PARTICLE SWARM OPTIMIZATION AND IT'S APPLICATION ON ANALYSIS OF LOAD FLOW STUDIES

A PROJECT REPORT

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF TECHNOLOGY

IN

ELECTRICAL ENGINEERING

Submitted By

RAJAT SHARMA (2K15/EE/098)

SIDDHARTH MEHROTRA (2K15/EE/125)

TARANG JAIN (2K15/EE/132)

VAIBHAV AHUJA (2K15/EE/134)

Under the supervision of:

Professor N.K. Jain

DEPARTMENT OF ELECTRICAL ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

MAY 2019

DEPARTMENT OF ELECTRICAL ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi - 110042

CANDIDATE'S DECLARATION

We hereby certify that the work which is presented in the Major Project titled "**PARTICLE SWARM OPTIMIZATION AND IT'S APPLICATION ON ANALYSIS OF LOAD FLOW STUDIES**" in fulfillment for the award of the Degree of Bachelor of Technology and submitted to the Department of Electrical Engineering, Delhi Technological University (formerly Delhi College of Engineering), New Delhi is an authentic record of our own, carried out during a period of January 2019 - May 2019, under the supervision of Professor N.K. Jain.

The matter presented in this report has not been submitted by us for the award of any other degree of this or any other Institute/University.

RAJAT SHARMA	**2K15/EE/098**
SIDDHARTH MEHROTRA	**2K15/EE/125**
TARANG JAIN	**2K15/EE/132**
VAIBHAV AHUJA	**2K15/EE/134**

Date: 25/05/2019
Place: New Delhi

DEPARTMENT OF INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi - 110042

CERTIFICATE

This is to certify that the project dissertation titled **PARTICLE SWARM OPTIMIZATION AND IT'S APPLICATION ON ANALYSIS OF LOAD FLOW STUDIES** is a bonafide record of the work done under my guidance by

RAJAT SHARMA	**2K15/EE/098**
SIDDHARTH MEHROTRA	**2K15/EE/125**
TARANG JAIN	**2K15/EE/132**
VAIBHAV AHUJA	**2K15/EE/134**

students of **Delhi Technological University, Delhi** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Electrical Engineering**. To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

It is further certified that the work done in this dissertation is a result of candidates' efforts.

Place: New Delhi
Date: 25/05/2019

Professor N.K. Jain
SUPERVISOR
Department of Electrical Engineering,DTU

ABSTRACT

Particle Swarm Optimization is an optimization technique that is used to find the minimum or maximum value of a given function and the values of variables or parameters that give this value. With the evolution of computer and the need to incorporate its usage in power system for a more efficient power system management and power system stability, this intelligent optimization technique is finding deeper roots in scholars' and electrical engineers' field of study.

In this project, Particle Swarm Optimization Technique is first implemented on Rosenbrock Function (Lab Model) and then implemented on 5-Bus Power System (Practical Model). Particle Swarm Optimization technique is used in Load Flow Analysis to find the unknown variables of voltage, power angle, load and generation of buses using the known values of buses in power system as parameters. Here, voltage and power angle of buses form the parameters. These parameters are used to calculate active and reactive power of each bus which is compared with the known active and reactive power of buses of the system. The mismatch between the two forms the objective function which is optimized by Particle Swarm Optimization technique finally giving us the unknown values.

This work also tries to make the technique more efficient. The higher efficiency is obtained by reducing the number of iterations ie. total time taken for program to give the result. Before particles reach their final value, they undergo a large number of variations. The particles, after a certain number of iterations, are repositioned such that they are close to final solution. The variations are thus reduced, iterations reduced and program's efficiency increases.

ACKNOWLEDGEMENT

On the very outset of this report, we would like to extend our sincere and heartfelt obligation towards all the people who have helped us in this endeavour.

We are ineffably indebted to **Professor N.K Jain** for his conscientious guidance, exceptional mentoring and constant monitoring to accomplish this assignment.

We would also like to extend our sincere gratitude to the **Department of Electrical Engineering**, for giving us the opportunity to execute this project, which is an integral part of the curriculum in B.Tech program at Delhi Technological University.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Evolutionary Algorithms

Recently, scientific and technological advancements have led to the invention of new techniques based on evolution which are fast becoming a noteworthy analysis field. It's not solely attributable to its effectiveness in searching for the correct solution but also to its close resemblance to natural social paradigms. Since it's influenced by nature, these methods are supported by biological evolution. The basic idea and main features of EAs are represented as a contest between a populations of people. The main characteristics of EAs are as follows: They deal with an entire group of candidate solutions at the same time i.e., EAs are population-based. EAs largely use recombination to combine data that is obtained from a huge variety of candidate answers into a brand new solution. These optimisation tools als incorporate a degree of randomness into their working algorithm.

Kennedy, a social scientist, and Eberhart, an electrical engineer, conceptualized the idea of particle swarms, to build computational intelligence capabilities, making use of the prevailing systems that interact naturally. The primary simulations were inspired by social functioning of concerned analogues of flocks of birds looking for food. This led to the development of a robust optimization method; Particle Swarm optimization (PSO). PSO tries to imitate the goal-oriented working procedure of swarms that can be viewed biologically.

1.2 Particle Swarm Optimization

In the PSO calculation, the group of particles in the search area intend to efficiently solve a fitness problem, trying to emulate a path like the movement of a flock of birds in indigenous surroundings unconsciously using this technique in food search. The particles start assessing their quality or wellness at a particular position as soon as they are put arbitrarily in the search space. At that point, every particle moves to a brand new location which produces a more preferable fitness over the past position. This movement relies on the history of particles personal best and current locations with those of the best and most effective positions achieved by different particles in the swarm, with some arbitrary fluctuations. Consequently, in ensuing iterations, the swarm accomplishes the most ideal answer for the function representing the fitness, with a particular number of particles cooperating with each other. The function that is to be optimized in the PSO algorithm is an exhibition assessment criterion that relies upon the application territory of the calculation. The mathematical definition typically outlines the performance basis to measure the framework execution accomplished through an exhibition list. Continuous and rigorous advancement as it happens in nature, spurs the genetic evaluation methodology and other populace based pursuit strategies. However, PSO, unlike most different strategies, depends on analogies with social conduct of creatures and fowls. Selection task is not at all involved in PSO calculation and in this way, the hunt procedure holds every one of the swarm particles throughout the operation. The position and speed of every one of the particles is refreshed in each iteration, as per the particles own claim and group's best positions achieved till that point of time. Consequently, in contrast to other developmental techniques, PSO never actualizes the arrangement of the survival of the fittest theory.

1.3 Basic Particle Swarm Optimization Algorithm

This calculation comprises of n particles which make up a swarm, along with every particle's position showing a potential answer to the d dimensional search area and its fitness function whose value is variably optimised. The particle updtes its location affected by essentially three ele-

ments:

- The self inertia
- Most optimal solution of the particle itself
- Most optimal global solution of the swarm

$$v_i, d = \omega v_i, d + c_p r_p (p_i, d - x_i, d) + c_g r_g (g_d - x_i, d)$$

$$x_i = x_i + v_i$$

Where ω is the inertia of particle i in the d dimension c_p and c_g are acceleration constants. r_p and r_g are random values in the range $[0, 1]$. $p_{i,d}$ speaks to the individual best position of the i^{th} molecule and g_d speaks to the worldwide best position.

$x_{i,d}$ rgive the present position of the i^{th} particle in the d^{th} dimension. The velocity updation statement in PSO comprises of 3 sections:
Momentum: t shows the propensity of the particle to manoeuvre in a similar way as it was going in the past cycle. It appends the impact of the past velocity on present speed of the particle.
Cognitive part: It shows the affinity of the particle's velocity towards its own best location (pbest). Alluded to as "memory", "self-information" or "recognition".
Social part: It constitutes the effect of the swarm's best position (g_{best}) obtained till that time on the current velocity. Also known as "collaboration", "social learning" or "shared data".

1.3.1 Inertia Weight

Particle's velocity on each dimension depends on a most extreme values in the search space, Vmax. This highest speed decides the goals or purpose with which areas between the current position and the objective position (best up until now) can be dealt with. The utilization of hard limits introduces a few issues. For example in the event that Vmax is excessively high,

at that point particles may disappear from some important areas ,and on the off chance that Vmax is excessively low, at that point some good locations will be unreachable. Accordingly so as to lessen the significance of Vmax, and to hone the searching capacity of particles, a weight term is added to the PSOs conditions. This is called inertia weight (ω) and it controls the impact which the previous iteration velocity has on the present value. Bigger estimation of ω improves worldwide hunt ability and the partial search capacity of PSO technique is improved by lesser value of ω. By and large, it is equivalent to 1, however in the end the inquiry capacity diminishes and the particles stall out at a non-ideal area. In experimental work, w is kept in the middle of 0.9 to 0.4 and the values are linearly decreased so the calculation enables the particles to investigate more extensive regions in starting and adjacent regions in later stages with decreased rates. This setting gives a more noteworthy probability of achieving the objective ideal position rapidly. In the event that we translate:

$$c_p r_p(p_i, d - x_{i,d}) + c_g r_g(g_d - x_i, d)$$

as the external power f_i, that the particle encounters, at that point the adjustment in a particle's speed (i.e., the acceleration gained by the particle) can be composed as;

$$\Delta v_i = f_i(1-\omega)v_i v_i = f_i - (1-\omega)v_i$$

The component 1 - ω acts adequately as a contact coefficient. Thus ω is said to be the ease of movement (fluidity) of the medium wherein a particle moves. Along these lines the best execution is obtained by setting ω at first to some generally high number (e.g., 0.9).the higher influence of ω relates to a framework where particles move in a medium with low viscoity and perform broad investigation. At the point when ω is slowly decreased to a much lower number (e.g., 0.4), the framework turns out to be increasingly exploitative and dissipative and would be better at homing into nearby optima. It is also conceivable to begin from estimations of $\omega > 1$, however the swarm will become insecure and doddery until and except if the value is diminished adequately to acquire the swarm a steady location

1.3.2 Acceleration constants C_p and C_g

These constants are associated with the speed with which the particles fly to the most optimal position of swarm and its very own best position. They control the distance and time spent by particle to achieve the most ideal position. These constants must be appropriately chosen to ensure that the particle lands in a right position. For too huge an estimation of these constants, the right position may get bypassed by the particle and for too little a value, the particle almost certainly not achieve the objective position. By and large every one of these constants are kept at the numerical value of 2 to make the iterations taken to manoeuvre towards the swarm's global best and the particle's own best as equivalent and total time is halved. These constants give the weighing of these terms towards Gbest and Pbest areas.

1.3.3 Random numbers R_p and R_g

The draw on the particles towards Gbest and Pbest locations are managed by including arbitrary numbers in the update rules. These are irregular fictitiously defined values and decide the magnitude of arbitrary powers towards the 2 most optimal positions. A random segment is added to the PSO calculation by them and they help keep the calculation from stalling out at a non-ideal nearby least or greatest arrangement.

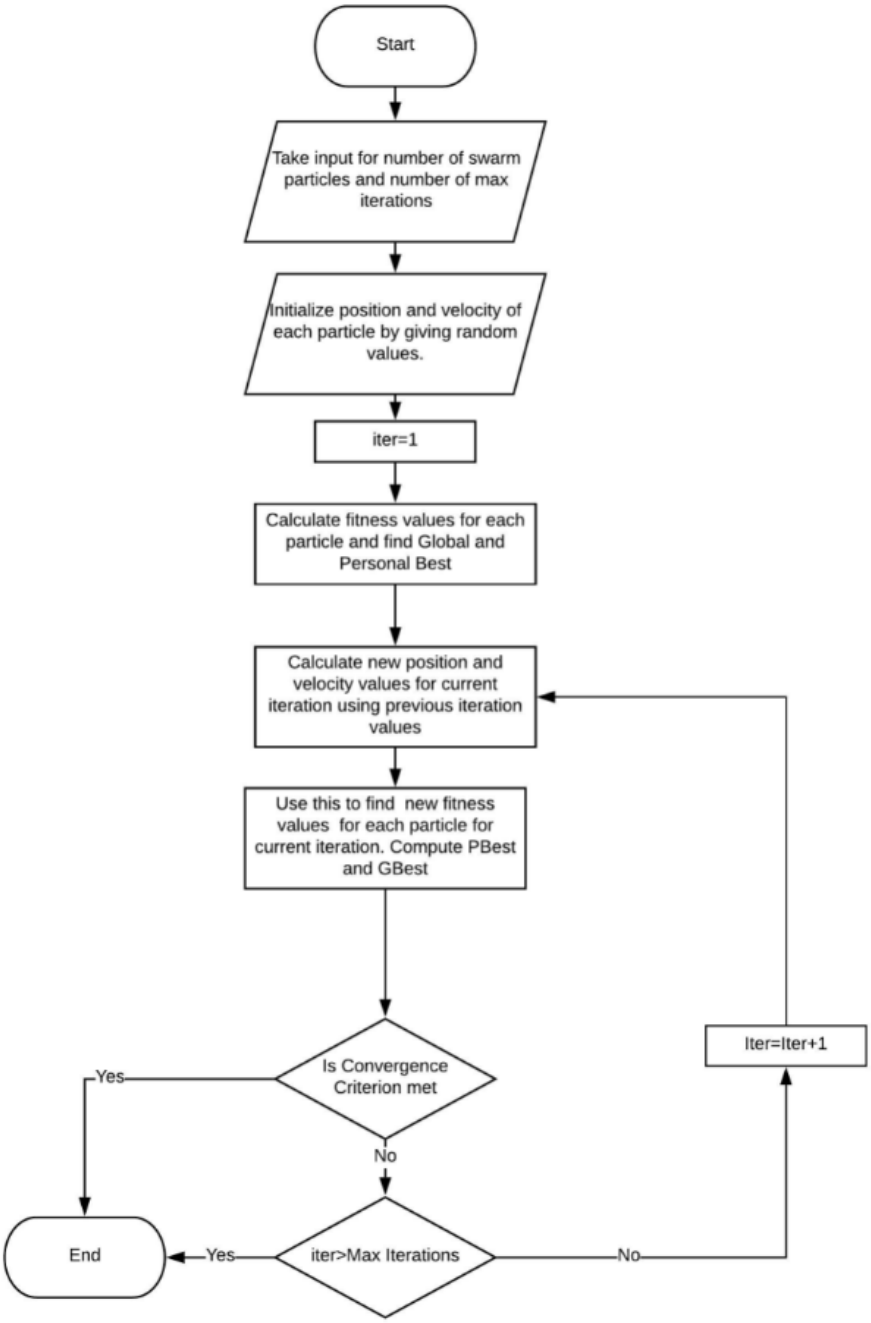


Figure 1.1: Flowchart of PSO Algorithm

Advantages

- PSO calculation does not include selection activity or transformation calculation.
- The hunt can be done by changing the particle's speed over and over again. By gaining valuable insights from group's location and position history, particles fly only to great regions (where there is a high plausibility of discovering food).

- PSO depends on artificial intelligence and thus, can be used in both logical engineering operations and research as well as scientific and technological applications.
- Simple estimations are associated with PSO calculation and with the advancement of more up to date assessment methods, they are done easily and inn a lot less time.

Disadvantages

- Standard PSO experiences a significant increase in the complexity of search operations with even a slight increase in the dimension or size of search space.
- The technique is defenceless against fractional confidence in the operation, which prompts a considerably less exact guideline of its speed and the direction.
- Due to the absence of dimensionality this strategy can't be utilized for analysis of the non-coordinate framework, for example, the problem pertaining to the energy field and the moving principles of the particles in this field.

CHAPTER 2

Laboratory Application of PSO

2.1 RosenBrock Function

The Particle Swarm Optimization Algorithm was used to find the minimum value of the Rosenbrock function.

In mathematical optimization, the Rosenbrock function is a non-convex function, which is applied as a performance test problem for optimization algorithms. It is also known as Rosenbrock's valley or Rosenbrock's banana function.

The global minimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. The function is defined by

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

It has a global minimum at $(x, y) = (1, 1)$ where $f(x, y) = 0$.

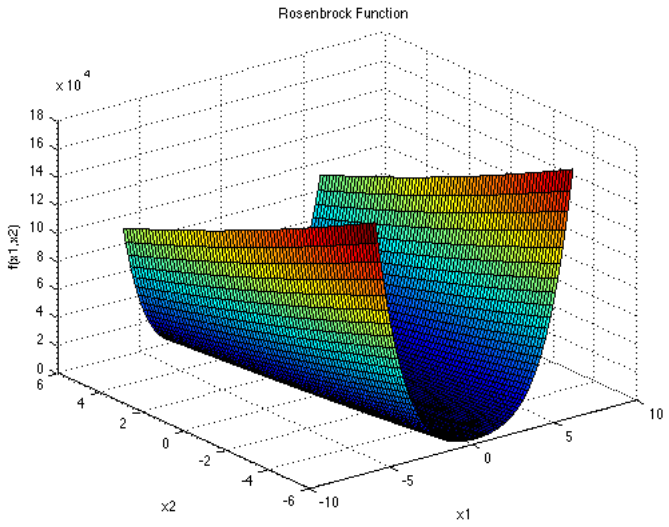


Figure 2.1: Graphical representation of Rosenbrock function

The MATLAB code for the implementation of PSO algorithm has been given in the appendix at the end. The results obtained were then analysed on the basis of variation in the various parameters.

2.2 Analysis Of Results

2.2.1 Analysis of Results with Variation in Random Variables and Acceleration Constants

Effect of variation of parameters on Results:

- Number of particles = 10
- Iterations=100
- Tolerance=10^{-4}
- $C_p = 2$
- $C_g = 2$

R_g	n	x_1	x_2	f_{min}
0.1	**51**	**1.000017**	**1.000041**	**6.779 x 10^{-9}**
0.2	55	0.810600	0.655274	3.61 x 10^{-2}
0.3	50	0.780632	0.607469	4.849 x 10^{-2}
0.4	50	0.913663	0.834657	7.45 x 10^{-3}
0.5	53	0.889621	0.790527	1.23 x 10^{-2}
0.6	58	0.752368	0.565055	6.14 x 10^{-2}
0.7	51	0.979989	0.960116	4.073 x 10^{-4}
0.8	55	0.858121	0.736081	2.01 x 10^{-2}
0.9	59	0.801204	0.640638	3.968 x 10^{-2}
1.0	56	0.933261	0.870666	4.463 x 10^{-3}

Table 2.1: VARIATION OF RESULT ACCORDING TO R_g BY KEEPING $R_p = 0.2$

Observation: According to the above table we observe that when we kept the value of $R_p = 0.2$ and varied the value of R_g, the most optimum solution is obtained at $R_g = 0.1$ after 51 iterations; i.e. $6.779x10^{-9}$. Whereas for the other values of R_g, we obtain vague values as result.

- Number of particles = 10
- Iterations=100
- Tolerance=10^{-4}
- $C_p = 2$
- $C_g = 2$

R_g	n	x_1	x_2	f_{min}
0.1	54	0.821723	0.673536	3.204 x 10^{-2}
0.2	38	0.860219	0.384371	1.977 x 10^{-2}
0.3	46	0.851775	0.724470	2.209 x 10^{-2}
0.4	50	0.896989	0.803815	1.061 x 10^{-2}
0.5	55	0.833493	0.692122	2.839 x 10^{-2}
0.6	43	0.834538	0.695015	2.758 x 10^{-2}
0.7	**54**	**1.00011**	**1.00002**	**1.208 x 10^{-10}**
0.8	53	0.999992	0.999982	4.189 x 10^{-10}
0.9	47	1.000200	1.000405	4.393 x 10^{-5}
1.0	60	1.000916	1.001883	1.095 x 10^{-6}

Table 2.2: VARIATION OF RESULT ACCORDING TO R_g BY KEEPING $R_p = 0.4$

Observation: According to the above table we observe that when we kept the value of $R_p = 0.4$ and varied the value of Rg, the most optimum solution is obtained at $R_g = 0.7$ after 54 iterations; i.e. $1.028x10^{-10}$. Whereas for the other values of R_g, we obtain vague values as result.

- Number of particles = 10
- Iterations=100
- Tolerance=10^{-4}
- $C_p = 2$
- $C_g = 2$

R_g	n	x_1	x_2	f_{min}
0.1	55	0.768956	0.588862	5.397 x 10 $^{-2}$
0.2	50	0.892328	0.795982	1.161 x 10 $^{-2}$
0.3	53	0.827069	0.682662	3.01x 10 $^{-2}$
0.4	47	0.956819	0.914889	1.903x 10 $^{-3}$
0.5	**51**	**1.00045**	**1.00088**	**2.436x 10** $^{-7}$
0.6	58	0.999475	0.998774	3.377x 10 $^{-6}$
0.7	55	0.999511	0.999031	2.491x 10 $^{-7}$
0.8	57	0.999688	0.999465	8.895 x 10 $^{-7}$
0.9	55	1.000291	1.000526	3.977x 10 $^{-7}$
1.0	100	1.001326	1.002635	1.792x 10 $^{-6}$

Table 2.3: VARIATION OF RESULT ACCORDING TO R_g BY KEEPING $R_p = 0.6$

Observation: According to the above table we observe that when we kept the value of $R_p = 0.6$ and varied the value of R_g, the most optimum solution is obtained at $R_g = 0.5$after 51 iterations; i.e. $2.436x10^{-7}$. Whereas for the other values of R_g, we obtain vague values as result.
The highlighted set of observations represent the best and minimum value of the Rosenbrock obtained at the corresponding values of the random variables and independent variables $x1$ and $x2$.

- Number of particles = 10
- Iterations=100
- Tolerance=10^{-4}
- $C_p = 2$
- $C_g = 2$

R_p	R_g	n	x_1	x_2	$Kount$	f_{min}
0.2	**0.1**	**51**	**1.000017**	**1.000041**	**1031**	**6.779 x 10^{-9}**
0.4	0.7	54	1.00011	1.00002	1091	1.208 x 10^{-10}
0.6	0.5	51	1.00045	1.00088	1031	2.436 x 10 $^{-7}$

Table 2.4: EFFECT OF VARIATION OF PARAMETERS ON CONVERGENCE TIME IN TERMS OF FUNCTION EVALUATIONS

Observation: Thus by analysing different pairs of R_g and R_p we observe that we get the most optimum solution at $R_p = 0.4$ and $R_g = 0.7$. Although the convergence time in this case(1091) is greater than others(1031), we can compromise with it since we are focused on getting better solution.

