (V_{i1},V_{i2},….V_{ip})$ respectively. Let Xpbest and Xgbest be the the personal and global best position of the particle i. The modified velocity and position of each particle can be calculated using current velocity and distance from Xpbest and Xgbest as follows:

$$V_{ij}^{k} = V_{ij}^{k-1} + C_p * r_p * \left(Xpbest_{ij} - X_{ij}^{K-1}\right) + C_g * r_g * \left(Xgbest_i - X_{ij}^{K-1}\right)$$

$$i=1, 2 \ldots NG, j=1, 2 \ldots p \qquad (2.1a)$$

Position update equation is given by

$$X_{ij}^{k} = X_{ij}^{k-1} + V_{ij}^{k}$$

$$i=1, 2 \ldots NG, \ j=1, 2 \ldots p \qquad (2.1b)$$

where

K Iteration count.

V_{ij}^{k} Value of velocity of j^{th} particle of i^{th} generator at iteration.

X_{ij}^{k} Value of position of j^{th} particle of i^{th} generator at K^{th} iteration.

C_p, C_g Acceleration coefficients.

$Xpbest_{ij}$ Value of personal best position of each particle in each iteration.

$Xgbest_j$ Value of global best position of each particle in each iteration.

NG No. of Generating buses;

p No. of particles in the swarm;

r_p, r_g two separately generated uniformly distributed random numbers in the range (0, 1).

Velocities are updated by the equation (2.1a) and the positions of each particle for each decision variable are calculated by equation (2.1b). The inertia weight is proposed in the velocity equation to increase the convergence rate of the PSO algorithm. By using the equation for the velocity with the inertia weight, the suggested particle velocity will be changed to:

$$V_{ij}^{k} = W * V_{ij}^{k-1} + C_p r_p \left(Xpbest_{ij} - X_{ij}^{k-1}\right) + C_g r_g (Xgbest_i - X_{ij}^{k-1})$$

$$i=1, 2 \ldots NG, j=1, 2 \ldots p \qquad (2.1c)$$

where:

W is the inertia weight.

Particle maintain their velocity from previous iteration to new iteration due to this inertia weight some of the particles and the chance to obtain a best solution for an optimization problem will be more if we are using the inertia weight function.

For SPSO(Standard Particle Swarm Optimization)

$$W=W_{max}-k*(W_{max}-W_{min})/iter_{max} \quad (2.1d)$$

Where

W	inertia weight factor
W_{max}	maximum value of velocity weighting factor
W_{min}	minimum value of velocity weighting factor
$iter_{max}$	maximum number of iteration
K	current number of iteration

While in IPSO, we have taken inertia weight function as a constant value.

2.2 PARTICLE SWARM OPTIMIZATION (PSO) AND TRADITIONAL SEARCH METHODS

- In PSO, each particle has given random velocity in the given search space, while in the other traditional method, each particle has given certain fixed velocity.
- In PSO, each particle follow the path of cooperation over competition, while in the traditional method, each particle use survival of fittest.
- PSO does not have any genetic operator like crossover between individuals, and in PSO, any individual never substitute the particle during the run.
- In PSO, each particle updates its position according to the initial velocity, and each particle has its own memory and according to its memory, each particle updates its position. So, whole population moves like one group towards the optimal solution. In PSO, gbest and pbest gives out the information to others.
- PSO takes generally large CPU time, while traditional method take small CPU time.

- The Evolutionary algorithm is applicable to variety of problems, while traditional method is applicable to the specific problem domain.
- The Evolutionary algorithm requires an objective function, which is to be optimized, while traditional method requires the knowledge of gradient vectors.
- In Evolutionary algorithm, initial guess is generated automatically, while in the traditional method initial guess is provided by user.
- The flow of control in the Evolutionary algorithm is mostly parallel, while in the traditional method it is mostly serial.
- The result in the Evolutionary algorithm is probably obtained as globally optimum, while in traditionally algorithm, result is obtained as local optimum depending upon initial guess.

2.3 BIOLOGICAL TERMINOLOGY

Optimization method (Particle Swarm Optimization) is developed by Eberhart and Kennedy in 1995 based on social behaviours of birds flocking and fish schooling. It is robust technique method and it has few parameter like inertia weight function and acceleration coefficient (lacal and global). The movement of position of each particle depends upon the personal best and global best value. In PSO, each particle has its own memory and according to its own memory and other particles memory, each particle update its own position to new value. PSO is population based technique which is called swarm of particle. Particle swarm optimization (PSO) is an artificial intelligence (AI) technique that can be used to find approximate solutions to extremely difficult or impossible numeric maximization and minimization problems.

Swarm: The population of moving particles is generally termed as swarm which moves towards a common point while each particle seems to move randomally. The no of population in the swarm remains same during the entire operation, and finally each particle merges to a common point.

Particle (X): A possible solution to the numeric optimization problem under investigation is represented by the position of a particle.In an i-dimensional space, the candidate solution is called particle.Each particle moves in the random direction

based upon the initial guess. At time t,the jth particle $X_j(t)$ can be described as $X_j(t)$=[X1j(t),X2j(t),.....,Xij(t)], where Xij are the optimized parameter and xij(t) is the position of jth particle with respect to the ith dimension; i.e. the value of the ith optimized, parameter in the jth candidate solution.

Velocity (V): Each particle is moving and hence has a velocity.Velocity of moving particle can be represented by an i-dimensional vector, where i is the number of optimized parameters. At time t, the jth particle Vj(t) can be described as Vj(t)=[Vij(t), V2j(t), V3j(t) .....Vij(t)]. In each time space, each particle moves to new position by adjusting its velocity. The new position is achieved by current velocity, weighted random position in the direction of personal best and weighted random position in the direction by global best.

Personal best (Pbest): The personal best position of the jth particle is the best position of the particle, which has been achieved by that particle so far.

Global best (Gbest): The best position of any paricle achieved so far, giving highest fitness value, is called global best.

In the optimization technique, there is pre-defined search space in which each particle moves to achieve its target position, which is done by acceleration coefficient. To find the optimal solution, each particle tracks its own position and neighbour's position. Each particle has its best position, defined by its highest fitness value, which is called personal best and the best fitness value achieved by any particle in the search space is called global best value. These two parameter, personal best and global best influences the updated position of each particle. Each particle updates its position according to distance between pbest and current position, and gbest and current position.

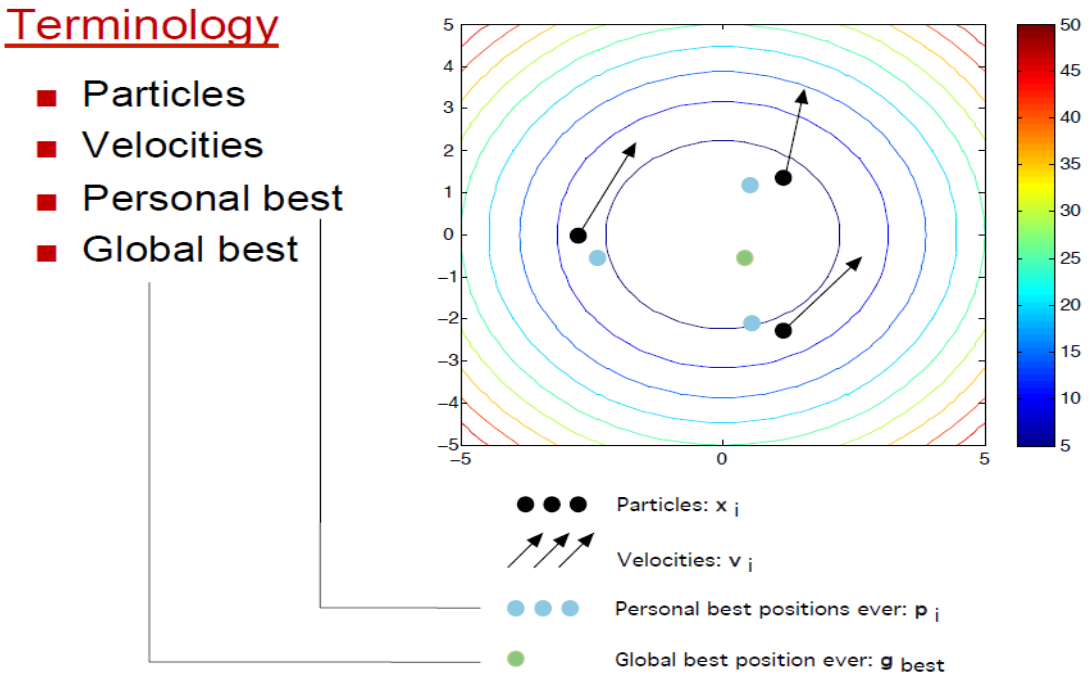


Fig 2.1: Various Biological Terminology

2.4 COMPUTATIONAL PROCEDURE

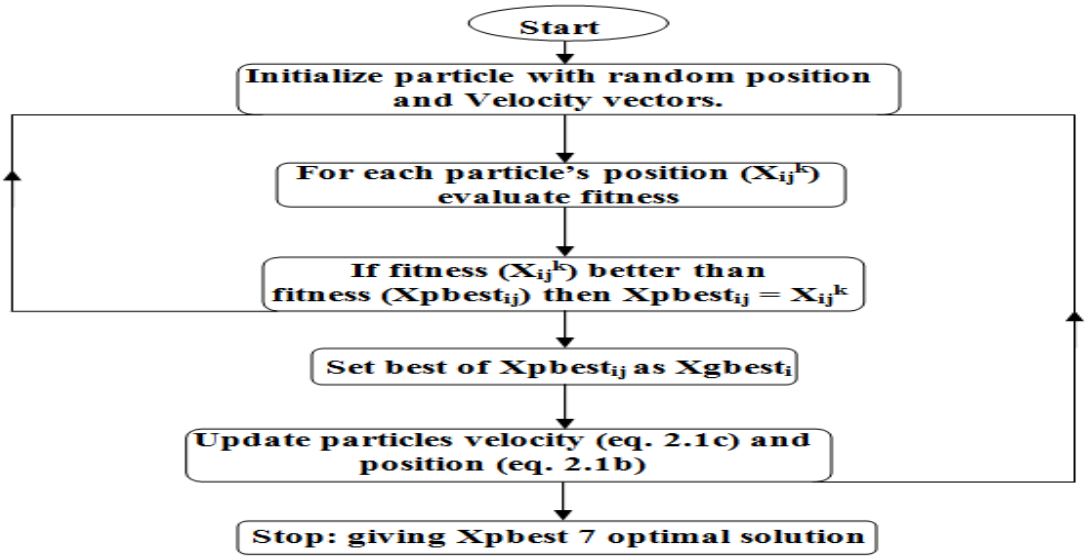


Fig 2.2: Generalized Flowchart for Particle Swarm Optimization Algorithm

The general computational procedure for the Particle Swarm Optimization is as follows:

1. Before the iteration starts, initialize the particle with random position and velocity vectors.
2. For each of the particle's position (X_{ij}^{k}) calculate the value of the objective function (F; we also call it fitness function).
3. If $F(X_{ij}^{k})$ is less than $F(Xpbest_{ij})$, then assign the value of $Xpbest_{ij}$ as X_i^{k} (do it for all the particles).
4. Determine the best value of Xpbestij, considering its fitness value. If F(best of $Xpbest_{ij}$) is less than $F(Xgbest_i)$, than assign the value of best of $Xpbest_{ij}$ to $Xgbest_i$.
5. Calculate the new velocity vector using equation and the new positionvector using equation.
6. Iterates the loop until either the stopping criteria met or the max iteration is achieved.
7. After iteration completes, give $Xgbest_i$ as the optimal solution and the fitness corresponding to it as the optimum value.

2.5 ADVANTAGES OF PARTICLE SWARM OPTIMIZATION

The PSO has many advantages as given below :-

I. PSO gives robust algorithm because it has very less impact of initial guess point, while the other evolutionary method has more impact of initial guess point.
II. PSO is a derivative free technique.
III. PSO needs mostly parallel flow of control.
IV. PSO has limited parameter like inertia weight function, acceleration coefficient (globally & local). The impact of these parameter on the solution of PSO is less compared to other optimization technique.
V. PSO generally takes less time to compute the solution.

It has also some limitation for real time ELD application. Sometimes, it take longer time to compute the ELD problem compared to

other mathematical approach. But for the real world ELD problem PSO can be applied. Also, the PSO-based approach is belived that it has less negative impact on the solutions than other heuristic-based approaches. However, it still has the problems of dependency on initial points and parameters, difficulty in finding their optimal design parameters, and the stochastic characteristic of the final outputs.

CHAPTER 3

SOLUTION OF MATHEMATICAL BENCHMARK FUNCTIONS USING PSO

3.1 STEPS OF PARTICLE SWARM OPTIMIZATION IN MATLAB

The basic steps for solving the optimization problem using Particle Swarm Optimization is same as explained in previous chapter. But with some modification we can use it for any type of objective function. Here PSO has been used for optimizing some mathematical functions, which are:

1. Rosenbrock Function:
   $f(x_1,x_2) = 100(x_2-x_1)^2 +(x_1-1)^2$
2. Beale's Function:
   $f(x_1,x_2) = (1.5 - x_1+ x_1x_2)2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$
3. Sphere Function:
   $f(x_1,x_2,x_3) = x_1^2 + x_2^2 + x_3^2$
4. Booth's Function:
   $f(x_1,x_2) = (x_1+2\ x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$

The steps for optimizing any mathematical bench mark test function :

a. Initialize all the variable matrices using the zero command of MATLAB.
b. Set the values of random numbers rp and rg asssigned to personal and global best expression respictively.
c. Set the values of acceleration constants cp and cg assigned to personal and global best expression respictively and ser the tolerance value.
d. Generate the random values of particles for both the x1 and x2 variables, also generate random valocity vectors v1 and v2 respectively.
e. Calculate the fitness the assumed values of the positions of the particles.
f. Using the above fitness Xpbest vector for both the variables X1 and X2 and Xgbest value is deduced.

g. Using the previous iteration vaues of personal best, global best and velocity vecgtor, new velocity vector is generated in the current iteration using equation.
h. Using the new velocity vectors and the old position vector a new position vector is generated for both the variables using equation.
i. Calculate fitness using the new position vectors in the current iteration.
j. Using the new fitness values, the personal and global best values are updated.
k. The difference between the previous and the current fitness is calculated and checked against the tolerance value, if within the tolerance iteration stops and global best value is the solution else iteration flow goes back to step g.

In this way optimized value of objective function and the corresponding variable values are found.

3.2 DIFFERENT PARAMETERS OF PARTICLE SWARM OPTIMIZATION

The various parameters of Particle Swarm Optimization are

1. No. of Particle in the swarm, p.
2. Max. no. of iteration, it.
3. Random no. for personal & global factor r_p & r_g.
4. Acceleration constant for personal and global factor, c_p & c_g.
5. Tolerance value, T.

3.3 APPLICATION OF PARTICLE SWARM OPTIMIZATION TO MATHEMATICAL BENCHMARK TEST FUNCTIONS

Test functions, known as **artificial landscapes**, are useful to evaluate characteristics of optimization algorithms. In this case of application of Particle Swarm Optimization to the mathematical benchmark functions, the PSO algorithm can be applied directly to the particular mathematical function, i.e. without any modification. As the mathematical functions are single objective functions and no equality criteria on the fitness function values, no further formulation for objective function is required and the inequality constraints on the variables, if present, are taken care of in the PSO algorithm itself.

The various benchmark test optimized are as follows:

1. Rosenbrock Function:

$$f(x_1,x_2) = 100(x_2-x_1)^2 +(x_1-1)^2$$

2. Beale's Function:

$$f(x_1,x_2) = (1.5 - x_1+ x_1x_2)2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$$

3. Sphere Function:

$$f(x_1,x_2,x_3) = x_1^2 + x_2^2 + x_3^2$$

4. Booth's Function:

$$f(x_1,x_2) = (x_1+2\,x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

A general form of the equation, a plot of the objective function, constraint of the objective functions and the coordinates of global minima actual and with PSO are given herein. The effect of particle size is studied on various mathematical functions.

3.4 COMPUTATIONAL RESULTS

1. Rosenbrock Function:

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

The minimum values for the rosenbrock's function are as shown below

$$\text{Minimum} = \begin{cases} n=2 \longrightarrow f(1,1) = 0, \\ n=3 \longrightarrow f(1,1,1) = 0, \\ \\ n>3 \longrightarrow f(-1,1,\ldots,1) = 0. \end{cases}$$

for $-\infty \le xi \le \infty$, $1 \le I \le n$

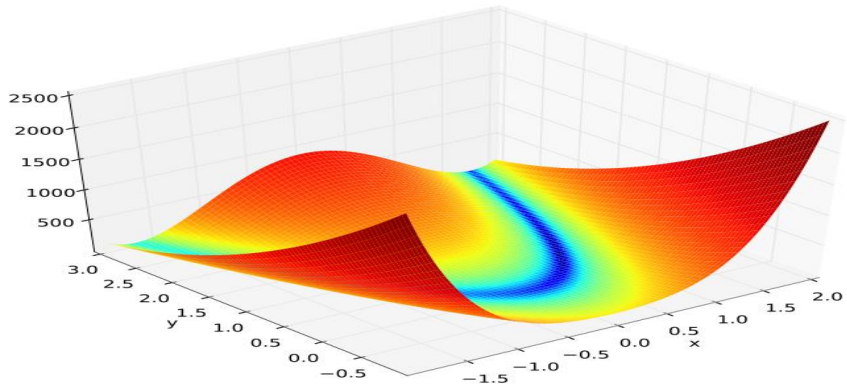


Fig 3.1: 3d Representation of Rosenbrock's Function

Here Rosenbrock function with n=2has been considered i.e.

$$f(x_1,x_2) = 100(x_2-x_1)^2 +(x_1-1)^2$$

Detail discussion to optimize Rosenbrock function with PSO is given below:

1. Generate the random position vectors for the two variables, i.e. X1{} and X2{}.
2. Generate the random velocity vectors for the two variables, i.e. V1{} and V2{}.(No. of elements in the vectors is equal to no. of particle i.e. 'p')
3. Personal best values for each particle will be their own position in the first iteration.
4. $X_{pbest1}=X_1^0$ and $X_{pbest2}=X_2^0$Gobal best value of position will be position of that particle, corresponding to which function value is minimum.
5. Now new velocity vectors can be determined for both the variables, i.e. vijk+1 for all the particle
6. Then new position vectors, i.e. X1j for all the particles using eq.
7. Again the objective function value is calculated using new position vectors of the two variables for all the particles.
8. Global best value of position will be changed to positions of that particle, corresponding to which function value is now minimum as compared to the last Global best value.
9. New personal best values are decided by comparing the last two iteration's function values.
10. Check if the difference in function values between two consecutive iterations is less than a prescribed value. If not, repeat the procedure.

