

# **A NEW PARTICLE SWARM OPTIMIZATION ALGORITHM AND ITS APPLICATION TO LOAD FLOW**

**M.Tech. Dissertation**

**BY**

**Uttam Kumar  
2K14/PSY/19**



**DEPARTMENT OF ELECTRICAL ENGINEERING  
DELHI TECHNOLOGICAL UNIVERSITY  
DELHI-110042 (INDIA)**

**JUNE, 2016**

# **A NEW PARTICLE SWARM OPTIMIZATION AND ITS APPLICATION TO LOAD FLOW**

## **M.Tech Dissertation**

*Submitted in partial fulfillment of the  
requirements for the award of the degree*

*of*

**Master of Technology**

*in*

**Power Systems**

**BY**

**UTTAM KUMAR**

**2K14/PSY/19**



**DEPARTMENT OF ELECTRICAL ENGINEERING  
DELHI TECHNOLOGICAL UNIVERSITY  
DELHI-110042 (INDIA)**

**JUNE, 2016**

## **CERTIFICATE**

I hereby certify that the work which is being presented in the M.Tech. Dissertation entitled “ **A New Particle Swarm Optimization Algorithm and its Application to Load Flow**”, in partial fulfillment of the requirements for the award of the **Degree of Master of Technology in Electrical Engineering** and submitted to the Department of Electrical Engineering of Delhi Technological University is an authentic record of my own work carried out under the supervision of **Prof. N.K. Jain (Professor) and Prof. Uma Nangia (Professor)**, EE Department.

The matter presented in this report has not been submitted by me for the award of any other **Degree/Diploma** elsewhere.

**Uttam Kumar**  
**2K14/PSY/19**

**Date: 20<sup>th</sup> June, 2016**

**Mentors**

**Prof. N.K. Jain**  
Professor  
EE Dept., DTU

**Prof. Uma Nangia**  
Professor  
EE Dept., DTU

## ***LIST OF FIGURES***

FIGURE 1. FLOWCHART OF PSO

FIGURE 2. FLOWCHART FOR SEPSO

FIGURE 3. PARAMETERS' WINDOW FOR ENTERING PSO PARAMETERS

FIGURE 4. WINDOW FOR SELECTING THE BENCHMARK FUNCTION

FIGURE 5. PARAMETERS' WINDOW FOR ENTERING SEPSO PARAMETERS

FIGURE 6. COMPARISON OF SEPSO AND CONVENTIONAL PSO FOR ROSENBROCK'S FUNCTION.

FIGURE 7. COMPARISON OF SEPSO AND CONVENTIONAL PSO FOR GRIEWANK'S FUNCTION.

FIGURE 8. COMPARISON OF SEPSO AND CONVENTIONAL PSO FOR SPHERE FUNCTION.

FIGURE 9. SEPSO COMPARED WITH PSO FOR RASTRIGIN'S FUNCTION.

FIGURE 10 COMPUTATIONAL EFFORT FOR CONVENTIONAL PSO AND SEPSO (IEEE 5 BUS SYSTEM).

FIGURE 11 COMPUTATIONAL EFFORT FOR CONVENTIONAL PSO AND SEPSO (IEEE 14 BUS SYSTEM).

FIGURE I. IEEE 5 BUS SYSTEM

FIGURE II. IEEE 14 BUS SYSTEM

## ***LIST OF TABLES***

TABLE 1. 2-D BENCHMARK FUNCTIONS

TABLE 2. PLOTS OF 2-D BENCHMARK FUNCTIONS

TABLE 3. RESULTS OF CONVENTIONAL PSO (ROSENBROCK)

TABLE 4. RESULTS OF SEPSO (R=2) (ROSENBROCK)

TABLE 5. RESULTS OF CONVENTIONAL PSO (GRIEWANK)

TABLE 6. RESULTS OF SEPSO (FOR R=2) (GRIEWANK)

TABLE 7. RESULTS OF CONVENTIONAL PSO (SPHERE)

TABLE 8. RESULTS OF SEPSO (SPHERE) FOR R=2

TABLE 9. RESULTS OF CONVENTIONAL PSO (RASTRIGIN)

TABLE 10. RESULTS OF SEPSO (R=2) (RASTRIGIN)

TABLE 11: COMPUTATIONAL EFFORT FOR LOAD FLOW USING CONVENTIONAL PSO (IEEE 5 BUS SYSTEM)

TABLE 12: COMPUTATIONAL EFFORT FOR LOAD FLOW USING SEPSO. (IEEE 5 BUS SYSTEM)

TABLE 13: RESULTS OF LOAD FLOW USING SEPSO; FS=2 AND R=2; (IEEE 5 BUS SYSTEM)

TABLE 14: COMPUTATIONAL EFFORT FOR LOAD FLOW USING CONVENTIONAL PSO (IEEE 14 BUS SYSTEM)

TABLE 15: COMPUTATIONAL EFFORT FOR LOAD FLOW USING SEPSO (IEEE 14 BUS SYSTEM).

TABLE 16: RESULTS OF LOAD FLOW USING SEPSO (IEEE 14 BUS SYSTEM).

TABLE 17. COMPARISON OF SEPSO WITH CONVENTIONAL PSO (IEEE 5 BUS SYSTEM).

TABLE 18. COMPARISON OF SEPSO WITH CONVENTIONAL PSO (IEEE 14 BUS SYSTEM).

TABLE 19: RESULTS OF LOAD FLOW USING NEWTON RAPHSON'S (NR) METHOD  
(IEEE 5 BUS SYSTEM)

TABLE 20: RESULTS OF LOAD FLOW USING NR METHOD (IEEE 14 BUS SYSTEM).

TABLE I: SPECIFIED VALUES OF LOAD(S), GENERATION(S), VOLTAGE(S) AND  
ANGLE(S).

TABLE II: LINE REACTANCE VALUES AND LINE RESISTANCE VALUES.

TABLE III: LINE REACTANCE VALUES, LINE RESISTANCE VALUES AND LINE  
SUSCEPTANCE VALUES.

TABLE IV: SPECIFIED VALUES OF LOAD(S), GENERATION(S), VOLTAGE(S) AND  
ANGLE(S).

TABLE V: REACTIVE POWER LIMITS.

## **ACKNOWLEDGEMENT**

Apart from the efforts of me, the success of any project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of my M.Tech thesis.

We also thank our Head of Electrical Department Dr. Madhusudan Singh for supporting our work.

I must give my high, respectful gratitude to my supervisors, Prof. N.K. Jain and Prof. Uma Nangia for their guidance, supervision and help throughout this project. I have learned a lot throughout this course, with many challenging yet valuable experience. My endless thanks to Prof. N.K. Jain and Prof. Uma Nangia for giving me the chance to explore a new knowledge of mine, as well as for giving me precious advices in order to improve myself in every aspect.

Finally, I would like to thank all those people who have directly or indirectly helped me successfully to complete my dissertation.

Uttam Kumar  
2K14/PSY/19

## **ABSTRACT**

A new variant of Particle Swarm Optimization (PSO) algorithm along with a novel approach of implementing reactive power constraints for PV buses has been presented in this thesis. The new PSO version, Selection Enabled PSO (SEPSO), “selects” particles with better objective function value and eliminates worse particles after a certain number of iterations. The computational efficiency of SEPSO over the conventional PSO is ratified by using it to minimize mathematical benchmark functions. The number of function evaluations up to convergence is significantly reduced in case of SEPSO. Henceforth, SEPSO is applied to perform load flow studies on IEEE 5 bus and IEEE 14 bus system. As expected, a considerable reduction in terms of function evaluations is observed when contrasted with the number of function evaluations required by conventional PSO to perform load flow on IEEE 5 bus and IEEE 14 bus system.



# Contents

<b>CHAPTER-1: INTRODUCTION.....</b>	<b>1</b>
1.1 Overview.....	1
1.2 Aim and Approach.....	2
1.3 Literature Review.....	3
1.4 Plan of Thesis.....	4
<b>CHAPTER-2: PARTICLE SWARM OPTIMIZATION.....</b>	<b>5</b>
2.1 Introduction.....	5
2.2 Model of PSO.....	6
2.3 Parameters of PSO.....	7
2.3.1 Swarm size.....	7
2.3.2 Inertia weight factor.....	7
2.3.3 Cognitive learning acceleration factor.....	8
2.3.4 Social learning acceleration factor.....	8
2.4 Algorithm .....	8
2.5 Flow chart.....	9
2.6 Applications of PSO.....	10
2.7 Advantages and limitations of PSO.....	10
<b>CHAPTER-3: SELECTION ENABLED PARTICLE SWARM OPTIMIZATION.....</b>	<b>11</b>
3.1 Introduction.....	11
3.2 The SEPSO model.....	12
3.3 Additional parameters for SEPSO.....	12
3.3.1 Reduction factor.....	12
3.3.2 Sorting frequency.....	12
3.4 Algorithm .....	12
3.5 Flow chart.....	13
3.6 Advantage of SEPSO.....	14

<b>CHAPTER-4: COMPARISON OF SEPSO WITH PSO.....</b>	<b>15</b>
4.1 Benchmark functions.....	15
4.2 Computational results .....	18
4.2.1 Rosenbrock function.....	18
4.2.2 Griewank function.....	22
4.2.3 Rastrigin function.....	23
4.2.4 Sphere function.....	24
4.3 Discussion.....	25
 <b>CHAPTER-5: LOAD FLOW STUDIES USING SEPSO.....</b>	 <b>26</b>
5.1 Problem formulation.....	27
5.2 Computational procedure .....	27
5.3 Computational results.....	29
5.3.1 Results for IEEE 5 bus system.....	29
5.3.2 Results for IEEE 14 bus system.....	31
5.4 Comparison of SEPSO and PSO for load flow.....	34
5.5 Results of N-R algorithm.....	38
 <b>CHAPTER-6: CONCLUSION AND FUTURE DIRECTION.....</b>	 <b>40</b>
6.1 Conclusion.....	40
6.2 Future direction.....	41
 APPENDIX.....	 42
A. IEEE 5 bus system.....	42
B. IEEE 14 bus system.....	44
i. MATLAB code for load flow on IEEE 5 bus system using SEPSO.....	47
ii. MATLAB code for load flow on IEEE 14 bus system using SEPSO.....	59
iii. MATLAB code for minimizing benchmark functions using PSO.....	68
iv. MATLAB code for minimizing benchmark functions using SEPSO.....	70
v. Benchmark functions implemented in MATLAB.....	72
 REFERENCES.....	 74

# **Chapter-1**

## **INTRODUCTION**

### **1.1 Overview**

This thesis puts forth a new variant of the conventional particle swarm optimization (PSO) algorithm, the selection enabled particle swarm optimization (SEPSO) algorithm to perform load flow.

Load flow studies are of cardinal importance when it comes to designing, planning and future capacity augmenting of a power system. Not only the load flow analysis yields voltages and corresponding voltage angles for each bus in the power system but also the total transmission loss occurred as generator(s) supply the load. Iterative numerical methods are usually employed to perform the load flow. Although methods like Newton Raphson, show fast convergence, they are more complex to implement than the conventional PSO. No research paper performing load flow and explicitly mentioning how reactive power constraints were implemented has yet been reported in any journal of repute. The novelty of the present work lies in successfully implementing PSO for load flow study. The computational efficiency of the new algorithm (SEPSO) has been compared to that of the conventional PSO and NR.

Particle swarm optimization, a stochastic metaheuristic algorithm, when applied to load flow, does away with the complexity of the Newton-Raphson (N-R) algorithm. In addition, the speed of convergence is also satisfactory. Building on these advantages, the SEPSO algorithm aims to further reduce the computational effort and consequently speeds up the convergence.

The modified PSO algorithm, SEPSO, presented in this work is implemented in computer code using MATLAB. The SEPSO algorithm is tested using mathematical benchmark functions and then tried on IEEE 5 bus and IEEE 14 bus systems for load flow.

## **1.2 Aim and approach**

The aim of the research work presented in this dissertation is two fold, first to implement PSO for load flow study and second to enhance the computational efficiency of PSO and to exploit this improvement to perform load flow. The proposed PSO is referred to as the selection enabled PSO (SEPSO) in this thesis.

The efficiency achieved, for SEPSO, in terms of reduction in number of function evaluations is first verified using four mathematical benchmark functions. The results for the conventional PSO and SEPSO are compared in terms of the number of function evaluations up to the convergence. Load flow is then performed on IEEE 5 bus and IEEE 14 bus system using SEPSO. Results obtained from Newton Raphson method are used to ascertain whether both the algorithms, conventional PSO and SEPSO, have properly converged. As in case of benchmark functions, the algorithms are compared in terms of the number of function evaluations for load flow.

### 1.3 Literature Review

During the past years a number of intelligent techniques – GA, PSO and ACO have been proposed by researchers. These intelligent techniques are derivative free and simple to implement. Though these techniques sometimes take large number of iterations to converge, yet they guarantee global optimum solution.

PSO is a population based stochastic algorithm, developed by Kennedy and Eberhart [1] in 1995, inspired by collective behaviour of bird flocking, fish schooling etc. Many modifications have followed the original algorithm, like Memory Enhanced PSO [2], Predator-Prey PSO [3] and PCPSO [4]. All these modifications add to the basic structure of conventional PSO. As an example, Predator-Prey PSO has two groups of particles called predator and prey. While position update for a prey is identical to that of a particle in case of conventional PSO, a predator has one additional component of velocity directing it towards a prey.

There are newer variants like: CLPSO [5], SLPSO [6] and OLPSO [7] which do not require parameter tuning. These algorithms, unlike previously mentioned variants, have a structure entirely different from the PSO. They make use of the current information and usually do not include the previous iteration(s) information. For example, in case of SLPSO, particles with poorer objective function values learn from particles with better objective function values in the current iteration and update their position. There is no need to store the previous velocities or the previous positions of the particles as in case of conventional PSO.

In all variants of conventional PSO a balance has to be obtained between exploration and exploitation stage for faster convergence. Feng Chen et. al. [8] studied the trade-off strategy between these two stages. Initially the change in the particles' positions are large on account of a larger value of the inertia weight factor. Afterwards when there is sufficient improvement in the objective function value pertaining to the global best, the exploitation stage is induced by reducing the inertia weight factor accordingly [9, 10]. Yuhong Chi et. al. [11] employed the concept of swarm diversity to ascertain if the particles are still exploring or they have found the region containing the global minimum.

Load flow studies can be performed using conventional PSO or any of its variant by formulating an appropriate objective function depending on real and reactive power mismatches. PSO has already been applied to solve various problems in Electrical Engineering including economic load dispatch [12] and optimal load flow [13, 14]. There have been attempts to perform load flow [15, 16] using PSO, these attempts, however, do not address the issue of reactive power generation limits for PV buses [17]. In this dissertation, a strategy has been developed and embedded in the PSO algorithm to deal with such cases. The devised approach is further extended to SEPSO to perform load flow.

## 1.4 Plan of Thesis

This thesis report is organised in six chapters as follows:

1. **Introduction:** This chapter summarizes the research work discussed in this report.
2. **Particle Swarm Optimization:** The inception, parameters, algorithm, flow chart, applications, advantages and limitations of conventional PSO are discussed in detail in this chapter.
3. **Selection Enabled Particle Swarm Optimization:** The SEPSO algorithm is presented in this chapter.
4. **Comparison of SEPSO with PSO:** SEPSO is compared with conventional PSO using four mathematical benchmark functions: Rosenbrock, Rastrigin, Sphere, and Griewank.
5. **Load Flow Studies using SEPSO:** Load flow studies are performed using SEPSO and conventional PSO on IEEE 5 bus system and IEEE 14 bus system.
6. **Conclusion and Future Direction:** This chapter consists of insights into the computational results and the strategies used. Future possibilities of improvement and fusion with other existing intelligent algorithms is also discussed.

## Chapter-2

### PARTICLE SWARM OPTIMIZATION

#### 2.1 Introduction

Particle swarm optimization was developed by Dr. Eberhart and Dr. Kennedy of Purdue School of Engineering and Technology in 1995. Inspired by food searching ability of a swarm, PSO optimizes a problem by having an initial population of solutions (here particles) and moving these particles around in the search space. Each particle's velocity is influenced by its previous velocity, local best known position,  $p_{best}$ , and by the best known position,  $g_{best}$ , of the swarm. These are updated as better positions are found by other particles. This is expected to move the swarm towards the best solution.

The conventional PSO has emerged as a promising research area in application of optimization techniques to solve various problems in the field of Electrical Engineering. The conventional PSO has gained acclaim as it can be employed to solve problems which involve non-linear elements. Though there exists always a specific approach for a specific problem, it is an innovative idea to opt for a generic method like PSO. The charm of this method is that it is simpler to code when compared to methods like GA, while the results are found to be sufficiently accurate. PSO wins hands down in computational efficiency and accuracy when compared to GA.

PSO shares many similarities with other intelligent computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles.

Recently, PSO has been applied successfully to various fields of power system optimization such as power system stabilizer design, reactive power and voltage control. The conventional PSO proposed by Kennedy and Eberhart is directly applicable to the problems with continuous domain

and without any constraints. Therefore, conventional PSO is used with some modifications to take into account these constraints.

## 2.2 Model of PSO

It is a meta-heuristic optimization technique which exploits food searching ability of a swarm. Here the “food” is used as a metaphor for the optimal solution and swarm represents the set of particles spread randomly in the search space. Each of these particles for a particular iteration undergoes displacement depending upon its previous velocity, the best performance for the swarm and each particle's personal best performance. The global best performance is the minimum value of the objective function when all the particles of the swarm and their respective OF (objective function) values are considered. This is the case when OF is to be minimized, for case where OF is to be maximized we consider maximum of OF as global best. The individual best is considered as the value of OF before iteration or after iteration, depending on which one is smaller. Larger of the two values shall be considered for a maximization problem. The global best and the personal best are defined below.

### *a) The global best, gbest*

The global best value corresponds to the best of all values corresponding to the defined objective function, in the swarm including all particles for a given iteration.

### *b) The Personal best, pbest*

The best value of a given particle corresponding to the objective function, up to the current iteration is referred to as the personal best value of the particle.

The velocity update equation and the position update equation are given by equation 1 and equation 2 respectively as follows:



Velocity update:

$$V_k^{i+1} = w \times V_k^i + r_p \times \text{rand}() \times (\text{pbest}_k^i - X_k^i) + r_g \times \text{rand}() \times (\text{gbest}^i - X_k^i) \rightarrow 1$$

Position update:

$$X_k^{i+1} = X_k^i + V_k^{i+1} \rightarrow 2$$

Where,  $V_k^{i+1}$  = velocity of particle k for  $i^{\text{th}}$  iteration.

