CHAPTER 1
INTRODUCTION

1.1 Background of Study

This project entitled "Optimal Parameters of Static Synchronous Series Compensator (SSSC) connected to a power system" is intended to develop an SSSC controller to control the power flow in the transmission line and minimize the transmission line losses. SSSC is able to control both active and reactive powers in an ac system simply by controlling the angular position of injected voltage into the transmission line with respect to the line current [1]. The parameters in the power flow are monitored and controlled to their optimized levels by using a SSSC controller.

1.2 Problem Statement

Nowadays, the demand for electricity supply has been increasing to meet the world's needs. Many analysis and researches of options available for maximising the existing transmission assets have been conducted that can substitute for conventional solutions that have slow response times and high maintenance costs [2].

Currently, most of the world's electrical power systems are widely interconnected. Network interconnection is made for economic reasons that are to reduce the cost of electricity and to improve reliability of power supply. Transmission interconnections enable taking advantage of diversity of loads, availability of sources, and fuel price in order to supply electricity to the loads at

minimum cost with a required reliability [2].

While power flows in some of the transmission lines are well below their normal limits, other lines are overloaded, which has an overall effect on deteriorating voltage profiles and decreasing system stability and security. Therefore, it becomes very crucial to control the power flow along the transmission lines to meet the needs of power transfer [3].

1.3 Objective of Study

The objectives of this project report are as follows:

1. To develop an SSSC controller to control the power flow in transmission lines.
2. To optimize the parameters of SSSC using an intelligent optimization technique.
3. To construct the sizing of SSSC controller parameters with the purpose of minimizing the transmission line losses in a network.

1.4 Scope of Study

The study on the optimization of parameters of SSSC connected to a power system and implementation is to be completed within approximately one year timeframe (two semesters). The scope for phase 1 of the project, which is research on power flow, SSSC and optimization problem solution, is completed by the end of first semester. Phase 2 which is the implementation of SSSC into power system and design programming is started after phase 1 is completed.

When the software implementation is completed, the testing is to be performed when all parameters set on the power systems can be controlled.

Therefore, the accuracy of the results obtained and the outcome of SSSC can be observed and assessed.

CHAPTER 2
LITERATURE REVIEW AND THEORY

2.1 Flexible AC Transmission System (FACTS)

Flexible AC Transmission System (FACTS) is an alternating current transmission system incorporating power electronic-based and other static controllers to enhance the controllability and increase the power transfer capability [1]. The introduction of FACTS in a power system improves the stability, reduces losses and improves the load ability of the system. With FACTS technology, such as Static Synchronous Series Compensator (SSSC), Interline Power Flow Controller (IPFC) and Unified Power Flow Controller (UPFC), the bus voltages, line impedances and phase angles in the power system are regulated rapidly and flexibly [3], thanks to their series-connected converter. The explanations for different controllers are provided in the following sections.

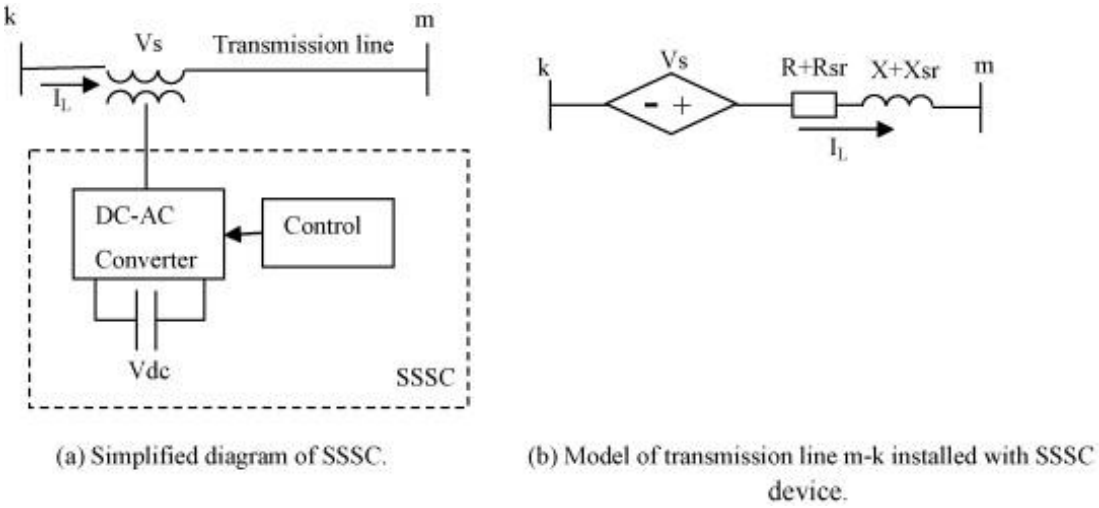
2.2 FACTS Controllers

2.2.1 Static Synchronous Series Compensator (SSSC)

SSSC is a solid-state voltage source converter that generates a controllable ac voltage source and connected in series to power transmission lines in a power system [1]. The main function of SSSC is to compensate for the voltage drop across the impedance in a transmission line. An SSSC injects a voltage in series with the line transmission voltage which is always kept in quadrature with the line current so that the SSSC can exchange only reactive power with the system. The injected voltage emulates an inductive or a capacitive reactance so as to influence the power flow in

the transmission lines.

The SSSC is generally connected in series with the transmission lines. It is operated without an external electrical energy source as a series compensator whose output voltage is in quadrature with, and controllable independently of, the line current with the purpose of increasing or decreasing the overall reactive voltage drop across the line and thereby controlling the transmitted electric power.



(a) Simplified diagram of SSSC.

(b) Model of transmission line m-k installed with SSSC device.

Figure 1: Basic diagram of Static Synchronous Series Compensator

2.2.2 Interline Power Flow Controller (IPFC)

IPFC is a combination of two or more SSSCs which are coupled via a common dc link to facilitate bi-directional flow of real power between the ac terminals of the SSSCs, and are controlled to provide independent reactive compensation for the adjustment of real power flow in each line and maintain the desired distribution of reactive power flow among the lines [1].

2.2.3 Unified Power Flow Controller (UPFC)

UPFC is a combination of static synchronous compensator (STATCOM) and an SSSC which are coupled via a common dc link, to allow bi-directional flow of real power between the series output terminals of the SSSC and the shunt output terminals of the STATCOM. Both the devices are controlled together to provide concurrent real and reactive series line compensation without an external electrical energy source. The UPFC, by means of angularly unconstrained series voltage injection, is able to control, concurrently or selectively, the transmission line voltage, impedance and angle, or alternatively, the real and reactive power flow in the line [1].

2.3 Voltage-source Converter

Converter-based FACTS controllers have two principle types of converters which are voltage-source converters and current-source converters. From, overall cost and performance point of view, the voltage-source controllers are preferred for converter-based FACTS controllers. Basically, a voltage-source converter (VSC) generates ac voltage from a dc voltage. The magnitude, the phase angle, and the frequency of the output voltage are controlled by using this converter. The three VSC-based controllers above share similar power system control capabilities. They are able to regulate either nodal voltage magnitude or injection of reactive power at one of its terminals, and active power flow through the controller [1] .