2. Beale function

$$f(x_1,x_2) = (1.5-x_1+x_1x_2)2 + (2.25-x_1+x_1x_2^2)^2 + (2.625-x_1+x_1x_2^3)^2$$

The minimum values for the Beale function are as follows:

Minimum = f (3, 0.5) =0 for $-4.5 \leq X_1, X_2 \leq 4.5$

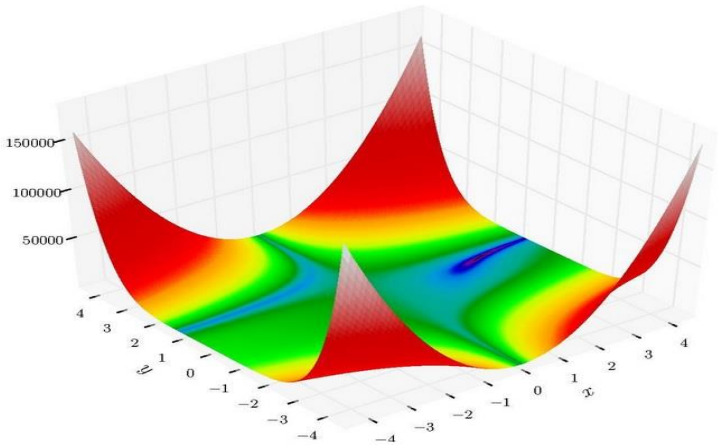


Fig 3.2: 3d Representation of Beale function

3. Sphere function

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

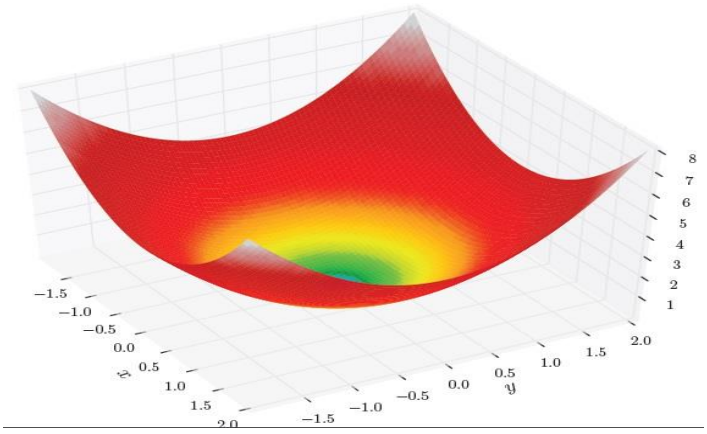


Fig 3.3: 3d Representation of Sphere function

4. Booth function

$$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

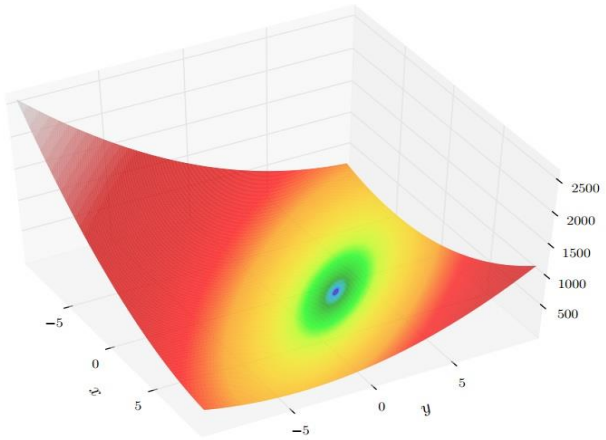


Fig 3.4: 3d Representation of Booth function

Table 3.1:Different type of PSO with varying parameter

Different PSO	**r_p**	**r_g**	**c_p**	**c_g**	**Inertia Weight Function**
SPSO	0.4	0.5	2	2	Linearly Decreasing
IPSO1	0.4	0.5	2	2	0.6
IPSO2	1	1	1	1	0.6
IPSO3	1	1	Linearly decreasing	Linearly increasing	0.6
IPSO4	1	1	Linearly increasing	Linearly decreasing	0.6

Table 3.2:Total no of iteration in different benchmark Function

Benchmark Function	SPSO	IPSO1	IPSO2	IPSO3	IPSO4
Rosenbrock	172	168	48	52	39
Beale's	253	87	91	1000	34
Sphere	202	56	31	32	30
Booth's	210	70	67	1000	1000

Here we have taken swarm size equal to 30 and maximum no.of iteration equals to 1000, while tolerance value is taken as 10^{-6}.

By table 3.2, we can conclude that if inertia weight is fixed (as in IPSO 1,2,3,4) then the no. of iteration required to achieve the desired tolerance value decreases.

IPSO 2 is better than SPSO & IPSO1, as no. of iteration required to achieve the tolerance value decreases.

In IPSO 4 :- Cp minimum to maximum

Cg maximum to minimum

Which is opposite to the conventional method. By IPSO4, we get less no. of iteration than SPSO, IPSO1, IPSO2 & IPSO3, because particle position don't get trap to local value. In IPSO3 as Cp & Cg vary according to conventional way, so to get rid of local influence, it need more iteration.

CHAPTER 4:

ECONOMIC LOAD DISPATCH

4.1 PROBLEM FORMULATION IN 2-D SPACE WITH EQUALITY CONSTRAINTS

The objective function to minimize the cost of generation is given as $F_C = \sum_{i=1}^{NG} F[C_i(P_{gi})]$ (4.1a)

Where:

$$C_i(P_{gi}) = \sum_{i=1}^{NG} a_i P_{gi}^2 + b_i P_{gi} + c_i \quad (4.1b)$$

Where:

P_{gi} is the active power generation at the i^{th} generator.

C_i is the cost of generation for i^{th} generator.

NG is the total number of generators in the system.

a_i, b_i, c_i are fuel cost coefficients of i^{th} generator.

The objective function to find the system transmission loss is given as

$$F_L = \sum_{m=1}^{NG} \sum_{n=1}^{NG} P_{gm} B_{mn} P_{gn} + \sum_{m=1}^{NG} B_{om} P_{gm} + B_{oo} \quad (4.1c)$$

Where

P_{gm}, P_{gn} is the active power at the m^{th} and n^{th} generator.

NG is the total number of generators in the system.

B_{mn}, B_{om}, B_{oo} are loss coefficients.

The cost and loss coefficients of various generators are given in Table 4.1 and 4.2.

TABLE 4.1 Values of Cost Coefficients

	Coefficient	G1	G2	G3
5-BUS	A	0.0050	0.0050	……
	B	3.510	3.889	……
	C	44.40	40.60	……
14-BUS	A	0.005	0.005	0.005
	B	3.510	3.890	2.450
	C	44.40	40.60	105
30-BUS	A	0.005	0.005	0.005
	B	3.510	3.890	2.450
	C	44.40	40.6	105

TABLE 4.2 Values of Loss Coefficients

B_{11}	0.0003489	0.0003489	0.0003069
B_{12}	0.000086	0.000068	0.0001289
B_{13}	---	-0.0000389	0.000002
B_{22}	0.000371	0.0001570	0.0001520
B_{23}	---	0.000015	0.0000110
B_{33}	---	0.000274	0.000189
B_{01}	---	0.000044	---
B_{02}	---	0.000024	---
B_{03}	---	0.000000	---
B_{00}	---	0.000254	---

The method used in this thesis, has been developed by Kron and adopted by Kirchmayer, which is the loss coefficient method.

Mathematically, the problem is to minimize

$F= [F_C (P_{g1}, P_{g2}, P_{g3}... P_{gNG})$

Where

$F = W_C F_C$ (4.1d)

Subject to the constraints:

Equality constraint

$$\sum_{i=1}^{NG} P_{gi} = P_D + P_L \qquad (4.1e)$$

Inequality constraint

$$P_{gimin} \leq P_{gi} \leq P_{gimax} \qquad i = 1, 2... NG \qquad (4.1f)$$

Power outputs from the generators are taken as the independent decision variables of the problem.

Where:

F	objective function to be optimized
F_C	cost of the generation
F_L	system transmission losses
$P_{g1}, P_{g2}... P_{gNG}$	are the generations at the generators.
P_D	is the total load demand.
P_L	is the system transmission losses.
NG	is the no. of generators.
P_{gi}	generation from i^{th} generator
P_{gimin}	minimum generation possible from i^{th} generator
P_{gimax}	maximum generation possible from i^{th} generator

4.2 COMPUTATIONAL PROCEDURE FOR APPLICATION OF PSO IN ECONOMIC LOAD DISPATCH

Particle Swarm Optimization (PSO) has been used to perform the optimization of ELD function. To consider the equality constraint of the problem, the function has been modified by inclusion of a parameter K. The objective function becomes as follows:

$$F = W_C F_C + K(P_D + P_L - P_G) \qquad (4.1g)$$

Where:

Parameter K is fixed at 50 for all three IEEE 5, 14 and 30 bus systems. Different

values of K were considered and it was observed that ELD problem converged when it was fixed to 50 for all the systems.

Inequality constraints have been considered in the PSO programming which is done in the MATLAB. The program checks the power output of each particle for each generator in each iterations and the power is tied to the corresponding limit violated. Logic to implement the inequality constraint is as shown below:

```
for i=1: NG
    for m=1: p
        if Pgi < Pgimin
          Pgi = Pgimin
        end
        if Pgi > Pgimax
        Pgi = Pgimax
        end
    end
end
```

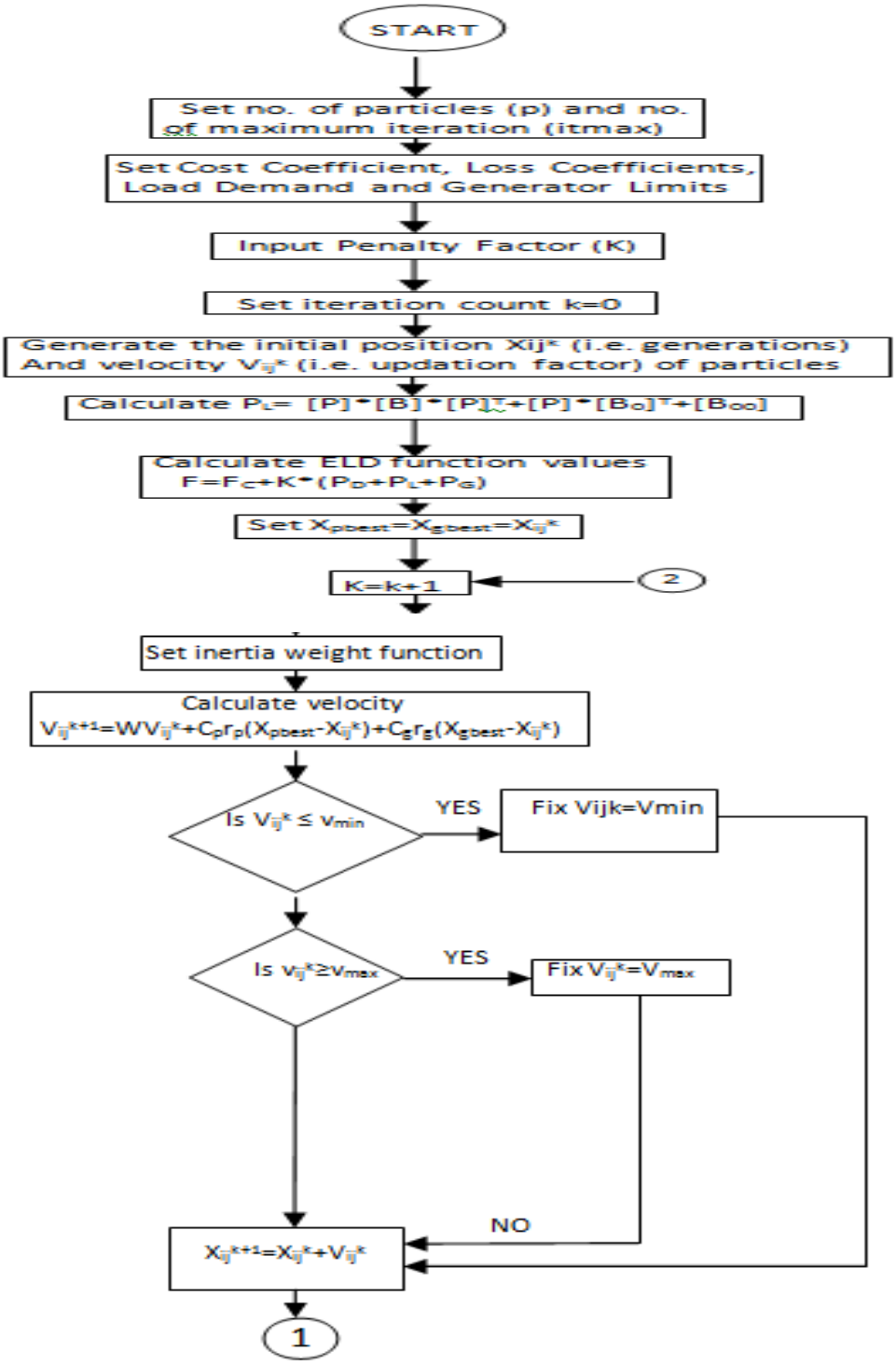
The optimum solution is obtained when the

i. Change in the value of Economic Load Dispatch function during successive iterations is less than the limit specified which is $\varepsilon=10^{-6}$ and
ii. The equality constraint is satisfied such that the absolute value of difference between generation, demand and losses is less than $\varepsilon=10^{-6}$.

Population size of the swarm and the maximum number of iteration can be selected by the user of the program in run time. We have chosen 30 particles in the swarm and 1000 as the maximum number of iterations.

With the help of MATLAB, we generate randomly the initial position and velocity of particles. To increase the convergence, limits are imposed on position of particles. Here positions i.e. the generations are decision variables. The maximum and minimum limits on the velocity have been assigned as $V_{min} = -P_{gimin}/2$ and $V_{max} = P_{gimax}/2$ respectively. The velocities are fixed to the values of corresponding limits if violated during the iterations. Initial values of personal best and global best have been taken as the initial value randomly generated by MATLAB.

Flowchart of solution of Economic Load Dispatch problem using PSO is shown in Fig. 4.1.


START
Set no. of particles (p) and no. of maximum iteration (itmax)
Set Cost Coefficient, Loss Coefficients, Load Demand and Generator Limits
Input Penalty Factor (K)
Set iteration count k=0
Generate the initial position Xij^k (i.e. generations) And velocity Vij^k (i.e. updation factor) of particles
Calculate PL= [P]*[B]*[P]T+[P]*[Bo]T+[Boo]
Calculate ELD function values F=FC+K*(PD+PL+PG)
Set Xpbest=Xgbest=Xij^k
K=k+1
2
Set inertia weight function
Calculate velocity Vij^k+1=WVij^k+CprP(Xpbest-Xij^k)+CgrG(Xgbest-Xij^k)
Is Vij^k ≤ vmin
YES
Fix Vijk=Vmin
Is vij^k≥vmax
YES
Fix Vij^k=Vmax
NO
Xij^k+1=Xij^k+Vij^k
1

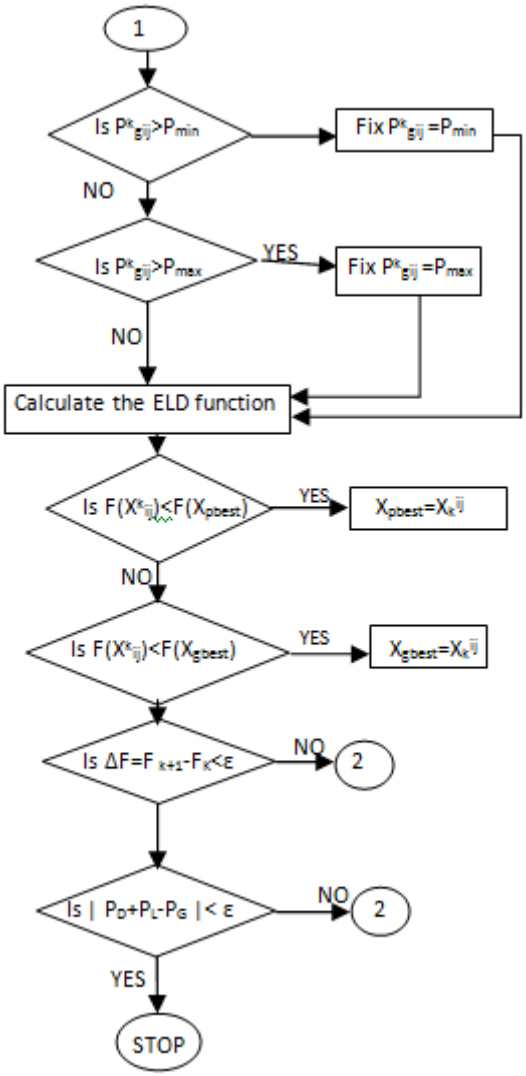


Fig. 4.1 Flow chart of implementation of PSO on ELD

The sequence for the solution of Economic Load Dispatch problem using Particle Swarm Optimization technique is explained as follows:

1. Fix the no. of particles 'p' in swarm and set the no. of maximum iterations it_{max}.
2. Fix the cost coefficients, loss coefficients, and load demand and generator limits of all the generators.
3. Generate X_{ij}^k and V_{ij}^k, the initial random positions (i.e. generations) and velocity (i.e. updation factor) respectively.
4. Set iteration count K = 0.
5. Calculate the losses for each particle, using the eq. (4.1c).
6. Calculate the value of ELD function using eq. (4.1g).

7. At 0^{th} iteration the personal and global best positions (i.e. generations) are same as the initial random positions (i.e. generations).
8. Increase the iteration count k by 1 using k=k+1.
9. Calculate the velocity (i.e. positions updation factor) of each particle using eq. (2.1a).
10. Check if velocity is within the limits. Fix the velocity to the limit violated.
11. Calculate the new positions (i.e. generations) of the particles by evaluating eq. (2.1b).
12. Check if generations (i.e. positions) of each particle are within the generator limits, if not fix the generation to the limit violated.
13. Calculate ELD function for the new positions (i.e. generations) generated.
14. Update Xpbest and Xgbest values by comparing ELD function values.
15. Check if both the stopping criteria are satisfied, if not then go to step 9, else stop.
16. Output the values of cost of generation and system transmission losses.

4.3 COMPUTATIONAL RESULTS IN 2D SPACE

Three standard test systems have been in the economic load dispatch function in order to examine the cost of generation aspects and detailed studies have been carried out in table 4.3 to 4.6.

TABLE 4.3 Variation of no of iteration required with different equality constraint parameter K for IEEE 5 bus system by taking SPSO(standard particle swarm optimization) parameter

P1	P2	C	IT	Different Value of K
82.19	82.92	764.42	240	10000
82.49	82.62	764.12	239	1000
77.05	80.10	766.70	217	500
89.48	75.64	761.98	224	100
96.85	68.35	761.14	199	50

TABLE 4.4 Variation of no of iteration required with different IPSO for IEEE 5 bus system

	P1(MW)	P2(MW)	Fc($/h)	No of iteration Required
SPSO	98.32	66.91	761.14	189
IPSO1	96.211	68.990	761.158	73
IPSO2	96.670	68.539	761.144	87
IPSO3	91.356	73.784	761.636	96
IPSO4	82.056	83.064	764.304	1000

TABLE 4.5 Variation of no of iteration required with different IPSO for IEEE 14 bus system

	P1(MW)	P2(MW)	P3(MW)	Fc($/h)	No of iteration required
SPSO	94.301	90.722	80.970	1172.844	213
IPSO1	103.578	87.379	75.374	1163.898	79
IPSO2	83.779	98.012	83.924	1184.093	95
IPSO3	94.389	102.066	69.640	1171.292	107
IPSO4	78.392	97.1392	90.079	1191.907	1000

TABLE 4.6 Variation of no of iteration required with different IPSO for IEEE 30 bus system

	P1(MW)	P2(MW)	P3(MW)	Fc($/h)	No of iteration required
SPSO	82.048	100.882	108.165	1308.926	227
IPSO1	103.184	83.014	105.609	1288.464	91
IPSO2	106.641	72.878	112.354	1290.669	112
IPSO3	95.879	98.443	97.363	1290.992	111
IPSO4	114.390	93.045	85.209	1273.327	1000

CHAPTER 5:

CONCLUSIONS AND FUTURE DIRECTION

5.1 CONCLUSIONS

The following are the significant contribution of this thesis :-

(a) Here ELD problem has been solved for IEEE 5, 14 and 30 bus systems by considering cost of generation as a objective. There has no limitation for considering more than one objective.