$w$  = Inertia weight factor.

$r_p$  = Cognitive learning acceleration factor.

$r_g$  = Social learning acceleration factor.

$X_k^{i+1}$  = position of particle k for  $i+1^{\text{th}}$  iteration.

$\text{pbest}_k^i$  = best position of particle k till  $i^{\text{th}}$  iteration.

$\text{gbest}^i$  = position of the overall best particle of the swarm till the  $i^{\text{th}}$  iteration.

## 2.3 Parameters of PSO

Swarm size, inertia weight factor, cognitive learning acceleration factor and social learning acceleration factors are the parameters of conventional PSO. Their values need to be chosen suitably as they are crucial for the algorithm's convergence.

**2.3.1 Swarm size:** Swarm size is the number of particles in a PSO swarm. If the number of particles in the swarm is less than a critical value, the algorithm does not converge.

**2.3.2 Inertia weight factor 'w' :** Inertia weight factor determines the weightage of a particle's previous velocity in the velocity update equation. Higher the value of this factor, greater the

influence of the previous velocity. It can, therefore, be said that this parameter determines the “inertia” of a particle, hence the name inertia weight factor.

**2.3.3 Cognitive learning acceleration factor:** This parameter, which appears as a constant coefficient in the second term of the velocity update equation, is represented by  $r_p$ . An increased value of  $r_p$ , improves the local search capability of the particles and a reduction in  $r_p$  hampers the local searching by the swarm.

**2.3.4 Social learning acceleration factor:** Social learning acceleration factor denoted by  $r_g$ , is used in the third term of the velocity update equation. Higher value of  $r_g$  enhances the global search ability of the swarm. The value  $r_p + r_g$  is usually 4, where  $r_p=2$  and  $r_g=2$ , which is sufficient for most optimization problems.

**2.3.5 Random Factors:** In equation 1, random factors are associated with the cognitive as well as social learning terms. These are useful for better exploration. These are important only when problem to be solved is new and problem solver does not have any idea about the solution of the problem. But, in our case, we have sufficiently good idea about the solution of the problem. For this reason these factors have been dropped in conventional PSO as well as SEPSO.

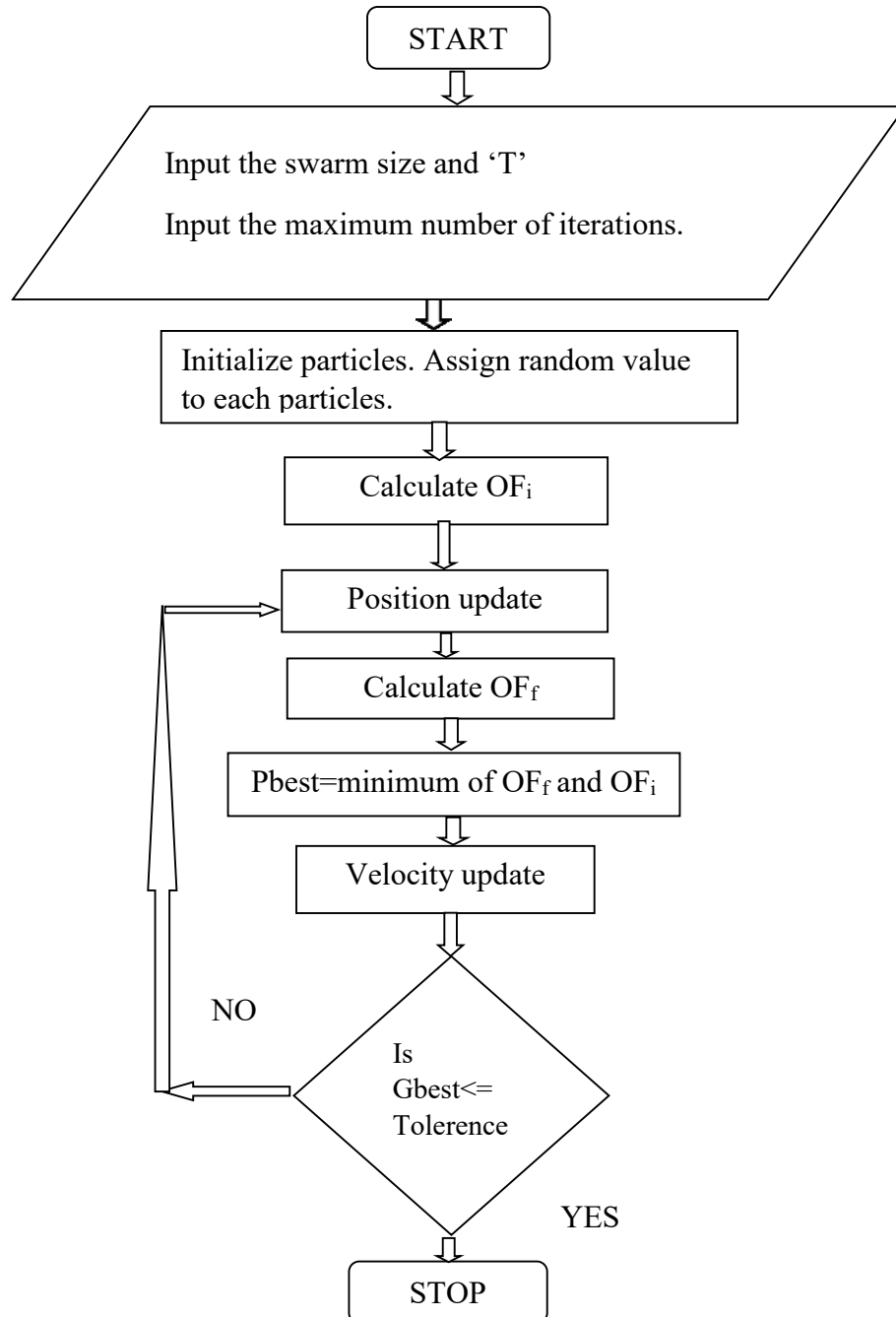
## 2.4 Algorithm

The algorithm for PSO is presented below.

- i) Initialize swarm by assigning positions and velocities to every particle in the swarm. Input ‘T’. It is the negative integer which appears as the exponent in expression for tolerance: tolerance= $10^T$ .
- i) Calculate the objective function value and call it  $OF_i$ .
- ii) Update positions for all the particles, using eq. 2), and calculate their objective function value and call it  $OF_f$ .
- iii) Compare  $OF_f$  and  $OF_i$  for each particle. The position of the particle corresponding to smaller of the two is taken as pbest.
- iv) Find out the gbest by considering the particle with the position yielding the least value of the objective function.
- v) Update velocity for all the particles using eq. 1).

- vi) Check whether the objective function value for the gbest is less than the tolerance specified. If it is less than the tolerance specified, then go to viii) else go to iii)
- vii) Exit and display the gbest's objective function value, the position of the gbest particle and number of function evaluations along with positions and velocities for all particles in the swarm.

## 2.5 Flowchart



**Fig.1. Flowchart of PSO**

## **2.6 Applications of PSO**

Particle swarm optimization, conventional or modified, has been applied to solve problems in various fields of knowledge. It has found its use in areas including but not limited to medical science, economics, operations research, antenna design, and power systems engineering. PSO because of its stochastic nature and simple computer code implementation can be applied to virtually any kind of problem which may or may not be solvable by conventional methods.

In training neural networks PSO has replaced the conventional approach of backpropagation because of its simplicity and efficacy [18]. Diagnosis of human tremor has been accomplished by the algorithm and the results are found to be reliable [19]. Milling operations, an integral part of production, is also optimized using PSO and the pitfalls faced using conventional approaches are averted [20]. For Economics and Operations Management where most of the problems need a “good enough” solution, PSO is highly required [21].

Reactive power control and voltage control in Power Systems can be achieved using PSO [22]. A fuel gauge equivalent for batteries used in battery power vehicles has also been implemented using PSO in conjunction with the neural network. The results for state of the battery pack were found to be sufficiently accurate [23]. Economic load dispatch and optimal power flow are other areas where PSO has been extensively used.

## **2.7 Advantages and limitations of PSO**

The two major advantages of PSO, in comparison to other stochastic algorithms, are that it is easier to implement and there are less parameters to tune. Also, unlike deterministic algorithms, Newton-Raphson for example, PSO is derivative free. As a result, PSO converges even for cases when the Jacobian matrix for a problem becomes zero. In comparison to other stochastic and conventional methods the objective function has less negative impact on the convergence of PSO. Less number of parameters, in contrast to other stochastic methods, to adjust is another strong point of the algorithm. Additionally, PSO seems to be more immune to the non-convergence which might occur when initial values are not chosen suitably or lie outside a desirable range [24].

That said, PSO has some limitations as well. Lack of a solid mathematical framework is a major drawback. Also, the time taken to solve problems increases with the number of variables faster than a linear relationship. However, this can be resolved to some extent by parameter tuning.

## Chapter-3

### Selection Enabled Particle Swarm Optimization

#### 3.1 Introduction

This thesis presents a modified version of PSO called selection enabled PSO (SEPSO) which involves the selection of better particles. The particles are arranged in ascending order of their objective function values and better particles are ‘selected’ or retained in the swarm, while the worse particles are eliminated. Though there is not much reduction in number of iterations in case of a selection enabled PSO (SEPSO), yet the time taken for convergence plummets. The computational results corroborate the fact that removing the worse particles do not affect convergence. Trapped particles with worse values of objective function, are always an overhead and it is better to do away with them in order to reduce the time taken up to convergence.

#### 3.2 The SEPSO model

The velocity update equation and the position update equation used are same as that of the conventional PSO. The difference lies in the algorithm for the SEPSO. The selection process is invoked whenever the iteration number is equal to the sorting frequency ‘fs’. At this point the swarm size is reduced using reduction factor. The reduction factor ‘r’ is an integer which, like sorting frequency, is inputted from the user and determines the new swarm size as follows,

$$\text{New Swarm Size} = \frac{\text{Old Swarm Size}}{r} \rightarrow 3$$

Various values of ‘r’ and ‘fs’ have been tried and the results corresponding to the best values have only been reported in this paper.

### 3.3 Additional parameters for SEPSO

As mentioned in the previous section, SEPSO has two additional parameters:

#### 3.3.1 Reduction factor (r)

The reduction factor is a user inputted value to decide the swarm size, as per equation 3, whenever the selection process is applied.

#### 3.3.2 Sorting frequency (fs)

A number of experiments have to be performed to determine the best combination of sorting frequency 'fs' and reduction factor 'r' to achieve maximum reduction in the function evaluations. Choosing a sorting frequency lower than a particular iteration number would be tantamount to calling the selection process before the exploration stage is over. On the other hand, a higher value leads to unnecessary function evaluations increasing computational time.

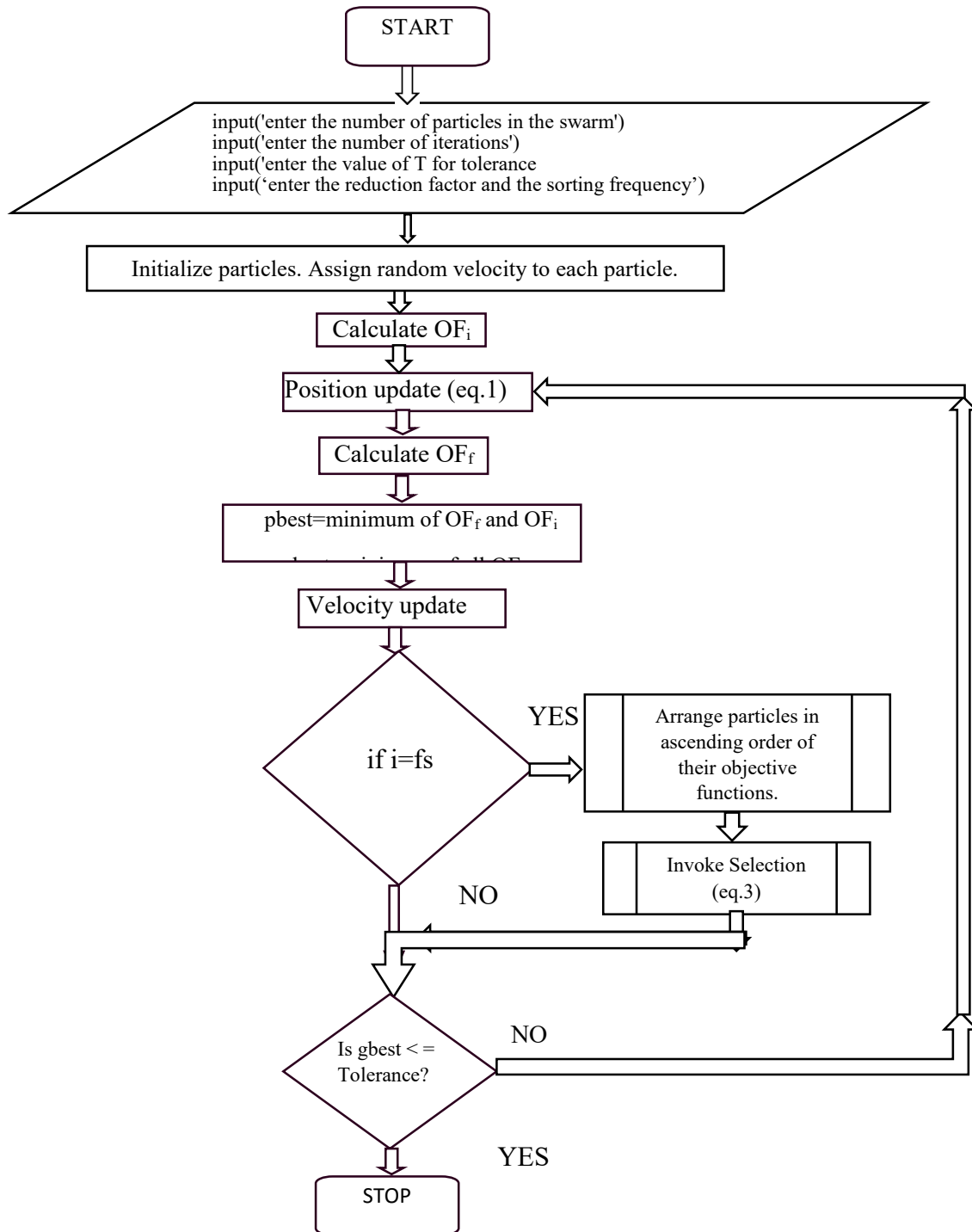
### 3.4 Algorithm

The algorithm for SEPSO is presented below.

- i) Initialize swarm and enter maximum number of iterations. Assign positions and velocities to every particle. Input reduction factor 'r' and sorting frequency 'fs'.
- ii) Input the value of 'T', the negative integer which appears as exponent in the convergence criterion:  $\text{tolerance} = 10^T$ .
- iii) Calculate the objective function value and call it  $OF_i$ .
- iv) Update positions for all the particles, using eq. 2), and calculate their objective function value and call it  $OF_f$ .
- v) Compare  $OF_f$  and  $OF_i$  for each particle. The position of the particle corresponding to smaller of the two is taken as pbest.
- vi) Find out the gbest by considering the particle with the position yielding the least value of the objective function.
- vii) Update velocity for all the particles using eq. 1).
- viii) Check if iteration number 'i' equals 'fs'. If it does, arrange the particles in the ascending order of their objective function value,  $OF_f$ .
- ix) Invoke selection and determine new swarm size of better particles using eq. 3.
- x) Check whether the objective function value for the gbest is less than the tolerance specified. If it is less than the tolerance specified, then go to xi) else go to iv)

- xi) Exit and display the gbest's objective function value, the position of the gbest particle and number of function evaluations along with positions and velocities for all particles in the swarm.

### 3.5 Flowchart



**Fig.2 Flowchart for SEPSO**



### **3.6 Advantage of SEPSO**

Several variants of conventional PSO suffer from convergence at local optima. Local version of conventional PSO does reliably converge at the global minima but its reliability is overshadowed by slow convergence. In cases of no convergence for both the local and the global version, there is occurrence of particles at the local minima. Even when the exploitation stage is imminent some particles appear to localize around a point and are inept at moving away from its vicinity. These particles can be removed from the swarm in SEPSO to make computation faster. Selection enabled PSO takes lesser number of function evaluations to converge as the particles not showing much improvement, after suitable number of iterations, are removed during selection and better particles are retained.

## Chapter-4

### Comparison of SEPSO with PSO

#### Introduction

In this chapter, four mathematical benchmark functions in two dimensions are used to compare the performance of SEPSO and conventional PSO. The parameters for both the algorithms are kept fixed to find the number of function evaluations and the number of iterations up to convergence. The reported observations for SEPSO are for those values of sorting frequency 'fs' and reduction factor 'r' for which best results were obtained.

#### 4.1 Benchmark functions

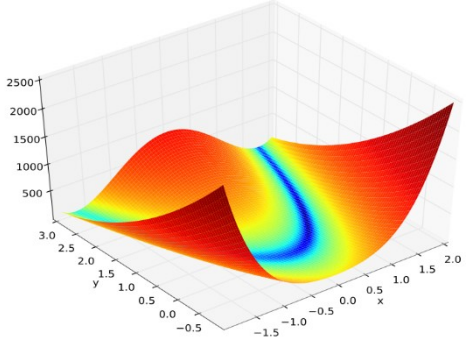
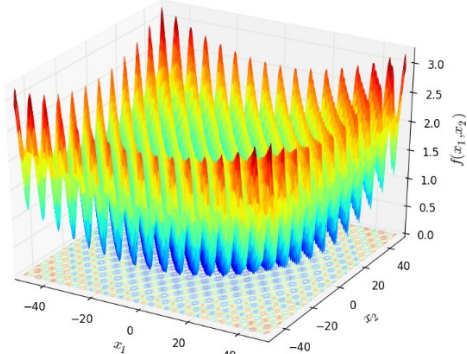
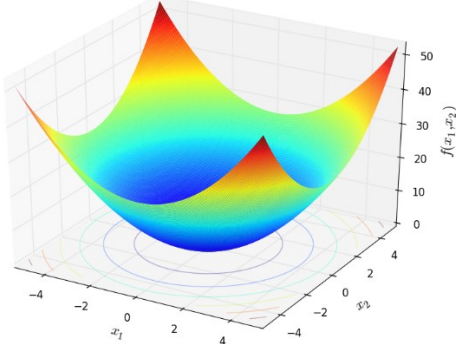
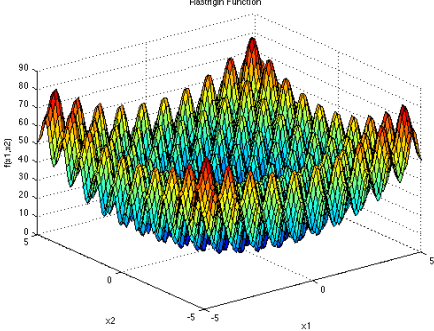
Table 1 below enlists, in detail, the benchmark functions used for evaluating the performance of SEPSO and PSO.