2.4 Power Flow Studies

Power flow studies deal with the steady-state analysis of an interconnected power system during normal operation. The system is assumed to be operating under balanced condition and is represented by a single-phase network. Power flow is a function of transmission line impedance, the magnitude of the sending end and

receiving end voltages and the phase angle between the voltages [3] [4]. By controlling one or a combination of the power flow arrangements, it is possible to control the active and the reactive power flow in the transmission line.

2.5 Static Synchronous Series Compensator

As discussed earlier, the primary function of SSSC is to control the power flow in the transmission line. SSSC is used to control the following parameters:

a) The active power flow of the transmission line
b) The reactive power flow of the transmission line
c) The bus voltage, and
d) The impedance of the transmission line [5].

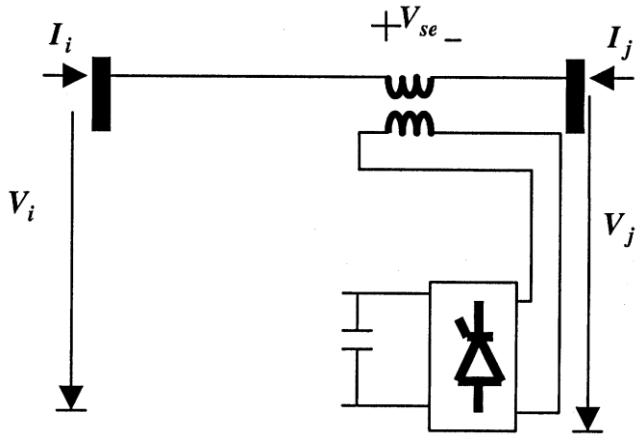


Figure 2: SSSC Operation principles

An SSSC usually consists of a coupling transformer, an inverter and a capacitor. As shown in Figure 2, the SSSC is series connected with a transmission line through the coupling transformer. It is assumed here that the transmission line is series connected with the SSSC via its bus j. The active and reactive power flows of the SSSC branch i-j entering the bus j are equal to the sending end active and reactive power flows of the transmission line, respectively. In principle, the SSSC generates and inserts a series voltage, which is regulated to change the impedance

(more precisely reactance) of the transmission line. In this way, the power flow of the transmission line or the voltage of the bus, in which the SSSC is connected with, is controlled [4].

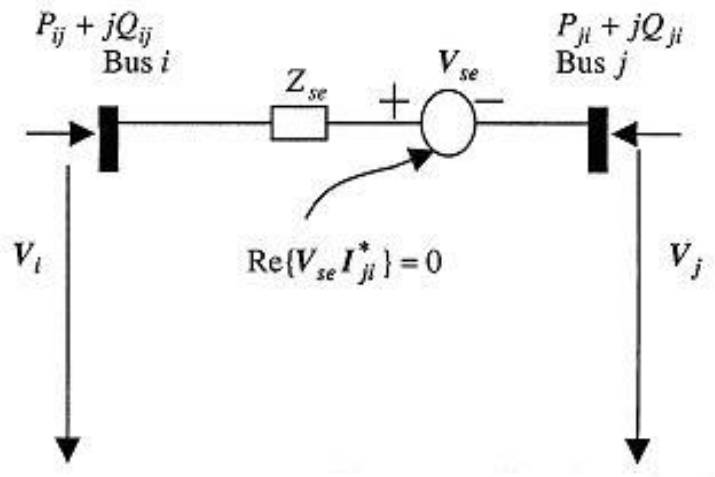


Figure 3: SSSC equivalent circuit

An equivalent circuit of the SSSC as shown in Figure 3 is derived based on the operation principle of the SSSC. In the equivalent circuit, the SSSC is represented by a voltage source, V_{se} in series with a transformer's impedance. In the practical operation of the SSSC, V_{se} can be regulated to control the power flow of line i-j or voltage of bus i or j. In the equivalent circuit, $V_{se} = |V_{se}| \angle \theta_{se}, V_i = |V_i| \angle \theta_i$, and $V_j = |V_j| \angle \theta_j$.

2.6 Method of Solving an Optimization Problem

In order to find the optimal sizing of the SSSC controller, this subject is formed as an optimization problem with the objective of minimizing the transmission line losses in a network. The problem is solved by using different methods such as Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) techniques which are widely used in many engineering applications.

2.6.1 Genetic Algorithm (GA)

A genetic algorithm (GA) is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover to form a solution to a problem.

To use a genetic algorithm, the solution to a particular problem must be represented as a genome (or chromosome). GA then creates a population of solutions and applies genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s).There are six important aspects to be determined when using GA:

1. chromosome (individual) presentation,
2. evaluation of objective function (fitness),
3. creation of the initial population,
4. choice of genetic operators,
5. selection function, and
6. stop criterion

Once these six aspects have been determined, the generic genetic algorithm should work properly [6] [7].

2.6.2 Particle Swarm Optimization Technique

Particle swarm optimization (PSO) is an algorithm modelled on the swarm intelligence that finds a solution to an optimization problem in a search space, or model. This technique which was inspired by social behaviour of bird flocking or fish schooling is not only used for an optimization problem, but also to predict or model social behaviour based on principles of social psychology [7].

PSO method is used to find an optimal solution to an objective function (fitness function) in a search space. The system is initialized with a population of random solutions (particles) and then searches the optimal solution by updating generations. The particles change their position with time or fly through the search space by following the current optimum particles. During flight, each particle adjusts its position according to its own experience, and according to the experience of a neighbouring particle, making use of the best position encountered by itself and its neighbour.

In PSO, there are different types of fitness to describe the best solution to a problem:

a. pbest - the best solution (fitness) a particle has achieved so far.
b. gbest - the best value that is tracked by the particle swarm optimizer, obtained so far by any particle in the population.
c. lbest – the best value when a particle takes part of the population as its topological neighbours.

2.7 Transmission Line Loss

Consider a line connecting two buses i and j, the line current at bus i which is positive and measured in the direction of i to j is

$$I_{ij} = Y_{ij}\left(V_i - V_j\right) \quad \text{(a)}$$

The line current at bus j, measured in the direction of j to i is,

$$I_{ji} = Y_{ij}\left(V_j - V_i\right) \quad \text{(b)}$$

The active powers from bus i to j and from bus j to i are

$$P_{ij} = V_i I_{ij} \quad \text{(c)}$$

$$P_{ji} = V_j I_{ji} \quad \text{(d)}$$

The power loss in line i-j is the algebraic sum of the active power flows in (c) and (d),

$$P_{Lij} = P_{ij} + P_{ji} \qquad \text{(e)}$$

CHAPTER 3
METHODOLOGY

There are some procedures to be followed in order to carry out and implement this project. This is to ensure that the project is accomplished within the given timeframe.