(b) Equality constraint of ELD problem has been considered using penalty parameter k whereas inequality constraints of the problem have been handled by the PSO programming.

(c) An optimization technique Particle Swarm Optimization has been used successfully to generate the solution for IEEE 5, 14 and 30 bus systems for ELD considering cost of generation.

5.2 FUTURE DIRECTION

Enough future scope is there to work in the area of PSO, a few aspects of possible future research work are given below:-

(a) Here one can work on the selection criteria of various constants in the PSO algorithm like acceleration constants (i.e. Cp & Cg) and random no. (i.e. r_p & r_g).

(b) Computation of inertia weight factor

(c) Consideration of other objectives of power systems like environment pollution, multiple valve effects, security etc.

APPENDIX I

1) IEEE 5 BUS SYSTEM:

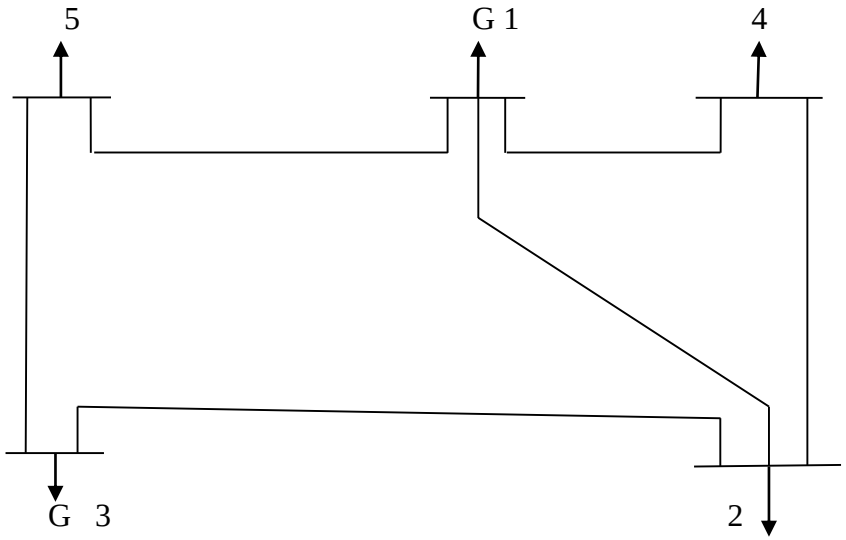


Fig. (I-A): BUS-CODE DIAGRAM 5 BUS SYSTEM

TABLE (I-A): LINE DATA or IMPEDANCE DATA (5 BUS SYSTEM)

LINE DESIGNATION	*R(p.u.)	*X(p.u.)	LINE CHARGING
1-2	0.10	0.4	0.0
1-4	0.15	0.6	0.0
1-5	0.05	0.2	0.0
2-3	0.05	0.2	0.0
2-4	0.10	0.4	0.0
3-5	0.05	0.2	0.0

*The impedance are based on MVA as 100

TABLE (I-B): BUS DATA or OPERATING CONDITION (5 BUS SYSTEM)

	GENERATION	GENERATION	LOAD	LOAD
BUS NO.	MW	VOLTAGE MAGNITUDE	MW	MVAR
1*	-----	1.02	-----	-----
2	-----	-----	60	30
3	100	1.04	-----	-----
4	-----	-----	40	10
5	-----	-----	60	20

*Slack Bus

TABLE (I-C): REGULATED BUS DATA (5 BUS SYSTEM)

BUS NO.	VOLTAGE MAGNITUDE	MINIMUM MVAR CAPACITY	MAXIMUM MVAR CAPACITY	MINIMUM MVAR CAPACITY	MAXIMUM MVAR CAPACITY
1	1.02	0.0	60	30	120
3	1.04	0.0	60	30	120

The nodal load voltage inequality constraints are $0.9 \leq Vi \leq 1.05$

Cost characteristics

The cost characteristics of the IEEE 5 Bus System are as follows:

$C_1=50p_1^2+351p_1+44.4$ \$/hr

$C_3=50p_3^2+389p_3+40.6$ \$/hr

Here, the total load demand of the system is 160 MW. Maximum and minimum active power constraint on the generator bus for the given system is 120 MW and 30 MW respectively. Voltage magnitude constraint for generator at bus 3 is 1.04 pu.

M-file For Calculating B- Coefficients:

```
Clear
basemva=100
accuracy=0.0001
maxiter=10
busdata=[1 1 1.02 0 0 0 0 0 0 60 0;2 0 1 0 60 30 0 0 0 0 0;3 2 1.04 0 0 0 82 0 0 60 0;4
0 1 0 40 10 0 0 0 0 0;5 0 1 0 60 20 0 0 0 0 0];
```

Linedata=[1 2 0.10 0.4 0 1;1 4 0.15 0.6 0 1; 1 5 0.05 0.2 0 1;2 3 0.05 0.2 0 1;2 4 0.10 0.4 0 1;3 5 0.05 0.2 0 1];

disp(busdata)

disp(linedata)

mwlimit=[30 120;30 120];

Ifybus

Ifnewton

busout

bloss

B-Coefficient Calculated are as:

B11 = 0.00035336

B12 = 0.0000103196

B21 = 0.0000103196

B22 = 0.000368992

2) IEEE 14 BUS SYSTEM:

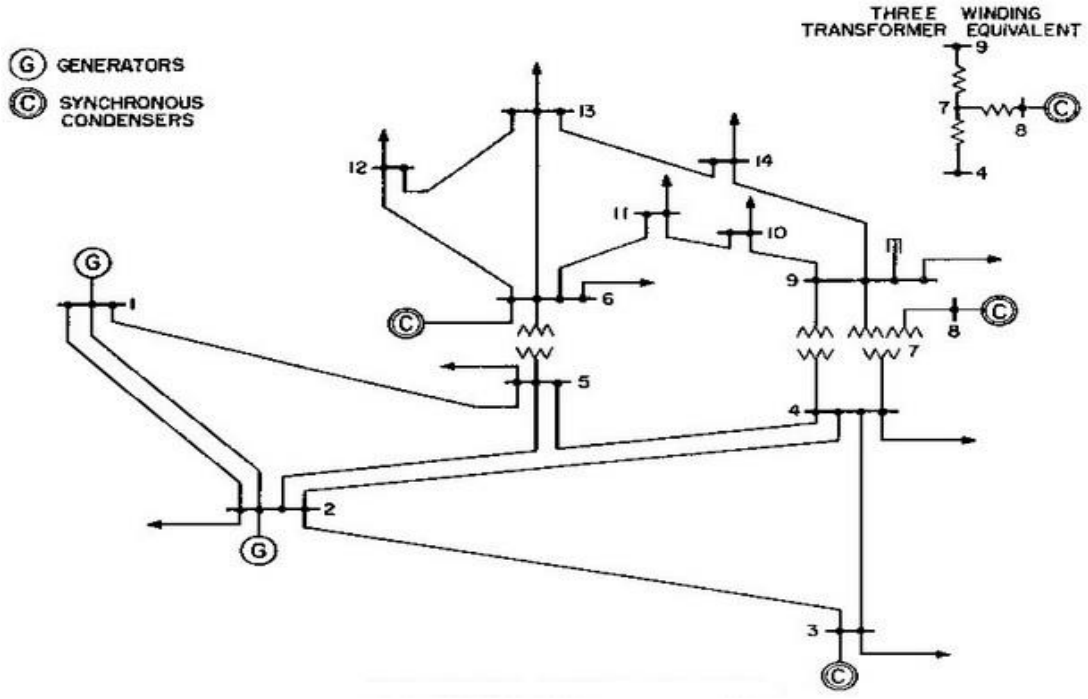


Fig. (I-B): BUS-CODE DIAGRAM 14 BUS SYSTEM

TABLE (I-D): IMPEDANCE AND LINE-CHARGING DATA (14 BUS SYSTEM)

Line Designation	Resistance p.u. *	Reactance p.u. *	Line Charging	Tap Setting
1-2	0.019379	0.059170	0.0264	1
1-5	0.054029	0.223040	0.0246	1
2-3	0.046980	0.197970	0.0219	1
2-4	0.058110	0.176320	0.0187	1
2-5	0.056950	0.173880	0.0170	1
3-4	0.067010	0.171030	0.0173	1
4-5	0.013350	0.042110	0.0064	1
4-7	0	0.20912	0	1
4-9	0	0.55618	0	1
5-6	0	0.25202	0	1
6-11	0.09498	0.19890	0	1
6-12	0.12291	0.25581	0	1
6-13	0.06615	0.13027	0	1
7-8	0	0.17615	0	1
7-9	0	0.11001	0	1
9-10	0.03181	0.08450	0	1
9-14	0.12711	0.27038	0	1
10-11	0.08205	0.19207	0	1
12-13	0.22092	0.19988	0	1
13-14	0.17093	0.34802	0	1

*** Impedance and line-charging susceptance in p.u. on a 100 MVA base. Line charging one-half of the total charging of line.**

TABLE (I-E): BUS DATA OR OPERATING CONDITIONS (14 BUS SYSTEM)

Bus no	Voltage		Generation	Generation	Load	Load
	Magnitude(p.u.)	Phase angle (deg.)	MW	MVAR	MW	MVAR
1*	1.06	0	0	0	0	0
2	1	0	40	0	21.7	12.7
3	1	0	0	0	94.2	19.0
4	1	0	0	0	47.8	-3.9
5	1	0	0	0	7.6	1.6
6	1	0	0	0	11.2	7.5
7	1	0	0	0	0	0
8	1	0	0	0	0	0
9	1	0	0	0	29.5	16.6
10	1	0	0	0	9.0	5.8
11	1	0	0	0	3.5	1.8
12	1	0	0	0	6.1	1.6
13	1	0	0	0	13.5	5.8
14	1	0	0	0	14.9	5.0

*Slack Bus

TABLE (I-F): REGULATED BUS DATA (14 BUS SYSTEM)

Bus no.	Votage magnitude (p.u.)	Minimum MVAR capability	Maximum MVAR capability
2	1.05	-40	50
3	1.010	0	40
6	1.070	-6	24
8	1.090	-6	24

Cost characteristics:

The cost characteristics of the IEEE 14 Bus System are as follows:

$C_1=50p_1^2+245p_1+105$ \$/hr

$C_2=50p_1^2+351p_2+44.4$ \$/hr

$C_6=50p_6^2+389p_6+40.6$ \$/hr

Here, the total load demand of the system is 259 MW. Maximum and minimum active power constraint on the generator bus for the given system is 150 MW and 50 MW

respectively. Voltage magnitude constraint for generator at bus 2 is 1.045, for bus no. 6 is 1.070, for bus no. 3 is 1.010 & for bus no. 8 is 1.090

M-file For Calculating B- Coefficients:

```
Clear
basemva=100
accuracy=0.0001
maxiter=10
busdata=[1 1 1.06 0 0 150 0 0 0 0 0;2 2 1.045 0 21.7 12.7 63.11 0 -40 50 0;3 0 1.01 0 94.2 19 0 0 0 40 0;4 0 1 0 47.8 -3.9 0 0 0 0 0;5 0 1 0 7.6 1.6 0 0 0 0 0;6 2 1.07 0 11.2 7.5 77.12 0 -6 24 0;7 0 1  0 0 0 0 0 0 0 0;8 0 1.09 0 0 0 0 0 -6 24 0 ;9 0 1 0 29.5 16.6 0 0 0 0 0; 10 0 1 0 9 5.8 0 0 0 0 0;11 0 1 0 3.5 1.8 0 0 0 0 0;12 0 1 0  6.1 1.6 0 0 0 0 0;13 0 1 0 13.5 5.8 0 0 0 0 0;14 0 1 0 14.9 5 0 0 0 0 0];
linedata=[1 2 0.01938 0.05917 0.0264 1;1 5 0.05403 0.22304 0.0246 1; 2 3 0.04699 0.19797 0.0219 1; 2 4 0.05811 0.17632 0.0170 1; 2 5 0.05695 0.17388 0.0173 1; 3 4 0.06701 0.17103 0.0064 1; 4 5 0.01335  0.04211 0.0 1; 4 7 0.0 0.20912 0.0 0.978; 4 9 0.0 0.55618 0.0 0.969;5 6 0.0 0.25202 0.0 0.932; 6 11 0.09498 0.19890 0.0 1;6 12 0.12291 0.25581 0.0 1;6 13 0.06615 0.13027 0.0 1;7 8 0.0  0.17615  0.0 1; 7 9 0.0 0.11001 0.0 1; 9 10 0.03181 0.08450 0.0 1;9 14  0.12711  0.27038  0.0 1; 10 11 0.08205 0.19207  0.0 1;12 13 0.22092 0.19988 0.0 1;13 14 0.17093 0.34802 0.0 1];
disp(busdata)
disp(linedata)
mwlimit=[50 150;50 150;50 150]
Ifybus
Ifnewton
busout
bloss
```

B-Coefficient Calculated are as:

B11 = 0.0231

B12 = 0.0078

B13 = -0.0007

B21 = 0.0078

B22=0.0182
B23= 0.0022
B31=-0.0007
B32= 0.0022
B33= 0.0329

IEEE 30 BUS SYSTEM

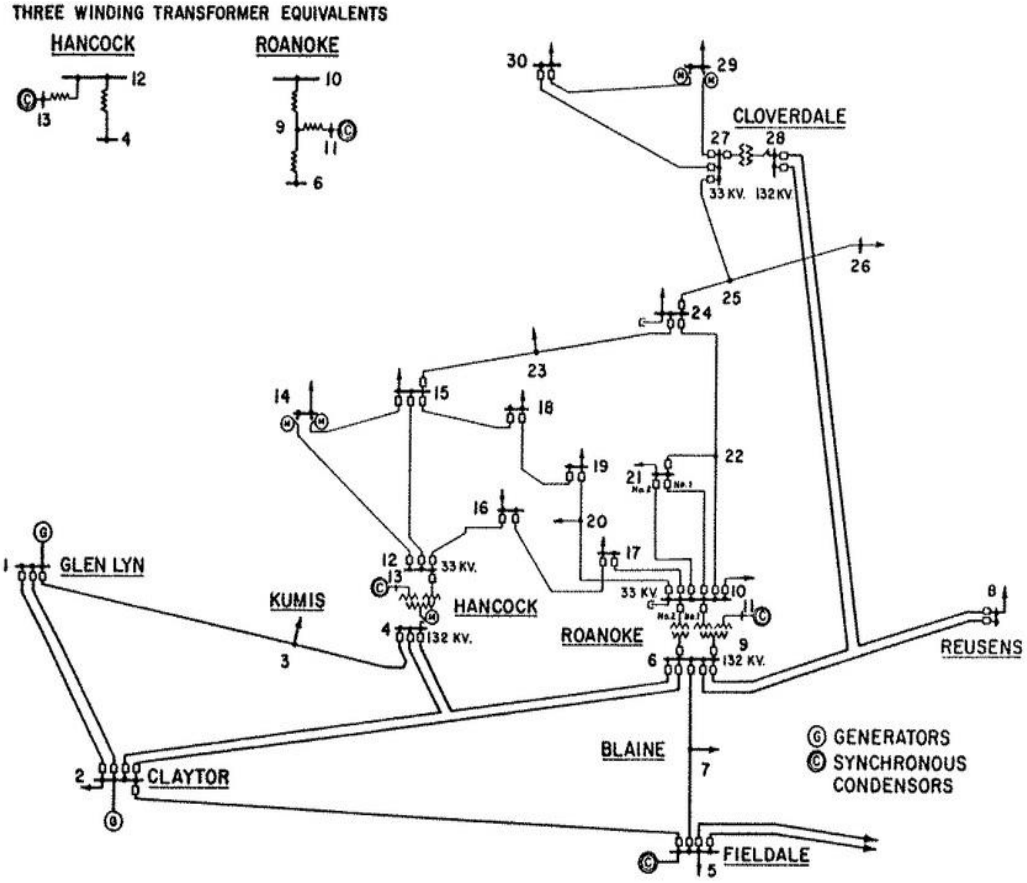


Fig. (I-C): BUS-CODE DIAGRAM 30 BUS SYSTEM

TABLE (I-G): IMPEDANCE AND LINE-CHARGING DATA (30 BUS SYSTEM)

Line Designation	Resistance p.u.*	Reactance p.u.*	Line Charging	TAp Setting
1-2	0.0192	0.0575	0.0264	1
1-3	0.0452	0.1852	0.0204	1
2-4	0.0570	0.1737	0.0184	1
3-4	0.0132	0.0379	0.0042	1
2-5	0.0472	0.1983	0.0209	1
2-6	0.0581	0.1763	0.0187	1
4-6	0.0119	0.0414	0.0045	1
5-7	0.0460	0.1160	0.0102	1
6-7	0.0267	0.0820	0.0085	1
6-8	0.0120	0.0420	0.0045	1
6-9	0	0.2080	0	0.978
6-10	0	0.5560	0	0.969
9-11	0	0.2080	0	1
9-10	0	0.1100	0	1
4-12	0	0.2560	0	0.932
12-13	0	0.1400	0	1
12-14	0.1231	0.2559	0	1
12-15	0.0662	0.1304	0	1
12-16	0.0945	0.1987	0	1
14-15	0.2210	0.1997	0	1
16-17	0.0824	0.1923	0	1
15-18	0.1070	0.2185	0	1
18-19	0.0639	0.1292	0	1
19-20	0.0340	0.0680	0	1
10-20	0.0936	0.2090	0	1
10-17	0.0324	0.0845	0	1
10-21	0.0348	0.0749	0	1
10-22	0.0727	0.1499	0	1
21-22	0.0116	0.0236	0	1
15-23	0.1000	0.2020	0	1
22-24	0.1150	0.1790	0	1
23-24	0.1320	0.2700	0	1
24-25	0.1885	0.3292	0	1
25-26	0.2544	0.3800	0	1
25-27	0.1093	0.2087	0	1
27-28	0	0.3960	0	0.968
27-29	0.2198	0.4153	0	1
27-30	0.3202	0.6027	0	1
29-30	0.2399	0.4533	0	1
8-28	0.0636	0.2000	0.0214	1
6-28	0.0169	0.0599	0.0065	1

*** Impedance and line-charging susceptance in p.u. on a 100 MVA base. Line charging one-half of the total charging of line.**

TABLE (I-H): BUS DATA OR OPERATING CONDITIONS (30 BUS SYSTEM)

Bus no.	voltage		Generation	Generation	Load	Load
	Magnitude (p.u.)	Phase Angle	MW	MVAR	MW	MVAR
1*	1.06	0	0	0	0	0
2	1	0	40	0	21.7	12.7
3	1	0	0	0	2.4	1.2
4	1	0	0	0	7.6	1.6
5	1	0	0	0	94.2	19.0
6	1	0	0	0	0	0
7	1	0	0	0	22.8	10.9
8	1	0	0	0	30.0	30.0
9	1	0	0	0	0	0
10	1	0	0	0	5.8	2.0
11	1	0	0	0	0	0
12	1	0	0	0	11.2	7.5
13	1	0	0	0	0	0
14	1	0	0	0	6.2	1.6
15	1	0	0	0	8.2	2.5
16	1	0	0	0	3.5	1.8
17	1	0	0	0	9.0	5.8
18	1	0	0	0	3.2	0.9
19	1	0	0	0	9.5	3.4
20	1	0	0	0	2.2	0.7
21	1	0	0	0	17.5	11.2
22	1	0	0	0	0	0
23	1	0	0	0	3.2	1.6
24	1	0	0	0	8.7	6.7
25	1	0	0	0	0	0
26	1	0	0	0	3.5	2.3
27	1	0	0	0	0	0
28	1	0	0	0	0	0
29	1	0	0	0	2.4	0.9
30	1	0	0	0	10.6	1.9

*Slack Bus

TABLE (I-I): REGULATED BUS DATA (30 BUS SYSTEM)

Bus no	Voltage magnitude p.u.	Mimimum MVAR capability	Maximum MVAR capability
2	1.045	-40	50
5	1.01	-40	40
8	1.01	-10	40
11	1.082	-6	24
13	1.071	-6	24

TABLE (I-J): TRANSFORMER DATA (30 BUS SYSTEM

Transformer designation	Tap setting*
4-12	0.932
6-9	0.978
6-10	0.969
28-27	0.968

*** Off nominal turns ratio, as determined by the actual transformer-tap position and the voltage bases. In the case of nominal turns ratio, this would equal to 1.**

TABLE (I-K): STATIC CAPACITOR DATA (30 BUS SYSTEM

Bus no	Susceptance*p.u.
10	0.19
24	0.043

* Susceptance in p.u. on 100 MVA base.