**Table 1. 2-D Benchmark functions**

S.No.	Function	Description	Search range	Minimum occurs at	Function value at minimum
1.	Rosenbrock	$f(x_1, x_2) = 100(x_2 - x_1^2) + (x_1 - 1)^2$	$x_i \in [-2.048, 2.048]$ $i=1,2$	$x^* = (1,1)$	$f(x^*)=0$
2.	Griewank	$f(x_1, x_2) = 1 + (1/4000) \sum_{i=1}^2 x_i^2 - \prod_{i=1}^2 \cos(x_i / \sqrt{i})$	$x_i \in [-600, 600]$ $i=1,2$	$x^* = (0,0)$	$f(x^*)=0$
3.	Sphere	$f(x_1, x_2) = \sum_{i=1}^2 x_i^2$	$x_i \in [-5.12, 5.12]$ $i=1,2$	$x^* = (0,0)$	$f(x^*)=0$
4.	Rastrigin	$f(x_1, x_2) = 20 + \sum_{i=1}^2 [x_i^2 - 10 \cos(2\pi x_i)]$	$x_i \in [-5.12, 5.12]$ $i=1,2$	$x^* = (0,0)$	$f(x^*)=0$

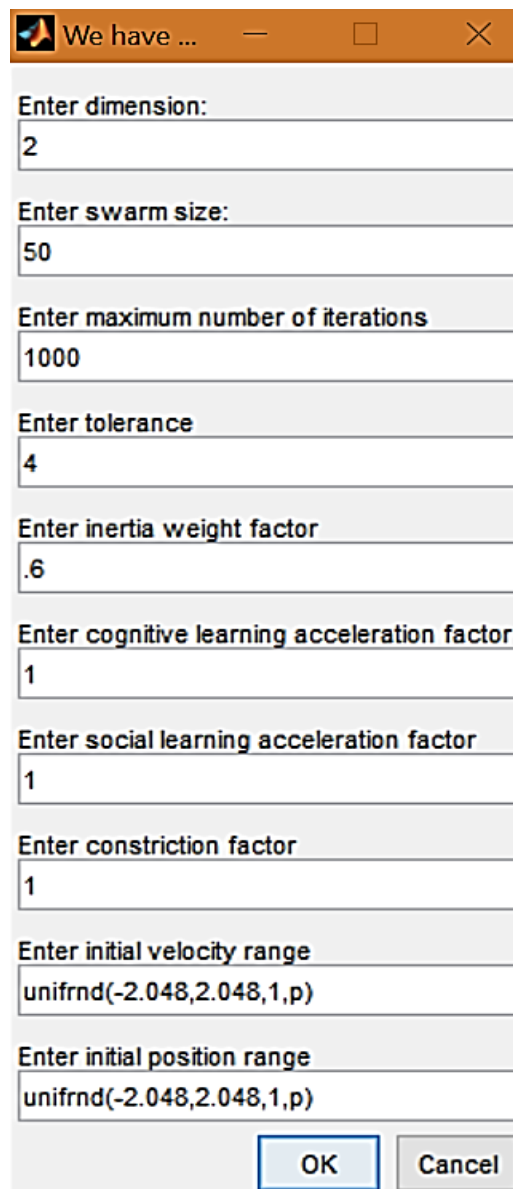
The plots of the above 2 D benchmark functions used are shown in table 2. It can be seen that while Rosenbrock and Sphere have only one minimum, Rastrigin and Griewank have several local minima and one global minimum which makes finding the global minimum tough. The sphere function is the simplest amongst all four and can be used to determine whether the algorithm designed is working properly i.e. there are no bugs in the computer program and/or the algorithm is properly designed.

**Table 2. Plots Of 2-D Benchmark Functions**

S.No.	Function	Plot
1.	Rosenbrock	
2.	Griewank	
3.	Sphere	
4.	Rastrigin	

## 4.2 Computational Results

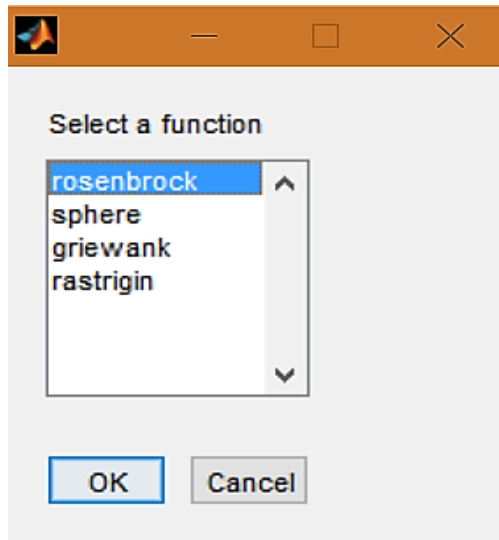
The performance of SEPSO is compared with that of conventional PSO with respect to computational efficiency in terms of function evaluations. Both these algorithms were applied on Rosenbrock, Griewank, Sphere, and Rastrigin functions to minimize them. The error in objective function's value for all cases is taken to be  $10^{-4}$ . For all functions, in case of both SEPSO and conventional PSO: the parameters are  $w=0.6$ ,  $r_p=1$  and  $r_g=1$ . The MATLAB program written in this thesis for testing PSO on four benchmark functions has been designed to enable the user to input parameters through a graphical user interface, as shown in fig 3.



Parameter	Value
Enter dimension:	2
Enter swarm size:	50
Enter maximum number of iterations	1000
Enter tolerance	4
Enter inertia weight factor	.6
Enter cognitive learning acceleration factor	1
Enter social learning acceleration factor	1
Enter constriction factor	1
Enter initial velocity range	unifrnd(-2.048,2.048,1,p)
Enter initial position range	unifrnd(-2.048,2.048,1,p)

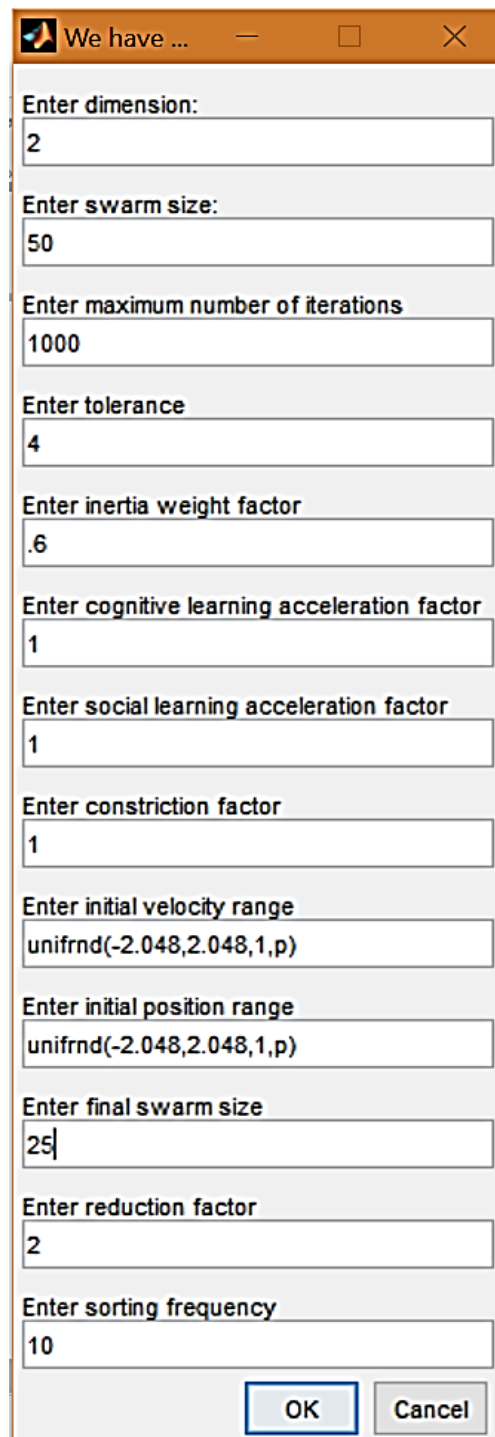
Fig 3. Parameters' window for entering PSO parameters

Any of the four benchmark functions can be selected using the window shown in figure 4. Figure 4 shows the window which appears after clicking on the OK button of the window in figure 3.



**Fig 4. Window for selecting the benchmark function**

A separate program for testing SEPSO algorithm is written using MATLAB. The parameters' window in this case contains all the parameters of SEPSO as shown below in figure 5. Rest of the interface is same as that for PSO.



Parameter	Value
Enter dimension:	2
Enter swarm size:	50
Enter maximum number of iterations	1000
Enter tolerance	4
Enter inertia weight factor	.6
Enter cognitive learning acceleration factor	1
Enter social learning acceleration factor	1
Enter constriction factor	1
Enter initial velocity range	unifrnd(-2.048,2.048,1,p)
Enter initial position range	unifrnd(-2.048,2.048,1,p)
Enter final swarm size	25
Enter reduction factor	2
Enter sorting frequency	10

**Fig 5. Parameters' window for entering SEPSO parameters**

#### 4.2.1 Rosenbrock function

Table 3 shows the number of iterations and function evaluations up to convergence for swarm sizes 50, 80, 100, 120, 140 and 180 when conventional PSO is applied to minimize Rosenbrock function.

**Table 3. Results of conventional PSO  
(Rosenbrock)**

S.No.	Swarm size	Function evaluations	Iterations	Error
1.	50	1500	30	9.77e-5
2.	80	2320	29	5.83e-5
3.	100	2900	29	1.55e-5
4.	120	4320	36	5.79e-5
5.	140	4060	29	4.56e-5
6.	180	4860	27	8.87e-5

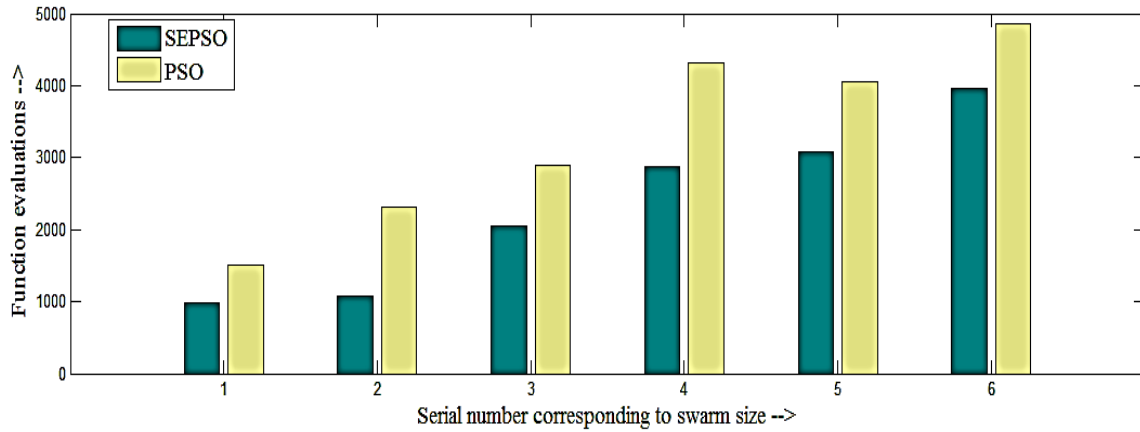
The proposed algorithm, SEPSO, is applied to Rosebrock's function and the results are presented in table 4.

**Table 4. Results of SEPSO (r=2)  
(Rosenbrock)**

Serial Number	Swarm size	Function evaluations (FEV)	% Reduction in FEV	Sorting frequency	Iterations	Error
1.	50	975	35	16	23	5.56e-5
2.	80	1075	53.66	16	27	1.17e-5
3.	100	2050	29.31	16	25	5.55e-5
4.	120	2880	33.33	18	30	2.29e-5
5.	140	3080	24.14	19	25	4.81e-5
6.	180	3960	18.52	19	25	6.55e-5

On comparing with table 3, it can be seen that there has been a significant reduction in the number of function evaluations in case of SEPSO.

Figure 6 depicts comparison between SEPSO and conventional PSO for Rosenbrock's function.



**Fig 6. Comparison of SEPSO and conventional PSO for Rosenbrock's function.**

It is clearly brought out by Fig. 6 that function evaluations are less in case of SEPSO for all the swarm sizes considered.

#### 4.2.2 Griewank function

Proceeding in the same manner as Rosenbrock, SEPSO and conventional PSO are compared by using them to minimize Griewank function. Table 5 shows the results of conventional PSO.

**Table 5. Results of conventional PSO  
(Griewank)**

S.No.	Swarm size	Function evaluations (FEV)	Iterations	Error
1.	80	32000	400	5.48e-5
2.	90	35280	392	7.81e-5
3.	100	19600	196	5.59e-5
4.	120	21360	178	9.28e-5

From table 5, above, it can be seen that conventional PSO takes larger number of function evaluations, as compared to that for Rosenbrock, to find the point corresponding to global minimum. Also swarm size needs to be larger in case of Griewank for convergence. This function does not for swarm size smaller than 80. These differences can be attributed to the fact that Griewank function has a large number of local minima and minimizing the function, therefore, becomes more difficult.



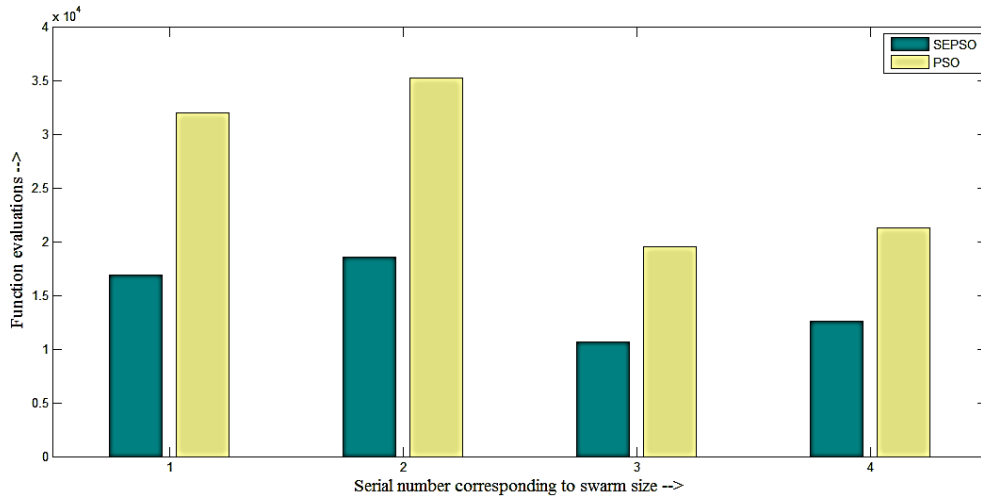
Table 6 shows the results for SEPSO.

**Table 6. Results of SEPSO (for  $r=2$ )**  
(Griewank)

S.No.	Swarm size	Function evaluations (FEV)	% Reduction in FEV	Sorting frequency	Iterations	Error
1.	80	16880	47.25	26	396	2.29e-5
2.	90	18630	47.19	26	388	4.81e-5
3.	100	10650	45.66	26	187	6.55e-5
4.	120	12600	41.01	30	180	4.92e-5

From table 6, it is observed that SEPSO takes less number of function evaluations than conventional PSO to converge in the case of Griewank function.

The comparison is of conventional PSO and SEPSO is also shown in Fig. 7.



**Fig 7. Comparison of SEPSO and conventional PSO for Griewank's function.**

Figure 7 shows that function evaluations decrease when SEPSO is applied to Griewank's function.

#### 4.2.3 Sphere function

Now, sphere function is minimized by using SEPSO and conventional PSO. Table 7 and 8 below show the results pertaining to PSO and SEPSO respectively.

**Table 7. Results of conventional PSO (Sphere)**

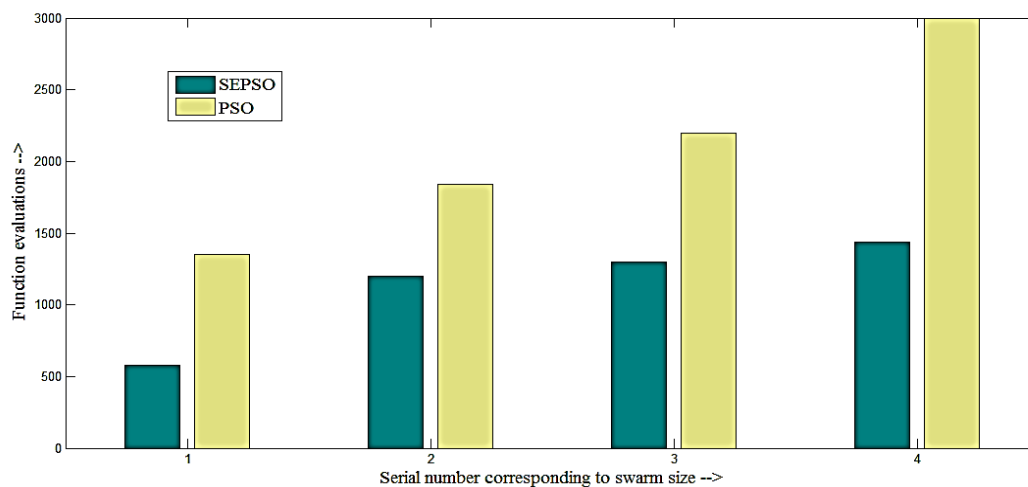
S.No.	Swarm size	Function evaluations(FEV)	Iterations	Error
1.	50	1350	27	3.62e-5
2.	80	1840	23	2.87e-5
3.	100	2200	22	2.86e-5
4.	120	3000	25	9.62e-5

**Table 8. Results of SEPSO  
(Sphere) for r=2**

S.No.	Swarm size	Function evaluations (FEV)	% Reduction in FEV	Sorting frequency	Iterations	Error
1.	50	575	57.41	6	17	6.63e-5
2.	80	1200	34.78	5	25	7.50e-5
3.	100	1300	40.91	5	21	6.03e-5
4.	120	1440	52.00	5	19	2.22e-5

From table 7, it is clear that conventional PSO takes lesser number of function evaluations for sphere in comparison to Griewank. Also, smaller swarm size can be used to minimize sphere function. This function converges when the swarm size is 50, whereas Griewank function converges for larger swarm size.

From table 8, it is observed that number of function evaluations taken by SEPSO is smaller than that of conventional PSO as in the case of Rosenbrock and Griewank function. Figure 8 compares SEPSO and conventional PSO in terms of the number of function evaluations when applied to sphere function.