CHAPTER 3

Improvements in PSO

3.1 Constant Inertia

Here, after comparing the results obtained by keeping different values of inertia i.e. constant weight, we chose the one which gave us the optimum solution. The optimum value was found to be at $\omega = 0.6$.

The code has been given in Appendix at the end for reference.

3.2 Positional Weightage PSO

From the previous methods, we could see that before the particle swarm approached the true solutions, each individual particle made a high number of vibrations, i.e. the particle's value oscillated about the true solutions before the particle finally converged there.

We also know that in order to make the PSO program more efficient, we have to reduce the number of iterations.

Thus, in order to achieve this, we aim to reduce the vibrations. These can be reduced if all the particles were repositioned after some time, so that each particle was closer to true solution. This reduces the vibrations of each individual particle as well as the effect of one particle on others that caused others to stray away from true solution..

The algorithm followed is:

1. Let program run normally till a fixed number of iterations. (Say 14)

2.or each particle, take into account the value that each particle gives throughout these 14 iterations. Assign weight to each position depending upon the value and kind of solution we need(Maxima/minima). Higher the weight,

better will be the position.
3. Finally, take the weighted mean and continue the program to run as normal.

This approach not only reduces the vibrations, but also prevents a possible solution part of the function, that in normal case might be left out or be reached after longer time, to be taken into account.

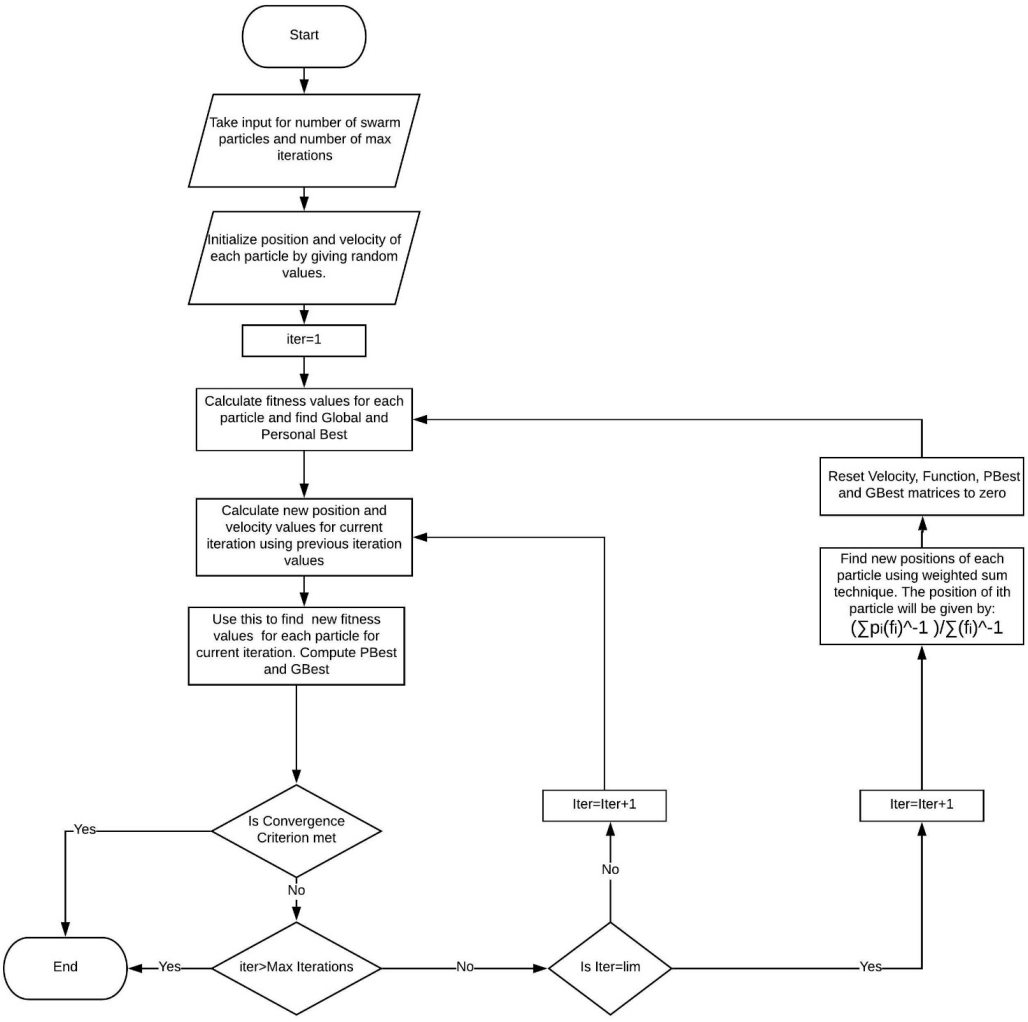


Figure 3.1: Flowchart for Positional Weightage Algorithm

x_1	x_2	F	It	$Kount$	$Nomenclature$
0.9603553	0.9209168	0.001758157	41	831	Normal
0.9999969	0.9999972	1.17099*10-9	34	691	Constant Inertia
1.000843	1.001637	9.47243*10-7	23	481	Positional weightage PSO

Table 3.1: COMPARISION OF RESULTS OBTAINED FROM DIFFERENT METHODS

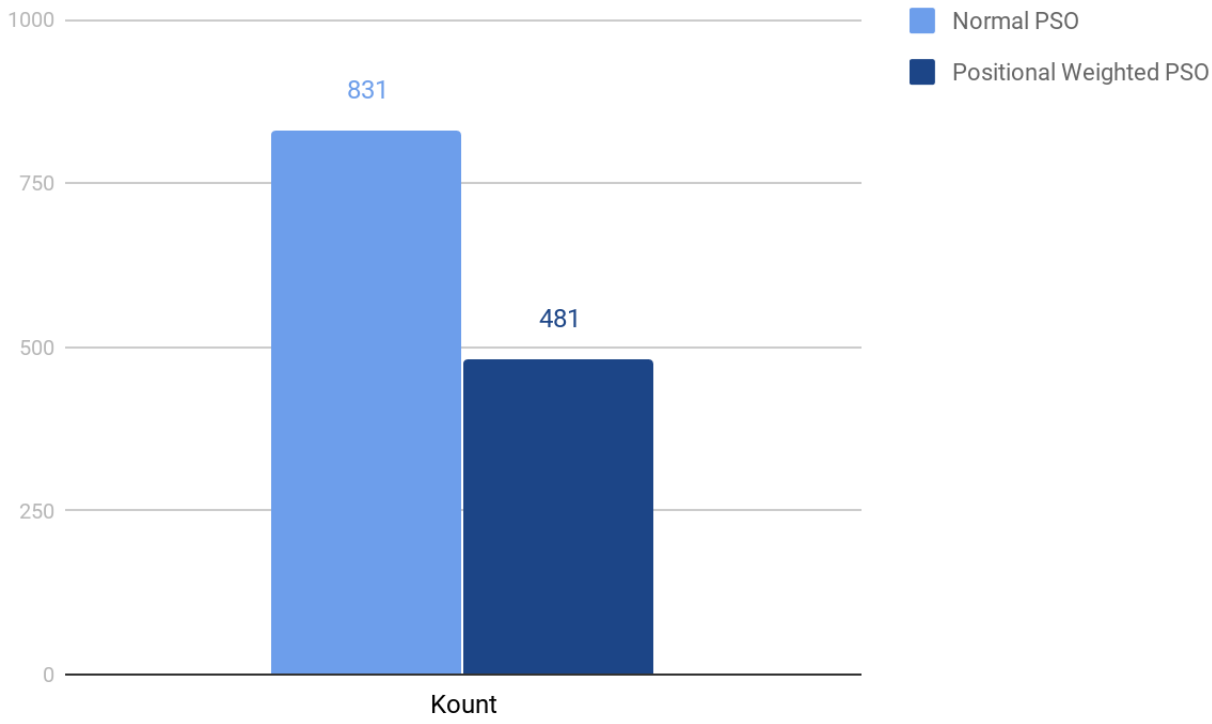


Figure 3.2: COMPARISON BETWEEN THE NORMAL AND THE POSITIONAL WEIGHTED PSO.

Graphical Visualization of Trajectory of Different Particles

The trajectory of each particle before finally converging to a minimum value is depicted along with vibrations in the results obtained for the function with the number of iterations for all the 3 methods. The first two graphs in each particle are for normal and fixed inertia methods and the subsequent ones denote the positional weighted average method.

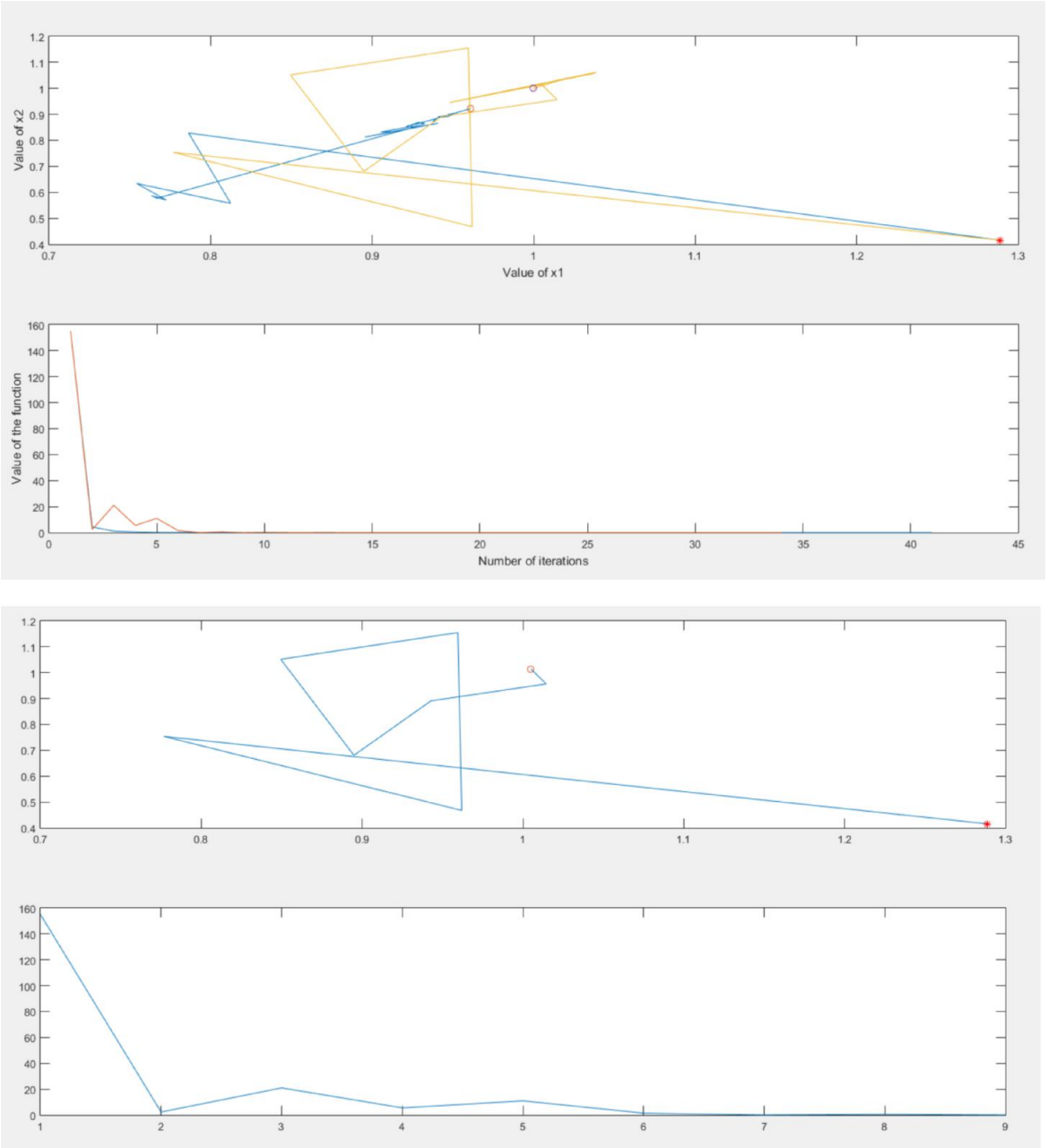
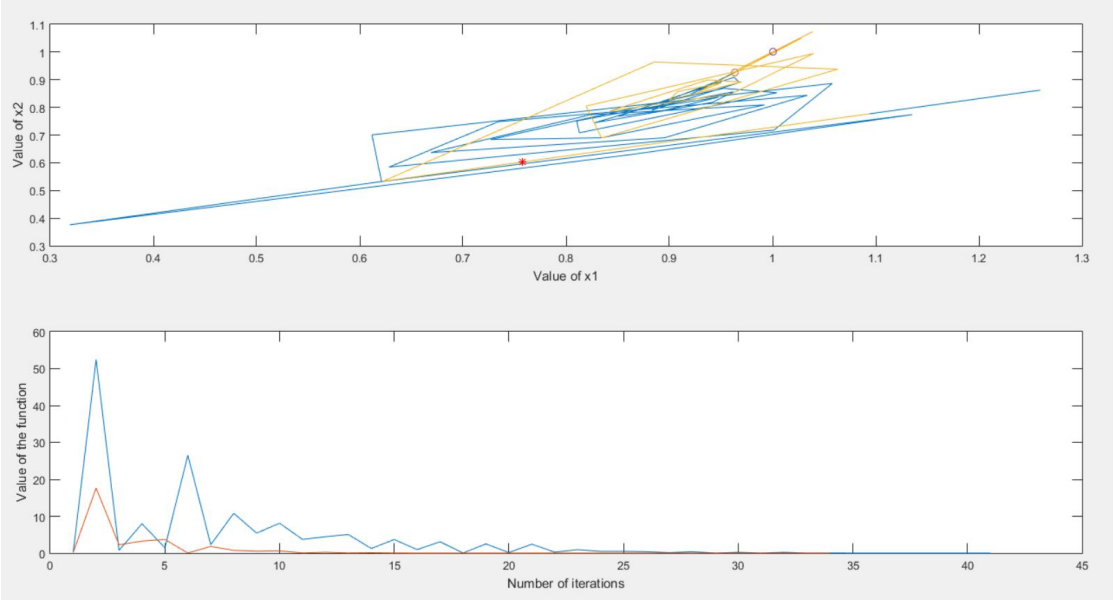


Figure 3.3: Particle 1



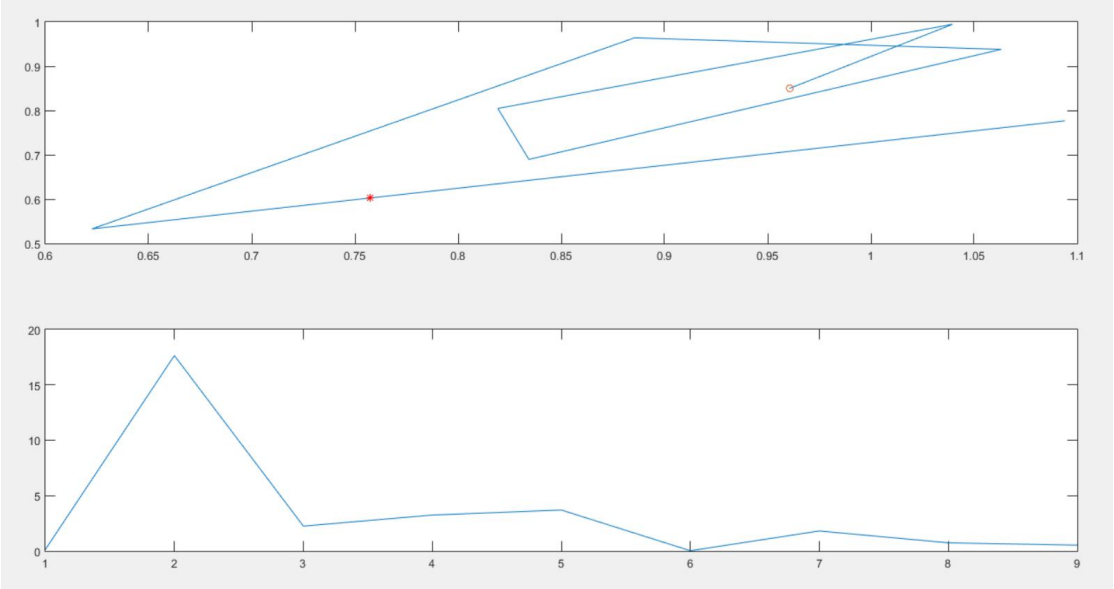
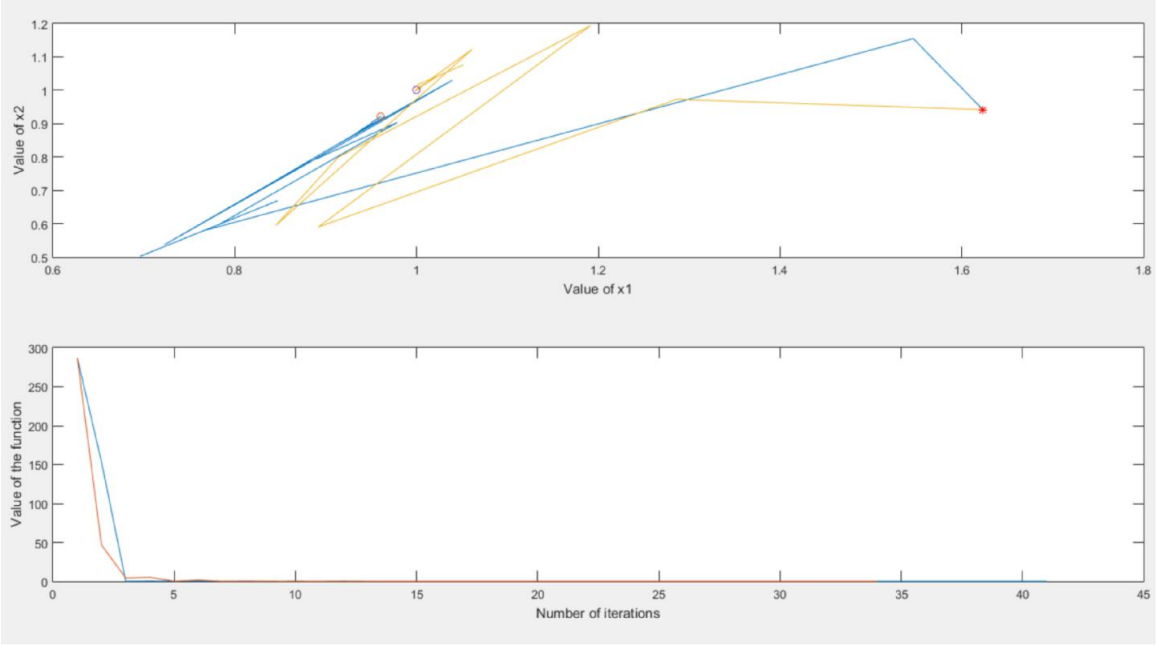


Figure 3.4: Particle 2



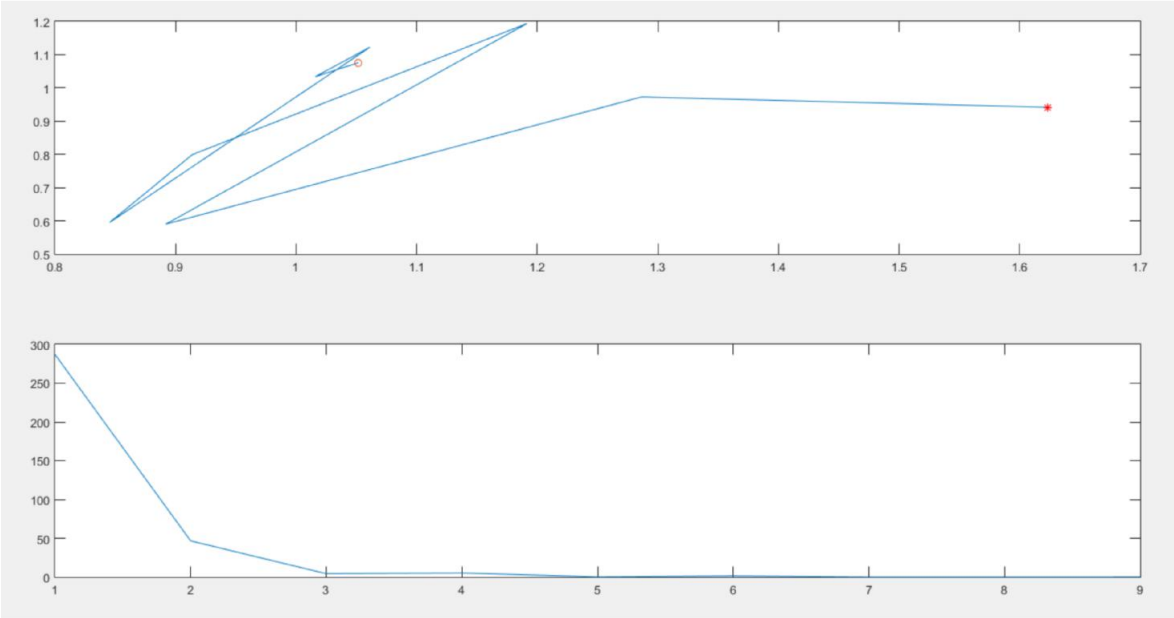
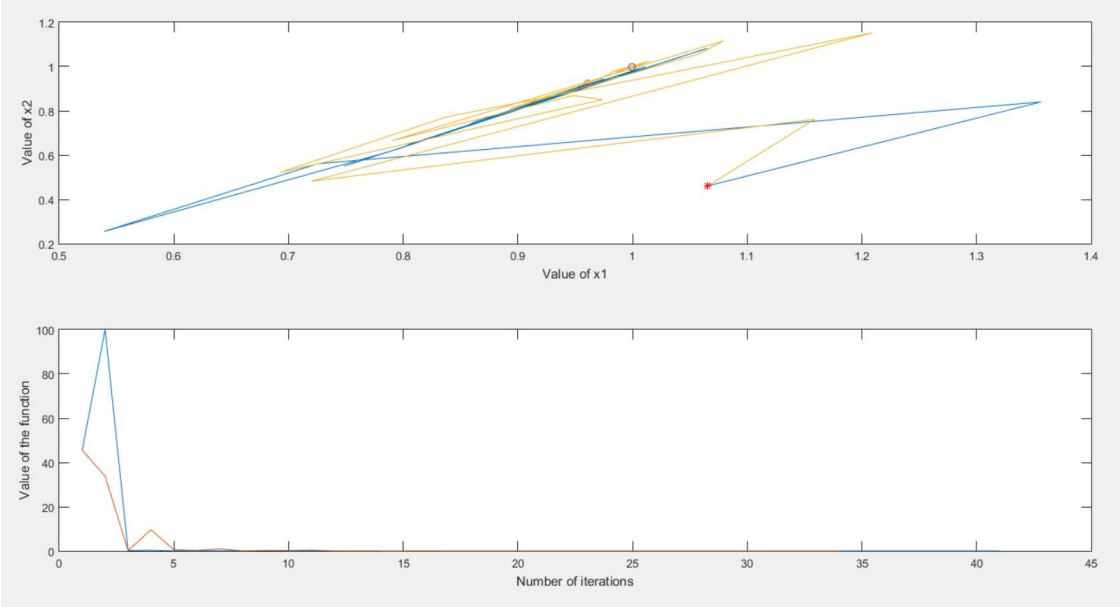