3.1 Procedure Identification

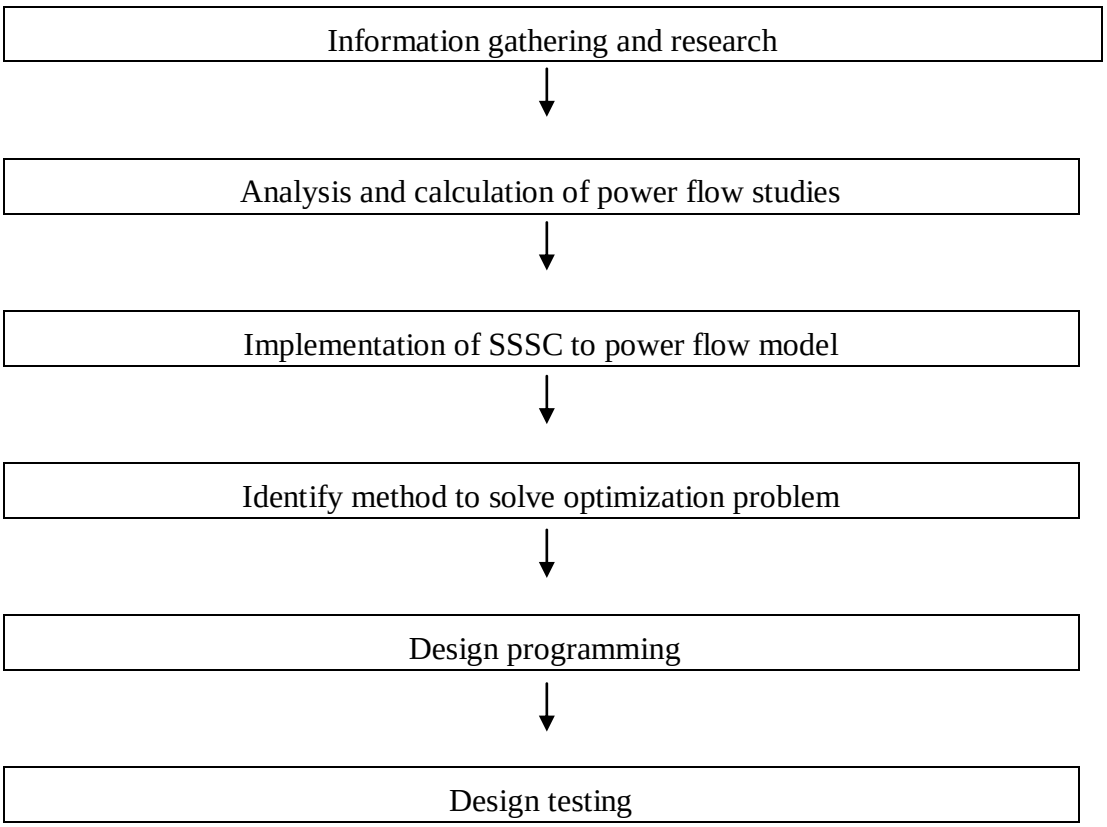


Figure 4: Flow chart of project procedures

3.1.1 Information Gathering and Research

At this stage, information and data included are the study of optimized power flow, operation principles of SSSC, SSSC controller, PSO technique and implementation methods.

3.1.2 Analysis and Calculation of Power Flow Studies

The important parameters of the power flow to be controlled are to be identified and analyzed. Power flow studies and analysis are performed by using the Newton-Raphson method to solve the power flow problems. It is observed from many power system problems that the Newton-Raphson method is used in the power flow problem since it is found to be more efficient and practical for large power systems. This method is reported to be a most widely used and accurate method for solving simultaneous nonlinear algebraic equations.

3.1.2.1 Power Flow Equations by using Newton-Raphson Method

The admittance matrix in a power system relates to current injections at a bus to the bus voltages. The equation describing the performance of the network in the bus admittance form is given by

$$I = YV \tag{1}$$

where I = the bus current vector

V = the bus voltage vector

Y = the bus admittance matrix

The expanded form of the equations is:

$$\begin{bmatrix} \mathbf{I_1} \\ \mathbf{I_2} \\ \vdots \\ \mathbf{I_N} \end{bmatrix} = \begin{bmatrix} \mathbf{Y_{11}} & \mathbf{Y_{12}} & \cdots & \mathbf{Y_{1N}} \\ \mathbf{Y_{21}} & \mathbf{Y_{22}} & \cdots & \mathbf{Y_{2N}} \\ \vdots & \vdots & & \vdots \\ \mathbf{Y_{N1}} & \mathbf{Y_{N2}} & \cdots & \mathbf{Y_{NN}} \end{bmatrix} \begin{bmatrix} \mathbf{V_1} \\ \mathbf{V_2} \\ \vdots \\ \mathbf{V_N} \end{bmatrix} \tag{2}$$

Considering a power system with two buses k and m, the complex power at bus k is

$$S_k = P_k + jQ_k = V_k I_k^* \tag{3}$$

By rearranging the equation, the current injection at bus k is expressed as

$$I_k = \frac{P_k - jQ_k}{V_k^*} \tag{4}$$

Now, the current at bus k is written as

$$I_k = \sum_{m=1}^{n} Y_{km} V_m \tag{5}$$

Expressing the equation (5) in polar form,

$$I_k = \sum_{m=1}^{n} |Y_{km}||V_m| \angle \theta_{km} + \delta_m \tag{6}$$

The complex power at bus k is

$$P_k - jQ_k = V_k^* I_k \tag{7}$$

Substituting I_k of equation (6) into equation (7),

$$P_k - jQ_k = |V_k| \angle -\delta_k \sum_{m=1}^{n} |Y_{km}||V_m| \angle \theta_{km} + \delta_k \tag{8}$$

Separating the real and imaginary parts,

$$P_k = \sum_{m=1}^{n} |V_k||V_m||Y_{km}| \cos(\theta_{km} - \delta_k + \delta_m) \tag{9}$$

$$Q_k = -\sum_{m=1}^{n} |V_k||V_m||Y_{km}| \sin(\theta_{km} - \delta_k + \delta_m) \tag{10}$$

Equations (9) and (10) for real and reactive powers, respectively constitute a set of nonlinear algebraic equations in terms of independent variables, voltage magnitude in per unit, and phase angle in radians. Both of the equations are linearized on compact form by Taylor's first order approximation results in

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} \frac{\partial P}{\partial \delta} & \frac{\partial P}{\partial |V|} \\ \frac{\partial Q}{\partial \delta} & \frac{\partial Q}{\partial |V|} \end{bmatrix} \begin{bmatrix} \Delta \delta \\ \Delta |V| \end{bmatrix} \tag{11}$$

To bring symmetry in the elements of the coefficient matrix, $\frac{\Delta |V|}{|V|}$ is taken as problem variable in place of $\Delta |V|$. Then, equation (11) changes to

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} \frac{\partial P}{\partial \delta} & \frac{\partial P}{\partial |V|}|V| \\ \frac{\partial Q}{\partial \delta} & \frac{\partial Q}{\partial |V|}|V| \end{bmatrix} \begin{bmatrix} \Delta \delta \\ \frac{\Delta |V|}{|V|} \end{bmatrix} \tag{12}$$

In symbolic form, the equation (12) is written as

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} H & N \\ M & L \end{bmatrix} \begin{bmatrix} \Delta \delta \\ \frac{\Delta |V|}{|V|} \end{bmatrix} \tag{13}$$

The matrix $\begin{bmatrix} H & N \\ M & L \end{bmatrix}$ is known as Jacobian matrix.