**Fig 8. Comparison of SEPSO and conventional PSO for Sphere function.**

#### 4.2.4 Rastrigin's function

Two dimensional variant of Rastrigin's function is used in this work, to compare the performance of SEPSON and conventional PSO. Table 9 and table 10 compare the performance of PSO and SEPSON respectively when applied to Rastrigin's function.

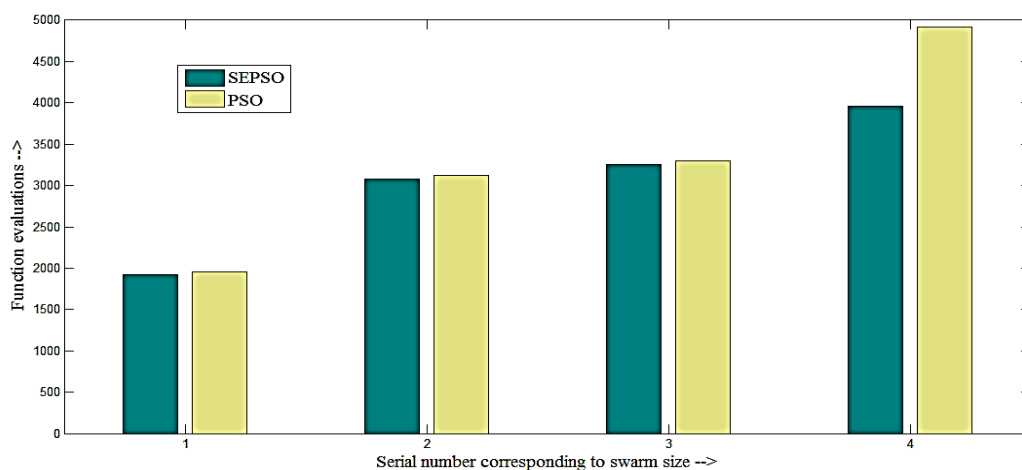
**Table 9. Results of conventional PSO**  
(Rastrigin)

S.No.	Swarm size	Function evaluations(FEV)	Iterations	Error
1.	50	1950	39	8.87e-5
2.	80	3120	39	8.97e-5
3.	100	3300	33	8.63e-5
4.	120	4920	41	9.28e-5

**Table 10. Results of SEPSON (r=2)**  
(Rastrigin)

S.No.	Swarm size	Function evaluations (FEV)	% Reduction in FEV	Sorting frequency	Iterations	Error
1.	50	1925	1.28	35	42	8.15e-5
2.	80	3080	1.28	36	41	9.06e-5
3.	100	3250	1.52	31	34	1.27e-5
4.	120	3960	19.51	31	35	9.13e-5

It can be seen from table 10 that for swarm sizes corresponding to S.No. 1,2, and 3 i.e. 50,80, and 100 respectively, very little reduction in function evaluations (FEV) is obtained for but for the swarm size of 120, a significant reduction of 19.51% is obtained. Fig. 9 shows the comparison between SEPSON and conventional PSO for Rastrigin's function.



**Fig 9. SEPSON compared with PSO for Rastrigin's function.**

### 4.3 Discussion

The selection enabled PSO is tested and compared with conventional PSO on four two dimensional mathematical benchmark functions. Reduction in number of function evaluations is observed for all functions.

Number of function evaluations has been taken as a measure of computational efficiency because it is independent of processor's clock speed and numerous ways in which an algorithm can be implemented.

Therefore, it is concluded that if sorting frequency 'fs' and reduction factor 'r' are chosen suitably, the number of function evaluations can be reduced while maintaining the same allowable error in the objective function value. Hence, SEPSO is more computationally efficient than conventional PSO and as accurate.

## Chapter-5

### Load Flow studies using SEPSO

#### 5.1 Problem formulation

Load flow solution gives the nodal voltages and phase angles at all the buses and power flows through interconnected power channels. Real and reactive power for k<sup>th</sup> bus is calculated [25] as,

$$P_k = \sum_{j=1}^N |V_k| |V_j| (G_{kj} \cos(\theta_k - \theta_j) + B_{kj} \sin(\theta_k - \theta_j)). \quad \rightarrow 4$$

$$Q_k = \sum_{j=1}^N |V_k| |V_j| (G_{kj} \sin(\theta_k - \theta_j) - B_{kj} \cos(\theta_k - \theta_j)) \quad \rightarrow 5$$

Where,

N = Number of buses.

$V_k$  = Voltage at k<sup>th</sup> bus.

$V_j$  = Voltage at j<sup>th</sup> bus.

$G_{kj}$  = Conductance between bus k and j.

$\theta_k, \theta_j$  = Angle of bus k and j respectively.

In load flow, the real and reactive power mismatches are minimized for every bus using the following equations,

$$\Delta P_k = P_k^{\text{calc}} - P_k^{\text{specified}} \quad \rightarrow 6$$

$$\Delta Q_k = Q_k^{\text{calc}} - Q_k^{\text{specified}} \quad \rightarrow 7$$

K=1,2,3, ..., N ; N=Total number of buses.

Where,

$\Delta P_k$  = Real power mismatch corresponding to k<sup>th</sup> bus

$\Delta Q_k$  = Reactive power mismatch corresponding to k<sup>th</sup> bus.

$P_k^{\text{calc}}$  = Calculated Real Power for the k<sup>th</sup> bus.

$P_k^{\text{specified}}$  = Specified Real power for the  $k^{\text{th}}$  bus.

$Q_k^{\text{calc}}$  = Calculated reactive power for the  $k^{\text{th}}$  bus.

$Q_k^{\text{specified}}$  = Specified Reactive power for the  $k^{\text{th}}$  bus.

Therefore, load flow problem is formulated as an optimization problem and the objective is to Minimize

$$f = \sum_{k=1}^N (\Delta P_k)^2 + \sum_{k=1}^N (\Delta Q_k)^2 \quad \rightarrow 8$$

In this SEPSO implementation, various types of buses are treated as follows,

**i) Load buses (PQ).**

The real and reactive power mismatches are calculated using eq. 6) and 7) respectively.

**ii) Generator bus/voltage regulated bus (PV buses)**

For these buses real power mismatch is calculated using eq. 6), the **reactive power mismatch** is equated to zero whenever the calculated reactive power is within limits. If the calculated reactive power violates the reactive power limits, the reactive power mismatch is calculated as the difference of the calculated value and the reactive power limit, upper or lower, whichever is violated. Mathematically, we have,

$$\Delta Q_i = \min[|Q_i^{\text{calc}} - Q_i^{\text{max}}|, |Q_i^{\text{min}} - Q_i^{\text{calc}}|] \quad \rightarrow 9$$

Where,

$Q_i^{\text{max}}$  = Upper reactive power generation limit.

$Q_i^{\text{min}}$  = Lower reactive power generation limit.

## 5.2 Computational procedure

This section presents the algorithm for the load flow using the proposed SEPSO algorithm.

**i)** Initialize m number of particles each with 2 rows and 5/14 columns. (5 columns for a 5 bus system and 14 for a 14 bus system.). First row has all the voltage magnitudes corresponding to buses and second row has all the corresponding angles. Initially the values of all voltage magnitudes is kept at 1 p.u. and the corresponding angles are kept at 0 radian. Initial velocity for voltage magnitude updation is determined by `unifrnd(-.1,.1)` in MATLAB. Initial velocity

for phase angle update is determined by `unifrnd(-.5,.5)` in MATLAB. The initial values for both velocities and positions is same for both IEEE 5 bus and IEEE 14 bus system. The values of sorting frequency 'fs', reduction factor 'r', tolerance 'T' and maximum number of iterations 'iter' are inputted.

**ii)** The initial value of objective function (OF) before position update is found and it is called  $OF_i$ .

**iii)** Update the position for all the particles as per the following equation,

$$X_m^{i+1} = X_m^i + V_m^{i+1} \quad \rightarrow 10$$

Where,

$i = i^{\text{th}}$  iteration,

$m = m^{\text{th}}$  particle,

**iv)** The final value of OF after position update is found and it is called  $OF_f$ .

**v)** Smaller of  $OF_f$  and  $OF_i$  is considered for the personal best position. (Pbest)

**vi)** Minimum of all  $OF_f$ 's of all particles is found for global best. (gbest)

**vii)** Update the velocity for all the particles using the following equation,

$$V_m^{i+1} = w \times V_m^i + r_p \times \text{rand}() \times (\text{pbest}_m^i - X_m^i) + r_g \times \text{rand}() \times (\text{gbest}^i - X_m^i) \quad \rightarrow 11$$

Where,

**x)** Exit. Display all voltages and power injections.

$\text{pbest}_m^i$  = best position of particle m till  $i^{\text{th}}$  iteration

$\text{gbest}^i$  = global best for the entire swarm up to iteration i

w=inertia weight factor.

$r_p, r_g$  = acceleration constants.

**viii)** Is the iteration number equal to sorting frequency? If NO, go to ix). Else, Sort the particles in ascending order of their objective function and remove the worst particles in the swarm as dictated by the reduction factor.

**ix)** Is the  $OF_f$  for the particle corresponding to the global best less than or equal to the specified tolerance? If NO, go to iii). Else go to x).

### 5.3 Computational results

Load flow has been performed for IEEE-5 and 14-bus systems using conventional PSO and selection enabled PSO (SEPSO). The parameters for the conventional PSO and SEPSO were fixed as;  $w=0.7$ ,  $r_p=1$  and  $r_g=1$ .

#### 5.3.1 IEEE 5 bus system

##### (i) Conventional PSO

Table 11 below shows the results of load flow by conventional PSO for IEEE-5 bus system. The accuracy for convergence for the value of function has been taken as  $1 \times 10^{-7}$ . In column 3 of table 11, the accuracy of final function value at convergence has been mentioned.

**Table 11: Computational Effort for Load flow using conventional PSO  
(IEEE 5 bus system)**

S. No. (1)	swarm size (2)	Accuracy (3)	Function evaluations. (4)	Iterations (5)	Time (ms) (6)
1.	30	-10	3780	125	815
2.	32	-10	5024	156	959
3.	64	-10	7232	112	1367
4.	128	-11	12672	98	2273
5.	256	-11	24320	94	4306
6.	512	-10	40960	79	6891
7.	1024	-11	86016	83	16743

From this table it is clear that the computational time increases with the number of particles. In this table, column 4,5 and 6 representing function evaluations, iterations and time are all measures of computational effort. The time taken to solve a problem depends on the processor speed of the computer, which goes on changing with the advancement of technology. Further, various researchers may use different processors for their research work. For these reasons, the time in seconds of ms cannot be taken as standard measure for computational effort. Computational effort for each iteration depends on the size of the problem and, therefore, doesn't prove to be a consistent measure of computational effort. However, number of function evaluations can prove to be the most consistent parameter for measuring computational effort



and would reflect as a measure of computational efficiency of a given algorithm. For these reasons number of function evaluations has been taken as index for measuring/comparing the computational effort in this thesis report.

**(ii) Selection Enabled PSO (SEPSO)**

Load flow has been performed for IEEE 5 bus system using SEPSO. In order to obtain the best sorting frequency 'fs' and reduction factor 'r', many load flows are performed for IEEE-5 bus system by varying sorting frequency 'fs' and reduction factor 'r'.

The initial swarm size of 1024, 512, 256, 128 and 64 is considered and sorting frequency is varied from 1 to 10 for reduction factors of 2, 4, 6 and 8. The sorting frequency is also varied from 50 to 500 in steps of 50 for reduction factor of 2. Table 12 shows the results for which minimum number of function evaluations are obtained. For IEEE-5 bus system. It can be observed from table 12 that the minimum number of function evaluations are obtained for fs=2 and r=2. This is shown at s.no. 3 of table 12 and is highlighted.

**Table 12: Computational Effort for Load flow using SEPSO.  
(IEEE 5 bus system)**

No.	a.	b.	Fs	r	Accuracy	i	Fev	t(ms)
1	32	32	-	-	-10	156	5024	959
2	1024	512	2	2	-10	88	46608	8086
3	512	256	2	2	-11	86	23127	3934
4	256	128	2	2	-10	95	12544	2432
5	128	64	1	2	-11	124	4224	914
6	64	32	2	2	-10	121	4000	858
7	128	32	1	4	-10	157	4992	634
a=initial swarm size ; b=final swarm size; i=number of iterations; fev=number of function evaluations ; t=time								

Table 13 shows the load flow results as obtained using SEPSO for IEEE 5 bus system.

**Table 13: Results of Load flow using SEPSO; fs=2 and r=2; (IEEE 5 bus system)**

Bus	V	$\delta$	$P_{Gk}$	$Q_{Gk}$	$P_{Dk}$	$Q_{Dk}$	$P_k$	$Q_k$
1	1.02	0	0.651	0.326	0.651	0.329	0.651	0.329
2	0.9552	-3.9453	0	0	0.6	0.3	-0.6	-0.3
3	1.04	2.0676	1.0	0.480	0	0	1.0	0.478
4	0.9237	-8.0075	0	0	0.4	0.1	-0.4	-0.1
5	0.9934	-2.0771	0	0	0.6	0.2	-0.6	-0.2

V= Voltage Magnitude (p.u.);  $\delta$ = Phase angle (in degrees);  
 $P_{Gk}$  = Generated Real Power(p.u.) at  $k^{th}$  bus;  $Q_{Gk}$  = Generated reactive power(p.u.) at  $k^{th}$  bus;  
 $P_{Dk}$  = Real Power Demand(p.u.) at  $k^{th}$  bus;  $Q_{Dk}$  = Reactive Power Demand(p.u.) at  $k^{th}$  bus;  
 $P_k$  = Calculated Real Power(p.u.) at  $k^{th}$  bus;  $Q_k$  = Calculated Reactive Power(p.u.) at  $k^{th}$  bus, Base values are 1kV and 100MVA

### 5.3.2 IEEE 14 bus system

#### (i) Conventional PSO

Table 14 shows the computational effort in terms of function evaluation, iterations and time taken of load flow by conventional PSO for IEEE-14 bus system.

**Table 14: Computational Effort for Load flow using conventional PSO (IEEE 14 bus system)**

S.No.	swarm size	Accuracy	Function evaluations.	Iterations	Time (in s)
1.	256	-7	5.13 Lakhs	5214	447.7
2.	512	-10	52 Lakhs	5080	1938

#### (ii) Selection Enabled PSO (SEPSO)

Table 15 shows the results of load flow for IEEE 14 bus system using SEPSO. For various combinations of 'fs' and 'r'. Here 'fs' is the number of iterations after which selection is to be performed and 'r' is the factor by which the swarm size is to be reduced.

**Table 15: Computational Effort for Load flow using SEPSO  
(IEEE 14 bus system).**

<b>S No.</b> <b>(1)</b>	<b>a</b> <b>(2)</b>	<b>b</b> <b>(3)</b>	<b>fs</b> <b>(4)</b>	<b>R</b> <b>(5)</b>	<b>Accuracy</b> <b>(6)</b>	<b>I</b> <b>(7)</b>	<b>Fev</b> <b>(8)</b>	<b>t(s)</b> <b>(9)</b>
1	512	512	-	-	-7	5080	52 Lakhs	1938
2	512	256	200	2	-7	1057	6.4 Lakhs	254.5
3	256	256	-	-	-7	5214	5.13 Lakhs	447.7
4	256	128	200	2	-7	1039	3.17 Lakhs	131.3
5	512	128	250	2	-6	1416	6 Lakhs	267
a=initial swarm size ; b=final swarm size; i=number of iterations; fev=number of function evaluations ; t=time								

In column 4 and 5 of row 1 and 3 of this table (-) indicates that selection has not been used. This means these are the results corresponding to conventional PSO. It is observed that minimum number of function evaluations are obtained for a sorting frequency of 200 and reduction factor 'r' of 2. This is shown at serial no. 4 of table 15 and is highlighted.

Table 16 shows the results as obtained using SEPSO for load flow studies on IEEE 14 bus system.

**Table 16: Results of load flow using SEPSO.  
(IEEE 14 bus system)**

Bus	V	$\delta$	$P_{Gk}$ (MW)	$Q_{Gk}$ (MVAR)	$P_{Dk}$ (MW)	$Q_{Dk}$ (MVAR)	$P_k$ (MW)	$Q_k$ (MVAR)
1	1.0600	0.0000	125.591	3.480	0	0	125.59	3.47
2	1.0450	-2.4590	70.000	15.532	21.70	12.700	48.30	2.83
3	1.0100	-9.1061	0.000	21.261	94.20	19.000	-94.20	2.26
4	1.0251	-5.8438	0.000	0.000	47.80	-3.900	-47.80	3.90
5	1.0262	-5.0419	0.000	0.000	7.600	1.600	-7.59	-1.60
6	1.0701	-8.9216	0.000	19.674	11.20	7.500	-11.20	12.17
7	1.0549	-3.9782	0.000	0.000	0.000	0.000	0.000	0.000
8	1.0900	2.1795	70.000	25.613	0.000	0.000	70.00	25.61
9	1.0415	-7.0081	0.000	0.000	29.50	16.600	-29.50	-16.60
10	1.0395	-7.6300	0.000	0.000	9.000	5.800	-8.98	-5.80
11	1.0515	-8.3860	0.000	0.000	3.500	1.800	-3.50	-1.80
12	1.0538	-9.6078	0.000	0.000	6.100	1.600	-6.10	-1.60
13	1.0486	-9.4870	0.000	0.000	13.50	5.800	-13.50	-5.80
14	1.0265	-9.1062	0.000	0.000	14.90	5.000	-14.91	-5.00
V= Voltage Magnitude (in p.u.); $\delta$ = Phase angle (in degrees); $P_{Gk}$ = Generated Real Power at $k^{th}$ bus; $Q_{Gk}$ = Generated reactive power at $k^{th}$ bus; $P_{Dk}$ = Real Power Demand at $k^{th}$ bus; $Q_{Dk}$ = Reactive Power Demand at $k^{th}$ bus; $P_k$ = Calculated Real Power at $k^{th}$ bus; $Q_k$ = Calculated Reactive Power at $k^{th}$ bus.								

## 5.4 Discussion

Load flow has been performed using SEPSO and conventional PSO for IEEE 5 and 14 bus systems. In case of SEPSO, each combination of the user's input comprising of the reduction factor 'r' and the sorting frequency 'fs' is executed thirty times in case of the IEEE 5 bus system and 10 times in case of the IEEE 14 bus system. The programs were written in MATLAB and executed on a PC with 4 GB RAM. From table 11 and 12, a comparative study for computational effort in terms of function evaluations has been carried out for IEEE 5 bus system and is shown in table 17 below.