Cost characteristics:

The cost characteristics of the IEEE 14 Bus System are as follows:

$C_1=50p_1^2+245p_1+105$ \$/hr

$C_2=50p_1^2+351p_2+44.4$ \$/hr

$C_6=50p_8^2+389p_8+40.6$ \$/hr

Here, the total load demand of the system is 259 MW. Maximum and minimum active power constraint on the generator bus for the given system is 150 MW and 50 MW respectively. Voltage magnitude constraint for generator at bus 2 is 1.045, for bus no. 5 is 1.01, for bus no. 8 is 1.010, for bus no. 11 is 1.082 &for bus no. 13 is 1.071

M-file For Calculating B- Coefficients:

```
Clear
basemva=100
accuracy=0.0001
maxiter=10
busdata=[1 1 1.06 0 0 0 0 0 0 0 0;2 2 1.045 0 21.7 12.7 90 0 -40 50 0; 3 0 1 0 2.4 1.2 0
0 0 0 0;4 0 1 0 7.6 1.6 0 0 0 0 0;5 0 1.01 0 94.2 19 0 0 -40 40 0; 6 0 1 0 0 0 0 0 0 0 0;
7 0 1 0 22.8 10.9 0 0 0 0 0;8 2 1.010 30 30150 0 -10 40 0; 9 0 1 0 0 0 0 0 0 0 0; 10 0 1
0 5.8 2 0 0 0 0 0.19; 11 0 1.082 0 0 0 0 0 -6 24 0; 12 0 1 0 11.2 7.5 0 0 0 0 0; 13 0
1.071 0 0 0 0 0 -6 24 0; 14 0 1 0 6.2 1.6 0 0 0 0 0;15 0 1 0 8.2 2.5 0 0 0 0 0;16 0 1 0
3.5 1.8 0 0 0 0 0; 17 0 1 0 9 5.8 0 0 0 0 0; 18 0 1 0 3.2 0.9 0 0 0 0 0; 19 0 1 0 9.5 3.4 0
0 0 0 0; 20 0 1 0 2.2 0.7 0 0 0 0 0;21 0 1 0 17.5 11.2 0  0 0 0 0;22 0 1 0 0 0 0 0 0 0
0;23 1 0 3.2 1.6 0 0 0 0 0; 24 0 1 0 8.7 6.7 0 0 0 0 0.043; 25 0 1 0 0 0 0 0 0 0 0;26 0 1
0 3.5 2.3 0 0 0 0 0; 27 0 1 0 0 0 0 0 0 0 0; 28 0 1 0 0 0 0 0 0 0 0;29 0 1 0 2.4 0.9 0 0 0
0 0; 30 0 1 0 10.6 1.9 0 0 0 0 0];
linedata=[1 2 0.0192 0.0575 0.0264 1;1 3 0.0452 0.1852 0.0204 1; 2 4 0.0570 0.19797
0.0219 1; 2 4 0.05811 0.17632 0.0170 1; 2 5 0.05695 0.17388 0.0173 1; 3 4 0.06701
0.17103 0.0064 1; 4 5 0.01335  0.04211 0.0 1; 4 7 0.0 0.20912 0.0 0.978; 4 9 0.0
0.55618 0.0 0.969;5 6 0.0 0.25202 0.0 0.932; 6 11 0.09498 0.19890 0.0 1;6 12 0.12291
0.25581 0.0 1;6 13 0.06615 0.13027 0.0 1;7 8 0.0  0.17615  0.0 1; 7 9 0.0  0.11001 0.0
1; 9 10 0.03181 0.08450 0.0 1;9 14  0.12711  0.27038  0.0 1; 10 11 0.08205 0.19207
0.0 1;12 13 0.22092 0.19988 0.0 1;13 14 0.17093 0.34802 0.0 1];

disp(busdata)
```

disp(linedata)

mwlimit=[50 150;50 150;50 150]

Ifybus

Ifnewton

busout

bloss

B-Coefficient Calculated are as:

B11 = 0.0307

B12 = 0.0129

B13 = 0.0002

B21 = 0.0129
B22=0.0152
B23= - 0.0011
B31=0.0002
B32=- 0.0011
B33= 0.0190

APPENDIX II

(1) MATLAB Program for optimization of benchmark functions using PSO.

Code for the Optimization of Rosenbrock's Function using SPSO

```
clc
disp
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
x1=zeros(p,it);
x2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
fp=zeros(1,p);
df=zeros(1,(it-1));
rp=0.4;
rg=0.5;
cp=2;
cg=2;
T=input('Enter the tolerance value');
% Initial values i.e. 0th iteration
% x1(:,1)=[1.2886 0.7572 1.6232 1.0657 0.7015 1.8780 1.7519 1.1003 1.2450 1.1741];
% x2(:,1)=[0.4155 0.6025 0.9418 0.4610 1.6886 0.3895 0.4518 0.3414 0.4553 0.8714];
% v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816 0.1564 0.8555 0.6448 0.3763 0.1909
0.4283 0.4820 0.1206 0.5895 0.2262 0.3846 0.5830];
% v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439 0.5841 0.1078 0.9063 0.8797 0.8178
0.2607 0.5944 0.0225 0.4253 0.3127 0.1615 0.1788];
x1(:,1)=unifrnd(0.5,1.5,1,p);
x2(:,1)=unifrnd(0.5,1.5,1,p);
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);
% i=0;
% disp (sprintf('enter the values of %dth iteration positions of %d particles for variable x1',i,p))
% for j=1:p
%     x1(j,1)=input(sprintf('enter the value of x1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable x2',p))
% for j=1:p
%     x2(j,1)=input(sprintf('enter the value of x2(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v1',p))
% for j=1:p
%     v1(j,1)=input(sprintf('enter the value of v1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v2',p))
% for j=1:p
%     v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
% end
for j=1:p
   f(j,1)= 100*( (x1(j,1))^2 - x2(j,1) )^2 + (1- x1(j,1))^2 ;
end
 %Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for k=1:p
   if f(k,1)==fmin
      gb=k;
   else
```

```
    end
end
%Initial global best value
x1g=zeros(p);
x2g=zeros(p);
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
end
fgm = min(f(:,1));

% fig=zeros(1,485);
% t=zeros(1,485);


for i=1:it
    disp(sprintf('This is %d no. of iteration',i))

%for inertia weight W
    wmax=0.9;
    wmin=0.4;
    w = wmax-i*((wmax-wmin)/it);

    for j=1:p
        v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g(j)-x1(j,i));
        v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-x2(j,i));
        x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
        x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
        f(j,(i+1))= 100*( (x1(j,(i+1)))^2 - x2(j,(i+1)) )^2 + (1- x1(j,(i+1)))^2 ;
    end

%To find change in the values of f
    for j=1:p
        df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
    end

%personal best values updation
    for j=1:p
        fp(j)= 100*( (x1p(j))^2 - x2p(j) )^2 + (1- x1p(j))^2 ;
    end
    for k=1:p
        if f(k,i)< fp(k)
            x1p(k)=x1(k,i);
            x2p(k)=x2(k,i);
        else
        end
    end

%for Global best values updation
    if min(f(:,(i+1)))<fgm
        fgm=min(f(:,(i+1)));
    else
    end

    for j=1:i
        for k=1:p
        if f(k,i)==fgm
```

```
            for l=1:p
                x1g(l) = x1(k,i);     %global best values
                x2g(l) = x2(k,i);
                %plot(x1g,x2g);
            end
        else
        end
        end
    end

    print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
    disp('     x1        x2        v1        v2        f')
    disp(print)

%     fig(i) = figure('Position', [100 100 500 350]);
%     t(i) = uitable('Parent', fig(i), 'Position', [25 25 450 200]);
%     print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
%     set(t(i), 'Data', print);
%     set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});

%Stoping criterion
    ki=0;
    for j=1:p
        if (df(j,i)<=10^(-T))
            ki=ki+1;
        end
    end
    if ki >= p
        break
    end

end
[r,c]=find(f==fgm);
minf=100*( (x1g(j))^2 - x2g(1) )^2 + (1- x1g(1))^2;
disp(sprintf('min value of function is %d and at values of x1=%d and x2=%d ',minf,x1g(1),x2g(1)))
z=1:1:200;
for z=1:50
    plot(z,f(1,z),'o');
pause(0.2)
end
```

Code for the Optimization of Rosenbrock's Function using IPSO

```
clc
disp(' we have to minimize f = 100(x1^2-x2)^2+(1-x1)^2 i.e. rosenbrock function')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
x1=zeros(p,it);
x2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
fp=zeros(1,p);
df=zeros(1,(it-1));
rp=1;
rg=1;
%cp=2;
%cg=2;
cpmax=input('cpmax= ');
```

```
cpmin=input('cpmin= ');
cgmax=input('cgmax= ');
cgmin=input('cgmin= ');
T=input('Enter the tolerance value');
% Initial values i.e. 0th iteration
% x1(:,1)=[1.2886 0.7572 1.6232 1.0657 0.7015 1.8780 1.7519 1.1003 1.2450 1.1741];
% x2(:,1)=[0.4155 0.6025 0.9418 0.4610 1.6886 0.3895 0.4518 0.3414 0.4553 0.8714];
% v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816 0.1564 0.8555 0.6448 0.3763 0.1909
0.4283 0.4820 0.1206 0.5895 0.2262 0.3846 0.5830];
% v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439 0.5841 0.1078 0.9063 0.8797 0.8178
0.2607 0.5944 0.0225 0.4253 0.3127 0.1615 0.1788];
x1(:,1)=unifrnd(0.5,1.5,1,p);
x2(:,1)=unifrnd(0.5,1.5,1,p);
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);
% i=0;
% disp (sprintf('enter the values of %dth iteration positions of %d particles for variable x1',i,p))
% for j=1:p
%     x1(j,1)=input(sprintf('enter the value of x1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable x2',p))
% for j=1:p
%     x2(j,1)=input(sprintf('enter the value of x2(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v1',p))
% for j=1:p
%     v1(j,1)=input(sprintf('enter the value of v1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v2',p))
% for j=1:p
%     v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
% end
for j=1:p
   f(j,1)= 100*( (x1(j,1))^2 - x2(j,1) )^2 + (1- x1(j,1))^2 ;
end
 %Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for k=1:p
   if f(k,1)==fmin
      gb=k;
   else
   end
end
%Initial global best value
x1g=zeros(p);
x2g=zeros(p);
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
end
fgm = min(f(:,1));

% fig=zeros(1,485);
% t=zeros(1,485);
```

```
for i=1:it
   disp(sprintf('This is %d no. of iteration',i))

%for inertia weight W
   wmax=0.6;
   wmin=0.6;
   w = wmax-i*((wmax-wmin)/it);

   cg=cgmax-i*((cgmax-cgmin)/it);
   cp=cpmin-i*((cpmin-cpmax)/it);
   disp(sprintf('cp= %d ',cp))
   disp(sprintf('cg= %d ',cg))

   for j=1:p
      v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g(j)-x1(j,i));
      v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-x2(j,i));
      x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
      x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
      f(j,(i+1))= 100*( (x1(j,(i+1)))^2 - x2(j,(i+1)) )^2 + (1- x1(j,(i+1)))^2 ;
   end

%To find change in the values of f
   for j=1:p
      df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
   end

%personal best values updation
   for j=1:p
      fp(j)= 100*( (x1p(j))^2 - x2p(j) )^2 + (1- x1p(j))^2 ;
   end
   for k=1:p
      if f(k,i)< fp(k)
         x1p(k)=x1(k,i);
         x2p(k)=x2(k,i);
      else
      end
   end

%for Global best values updation
   if min(f(:,(i+1)))<fgm
      fgm=min(f(:,(i+1)));
   else
   end

   for j=1:i
      for k=1:p
      if f(k,i)==fgm
         for l=1:p
            x1g(l) = x1(k,i);     %global best values
            x2g(l) = x2(k,i);
            %plot(x1g,x2g);
         end
      else
      end
      end
   end
```

```
    print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
    disp('    x1       x2       v1       v2       f')
    disp(print)

%    fig(i) = figure('Position', [100 100 500 350]);
%    t(i) = uitable('Parent', fig(i), 'Position', [25 25 450 200]);
%    print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
%    set(t(i), 'Data', print);
%    set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});

%Stoping criterion
    ki=0;
    for j=1:p
        if (df(j,i)<=10^(-T))
            ki=ki+1;
        end
    end
    if ki >= p
        break
    end

end
[r,c]=find(f==fgm);
minf=100*( (x1g(j))^2 - x2g(1) )^2 + (1- x1g(1))^2;
%disp(sprintf('min value of function is %d and at values of x1=%d and x2=%d ',minf,x1g(1),x2g(1)))
disp(sprintf('          R  E  S  U  L  T  S %d'))
disp(sprintf('       x1=%d', x1g(1)));
disp(sprintf('       x2=%d', x2g(1)));
disp(sprintf('       F=%d', minf(1)));
disp(sprintf('       it=%d', i));
z=1:1:200;
for z=1:50
    plot(z,f(1,z),'o');
pause(0.2)
end
```

Code for the Optimization of Beale's Function using SPSO

```
clc
disp(' we have to minimize f = (1.5-x1+x1*x2)^2+(2.25-x1+x1*x2^2)^2+(2.625-x1+x1*x2^3)^2 i.e. beale
function')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
x1=zeros(p,it); %no. of iterations are pre decided that is it will be maximum 50
x2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
fp=zeros(1,p);
df=zeros(1,(it-1));
rp=0.4;
rg=0.5;
cp=2;
cg=2;
T=input('Enter the tolerance value');
% Initial values i.e. 0th iteration
% x1(:,1)=[1.5605 0.7795 0.4834 0.8078 0.1929 0.2639 1.8841 1.9123 1.1504 0.1196 0.4696 0.7063 1.6424
0.0308 0.0860 0.3380 1.2982 1.4634 1.2955 0.9018];
```

```
% x2(:,1)=[1.0940 0.5926 1.4894 0.3779 1.3736 0.3670 0.7370 1.2512 1.5605 0.1623 1.8588 1.5514 0.9736
0.8717 0.8936 0.6127 1.0170 1.0215 1.6353 1.5897];
% v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816 0.1564 0.8555 0.6448 0.3763 0.1909
0.4283 0.4820 0.1206 0.5895 0.2262 0.3846 0.5830];
% v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439 0.5841 0.1078 0.9063 0.8797 0.8178
0.2607 0.5944 0.0225 0.4253 0.3127 0.1615 0.1788];
x1(:,1)=unifrnd(0,4.5,1,p);
x2(:,1)=unifrnd(0,4.5,1,p);
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);
% i=0;
% disp (sprintf('enter the values of %dth iteration positions of %d particles for variable x1',i,p))
% for j=1:p
%     x1(j,1)=input(sprintf('enter the value of x1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable x2',p))
% for j=1:p
%     x2(j,1)=input(sprintf('enter the value of x2(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v1',p))
% for j=1:p
%     v1(j,1)=input(sprintf('enter the value of v1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v2',p))
% for j=1:p
%     v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
% end
for j=1:p
   f(j,1)=(1.5-x1(j,1)+x1(j,1)*x2(j,1))^2+(2.25-x1(j,1)+x1(j,1)*x2(j,1)^2)^2+(2.625-
x1(j,1)+x1(j,1)*x2(j,1)^3)^2;
end


%Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for k=1:p
   if f(k,1)==fmin
      gb=k;
   else
   end
end
%Initial global best value
x1g=zeros(p);
x2g=zeros(p);
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
end
fgm = min(f(:,1));

% fig=zeros(1,485);
% t=zeros(1,485);
```

```
for i=1:it
   disp(sprintf('This is %d no. of iteration',i))

%for inertia weight W
   wmax=0.9;
   wmin=0.4;
   w = wmax-i*((wmax-wmin)/it);

   for j=1:p
      v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g(j)-x1(j,i));
      v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-x2(j,i));
      x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
      x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
      f(j,(i+1))=(1.5-x1(j,(i+1))+x1(j,(i+1))*x2(j,(i+1)))^2+(2.25-
x1(j,(i+1))+x1(j,(i+1))*x2(j,(i+1))^2)^2+(2.625-x1(j,(i+1))+x1(j,(i+1))*x2(j,(i+1))^3)^2;
   end

%To find change in the values of f
   for j=1:p
      df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
   end

%personal best values updation
   for j=1:p
      f(j)=(1.5-x1p(j)+x1p(j)*x2p(j))^2+(2.25-x1p(j)+x1p(j)*x2p(j)^2)^2+(2.625-x1p(j)+x1p(j)*x2p(j)^3)^2;
   end
   for k=1:p
      if f(k,i)< fp(k)
         x1p(k)=x1(k,i);
         x2p(k)=x2(k,i);
      else
      end
   end

%for Global best values updation
   if min(f(:,(i+1)))<fgm
      fgm=min(f(:,(i+1)));
   else
   end

   for j=1:i
      for k=1:p
      if f(k,i)==fgm
         for l=1:p
            x1g(l) = x1(k,i);     %global best values
            x2g(l) = x2(k,i);
         end
      else
      end
      end
   end

   print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
   disp('    x1       x2       v1       v2       f')
   disp(print)

%     fig(i) = figure('Position', [100 100 500 350]);
%     t(i) = uitable('Parent', fig(i), 'Position', [25 25 450 200]);
```

```
%     print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
%     set(t(i), 'Data', print);
%     set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});

%Stoping criterion
   ki=0;
   for j=1:p
      if (df(j,i)<=10^(-T))
         ki=ki+1;
      end
   end
   if ki >= p
      break
   end

end
[r,c]=find(f==fgm);
disp(sprintf('min value of function is %d and at values of x1=%d and x2=%d ',fgm,x1(r,c),x2(r,c)))
```