The diagonal and off-diagonal elements of H are

$$\frac{\partial P_k}{\partial \delta_k} = \sum_{m \neq k} |V_k||V_m||Y_{km}| \sin(\theta_{km} - \delta_k + \delta_m) \tag{14}$$

$$\frac{\partial P_k}{\partial \delta_m} = -|V_k||V_m||Y_{km}| \sin(\theta_{km} - \delta_k + \delta_m) \quad \text{j} \neq 1 \tag{15}$$

The diagonal and off-diagonal elements of N are

$$\frac{\partial P_k}{\partial |V_k|} = 2|V_k||Y_{kk}|\cos(\theta_{kk}) + \sum_{m \neq k} |V_m||Y_{km}|\sin(\theta_{km} - \delta_k + \delta_m) \quad (16)$$

$$\frac{\partial P_k}{\partial |V_m|} = |V_k||Y_{km}|\cos(\theta_{km} - \delta_k + \delta_m) \quad j \neq 1 \quad (17)$$

The diagonal and off-diagonal elements of M are

$$\frac{\partial Q_k}{\partial \delta_k} = \sum_{m \neq k} |V_k||V_m||Y_{km}|\cos(\theta_{km} - \delta_k + \delta_m) \quad (18)$$

$$\frac{\partial Q_k}{\partial \delta_m} = -|V_k||V_m||Y_{km}|\cos(\theta_{km} - \delta_k + \delta_m) \quad j \neq 1 \quad (19)$$

The diagonal and off-diagonal elements of L are

$$\frac{\partial Q_k}{\partial |V_k|} = -2|V_k||Y_{kk}|\sin(\theta_{kk}) - \sum_{m \neq k} |V_m||Y_{km}|\sin(\theta_{km} - \delta_k + \delta_m) \quad (20)$$

$$\frac{\partial Q_k}{\partial |V_m|} = -|V_k||Y_{km}|\sin(\theta_{km} - \delta_k + \delta_m) \quad j \neq 1 \quad (21)$$

The solution procedures for Newton Raphson method of power flow analysis are as follows:

1. Read the line data and bus data of the power network; construct the bus admittance matrix.
2. Set k = 0. Assume a starting solution. Usually a flat start is assumed in which all the unknown phase angles are taken as zero and the unknown voltage magnitudes are taken as 1.0 p.u.
3. Compute the mismatch powers i.e. the error vector. If the elements of error vector are less than the specified tolerance, the problem is solved and hence go to Step 7; otherwise proceed to Step 4.
4. Compute the elements of sub-matrices H, N, M and L. Solve

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix}_K = \begin{bmatrix} H & N \\ M & L \end{bmatrix}_K \begin{bmatrix} \Delta\delta \\ \frac{\Delta|V|}{|V|} \end{bmatrix} \quad \text{for} \quad \begin{bmatrix} \Delta\delta \\ \frac{\Delta|V|}{|V|} \end{bmatrix} \tag{22}$$

5. Update the solution as

$$\begin{bmatrix} \boldsymbol{\delta} \\ |\mathbf{V}| \end{bmatrix}_{\mathbf{k+1}} = \begin{bmatrix} \boldsymbol{\delta} \\ |\mathbf{V}| \end{bmatrix}_{\mathbf{k}} + \begin{bmatrix} \boldsymbol{\Delta\delta} \\ \boldsymbol{\Delta}|\mathbf{V}| \end{bmatrix} \tag{23}$$

6. Set k = k + 1 and go to Step 3.
7. Calculate line flows, transmission line loss and slack bus power.

3.1.3 Implementation of SSSC to Power Flow Model

The method of using SSSC is implemented into a load flow model that is used to calculate the power losses and check the system operating constraints such as voltage profile. The model is to be modified to consider the insertion of SSSC devices into the network.

3.1.4 Identify Method to Solve Optimization Problem

The sizing of SSSC controllers in transmission network is formed as an optimization problem and is solved by using the identified optimization technique. After revising two types of evolutionary optimization techniques, GA and PSO that are widely used in power system applications, the PSO technique is chosen by considering its advantages over GA technique. The similarities between PSO and GA are that both algorithms are using population-based search approaches and depend on information sharing among their population members to enhance their search processes.

However, PSO does not have genetic operators like GA such as mutation and crossover. The particles in PSO update themselves with the internal velocity. They

also have memory, which is important to the algorithm. In terms of information sharing mechanism, in GAs, chromosomes share information with each other making the whole population moves like one group towards optimal area. In PSO, only gbest or lbest gives out the information to others making the particles only looks for the best solution. Compared with GA, all the particles tend to converge to the best solution quickly in most cases, resulting in global optimal solution [8] [9].

Also, PSO is more computationally efficient in a sense that the coding is less complicated than the GA since it contains less function evaluations than the GA.

The PSO algorithm used follows the following procedure in solving the optimization problem defined in this project.

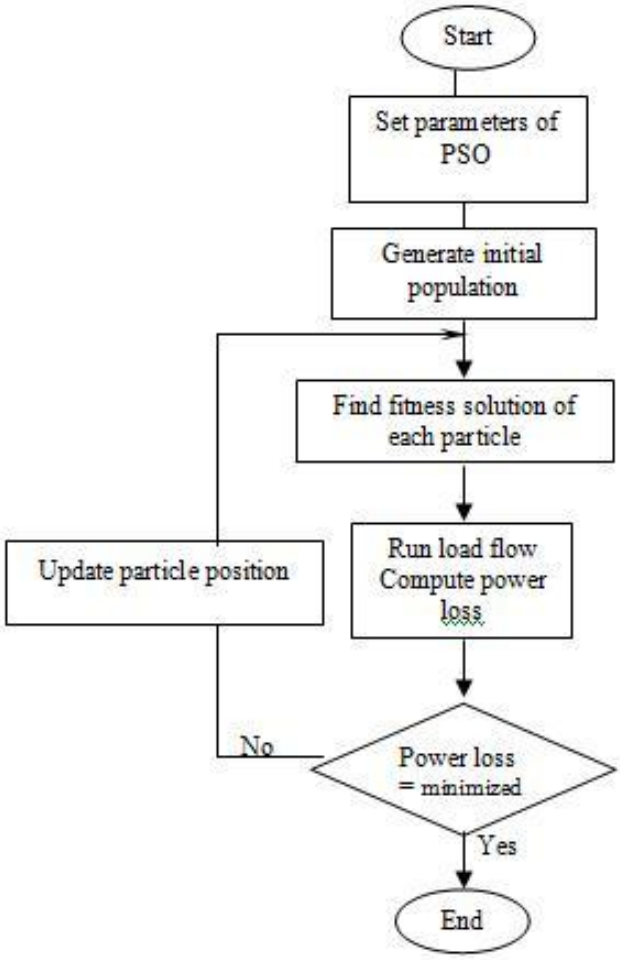


Figure 5: Flow chart of PSO algorithm

Based on Figure 5, the PSO algorithm is performed by following the procedures as follows:

1. Set the parameters of PSO such as the swarm size and number of iteration.
2. Generate the initial population with random solutions.
3. Find fitness solution of each particle based on pbest, lbest and gbest.
4. Run the power flow model and determine the power loss of the system.
5. Perform the position check. If the power loss is minimized, the final position is the optimal parameters of SSSC. If the power loss is not minimized, update the particle position and go back to step 3 until the power loss is minimized.