**Table 17. Comparison of SEPSO with conventional PSO  
(IEEE 5 bus system).**

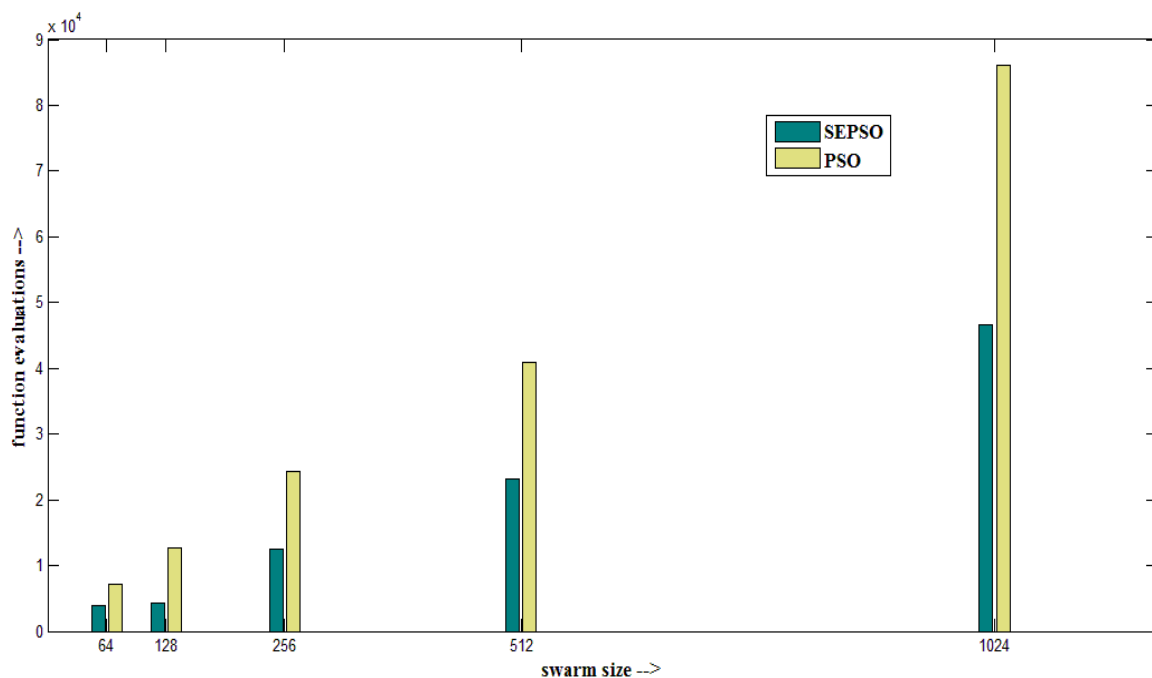
<b>S.No.</b>	<b>Swarm size</b>	<b>Function evaluations (conventional PSO)</b>	<b>Function evaluations (SEPSO)</b>	<b>% Saving</b>
1	1024	86016	46608	41.81
2	512	40960	23127	43.54
3	256	24320	12544	48.42
4	128	12672	4224	66.67
5	64	7232	4000	44.67

Similarly, from table 14 and 15, a comparative study for computational effort in terms of function evaluations has been carried out for IEEE 14 bus system and is shown in table 18 below.

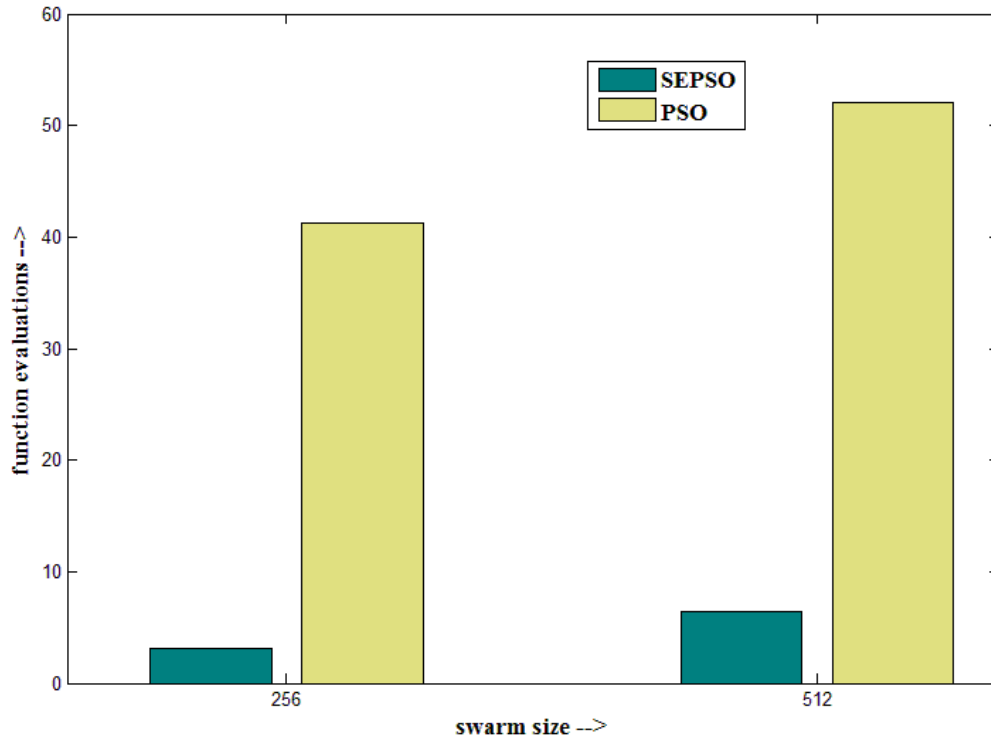
**Table 18. Comparison of SEPSO with conventional PSO  
(IEEE 14 bus system).**

S.No.	Swarm size	Function evaluations (conventional PSO)	Function evaluations (SEPSO)	% Saving
1	512	52 Lakhs	6.4 Lakhs	87.7
2	256	41.3 Lakhs	3.15 Lakhs	92.4

From table 17 and table 18, it is observed that SEPSO takes lesser function evaluations to converge for both IEEE 5 and 14 bus systems when compared to the conventional PSO. The last column in both tables show the percentage saving in terms of function evaluations obtained by SEPSO. The comparison of both the algorithms is also shown in Figure 10 and 11 for IEEE 5 and 14 bus system respectively.



**Fig.10 Computational Effort for conventional PSO and SEPSO  
(IEEE 5 bus system).**



**Fig.11 Computational Effort for conventional PSO and SEPSO (IEEE 14 bus system).**

It is observed from the above analysis that SEPSO takes less function evaluations to converge for both IEEE 5 and 14 bus systems when compared to the conventional PSO. The results obtained from proposed algorithm SEPSO are as accurate as NR method. The results of load flow using NR method for IEEE 5 and 14 bus system are shown in table 19 and 20 respectively.

## 5.5 Results of N-R algorithm

The values of voltages and their corresponding angles for all buses are obtained by load flow studies using Newton-Raphson's (NR) method. The results from the same are shown in table 19 and 20.

**Table 19: Results of load flow using Newton Raphson's (NR) method  
(IEEE 5 bus system)**

Bus	V	$\delta$	$P_{Gk}$	$Q_{Gk}$	$P_{Dk}$	$Q_{Dk}$	$P_k$	$Q_k$
1.	1.02	0	0.651	0.326	0	0	0.651	0.327
2.	0.9552	-3.9450	0	0	0.6	0.3	-0.6	-0.3
3	1.04	2.0675	1.0	0.480	0	0	1.0	0.480
4	0.9237	-8.0073	0	0	0.4	0.1	-0.4	-0.1
5	0.9934	-2.0770	0	0	0.6	0.2	-0.6	-0.2

V= Voltage Magnitude (p.u.) ;  $\delta$ = Phase angle (in degrees); All calculated and specified powers are at k<sup>th</sup> bus,  $P_{Gk}$  = Generated Real Power(p.u.);  $Q_{Gk}$  = Generated reactive power(p.u.);  $P_{Dk}$  = Real Power Demand(p.u.);  $Q_{Dk}$  = Reactive Power Demand(p.u.);  $P_k$  = Calculated Real Power(p.u.);  $Q_k$  = Calculated Reactive Power(p.u.) , Base values are 1kV and 100MVA



**Table 20: Results of load flow using NR method.  
(IEEE 14 bus system)**

Bus	V	$\delta$	$P_{Gk}$	$Q_{Gk}$	$P_{Dk}$	$Q_{Dk}$	$P_k$	$Q_k$
1	1.0600	0.0000	125.59	3.48	0	0	125.59	3.48
2	1.0450	-2.4591	70.00	15.53	21.70	12.700	48.30	2.83
3	1.0100	-9.1059	0.000	21.26	94.20	19.000	-94.20	2.26
4	1.0252	-5.8436	0.000	0.000	47.80	-3.900	-47.80	3.90
5	1.0262	-5.0418	0.000	0.000	7.600	1.600	-7.60	-1.60
6	1.0700	-8.9214	0.000	19.67	11.20	7.500	-11.20	12.17
7	1.0547	-3.9781	0.000	0.000	0.000	0.000	0.000	0.000
8	1.0900	2.1792	70.00	25.613	0.000	0.000	70.00	25.61
9	1.0417	-7.0080	0.000	0.000	29.50	16.600	-29.50	-16.60
10	1.0395	-7.6301	0.000	0.000	9.000	5.800	-9.00	-5.80
11	1.0514	-8.3862	0.000	0.000	3.500	1.800	-3.50	-1.80
12	1.0537	-9.6076	0.000	0.000	6.100	1.600	-6.10	-1.60
13	1.0486	-9.4869	0.000	0.000	13.50	5.800	-13.50	-5.80
14	1.0265	-9.1060	0.000	0.000	14.90	5.000	-14.90	-5.00
V= Voltage Magnitude (in p.u.) ; $\delta$ = Phase angle (in degrees); All calculated and specified powers are at k <sup>th</sup> bus, $P_{Gk}$ = Generated Real Power(MW); $Q_{Gk}$ = Generated reactive power(MVAR); $P_{Dk}$ = Real Power Demand(MW); $Q_{Dk}$ = Reactive Power Demand(MVAR); $P_k$ = Calculated Real Power(MW); $Q_k$ = Calculated Reactive Power(MVAR)								

The results of load flow using SEPSON are exactly same as that of NR method for IEEE 5 and 14 bus systems. This establishes the fact that the proposed method Selection enabled PSO (SEPSON) has been correctly designed and has converged to global minimum.

## Chapter-6

### Conclusions and Future Direction

#### 6.1 Conclusions

The following are the conclusions of this research work:

- i) A modified version of conventional PSO called Selection enabled PSO (SEPSO) has been developed which involves the selection of better particles. The strategy of retaining only the better particles reduces the time taken up to convergence. It has been observed from the tables of computational results that retaining half of the total number of particles leads to reliable convergence than any other value of reduction factor. The trapped particles create computational overhead which is undesirable. Such particles are needed only up to the exploration phase. During the exploitation phase, these ‘trapped’ particles become stationary.
- ii) Load flow is performed using SEPSO for IEEE 5 and 14 bus system successfully **for the first time**.
- iii) A strategy has been developed to treat reactive power mismatches for PV buses. Voltage buses have been kept fixed and therefore each particle in the swarm unambiguously moves towards the region where reactive power limits are not violated.
- iv) The results have been compared with conventional PSO and NR method.
- v) SEPSO is found to converge faster than conventional PSO and is as accurate as NR method.

Selection enabled PSO (SEPSO) is applied to the IEEE 5 bus system and the IEEE 14 bus system for load flow. Load flow results obtained from SEPSO are as accurate as that for the Newton-Raphson method. The results from Newton-Raphson’s method are used just to ensure that SEPSO has converged properly at the global minimum.

## **6.2 Future direction**

The process of trying several combinations of reduction factor ‘fs’ and sorting frequency ‘r’ can be replaced by making the selection adaptive using a well-designed criterion. Further SEPSO can be used along with other swarm intelligence algorithms to reduce the number of function evaluations in load flow problems. Those particles that reach near the global optimum and keep oscillating around it, can also be slowed down to reach the global optimum earlier without oscillating while the particles unable to come sufficiently close to global optimum, in a certain number of iterations, can be eliminated.

## APPENDIX

### *A: IEEE 5 bus system*

**Table I: Specified values of load(s), generation(s), voltage(s) and angle(s).**

Bus No.	Voltage Magnitude (p.u.)	Phase Angle (radians)	Generated Real Power(p.u.)	Generated Reactive Power(p.u.)	Real Power Demand(p.u.)	Reactive Power Demand(p.u.)
1.	1.02	0	-	-	-	-
2.	-	-	-	-	0.6	0.3
3.	1.04	-	1.0	-	-	-
4.	-	-	-	-	0.4	0.1
5.	-	-	-	-	0.6	0.2

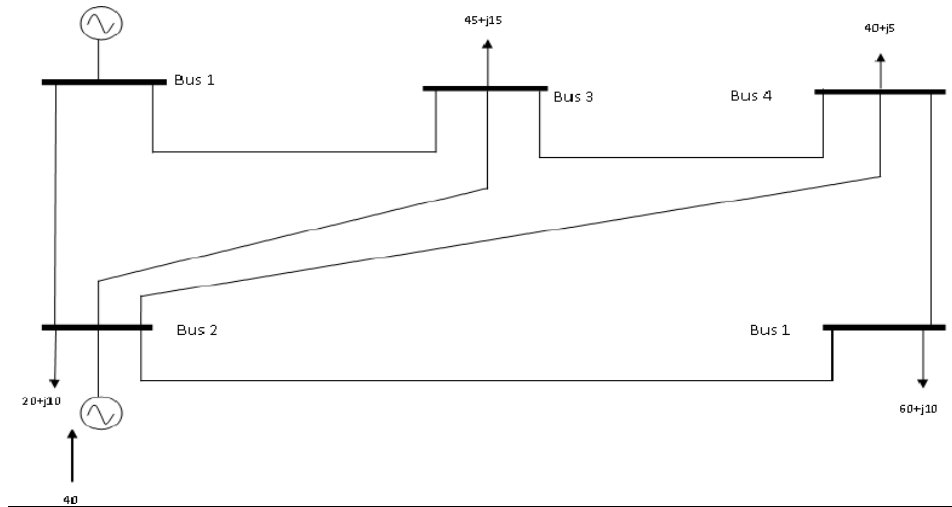


Fig.I. IEEE 5 Bus System

**Table II: Line reactance values and line resistance values.**

From bus	To bus	R(in p.u.)	X(in p.u.)
1	2	0.1	0.4
1	4	0.15	0.6
1	5	0.05	0.2
3	5	0.05	0.2
2	3	0.05	0.2
2	4	0.05	0.4

*B. IEEE 14 bus system*

**Table III: Line reactance values, line resistance values and line susceptance values.**

From bus	To bus	R(in p.u.)	X(in p.u.)	B/2(in p.u.)
1	2	0.01938	0.05917	0.0264
1	5	0.05403	0.22304	0.0246
2	3	0.04699	0.19797	0.0219
2	4	0.05811	0.17632	0.0170
2	5	0.05695	0.17388	0.0173
3	4	0.06701	0.17103	0.0064
4	5	0.01335	0.04211	0.0
4	7	0.0	0.20912	0.0
4	9	0.0	0.55618	0.0
5	6	0.0	0.25202	0.0
6	11	0.09498	0.19890	0.0
6	12	0.12291	0.25581	0.0
6	13	0.06615	0.13027	0.0
7	8	0.0	0.17615	0.0
7	9	0.0	0.11001	0.0
9	10	0.03181	0.08450	0.0
9	14	0.12711	0.27038	0.0
10	11	0.08205	0.19207	0.0
12	13	0.22092	0.19988	0.0
13	14	0.17093	0.34802	0.0

**Table IV: Specified values of load(s), generation(s), voltage(s) and angle(s).**

Bus No.	Voltage magnitude (p.u.)	Phase angle (radians)	Generated Real Power (MW)	Generated reactive Power(MVAR)	Real Power Demand (MW)	Reactive Power Demand (MVAR)
1	1.060	0	0	0	0	0
2	1.045	0	70	42.4	21.7	12.7
3	1.010	0	0	23.4	94.2	19.0
4	1.0	0	0	0	47.8	-3.9
5	1.0	0	0	0	7.6	1.6
6	1.0	0	0	12.2	11.2	7.5
7	1.070	0	0	0	0.0	0.0
8	1.0	0	70	17.4	0.0	0.0
9	1.090	0	0	0	29.5	16.6
10	1.0	0	0	0	9.0	5.8
11	1.0	0	0	0	3.5	1.8
12	1.0	0	0	0	6.1	1.6
13	1.0	0	0	0	13.5	5.8
14	1.0	0	0	0	14.9	5.0

**Table V: Reactive Power Limits**

Bus No.	Qmin (MVAR)	Qmax (MVAR)
2	-40	50
3	0	40
6	-6	24
8	-6	30

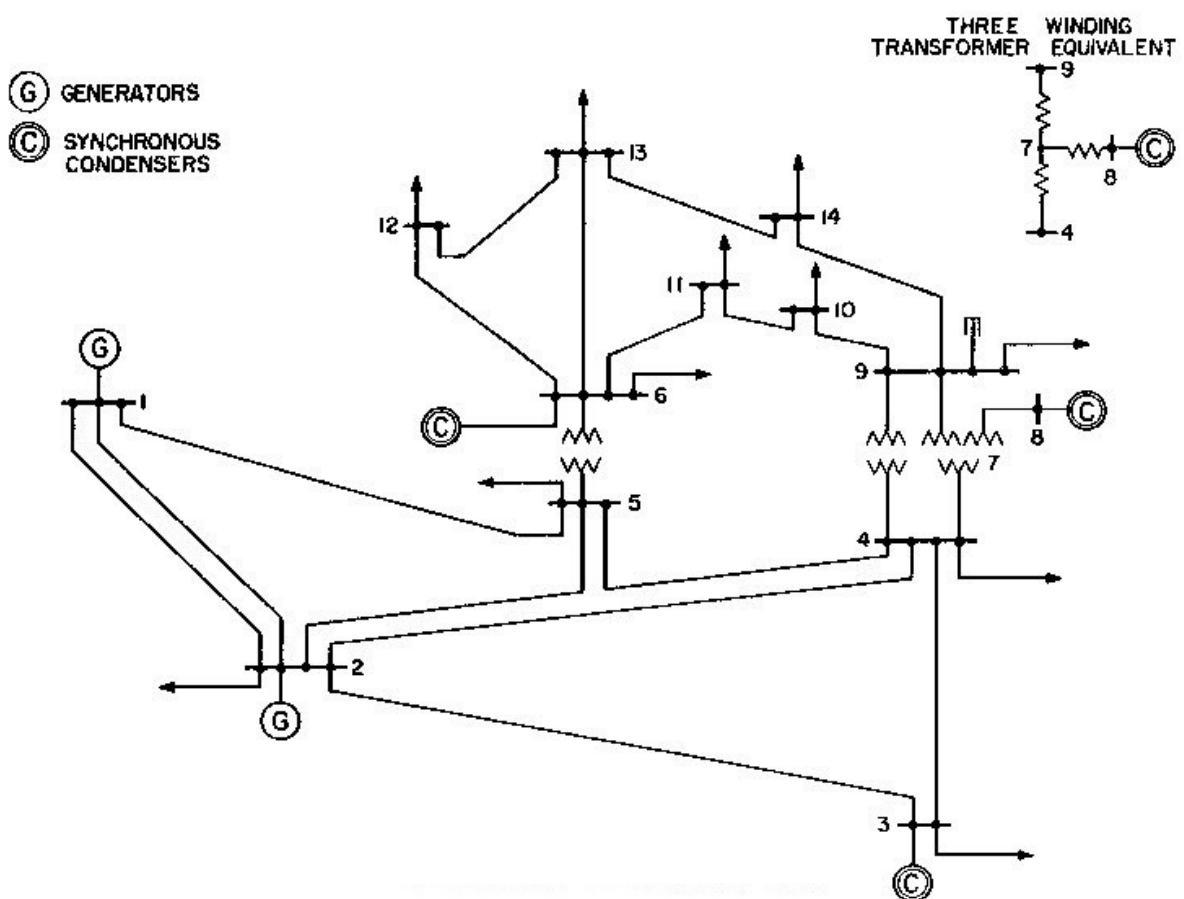


Fig.II. IEEE 14 Bus System



i) MATLAB code for load flow on IEEE 5 bus system using SEPSO.