Code for the Optimization of Beale's Function using IPSO

```
clc
disp(' we have to minimize f = 100(x1^2-x2)^2+(1-x1)^2 i.e. rosenbrock function')
%p=input('Enter the no. of particles in a swarm');          %no. of particles
p=30;
%it=input('Enter the no. of iterations');
it=5000;
x1=zeros(p,it);
x2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
fp=zeros(1,p);
df=zeros(1,(it-1));
rp=1;
rg=1;
%cp=2;
%cg=2;
cpmax=input('cpmax= ');
cpmin=input('cpmin= ');
cgmax=input('cgmax= ');
cgmin=input('cgmin= ');
T=input('Enter the tolerance value');
% Initial values i.e. 0th iteration
% x1(:,1)=[1.2886 0.7572 1.6232 1.0657 0.7015 1.8780 1.7519 1.1003 1.2450 1.1741];
% x2(:,1)=[0.4155 0.6025 0.9418 0.4610 1.6886 0.3895 0.4518 0.3414 0.4553 0.8714];
% v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816 0.1564 0.8555 0.6448 0.3763 0.1909
0.4283 0.4820 0.1206 0.5895 0.2262 0.3846 0.5830];
% v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439 0.5841 0.1078 0.9063 0.8797 0.8178
0.2607 0.5944 0.0225 0.4253 0.3127 0.1615 0.1788];
x1(:,1)=unifrnd(0.5,4.5,1,p);
x2(:,1)=unifrnd(0.5,4.5,1,p);
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);
% i=0;
% disp (sprintf('enter the values of %dth iteration positions of %d particles for variable x1',i,p))
% for j=1:p
%     x1(j,1)=input(sprintf('enter the value of x1(%d,%d)',j,i));
% end
```

```
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable x2',p))
% for j=1:p
%     x2(j,1)=input(sprintf('enter the value of x2(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v1',p))
% for j=1:p
%     v1(j,1)=input(sprintf('enter the value of v1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v2',p))
% for j=1:p
%     v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
% end
for j=1:p
   %f(j,1)= 100*( (x1(j,1))^2 - x2(j,1) )^2 + (1- x1(j,1))^2 ;
   f(j,1)=(1.5-x1(j,1)+x1(j,1)*x2(j,1))^2+(2.25-x1(j,1)+x1(j,1)*x2(j,1)^2)^2+(2.625-
x1(j,1)+x1(j,1)*x2(j,1)^3)^2;
   %f(j,1)=(x1(j,1)+2*(x2(j,1))-7)^2+(2*(x1(j,1))+x2(j,1)-5)^2;

end
 %Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for k=1:p
   if f(k,1)==fmin
      gb=k;
   else
   end
end
%Initial global best value
x1g=zeros(p);
x2g=zeros(p);
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
end
fgm = min(f(:,1));

% fig=zeros(1,485);
% t=zeros(1,485);


for i=1:it
   disp(sprintf('This is %d no. of iteration',i))

%for inertia weight W
   wmax=0.6;
   wmin=0.6;
   w = wmax-i*((wmax-wmin)/it);

   cg=cgmin-i*((cgmin-cgmax)/it);
   cp=cpmax-i*((cpmax-cpmin)/it);
   disp(sprintf('cp= %d ',cp))
   disp(sprintf('cg= %d ',cg))

   for j=1:p
```

```
        v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g(j)-x1(j,i));
        v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-x2(j,i));
        x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
        x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
        %f(j,(i+1))= 100*( (x1(j,(i+1)))^2 - x2(j,(i+1)) )^2 + (1- x1(j,(i+1)))^2 ;
        f(j,(i+1))=(1.5-x1(j,(i+1))+x1(j,(i+1))*x2(j,(i+1)))^2+(2.25-
x1(j,(i+1))+x1(j,(i+1))*x2(j,(i+1))^2)^2+(2.625-x1(j,(i+1))+x1(j,(i+1))*x2(j,(i+1))^3)^2
        %f(j,(i+1))=(x1(j,(i+1))+2*(x2(j,(i+1)))-7)^2+(2*(x1(j,(i+1)))+x2(j,(i+1))-5)^2;


    end

%To find change in the values of f
    for j=1:p
        df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
    end

%personal best values updation
    for j=1:p
        %fp(j)= 100*( (x1p(j))^2 - x2p(j) )^2 + (1- x1p(j))^2 ;
        f(j)=(1.5-x1p(j)+x1p(j)*x2p(j))^2+(2.25-x1p(j)+x1p(j)*x2p(j)^2)^2+(2.625-x1p(j)+x1p(j)*x2p(j)^3)^2;
        %f(j)=(x1p(j)+2*(x2p(j))-7)^2+(2*(x1p(j))+x2p(j)-5)^2;

    end
    for k=1:p
        if f(k,i)< fp(k)
            x1p(k)=x1(k,i);
            x2p(k)=x2(k,i);
        else
        end
    end

%for Global best values updation
    if min(f(:,(i+1)))<fgm
        fgm=min(f(:,(i+1)));
    else
    end

    for j=1:i
        for k=1:p
        if f(k,i)==fgm
            for l=1:p
                x1g(l) = x1(k,i);     %global best values
                x2g(l) = x2(k,i);
                %plot(x1g,x2g);
            end
        else
        end
        end
    end

    print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
    disp('    x1       x2       v1       v2       f')
    disp(print)

%     fig(i) = figure('Position', [100 100 500 350]);
%     t(i) = uitable('Parent', fig(i), 'Position', [25 25 450 200]);
```

```
%     print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
%     set(t(i), 'Data', print);
%     set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});

%Stoping criterion
   ki=0;
   for j=1:p
      if (df(j,i)<=10^(-T))
         ki=ki+1;
      end
   end
   if ki >= p
      break
   end

end
[r,c]=find(f==fgm);
minf=100*( (x1g(j))^2 - x2g(1) )^2 + (1- x1g(1))^2;
%disp(sprintf('min value of function is %d and at values of x1=%d and x2=%d ',minf,x1g(1),x2g(1)))
disp(sprintf('          R  E  S  U  L  T  S %d'))
disp(sprintf('       x1=%d', x1g(1)));
disp(sprintf('       x2=%d', x2g(1)));
disp(sprintf('       F=%d', minf(1)));
disp(sprintf('       it=%d', i));
z=1:1:200;
for z=1:50
   plot(z,f(1,z),'o');
pause(0.2)
end
```

Code for the Optimization of Sphere Function using SPSO

```
clc
disp(' we have to minimize f = 100(x1^2-x2)^2+(1-x1)^2 i.e. rosenbrock function')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
x1=zeros(p,it); %no. of iterations are pre decided that is it will be maximum 50
x2=zeros(p,it);
x3=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
v3=zeros(p,it);
f=zeros(p,it);
x1g=zeros(p);
x2g=zeros(p);
x3g=zeros(p);
fp=zeros(1,p);
df=zeros(1,(it-1));
rp=0.4;
rg=0.5;
cp=2;
cg=2;
T=input('Enter the tolerance value');
% Initial values i.e. 0th iteration
% x1(:,1)=[1.5605 0.7795 0.4834 0.8078 0.1929 0.2639 1.8841 1.9123 1.1504 0.1196 0.4696 0.7063 1.6424
0.0308 0.0860 0.3380 1.2982 1.4634 1.2955 0.9018];
% x2(:,1)=[1.0940 0.5926 1.4894 0.3779 1.3736 0.3670 0.7370 1.2512 1.5605 0.1623 1.8588 1.5514 0.9736
0.8717 0.8936 0.6127 1.0170 1.0215 1.6353 1.5897];
```

```
% v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816 0.1564 0.8555 0.6448 0.3763 0.1909
0.4283 0.4820 0.1206 0.5895 0.2262 0.3846 0.5830];
% v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439 0.5841 0.1078 0.9063 0.8797 0.8178
0.2607 0.5944 0.0225 0.4253 0.3127 0.1615 0.1788];
x1(:,1)=unifrnd(-1,1,1,p);
x2(:,1)=unifrnd(-1,1,1,p);
x3(:,1)=unifrnd(-1,1,1,p);
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);
v3(:,1)=rand(1,p);
% i=0;
% disp (sprintf('enter the values of %dth iteration positions of %d particles for variable x1',i,p))
% for j=1:p
%     x1(j,1)=input(sprintf('enter the value of x1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable x2',p))
% for j=1:p
%     x2(j,1)=input(sprintf('enter the value of x2(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v1',p))
% for j=1:p
%     v1(j,1)=input(sprintf('enter the value of v1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v2',p))
% for j=1:p
%     v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
% end
for j=1:p
     f(j,1)= x1(j,1)^2 + x2(j,1)^2 + x3(j,1)^2;
end


%Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);
x3p=x2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for k=1:p
   if f(k,1)==fmin
      gb=k;
   else
   end
end
%Initial global best value
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
x3g(k) = x3(gb,1);
end
fgm = min(f(:,1));

% fig=zeros(1,485);
% t=zeros(1,485);


for i=1:it
```

```
    disp(sprintf('This is %d no. of iteration',i))

  %for inertia weight W
    wmax=1;
    wmin=0.3;
    w = wmax-i*((wmax-wmin)/it);

    for j=1:p
      v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g(j)-x1(j,i));
      v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-x2(j,i));
      v3(j,(i+1)) = w*v3(j,i) + rp*cp*(x3p(j)-x3(j,i)) + rg*cg*(x3(j)-x3(j,i));
      x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
      x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
      x3(j,(i+1)) = x3(j,i) + v3(j,(i+1));
      f(j,(i+1))= x1(j,(i+1))^2 + x2(j,(i+1))^2 + x3(j,(i+1))^2;
    end

  %To find change in the values of f
    for j=1:p
      df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
    end

  %personal best values updation
    for j=1:p
    fp(j)= x1p(j)^2 + x2p(j)^2;
    end
    for k=1:p
      if f(k,i)< fp(k)
        x1p(k)=x1(k,i);
        x2p(k)=x2(k,i);
        x3p(k)=x3(k,i);
      else
      end
    end

  %for Global best values updation
    if min(f(:,(i+1)))<fgm
      fgm=min(f(:,(i+1)));
    else
    end

    for j=1:i
      for k=1:p
      if f(k,i)==fgm
        for l=1:p
          x1g(l) = x1(k,i);     %global best values
          x2g(l) = x2(k,i);
          x3g(l) = x3(k,i);
        end
      else
      end
      end
    end

    print = [x1(:,i)     x2(:,i)     x3(:,i)     v1(:,i)     v2(:,i)     f(:,i)];
    disp('     x1          x2        x3        v1        v2         f')
    disp(print)
```

```
%     fig(i) = figure('Position', [100 100 500 350]);
%     t(i) = uitable('Parent', fig(i), 'Position', [25 25 450 200]);
%     print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
%     set(t(i), 'Data', print);
%     set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});

%Stoping criterion
   ki=0;
   for j=1:p
      if (df(j,i)<=10^(-T))
         ki=ki+1;
      end
   end
   if ki >= p
      break
   end

end
[r,c]=find(f==fgm);
disp(sprintf('min value of function is %d and at values of x1=%d, x2=%d and x3=%d
',fgm,x1g(1),x2g(1),x3g(1)))
```

Code for the Optimization of Sphere Function using IPSO

```
clc
disp(' we have to minimize f = 100(x1^2-x2)^2+(1-x1)^2 i.e. rosenbrock function')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
x1=zeros(p,it); %no. of iterations are pre decided that is it will be maximum 50
x2=zeros(p,it);
x3=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
v3=zeros(p,it);
f=zeros(p,it);
x1g=zeros(p);
x2g=zeros(p);
x3g=zeros(p);
fp=zeros(1,p);
df=zeros(1,(it-1));
rp=0.4;
rg=0.5;
cp=2;
cg=2;
T=input('Enter the tolerance value');
% Initial values i.e. 0th iteration
% x1(:,1)=[1.5605 0.7795 0.4834 0.8078 0.1929 0.2639 1.8841 1.9123 1.1504 0.1196 0.4696 0.7063 1.6424
0.0308 0.0860 0.3380 1.2982 1.4634 1.2955 0.9018];
% x2(:,1)=[1.0940 0.5926 1.4894 0.3779 1.3736 0.3670 0.7370 1.2512 1.5605 0.1623 1.8588 1.5514 0.9736
0.8717 0.8936 0.6127 1.0170 1.0215 1.6353 1.5897];
% v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816 0.1564 0.8555 0.6448 0.3763 0.1909
0.4283 0.4820 0.1206 0.5895 0.2262 0.3846 0.5830];
% v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439 0.5841 0.1078 0.9063 0.8797 0.8178
0.2607 0.5944 0.0225 0.4253 0.3127 0.1615 0.1788];
x1(:,1)=unifrnd(-1,1,1,p);
x2(:,1)=unifrnd(-1,1,1,p);
x3(:,1)=unifrnd(-1,1,1,p);
```

```
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);
v3(:,1)=rand(1,p);
% i=0;
% disp (sprintf('enter the values of %dth iteration positions of %d particles for variable x1',i,p))
% for j=1:p
%     x1(j,1)=input(sprintf('enter the value of x1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable x2',p))
% for j=1:p
%     x2(j,1)=input(sprintf('enter the value of x2(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v1',p))
% for j=1:p
%     v1(j,1)=input(sprintf('enter the value of v1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v2',p))
% for j=1:p
%     v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
% end
for j=1:p
     f(j,1)= x1(j,1)^2 + x2(j,1)^2 + x3(j,1)^2;
end


%Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);
x3p=x2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for k=1:p
   if f(k,1)==fmin
      gb=k;
   else
   end
end
%Initial global best value
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
x3g(k) = x3(gb,1);
end
fgm = min(f(:,1));

% fig=zeros(1,485);
% t=zeros(1,485);


for i=1:it
   disp(sprintf('This is %d no. of iteration',i))

%for inertia weight W
   wmax=0.9;
   wmin=0.4;
   w = wmax-i*((wmax-wmin)/it);
   %cg=cgmin-i*((cgmin-cgmax)/it);
```

```
    %cp=cpmax-i*((cpmax-cpmin)/it);

    for j=1:p
       v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g(j)-x1(j,i));
       v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-x2(j,i));
       v3(j,(i+1)) = w*v3(j,i) + rp*cp*(x3p(j)-x3(j,i)) + rg*cg*(x3(j)-x3(j,i));
       x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
       x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
       x3(j,(i+1)) = x3(j,i) + v3(j,(i+1));
       f(j,(i+1))= x1(j,(i+1))^2 + x2(j,(i+1))^2 + x3(j,(i+1))^2;
    end

%To find change in the values of f
    for j=1:p
       df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
    end

%personal best values updation
    for j=1:p
    fp(j)= x1p(j)^2 + x2p(j)^2;
    end
    for k=1:p
       if f(k,i)< fp(k)
          x1p(k)=x1(k,i);
          x2p(k)=x2(k,i);
          x3p(k)=x3(k,i);
       else
       end
    end

%for Global best values updation
    if min(f(:,(i+1)))<fgm
       fgm=min(f(:,(i+1)));
    else
    end

    for j=1:i
       for k=1:p
       if f(k,i)==fgm
          for l=1:p
             x1g(l) = x1(k,i);     %global best values
             x2g(l) = x2(k,i);
             x3g(l) = x3(k,i);
          end
       else
       end
       end
    end

    print = [x1(:,i)     x2(:,i)     x3(:,i)     v1(:,i)     v2(:,i)     f(:,i)];
    disp('     x1          x2        x3        v1        v2         f')
    disp(print)

%     fig(i) = figure('Position', [100 100 500 350]);
%     t(i) = uitable('Parent', fig(i), 'Position', [25 25 450 200]);
%     print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
%     set(t(i), 'Data', print);
%     set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});
```

```
%Stoping criterion
   ki=0;
   for j=1:p
      if (df(j,i)<=10^(-T))
         ki=ki+1;
      end
   end
   if ki >= p
      break
   end

end
[r,c]=find(f==fgm);
disp(sprintf('min value of function is %d and at values of x1=%d, x2=%d and x3=%d
',fgm,x1g(1),x2g(1),x3g(1)))
```

Code for the Optimization of Booth's Function using SPSO

```
clc
disp(' we have to minimize f = (x1+2*x2-7)^2+(2*x1+x2-5)^2 i.e. b00TH function')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
x1=zeros(p,it); %no. of iterations are pre decided that is it will be maximum 50
x2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
fp=zeros(1,p);
df=zeros(1,(it-1));
rp=0.4;
rg=0.5;
cp=2;
cg=2;
T=input('Enter the tolerance value');
% Initial values i.e. 0th iteration
% x1(:,1)=[1.5605 0.7795 0.4834 0.8078 0.1929 0.2639 1.8841 1.9123 1.1504 0.1196 0.4696 0.7063 1.6424
0.0308 0.0860 0.3380 1.2982 1.4634 1.2955 0.9018];
% x2(:,1)=[1.0940 0.5926 1.4894 0.3779 1.3736 0.3670 0.7370 1.2512 1.5605 0.1623 1.8588 1.5514 0.9736
0.8717 0.8936 0.6127 1.0170 1.0215 1.6353 1.5897];
% v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816 0.1564 0.8555 0.6448 0.3763 0.1909
0.4283 0.4820 0.1206 0.5895 0.2262 0.3846 0.5830];
% v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439 0.5841 0.1078 0.9063 0.8797 0.8178
0.2607 0.5944 0.0225 0.4253 0.3127 0.1615 0.1788];
x1(:,1)=unifrnd(0,4.5,1,p);
x2(:,1)=unifrnd(0,4.5,1,p);
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);
% i=0;
% disp (sprintf('enter the values of %dth iteration positions of %d particles for variable x1',i,p))
% for j=1:p
%     x1(j,1)=input(sprintf('enter the value of x1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable x2',p))
% for j=1:p
%     x2(j,1)=input(sprintf('enter the value of x2(%d,%d)',j,i));
% end
```

```
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v1',p))
% for j=1:p
%     v1(j,1)=input(sprintf('enter the value of v1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v2',p))
% for j=1:p
%     v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
% end
for j=1:p
   f(j,1)=(x1(j,1)+2*(x2(j,1))-7)^2+(2*(x1(j,1))+x2(j,1)-5)^2;
end


%Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for k=1:p
   if f(k,1)==fmin
      gb=k;
   else
   end
end
%Initial global best value
x1g=zeros(p);
x2g=zeros(p);
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
end
fgm = min(f(:,1));

% fig=zeros(1,485);
% t=zeros(1,485);


for i=1:it
   disp(sprintf('This is %d no. of iteration',i))

%for inertia weight W
   wmax=0.9;
   wmin=0.4;
   w = wmax-i*((wmax-wmin)/it);

   for j=1:p
      v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g(j)-x1(j,i));
      v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-x2(j,i));
      x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
      x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
       f(j,(i+1))=(x1(j,(i+1))+2*(x2(j,(i+1)))-7)^2+(2*(x1(j,(i+1)))+x2(j,(i+1))-5)^2;
   end

%To find change in the values of f
   for j=1:p
      df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
```

```
    end

%personal best values updation
    for j=1:p
        f(j)=(x1p(j)+2*(x2p(j))-7)^2+(2*(x1p(j))+x2p(j)-5)^2;
    end
    for k=1:p
        if f(k,i)< fp(k)
            x1p(k)=x1(k,i);
            x2p(k)=x2(k,i);
        else
        end
    end

%for Global best values updation
    if min(f(:,(i+1)))<fgm
        fgm=min(f(:,(i+1)));
    else
    end

    for j=1:i
        for k=1:p
        if f(k,i)==fgm
            for l=1:p
                x1g(l) = x1(k,i);     %global best values
                x2g(l) = x2(k,i);
            end
        else
        end
        end
    end

    print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
    disp('    x1       x2       v1       v2       f')
    disp(print)

%     fig(i) = figure('Position', [100 100 500 350]);
%     t(i) = uitable('Parent', fig(i), 'Position', [25 25 450 200]);
%     print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
%     set(t(i), 'Data', print);
%     set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});

%Stoping criterion
    ki=0;
    for j=1:p
        if (df(j,i)<=10^(-T))
            ki=ki+1;
        end
    end
    if ki >= p
        break
    end

end
[r,c]=find(f==fgm);
disp(sprintf('min value of function is %d and at values of x1=%d and x2=%d ',fgm,x1(r,c),x2(r,c)))
```