For this optimization problem, the number of particles used is 20 and number of iterations is 50. The objective function for this problem is

$$MinF = \sum_{i=1}^{n} P_L$$

where n is the number of buses.

The parameters constraints set for the problem are as follows:

$$0 < V_{se} < 0.15 pu$$
$$0 < \theta_{se} < 90°$$

3.1.5 Design and programming

The programming and source code are to be developed in C language into its designed system according to the proposed methods.

3.1.6 Design Testing

The controller is to be tested by using the identified parameters to verify the validity of the design. The test includes the determination of the optimal parameters of magnitude and phase angle of the voltage injection of the SSSC into the network

and also the verification for the objectives of the project which are to improve the voltage profile and minimize the transmission line loss of the network.

3.2 Tools Required

The software tool that is used in this project is MATLAB software version 7.1 to model the power flow and the determination of the optimal parameters of SSSC controller. C programming is used to develop the optimization problem solution's coding in the M-file of MATLAB.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Implementation of SSSC to Power Flow Model

Considering the insertion of an SSSC into the power system network, the existing load flow model needs to be modified to include the newly added parameters of SSSC. Referring to Figure 3 in Section 2.5, the voltage source equivalent of SSSC is represented by

$$V_{se} = |V_{se}|(\cos\theta_{se} + j\sin\theta_{se}) = |V_{se}|\angle\theta_{se} \tag{24}$$

Based on the equivalent circuit of Figure 3, the following bus admittance matrix is constructed:

$$\begin{bmatrix} I_i \\ I_j \end{bmatrix} = \begin{bmatrix} Y_{se} & -Y_{se} & -Y_{se} \\ -Y_{se} & Y_{se} & Y_{se} \end{bmatrix} \begin{bmatrix} V_i \\ V_j \\ V_{se} \end{bmatrix} \tag{25}$$

The bus admittance matrix (25) is used to derive the mathematical model of the SSSC for inclusion in the power flow Newton Raphson method.

Based on the equivalent circuit and the bus admittance matrix, the complex power at bus i is written as

$$S_i = V_i I_i^* = V_i\left[Y_{ii}^* V_i^* - Y_{ii}^* V_{se}^* + Y_{ij}^* V_j^*\right] \tag{26}$$

From equations (4) until (8), correct substitutions are performed until the following expressions for active and reactive powers are obtained for node i,

$$P_i = V_i \sum_{n=i,j} V_n \left[G_{in} \cos(\theta_i - \theta_n) + B_{in} \sin(\theta_i - \theta_n) \right] - V_i V_{se} \left[G_{ij} \cos(\theta_i - \theta_{se}) + B_{ij} \sin(\theta_i - \theta_{se}) \right] \quad (27)$$

$$Q_i = V_i \sum_{n=i,j} V_n \left[G_{in} \sin(\theta_i - \theta_n) - B_{in} \cos(\theta_i - \theta_n) \right] - V_i V_{se} \left[G_{ij} \sin(\theta_i - \theta_{se}) - B_{ij} \cos(\theta_i - \theta_{se}) \right] \quad (28)$$

An identical set of active and reactive power equations are obtained for node j, where the letter i and j are interchanged.

Referring to equations (11) and (12), the power equations are linearized giving the Jacobian matrix,

$$\begin{bmatrix} \Delta P_i \\ \Delta P_j \\ \Delta Q_i \\ \Delta Q_j \\ \Delta P_{ij} \\ \Delta Q_{ij} \end{bmatrix} = \begin{bmatrix} \frac{\partial P_i}{\partial \theta_i} & \frac{\partial P_i}{\partial \theta_j} & \frac{\partial P_i}{\partial |V_i|}|V_i| & \frac{\partial P_i}{\partial |V_j|}|V_j| & \frac{\partial P_i}{\partial \theta_{se}} & \frac{\partial P_i}{\partial |V_{se}|}|V_{se}| \\ \frac{\partial P_j}{\partial \theta_i} & \frac{\partial P_j}{\partial \theta_j} & \frac{\partial P_j}{\partial |V_i|}|V_i| & \frac{\partial P_j}{\partial |V_j|}|V_j| & \frac{\partial P_j}{\partial \theta_{se}} & \frac{\partial P_j}{\partial |V_{se}|}|V_{se}| \\ \frac{\partial Q_i}{\partial \theta_i} & \frac{\partial Q_i}{\partial \theta_j} & \frac{\partial Q_i}{\partial |V_i|}|V_i| & \frac{\partial Q_i}{\partial |V_j|}|V_j| & \frac{\partial Q_i}{\partial \theta_{se}} & \frac{\partial Q_i}{\partial |V_{se}|}|V_{se}| \\ \frac{\partial Q_j}{\partial \theta_i} & \frac{\partial Q_j}{\partial \theta_j} & \frac{\partial Q_j}{\partial |V_i|}|V_i| & \frac{\partial Q_j}{\partial |V_j|}|V_j| & \frac{\partial Q_j}{\partial \theta_{se}} & \frac{\partial Q_j}{\partial |V_{se}|}|V_{se}| \\ \frac{\partial P_{ij}}{\partial \theta_i} & \frac{\partial P_{ij}}{\partial \theta_j} & \frac{\partial P_{ij}}{\partial |V_i|}|V_i| & \frac{\partial P_{ij}}{\partial |V_j|}|V_j| & \frac{\partial P_{ij}}{\partial \theta_{se}} & \frac{\partial P_{ij}}{\partial |V_{se}|}|V_{se}| \\ \frac{\partial Q_{ij}}{\partial \theta_i} & \frac{\partial Q_{ij}}{\partial \theta_j} & \frac{\partial Q_{ij}}{\partial |V_i|}|V_i| & \frac{\partial Q_{ij}}{\partial |V_j|}|V_j| & \frac{\partial Q_{ij}}{\partial \theta_{se}} & \frac{\partial Q_{ij}}{\partial |V_{se}|}|V_{se}| \end{bmatrix} = \begin{bmatrix} \Delta\theta_i \\ \Delta\theta_j \\ \frac{\Delta V_i}{V_i} \\ \frac{\Delta V_j}{V_j} \\ \Delta\theta_{se} \\ \frac{\Delta V_{se}}{V_{se}} \end{bmatrix} \quad (29)$$

From equation (29), the diagonal and off-diagonal elements of the Jacobian matrix are derived based on the power equations of each bus.

Following the solution procedures of Newton-Raphson method for power flow analysis in Section 3.1.2.1, the power mismatch equations at node i and k are computed as follows:

$$\Delta P_{i,j} = (P_{i,j})^{sp} - (P_{i,j})^{calc}$$
$$\Delta Q_{i,j} = (Q_{i,j})^{sp} - (Q_{i,j})^{calc} \quad (30)$$

Note:

$\left(P_{i,j}\right)^{sp}$ and $\left(Q_{i,j}\right)^{sp}$ are the specified active and reactive powers while $\left(P_{i,j}\right)^{calc}$ and $\left(Q_{i,j}\right)^{calc}$ are the calculated active and reactive powers.