Main program:

```
clear all
clc
nbus = 5; % IEEE-5
Y = ybusppg(nbus); % Calling ybusppg.m to get Y-Bus Matrix..
busd = busdatas(nbus); % Calling busdatas.. % Base
BMva=100;
MVA..
Pg = busd(:,5)/BMva; % generated real power
Qg = busd(:,6)/BMva; % generated reactive power.
Pl = busd(:,7)/BMva; % load real power
Ql = busd(:,8)/BMva; % load reactive power
Qlim1 = busd(:,9)/BMva;
Qlim2 = busd(:,10)/BMva;
P = Pg - Pl; % Pi = PGi - PLi..
Q = Qg - Ql; % Qi = QGi - QLi..
Psp = P; % P Specified..
Qsp = Q; % q specified
Qmin=Qlim1(3);
Qmax=Qlim2(3);
G = real(Y); % Conductance matrix..
B = imag(Y); % Susceptance matrix..

%-----PSO PARAMETERS INITIALIZATION -----%
particle=[];
mn=[];
fr1=[];
it=input('maximum no. of iterations');
p=input('enter initial no. of particles');
pf=input('final no. of particles'); % no of particle
fs=input('frequency for sorting');
r=input('enter r');
% rfv=input('enter rfv');
% rft=input('enter rft');
c12=[];
tic
particle(1)=p;
rp=1;
T=10;
fr=[];
count=1;
deltai=zeros(p,1);
zeta=0;
i2=0;
rg=1;
rf=1;
%f=[];
f=zeros(p,it);
fp=zeros(p,1);
thp=zeros(p,5);
thg=zeros(p,5);
vp=zeros(p,5);
vg=zeros(p,5);
rft=[];
rfv=[];
for j=1:p
    rft(j)=1;
    rfv(j)=1;
```

```

end
%v=[];
%th=[];
%vv=[];
%vth=[];
v=zeros(p,it,5);
th=zeros(p,it,5);
vth=zeros(p,it,5);
vv=zeros(p,it,5);
%vtemp=zeros(p,it,5);
%thtemp=zeros(p,it,5);
%vthtemp=zeros(p,it,5);
%vvtemp=zeros(p,it,5);
%ftemp=zeros(p,it);
a=.5;
b=-0.5;
vth(:,1,:)=a+(b-a)*rand(p,5); %initial velocity of theta vector%
a=-.1;
b=0.1;
vv(:,1,:)=a+(b-a)*rand(p,5); %initial velocity of voltage vector%
vth(:, :, 1)=0;
vv(:, :, 1)=0;
a=0.5;
b=-0.5;
th(:,1,:)=a+(b-a)*rand(p,5);
a=1.1;
b=0.9;
v(:,1,:)=a+(b-a)*rand(p,5);
v(:, :, 1)=1.02;
v(:, :, 3)=1.04;
th(:, :, 1)=0;
vg(:,1)=1.02;
vp(:,1)=1.02;
vg(:,3)=1.04;
vp(:,3)=1.04;
thp(:,1)=0;
thg(:,1)=0;
%-----initial value of objective function-----%
% Calculate P and Q
PVIND=zeros(p,1);
fev=0;
for j=1:p
    P = zeros(nbus,1);
    Q = zeros(nbus,1);
    MPS=zeros(p,1);
    MQS=zeros(p,1);

for i = 2:nbus
    for k = 1:nbus
        P(i) = P(i) + v(j,1,i)* v(j,1,k)*(G(i,k)*cos(th(j,1,i)-
th(j,1,k)) + B(i,k)*sin(th(j,1,i)-th(j,1,k)));
    end
end

for i = 2:nbus
    for k = 1:nbus
        Q(i) = Q(i) + v(j,1,i)* v(j,1,k)*(G(i,k)*sin(th(j,1,i)-
th(j,1,k)) - B(i,k)*cos(th(j,1,i)-th(j,1,k)));
    end
end

end

```

```

% real power mismatch
MP=P-Psp;
MPS=MP.^2;
%reactive power mismatch in third bus
Qsp(3)=Q(3);
if Q(3)<Qmin;
    Q(3)=Qmin;
    PVIND(j,1)=1;
else PVIND(j,1)=0;
end
    if Q(3)>Qmax;
        Q(3)=Qmax;
        PVIND(j,1)=1;
    else PVIND(j,1)=0;
    end

%reactive power mismatch
MQ=Q-Qsp;
MQ(3)=0;
MQS=MQ.^2;
%objective function value
f(j,1)=sum(MPS)+sum(MQS);
% fr(j,1)=f(j,1);
% fr(j,2)=j;
fev=fev+1;
end

%Initial personal best values
for i=1:p
    for k=2:5
        thp(i,k)=th(i,1,k);
    end
    for k=2:5
        vp(i,k)=v(i,1,k);
    end
end
%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
    for j=2:5
        thg(k,j)=th(gb,1,j);
    end
    for j=2:5
        vg(k,j)=v(gb,1,j);
    end
end
fgm = min(f(:,1));
Q3=zeros(p,it);
for i=1:it
    %for inertia weight W
    %wmax=.4;

```

```

    %wmin=.4;
    % w=wmax-((wmax-wmin)*i/it);
    %w =0.1+(rand()/2);
    %velocity update
    %position update
    w=.7;
for j=1:p
    for k=2:5

        vth(j,(i+1),k) = w*vth(j,i,k) + rp*rand()*(thp(j,k)-th(j,i,k)) +
rg*rand()*(thg(j,k)-th(j,i,k));
        % if vth(j,(i+1),k)<-0.1
        %     vth(j,(i+1),k)=-0.1;
        %end
        %if vth(j,(i+1),k)>0.1
        %     vth(j,(i+1),k)=0.1;
        %end
        th(j,(i+1),k) = th(j,i,k) + rft(j)*vth(j,(i+1),k);
    end

    for q=2:5
        vv(j,(i+1),q) = w*vv(j,i,q) + rp*rand()*(vp(j,q)-v(j,i,q)) +
rg*rand()*(vg(j,q)-v(j,i,q));

        v(j,(i+1),q) = v(j,i,q) + rfv(j)*vv(j,(i+1),q);
    end

    for q=3
        if PVIND(j,1)==0
            v(j,(i+1),q)=1.04;
        end
    end
end

end

%th(:,i,5)

%objective function value
for j=1:p
    P = zeros(nbus,1);
    Q = zeros(nbus,1);
    MPS=zeros(p,1);
    MQS=zeros(p,1);

    for m = 2:nbus
        for k = 1:nbus
            P(m) = P(m) + v(j,(i+1),m)*
v(j,(i+1),k)*(G(m,k)*cos(th(j,(i+1),m)-th(j,(i+1),k)) +
B(m,k)*sin(th(j,(i+1),m)-th(j,(i+1),k)));
        end
    end
    for m = 2:5
        for k = 1:nbus

```

```

        Q(m) = Q(m) + v(j, (i+1), m) *
v(j, (i+1), k) * (G(m, k) * sin(th(j, (i+1), m) - th(j, (i+1), k)) -
B(m, k) * cos(th(j, (i+1), m) - th(j, (i+1), k)));

        end
        end
    % real power mismatch
    MP=P-Psp;
    MPS=MP.^2;
    %reactive power mismatch in third bus
    Qsp(3)=Q(3);
    if Q(3)<Qmin;
        Q(3)=Qmin;
        PVIND(j,1)=1;
    else PVIND(j,1)=0;
    end
        if Q(3)>Qmax;
            Q(3)=Qmax;
            PVIND(j,1)=1;
        else PVIND(j,1)=0;
        end
        Q3(j,i)=Q(3);
    %reactive power mismatch
    MQ=Q-Qsp;
    MQ(3)=0;
    MQS=MQ.^2;
    %objective function value

    f(j, (i+1))=sum(MPS)+sum(MQS);
    fr(j,1)=f(j, i+1);
    fr(j,2)=j;
    fev=fev+1;

end

    %personal best values updatio
for j=1:p
    P = zeros(nbus,1);
    Q = zeros(nbus,1);
    MPS=zeros(p,1);
    MQS=zeros(p,1);
        for t =2:nbus
            for k = 1:nbus
                P(t) = P(t) + vp(j, t) * vp(j, k) * (G(t, k) * cos(thp(j, t) - thp(j, k)) +
B(t, k) * sin(thp(j, t) - thp(j, k)));

                end
            end
            for t = 2:nbus
                for k = 1:nbus
                    Q(t) = Q(t) + vp(j, t) * vp(j, k) * (G(t, k) * sin(thp(j, t) - thp(j, k)) -
B(t, k) * cos(thp(j, t) - thp(j, k)));

                end
            end

        end
    % real power mismatch
    MP=P-Psp;
    MPS=MP.^2;
    %reactive power mismatch in third bus
    Qsp(3)=Q(3);

```

```

if Q(3)<Qmin;
    Q(3)=Qmin;
    PVIND(j,1)=1;
else PVIND(j,1)=0;
end
    if Q(3)>Qmax;
        Q(3)=Qmax;
        PVIND(j,1)=1;
    else PVIND(j,1)=0;
    end

%reactive power mismatch
MQ=Q-Qsp;
MQ(3)=0;
MQS=MQ.^2;
%objective function value
%objective function value
fp(j)=sum(MPS)+sum(MQS);
end

%calculating delta i and zeta

% sum1=0;
% for il=1:p
%     deltai(il,1)=f(il,i)-f(il,(i+1));
%     sum1=sum1+deltai(il,1);
% end
% zeta=sum1/(p);

%retaining particles with better performance

% il=1;
% k2=50;
% if(p>=k2+1)
%     while(il<=p)
%         count=count+1;
%         if(p<=k2)
%             break
%         end
%         if((f(il,i)-f(il,(i+1)))<0)
%             v(il,:,:)=[];
%             th(il,:,:)=[];
%             vv(il,:,:)=[];
%             vth(il,:,:)=[];
%             f(il,:)=[];
%             p=p-1;
%             if(p==k2)
%                 break
%             end
%         end
%     end
%     il=il+1;
% end
%end
%count=1;
%particle(i+1)=p;

%sorting and eliminating

```

```

if (p>pf && mod(i,fs)==0)
    fr=sortrows(fr);
    fr1=fr;
    k1=1;
    k2=0;
    k3=(p/r);
    k4=1;
    for i1=(k3+1):p
        mn(1,k1)=fr(i1,2);
        k1=k1+1;
    end
    v(mn, :, :)=[];
    th(mn, :, :)=[];
    vv(mn, :, :)=[];
    vth(mn, :, :)=[];
    f(mn, :, :)=[];
    fr(mn, :)=[];
    p=p/r;
end
mn=[];

%personal best value updation
for k=1:p
    for m=2:5
        if f(k,i+1)<fp(k)
            thp(k,m)=th(k,i+1,m);
        else
            end

    end

end

for k=1:p
    for m=2:5
        if f(k,i+1)<fp(k)
            vp(k,m)=v(k,i+1,m);
        else
            end

    end
end

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

for m=2:5

    for k=1:p
        if f(k,i+1)==fgm
            for l=2:p
                thg(l,m) = th(k,i+1,m); %global best values
            end
        else
            end
    end

end

```

```

end
    for m=2:5

        for k=1:p
            if f(k,i+1)==fgm
                for l=2:p
                    vg(l,m) = v(k,i+1,m);           %global best values
                end
            else
            end

        end
    end

    %stopping

    gbl=gb;
    fgm=min(f(:, (i+1)));
    for k=1:p
        if f(k,i+1)==fgm
            gbl=k;
        else
        end
    end
    end
    if (p==1)
        break
    end
    if (abs(f(gbl,i+1)-f(gbl,i))<=10^(-T))
        if (abs(f(gbl,i+1))<=10^(-T))
            break
        end
    end
    if abs(max(vv(gbl,i,:)))<=10^-(2.9) && abs(max(vth(gbl,i,:)))<=10^-(2.9) && i>=10 && abs(max(vv(gbl,i,:)))>=10^-(3.1) && abs(max(vth(gbl,i,:)))>=10^-(3.1)
        for j=1:p
            rfv(j)=(((abs(max(vv(gbl,i,:)))/max((vv(j,i,:))))^-1));
            rft(j)=(((abs(max(vth(gbl,i,:)))/max((vth(j,i,:))))^-1));
        end
    end
    % if abs(max(vv(gbl,i,:)))<=10^-(3.9) && abs(max(vth(gbl,i,:)))<=10^-(3.9) && i>=10 && abs(max(vv(gbl,i,:)))>=10^-(4.1) && abs(max(vth(gbl,i,:)))>=10^-(4.1)
    %     rfv=min(1.2, ((abs(max(vv(gbl,i,:)))/max((vv(gb,1,:))))^-1));
    %     rft=min(1.2, ((abs(max(vth(gbl,i,:)))/max((vth(gb,1,:))))^-1));
    % end
    % if abs(max(vv(gbl,i,:)))<=10^-(4.9) && abs(max(vth(gbl,i,:)))<=10^-(4.9) && i>=10 && abs(max(vv(gbl,i,:)))>=10^-(5.1) && abs(max(vth(gbl,i,:)))>=10^-(5.1)
    %     rfv=min(1.28, (abs(max(vv(gbl,i,:)))/max((vv(gb,1,:))))^-1);
    %     rft=min(1.28, (abs(max(vth(gbl,i,:)))/max((vth(gb,1,:))))^-1);
    % end
    % if max(vv(gbl,i,:))<=10^-(5) && max(vth(gbl,i,:))<=10^-(5) && i>=10
    %     rfv=1.21;
    %     rft=1.21;
    % end

    vth(gbl, (i+1),2);

```



```

end
%Q3(:,it)
bus=zeros(5,10) ;
bus(1,3)=v(gb1,i,1);
bus(2,3)=v(gb1,i,2);
bus(3,3)=v(gb1,i,3);
bus(4,3)=v(gb1,i,4);
bus(5,3)=v(gb1,i,5);
%bus angle updation
bus(1,4)=th(gb1,i,1);
bus(2,4)=th(gb1,i,2);
bus(3,4)=th(gb1,i,3);
bus(4,4)=th(gb1,i,4);
bus(5,4)=th(gb1,i,5);
bus(3,6)=Q3(gb1,it)*BMva;
x=bus(3,6);
V = bus(:,3) ; % Specified Voltage..
del = bus(:,4) ; % Voltage Angle..
toc
%load flow function calling
loadflow(nbus,V,del,BMva,x);
disp('function value');
f(gb1,i)
disp('no. of function evaluations =');
fev

```

## ybusppg.m

% Program to for Admittance And Impedance Bus Formation....

```

function Y = ybusppg(num) % Returns Y

linedata = linedatas(num); % Calling Linedatas...
fb = linedata(:,1); % From bus number...
tb = linedata(:,2); % To bus number...
r = linedata(:,3); % Resistance, R...
x = linedata(:,4); % Reactance, X...
b = linedata(:,5); % Ground Admittance, B/2...
a = linedata(:,6); % Tap setting value..
z = r + i*x; % z matrix...
y = 1./z; % To get inverse of each element...
b = i*b; % Make B imaginary...

nb = max(max(fb),max(tb)); % No. of buses...
nl = length(fb); % No. of branches...
Y = zeros(nb,nb); % Initialise YBus...

% Formation of the Off Diagonal Elements...
for k = 1:nl
    Y(fb(k),tb(k)) = Y(fb(k),tb(k)) - y(k)/a(k);
    Y(tb(k),fb(k)) = Y(fb(k),tb(k));
end

% Formation of Diagonal Elements....
for m = 1:nb
    for n = 1:nl
        if fb(n) == m

```

```

        Y(m,m) = Y(m,m) + y(n)/(a(n)^2) + b(n);
    elseif tb(n) == m
        Y(m,m) = Y(m,m) + y(n) + b(n);
    end
end
end
%Y; % Bus Admittance Matrix
%Z = inv(Y); % Bus Impedance Matrix

```

## linedatas.m

% Returns Line datas of the system...

```
function linedt = linedatas(num)
```

```

%      | From | To   | R      | X      | B/2   | X'mer  |
%      | Bus  | Bus  | pu     | pu     | pu     | TAP (a) |
linedat5 = [ 1      2      .1      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];

```

```

switch num
    case 5
        linedt = linedat5;
    case 30
        linedt = linedat30;
    case 57
        linedt = linedat57;
end