Figure 3.5: Particle 3



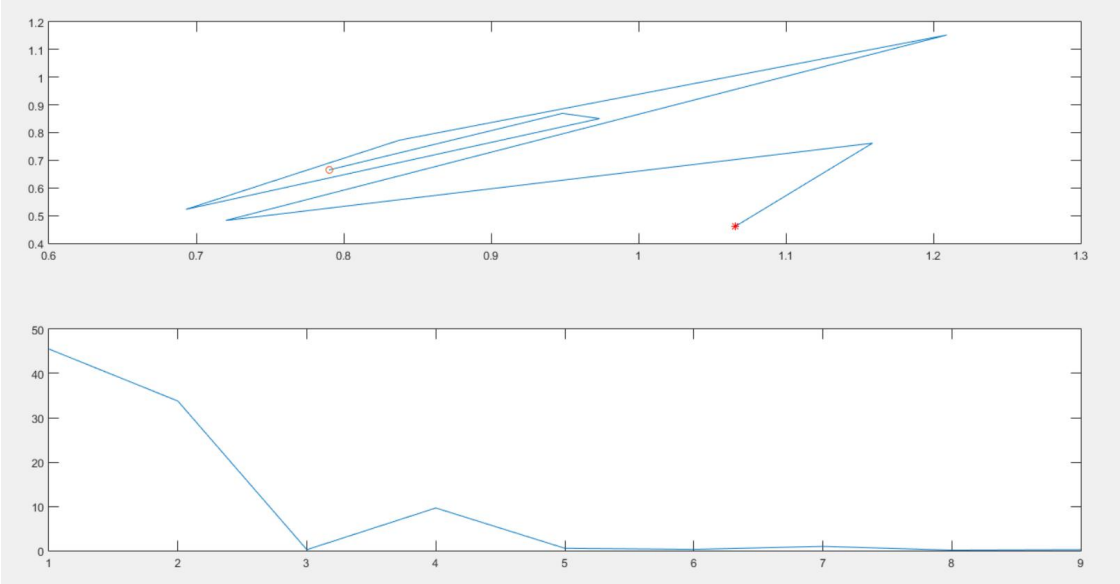
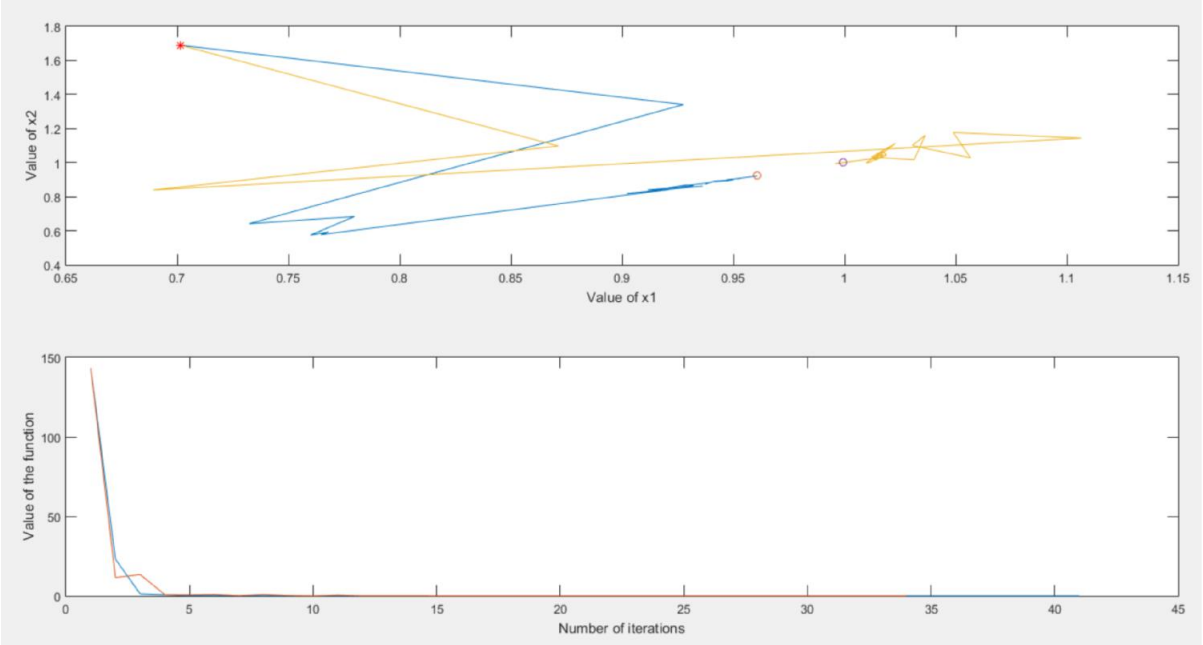


Figure 3.6: Particle 4



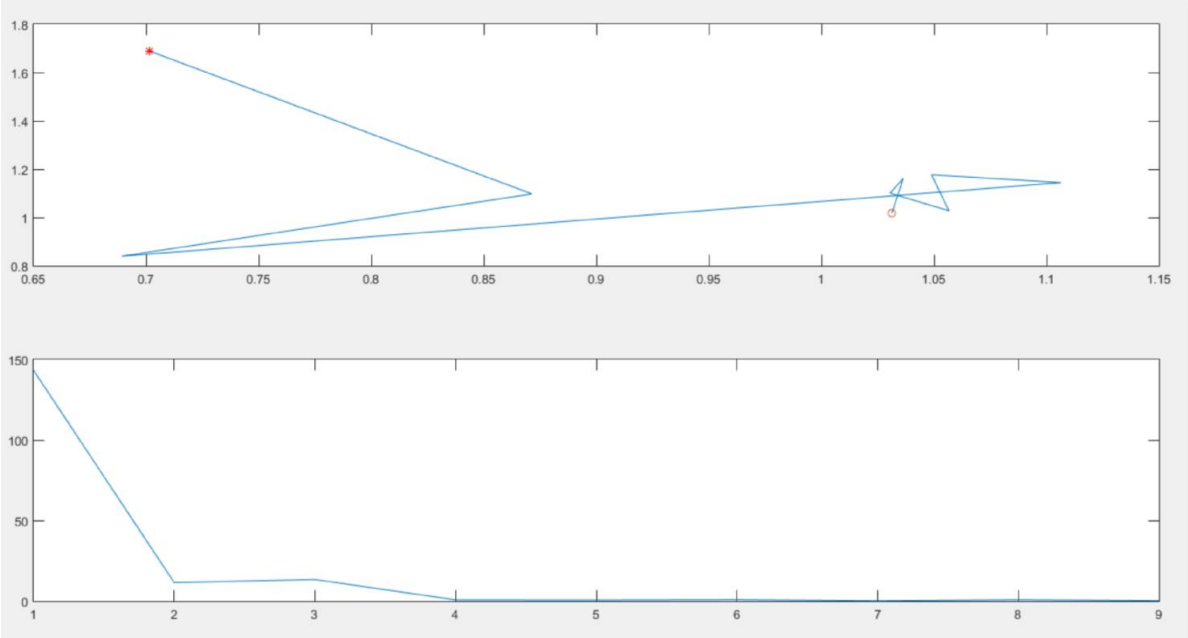
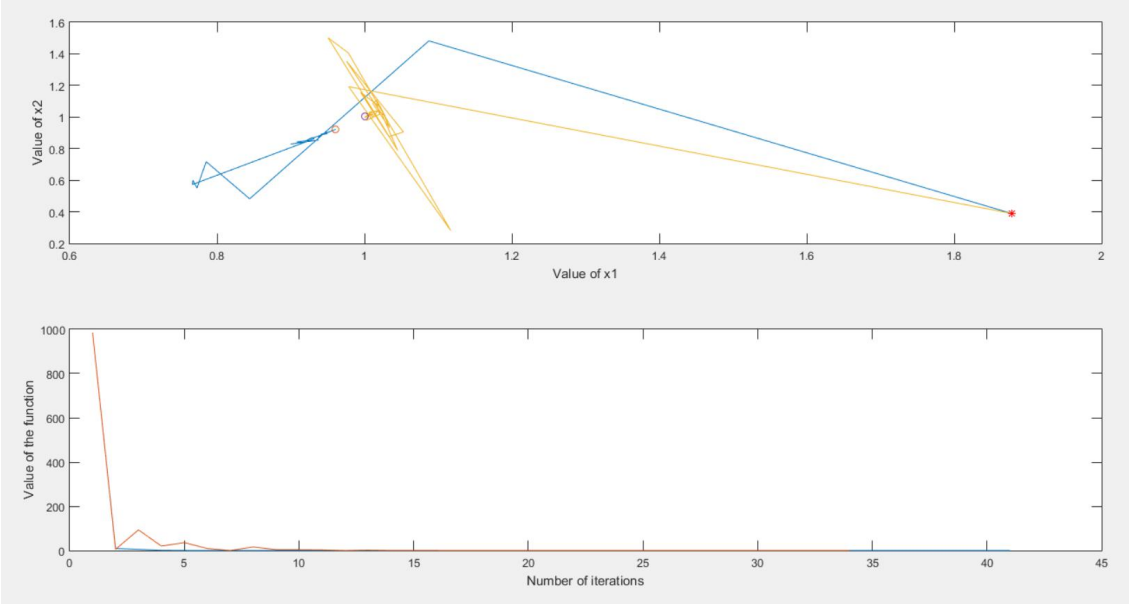


Figure 3.7: Particle 5



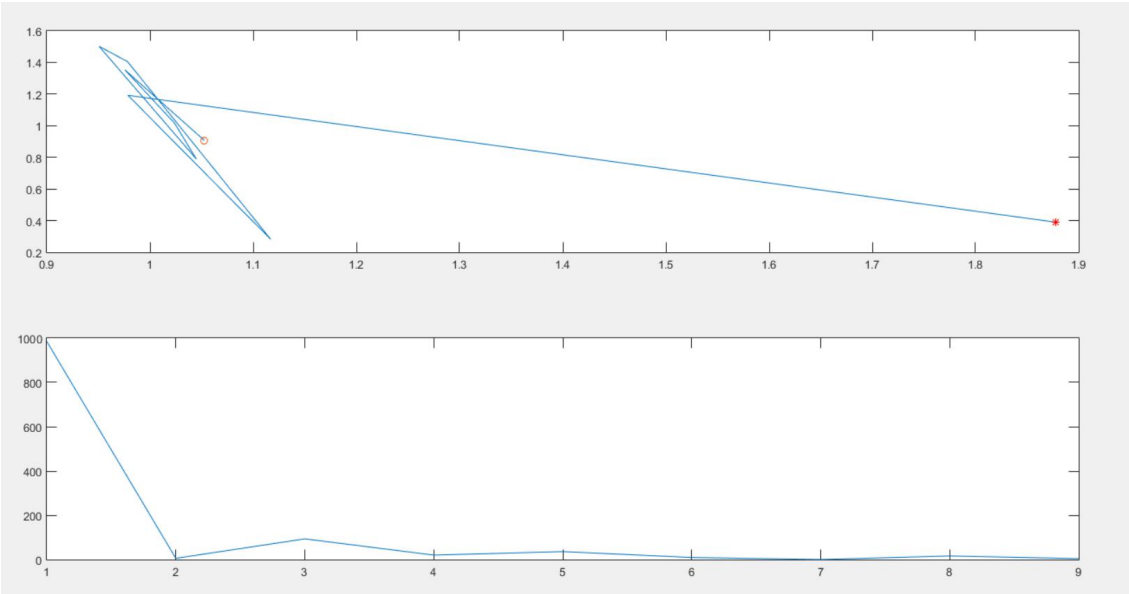


Figure 3.8: Particle 6

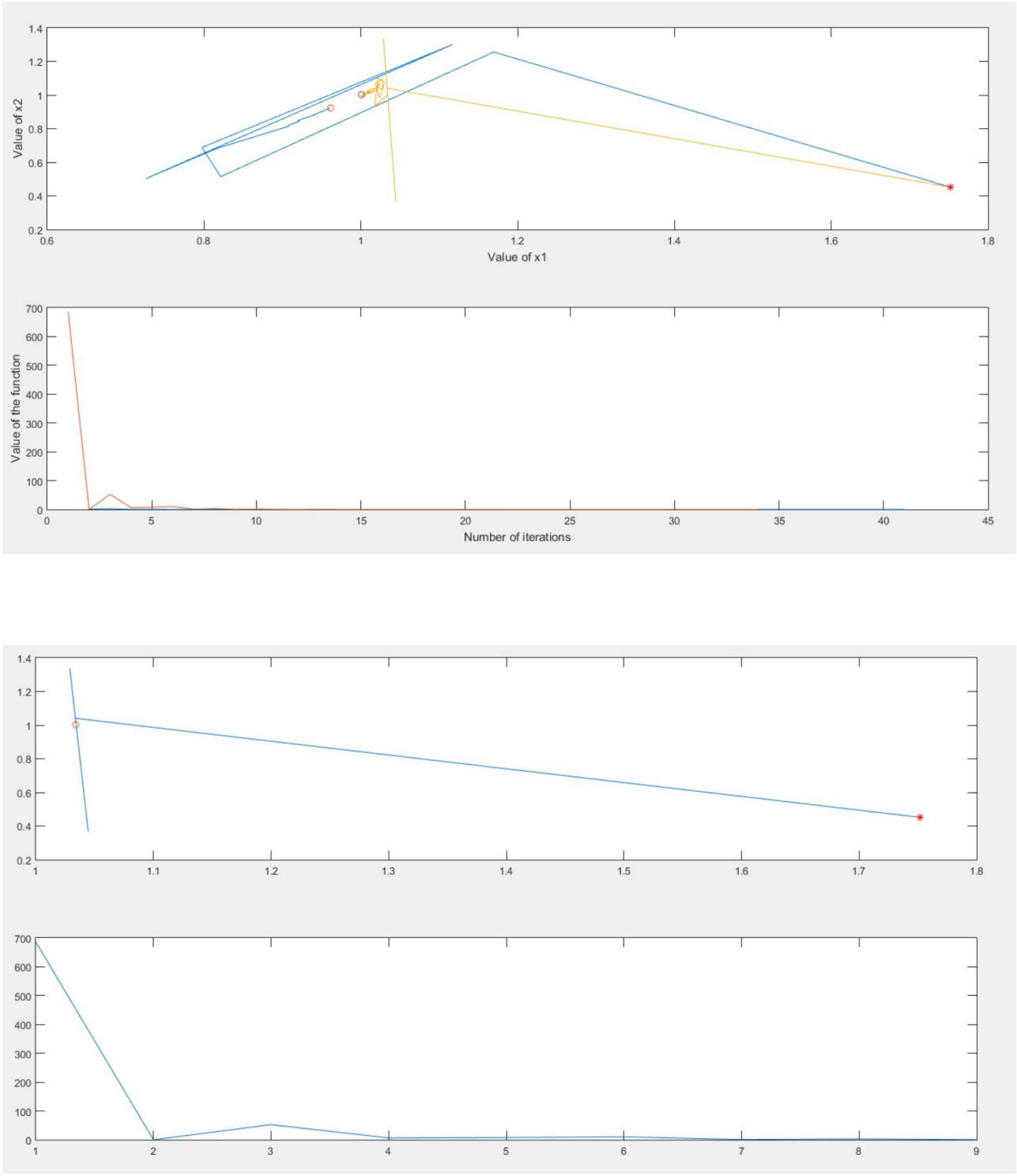
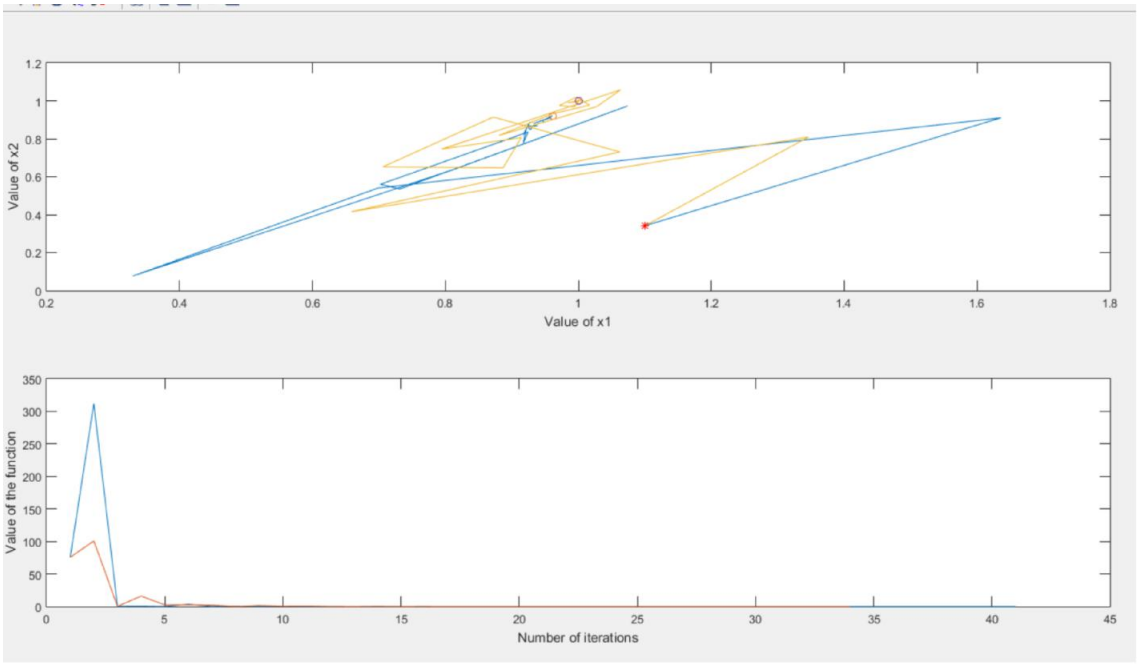


Figure 3.9: Particle 7



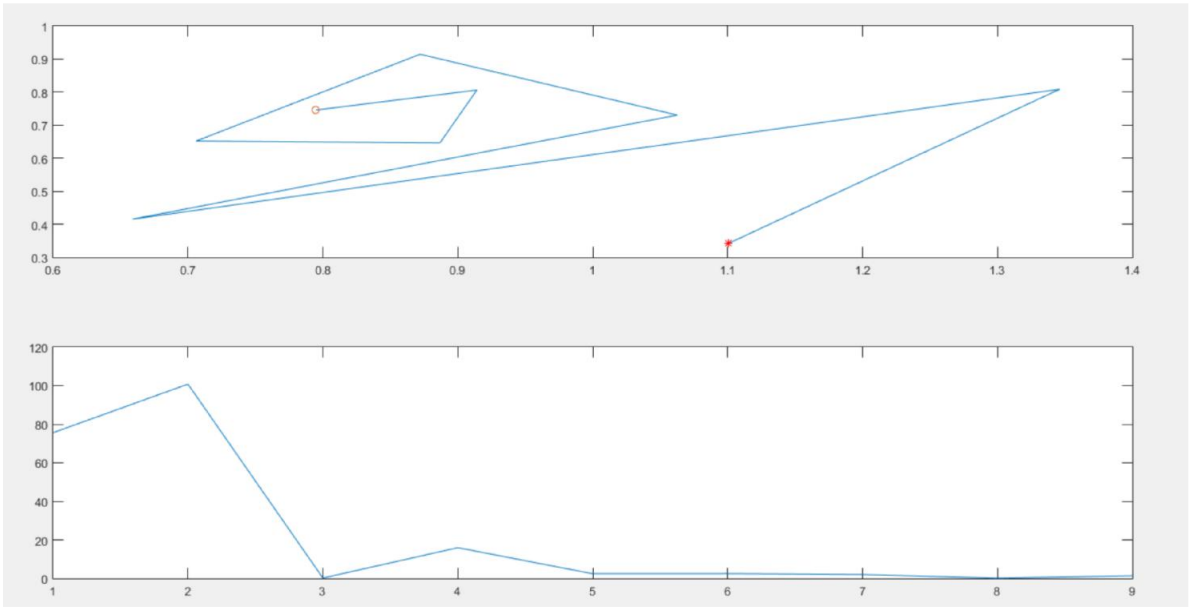
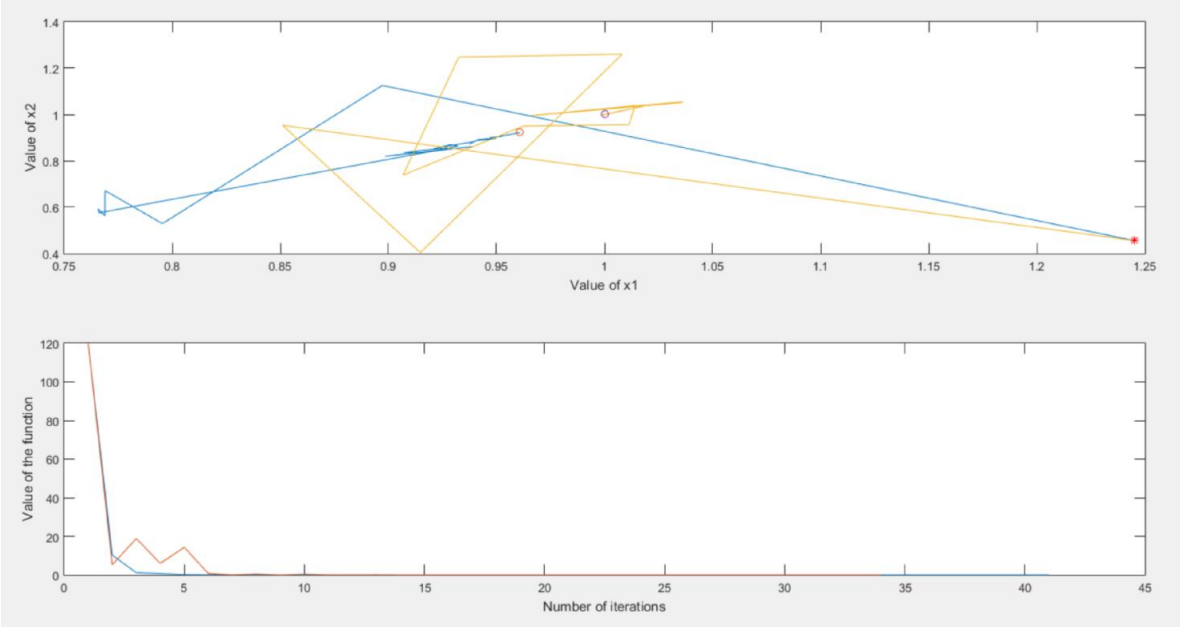