After solving for the $\begin{bmatrix} \Delta\theta \\ \dfrac{\Delta|V|}{|V|} \end{bmatrix}$, the voltage magnitude and phase angle are updated using the following matrix expression,

$$\begin{bmatrix} \theta_{i,j,se} \\ |V_{i,j,se}| \end{bmatrix}_{k+1} = \begin{bmatrix} \theta_{i,j,se} \\ |V_{i,j,se}| \end{bmatrix}_{k} + \begin{bmatrix} \Delta\theta_{i,j,se} \\ \dfrac{\Delta|V_{i,j,se}|}{|V_{i,j,se}|}|V_{i,j,se}| \end{bmatrix} \tag{31}$$

Then, the procedures are implemented to calculate the line flows, transmission line loss and the bus powers.

4.2 Results

The coding of MATLAB for the Newton-Raphson method and the modified Newton-Raphson method are tested on the IEEE 14-bus benchmark test system and the following results are yielded.

Table 1: The system transmission line loss

Condition	Without SSSC	With SSSC
Power Loss (MW)	13.503	9.018

Table 2: Comparison of the voltage profiles without SSSC and with SSSC

Bus Number	Voltage (p.u.)	
	Without SSSC	With SSSC
1	1.06	1.06
2	1.045	1.05
3	1.01	1.0222
4	1.019	1.0306
5	1.02	1.03
6	1.07	1.0731
7	1.062	1.0639
8	1.09	1.09
9	1.056	1.0536
10	1.051	1.0606
11	1.057	1.06
12	1.055	1.0598
13	1.050	1.06
14	1.036	1.0383

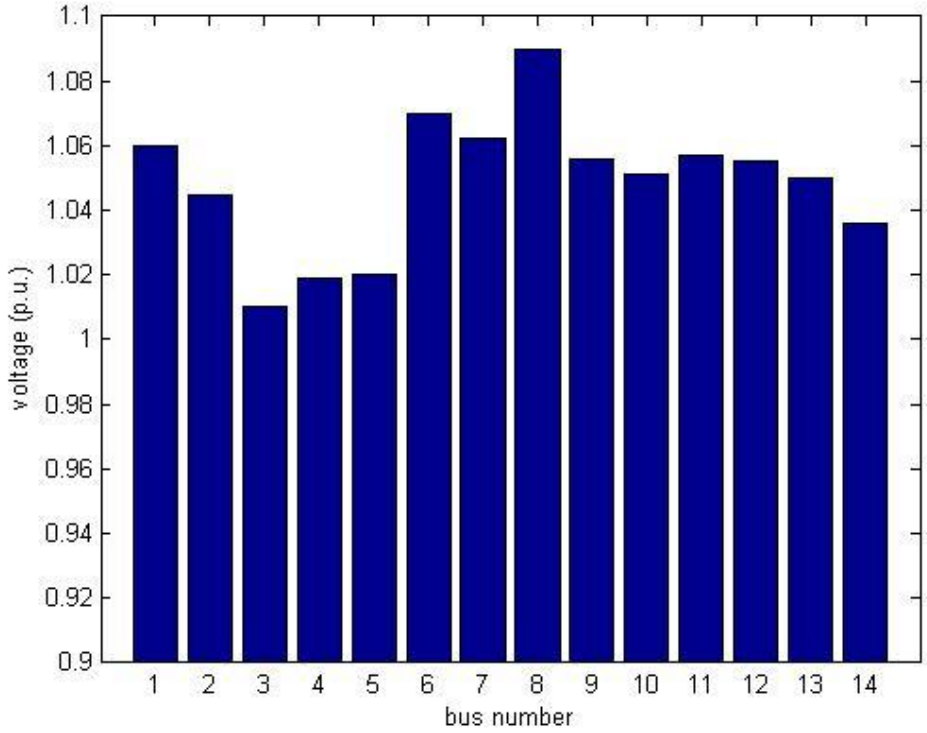


Figure 6: Voltage profile of the 14-bus system without SSSC

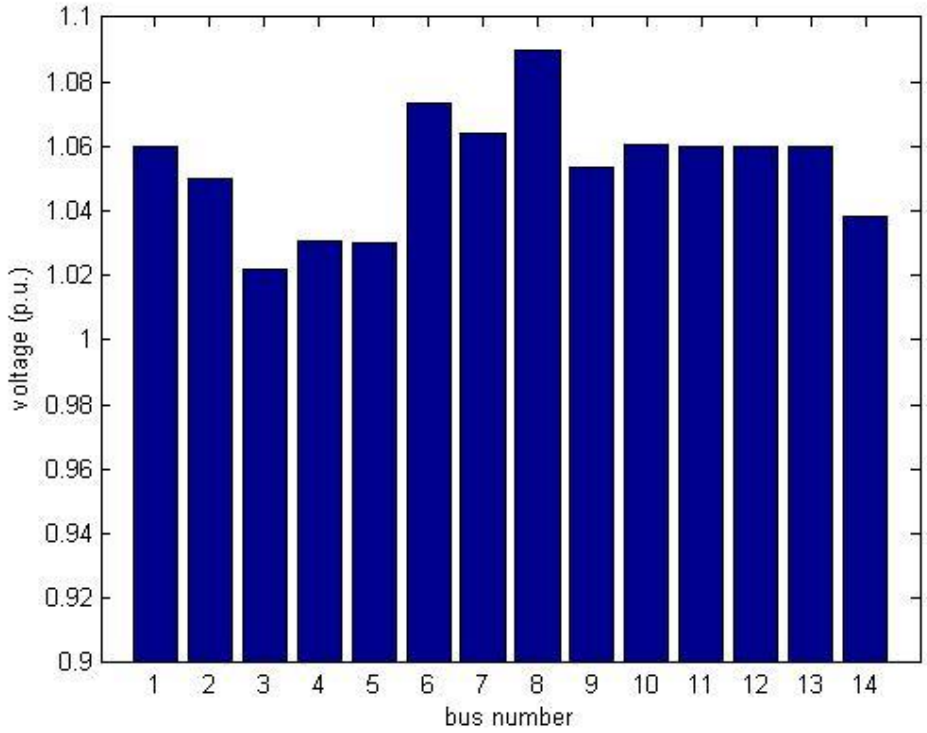


Figure 7: Voltage profile of the 14-bus system with SSSC

Table 3: The control parameters of SSSC

Condition	Before iteration	After iteration
V_{se} (p.u.)	0.10	0.15
θ_{se} (Degree)	90	88

4.3 Discussion

From the results obtained in Table 1, the transmission line loss of the IEEE 14-bus system before the insertion of SSSC is 13.503 MW. After the SSSC device is inserted in series with the system, the transmission line loss is reduced to 9.018 MW. This result shows that the effect of installing an SSSC controller in the power system network, a large reduction in transmission line loss is obtained.

The voltage profile graphs are constructed for both conditions when the SSSC device is not connected and is connected in the power network. From the comparison of the both Figures 5 and 6, the voltages are improved at certain buses after the SSSC is connected to the system, hence improving the performance of the system.