```

## loadflow.m

% Program for Bus Power Injections, Line & Power flows (p.u)...

```

function [Pi Qi Pg Qg Pl Ql] = loadflow(nb,V,del,BMva,x)
Y = ybusppg(nb); % Calling Ybus program..
lined = linedatas(nb); % Get linedats..
busd = busdatas(nb); % Get busdatas..
Vm = pol2rect(V,del); % Converting polar to rectangular..
Del = 180/pi*del; % Bus Voltage Angles in Degree...
fb = lined(:,1); % From bus number...
tb = lined(:,2); % To bus number...
nl = length(fb); % No. of Branches..
k1=busd(:,2); %bus type
Pl = busd(:,7) ; % PLi..
Ql = busd(:,8) ; % QLi..
Pl2 = busd(:,5) ; % PLi..
Ql2 = busd(:,6) ; % QLi..
Iij = zeros(nb,nb);
Sij = zeros(nb,nb);
Si = zeros(nb,1);

% Bus Current Injections..
I = Y*Vm;
Im = abs(I);

```

```

Ia = angle(I);

%Line Current Flows..
for m = 1:nl
    p = fb(m); q = tb(m);
    Iij(p,q) = -(Vm(p) - Vm(q))*Y(p,q); % Y(m,n) = -y(m,n)..
    Iij(q,p) = -Iij(p,q);
end
Iij = sparse(Iij);
Iijm = abs(Iij);
Iija = angle(Iij);

% Line Power Flows..
for m = 1:nb
    for n = 1:nb
        if m ~= n
            Sij(m,n) = Vm(m)*conj(Iij(m,n))*BMva;
        end
    end
end
Pij = real(Sij);
Qij = imag(Sij);

% Line Losses..
Lij = zeros(nl,1);
for m = 1:nl
    p = fb(m); q = tb(m);
    Lij(m) = Sij(p,q) + Sij(q,p);
end
Lpij = real(Lij);
Lqij = imag(Lij);

% Bus Power Injections..
for i = 1:nb
    for k = 1:nb
        Si(i) = Si(i) + conj(Vm(i))*Vm(k)*Y(i,k)*BMva;
    end
end
Pi = real(Si);
Qi = -imag(Si);
Pg = busd(:,5);
Qg = busd(:,6);
Pg(1)=sum(P1)+sum(Lpij)-Pg(3);
Qg(3)=x;
Qg(1)=sum(Q1)+sum(Lqij)-Qg(3);

disp('#####');
disp('-----');
disp('-----');
disp('PSO Loadflow Analysis ');
disp('-----');
disp('-----');
disp('| Bus | V | Angle | Injection | Generation |');
disp('Load |');
disp('| No | pu | Degree | MW | MVar | MW | Mvar |');
disp('MW | MVar | ');
for m = 1:nb

```

```

disp('-----');
disp('-----');
fprintf('%3g', m); fprintf(' %8.4f', V(m)); fprintf(' %8.4f',
Del(m));
fprintf(' %8.3f', Pi(m)); fprintf(' %8.3f', Qi(m));
fprintf(' %8.3f', Pg(m)); fprintf(' %8.3f', Qg(m));
fprintf(' %8.3f', Pl(m)); fprintf(' %8.3f', Ql(m)); fprintf('\n');
end
disp('-----');
disp('-----');
fprintf(' Total '); fprintf(' %8.3f', sum(Pi)); fprintf('
%8.3f', sum(Qi));
fprintf(' %8.3f', sum(Pi+Pl)); fprintf(' %8.3f', sum(Qi+Ql));
fprintf(' %8.3f', sum(Pl)); fprintf(' %8.3f', sum(Ql)); fprintf('\n');
disp('#####');
disp('#####');

disp('-----');
disp('-----');
disp(' Line FLOW and Losses ');
disp('-----');
disp('-----');
disp('|From|To | P | Q | From| To | P | Q |
Line Loss |');
disp('|Bus |Bus| MW | MVar | Bus | Bus| MW | MVar |
MW | MVar |');
for m = 1:n1
p = fb(m); q = tb(m);
disp('-----');
disp('-----');
fprintf('%4g', p); fprintf('%4g', q); fprintf(' %8.3f', Pij(p,q));
fprintf(' %8.3f', Qij(p,q));
fprintf('%4g', q); fprintf('%4g', p); fprintf(' %8.3f', Pij(q,p));
fprintf(' %8.3f', Qij(q,p));
fprintf(' %8.3f', Lpij(m)); fprintf(' %8.3f', Lqij(m));
fprintf('\n');
end
disp('-----');
disp('-----');
fprintf(' Total Loss ');
fprintf(' %8.3f', sum(Lpij)); fprintf(' %8.3f', sum(Lqij));
fprintf('\n');
disp('-----');
disp('-----');
disp('#####');
disp('#####');

```

ii) MATLAB code for load flow on IEEE 14 bus system using SEPSON.

Main program:

```
% objective function multiplied by 100 on 7th September , Monday , 2015
clc
nbus =14; % IEEE-5
Y = ybusppg(nbus); % Calling ybusppg.m to get Y-Bus Matrix..
busd = busdatas(nbus);
BMva=100; % Calling busdatas.. % Base
MVA..
Pg = busd(:,5)/BMva; % generated real power
Qg = busd(:,6)/BMva; % generated reactive power.
Pl = busd(:,7)/BMva; % load real power
Ql = busd(:,8)/BMva; % load reactive power
Qlim1 = busd(:,9)/BMva;
Qlim2 = busd(:,10)/BMva;
P = Pg - Pl; % Pi = PGi - PLi..
Q = Qg - Ql; % Qi = QGi - QLi..
Psp = P ; % P Specified..
Qsp = Q ; % q specified
for i=[2,3,6,8]
Qmin(i)=Qlim1(i)-Ql(i);
Qmax(i)=Qlim2(i)-Ql(i);
end
G = real(Y) ; % Conductance matrix..
B = imag(Y) ; % Susceptance matrix..
%kv=input('kv')
%-----PSO PARAMETERS INITIALIZATION -----%
fev=0;
c12=0;
kcm=1;
kpm=input('enter kpm');
kqm=input('enter kqm');
%kp=input('enter kp');
%kq=input('enter kq');
%kthm=input('enter kthm');
p=input('enter no. of initial particles'); % no of
particle
it=input('enter maximum no. of iterations'); % no of
iteration
rp=1;
T=input('enter tolerance'); %tolerance factor
particle=[];
mn=[];
fr1=[];
fs=input('frequency for sorting');
pf=input('final no. of particles'); % no of particle
r=input('enter r');
tic
c12=[];
rg=1;
rf=1;
f=zeros(p,it);
MQS1=zeros(4,1);
fp=zeros(p,1);
thp=zeros(p,nbus);
thg=zeros(p,nbus);
vp=zeros(p,nbus);
vg=zeros(p,nbus);
v=zeros(p,it,nbus);
```

```

th=zeros(p,it,nbus);
vth=zeros(p,it,nbus);
vv=zeros(p,it,nbus);
a=-0.5;
b=0.5;
vth(:,1,:)=a+(b-a)*rand(p,nbus); %initial velocity of theta vector%
a=1;
b=-0.1;
vv(:,1,:)=a+(b-a)*rand(p,nbus); %initial velocity of voltage vector%
%vth(:, :,1)=0;
a=0;
b=0;
th(:,1,:)=a+(b-a)*rand(p,nbus);
a=1;
b=1;
v(:,1,:)=a+(b-a)*rand(p,nbus);
% volage asuumption
v(:, :,1)=1.060;
v(:, :,2)=1.045;
v(:, :,3)=1.010;
v(:, :,6)=1.070;
v(:, :,8)=1.090;
vv(:, :,1)=0;
th(:, :,1)=0;
thp(:,1)=0;
thg(:,1)=0;
vp(:,1)=1.060;
vp(:,2)=1.045;
vp(:,3)=1.010;
vp(:,6)=1.070;
vp(:,8)=1.090;
vg(:,1)=1.060;
vg(:,2)=1.045;
vg(:,3)=1.010;
vg(:,6)=1.070;
vg(:,8)=1.090;
vmin=[0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9];
vmax=[1.06 1.045 1.010 1.1 1.1 1.070 1.1 1.090 1.1 1.1 1.1 1.1 1.1 1.1];

%-----initial value of objective function-----%
% Calculate P and Q
PVIND=zeros(p,nbus);
MV=zeros(p,nbus);
check_no=0;
for j=1:p
    P = zeros(nbus,1);
    Q = zeros(nbus,1);
    MPS=zeros(14,1);
    MQS=zeros(14,1);
    MQS1=zeros(4,1);
    MVS=zeros(14,1);

    for i = 1:nbus
        for k = 1:nbus
            P(i) = P(i) + v(j,1,i)* v(j,1,k)*(G(i,k)*cos(th(j,1,i)-
th(j,1,k)) + B(i,k)*sin(th(j,1,i)-th(j,1,k)));
        end
    end

    for i = 1:nbus
        for k = 1:nbus

```

```

        Q(i) = Q(i) + v(j,1,i)*v(j,1,k)*(G(i,k)*sin(th(j,1,i)-
th(j,1,k)) - B(i,k)*cos(th(j,1,i)-th(j,1,k)));
    end

end

% real power mismatch
MP=P-Psp;
if (P(1)-.5)*(2-(P(1))) >= 0
    MP(1)=0;
end
if (P(1)-.5)*(2-(P(1))) < 0
    MP(1)=abs(min((P(1)-.5),(2-(P(1)))));
end
%if (P(2)-.2)*(1-(P(2))) >= 0
%    MP(2)=P(2)-Psp(2);
%end
%if (P(2)-.2)*(1-(P(2))) < 0
%    MP(2)=abs(min((P(2)-.5),(2-(P(2)))));
%end
MPS=MP.^2;

%reactive power mismatch and voltage mismatch
MQ=Q-Qsp;
MQS=MQ.^2;
MQS(1)=0;

% MVS=zeros(14,1);
% for jk=2:14
%     if (v(j,1,jk)-0.9)*(1.1-v(j,1,jk)) < 0
%         MVS(jk)=(min((v(j,1,jk)-0.9),(1.1-v(j,1,jk))))^2;
%     end
%     if (v(j,1,jk)-0.9)*(1.1-v(j,1,jk)) >= 0
%         MVS(jk)=0;
%     end
% end

for jk=[2,3,6,8]
    if ((Q(jk)-Qmin(jk))*(Qmax(jk)-Q(jk))<0)
        % MVS(jk)=(v(j,1,jk)-vmax(jk))^2;
        MQS(jk)=(min((Q(jk)-Qmin(jk)), (Qmax(jk)-Q(jk))))^2;
        vv(j,1,jk)=0; %change made at 1906 hrs , 6th September , Sunday
, 2015
    end
    if ((Q(jk)-Qmin(jk))*(Qmax(jk)-Q(jk))>=0)
        MQS(jk)=0;
        % MVS(jk)=(vmax(jk)-v(j,1,jk))^2;
        vv(j,1,jk)=0;
    end
end

%objective function value
%
f(j,1)=max(max(kpm*max(MPS),kqm*max(MQS)),max(kvm*max(MVS),kthm*sum(MTHS)));
;
    f(j,1)=(kpm*sum(MPS)+kqm*sum(MQS));%+kvm*sum(MVS);%+kthm*sum(MTHS);
    %fr(j,1)=f(j,1);
    %fr(j,2)=j;
    fev=fev+1;
end

```

```

%Initial personal best values
for i=1:p
    for k=1:nbus
        thp(i,k)=th(i,1,k);
    end
    for k=1:nbus
        vp(i,k)=v(i,1,k);
    end
end
%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
    for j=1:nbus
        thg(k,j)=th(gb,1,j);
    end
    for j=1:nbus
        vg(k,j)=v(gb,1,j);
    end
end
fgm = min(f(:,1));
Q3=zeros(p,it,nbus);
for i=1:it
    %for inertia weight W
    %wmax=.4;
    %wmin=.3;
    %w=wmax-((wmax-wmin)*i/it);
    %velocity update
    %position update
    w=0.8;
    for j=1:p
        % w(j)=.4+(f(j,i)*(min(f(:,i))-f(j,i)))/(fp(j)*(min(f(:,i))-f(j,i))));
        % L1(j)=sqrt(fp(j)/f(j,i));
        % L2(j)=sqrt(min(f(:,i))/f(j,i));
        for k=1:nbus

            vth(j,(i+1),k) = w*vth(j,i,k) + rp*rand()*(thp(j,k)-th(j,i,k)) +
            rg*rand()*(thg(j,k)-th(j,i,k));
            th(j,(i+1),k) = th(j,i,k) + rf*vth(j,(i+1),k);
        end

        for q=2:nbus
            vv(j,(i+1),q) = w*vv(j,i,q) + rp*rand()*(vp(j,q)-v(j,i,q)) +
            rg*rand()*(vg(j,q)-v(j,i,q));
            v(j,(i+1),q) = v(j,i,q) + rf*vv(j,(i+1),q);
        end
    end
end
%objective function value
for j=1:p

```



```

P = zeros(nbus,1);
Q = zeros(nbus,1);
MPS=zeros(14,1);
MQS=zeros(14,1);
MQS1=zeros(14,1);
    for m = 1:nbus
        for k = 1:nbus
            P(m) = P(m) + v(j, (i+1),m)*
v(j, (i+1),k)*(G(m,k)*cos(th(j, (i+1),m)-th(j, (i+1),k)) +
B(m,k)*sin(th(j, (i+1),m)-th(j, (i+1),k)));

        end
    end
    for m = 1:nbus
        for k = 1:nbus
            Q(m) = Q(m) + v(j, (i+1),m)*
v(j, (i+1),k)*(G(m,k)*sin(th(j, (i+1),m)-th(j, (i+1),k)) -
B(m,k)*cos(th(j, (i+1),m)-th(j, (i+1),k)));

        end
    end

% real power mismatch
MP=P-Psp;
if (P(1)-.5)*(2-(P(1))) >= 0
    MP(1)=0;
end
if (P(1)-.5)*(2-(P(1))) < 0
    MP(1)=abs(min((P(1)-.5), (2-(P(1))))));
end
% if (P(2)-.2)*(1-(P(2))) >= 0
%     MP(2)=P(2)-Psp(2);
% end
% if (P(2)-.2)*(1-(P(2))) < 0
%     MP(2)=abs(min((P(2)-.5), (2-(P(2))))));
% end
MPS=MP.^2;

%reactive power mismatch and voltage mismatch
MQ=Q-Qsp;
MQS=MQ.^2;
MQS(1)=0;

% MVS=zeros(14,1);
% for jk=2:14
%     if (v(j,1,jk)-0.9)*(1.1-v(j,1,jk)) < 0
%         MVS(jk)=(min((v(j,1,jk)-0.9), (1.1-v(j,1,jk))))^2;
%     end
%     if (v(j,1,jk)-0.9)*(1.1-v(j,1,jk)) >= 0
%         MVS(jk)=0;
%     end
% end

for jk=[2,3,6,8]
    if ((Q(jk)-Qmin(jk))*(Qmax(jk)-Q(jk))<0)
        % MVS(jk)=((v(j,1,jk)-vmax(jk))^2);
        MQS(jk)=(min((Q(jk)-Qmin(jk)), (Qmax(jk)-Q(jk))))^2;
    end
    if ((Q(jk)-Qmin(jk))*(Qmax(jk)-Q(jk))>=0)
        MQS(jk)=0;
    end
end

```

```

        end
    end

    %objective function value

    %f(j,i+1)=max(max(kpm*max(MPS),kqm*max(QQS)),max(kvm*max(MVS),kthm*sum(MTHS
    )));
    f(j,(i+1))=(kpm*sum(MPS)+kqm*sum(QQS));%+kvm*sum(MVS);%+kthm*sum(MTHS);
    fr(j,1)=f(j,i+1);
    fr(j,2)=j;
    fev=fev+1;
end

%personal best values updation
for j=1:p
    P = zeros(nbus,1);
    Q = zeros(nbus,1);
    MPS=zeros(p,1);
    MQS=zeros(p,1);
        for t =1:nbus
            for k = 1:nbus
                P(t) = P(t) + vp(j,t)* vp(j,k)*(G(t,k)*cos(thp(j,t)-thp(j,k)) +
                B(t,k)*sin(thp(j,t)-thp(j,k)));

                end
            end
            for t = 1:nbus
                for k = 1:nbus
                    Q(t) = Q(t) + vp(j,t)* vp(j,k)*(G(t,k)*sin(thp(j,t)-thp(j,k)) -
                    B(t,k)*cos(thp(j,t)-thp(j,k)));

                end
            end
        end

% real power mismatch
MP=P-Psp;
if (P(1)-.5)*(2-(P(1))) >= 0
    MP(1)=0;
end
if (P(1)-.5)*(2-(P(1))) < 0
    MP(1)=abs(min((P(1)-.5),(2-(P(1))))));
end
% if (P(2)-.2)*(1-(P(2))) >= 0
%     MP(2)=P(2)-Psp(2);
% end
% if (P(2)-.2)*(1-(P(2))) < 0
%     MP(2)=abs(min((P(2)-.5),(2-(P(2))))));
% end
MPS=MP.^2;

%reactive power mismatch and voltage mismatch
MQ=Q-Qsp;
MQS=MQ.^2;
MQS(1)=0;

% MVS=zeros(14,1);
% for jk=2:14
%     if (v(j,1,jk)-0.9)*(1.1-v(j,1,jk)) < 0
%         MVS(jk)=(min((v(j,1,jk)-0.9),(1.1-v(j,1,jk))))^2;
%     end

```

```

%      if (v(j,1,jk)-0.9)*(1.1-v(j,1,jk)) >= 0
%          MVS(jk)=0;
%      end
% end

for jk=[2,3,6,8]
    if ((Q(jk)-Qmin(jk))*(Qmax(jk)-Q(jk))<0)
        % MVS(jk)=(v(j,1,jk)-vmax(jk))^2;
        MQS(jk)=(min((Q(jk)-Qmin(jk)), (Qmax(jk)-Q(jk))))^2;
    end
    if ((Q(jk)-Qmin(jk))*(Qmax(jk)-Q(jk))>=0)
        MQS(jk)=0;
    end
end

%objective function value

%fp(j)=max(max(kpm*max(MPS), kqm*max(MQS)), max(kvm*max(MVS), kthm*sum(MTHS)))
;
    fp(j)=(kpm*sum(MPS)+kqm*sum(MQS)); %+kvm*sum(MVS); %+kthm*sum(MTHS);
    fev=fev+1;
end

if(p>pf && mod(i,fs)==0)
    fr=sortrows(fr);
    fr1=fr;
    k1=1;
    k2=0;
    k3=(p/r);
    k4=1;
    for i1=(k3+1):p
        mn(1,k1)=fr(i1,2);
        k1=k1+1;
    end
    v(mn, :, :)=[];
    th(mn, :, :)=[];
    vv(mn, :, :)=[];
    vth(mn, :, :)=[];
    vp(mn, :)=[];
    thp(mn, :)=[];
    vg(mn, :)=[];
    thg(mn, :)=[];
    f(mn, :, :)=[];
    fr(mn, :)=[];
    p=p/r;
end
mn=[];

%personal best value updation
for k=1:p
    for m=1:nbus
        if f(k,i+1)<fp(k)
            thp(k,m)=th(k,i+1,m);
            vp(k,m)=v(k,i+1,m);
        else
            end
        end
    end
end
end