Figure 3.10: Particle 8



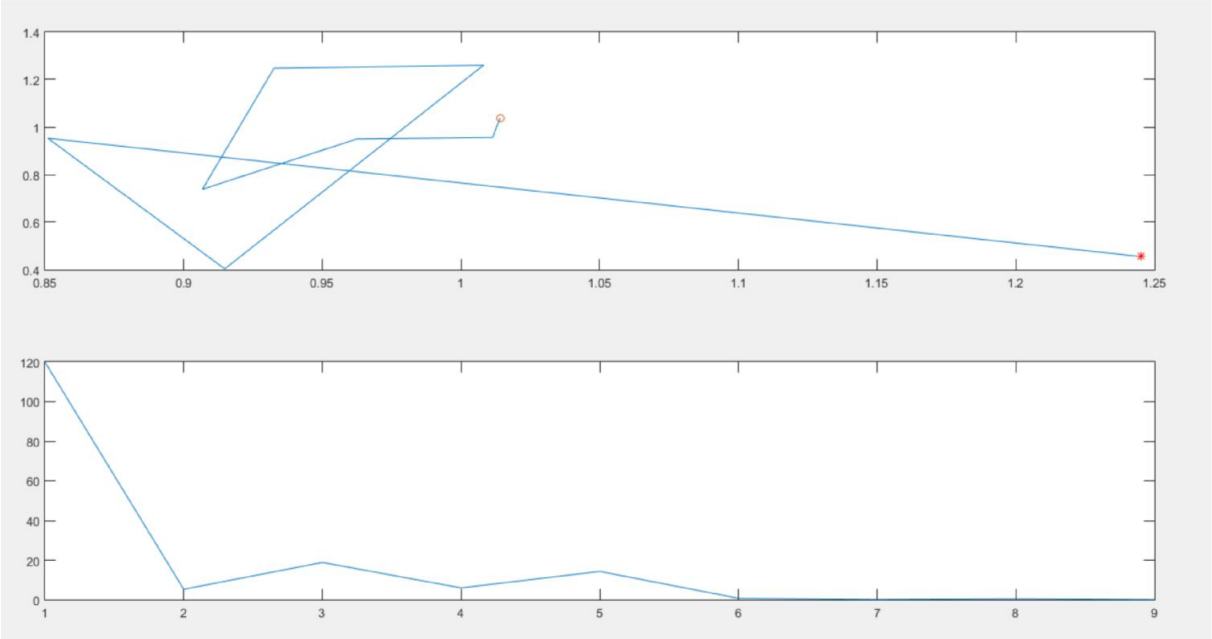
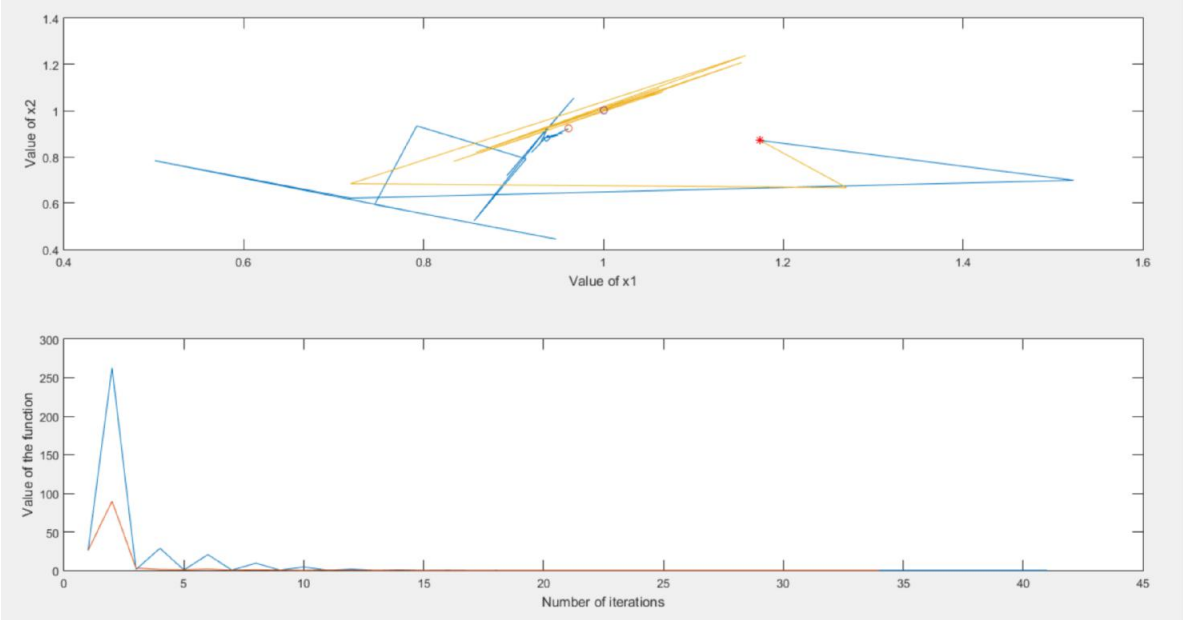


Figure 3.11: Particle 9



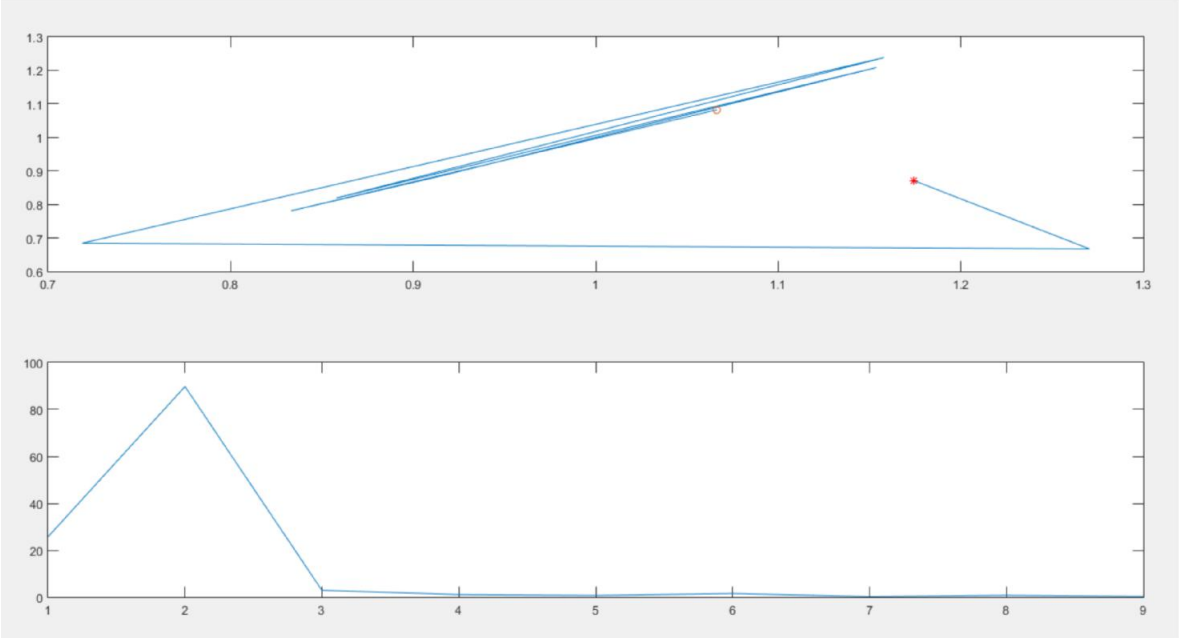


Figure 3.12: Particle 10

3.3 Conclusion

The star is the initial point for the particles and the circle denotes the last iteration at which the Rosenbrock function attains the minimum value i.e. near to zero. At this point the values of x1 and x2 are close to 1.

At the initial stages, the positions of the particles at each iteration are far apart from each other because initially the velocity of the particles is high. When the function starts approaching its optimum value, the velocity slows down and the points come closer to each other.

Varying the inertia weight ω, greatly affects the trajectory of the particles. The best value was found to be at ω=0.6. The values of x1 and x2 also converge at a value that is closer to 1 i.e. the actual solution.

As the value of the function approaches 0, the particles start vibrating or oscillating between roughly two nearby values till they reach the optimum point.

The particle 3 seems to be the best particle because the value of the function here along with the values of x1 and x2 converge to their ideal values quicker than the others.

Particle 2 is the worst particle as the values of the function as well as x1 and x2 are most far apart from each other in successive iterations and fluctuate a lot more as compared to the others.

CHAPTER 4

Analysys of Load Flow Studies using PSO: Field Application

4.1 Analysis for Load Flow Studies

n electrical engineering, the power flow study, also known as load flow study, may be used as a numerical tool in the analysis of the flow of electrical power in an interconnected load bus and generator bus system. A power flow study typically uses simplified notations and methods like a one-line diagram and per-unit system and focuses on numerous aspects of AC power parameters, like voltages, voltage angles, real power, and reactive power. It analyses the ability of systems in traditional steady state operation.

Load-flow studies are necessary for coming up with the long run enlargement of power systems along with determining the most effective operation requirements of existing systems. The principal data obtained from the power-flow study is the magnitude and phase angle of the voltage at every bus, and therefore the real and reactive power passing in every line.

Commercial power systems are typically too complicated to permit for manual resolution of the power flow study. For this purpose, between 1929 and the early sixties, special network analysers were engineered to produce laboratory scale physical models of power systems. Analogue ways with numerical solutions were replaced by large scale digital computers owing to a boom in technology and science..

In addition to a power flow calculative effort, some other connected functions like short-circuit fault analysis, stability studies (transient and steady state), unit commitment and economic load dispatch are also effec-

tively performed by computer programs. Specifically, some coded algorithms use applied math to search out the best power flow, the conditions that provide an all-time low price per kilowatt hour delivered to the arrangement.

A load flow study is particularly valuable for a system with more than one load centres like a complex refinery etc. Analysis of the system's ability to adequately offer and supply the connected load is done using power low study. The whole system losses, added together as individual line losses, are also tabulated in a proper form. Working with a load flow study provides insights and proposals on the system operation and improvement of management settings to get the most capability whereas minimizing the operational prices. The outcomes of such analysis are measured in terms of active power generated and injected, reactive power generated and injected, their magnitudes, and phases. Moreover, power-flow results and solutions obtained are extremely important for optimal operations of teams of generating units.

In term of its approach towards irregularities, load flow study is divided into deterministic load flow and load flow handling uncertainties. The uncertainties that arise from each power generation cycle and unorganized load behavioural changes are not taken into consideration in deterministic load flow studies. To take the uncertainties into thought, there are many methods employed like possibility, data gap call theory, probabilistic, interval analysis, and strong improvement.

Load flow problem is iterative, utilizing the Gauss-Seidel methods or the Newton-Raphson techniques. The Newton-Raphson method, or Newton method, is an incredible system for unravelling conditions numerically. Like most of the infinitesimal calculus, it's supported the easy plan of linear approximation. But, in recent times, there has been abundant interest within the application of random search ways, like Genetic Algorithms or particle swarm optimization technique to resolution power grid issues. This problem typically targets to get the system buses voltages – module and angle – so as to see later the ability changes within the generation buses and therefore the power flow within the system lines. The system operating within the steady-state, i.e., its parameters not varying with the time variation is provided by the load flow study. The power flow calculation

is additionally important pertaining to the setting of an optimal point of operation, concerning economy and quality. When the system has known steady state, it's attainable to approximate the quantity of power required to be generated necessarily to produce the ability demand and the power losses. within the system lines, furthermore the voltage levels should be unbroken inside the boundaries and full operations, besides the operations within the stability limit should be avoided. Load flow analysis depend on a powerful framework and its voltage investigation performed nodally.

The **Static Load Flow Equations (SLFE)** in its general form is characterised by:

$$P_i - jQ_i = y_{i1}V_1V_i * -y_{i2}V_2V_1 * -... - y_{in}V_nV_i* = 0 \tag{4.1}$$

where: $i = 1, ..., n$*, bus number;*
P_i *= active power generated or injected in the bus i;*
Q_i *= reactive power generated or injected in the bus i;*
$|V_i|$ *= voltage module of the bus i;*
δi *= voltage angle of the bus i;*
$Vi = |Vi|ej\delta i$
;
i = voltage angle of the bus i;

The subsequent explanation explains how the nodal admittance matrix is obtained: if $i = k$, y_{ik} is that the total of the admittances that startup of the bus i; and if ik, y_{ik} is that the admittance between the buses i and k, multiplied by -1. The power framework transports are characterized in sorts, as indicated by the factors known from the earlier and to the factors that will be gotten through the SLFE.
• Type 1 Bus or PQ Bus: P_i Q_i are indicated and —Vi— and δi are gotten through the SLFE;
• Type 2 Bus or PV Bus: P_i —Vi— are determined and Q_i and δi are acquired through the SLFE;
• Type 3 Bus or V Bus ("Slack Bus"): $|Vi|$ and δi are determined and P_i Q_i are acquired through the SLFE.

Equation plays out indirect and complicated conditions framework, and its answer is gotten by using estimations utilizing mathematical computational techniques. The Gauss, Gauss-Seidel and Newton Raphson strategies are regularly connected to tackle these frameworks. The most utilized technique is Newton Raphson because of its more noteworthy assurance and fast achievement of union. The strategies comprise in the appropriation of starting evaluated qualities to the transport voltages, module 1,0 [pu] and point 0 [rad], for example, and in the use of the SLFE in progressive cycles, hunting down finer estimations for the voltages. The stopping paradigm differs as indicated by the required exactness.

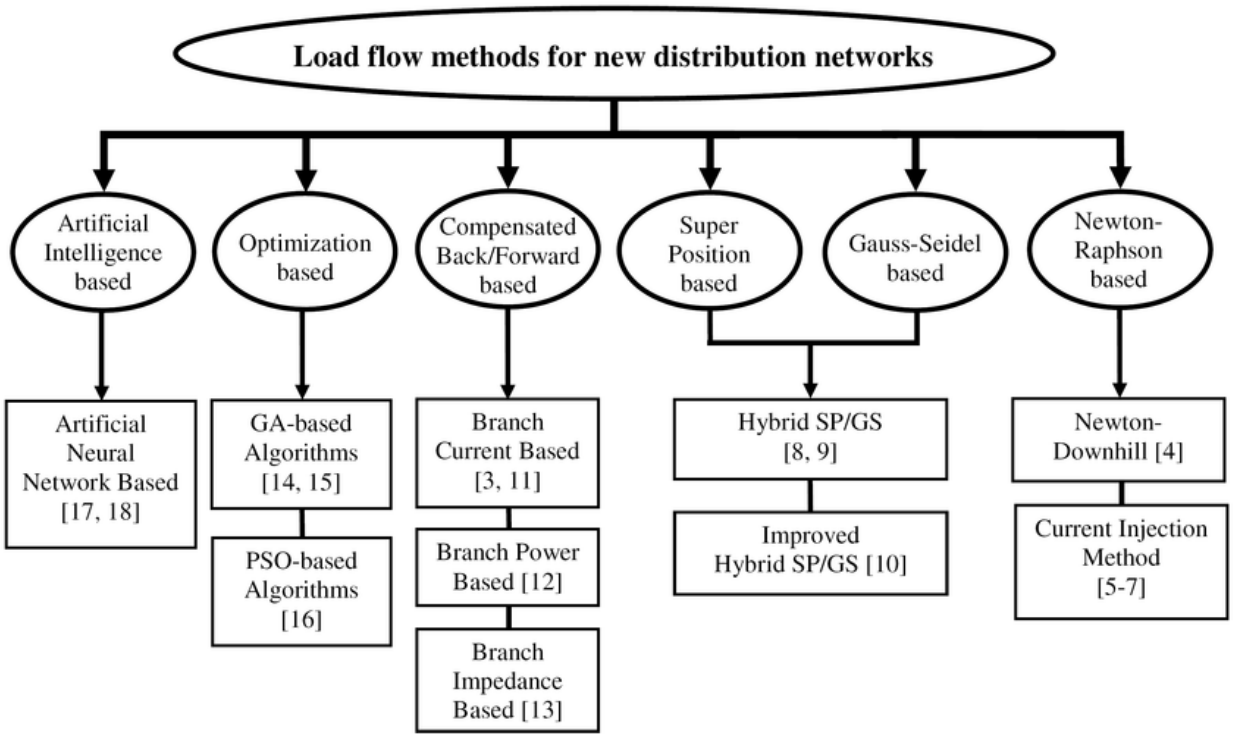


Figure 4.1: Loadflow methods

The conventional load flow methods used for power systems are as follows:-

1. Newton-Raphson (NR) method
2. Gauss-Seidel method with impedance matrix (ZGS)
3. Gauss-Seidel method with admittance matrix (YGS)
4. Fast Decoupled Newton-Raphson (FDNR)
5. Decoupled Newton-Raphson (DNR)

The extremely necessary load flow techniques, that can be applied to new distribution arrangements, are classified intoto 6 distinct areas: NR based strategies, Gauss-Seidel based techniques, superposition based strategies, remunerated in reverse/forward range techniques, enhancement based strategies and computerized reasoning based techniques. In our venture we expect to tackle the above issue utilizing an Evolutionary Algorithm i.e. **Particle Swarm Optimization**

4.2 Implementation Details

The PSO algorithm to a 5 bus system containing 3 load buses, 1 generator bus and 1 slack bus. The injection and generation of both active and reactive powers to and from each buses been calculated along with the cost of generation. The mismatch in the calculated and the specified values of the active and reactive powers has been taken as the objective function that is minimised using the Particle swarm optimization. The line losses in each line have also been calculated. The step by step flowchart along with the Output is given below:

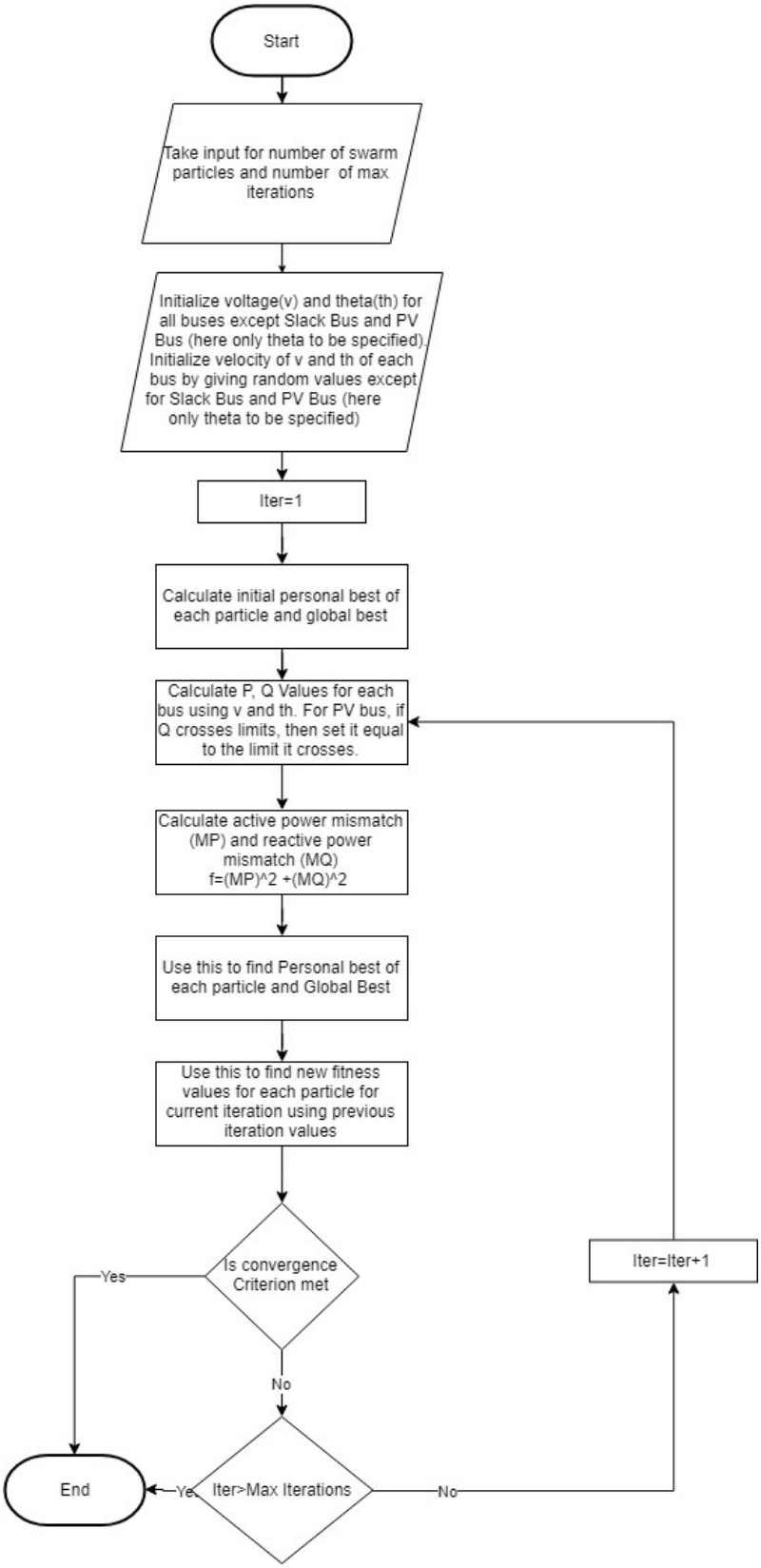


Figure 4.2: Loadflow methods

4.2.1 Output

PSO Loadflow Analysis

Bus No	V pu	Angle Degree	Injection MW	Injection MVar	Generation MW	Generation Mvar	Load MW	Load MVar
1	1.0200	0.0000	65.150	32.916	65.150	80.599	0.000	0.000
2	0.9548	-3.9413	-60.000	-30.000	0.000	0.000	60.000	30.000
3	1.0400	2.0008	100.000	47.684	100.000	0.000	0.000	0.000
4	0.9235	-8.0078	-40.000	-10.000	0.000	0.000	40.000	10.000
5	0.9931	-2.0726	-60.000	-20.000	0.000	0.000	60.000	20.000
Total			5.150	20.599	165.150	80.599	160.000	60.000

Table 4.1: PSO Loadflow Analysis

Line FLow and Losses

From Bus	To Bus	P MW	Q MVar	From Bus	To Bus	P MW	Q MVar	Line Loss MW	Line Loss MVar
1	2	19.800	12.264	2	1	-19.279	-10.178	0.521	2.086
1	4	24.805	11.743	4	1	-23.719	-7.399	1.086	4.344
1	5	20.545	8.909	5	1	-20.304	-7.945	0.241	0.964
2	3	-57.321	-23.698	3	2	59.431	32.139	2.110	8.441
2	4	16.600	3.876	4	2	-16.281	-2.601	0.319	1.275
3	5	40.569	15.545	5	3	-39.696	-12.055	0.873	3.490
Total Loss								5.150	20.599

Total Loss = 5.150

Table 4.2: Line Flow and Losses

Description for Output

- Total number of iterations=258
- In the injection column of PSO load flow analysis, the positive quantities indicate that power flows out of the bus and negative quantities indicate that power flows into the bus.
- The sum of all injections gives Total Losses which is equal to 5.150 MW. This can also be seen from Line Loss (MW) of Line Flow and Losses Table.
- From output, Total generation is 165.150 MW and total load is 160 MW. This again gives losses=5.150 MW.

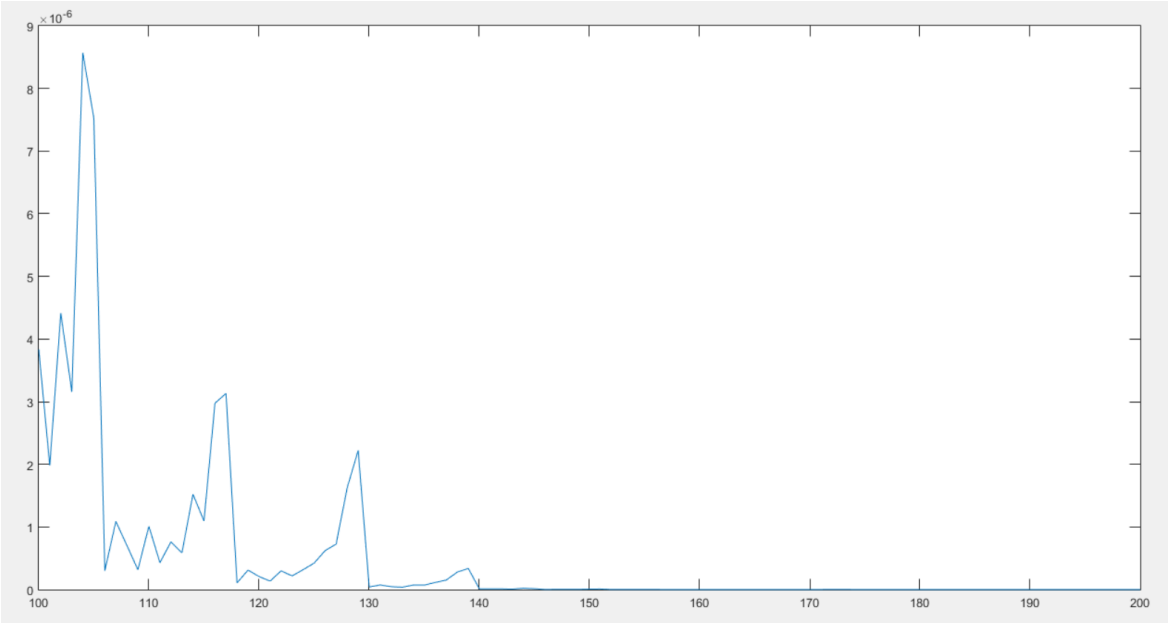


Figure 4.3: OBJECTIVE FUNCTION

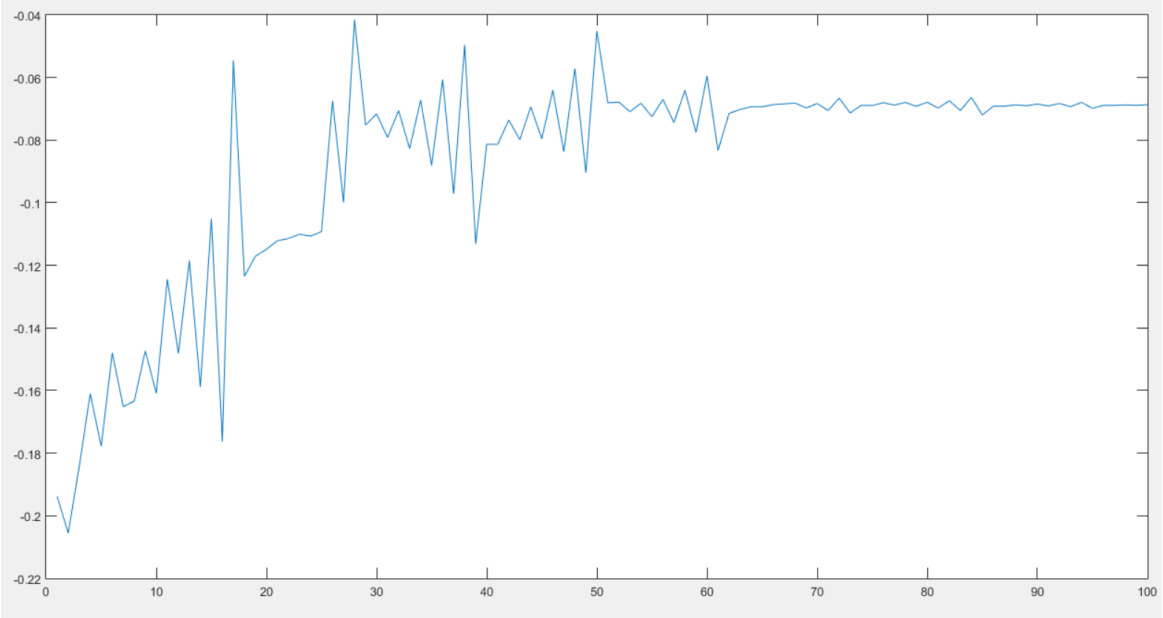


Figure 4.4: THETA BUS 2

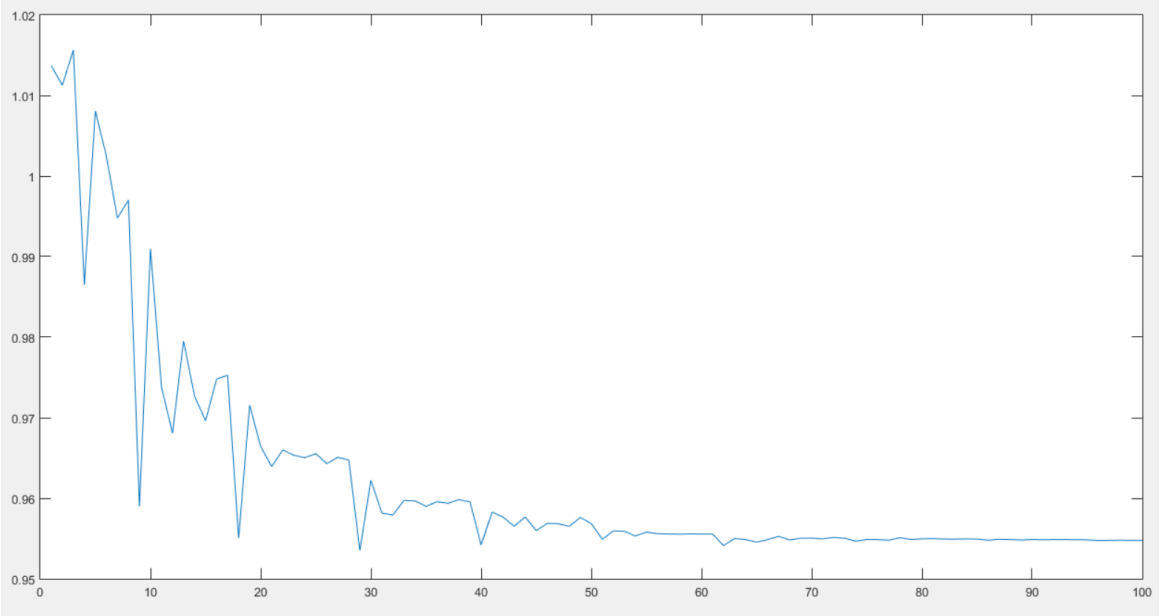


Figure 4.5: VOLTAGE BUS 2

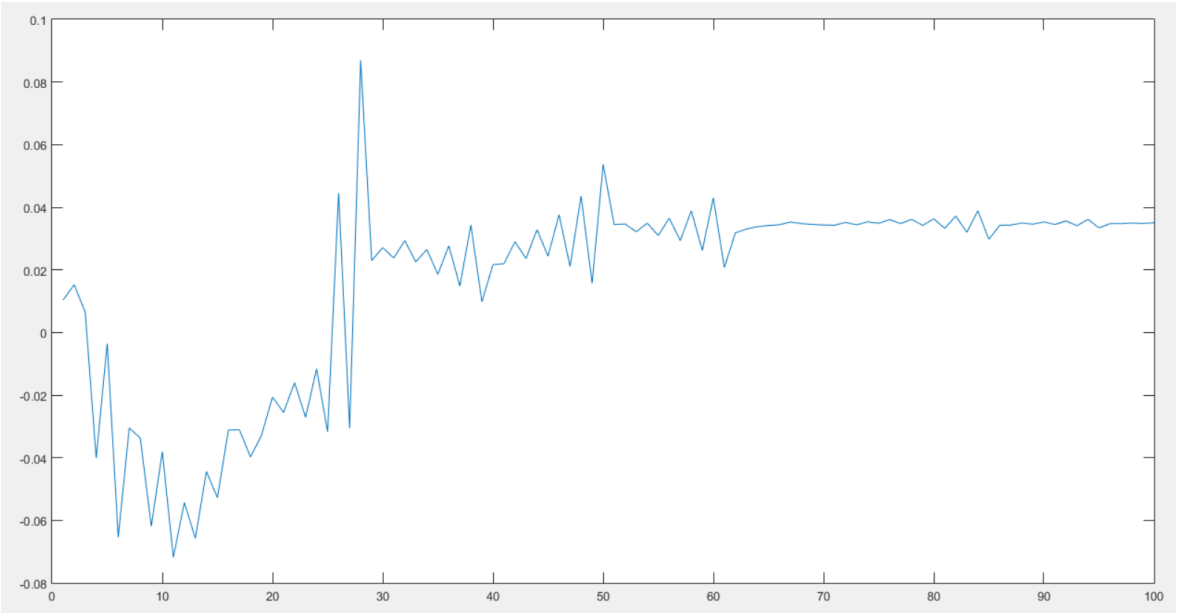


Figure 4.6: THETA BUS 3

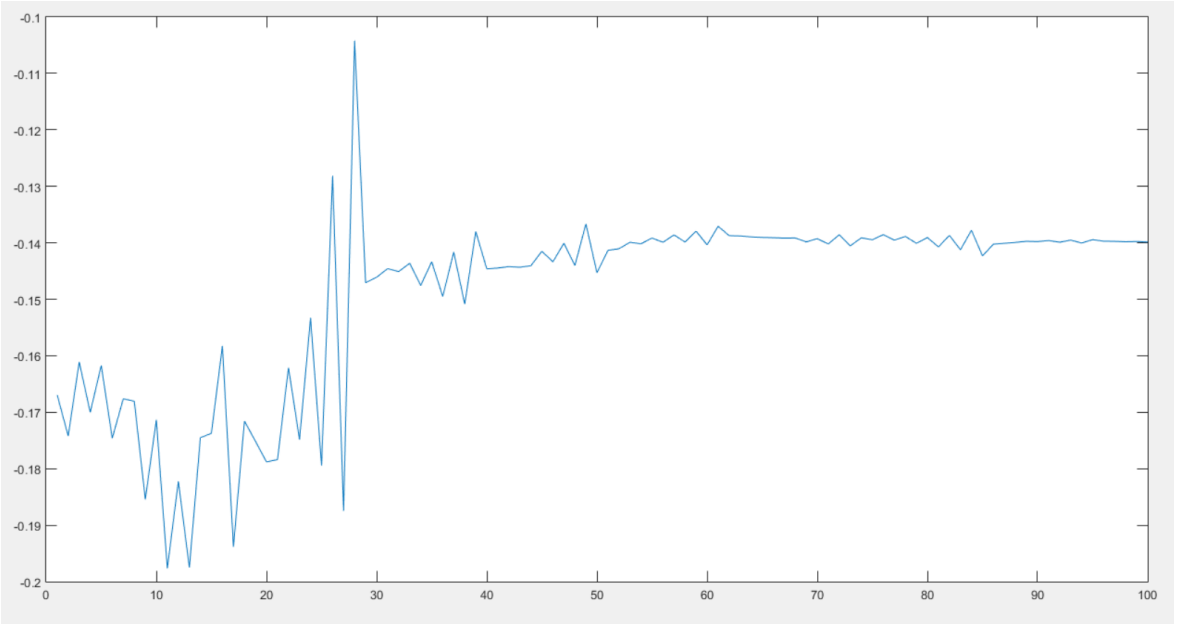


Figure 4.7: THETA BUS 4

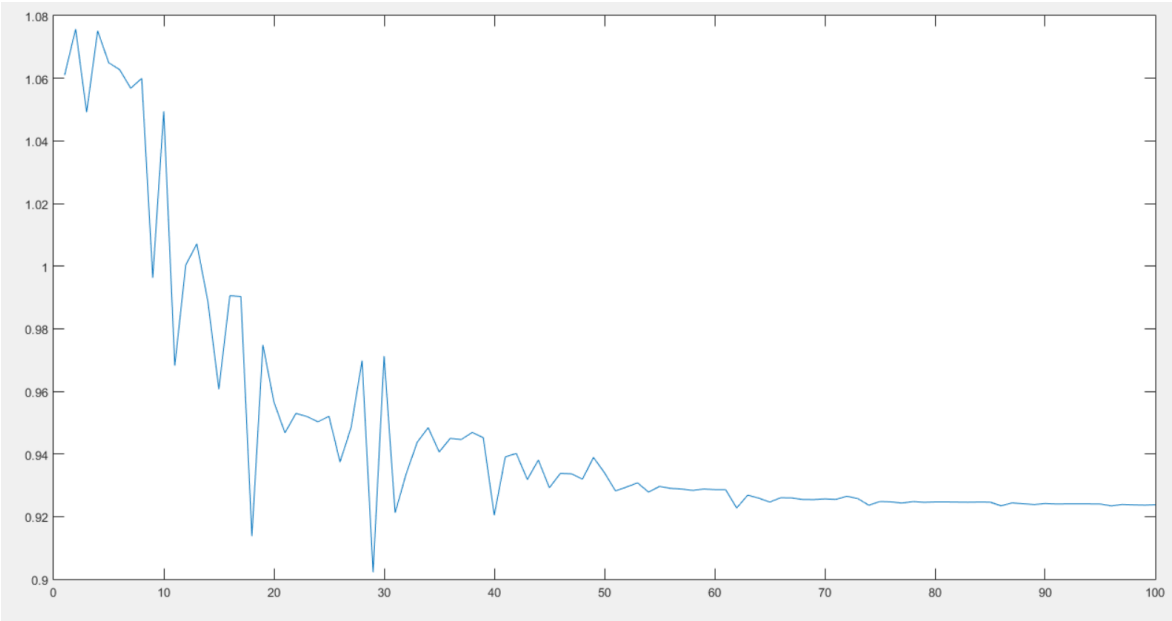


Figure 4.8: VOLTAGE BUS 4

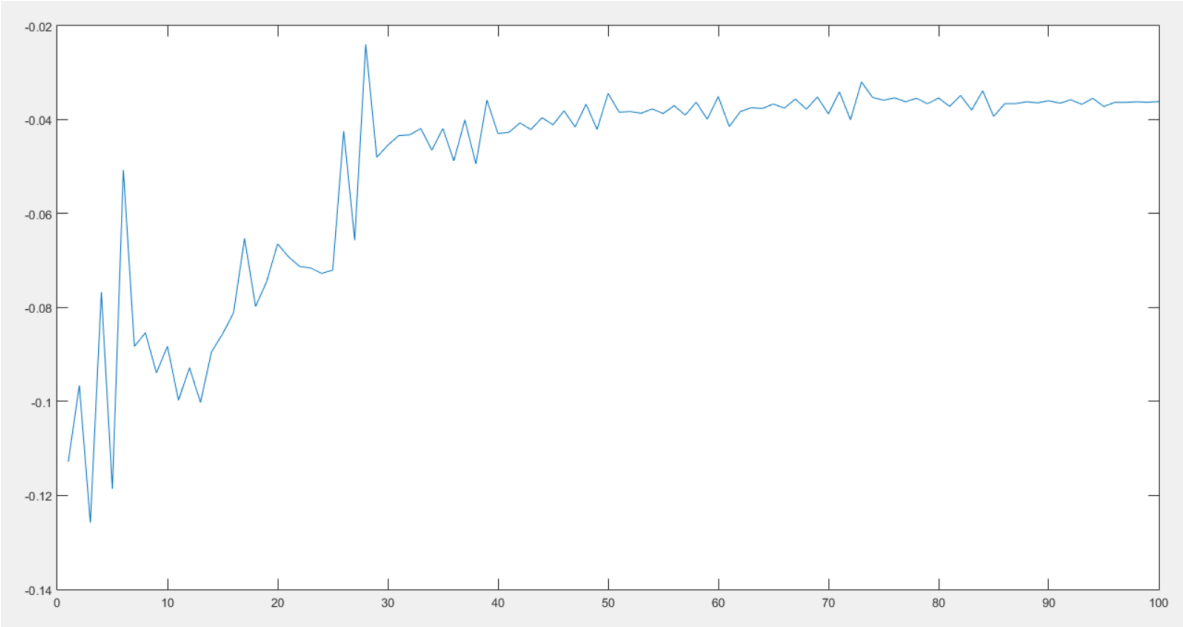


Figure 4.9: THETA BUS 5

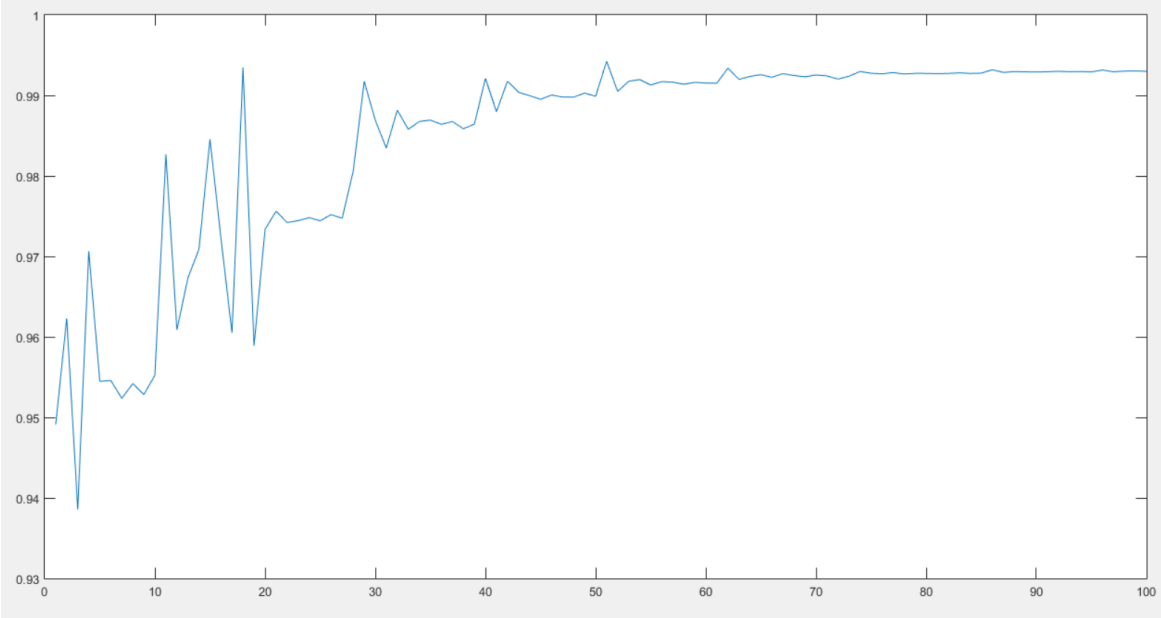


Figure 4.10: VOLTAGE BUS 5

CHAPTER 5

Improvement in PSO applied to Load Flow Studies

5.1 Positional Weightage PSO in Load Flow

From the previous method, it can be seen that before the particle swarm approached the true solutions, each individual particle made a high number of vibrations, i.e. the particle's value oscillated about the true solutions before the particle finally converged there.