Code for the Optimization of Booth's Function using IPSO

```
clc
disp(' we have to minimize f = 100(x1^2-x2)^2+(1-x1)^2 i.e. rosenbrock function')
%p=input('Enter the no. of particles in a swarm');          %no. of particles
p=30;
%it=input('Enter the no. of iterations');
it=5000;
x1=zeros(p,it);
x2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
fp=zeros(1,p);
df=zeros(1,(it-1));
rp=1;
rg=1;
%cp=2;
%cg=2;
cpmax=input('cpmax= ');
cpmin=input('cpmin= ');
cgmax=input('cgmax= ');
cgmin=input('cgmin= ');
T=input('Enter the tolerance value');
% Initial values i.e. 0th iteration
% x1(:,1)=[1.2886 0.7572 1.6232 1.0657 0.7015 1.8780 1.7519 1.1003 1.2450 1.1741];
% x2(:,1)=[0.4155 0.6025 0.9418 0.4610 1.6886 0.3895 0.4518 0.3414 0.4553 0.8714];
% v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816 0.1564 0.8555 0.6448 0.3763 0.1909
0.4283 0.4820 0.1206 0.5895 0.2262 0.3846 0.5830];
% v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439 0.5841 0.1078 0.9063 0.8797 0.8178
0.2607 0.5944 0.0225 0.4253 0.3127 0.1615 0.1788];
x1(:,1)=unifrnd(0.5,4.5,1,p);
x2(:,1)=unifrnd(0.5,4.5,1,p);
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);
% i=0;
% disp (sprintf('enter the values of %dth iteration positions of %d particles for variable x1',i,p))
% for j=1:p
%     x1(j,1)=input(sprintf('enter the value of x1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable x2',p))
% for j=1:p
%     x2(j,1)=input(sprintf('enter the value of x2(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v1',p))
% for j=1:p
%     v1(j,1)=input(sprintf('enter the value of v1(%d,%d)',j,i));
% end
% disp (sprintf('enter the values of 0th iteration positions of %d particles for variable v2',p))
% for j=1:p
%     v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
% end
for j=1:p
   %f(j,1)= 100*( (x1(j,1))^2 - x2(j,1) )^2 + (1- x1(j,1))^2 ;
   %f(j,1)=(1.5-x1(j,1)+x1(j,1)*x2(j,1))^2+(2.25-x1(j,1)+x1(j,1)*x2(j,1)^2)^2+(2.625-
x1(j,1)+x1(j,1)*x2(j,1)^3)^2;
   f(j,1)=(x1(j,1)+2*(x2(j,1))-7)^2+(2*(x1(j,1))+x2(j,1)-5)^2;

end
```

```
 %Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for k=1:p
   if f(k,1)==fmin
      gb=k;
   else
   end
end
%Initial global best value
x1g=zeros(p);
x2g=zeros(p);
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
end
fgm = min(f(:,1));

% fig=zeros(1,485);
% t=zeros(1,485);


for i=1:it
   disp(sprintf('This is %d no. of iteration',i))

%for inertia weight W
   wmax=0.6;
   wmin=0.6;
   w = wmax-i*((wmax-wmin)/it);

   cg=cgmin-i*((cgmin-cgmax)/it);
   cp=cpmax-i*((cpmax-cpmin)/it);
   disp(sprintf('cp= %d ',cp))
   disp(sprintf('cg= %d ',cg))

   for j=1:p
      v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g(j)-x1(j,i));
      v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-x2(j,i));
      x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
      x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
      %f(j,(i+1))= 100*( (x1(j,(i+1)))^2 - x2(j,(i+1)) )^2 + (1- x1(j,(i+1)))^2 ;
      %f(j,(i+1))=(1.5-x1(j,(i+1))+x1(j,(i+1))*x2(j,(i+1)))^2+(2.25-
x1(j,(i+1))+x1(j,(i+1))*x2(j,(i+1))^2)^2+(2.625-x1(j,(i+1))+x1(j,(i+1))*x2(j,(i+1))^3)^2
      f(j,(i+1))=(x1(j,(i+1))+2*(x2(j,(i+1)))-7)^2+(2*(x1(j,(i+1)))+x2(j,(i+1))-5)^2;


   end

%To find change in the values of f
   for j=1:p
      df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
   end

%personal best values updation
```

```
    for j=1:p
       %fp(j)= 100*( (x1p(j))^2 - x2p(j) )^2 + (1- x1p(j))^2 ;
       %f(j)=(1.5-x1p(j)+x1p(j)*x2p(j))^2+(2.25-x1p(j)+x1p(j)*x2p(j)^2)^2+(2.625-x1p(j)+x1p(j)*x2p(j)^3)^2;
       f(j)=(x1p(j)+2*(x2p(j))-7)^2+(2*(x1p(j))+x2p(j)-5)^2;

    end
    for k=1:p
       if f(k,i)< fp(k)
          x1p(k)=x1(k,i);
          x2p(k)=x2(k,i);
       else
       end
    end

%for Global best values updation
    if min(f(:,(i+1)))<fgm
       fgm=min(f(:,(i+1)));
    else
    end

    for j=1:i
       for k=1:p
       if f(k,i)==fgm
          for l=1:p
             x1g(l) = x1(k,i);     %global best values
             x2g(l) = x2(k,i);
             %plot(x1g,x2g);
          end
       else
       end
       end
    end

    print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
    disp('     x1        x2        v1        v2        f')
    disp(print)

%     fig(i) = figure('Position', [100 100 500 350]);
%     t(i) = uitable('Parent', fig(i), 'Position', [25 25 450 200]);
%     print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
%     set(t(i), 'Data', print);
%     set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});

%Stoping criterion
    ki=0;
    for j=1:p
       if (df(j,i)<=10^(-T))
          ki=ki+1;
       end
    end
    if ki >= p
       break
    end

end
[r,c]=find(f==fgm);
minf=100*( (x1g(j))^2 - x2g(1) )^2 + (1- x1g(1))^2;
%disp(sprintf('min value of function is %d and at values of x1=%d and x2=%d ',minf,x1g(1),x2g(1)))
```

```
disp(sprintf('          R  E  S  U  L  T  S %d'))
disp(sprintf('       x1=%d', x1g(1)));
disp(sprintf('       x2=%d', x2g(1)));
disp(sprintf('       F=%d', minf(1)));
disp(sprintf('       it=%d', i));
z=1:1:200;
for z=1:50
   plot(z,f(1,z),'o');
pause(0.2)
end
```

(ii) MATLAB Programs for ELD in IEEE 5,14 and 30 bus systems using PSO

Code for the ELD in IEEE 5 bus system using SPSO

```
clear all
clc
disp(' we have to minimize the cost function of a 3 machine system')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
a=10^(-4)*[50 50];
b=10^(-2)*[351 389];
c=[44.4 40.6];
B=10^(-2)*[0.0349 0.0086; -0.0055 0.0371];
p1=zeros(p,it); %no. of iterations are pre decided that is it will be maximum 50
p2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
df=zeros(p,it);
sp=zeros(p,it);
csp=zeros(p,it);
pl=zeros(p,it);
c1=zeros(p,it);
c2=zeros(p,it);
C=zeros(p,it);
rp=0.4;
rg=0.5;
cp=2;
cg=2;
%cpmax=input('cpmax= ');
%cpmin=input('cpmin= ');
%cgmax=input('cgmax= ');
%cgmin=input('cgmin= ');
pd=160;
   p1g=zeros(p);
   p2g=zeros(p);
   fp=zeros(1,p);
   plp=zeros(1,p);

k=50;
```

```
w1=1;
w2=0;
% Initial values i.e. 0th iteration
% p1(:,1)=[0.2356 0.5478 1.2453 1.5897 ];
% p2(:,1)=[1.1254 1.3658 1.9875 1.5632 ];
n=1;
while n==1
   for j=1:p
   p1(j,1)=unifrnd(30,120,1);
   p2(j,1)=pd-p1(j,1);
   if p2(j,1)<30&&p3(j,1)>120
      n=1;
      break;
   else
      n=0;
   end
   end
end
% v1(:,1)=[-0.2 -0.1 -0.2 -0.2 -0.1 -0.1 -0.2 -0.1 -0.3 -0.2];
% v2(:,1)=[-0.2 -0.1 -0.3 -0.1 -0.2 -0.1 -0.3 -0.1 -0.2 -0.1];
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);

%Total cost calculation
   for j=1:p
      c1(j,1) = a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1);
      c2(j,1) = a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2);
      C(j,1) = c1(j,1) + c2(j,1);
   end

%To calculate initial value of cost function we need PL
for j=1:p
   pl(j,1)= [p1(j,1) p2(j,1)]*B*[p1(j,1) p2(j,1)]';
end

%To calculate initial value of cost function
for j=1:p
   f(j,1)= w1*((a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1)) + (a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2)))...
         + w2*pl(j,1) + k*abs(pd+pl(j,1)-p1(j,1)-p2(j,1));
end
%0th iteration data display
   disp('this is the 0th iteration')
   print0 = [p1(:,1) p2(:,1) v1(:,1) v2(:,1) f(:,1) c1(:,1) c2(:,1) C(:,1)];
   disp('     P1        P2        V1        v2       f        c1        c2        C ')
   disp(print0)

%Initial personal besst values
p1p=p1(:,1);
p2p=p2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for m=1:p
   if f(m,1)==fmin
      gb=m;
   else
   end
end
```

```
%Initial global best value
for m=1:p
p1g(m) = p1(gb,1);
p2g(m) = p2(gb,1);
end
fgm = min(f(:,1));

%Main iterations starts from here
for i=1:it
   disp(sprintf('This is iteration no.= %d',i))

%for inertia weight W
   wmax=0.9;
   wmin=0.4;
   w = wmax-i*((wmax-wmin)/it);
   %cg=cgmax-i*((cgmax-cgmin)/it);
   %cp=cpmin-i*((cpmin-cpmax)/it);
   %disp(sprintf('cp= %d ',cp))
   %disp(sprintf('cg= %d ',cg))

   %For calculatiing velocities for updation
   for j=1:p
      v1(j,(i+1)) = w*v1(j,i) + rp*cp*(p1p(j)-p1(j,i)) + rg*cg*(p1g(j)-p1(j,i));
      v2(j,(i+1)) = w*v2(j,i) + rp*cp*(p2p(j)-p2(j,i)) + rg*cg*(p2g(j)-p2(j,i));
   end
   %V(min) and V(max) constraint
   for j=1:p
      if v1(j,(i+1))< -15
         v1(j,(i+1))= -15;
      end
      if v2(j,(i+1))< -15
         v2(j,(i+1))= -15;
      end
      if v1(j,(i+1))> 60
         v1(j,(i+1))= 60;
      end
      if v2(j,(i+1))> 60
         v2(j,(i+1))= 60;
      end
   end

%Updation of p values
   for j=1:p
      p1(j,(i+1)) = p1(j,i) + v1(j,(i+1));
      p2(j,(i+1)) = p2(j,i) + v2(j,(i+1));
   end
   %Pmin and Pmax constraint
   for j=1:p
      if p1(j,(i+1))< 30
         p1(j,(i+1))= 30;
      end
      if p2(j,(i+1))< 30
         p2(j,(i+1))= 30;
      end
      if p1(j,(i+1))> 120
         p1(j,(i+1))= 120;
      end
      if p2(j,(i+1))> 120
```

```
        p2(j,(i+1))= 120;
      end
    end

    %For losses formulation (PL)
    for j=1:p
      pl(j,(i+1))= [p1(j,(i+1)) p2(j,(i+1))]*B*[p1(j,(i+1)) p2(j,(i+1))]';
    end

  %Main objective function
    for j=1:p
      f(j,(i+1))= w1*((a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1)) +...
                  (a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2)))+...
                  w2*pl(j,(i+1)) + k*abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
    end

  %personal best values updation
    %For losses formulation (PL)
    for j=1:p
      plp(j)= [p1p(j) p2p(j)]*B*[p1p(j) p2p(j)]';
    end

    for j=1:p
      fp(j)= w1*((a(1)*(p1p(j))^2 + b(1)*p1p(j) + c(1)) + (a(2)*(p2p(j))^2 + b(2)*p2p(j) + c(2)))...
           + w2*plp(j) +  k*abs(pd+plp(j)-p1p(j)-p2p(j));
    end
    for m=1:p
      if f(m,i)< fp(m)
        p1p(m)=p1(m,(i+1));
        p2p(m)=p2(m,(i+1));

      else
      end
    end

  %for Global best values updation
    if min(f(:,(i+1)))<fgm
      fgm=min(f(:,(i+1)));
    else
    end

    for j=1:(i+1)
      for m=1:p
        if f(m,j)==fgm
          for l=1:p
            p1g(l) = p1(m,j);     %global best values
            p2g(l) = p2(m,j);
          end
        else
        end
      end
    end

    %For Cost Calculation
    for j=1:p
      c1(j,(i+1)) = a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1);
      c2(j,(i+1)) = a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2);
      C(j,(i+1)) = c1(j,(i+1)) + c2(j,(i+1));
```

```
    end

    %To find change in the values of f
    for j=1:p
       df(j,i)= abs(f(j,(i+1))-f(j,i));
       sp(j,i)= abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
       csp(j,i)= abs(C(j,(i+1))-C(j,i));
    end

    print = [p1(:,(i+1)) p2(:,(i+1)) v1(:,(i+1)) v2(:,(i+1)) f(:,(i+1)) c1(:,(i+1)) c2(:,(i+1))  C(:,(i+1))];
    disp('     P1        P2        V1        v2        f        c1        c2        C ')
    disp(print)


%Stoping criterion (df(j,i)<=10^(-6))&& &&(csp(j,i)<=10^(-6))
    ki=0;
    for j=1:p
       if ((df(j,i)<=10^(-6))&&(sp(j,i)<=10^(-6)))
          ki=ki+1;
       end
    end
    if ki >= p
       break
    end

end

disp(' we have to minimize the cost function of a 2 machine 5 bus system')
disp(sprintf('No. of particles used in a swarm = %d',p))
disp(sprintf('Max. no. of iterations entered = %d\n',it))

disp(sprintf('Total demand of power Pd = %d \n',pd))

disp('Initial values of generations of 2 generators')
initial=[p1(:,1) p2(:,1) ];
disp('    P1        P2    ')
disp(initial)

disp(sprintf('\nPD+Pl = %d',pd+pl(1,i)))
disp(sprintf('\nP1+P2=%d\n',p1(1,i)+p2(1,i)))

disp(sprintf('No. of total Iterations took place = %d \n',i))
disp(sprintf('Total loses in the lines Pl = %d \n',pl(1,i)))
disp(sprintf('Minimum cost incured = %d \n',C(1,i)))
disp('\nFinal values of generations of the three generators')
disp(sprintf('P1=%d',p1(1,i)))
disp(sprintf('P2=%d',p2(1,i)))

disp('About this run')
disp('1. The Constraints has been included as absolute value.')
disp('2. Random values between the limits of generation have been taken for each generator as the different
starting point.')
disp('3. Correct values of B coefficients have been fed.')
disp(sprintf('4. K taken = %d',k))
disp(sprintf('5. w1 and w2 taken = %d and %d',w1,w2))
%disp(sprintf('           R  E  S  U  L  T  S %d'))
%disp(sprintf('       x1=%d', x1g(1)));
```

```
%disp(sprintf('       x2=%d', x2g(1)));
%disp(sprintf('       F=%d', minf(1)));
%disp(sprintf('       it=%d', i));
```