Next, the PSO technique is included in the power flow mathematical model for when the SSSC is connected between the buses of IEEE 14-bus system. The PSO method is set with 20 numbers of particles and 50 iterations. The SSSC parameters limits considered for the voltage is from 0 to 0.15 p.u. and for the phase angle is from 0 to 90 degrees. After the iterating process by the PSO, the optimal parameters obtained for the SSSC controller are 0.15 p.u. for voltage while 88 degrees for phase angle. These values of the SSSC controller yield the optimum performance of the system while minimizing the transmission line losses of the network.

CHAPTER 5
CONCLUSION AND RECOMMENDATION

5.1 Conclusion

SSSC controller has multiple attributes of what it can do in terms of controlling the voltage, power flow, stability and so on. It can control power flow as ordered, reduce reactive power flows allowing transmission lines to carry more active power and increase utilization of lowest cost generation.

This project is focused on optimal sizing of the SSSC controller using the implementation of both theoretical and practical knowledge that have been defined in this report. The power flow problem follows the Newton-Raphson method into its solution and an optimization technique which is Particle Swarm Optimization technique is required in determining the optimal parameters of the SSSC controller. The Newton-Raphson power flow algorithm was modified to consider the insertion of SSSC into the network.

The MATLAB software is used to model and simulate the transmission system with and without SSSC whereby the effect of SSSC to reduce the transmission line loss and improve the voltage profile, are observed. The Particle Swarm Optimization technique used has determined the optimal magnitude and phase angle of SSSC's series injected voltage into the network to minimize the transmission line loss in the network.

5.2 Recommendation

For future studies, two suggestions are SSSC controller to be applied and tested on larger power systems to determine the optimal parameters of the SSSC controller and optimal location in which the controller can be inserted and also, more type of FACTS controller can be used on the same case study so that the performance of each controller can be observed and compared.

REFERENCES

[1] Hingorani, N. G. and Gyugyi, L., 2000, *Understanding FACTS: Concepts and Technology of Flexible AC Transmission Systems*, New York, Wiley-Interscience

[2] Acha, E., Fuerte-Esquivel, C. R., Ambriz-Perez, H. and Angeles-Camacho, C., 2004, *FACTS: Modelling and Simulation in Power Networks*, UK, John Wiley & Sons

[3] El-Zonkoly, A., 2008, "Optimal sizing of SSSC controllers to minimize transmission loss and a novel model of SSSC to study transient response", http://www.sciencedirect.com

[4] Saadat, H., 2004, *Power System Analysis*, Singapore, McGraw Hill

[5] Zhang, X. P., 2003, "Advanced modelling of the multicontrol functional static synchronous series compensator (SSSC) in Newton power flow", http://ieeexplore.ieee.org

[6] Baskaran, J. and Palanisamy, V., 2005, "Genetic algorithm applied to optimal location of FACTS device in a power system network considering economic saving cost", http://www.acadjournal.com

[7] Hassan, R., Conahim, B. and de Weck, O., 2004, "A Comparison of Particle Swarm Optimization and The Genetic Algorithm", American Institute of Aeronautics and Astronautics

[8] Shayegi, H., Shayanfar, H. A. and Shojaei, A., 2008, "An improved PSO based solution for the optimal power flow problems", http://www.sciencedirect.com

[9] Vimal Raj, P., Senthikumar, S., Ravichandran, S., and Palanivelu, T. G., 2008, "Optimization of distributed generation capacity for line loss reduction and voltage profile improvement using PSO", http://fke.utm.my/elektrika

APPENDICES

APPENDIX A

IEEE 14-bus System Data

Table A.1: Bus data

Bus No.	P Generated (p.u.)	Q Generated (p.u.)	P Load (p.u.)	Q Load (p.u.)	Bus Type*	Q Generated max.(p.u.)	Q Generated min.(p.u.)
1	2.32	0.00	0.00	0.00	2	10.0	-10.0
2	0.4	-0.424	0.2170	0.1270	1	0.5	-0.4
3	0.00	0.00	0.9420	0.1900	2	0.4	0.00
4	0.00	0.00	0.4780	0.00	3	0.00	0.00
5	0.00	0.00	0.0760	0.0160	3	0.00	0.00
6	0.00	0.00	0.1120	0.0750	2	0.24	-0.06
7	0.00	0.00	0.00	0.00	3	0.00	0.00
8	0.00	0.00	0.00	0.00	2	0.24	-0.06
9	0.00	0.00	0.2950	0.1660	3	0.00	0.00
10	0.00	0.00	0.0900	0.0580	3	0.00	0.00
11	0.00	0.00	0.0350	0.0180	3	0.00	0.00
12	0.00	0.00	0.0610	0.0160	3	0.00	0.00
13	0.00	0.00	0.1350	0.0580	3	0.00	0.00
14	0.00	0.00	0.1490	0.0500	3	0.00	0.00

*Bus Type: (1) swing bus, (2) generator bus (PV bus), and (3) load bus (PQ bus)

APPENDIX A

IEEE 14-bus System Data

Table A.2: Line data

From Bus	To Bus	Resistance (p.u.)	Reactance (p.u)	Line charging (p.u.)	tap ratio
1	2	0.01938	0.05917	0.0528	1
1	5	0.05403	0.22304	0.0492	1
2	3	0.04699	0.19797	0.0438	1
2	4	0.05811	0.17632	0.0374	1
2	5	0.05695	0.17388	0.034	1
3	4	0.06701	0.17103	0.0346	1
4	5	0.01335	0.04211	0.0128	1
4	7	0.00	0.20912	0.00	0.978
4	9	0.00	0.55618	0.00	0.969
5	6	0.00	0.25202	0.00	0.932
6	11	0.09498	0.1989	0.00	1
6	12	0.12291	0.25581	0.00	1
6	13	0.06615	0.13027	0.00	1
7	8	0.00	0.17615	0.00	1
7	9	0.00	0.11001	0.00	1
9	10	0.03181	0.08450	0.00	1
9	14	0.12711	0.27038	0.00	1
10	11	0.08205	0.19207	0.00	1
12	13	0.22092	0.19988	0.00	1
13	14	0.17093	0.34802	0.00	1