```

```

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

for m=1:nbus

    for k=1:p
        if f(k,i+1)==fgm
            for l=1:p
                thg(l,m) = th(k,i+1,m);    %global best values
            end
        else
            end
        end

    end
end
for m=2:nbus

    for k=1:p
        if f(k,i+1)==fgm
            for l=1:p
                vg(l,m) = v(k,i+1,m);    %global best values
            end
        else
            end
        end

    end
end
%stopping criteria
%    gbl=gb;
fgm=min(f(:, (i)));
for k=1:p
    if f(k,i)==fgm
        gbl=k;
    else
        end
end
if (abs(f(gbl,i))<=10^(-T))

    break
end

%vth(gbl, (i+1),2)
end
toc;
c12=toc;
P=zeros(14,1);
Q=zeros(14,1);
for m = 1:nbus
    for k = 1:nbus
        P(m) = P(m) + v(gbl, (i),m)*
v(gbl, (i),k)*(G(m,k)*cos(th(gbl, (i),m)-th(gbl, (i),k)) +
B(m,k)*sin(th(gbl, (i),m)-th(gbl, (i),k)));

    end
end
for m = 1:nbus
    for k = 1:nbus

```

```

        Q(m) = Q(m) + v(gb1, (i), m) *
v(gb1, (i), k) * (G(m, k) * sin(th(gb1, (i), m) - th(gb1, (i), k)) -
B(m, k) * cos(th(gb1, (i), m) - th(gb1, (i), k)));

    end
end
for j=1:1:14
    fprintf('\n power calculated at bus no. %d is %f + j %f', j, P(j), Q(j));
    fprintf('\n power demand at bus no. %d is %f + j %f', j, Psp(j), Qsp(j));
    fprintf('\n      voltage      %f', v(gb1, i, j));
    fprintf('\n      delta      %f', th(gb1, i, j) * 180 / (pi));
    fprintf('\n');
end
disp('function value =');
f(gb1, i)
disp('no. of function evaluations');
fev
disp('time elapsed')
c12
disp('no. of iterations')
i

```

### iii) MATLAB code for minimizing benchmark functions using PSO.

#### Main program

```
clc
clear all;
p1=0;
p2=0;
prompt = {'Enter dimension:', 'Enter swarm size:', 'Enter maximum number of
iterations', 'Enter tolerance', 'Enter inertia weight factor', 'Enter
cognitive learning acceleration factor', 'Enter social learning acceleration
factor', 'Enter constriction factor', 'Enter initial velocity range', 'Enter
initial position range'};
dlg_title = 'We have to minimize inputted function';
num_lines = 1;
def = {'2', '400', '1000', '4', '.6', '1', '1', '1', 'unifrnd(-
5,10,1,p)', 'unifrnd(-5,10,1,p)'};
answer = inputdlg(prompt,dlg_title,num_lines,def);
[D vald] = str2num(answer{1});
[p valp] = str2num(answer{2});
[it valit]=str2num(answer{3});
[T valT] = str2num(answer{4});
[w valw] = str2num(answer{5});
[rp valrp] = str2num(answer{6});
[rg valrg]=str2num(answer{7});
[rf valrf] = str2num(answer{8});
str={'Select a function'};
S={'rosenbrock'; 'sphere'; 'griewank'; 'rastrigin'};
choice=listdlg('PromptString',str, 'ListSize', [100,100], 'ListString', S, 'Sele
ctionMode', 'Single');
switch choice
    case 1
        str1=rosenbrock(D);
    case 2
        str1=sphere(D);
    case 3
        str1=griewank(D);
    case 4
        str1=rastrigin(D);
end
fev=0;
x=zeros(D,p,it);
v=zeros(D,p,it);
f=zeros(p,it);
fr=zeros(p,2);
df=zeros(1,(it-1));
mn=[];
tic
for d=1:D
    x(d,:,1)=eval(answer{9});
    v(d,:,1)=eval(answer{10});
end

for j=1:p
    i=0;
    f(j,1)=0;
    f(j,1)=f(j,1)+ eval(str1);
end
```

```

[val gbl]=min(f(:,1));
gbest=gbl;
pbest=ones(1,p);
for i=1:it

%for inertia weight W

    for j=1:p
        for d=1:D
            k2=0;
            kstr='w*v(d,j,i) + (rp*(x(d,j,pbest(j))-x(d,j,i))) +
(rg*(x(d,gbest,i)-x(d,j,i)))';
            k2=k2+eval(kstr);
            v(d,j,(i+1)) = (rf*k2);
            x(d,j,(i+1)) = x(d,j,i) + v(d,j,(i+1));
        end
    end

    for j=1:p
        f(j,i+1)=0;
        f(j,i+1)= f(j,i+1)+ eval(str1);
        fev=fev+1;
    end
    choicel=2;
    if mod(i,20000)==0
        str23={'Exit ?'};
        S1={'Yes';'No'};

    choicel=listdlg('PromptString',str23,'ListSize',[100,100],'ListString',S1,'
SelectionMode','Single');
    end
    %stopping criterion
    if i>1
%
        print = [x(d(:,i-1) x2(:,i-1) v1(:,i-1) v2(:,i-1) f(:,i-1)];
%
        disp('      x1      x2      v1      v2      f')
%
        disp(print)
        [val gbl]=min(f(:,i+1));
        if val<10^(-T) || choicel==1
            break
        end
    end
    gbest=gbl;
    for i2=1:p
        if f(i2,i+1)<= f(i2,i)
            pbest(i2)=i+1;
        end
    end
end
end
toc
sprintf('the value of objective function is %d',(f(gbl,i+1)))
sprintf('the number of function evaluations is %d',fev)
for d=1:D
    fprintf('the value of x %d is',d)
    x(d,gbl,i+1)
end
end

```

iv) MATLAB code for minimizing benchmark functions using SEPSON.

Main program

```
clc
mn=[];
fr1=[];
clear all;
p1=0;
p2=0;
prompt = {'Enter dimension:', 'Enter swarm size:', 'Enter maximum number of
iterations', 'Enter tolerance', 'Enter inertia weight factor', 'Enter
cognitive learning acceleration factor', 'Enter social learning acceleration
factor', 'Enter constriction factor', 'Enter initial velocity range', 'Enter
initial position range', 'Enter final swarm size', 'Enter reduction
factor', 'Enter sorting frequency'};
dlg_title = 'We have to minimize inputted function';
num_lines = 1;
def = {'2', '400', '1000', '4', '.6', '1', '1', '1', 'unifrnd(-
5,10,1,p)', 'unifrnd(-5,10,1,p)', '200', '2', '10'};
answer = inputdlg(prompt,dlg_title,num_lines,def);
[D vald] = str2num(answer{1});
[p valp] = str2num(answer{2});
[it valit]=str2num(answer{3});
[T valT] = str2num(answer{4});
[w valw] = str2num(answer{5});
[rp valrp] = str2num(answer{6});
[rg valrg]= str2num(answer{7});
[rf valrf] = str2num(answer{8});
[pf valPf]=str2num(answer{11});
[r valr]=str2num(answer{12});
[fs valfs]=str2num(answer{13});
str={'Select a function'};
S={'rosenbrock'; 'sphere'; 'griewank'; 'rastrigin'};
choice=listdlg('PromptString',str, 'ListSize', [100,100], 'ListString', S, 'Sele
ctionMode', 'Single');
switch choice
    case 1
        str1=rosenbrock(D);
    case 2
        str1=sphere(D);
    case 3
        str1=griewank(D);
    case 4
        str1=rastrigin(D);
end
fev=0;
x=zeros(D,p,it);
v=zeros(D,p,it);
f=zeros(p,it);
fr=zeros(p,2);
df=zeros(1,(it-1));
mn=[];
tic
for d=1:D
    x(d,:,1)=eval(answer{9});
    v(d,:,1)=eval(answer{10});
end

for j=1:p
```



```

        i=0;
        f(j,1)=0;
        f(j,1)=f(j,1)+ eval(str1);
    end

    [val gb1]=min(f(:,1));
    gbest=gb1;
    pbest=ones(1,p);
    for i=1:it

        %for inertia weight W

            for j=1:p
                for d=1:D
                    k2=0;
                    kstr='w*v(d,j,i) + (rp*(x(d,j,pbest(j))-x(d,j,i))) +
(rg*(x(d,gbest,i)-x(d,j,i)))';
                    k2=k2+eval(kstr);
                    v(d,j,(i+1)) = (rf*k2);
                    x(d,j,(i+1)) = x(d,j,i) + v(d,j,(i+1));
                end
            end

            for j=1:p
                f(j,i+1)=0;
                f(j,i+1)= f(j,i+1)+ eval(str1);
                fr(j,1)=f(j,i+1);
                fr(j,2)=j;
                fev=fev+1;
            end
            choicel=2;
            if mod(i,20000)==0
                str23={'Exit ?'};
                S1={'Yes';'No'};

                choicel=listdlg('PromptString',str23,'ListSize',[100,100],'ListString',S1,'
SelectionMode','Single');
            end

            if(p>pf && mod(i,fs)==0)
                fr=sortrows(fr);
                fr1=fr;
                k1=1;
                k2=0;
                k3=(p/r);
                k4=1;
                for i1=(k3+1):p
                    mn(1,k1)=fr(i1,2);
                    k1=k1+1;
                end
                v(:,mn,:)=[];
                x(:,mn,:)=[];
                f(mn,:)=[];
                fr(mn,:)=[];
                p=p/r;
            end
            mn=[];

            %stopping criterion
            if i>1

```

```

%      print = [x(d,:,i-1) x2(:,i-1) v1(:,i-1) v2(:,i-1) f(:,i-1)];
%      disp('      x1      x2      v1      v2      f')
%      disp(print)
[val gbl]=min(f(:,i+1));
if val<10^(-T) || choice1==1
    break
end
end
gbest=gbl;
for i2=1:p
    if f(i2,i+1)<= f(i2,i)
        pbest(i2)=i+1;
    end
end
end
toc
sprintf('the value of objective function is %d', (f(gbl,i+1)))
sprintf('the number of function evaluations is %d', fev)
for d=1:D
    fprintf('the value of x %d is', d)
    x(d,gbl,i+1)
end

```

#### v) Benchmark functions implemented in MATLAB

##### Rosenbrock

```

function f = rosenbrock(D)
str2=' ';
str1='(100*( (x(d,j, (i+1)))^2 - x(d+1,j, (i+1)) )^2) +((1-
x(d,j, (i+1)))^2)';
for d=1:D-1
    str2=strcat(str2, '+', str1);
    str2=strrep(str2, 'd', num2str(d));
end
f=str2;
f(1)=[];
end

```

##### Griewank

```

function f = griewank(D)
str2=' ';
str1='( (1/4000)*(x(d,j, (i+1)))^2)';
str3='(cos(x(d,j, (i+1)))/(sqrt(d)))';
for d=1:D
    str2=strcat(str2, '+', str1);
    str2=strrep(str2, 'd', num2str(d));
end
str4='(cos(x(1,j, (i+1)))/(sqrt(1)))';
for d=2:D
    str4=strcat(str4, '*', str3);
    str4=strrep(str4, 'd', num2str(d));
end
f=strcat('1', str2, '-', str4);
end

```

## Sphere

```
function f = sphere(D)
str2=' ';
str1='(x(d,j,(i+1)))^2';
for d=1:D
    str2=strcat(str2,'+',str1);
    str2=strrep(str2,'d',num2str(d));
end
f=str2;
f(1)=[];
end
```

## Rastrigin

```
function f = rastrigin(D)
str2=' ';
str1='((x(d,j,(i+1)))^2)';
str3='(10*(cos(2*pi*(x(d,j,(i+1))))))';
for d=1:D
    str2=strcat(str2,'+',str1);
    str2=strrep(str2,'d',num2str(d));
end
str4='(10*(cos(2*pi*(x(1,j,(i+1))))))';
for d=2:D
    str4=strcat(str4,'+',str3);
    str4=strrep(str4,'d',num2str(d));
end
f=strcat(num2str(10*D),str2,'-', '(' ,str4, ') ');
end
```

## REFERENCES

- [1] Kennedy J., Eberhart R., “Particle Swarm Optimization”, Proceedings of IEEE International Conference on Neural Networks, 1995, vol. 4, Nov/Dec 1995, pp.1942-1948.
- [2] Benedetti M., Massa, A., “Memory Enhanced PSO-Based Optimization Approach for Smart Antennas Control in Complex Interference Scenarios”, IEEE Transactions on Antennas and Propagation Magazine, vol. 56, no. 7, July 2008, pp.1939-1947.
- [3] Haibin Duan, Pei Li, Yaxiang Yu., “A predator-prey Particle swarm optimization approach to multiple UCAV air combat modeled by Dynamic game theory”, IEEE/CAA Journal of Automatica Sinica, vol. 2, no. 1, January 10 2015, pp.11-18.
- [4] J Voss M.S.,” Principal Component Particle Swarm Optimization (PCPSO)”, IEEE Proceedings on Swarm Intelligence Symposium, 2005, 8-10 June 2005, pp.401-404.
- [5] Liang J. J., A. K. Qin, Ponnuthurai Nagarathnam Suganthan, S. Baskar, “Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions.”, IEEE Transactions On Evolutionary Computation, Vol. 10, No. 3, June 2006.
- [6] Changhe Li, Shengxiang Yang, Trung Thanh Nguyen,” A Self-Learning Particle Swarm Optimizer for Global Optimization Problems”, IEEE Transactions On Systems, Man, And Cybernetics—Part B: Cybernetics, Vol. 42, No .3, June 2012.
- [7] Zhi-Hui Zhan, Jun Zhang, Yun Li, Yu-Hui Shi, “Orthogonal Learning Particle Swarm Optimization.”, IEEE Transactions On Evolutionary Computation, Vol. 15, No. 6, December 2011.
- [8] Feng Chen, Xinxin Sun, Dali Wei, Yongning Tang, ”Tradeoff Strategy between Exploration and Exploitation for PSO”, 2011 Seventh International Conference on Natural Computation.
- [9] J.C. Bansal, P.K. Singh, Mukesh Saraswat, Abhishek Verma, Shimpi Singh Jadon, Ajith Abraham, “Inertia Weight Strategies in Particle Swarm Optimization”, IEEE Third World Congress on Nature and Biologically Inspired Computing, 2011, pp.633-640.
- [10] Xia Yu, Jianchang Liu, Hongru Li, “An Adaptive Inertia Weight Particle Swarm Optimization Algorithm for IIR Digital Filter”, IEEE Artificial Intelligence and Computational Intelligence Magazine, vol. 1, November 2009, pp.114-118 .

- [11] Yuhong Chi, Fuchun Sun, Langfan Jiang, Chunming Yu, “An efficient population diversity measure for improved particle swarm optimization algorithm”, Intelligent Systems (IS), 2012 6th IEEE International Conference, 6-8 Sept. 2012, pp. 361 – 367.
- [12] Narendra Kumar Jain, Uma Nangia, Aishwary Jain, “PSO for Multiobjective Economic Load Dispatch (MELD) for Minimizing Generation Cost and Transmission Losses”, Journal of The Institution of Engineers (India): Series B, 7 February 2015.
- [13] M.A. Abido, "Optimal power flow using particle swarm optimization", International Journal of Electrical and Power Energy systems, Volume 24, Issue 7, October 2002, pp. 563–571.
- [14] Ruey-Hsun Liang, Ruey-Hsun Liang, Yie-Tone Chen, Wan-Tsun Tseng, “Optimal power flow by a fuzzy based hybrid particle swarm optimization approach”, Electric Power Systems Research, Volume 81, Issue 7, July 2011, pp. 1466–1474.
- [15] Salomon C.P., Lambert-Torres G., Martins H.G., Ferreira, C. Costa C.I.A. “Load flow computation via Particle Swarm Optimization”, 9th IEEE/IAS International Conference on Industry Applications (INDUSCON) 2010, 8-10 Nov. 2010, pp. 1 – 6
- [16] Acharjee P., Goswami S.K., “Chaotic Particle Swarm Optimization based reliable algorithm to overcome the limitations of conventional power flow methods, “ Power Systems Conference and Exposition, 2009. PSCE '09. IEEE/PES, 15-18 March 2009, pp. 1 – 7.
- [17] “Flexible AC Transmission System (FACTS) Devices” by Nkusi Ernest.
- [18] E. A. Grimaldi, F. Grimaccia, M. Mussetta, R. E. Zich, “PSO as an effective learning algorithm for neural network applications”, Proceedings of 3rd International Conference on Computational Electromagnetics and Its Applications, 2004. ICCEA 2004, 1-4 Nov. 2004, pp. 557-560.
- [19] R.C. Eberhart, Xiaohui Hu, “Human Tremor Analysis Using Particle Swarm Optimization”, Purdue School of Engineering and Technology.
- [20] Norfadzlan Yusupa, Azlan Mohd Zainb, Siti Zaiton Mohd Hashimc, “Overview of PSO for Optimizing Process Parameters of Machining”, International Workshop on Information and Electronics Engineering 2012, Procedia Engineering vol. 29, 2012, pp. 914–923.

- [21] Taher Niknam, Hamed Zeinoddini Meymand, Hasan Doagou Mojarad “A practical multi-objective PSO algorithm for optimal operation management of distribution network with regard to fuel cell power plants”, *Renewable Energy*, vol. 36, Issue 5, May 2011, pp. 1529–1544.
- [22] Hirotaka Yoshida, Kenichi Kawata, Yoshikazu Fukuyama, Member, Shinichi Takayama, Yosuke Nakanishi, “A Particle Swarm Optimization for Reactive Power and Voltage Control Considering Voltage Security Assessment”, *IEEE Transactions On Power Systems*, vol. 15, no. 4, November 2000, pp. 1232-1239.
- [23] Jianxun Lv, Haiwen Yuan, Yingming Lv, “Battery State-of-charge Estimation Based on Fuzzy Neural Network and Improved Particle Swarm Optimization Algorithm”, *Second International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC)*, 2012 , 8-10 Dec. 2012, pp. 22-27.
- [24] Z. L. Gaing, “Particle swarm optimization to solving the economic dispatch considering the generator constraints,” *IEEE Trans. on Power Systems*, Aug. 2003, vol. 18, no. 3, pp. 1187-1195.
- [25] John Grainger, Stevenson Jr. William, “Power Systems Analysis ,” Tata McGraw Hill, 5<sup>th</sup> December, 2003.