Code for the ELD in IEEE 5 bus system using IPSO 1

```
clear all
clc
disp(' we have to minimize the cost function of a 3 machine system')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
a=10^(-4)*[50 50];
b=10^(-2)*[351 389];
c=[44.4 40.6];
B=10^(-2)*[0.0349 0.0086; -0.0055 0.0371];
p1=zeros(p,it); %no. of iterations are pre decided that is it will be maximum 50
p2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
df=zeros(p,it);
sp=zeros(p,it);
csp=zeros(p,it);
pl=zeros(p,it);
c1=zeros(p,it);
c2=zeros(p,it);
C=zeros(p,it);
rp=0.4;
rg=0.5;
cp=2;
cg=2;
%cpmax=input('cpmax= ');
%cpmin=input('cpmin= ');
%cgmax=input('cgmax= ');
%cgmin=input('cgmin= ');
pd=160;
   p1g=zeros(p);
   p2g=zeros(p);
   fp=zeros(1,p);
   plp=zeros(1,p);

k=50;
w1=1;
w2=0;
% Initial values i.e. 0th iteration
% p1(:,1)=[0.2356 0.5478 1.2453 1.5897 ];
% p2(:,1)=[1.1254 1.3658 1.9875 1.5632 ];
n=1;
while n==1
   for j=1:p
   p1(j,1)=unifrnd(30,120,1);
   p2(j,1)=pd-p1(j,1);
   if p2(j,1)<30&&p3(j,1)>120
      n=1;
      break;
   else
      n=0;
   end
   end
```

```
end
% v1(:,1)=[-0.2 -0.1 -0.2 -0.2 -0.1 -0.1 -0.2 -0.1 -0.3 -0.2];
% v2(:,1)=[-0.2 -0.1 -0.3 -0.1 -0.2 -0.1 -0.3 -0.1 -0.2 -0.1];
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);

%Total cost calculation
   for j=1:p
       c1(j,1) = a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1);
       c2(j,1) = a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2);
       C(j,1) = c1(j,1) + c2(j,1);
   end

%To calculate initial value of cost function we need PL
for j=1:p
   pl(j,1)= [p1(j,1) p2(j,1)]*B*[p1(j,1) p2(j,1)]';
end

%To calculate initial value of cost function
for j=1:p
   f(j,1)= w1*((a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1)) + (a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2)))...
         + w2*pl(j,1) + k*abs(pd+pl(j,1)-p1(j,1)-p2(j,1));
end
%0th iteration data display
   disp('this is the 0th iteration')
   print0 = [p1(:,1) p2(:,1) v1(:,1) v2(:,1) f(:,1) c1(:,1) c2(:,1) C(:,1)];
   disp('     P1        P2        V1        v2      f       c1       c2       C ')
   disp(print0)

%Initial personal besst values
p1p=p1(:,1);
p2p=p2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for m=1:p
   if f(m,1)==fmin
      gb=m;
   else
   end
end
%Initial global best value
for m=1:p
p1g(m) = p1(gb,1);
p2g(m) = p2(gb,1);
end
fgm = min(f(:,1));

%Main iterations starts from here
for i=1:it
   disp(sprintf('This is iteration no.= %d',i))

%for inertia weight W
   wmax=0.6;
   wmin=0.6;
   w = wmax-i*((wmax-wmin)/it);
   %cg=cgmax-i*((cgmax-cgmin)/it);
   %cp=cpmin-i*((cpmin-cpmax)/it);
```

```
    %disp(sprintf('cp= %d ',cp))
    %disp(sprintf('cg= %d ',cg))

    %For calculatiing velocities for updation
    for j=1:p
       v1(j,(i+1)) = w*v1(j,i) + rp*cp*(p1p(j)-p1(j,i)) + rg*cg*(p1g(j)-p1(j,i));
       v2(j,(i+1)) = w*v2(j,i) + rp*cp*(p2p(j)-p2(j,i)) + rg*cg*(p2g(j)-p2(j,i));
    end
    %V(min) and V(max) constraint
    for j=1:p
       if v1(j,(i+1))< -15
          v1(j,(i+1))= -15;
       end
       if v2(j,(i+1))< -15
          v2(j,(i+1))= -15;
       end
       if v1(j,(i+1))> 60
          v1(j,(i+1))= 60;
       end
       if v2(j,(i+1))> 60
          v2(j,(i+1))= 60;
       end
    end

%Updation of p values
    for j=1:p
       p1(j,(i+1)) = p1(j,i) + v1(j,(i+1));
       p2(j,(i+1)) = p2(j,i) + v2(j,(i+1));
    end
    %Pmin and Pmax constraint
    for j=1:p
       if p1(j,(i+1))< 30
          p1(j,(i+1))= 30;
       end
       if p2(j,(i+1))< 30
          p2(j,(i+1))= 30;
       end
       if p1(j,(i+1))> 120
          p1(j,(i+1))= 120;
       end
       if p2(j,(i+1))> 120
          p2(j,(i+1))= 120;
       end
    end

    %For losses formulation (PL)
    for j=1:p
       pl(j,(i+1))= [p1(j,(i+1)) p2(j,(i+1))]*B*[p1(j,(i+1)) p2(j,(i+1))]';
    end

%Main objective function
   for j=1:p
       f(j,(i+1))= w1*((a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1)) +...
                   (a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2)))+...
                   w2*pl(j,(i+1)) + k*abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
    end

%personal best values updation
```

```
    %For losses formulation (PL)
    for j=1:p
       plp(j)= [p1p(j) p2p(j)]*B*[p1p(j) p2p(j)]';
    end

    for j=1:p
       fp(j)= w1*((a(1)*(p1p(j))^2 + b(1)*p1p(j) + c(1)) + (a(2)*(p2p(j))^2 + b(2)*p2p(j) + c(2)))...
            + w2*plp(j) +  k*abs(pd+plp(j)-p1p(j)-p2p(j));
    end
    for m=1:p
       if f(m,i)< fp(m)
          p1p(m)=p1(m,(i+1));
          p2p(m)=p2(m,(i+1));

       else
       end
    end

 %for Global best values updation
    if min(f(:,(i+1)))<fgm
       fgm=min(f(:,(i+1)));
    else
    end

    for j=1:(i+1)
       for m=1:p
          if f(m,j)==fgm
             for l=1:p
                p1g(l) = p1(m,j);     %global best values
                p2g(l) = p2(m,j);
             end
          else
          end
       end
    end

    %For Cost Calculation
    for j=1:p
        c1(j,(i+1)) = a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1);
        c2(j,(i+1)) = a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2);
        C(j,(i+1)) = c1(j,(i+1)) + c2(j,(i+1));
    end

    %To find change in the values of f
    for j=1:p
       df(j,i)= abs(f(j,(i+1))-f(j,i));
       sp(j,i)= abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
       csp(j,i)= abs(C(j,(i+1))-C(j,i));
    end

    print = [p1(:,(i+1)) p2(:,(i+1)) v1(:,(i+1)) v2(:,(i+1)) f(:,(i+1)) c1(:,(i+1)) c2(:,(i+1))  C(:,(i+1))];
    disp('     P1        P2        V1        v2        f        c1        c2        C ')
    disp(print)


 %Stoping criterion (df(j,i)<=10^(-6))&& &&(csp(j,i)<=10^(-6))
    ki=0;
```

```
    for j=1:p
       if ((df(j,i)<=10^(-6))&&(sp(j,i)<=10^(-6)))
          ki=ki+1;
       end
    end
    if ki >= p
       break
    end

end

disp(' we have to minimize the cost function of a 2 machine 5 bus system')
disp(sprintf('No. of particles used in a swarm = %d',p))
disp(sprintf('Max. no. of iterations entered = %d\n',it))

disp(sprintf('Total demand of power Pd = %d \n',pd))

disp('Initial values of generations of 2 generators')
initial=[p1(:,1) p2(:,1) ];
disp('    P1        P2    ')
disp(initial)

disp(sprintf('\nPD+Pl = %d',pd+pl(1,i)))
disp(sprintf('\nP1+P2=%d\n',p1(1,i)+p2(1,i)))

disp(sprintf('No. of total Iterations took place = %d \n',i))
disp(sprintf('Total loses in the lines Pl = %d \n',pl(1,i)))
disp(sprintf('Minimum cost incured = %d \n',C(1,i)))
disp('\nFinal values of generations of the three generators')
disp(sprintf('P1=%d',p1(1,i)))
disp(sprintf('P2=%d',p2(1,i)))

disp('About this run')
disp('1. The Constraints has been included as absolute value.')
disp('2. Random values between the limits of generation have been taken for each generator as the different
starting point.')
disp('3. Correct values of B coefficients have been fed.')
disp(sprintf('4. K taken = %d',k))
disp(sprintf('5. w1 and w2 taken = %d and %d',w1,w2))
%disp(sprintf('          R  E  S  U  L  T  S %d'))
%disp(sprintf('       x1=%d', x1g(1)));
%disp(sprintf('       x2=%d', x2g(1)));
%disp(sprintf('       F=%d', minf(1)));
%disp(sprintf('       it=%d', i));
```

Code for the ELD in IEEE 5 bus system using IPSO 2

```
clear all
clc
disp(' we have to minimize the cost function of a 3 machine system')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
a=10^(-4)*[50 50];
b=10^(-2)*[351 389];
c=[44.4 40.6];
B=10^(-2)*[0.0349 0.0086; -0.0055 0.0371];
p1=zeros(p,it); %no. of iterations are pre decided that is it will be maximum 50
```

```
p2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
df=zeros(p,it);
sp=zeros(p,it);
csp=zeros(p,it);
pl=zeros(p,it);
c1=zeros(p,it);
c2=zeros(p,it);
C=zeros(p,it);
rp=1;
rg=1;
cp=1;
cg=1;
%cpmax=input('cpmax= ');
%cpmin=input('cpmin= ');
%cgmax=input('cgmax= ');
%cgmin=input('cgmin= ');
pd=160;
   p1g=zeros(p);
   p2g=zeros(p);
   fp=zeros(1,p);
   plp=zeros(1,p);

k=50;
w1=1;
w2=0;
% Initial values i.e. 0th iteration
% p1(:,1)=[0.2356 0.5478 1.2453 1.5897 ];
% p2(:,1)=[1.1254 1.3658 1.9875 1.5632 ];
n=1;
while n==1
   for j=1:p
   p1(j,1)=unifrnd(30,120,1);
   p2(j,1)=pd-p1(j,1);
   if p2(j,1)<30&&p3(j,1)>120
      n=1;
      break;
   else
      n=0;
   end
   end
end
% v1(:,1)=[-0.2 -0.1 -0.2 -0.2 -0.1 -0.1 -0.2 -0.1 -0.3 -0.2];
% v2(:,1)=[-0.2 -0.1 -0.3 -0.1 -0.2 -0.1 -0.3 -0.1 -0.2 -0.1];
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);

%Total cost calculation
   for j=1:p
      c1(j,1) = a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1);
      c2(j,1) = a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2);
      C(j,1) = c1(j,1) + c2(j,1);
   end

%To calculate initial value of cost function we need PL
for j=1:p
   pl(j,1)= [p1(j,1) p2(j,1)]*B*[p1(j,1) p2(j,1)]';
```

```
end

%To calculate initial value of cost function
for j=1:p
   f(j,1)= w1*((a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1)) + (a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2)))...
         + w2*pl(j,1) + k*abs(pd+pl(j,1)-p1(j,1)-p2(j,1));
end
%0th iteration data display
   disp('this is the 0th iteration')
   print0 = [p1(:,1) p2(:,1) v1(:,1) v2(:,1) f(:,1) c1(:,1) c2(:,1) C(:,1)];
   disp('     P1        P2        V1        v2      f       c1        c2        C ')
   disp(print0)

%Initial personal besst values
p1p=p1(:,1);
p2p=p2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for m=1:p
   if f(m,1)==fmin
      gb=m;
   else
   end
end
%Initial global best value
for m=1:p
p1g(m) = p1(gb,1);
p2g(m) = p2(gb,1);
end
fgm = min(f(:,1));

%Main iterations starts from here
for i=1:it
   disp(sprintf('This is iteration no.= %d',i))

%for inertia weight W
   wmax=0.6;
   wmin=0.6;
   w = wmax-i*((wmax-wmin)/it);
   %cg=cgmax-i*((cgmax-cgmin)/it);
   %cp=cpmin-i*((cpmin-cpmax)/it);
   %disp(sprintf('cp= %d ',cp))
   %disp(sprintf('cg= %d ',cg))

   %For calculatiing velocities for updation
   for j=1:p
      v1(j,(i+1)) = w*v1(j,i) + rp*cp*(p1p(j)-p1(j,i)) + rg*cg*(p1g(j)-p1(j,i));
      v2(j,(i+1)) = w*v2(j,i) + rp*cp*(p2p(j)-p2(j,i)) + rg*cg*(p2g(j)-p2(j,i));
   end
   %V(min) and V(max) constraint
   for j=1:p
      if v1(j,(i+1))< -15
         v1(j,(i+1))= -15;
      end
      if v2(j,(i+1))< -15
         v2(j,(i+1))= -15;
      end
```

```
        if v1(j,(i+1))> 60
            v1(j,(i+1))= 60;
        end
        if v2(j,(i+1))> 60
            v2(j,(i+1))= 60;
        end
    end

%Updation of p values
    for j=1:p
        p1(j,(i+1)) = p1(j,i) + v1(j,(i+1));
        p2(j,(i+1)) = p2(j,i) + v2(j,(i+1));
    end
    %Pmin and Pmax constraint
    for j=1:p
        if p1(j,(i+1))< 30
            p1(j,(i+1))= 30;
        end
        if p2(j,(i+1))< 30
            p2(j,(i+1))= 30;
        end
        if p1(j,(i+1))> 120
            p1(j,(i+1))= 120;
        end
        if p2(j,(i+1))> 120
            p2(j,(i+1))= 120;
        end
    end

    %For losses formulation (PL)
    for j=1:p
        pl(j,(i+1))= [p1(j,(i+1)) p2(j,(i+1))]*B*[p1(j,(i+1)) p2(j,(i+1))]';
    end

%Main objective function
   for j=1:p
        f(j,(i+1))= w1*((a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1)) +...
                    (a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2)))+...
                    w2*pl(j,(i+1)) + k*abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
    end

%personal best values updation
    %For losses formulation (PL)
    for j=1:p
        plp(j)= [p1p(j) p2p(j)]*B*[p1p(j) p2p(j)]';
    end

    for j=1:p
        fp(j)= w1*((a(1)*(p1p(j))^2 + b(1)*p1p(j) + c(1)) + (a(2)*(p2p(j))^2 + b(2)*p2p(j) + c(2)))...
              + w2*plp(j) +  k*abs(pd+plp(j)-p1p(j)-p2p(j));
    end
    for m=1:p
        if f(m,i)< fp(m)
            p1p(m)=p1(m,(i+1));
            p2p(m)=p2(m,(i+1));

        else
        end
```

```
    end

%for Global best values updation
    if min(f(:,(i+1)))<fgm
        fgm=min(f(:,(i+1)));
    else
    end

    for j=1:(i+1)
        for m=1:p
            if f(m,j)==fgm
                for l=1:p
                    p1g(l) = p1(m,j);     %global best values
                    p2g(l) = p2(m,j);
                end
            else
            end
        end
    end

    %For Cost Calculation
    for j=1:p
        c1(j,(i+1)) = a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1);
        c2(j,(i+1)) = a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2);
        C(j,(i+1)) = c1(j,(i+1)) + c2(j,(i+1));
    end

    %To find change in the values of f
    for j=1:p
        df(j,i)= abs(f(j,(i+1))-f(j,i));
        sp(j,i)= abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
        csp(j,i)= abs(C(j,(i+1))-C(j,i));
    end

    print = [p1(:,(i+1)) p2(:,(i+1)) v1(:,(i+1)) v2(:,(i+1)) f(:,(i+1)) c1(:,(i+1)) c2(:,(i+1))  C(:,(i+1))];
    disp('     P1        P2        V1        v2        f        c1        c2        C ')
    disp(print)


%Stoping criterion (df(j,i)<=10^(-6))&& &&(csp(j,i)<=10^(-6))
    ki=0;
    for j=1:p
        if ((df(j,i)<=10^(-6))&&(sp(j,i)<=10^(-6)))
            ki=ki+1;
        end
    end
    if ki >= p
        break
    end

end

disp(' we have to minimize the cost function of a 2 machine 5 bus system')
disp(sprintf('No. of particles used in a swarm = %d',p))
disp(sprintf('Max. no. of iterations entered = %d\n',it))

disp(sprintf('Total demand of power Pd = %d \n',pd))
```

```
disp('Initial values of generations of 2 generators')
initial=[p1(:,1) p2(:,1) ];
disp('    P1        P2    ')
disp(initial)

disp(sprintf('\nPD+Pl = %d',pd+pl(1,i)))
disp(sprintf('\nP1+P2=%d\n',p1(1,i)+p2(1,i)))

disp(sprintf('No. of total Iterations took place = %d \n',i))
disp(sprintf('Total loses in the lines Pl = %d \n',pl(1,i)))
disp(sprintf('Minimum cost incured = %d \n',C(1,i)))
disp('\nFinal values of generations of the three generators')
disp(sprintf('P1=%d',p1(1,i)))
disp(sprintf('P2=%d',p2(1,i)))

disp('About this run')
disp('1. The Constraints has been included as absolute value.')
disp('2. Random values between the limits of generation have been taken for each generator as the different
starting point.')
disp('3. Correct values of B coefficients have been fed.')
disp(sprintf('4. K taken = %d',k))
disp(sprintf('5. w1 and w2 taken = %d and %d',w1,w2))
%disp(sprintf('          R  E  S  U  L  T  S %d'))
%disp(sprintf('       x1=%d', x1g(1)));
%disp(sprintf('       x2=%d', x2g(1)));
%disp(sprintf('       F=%d', minf(1)));
%disp(sprintf('       it=%d', i));
```

Code for the ELD in IEEE 5 bus system using IPSO 3

```
clear all
clc
disp(' we have to minimize the cost function of a 3 machine system')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
a=10^(-4)*[50 50];
b=10^(-2)*[351 389];
c=[44.4 40.6];
B=10^(-2)*[0.0349 0.0086; -0.0055 0.0371];
p1=zeros(p,it); %no. of iterations are pre decided that is it will be maximum 50
p2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
df=zeros(p,it);
sp=zeros(p,it);
csp=zeros(p,it);
pl=zeros(p,it);
c1=zeros(p,it);
c2=zeros(p,it);
C=zeros(p,it);
rp=1;
rg=1;
%cp=1;
%cg=1;
```

```
cpmax=input('cpmax= ');
cpmin=input('cpmin= ');
cgmax=input('cgmax= ');
cgmin=input('cgmin= ');
pd=160;
   p1g=zeros(p);
   p2g=zeros(p);
   fp=zeros(1,p);
   plp=zeros(1,p);

k=50;
w1=1;
w2=0;
% Initial values i.e. 0th iteration
% p1(:,1)=[0.2356 0.5478 1.2453 1.5897 ];
% p2(:,1)=[1.1254 1.3658 1.9875 1.5632 ];
n=1;
while n==1
   for j=1:p
   p1(j,1)=unifrnd(30,120,1);
   p2(j,1)=pd-p1(j,1);
   if p2(j,1)<30&&p3(j,1)>120
      n=1;
      break;
   else
      n=0;
   end
   end
end
% v1(:,1)=[-0.2 -0.1 -0.2 -0.2 -0.1 -0.1 -0.2 -0.1 -0.3 -0.2];
% v2(:,1)=[-0.2 -0.1 -0.3 -0.1 -0.2 -0.1 -0.3 -0.1 -0.2 -0.1];
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);

%Total cost calculation
   for j=1:p
      c1(j,1) = a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1);
      c2(j,1) = a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2);
      C(j,1) = c1(j,1) + c2(j,1);
   end

%To calculate initial value of cost function we need PL
for j=1:p
   pl(j,1)= [p1(j,1) p2(j,1)]*B*[p1(j,1) p2(j,1)]';
end

%To calculate initial value of cost function
for j=1:p
   f(j,1)= w1*((a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1)) + (a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2)))...
         + w2*pl(j,1) + k*abs(pd+pl(j,1)-p1(j,1)-p2(j,1));
end
%0th iteration data display
   disp('this is the 0th iteration')
   print0 = [p1(:,1) p2(:,1) v1(:,1) v2(:,1) f(:,1) c1(:,1) c2(:,1) C(:,1)];
   disp('     P1        P2        V1        v2      f       c1        c2        C ')
   disp(print0)

%Initial personal besst values
```

```
p1p=p1(:,1);
p2p=p2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for m=1:p
   if f(m,1)==fmin
      gb=m;
   else
   end
end
%Initial global best value
for m=1:p
p1g(m) = p1(gb,1);
p2g(m) = p2(gb,1);
end
fgm = min(f(:,1));

%Main iterations starts from here
for i=1:it
   disp(sprintf('This is iteration no.= %d',i))

%for inertia weight W
   wmax=0.6;
   wmin=0.6;
   w = wmax-i*((wmax-wmin)/it);
   cg=cgmax-i*((cgmax-cgmin)/it);
   cp=cpmin-i*((cpmin-cpmax)/it);
   disp(sprintf('cp= %d ',cp))
   disp(sprintf('cg= %d ',cg))

   %For calculatiing velocities for updation
   for j=1:p
      v1(j,(i+1)) = w*v1(j,i) + rp*cp*(p1p(j)-p1(j,i)) + rg*cg*(p1g(j)-p1(j,i));
      v2(j,(i+1)) = w*v2(j,i) + rp*cp*(p2p(j)-p2(j,i)) + rg*cg*(p2g(j)-p2(j,i));
   end
   %V(min) and V(max) constraint
   for j=1:p
      if v1(j,(i+1))< -15
         v1(j,(i+1))= -15;
      end
      if v2(j,(i+1))< -15
         v2(j,(i+1))= -15;
      end
      if v1(j,(i+1))> 60
         v1(j,(i+1))= 60;
      end
      if v2(j,(i+1))> 60
         v2(j,(i+1))= 60;
      end
   end

%Updation of p values
   for j=1:p
      p1(j,(i+1)) = p1(j,i) + v1(j,(i+1));
      p2(j,(i+1)) = p2(j,i) + v2(j,(i+1));
   end
   %Pmin and Pmax constraint
```

```
    for j=1:p
       if p1(j,(i+1))< 30
          p1(j,(i+1))= 30;
       end
       if p2(j,(i+1))< 30
          p2(j,(i+1))= 30;
       end
       if p1(j,(i+1))> 120
          p1(j,(i+1))= 120;
       end
       if p2(j,(i+1))> 120
          p2(j,(i+1))= 120;
       end
    end

    %For losses formulation (PL)
    for j=1:p
       pl(j,(i+1))= [p1(j,(i+1)) p2(j,(i+1))]*B*[p1(j,(i+1)) p2(j,(i+1))]';
    end

  %Main objective function
     for j=1:p
        f(j,(i+1))= w1*((a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1)) +...
                    (a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2)))+...
                    w2*pl(j,(i+1)) + k*abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
    end

  %personal best values updation
     %For losses formulation (PL)
    for j=1:p
       plp(j)= [p1p(j) p2p(j)]*B*[p1p(j) p2p(j)]';
    end

    for j=1:p
       fp(j)= w1*((a(1)*(p1p(j))^2 + b(1)*p1p(j) + c(1)) + (a(2)*(p2p(j))^2 + b(2)*p2p(j) + c(2)))...
            + w2*plp(j) +  k*abs(pd+plp(j)-p1p(j)-p2p(j));
    end
    for m=1:p
       if f(m,i)< fp(m)
          p1p(m)=p1(m,(i+1));
          p2p(m)=p2(m,(i+1));

       else
       end
    end

  %for Global best values updation
    if min(f(:,(i+1)))<fgm
       fgm=min(f(:,(i+1)));
    else
    end

    for j=1:(i+1)
       for m=1:p
          if f(m,j)==fgm
             for l=1:p
                p1g(l) = p1(m,j);     %global best values
                p2g(l) = p2(m,j);
```

```
            end
        else
        end
    end
end

%For Cost Calculation
for j=1:p
    c1(j,(i+1)) = a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1);
    c2(j,(i+1)) = a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2);
    C(j,(i+1)) = c1(j,(i+1)) + c2(j,(i+1));
end

%To find change in the values of f
for j=1:p
   df(j,i)= abs(f(j,(i+1))-f(j,i));
   sp(j,i)= abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
   csp(j,i)= abs(C(j,(i+1))-C(j,i));
end

print = [p1(:,(i+1)) p2(:,(i+1)) v1(:,(i+1)) v2(:,(i+1)) f(:,(i+1)) c1(:,(i+1)) c2(:,(i+1))  C(:,(i+1))];
disp('     P1        P2        V1        v2        f        c1        c2        C ')
disp(print)


%Stoping criterion (df(j,i)<=10^(-6))&& &&(csp(j,i)<=10^(-6))
   ki=0;
   for j=1:p
      if ((df(j,i)<=10^(-6))&&(sp(j,i)<=10^(-6)))
         ki=ki+1;
      end
   end
   if ki >= p
      break
   end

end

disp(' we have to minimize the cost function of a 2 machine 5 bus system')
disp(sprintf('No. of particles used in a swarm = %d',p))
disp(sprintf('Max. no. of iterations entered = %d\n',it))

disp(sprintf('Total demand of power Pd = %d \n',pd))

disp('Initial values of generations of 2 generators')
initial=[p1(:,1) p2(:,1) ];
disp('    P1        P2    ')
disp(initial)

disp(sprintf('\nPD+Pl = %d',pd+pl(1,i)))
disp(sprintf('\nP1+P2=%d\n',p1(1,i)+p2(1,i)))

disp(sprintf('No. of total Iterations took place = %d \n',i))
disp(sprintf('Total loses in the lines Pl = %d \n',pl(1,i)))
disp(sprintf('Minimum cost incured = %d \n',C(1,i)))
disp('\nFinal values of generations of the three generators')
disp(sprintf('P1=%d',p1(1,i)))
```

```
disp(sprintf('P2=%d',p2(1,i)))

disp('About this run')
disp('1. The Constraints has been included as absolute value.')
disp('2. Random values between the limits of generation have been taken for each generator as the different starting point.')
disp('3. Correct values of B coefficients have been fed.')
disp(sprintf('4. K taken = %d',k))
disp(sprintf('5. w1 and w2 taken = %d and %d',w1,w2))
%disp(sprintf('          R  E  S  U  L  T  S %d'))
%disp(sprintf('       x1=%d', x1g(1)));
%disp(sprintf('       x2=%d', x2g(1)));
%disp(sprintf('       F=%d', minf(1)));
%disp(sprintf('       it=%d', i));
```