In order to make the PSO program more efficient, the number of iterations have to be reduced.

Thus, in order to achieve this, we aim to reduce the vibrations. These can be reduced if all the particles were repositioned after some time, so that each particle was closer to true solution. This reduces the vibrations of each individual particle as well as the effect of one particle on others that caused others to stray away from true solution.

The particles have higher vibrations or higher amplitude of variations in the initial set of iterations ie. from 1-40. If it is desired to reposition the particles, it is best to do it within this initial set of iterations because if we are able to reduce the larger variations at the beginning itself, we will be able to reach the result faster.

The algorithm followed is:

1. Let program run normally till a fixed number of iterations. (Say 15)
2. For each particle, take into account itspersonal best and global best till the present iteration. Assign weight to each position depending upon the value and kind of solution we need(Maxima/minima). Higher the weight, better will be the position.
3. A factor called global weightage factor (s1) is also present along with function value at global best in order to further control and optimize the weight assigned to global best position.
4. Finally, take the weighted mean and continue the program to run as normal.

This approach not only reduces the vibrations, but also prevents a possible solution part of the function, that in normal case might be left out or be reached after longer time, to be taken into account.

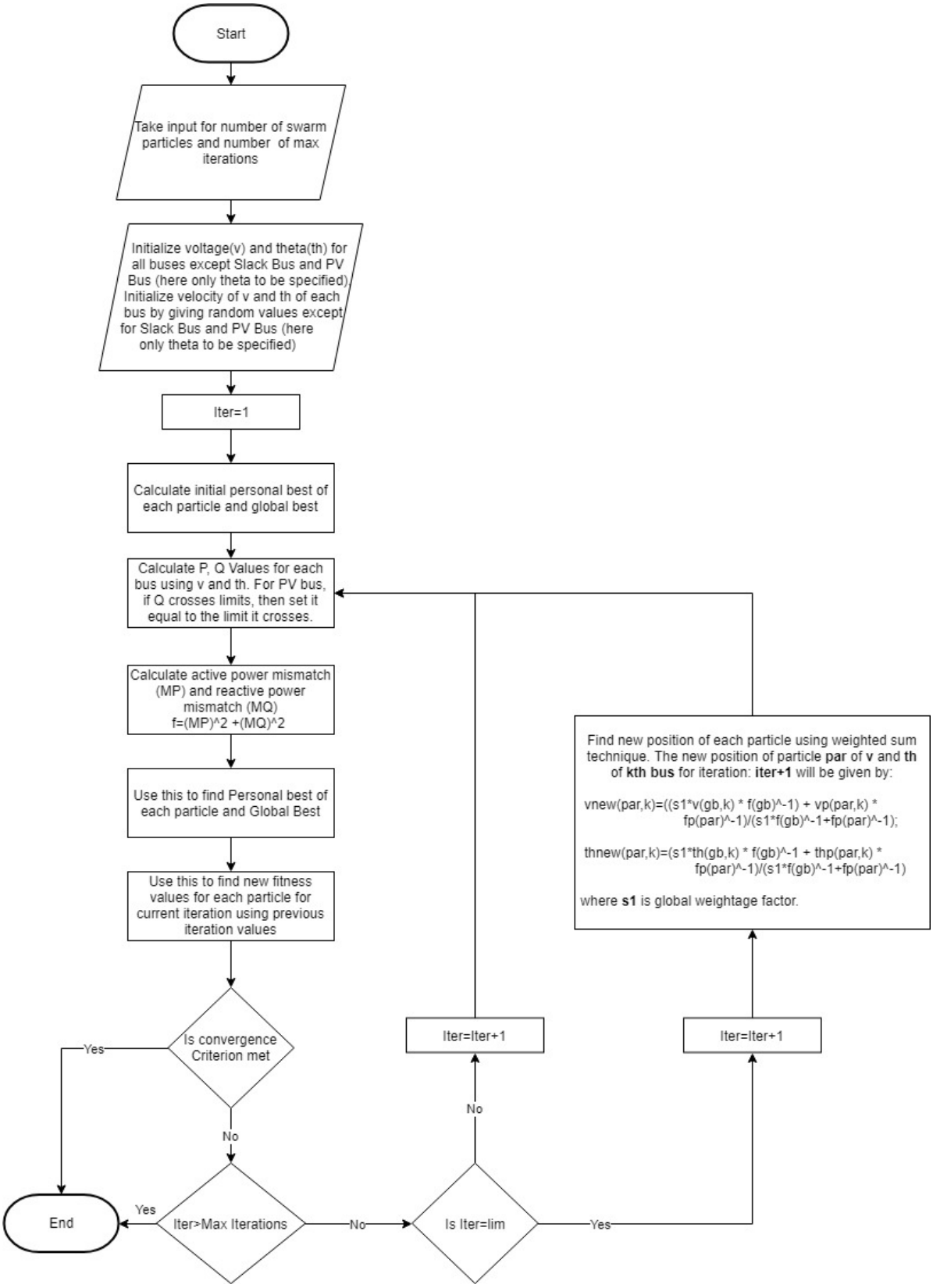


Figure 5.1: Flowchart for the process

5.2 Output

$$LIM = 15$$

$$S_1 = 0.493$$

PSO Loadflow Analysis

Bus No	V pu	Angle Degree	Injection MW	Injection MVar	Generation MW	Generation Mvar	Load MW	Load MVar
1	1.0200	0.0000	65.151	32.916	65.150	80.599	0.000	0.000
2	0.9548	-3.9414	-60.000	-30.000	0.000	0.000	60.000	30.000
3	1.0400	2.0007	99.999	47.684	100.000	0.000	0.000	0.000
4	0.9235	-8.0078	-40.000	-10.000	0.000	0.000	40.000	10.000
5	0.9931	-2.0726	-60.000	-20.000	0.000	0.000	60.000	20.000
Total			5.150	20.599	165.150	80.599	160.000	60.000

Table 5.1: PSO Loadflow Analysis using Improved PSO

Line FLow and Losses

From Bus	To Bus	P MW	Q MVar	From Bus	To Bus	P MW	Q MVar	Line Loss MW	Line Loss MVar
1	2	19.801	12.264	2	1	-19.279	-10.178	0.521	2.086
1	4	24.805	11.743	4	1	-23.719	-7.399	1.086	4.344
1	5	20.545	8.909	5	1	-20.304	-7.945	0.241	0.964
2	3	-57.321	-23.698	3	2	59.431	32.139	2.110	8.441
2	4	16.599	3.876	4	2	-16.281	-2.601	0.319	1.275
3	5	40.569	15.545	5	3	-39.696	-12.055	0.873	3.490
Total Loss								5.150	20.599