Code for the ELD in IEEE 5 bus system using IPSO 4

```
clear all
clc
disp(' we have to minimize the cost function of a 3 machine system')
p=input('Enter the no. of particles in a swarm');          %no. of particles
it=input('Enter the no. of iterations');
a=10^(-4)*[50 50];
b=10^(-2)*[351 389];
c=[44.4 40.6];
B=10^(-2)*[0.0349 0.0086; -0.0055 0.0371];
p1=zeros(p,it); %no. of iterations are pre decided that is it will be maximum 50
p2=zeros(p,it);
v1=zeros(p,it);
v2=zeros(p,it);
f=zeros(p,it);
df=zeros(p,it);
sp=zeros(p,it);
csp=zeros(p,it);
pl=zeros(p,it);
c1=zeros(p,it);
c2=zeros(p,it);
C=zeros(p,it);
rp=1;
rg=1;
%cp=1;
%cg=1;
cpmax=input('cpmax= ');
cpmin=input('cpmin= ');
cgmax=input('cgmax= ');
cgmin=input('cgmin= ');
pd=160;
   p1g=zeros(p);
   p2g=zeros(p);
   fp=zeros(1,p);
   plp=zeros(1,p);

k=50;
w1=1;
w2=0;
% Initial values i.e. 0th iteration
% p1(:,1)=[0.2356 0.5478 1.2453 1.5897 ];
% p2(:,1)=[1.1254 1.3658 1.9875 1.5632 ];
```

```
n=1;
while n==1
   for j=1:p
   p1(j,1)=unifrnd(30,120,1);
   p2(j,1)=pd-p1(j,1);
   if p2(j,1)<30&&p3(j,1)>120
      n=1;
      break;
   else
      n=0;
   end
   end
end
% v1(:,1)=[-0.2 -0.1 -0.2 -0.2 -0.1 -0.1 -0.2 -0.1 -0.3 -0.2];
% v2(:,1)=[-0.2 -0.1 -0.3 -0.1 -0.2 -0.1 -0.3 -0.1 -0.2 -0.1];
v1(:,1)=rand(1,p);
v2(:,1)=rand(1,p);

%Total cost calculation
   for j=1:p
      c1(j,1) = a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1);
      c2(j,1) = a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2);
      C(j,1) = c1(j,1) + c2(j,1);
   end

%To calculate initial value of cost function we need PL
for j=1:p
   pl(j,1)= [p1(j,1) p2(j,1)]*B*[p1(j,1) p2(j,1)]';
end

%To calculate initial value of cost function
for j=1:p
   f(j,1)= w1*((a(1)*(p1(j,1))^2 + b(1)*p1(j,1) + c(1)) + (a(2)*(p2(j,1))^2 + b(2)*p2(j,1) + c(2)))...
         + w2*pl(j,1) + k*abs(pd+pl(j,1)-p1(j,1)-p2(j,1));
end
%0th iteration data display
   disp('this is the 0th iteration')
   print0 = [p1(:,1) p2(:,1) v1(:,1) v2(:,1) f(:,1) c1(:,1) c2(:,1) C(:,1)];
   disp('     P1        P2        V1        v2      f       c1        c2        C ')
   disp(print0)

%Initial personal besst values
p1p=p1(:,1);
p2p=p2(:,1);

%for Initial Global best values updation
fmin=min(f(:,1));
for m=1:p
   if f(m,1)==fmin
      gb=m;
   else
   end
end
%Initial global best value
for m=1:p
p1g(m) = p1(gb,1);
p2g(m) = p2(gb,1);
end
```

```
fgm = min(f(:,1));

%Main iterations starts from here
for i=1:it
   disp(sprintf('This is iteration no.= %d',i))

%for inertia weight W
   wmax=0.6;
   wmin=0.6;
   w = wmax-i*((wmax-wmin)/it);
   cg=cgmin-i*((cgmin-cgmax)/it);
   cp=cpmax-i*((cpmax-cpmin)/it);
   disp(sprintf('cp= %d ',cp))
   disp(sprintf('cg= %d ',cg))

   %For calculatiing velocities for updation
   for j=1:p
      v1(j,(i+1)) = w*v1(j,i) + rp*cp*(p1p(j)-p1(j,i)) + rg*cg*(p1g(j)-p1(j,i));
      v2(j,(i+1)) = w*v2(j,i) + rp*cp*(p2p(j)-p2(j,i)) + rg*cg*(p2g(j)-p2(j,i));
   end
   %V(min) and V(max) constraint
   for j=1:p
      if v1(j,(i+1))< -15
         v1(j,(i+1))= -15;
      end
      if v2(j,(i+1))< -15
         v2(j,(i+1))= -15;
      end
      if v1(j,(i+1))> 60
         v1(j,(i+1))= 60;
      end
      if v2(j,(i+1))> 60
         v2(j,(i+1))= 60;
      end
   end

%Updation of p values
   for j=1:p
      p1(j,(i+1)) = p1(j,i) + v1(j,(i+1));
      p2(j,(i+1)) = p2(j,i) + v2(j,(i+1));
   end
   %Pmin and Pmax constraint
   for j=1:p
      if p1(j,(i+1))< 30
         p1(j,(i+1))= 30;
      end
      if p2(j,(i+1))< 30
         p2(j,(i+1))= 30;
      end
      if p1(j,(i+1))> 120
         p1(j,(i+1))= 120;
      end
      if p2(j,(i+1))> 120
         p2(j,(i+1))= 120;
      end
   end

   %For losses formulation (PL)
```

```
    for j=1:p
       pl(j,(i+1))= [p1(j,(i+1)) p2(j,(i+1))]*B*[p1(j,(i+1)) p2(j,(i+1))]';
    end

  %Main objective function
     for j=1:p
        f(j,(i+1))= w1*((a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1)) +...
                    (a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2)))+...
                    w2*pl(j,(i+1)) + k*abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
    end

  %personal best values updation
     %For losses formulation (PL)
    for j=1:p
       plp(j)= [p1p(j) p2p(j)]*B*[p1p(j) p2p(j)]';
    end

    for j=1:p
       fp(j)= w1*((a(1)*(p1p(j))^2 + b(1)*p1p(j) + c(1)) + (a(2)*(p2p(j))^2 + b(2)*p2p(j) + c(2)))...
            + w2*plp(j) +  k*abs(pd+plp(j)-p1p(j)-p2p(j));
    end
    for m=1:p
       if f(m,i)< fp(m)
          p1p(m)=p1(m,(i+1));
          p2p(m)=p2(m,(i+1));

       else
       end
    end

  %for Global best values updation
     if min(f(:,(i+1)))<fgm
        fgm=min(f(:,(i+1)));
     else
     end

    for j=1:(i+1)
       for m=1:p
          if f(m,j)==fgm
             for l=1:p
                p1g(l) = p1(m,j);     %global best values
                p2g(l) = p2(m,j);
             end
          else
          end
       end
    end

    %For Cost Calculation
    for j=1:p
        c1(j,(i+1)) = a(1)*(p1(j,(i+1)))^2 + b(1)*p1(j,(i+1)) + c(1);
        c2(j,(i+1)) = a(2)*(p2(j,(i+1)))^2 + b(2)*p2(j,(i+1)) + c(2);
        C(j,(i+1)) = c1(j,(i+1)) + c2(j,(i+1));
    end

    %To find change in the values of f
    for j=1:p
       df(j,i)= abs(f(j,(i+1))-f(j,i));
```

```
        sp(j,i)= abs(pd+pl(j,(i+1))-p1(j,(i+1))-p2(j,(i+1)));
        csp(j,i)= abs(C(j,(i+1))-C(j,i));
    end

    print = [p1(:,(i+1)) p2(:,(i+1)) v1(:,(i+1)) v2(:,(i+1)) f(:,(i+1)) c1(:,(i+1)) c2(:,(i+1))  C(:,(i+1))];
    disp('     P1        P2        V1        v2        f        c1        c2        C ')
    disp(print)


%Stoping criterion (df(j,i)<=10^(-6))&& &&(csp(j,i)<=10^(-6))
    ki=0;
    for j=1:p
        if ((df(j,i)<=10^(-6))&&(sp(j,i)<=10^(-6)))
            ki=ki+1;
        end
    end
    if ki >= p
        break
    end

end

disp(' we have to minimize the cost function of a 2 machine 5 bus system')
disp(sprintf('No. of particles used in a swarm = %d',p))
disp(sprintf('Max. no. of iterations entered = %d\n',it))

disp(sprintf('Total demand of power Pd = %d \n',pd))

disp('Initial values of generations of 2 generators')
initial=[p1(:,1) p2(:,1) ];
disp('    P1        P2    ')
disp(initial)

disp(sprintf('\nPD+Pl = %d',pd+pl(1,i)))
disp(sprintf('\nP1+P2=%d\n',p1(1,i)+p2(1,i)))

disp(sprintf('No. of total Iterations took place = %d \n',i))
disp(sprintf('Total loses in the lines Pl = %d \n',pl(1,i)))
disp(sprintf('Minimum cost incured = %d \n',C(1,i)))
disp('\nFinal values of generations of the three generators')
disp(sprintf('P1=%d',p1(1,i)))
disp(sprintf('P2=%d',p2(1,i)))

disp('About this run')
disp('1. The Constraints has been included as absolute value.')
disp('2. Random values between the limits of generation have been taken for each generator as the different
starting point.')
disp('3. Correct values of B coefficients have been fed.')
disp(sprintf('4. K taken = %d',k))
disp(sprintf('5. w1 and w2 taken = %d and %d',w1,w2))
%disp(sprintf('          R  E  S  U  L  T  S %d'))
%disp(sprintf('       x1=%d', x1g(1)));
%disp(sprintf('       x2=%d', x2g(1)));
%disp(sprintf('       F=%d', minf(1)));
%disp(sprintf('       it=%d', i));
```

REFERENCES

[1] K.E. PARSOPOULOS and M.N. VRAHATIS "Recent approaches to global optimization problems through Particle Swarm Optimization" University of Patras, GR-26110 Patras, Greece 2002.

[2] Zhiyu You,Weirong Chen,Guojun He,Xiaoqiang Nan "Adaptive Weight Particle Swarm Optimization Algorithm with Constriction Factor" Southwest Jiaotong University Chengdu, China 2010.

[3] J. C. Bansal, P. K. Singh Mukesh Saraswat, Abhishek Verma, Shimpi Singh Jadon,Ajith Abraham "Inertia Weight Strategies in Particle Swarm Optimization" VSB Technical University of Ostrava, Czech Republic 2011

[4] Russell Eberhart & James Kennedy "A New Optimizer Using Particle Swarm Theory" Sixth International Symposium on Micro Machine and Human Science 0-7803-2676-8/95 01995 IEEE.

[5] Ajith Abraham, He Guo2 and Hongbo Liu "Swarm Intelligence: Foundations, Perspectives and Applications" A. Abraham et al.: Swarm Intelligence: Foundations, Perspectives and Applications, Studies in Computational Intelligence (SCI) 26, 3–25 (2006)

[6] Ehsan Shahamatnia & Mohamad Mehdi Ebadzadeh "Application of Particle Swarm Optimization and Snake Model Hybrid on Medical Imaging" 978-1-61284-336-0/11/ 2011 IEEE

[7] Changming Ji, Fang Liu, Xinming Zhang "Particle Swarm Optimization Based on Catfish Effect for Flood Optimal Operation of Reservoir" 978-1-4244-9953-3/11/$26.00 ©2011 IEEE

[8] Steffen Freitag, Rafi L. Muhanna and Wolfgang Graf "A Particle Swarm Optimization Approach for Training Artificial Neural Networks with Uncertain Data" Institute of Structural Mechanics ISBN 978-80-214-4507-9. Published by Ing. Vladislav Pokorný – LITERA

[9] Ismail Ibrahim, Zulkifli Md. Yusof, Sophan,Wahyudi Nawawi, Muhammad Arif Abdul Rahim,,Kamal Khalil, Hamzah Ahmad, Zuwairie Ibrahim" A Novel Multi-State Particle Swarm Optimization for Discrete Combinatorial Optimization Problems" 2166-8531/12 $26.00 © 2012 IEEE

[10] J.J. Jamian, M.W. Mustafa, H. Mokhlis, M.N. Abdullah" Comparative Study on Distributed Generator Sizing Using Three Types of Particle Swarm Optimization" 978-0-7695-4668-1/12 $26.00 © 2012 IEEE

[11] Chien-Ching Chiu, Chi-Hsien Sun&Chun-Fu Li "Comparison of Dynamic Differential Evolution and Asynchronous Particle Swarm Optimization for Inverse Scattering" ICCIT 2012

[12] Liu Jin-yue, Zhu Bao-ling" The Application of Particle Swarm Optimization Algorithm in the Extremum Optimization of Nonlinear Function" 978-0-7695-4858-6/12 $26.00 © 2012 IEEE

[13] Jun Sun" Solving the Power Economic Dispatch Problem With Generator Constraints by Random Drift Particle Swarm Optimization" IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 10, NO. 1, FEBRUARY 2014

[14] C.L. Wadhwa, N.K. Jain: "Multiple objective optimal load flow: a new perspective", IEE Proceedings, Vol. 137, Pt. C, No. 1, January 1990, pp. 59-64

[15] Nangia, Uma, Jain, N.K., Wadhwa, C.L., 'Investigations of Multiobjective Optimal Load Flow Study by Sequential Goal Programming ', Journal of IE, Vol. 77, August 1996, pp. 99-103.

[16] Nangia, Uma, Jain, N.K., Wadhwa, C.L., ' Multiobjective Load Flow Studies through Maximization of Minimum Relative Attainments ', Journal of IE, Vol. 77, Nov. 96, pp. 154-159.

[17] Nangia U., Jain, N.K., Wadhwa, C.L., ' Surrogate Worth Tradeoff Technique for Multiobjective Optimal Power Flows, IEE Proc.- Generation, Transmission, Distribution, November 1997, Vol. 144, No. 6, pp. 547-553.

[18] Nangia, Uma, Jain, N.K., Wadhwa, C.L., ' A New Interactive Step Method for Multiobjective Optimal Load Flow Study', Journal of IE, Vol. 78, March 1998, pp. 225-228.

[19] Nangia, Uma, Jain, N.K., Wadhwa, C.L., 'Noninferior Set Estimation for Multiobjective Optimal Load Flow Study', Journal of IE, Vol. 78, March 1998, pp. 229-235.

[20] Nangia, Uma, Jain, N.K., Wadhwa, C.L., ' Optimal Weight Assessment based on Range of Objectives in Multiobjective Optimal Load Flow Study', IEE Proc. C, Vol. 145, No. 1, January 1998, pp. 65-69.

[21] Stevenson, W.D., JR.: 'Elements of power system analysis' (McGraw Hill Book Co., 2nd Ed., 1962) p. 219.

[22] M. A. Abido: "Multiobjective Evolutionary Algorithms for Electric Power Dispatch Problem". IEEE Transactions on Evolutionary Computation, Vol. 10, no. 3, June 2006, pp. 315-329.

[23] K. Basu, A. Bhattacharya, S. Chowdhury, S. P. Chowdhury: "Planned Scheduling for Economic Power Sharing in a CHP-Based Micro-Grid". IEEE Transactions on Power Systems, Vol. 27, No. 1, February 2012. pp.30-38.