Total Loss = 5.150

Table 5.2: Line Flow and Losses

GRAPHICAL OUTPUTS IN THE ANALYSIS OF LOAD FLOW STUDY USING IMPROVED PSO SHOWING VARIATION OF PARTICLE WITH ITERATIONS

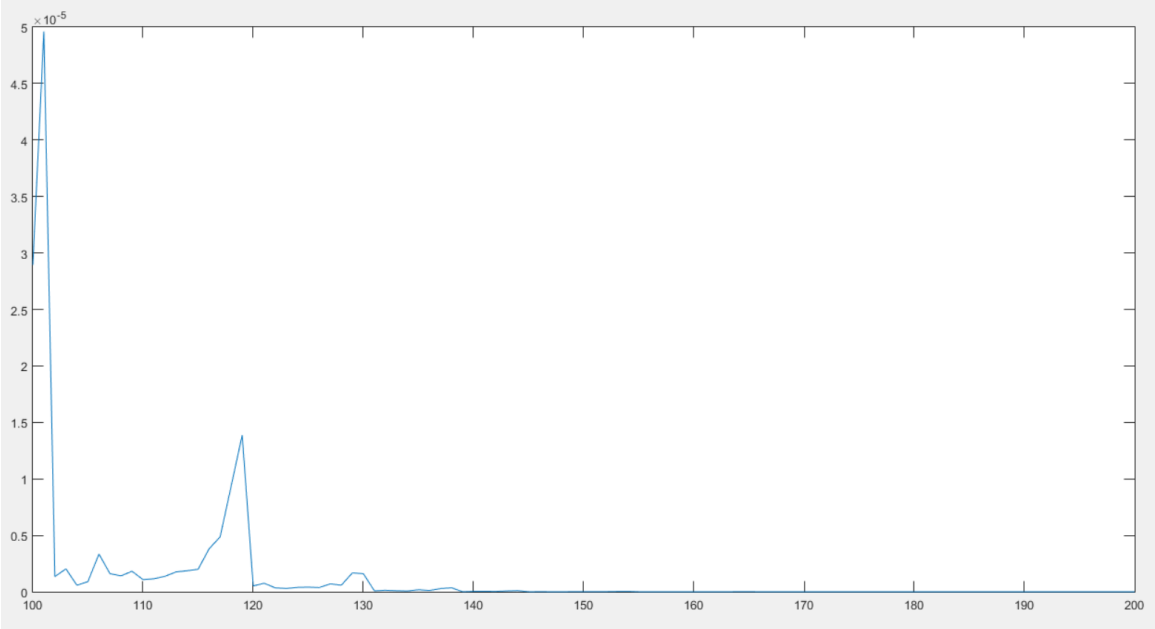


Figure 5.2: GRAPH FUNCTION IMPROVED PSO

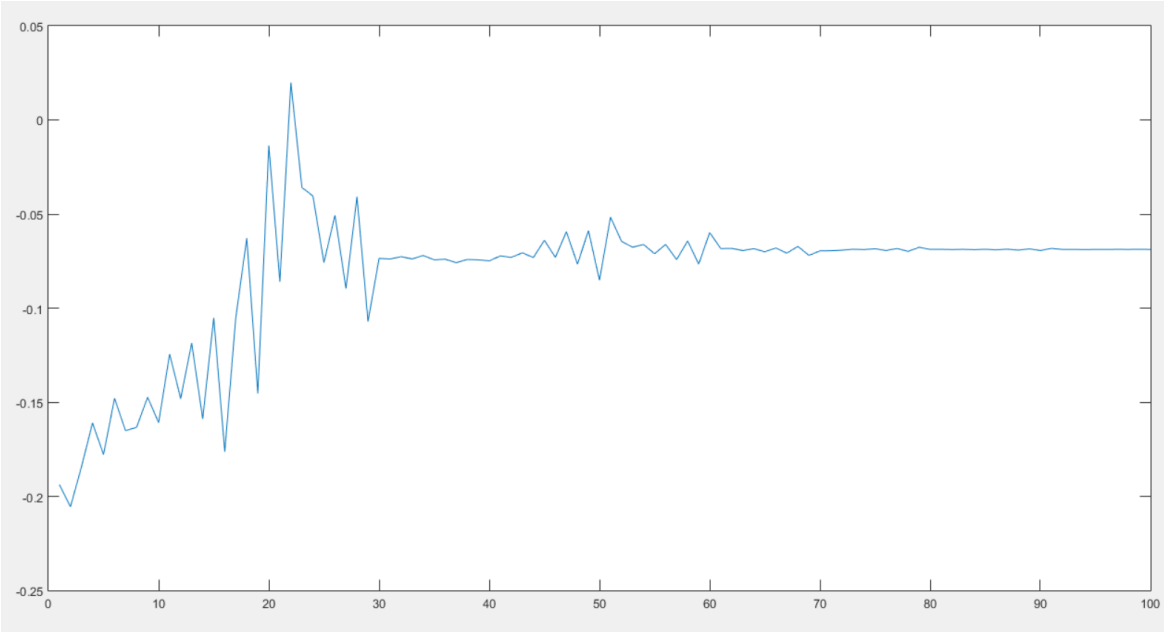


Figure 5.3: THETA BUS 2

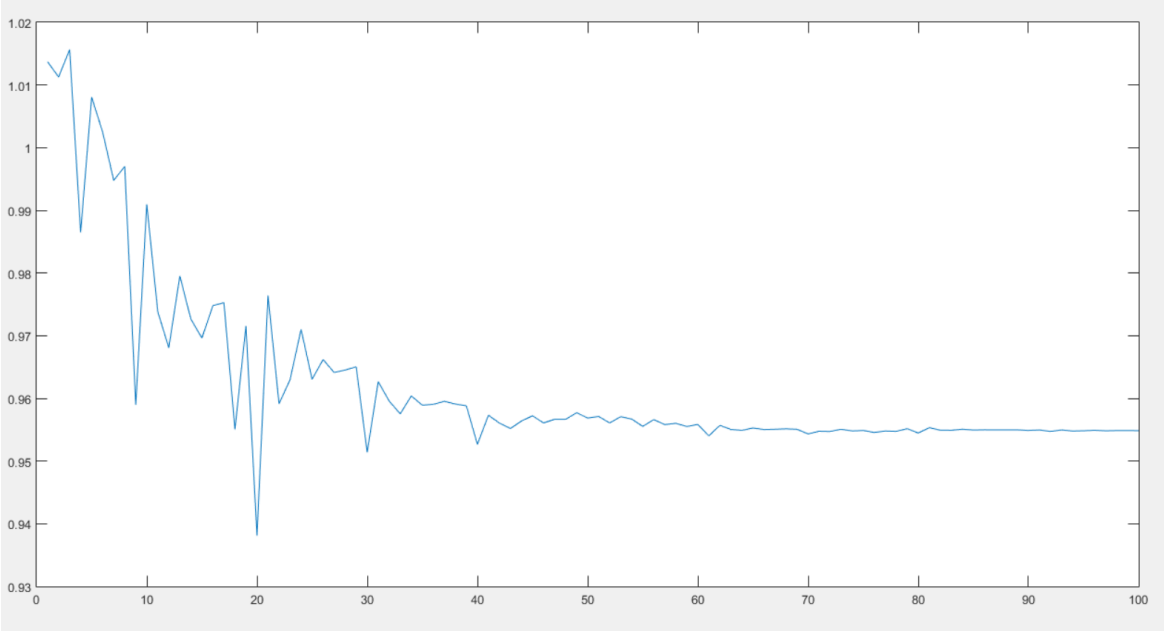


Figure 5.4: VOLTAGE BUS 2

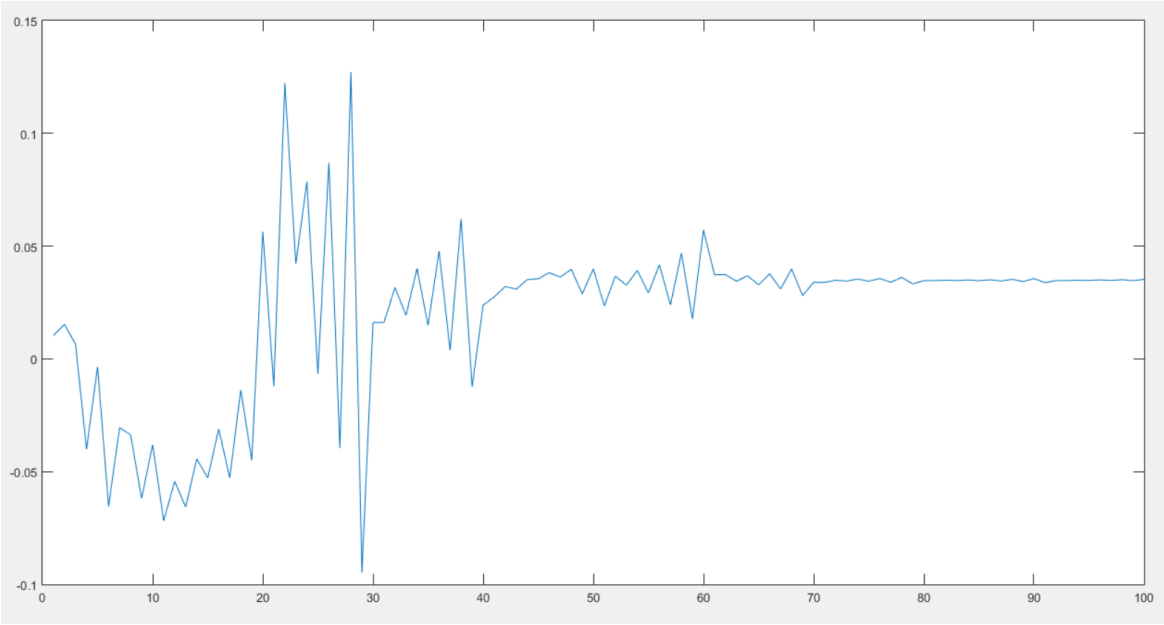


Figure 5.5: THETA BUS 3

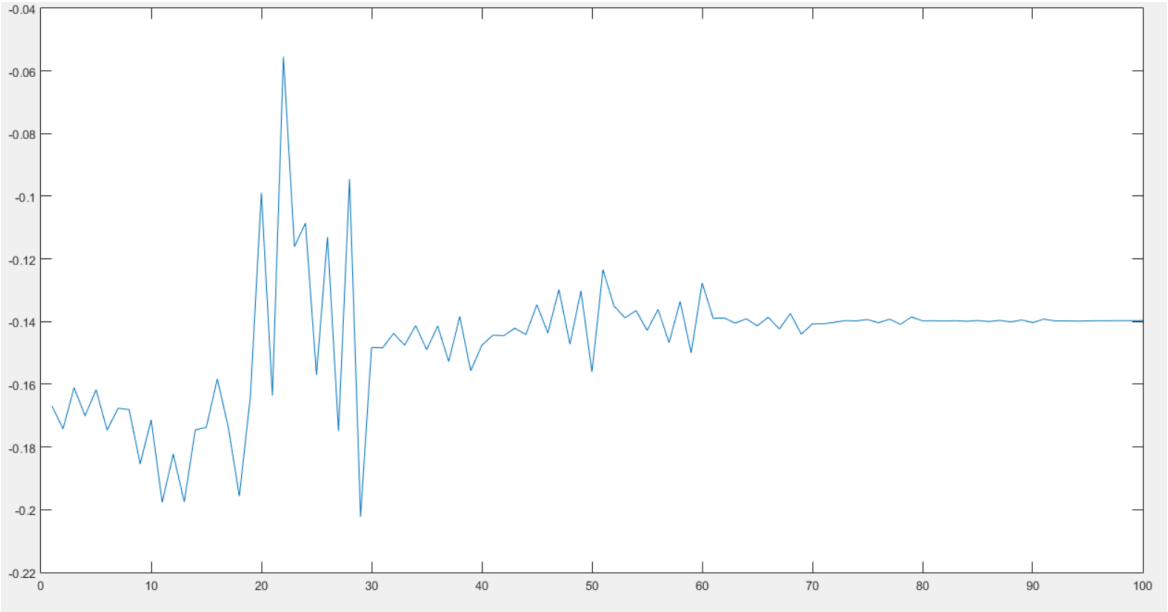


Figure 5.6: THETA BUS 4

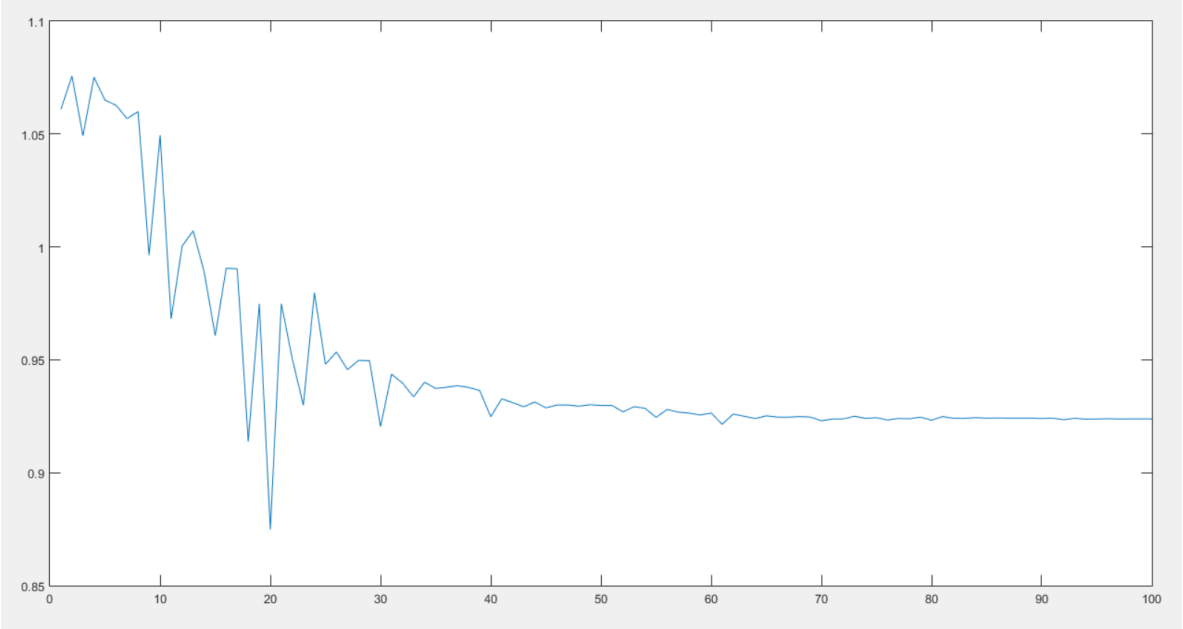


Figure 5.7: VOLTAGE BUS 4

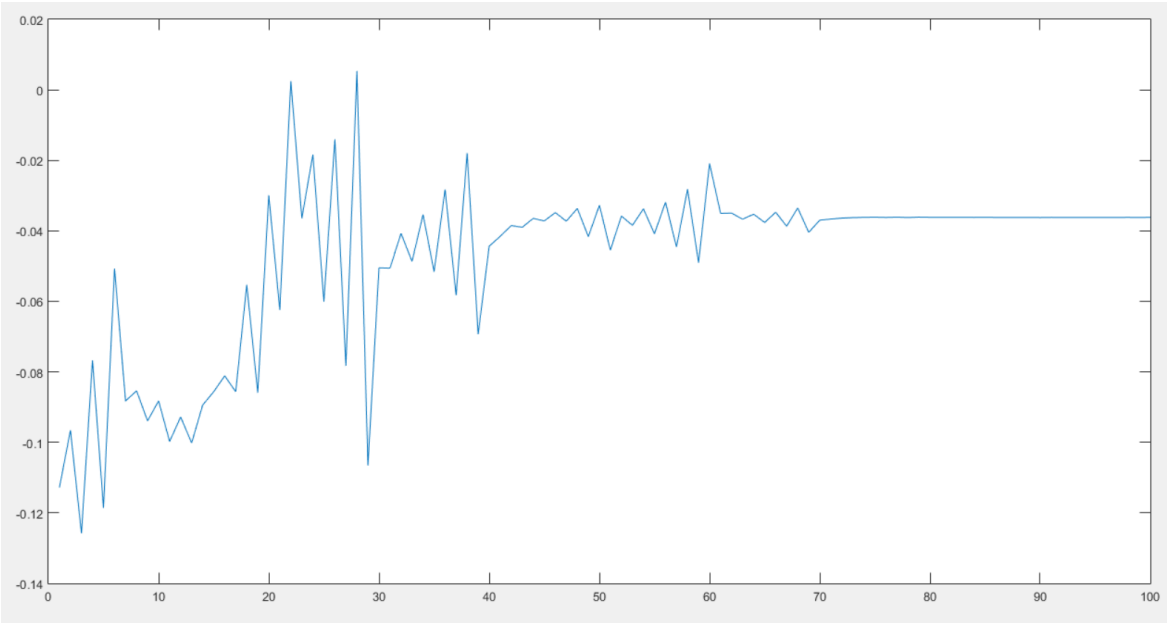


Figure 5.8: THETA BUS 5

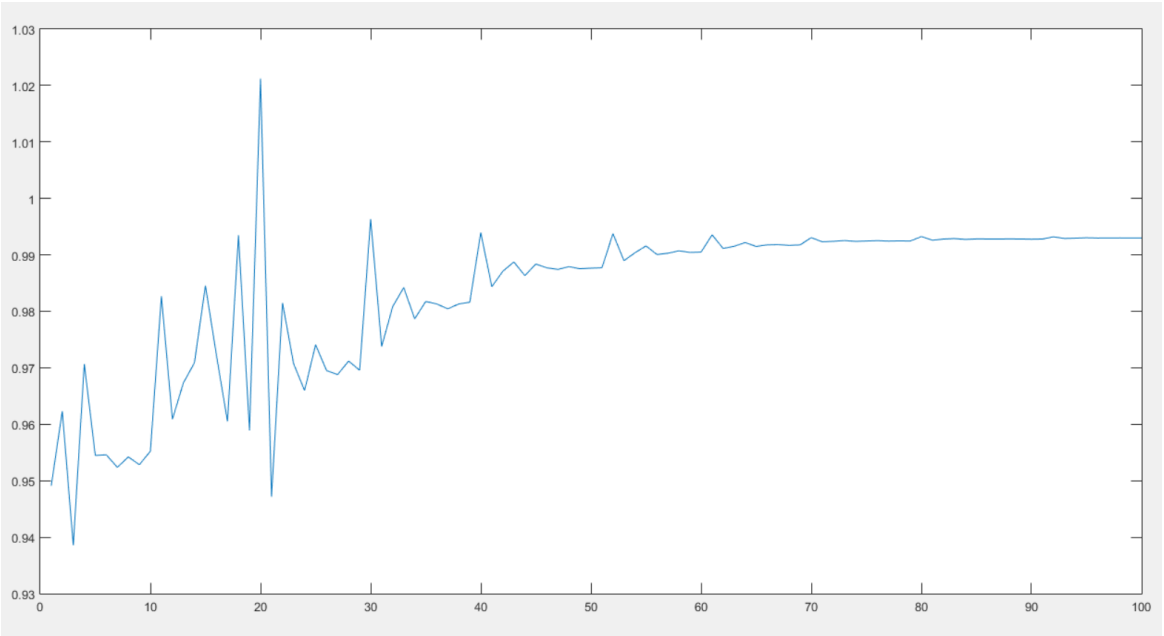


Figure 5.9: VOLTAGE BUS 5

S.No	limit	S1	It	K	Injection (slack)	Injection (generator)
1	15	0.493	211	42300	65.151	99.999
2	25	0.679	230	46100	65.150	100.00
3	30	0.50	231	46300	65.151	99.999
4	45	0.85	229	45900	65.150	100.00
5	70	0.57	227	45500	65.149	100.00
6	99	0.57	213	42700	65.150	100.00
7	120	0.60	228	45700	65.150	100.00
8	150	0.50	219	43900	65.150	100.00
9	180	0.57	230	46100	65.151	99.999

Table 5.3: VARIATON OF PARAMETERS WITH DIFFERENT VALUES OF LIMIT AND S1

The original program without the improvements gave the result in 258 iterations with the objective function being calculated 51700 times

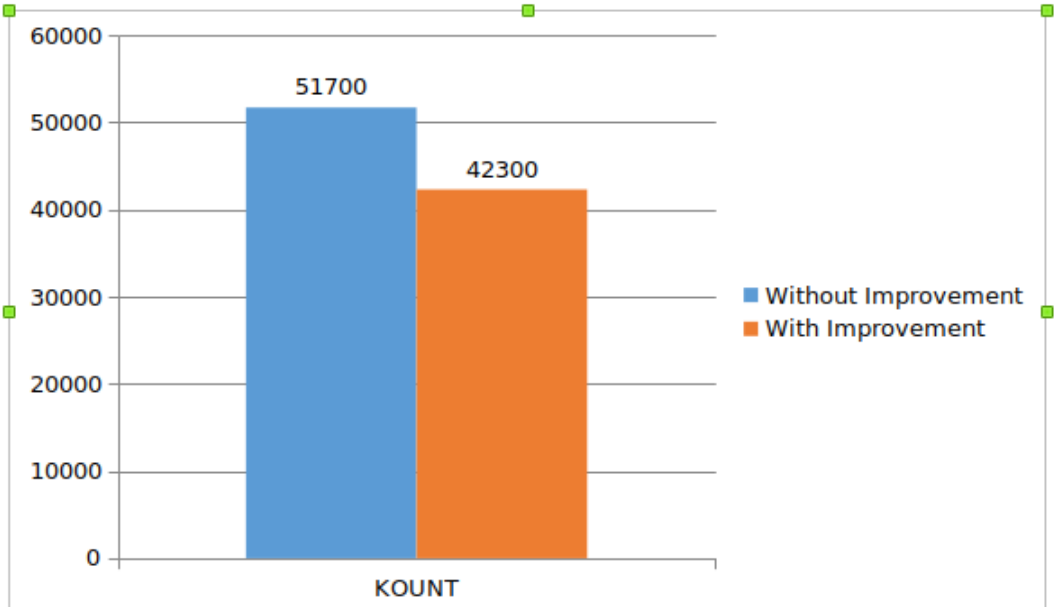


Figure 5.10: COMPARISION OF KOUNT VALUE OBTAINED IN IMPROVED PSO AND NORMAL PSO

5.3 Conclusion

After comparing the graphical visualization of the results for a particular particle and the observation table presented above, we can conclude that after applying the proposed improvement to the particle swarm optimization, we are getting significantly lesser number of iterations with no alteration in the result obtained.
Using the original technique, the result is obtained in 258 iterations. Whereas, using our method, after varying the values of the limit of recombination and the constant S1.the result is obtained in 211 iterations at limit=15 and S1=0.493. Thus, the optimization efficiency is improved by almost 18.2%. From the graph, we can see the particle converges towards the solution faster (lesser no. of iterations) than the original algorithm. Thus the improvement technique proves to be a worthy addition to the existing PSO algorithm.

CHAPTER 6

APPENDIX

RosenBrock PSO

```
1
2
3  clc
4  clear
5
6  disp(' we have to minimize f = 100 * (x1^2-x2)^2+(1-x1)^2 i.e.
       rosenbrockfunction')
7  p=input('Enter the no. of particles in a swarm');
8
9  %no. of particles
10 it=input('Enter the no. of iterations');
11
12 x1=zeros(p,it);
13
14 x2=zeros(p,it);
15
16 v1=zeros(p,it);
17
18 v2=zeros(p,it);
19
20 f=zeros(p,it);
21
22 kount=0;
23
24
25 fp=zeros(1,p);
26
27
28 df=zeros(1,(it-1));
29
30 rp=0.4;
31
32
33 rg=0.1;
34
35 cp=2;
```

```
36
37 cg = 0.2
38
39 T=input('Enter the tolerance value');
40
41 % Initial values i.e. 0th iteration
42
43 x1(:,1)=[1.2886 0.7572 1.6232 1.0657 0.7015 1.8780 1.7519 1.1003
       1.2450 1.1741];
44 x2(:,1)=[0.4155 0.6025 0.9418 0.4610 1.6886 0.3895 0.4518 0.3414
       0.4553 0.8714];
45 v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816
       0.1564 0.8555];
46 v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439
       0.5841 0.1078];
47
48  x1(:,1)=unifrnd(0,2,1,p);
49  x2(:,1)=unifrnd(0,2,1,p);
50  v1(:,1)=rand(1,p);
51  v2(:,1)=rand(1,p);
52
53  i=0;
54
55  disp (sprintf('enter the values of %dth iteration positions of %d
       particles for variable x1',i,p))
56
57  for j=1:p
58  x1(j,1)=input(sprintf('enter the value of x1(%d,%d)',j,i));
59  end
60
61  disp (sprintf('enter the values of 0th iteration positions of %d
       particles for variable x2',p))
62
63  for j=1:p
64  x2(j,1)=input(sprintf('enter the value of x2(%d,%d)',j,i));
65
66  end
67
68  disp (sprintf('enter the values of 0th iteration positions of %d
       particles for variable v1',p))
69
70  for j=1:p
71  v1(j,1)=input(sprintf('enter the value of v1(%d,%d)',j,i));
72  end
73
74  disp (sprintf('enter the values of 0th iteration positions of %d
       particles for variable v2',p))
75
76  for j=1:p
77  v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
78  end
79 for j=1:p
80
81      f(j,1)= 100*( (x1(j,1))^2 - x2(j,1) )^2 + (1- x1(j,1))^2 ; kount=
          kount+1;
82 end
83
84
```

```
%Initial personal besst values

x1p=x1(:,1);
x2p=x2(:,1);

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

    if f(k,1)==fmin
        gb=k;
    else
        end
    end

    %Initial global best value
    x1g=zeros(p);

    x2g=zeros(p);
    for k=1:p
        x1g(k) = x1(gb,1);

    x2g(k) = x2(gb,1);
    end
    fgm = min(f(:,1));

     fig=zeros(1,485);

     t=zeros(1,485);


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

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

    for j=1:p
        v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g
            (j)-
            x1(j,i));

    v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-
        x2(j,i));

    x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
    x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));

    f(j,(i+1))= 100*( (x1(j,(i+1)))^2 - x2(j,(i+1)) )^2 + (1-x1(j,(i
       +1)))^2 ;
    kount=kount+1;

    end

    %To find change in the values of f

    for j=1:p
```

```
141         df(j,i)= abs(f(j,(i+1)) - f(j,i)) ;
142     end
143
144     %personal best values updation
145
146     for j=1:p
147
148         fp(j)= 100*( (x1p(j))^2 - x2p(j) )^2 + (1- x1p(j))^2 ; kount=
               kount+1;
149
150     end
151     for k=1:p
152         if f(k,i)< fp(k)
153             x1p(k)=x1(k,i);
154         x2p(k)=x2(k,i);
155
156         else
157             end
158         end
159
160         %for Global best values updation
161
162         if min(f(:,(i+1)))<fgm
163             fgm=min(f(:,(i+1)));
164         else
165
166             end
167
168         for j=1:i
169             for k=1:p
170                 if f(k,i)==fgm
171
172                     for l=1:p
173
174                         x1g(l) = x1(k,i);
175
176                     x2g(l) = x2(k,i);
177
178
179
180
181
182
183                     %global best values
184
185                     end
186                     else
187
188                         end
189                     end
190                     end
191
192                     print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
193
194                     disp('      x1      x2      v1      v2      f')
195                     disp(print)
196
197                      fig(i) = figure('Position', [100 100 500 350]);
```

```
198
199                  t(i) = uitable('Parent', fig(i), 'Position', [25
                         25 450 200]);
200
201                  print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)
                         ];
202                  set(t(i), 'Data', print);
203                  set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2',
                         'f'});
204
205                 %Stoping criterion
206                 ki=0;
207
208                 for j=1:p
209
210                     if (df(j,i)<=10^(-T))
211                         ki=ki+1;
212
213                     end
214                     end
215                     if ki >= p
216                         break
217                     end
218
219                     end
220                     [r,c]=find(f==fgm);
221                     minf=100*( (x1g(j))^2 - x2g(1) )^2 + (1- x1g
                            (1))^2;
222                     disp(sprintf('min value of function is %d and
                             at values of x1=%d and x2=%d
223                             ',minf,x1g(1),x2g(1)))
224                     kount+1
```

Constant Weight PSO

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

```
            positions of %d particles for variable v2',p))
        for j=1:p
        v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i)
            );
        end
    for j=1:p
    f(j,1)= 100*( (x1(j,1))^2 - x2(j,1) )^2 + (1- x1(j,1))^2 ;
        kount=kount+1;
    end
    %Initial personal besst values
    x1p=x1(:,1);
    x2p=x2(:,1);
    %for Initial Global best values updation fmin=min(f(:,1));
    for k=1:p
    if f(k,1)==fmin
    gb=k;
    else
    end
    end
    %Initial global best value
    x1g=zeros(p);
    x2g=zeros(p);
    for k=1:p
    x1g(k) = x1(gb,1);
    x2g(k) = x2(gb,1);
    end
    fgm = min(f(:,1));
        fig=zeros(1,485);
        t=zeros(1,485);
    for i=1:it
    disp(sprintf('This is %d no. of iteration',i))
    %for inertia weight W
    w=0.6
    for j=1:p
    v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g
        (j)-
    x1(j,i));
    v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g
        (j)-
    x2(j,i));
    x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
    x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
    f(j,(i+1))= 100*( (x1(j,(i+1)))^2 - x2(j,(i+1)) )^2 + (1-x1(j
        ,(i+1)))^2 ;
    kount=kount+1;
    end
    %To find change in the values of f
    for j=1:p
    df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
    end
    %personal best values updation
    for j=1:p
    fp(j)= 100*( (x1p(j))^2 - x2p(j) )^2 + (1- x1p(j))^2 ; kount=
        kount+1;
    end
    for k=1:p
    if f(k,i)< fp(k)
    x1p(k)=x1(k,i);
```

```
95          x2p(k)=x2(k,i);
96          else
97          end
98          end
99          %for Global best values updation
100         if min(f(:,(i+1)))<fgm
101         fgm=min(f(:,(i+1)));
102         else
103         end
104         for j=1:i
105         for k=1:p
106         if f(k,i)==fgm
107         for l=1:p
108         x1g(l) = x1(k,i);
109         x2g(l) = x2(k,i);
110         %global best values
111         end
112         else
113         end
114         end
115         end
116         print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
117         disp('  x1   x2   v1   v2   f')
118         disp(print)
119             fig(i) = figure('Position', [100 100 500 350]);
120             t(i) = uitable('Parent', fig(i), 'Position', [25 25 450
                    200]);
121             print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
122             set(t(i), 'Data', print);
123             set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});
124         %Stoping criterion
125         ki=0;
126         for j=1:p
127         if (df(j,i)<=10^(-T))
128         ki=ki+1;
129         end
130         end
131         if ki >= p
132         break
133         end
134         end
135         [r,c]=find(f==fgm);
136         minf=100*( (x1g(j))^2 - x2g(1) )^2 + (1- x1g(1))^2;
137         disp(sprintf('min value of function is %d and at values of x1
                =%d and x2=%d
138         ',minf,x1g(1),x2g(1)))
139         kount+1
140         for m=1:p
141         figure(m)
142         subplot(2,1,1)
143         plot(x1(m,1:i),x2(m,1:i))
144         hold on
145         plot(x1(m,1),x2(m,1),'r*')
146         hold on
147         plot(x1(m,i),x2(m,i),'o')
148         hold on
149         end
```

Positional PSO

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

```
      disp (sprintf('enter the values of 0th iteration positions of %d
          particles for variable v2',p))
      for j=1:p
      v2(j,1)=input(sprintf('enter the value of v2(%d,%d)',j,i));
      end
for j=1:p
f(j,1)= 100*( (x1(j,1))^2 - x2(j,1) )^2 + (1- x1(j,1))^2 ; kount=
    kount+1;
end
%Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);
%for Initial Global best values updation fmin=min(f(:,1));
for k=1:p
if f(k,1)==fmin
gb=k;
else
end
    end
%Initial global best value
x1g=zeros(p);
x2g=zeros(p);
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
end
fgm = min(f(:,1));
     fig=zeros(1,485);
     t=zeros(1,485);
i=1;
lim=14;
for iter=1:it
disp(sprintf('This is %d no. of iteration',iter))
if iter==lim+1
fold=f(:,1:lim);
x1old(:,1:lim)=x1(:,1:lim);
x2old(:,1:lim)=x2(:,1:lim);
x1new=zeros(p,it-lim);
x2new=zeros(p,it-lim);
for par=1:p
x1new(par,1)=(sum(x1(par,1:i).*(f(par,1:i)).^-
1))/sum(f(par,1:i).^-1);
x2new(par,1)=(sum(x2(par,1:i).*(f(par,1:i)).^-
1))/sum(f(par,1:i).^-1);
end
x1=x1new;
x2=x2new;
%v1(:,1)=[0.0326 0.5612 0.8819 0.6692 0.1904 0.3689 0.4607 0.9816
    0.1564 0.8555];
%v2(:,1)=[0.2518 0.2904 0.6171 0.2653 0.8244 0.9827 0.7302 0.3439
    0.5841 0.1078];
v1=zeros(p,it-lim);
v2=zeros(p,it-lim);
f=zeros(p,it-lim);
fp=zeros(1,p);
df=zeros(1,(it-1-lim));
for j=1:p
f(j,1)= 100*( (x1(j,1))^2 - x2(j,1) )^2 + (1- x1(j,1))^2 ; kount=
```

```
    kount+1;
end
%Initial personal besst values
x1p=x1(:,1);
x2p=x2(:,1);
%for Initial Global best values updation fmin=min(f(:,1));
for k=1:p
if f(k,1)==fmin
gb=k;
else
end
end
%Initial global best value
x1g=zeros(p);
x2g=zeros(p);
for k=1:p
x1g(k) = x1(gb,1);
x2g(k) = x2(gb,1);
end
fgm = min(f(:,1));
     fig=zeros(1,485);
     t=zeros(1,485);
i=1;
end
%for inertia weight W
w=0.6
for j=1:p
v1(j,(i+1)) = w*v1(j,i) + rp*cp*(x1p(j)-x1(j,i)) + rg*cg*(x1g(j)-x1(j
    ,i));
v2(j,(i+1)) = w*v2(j,i) + rp*cp*(x2p(j)-x2(j,i)) + rg*cg*(x2g(j)-x2(j
    ,i));
x1(j,(i+1)) = x1(j,i) + v1(j,(i+1));
x2(j,(i+1)) = x2(j,i) + v2(j,(i+1));
f(j,(i+1))= 100*( (x1(j,(i+1)))^2 - x2(j,(i+1)) )^2 + (1-x1(j,(i+1)))
    ^2 ;
kount=kount+1;
end
%To find change in the values of f
for j=1:p
df(j,i)= abs(f(j,(i+1))-f(j,i)) ;
end
%personal best values updation
for j=1:p
fp(j)= 100*( (x1p(j))^2 - x2p(j) )^2 + (1- x1p(j))^2 ; kount=kount+1;
end
for k=1:p
if f(k,i)< fp(k)
x1p(k)=x1(k,i);
x2p(k)=x2(k,i);
else
end
end
%for Global best values updation
if min(f(:,(i+1)))<fgm
fgm=min(f(:,(i+1)));
else
end
    for j=1:i
```

```
for k=1:p
if f(k,i)==fgm
for l=1:p
                    x1g(l) = x1(k,i);    %global best values
x2g(l) = x2(k,i);
end
else
        end
end
end
print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
disp('  x1      x2      v1      v2      f')
disp(print)
     fig(i) = figure('Position', [100 100 500 350]);
     t(i) = uitable('Parent', fig(i), 'Position', [25 25 450 200]);
     print = [x1(:,i) x2(:,i) v1(:,i) v2(:,i) f(:,i)];
     set(t(i), 'Data', print);
     set(t(i), 'ColumnName', {'x1', 'x2', 'v1', 'v2', 'f'});
%Stoping criterion
ki=0;
for j=1:p
if (df(j,i)<=10^(-T))
ki=ki+1;
end
end
if ki >= p
break
end
i=i+1;
end
[r,c]=find(f==fgm);
minf=100*( (x1g(j))^2 - x2g(1) )^2 + (1- x1g(1))^2;
disp(sprintf('min value of function is %d and at values of x1=%d and
   x2=%d
',minf,x1g(1),x2g(1)))
kount+1
x1all(1:p,1:lim)=x1old;
x1all(1:p,15:it)=x1;
x2all(1:p,1:lim)=x2old;
x2all(1:p,15:it)=x2;
fall(1:p,1:lim)=fold;
fall(1:p,15:it)=f;
for m=1:p
figure(m)
subplot(2,1,1)
plot(x1all(m,1:iter),x2all(m,1:iter))
hold on
plot(x1all(m,1),x2all(m,1),'r*')
hold on
plot(x1all(m,i),x2all(m,i),'o')
hold on
subplot(2,1,2)
plot(1:iter,fall(m,1:iter))
hold on
end
```

PSO APPLIED TO ANALYSIS OF LOAD FLOW STUDIES

```
clear all
clc
kount=0;
nbus = 5;
busd = busdatas(nbus);      % Calling busdatas..
Y = ybusppg(nbus);          % Calling ybusppg.m to get Y-Bus Matrix..
BMva=100;                   % Base MVA..
Ql = busd(:,8)/BMva;        % load reactive power
Pl = busd(:,7)/BMva;        % load real power
Qg = busd(:,6)/BMva;        % generated reactive power.
Pg = busd(:,5)/BMva;        % gen  erated real power
Qlim1 =busd(:,9)/BMva;
Qlim2 =busd(:,10)/BMva;
Q =Qg - Ql;                 % Qi =QGi - QLi
P =Pg - Pl;                 % Pi =PGi - PLi
Qsp = Q    ;                 % Q--> specified
Psp = P   ;                 % P-->Specified..
Qmin=Qlim1(3);
Qmax=Qlim2(3);
B =imag(Y)                  % Suscept. matrix
G =real(Y)                  % Conduct. matrix

%---------------------PSO PARAMETERS INITIALIZATION --------------%
p=100;                     % no of particle
it=1000;
rp=1.1;
T=14;
rg=1.1;
rf=1;
f=zeros(p,it);
fp=zeros(p,1);
thp=zeros(p,5);
thg=zeros(p,5);
vp=zeros(p,5);
vg=zeros(p,5);
v=zeros(p,it,5);
th=zeros(p,it,5);
vth=zeros(p,it,5);
vv=zeros(p,it,5);
a=.05;
b=-0.05;
vth(:,1,:)=a+(b-a)*rand(p,5); %initial velocity of theta vector%
a=.05;
b=-0.05;
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
    -------------%
% Calc. P & Q
PVIND=zeros(p,1);

 for j=1:p
 Q =zeros(nbus,1);
 P =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

 % 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);
  kount=kount+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
for j=1:p
        for k=2:5

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


        for q=2:5
        vv(j,(i+1),q) = w*vv(j,i,q) + rp*(vp(j,q)-v(j,i,q)) + rg*(vg(
            j,q)-v(j,i,q));
        if  vv(j,(i+1),q)<-0.1
```

```
                vv(j,(i+1),q)=-0.1;
        end
                 if  vv(j,(i+1),q)>0.1
                vv(j,(i+1),q)=0.1;
                 end

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


end


%th(:,i,5)

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

```
219     Q3(j,i)=Q(3);
220    %reactive power mismatch
221    MQ=Q-Qsp;
222    MQ(3)=0;
223    MQS=MQ.^2;
224    %objective function value
225
226    f(j,(i+1))=sum(MPS)+sum(MQS);
227    kount=kount+1;
228
229
230  end
231             %personal best values updatio
232  for j=1:p
233   Q = zeros(nbus,1);
234   P =zeros(nbus,1);
235   MPS=zeros(p,1);
236   MQS=zeros(p,1);
237          for t =2:nbus
238          for k = 1:nbus
239              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)));
240
241          end
242          end
243          for t = 2:nbus
244          for k = 1:nbus
245              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)));
246
247          end
248          end
249
250   % real power mismatch
251    MP=P-Psp;
252    MPS=MP.^2;
253    %reactive power mismatch in third bus
254    Qsp(3)=Q(3);
255    if Q(3)<Qmin;
256        Q(3)=Qmin;
257        PVIND(j,1)=1;
258    else PVIND(j,1)=0;
259    end
260        if Q(3)>Qmax;
261            Q(3)=Qmax;
262            PVIND(j,1)=1;
263    else PVIND(j,1)=0;
264        end
265
266    %reactive power mismatch
267    MQ=Q-Qsp;
268    MQ(3)=0;
269    MQS=MQ.^2;
270    %objective function value
271    %objective function value
272    fp(j)=sum(MPS)+sum(MQS);
273    kount=kount+1;
274  end
```

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

        end

       end

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

        end
      end

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

  for m=2:5

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

   end
  end
       for m=2:5

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

   end
       end


%  print = [th(:,i,1) th(:,i,2) th(:,i,3) th(:,i,4) th(:,i,5)  v(:,i
    ,1) v(:,i,2) v(:,i,3) v(:,i,4) v(:,i,4)  f(:,i)];
    i
     f(gb,i)
```

```
%    disp('  th1        th2         th3         th4       th5         v1
          v2         v3        v4       v5           f')

 % disp(print)

       %stopping
       gb1=gb;
for k=1:p
    if f(k,1+1)==fgm
        gb1=k;
    else
    end
end
    if (f(gb1,i+1)<=10^(-T))

        break
    end

vth(gb1,(i+1),2);
end
%Q3(:,it)
bus=zeros(5,10) ;
bus(1,3)=v(gb,i,1);
bus(2,3)=v(gb,i,2);
bus(3,3)=v(gb,i,3);
bus(4,3)=v(gb,i,4);
bus(5,3)=v(gb,i,5);
%bus angle updation
bus(1,4)=th(gb,i,1);
bus(2,4)=th(gb,i,2);
bus(3,4)=th(gb,i,3);
bus(4,4)=th(gb,i,4);
bus(5,4)=th(gb,i,5);
bus(3,6)=Q3(gb1,it)*BMva;
x=bus(3,6);
V = bus(:,3)    ;        % Specified Voltage..
del = bus(:,4) ; % Voltage Angle..
%load flow function calling
fprintf('kount value is %d\n',kount);
loadflow(nbus,V,del,BMva,x);
```

Description for above code :

1. This is the main program that performs PSO on given data in order to give the load flow results.
2. There are 5 buses in the system:
    - 1 Slack Bus(Bus 1)
    - 3 Load Buses (Bus 2,4,5)- P,Q Known
    - 1 Generator Bus (Bus 3)- P,—V— Known
3. Various parameters like Total Load, Total Generation, Net Generation (Total Gen-Total Load), bus admittance matrix are calculated. These will be used in PSO operation.
4. The variables that will be optimized by PSO are voltage magnitude (v) andvoltage angle (th).
5. For the PSO operation, we have taken:
    - No. of particles=100
    - Max. iterations=1000
    - Tolerance in objective function=10^{-6}
6. In each iteration, **v and theta** are updated in order to find P, Q values for each bus. The given or specified values of P and Q are checked against calculated P and Q values respectively and the mismatch of P values and that of Q values is calculated.
7. **The objective function is a sum of square of mismatch of Pand mismatch of Q.** Squaring is done in order to get rid of signs.
8. **The task of PSO is to minimize this objective function.** When this is done, we get the unknown values which can be used for Load Flow analysis.

Bus Data

busdata.m

```
% This function provides Initial Bus data of the system to the main
    program
function busdt = busdatas(num)

% Bus Type      Type Descrpition
% 1             Slack Bus
% 2             P-V Bus
% 3             P-Q Bus

%          |Bus | Type | Vsp | theta | PGi | QGi | PLi | QLi |  Qmin |
     Qmax |
busdat5 = [1      1      1.02   0      0      0      0      0        0
          0;
            2      3      1      0      0      0     60      30       0
                       0;
            3      3      1.04   0      100    0      0      0        0
                       60;
            4      3      1      0      0      0     40      10       0
                       0;
            5      3      1      0      0      0     60      20       0
                       0;];

busdt = busdat5;
```

Description for above code :

1. This is the part of the program that contains information regarding buses in the given power system.
2. From here voltage magnitude and angle, power generation, load, etc. are imported into main program and supporting functions for further calculations.

Line Data

linedata.m

```
% This function provides Line Bus data of the system to the main
   program

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];

linedt = linedat5;
```

Description for above code :

1. This part contains information regarding transmission line joining the buses in the given power system.
2. From here data required for calculation of bus admittance matrix is obtained.

Ybus

ybusppg.m

```
% This function forms and returns the Bus Admittance Matrix

function Y = ybusppg(num)        % Returns Y-->bus admittance matrix

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

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

% For Off Diagonal elem
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

% For diagonal elem
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
```

Description for above code :

1. This function forms the bus admittance matrix from the information present in linedatas.m

pol2rect

pol2rect.m

```
% This function converts Rectangular to Polar form
% [RECT]=RECT2POL(RHO, THETA)
% RECT= Complex matrix or number
% RECT = A + jB; where: A-->Real, B-->Imaginary
% RHO= magnitude
% THETA = angle(rad)

function rect =pol2rect(rho, theta)
rect =rho.*cos(theta)+j*rho.*sin(theta);
```

Description for above code :

1. This function coverts from polar coordinates to rectangular coordinates.

loadflow

loadflow.m

```
% This function takes as parameters all the final values calculated
    from PSO operation and also takes in values from above functions
    to calculate Bus current injections, Line current flows, Line
    power flows and Bus Power Injections.


function [Pi Qi Pg Qg Pl Ql] =loadflow(nb,V,del,BMva,x)

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

% Bus Current-->Injections
 I =Y*Vm;
 Ia =angle(I);
 Im =abs(I);

%Line Current
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);
Iija =angle(Iij);
Iijm =abs(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
Qij =imag(Sij);
Pij = real(Sij) ;


% Line Loss
```

```
Lij = zeros(nl,1);
for m = 1:nl
    q = tb(m);
    p = fb(m);
    Lij(m) =Sij(p,q) + Sij(q,p);
end

Lqij =imag(Lij);
Lpij = real(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
Qi = -imag(Si);
Pi =real(Si);
Pg= busd(:,5);
Qg= busd(:,6);
Pg(1)=sum(Pl)+sum(Lpij)-Pg(3);
Qg(3)=x;
Qg(1)=sum(Ql)+sum(Lqij)-Qg(3);


disp('
    ##########################################################################
    ');

disp('
    --------------------------------------------------------------------------
    ');

disp('                                PSO Loadflow Analysis ');

disp('
    --------------------------------------------------------------------------
    ');

disp('| Bus |    V    |  Angle  |     Injection      |     Generation
        |          Load      |');

disp('| No  |   pu    |  Degree |    MW    |   MVar   |    MW    |  Mvar
        |     MW     |  MVar | ');

for m = 1:nb
    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');
```

```
98  end
99
100 disp('
        ----------------------------------------------------------------------------
        ');
101
102 fprintf(' Total                    ');fprintf('  %8.3f', sum(Pi));
        fprintf('   %8.3f', sum(Qi));
103
104 fprintf('  %8.3f', sum(Pi+Pl)); fprintf('   %8.3f', sum(Qi+Ql));
105
106 fprintf('  %8.3f', sum(Pl)); fprintf('   %8.3f', sum(Ql)); fprintf('\
        n');
107
108 disp('
        ############################################################################
        ');
109
110 disp('
        ----------------------------------------------------------------------------
        ');
111 disp('                                Line Flow and Losses ');
112 disp('
        ----------------------------------------------------------------------------
        ');
113 disp('|From|To |    P     |    Q     | From| To |    P     |   Q     |
              Line Loss      |');
114 disp('|Bus |Bus|   MW     |   MVar   | Bus | Bus|    MW    |  MVar   |
             MW   |    MVar  |');
115 for m = 1:nl
116     p = fb(m); q = tb(m);
117
118     disp('
            ----------------------------------------------------------------------
            ');
119
120     fprintf('%4g', p); fprintf('%4g', q); fprintf('  %8.3f', Pij(p,q)
            ); fprintf('   %8.3f', Qij(p,q));
121
122     fprintf('%4g', q); fprintf('%4g', p); fprintf('   %8.3f', Pij(q,p
            )); fprintf('   %8.3f', Qij(q,p));
123
124     fprintf(' %8.3f', Lpij(m)); fprintf('   %8.3f', Lqij(m));
125     fprintf('\n');
126 end
127
128 disp('
        ----------------------------------------------------------------------------
        ');
129
130 fprintf('   Total Loss
                                                        ');
131
132 fprintf('  %8.3f', sum(Lpij)); fprintf('   %8.3f', sum(Lqij));
        fprintf('\n');
133
134 disp('
        ----------------------------------------------------------------------------
```

```
    ');
135
136 disp('
    ##########################################################################
    ');
137 clearvars -global
```

Description for above code :

1. This function operates at the end of pso.m ie. at the end of PSO operation. takes as parameters all the final values calculated from PSO operation and also takes in values from above functions to calculate Bus current injections, Line current flows, Line power flows and Bus Power Injections. finally displays all calculated values and final load flow analysis.

IMPROVED PSO APPLIED TO LOAD FLOW STUDIES

improvedpso.m

```
lear all
clc
kount=0;
lim=15;
s1=0.493;
nbus = 5;
BMva=100;                    % Base MVA..
busd = busdatas(nbus);       % Calling busdatas
Y = ybusppg(nbus);           % Calling ybusppg.m
Qg =busd(:,6)/BMva;         % generated reactive power.
Pg =busd(:,5)/BMva;         % gen  erated real power
Ql =busd(:,8)/BMva;         % load reactive power
Pl =busd(:,7)/BMva;         % load real power
Qlim1 = busd(:,9)/BMva;
Qlim2 = busd(:,10)/BMva;
Q =Qg -Ql;                  % Qi =(QGi     Qli)
P =Pg -Pl;                  % Pi =(PGi     Pli)
Qsp =Q    ;                 % Q specified
Psp =P   ;                   % P Specified
Qmin=Qlim1(3);
Qmax=Qlim2(3);
B =imag(Y)                 % Susceptance matrix..
G =real(Y)                 % Conductance matrix..

%---------------------PSO PARAMETERS INITIALIZATION --------------%
p=100;                     % no of particle
it=1000;
rp=1.1;
T=12;
rg=1.1;
rf=1;
f=zeros(p,it);
fp=zeros(p,1);
thp=zeros(p,5);
thg=zeros(p,5);
vp=zeros(p,5);
vg=zeros(p,5);
v=zeros(p,it,5);
th=zeros(p,it,5);
vth=zeros(p,it,5);
vv=zeros(p,it,5);
a=.05;
b=-0.05;
vth(:,1,:)=a+(b-a)*rand(p,5); %initial velocity of theta vector%
a=.05;
b=-0.05;
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);

 for j=1:p
 Q =zeros(nbus,1);
 P =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

 % 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;
```

```
108    %objective function value
109    f(j,1)=sum(MPS)+sum(MQS);
110    kount=kount+1;
111   end
112
113   %Initial personal best values
114  for i=1:p
115      for k=2:5
116          thp(i,k)=th(i,1,k);
117      end
118      for k=2:5
119          vp(i,k)=v(i,1,k);
120      end
121  end
122  %for Initial Global best values updation
123  fmin=min(f(:,1));
124  for k=1:p
125      if f(k,1)==fmin
126          gb=k;
127      else
128      end
129  end
130  %Initial global best value
131
132  for k=1:p
133      for j=2:5
134          thg(k,j)=th(gb,1,j);
135      end
136         for j=2:5
137          vg(k,j)=v(gb,1,j);
138      end
139  end
140  fgm = min(f(:,1));
141   Q3=zeros(p,it);
142  for i=1:it
143      %Repositioning
144
145     %for inertia weight W
146      wmax=.4;
147      wmin=.4;
148      w=wmax-((wmax-wmin)*i/it);
149      %w =0.1+(rand()/2);
150        %velocity update
151        %position update
152  for j=1:p
153          for k=2:5
154
155          vth(j, i+1 ,k) = w*vth(j,i,k) + rp*(thp(j,k)-th(j,i,k)) + rg
                 *(thg(j,k)-th(j,i,k));
156  %        if vth(j, i+1 ,k)<-0.1
157  %            vth(j, i+1 ,k)=-0.1;
158  %         end
159  %         if vth(j, i+1 ,k)>0.1
160  %            vth(j, i+1 ,k)=0.1;
161  %         end
162          th(j,(i+1),k) = th(j,i,k) + rf*vth(j,(i+1),k);
163          %Repositioning
164          if i==lim+1
```

```
165             th(j,i+1,k)=(s1*th(gb,lim,k)*f(gb,lim)^-1+thp(j,k)*fp(j)
                    ^-1)/(s1*f(gb,lim)^-1+fp(j)^-1);
166         end
167         end
168
169
170         for q=2:5
171         vv(j,(i+1),q) = w*vv(j,i,q) + rp*(vp(j,q)-v(j,i,q)) + rg*(vg(
                j,q)-v(j,i,q));
172         if  vv(j,(i+1),q)<-0.1
173               vv(j,(i+1),q)=-0.1;
174         end
175                if  vv(j,(i+1),q)>0.1
176               vv(j,(i+1),q)=0.1;
177                end
178
179         v(j,(i+1),q) = v(j,i,q) + rf*vv(j,(i+1),q);
180         %Repositioning
181         if i==lim+1
182             th(j,i+1,k)=(s1*th(gb,lim,k)*f(gb,lim)^-1+thp(j,k)*fp(j)
                    ^-1)/(s1*f(gb,lim)^-1+fp(j)^-1);
183         end
184         end
185         for q=3
186             if PVIND(j,1)==0
187                 v(j,(i+1),q)=1.04;
188             end
189         end
190
191
192 end
193
194
195 %th(:,i,5)
196
197 %objective function value
198 for j=1:p
199  Q =zeros(nbus,1);
200  P =zeros(nbus,1);
201  MPS=zeros(p,1);
202  MQS=zeros(p,1);
203
204      for m =2:nbus
205         for k =1:nbus
206             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)));
207
208         end
209      end
210         for m = 2:5
211         for k = 1:nbus
212             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)));
213
214         end
215         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);
  kount=kount+1;


end
          %personal best values updatio
for j=1:p
 Q =zeros(nbus,1);
 P =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

 % 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
```

```
272         if Q(3)>Qmax;
273             Q(3)=Qmax;
274             PVIND(j,1)=1;
275    else PVIND(j,1)=0;
276         end
277
278    %reactive power mismatch
279    MQ=Q-Qsp;
280    MQ(3)=0;
281    MQS=MQ.^2;
282    %objective function value
283    %objective function value
284    fp(j)=sum(MPS)+sum(MQS);
285    kount=kount+1;
286  end
287  %personal best value updation
288          for k=1:p
289              for m=2:5
290                 if f(k,i)<fp(k)
291                 thp(k,m)=th(k,i,m);
292                 else
293              end
294
295          end
296
297         end
298
299          for k=1:p
300               for m=2:5
301               if f(k,i)<fp(k)
302               else
303               vp(k,m)=v(k,i,m);
304              end
305
306          end
307        end
308
309     %for Global best values updation
310      if min(f(:,(i+1)))<fgm
311          fgm=min(f(:,(i+1)));
312      else
313      end
314
315    for m=2:5
316
317          for k=1:p
318          if f(k,i)==fgm
319              for l=2:p
320                  thg(l,m) = th(k,i,m);      %global best values
321                end
322          else
323         end
324
325     end
326    end
327         for m=2:5
328
329          for k=1:p
```

```
330          if f(k,i)==fgm
331              for l=2:p
332                    vg(l,m) = v(k,i,m);           %global best values
333              end
334          else
335          end
336
337     end
338         end
339
340
341 %  print = [th(:,i,1) th(:,i,2) th(:,i,3) th(:,i,4) th(:,i,5)  v(:,i
        ,1) v(:,i,2) v(:,i,3) v(:,i,4) v(:,i,4)  f(:,i)];
342     i
343      f(gb,i)
344  %     disp('  th1        th2         th3         th4       th5         v1
              v2         v3        v4       v5           f')
345
346    % disp(print)
347
348         %stopping
349
350      gb1=gb;
351 for k=1:p
352      if f(k,1+1)==fgm
353          gb1=k;
354      else
355      end
356 end
357      if (f(gb1,i+1)<=10^(-T))
358
359          break
360      end
361
362 vth(gb1,(i+1),2);
363 end
364 %Q3(:,it)
365 bus=zeros(5,10) ;
366 bus(1,3)=v(gb,i,1);
367 bus(2,3)=v(gb,i,2);
368 bus(3,3)=v(gb,i,3);
369 bus(4,3)=v(gb,i,4);
370 bus(5,3)=v(gb,i,5);
371 %bus angle updation
372 bus(1,4)=th(gb,i,1);
373 bus(2,4)=th(gb,i,2);
374 bus(3,4)=th(gb,i,3);
375 bus(4,4)=th(gb,i,4);
376 bus(5,4)=th(gb,i,5);
377 bus(3,6)=Q3(gb1,it)*BMva;
378 x=bus(3,6);
379 V = bus(:,3)    ;        % Specified Voltage..
380 del = bus(:,4) ; % Voltage Angle..
381 %load flow function calling
382 fprintf('kount value is %d\n',kount);
383 loadflow(nbus,V,del,BMva,x);
```

References

- https://www.researchgate.net/publication/224226725/Load/flow/computation-/via/Particle/Swarm/Optimization
- http://www.iraj.in/journal/journal/file/journal/pdf/3-278-147143059870-77.pdf
- https://pdfs.semanticscholar.org/5390 b8ef0957312242199fb4b95c10c94d4cc540.pdf
- https://www.researchgate.net/figure/Classification-of-load-flow-methods-for-new-distribution-networks/fig1/234026001
- https://en.wikipedia.org/wiki/Power-flow/study