APPENDIX A

IEEE 14-bus System Data

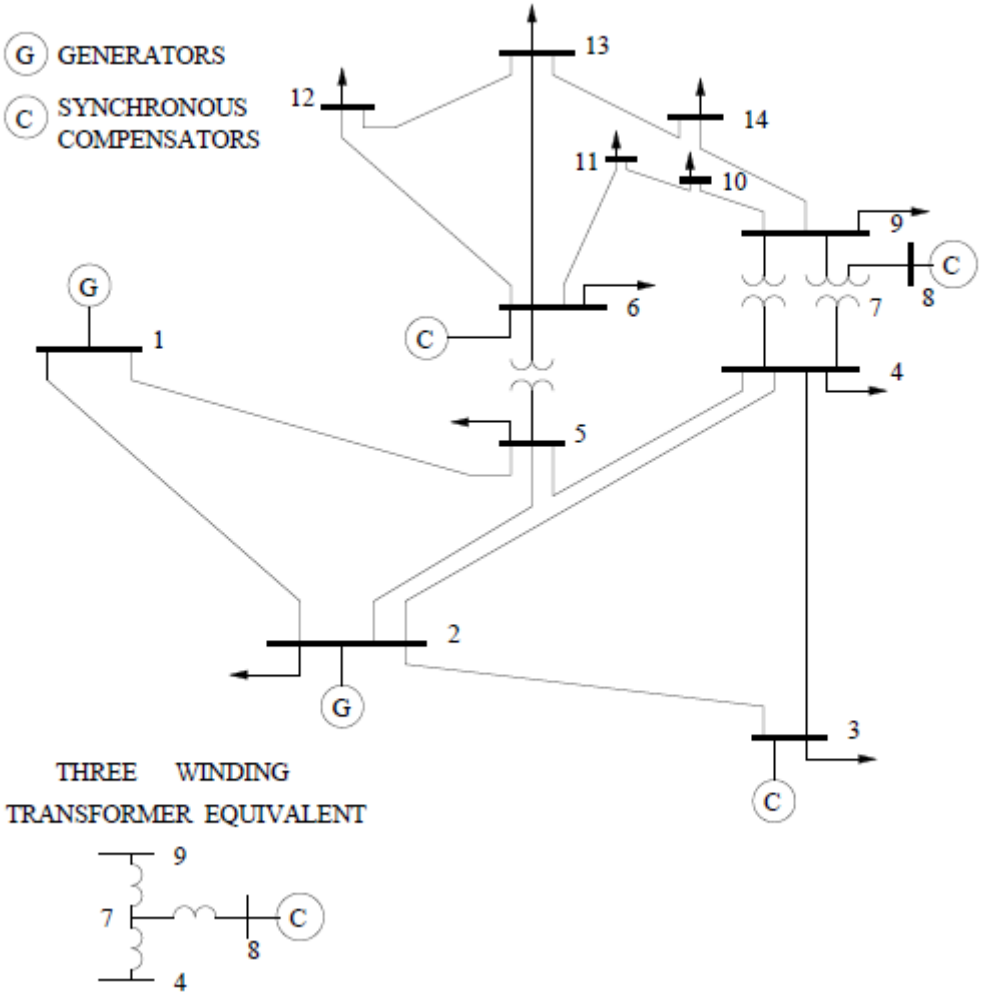


Figure A.1: IEEE 14-bus system diagram

APPENDIX B

Newton-Raphson Method for Power Flow Model Matlab Code

```
ns=0; ng=0; Vm=0; delta=0; yload=0; deltad=0;
nbus = length(busdata(:,1));
kb=[];Vm=[]; delta=[]; Pd=[]; Qd=[]; Pg=[]; Qg=[]; Qmin=[]; Qmax=[];
Pk=[]; P=[]; Qk=[]; Q=[]; S=[]; V=[];
for k=1:nbus
n=busdata(k,1);
kb(n)=busdata(k,2); Vm(n)=busdata(k,3); delta(n)=busdata(k, 4);
Pd(n)=busdata(k,5); Qd(n)=busdata(k,6); Pg(n)=busdata(k,7); Qg(n) = busdata(k,8);
Qmin(n)=busdata(k, 9); Qmax(n)=busdata(k, 10);
Qsh(n)=busdata(k, 11);
    if Vm(n) <= 0  Vm(n) = 1.0; V(n) = 1 + j*0;
    else delta(n) = pi/180*delta(n);
         V(n) = Vm(n)*(cos(delta(n)) + j*sin(delta(n)));
         P(n)=(Pg(n)-Pd(n))/basemva;
         Q(n)=(Qg(n)-Qd(n)+ Qsh(n))/basemva;
         S(n) = P(n) + j*Q(n);
    end
end
for k=1:nbus
if kb(k) == 1, ns = ns+1; else, end
if kb(k) == 2 ng = ng+1; else, end
ngs(k) = ng;
nss(k) = ns;
end
Ym=abs(Ybus); t = angle(Ybus);
m=2*nbus-ng-2*ns;
maxerror = 1; converge=1;
iter = 0;
mline=ones(nbr,1);
for k=1:nbr
      for m=k+1:nbr
         if((nl(k)==nl(m)) & (nr(k)==nr(m)));
            mline(m)=2;
         elseif ((nl(k)==nr(m)) & (nr(k)==nl(m)));
         mline(m)=2;
         else, end
      end
   end

% Start of iterations
clear A  DC   J  DX
while maxerror >= accuracy & iter <= maxiter % Test for max. power mismatch
for ii=1:m
for k=1:m
   A(ii,k)=0;      %Initializing Jacobian matrix
end, end
iter = iter+1;
for n=1:nbus
nn=n-nss(n);
lm=nbus+n-ngs(n)-nss(n)-ns;
J11=0; J22=0; J33=0; J44=0;
   for ii=1:nbr
    if mline(ii)==1
        if nl(ii) == n | nr(ii) == n
            if nl(ii) == n ,  l = nr(ii); end
            if nr(ii) == n , l = nl(ii); end
         J11=J11+ Vm(n)*Vm(l)*Ym(n,l)*sin(t(n,l)- delta(n) + delta(l));
         J33=J33+ Vm(n)*Vm(l)*Ym(n,l)*cos(t(n,l)- delta(n) + delta(l));
                if kb(n)~=1
                J22=J22+ Vm(l)*Ym(n,l)*cos(t(n,l)- delta(n) + delta(l));
                J44=J44+ Vm(l)*Ym(n,l)*sin(t(n,l)- delta(n) + delta(l));
```

```
                else, end
                if kb(n) ~= 1  & kb(l) ~=1
                    lk = nbus+l-ngs(l)-nss(l)-ns;
                    ll = l -nss(l);
                % off diagonalelements of J1
                    A(nn, ll) =-Vm(n)*Vm(l)*Ym(n,l)*sin(t(n,l)- delta(n) + delta(l));
                    if kb(l) == 0  % off diagonal elements of J2
                    A(nn, lk) =Vm(n)*Ym(n,l)*cos(t(n,l)- delta(n) + delta(l));end
                    if kb(n) == 0  % off diagonal elements of J3
                    A(lm, ll) =-Vm(n)*Vm(l)*Ym(n,l)*cos(t(n,l)- delta(n)+delta(l));
end
                    if kb(n) == 0 & kb(l) == 0  % off diagonal elements of  J4
                    A(lm, lk) =-Vm(n)*Ym(n,l)*sin(t(n,l)- delta(n) + delta(l));end
               else end
            else , end
      else, end
   end
   Pk = Vm(n)^2*Ym(n,n)*cos(t(n,n))+J33;
   Qk = -Vm(n)^2*Ym(n,n)*sin(t(n,n))-J11;
   if kb(n) == 1 P(n)=Pk; Q(n) = Qk; end   % Swing bus P
     if kb(n) == 2  Q(n)=Qk;
         if Qmax(n) ~= 0
           Qgc = Q(n)*basemva + Qd(n) - Qsh(n);
           if iter <= 7                  % Between the 2th & 6th iterations
              if iter > 2                % the Mvar of generator buses are
                if Qgc  < Qmin(n),       % tested. If not within limits Vm(n)
                Vm(n) = Vm(n) + 0.01;    % is changed in steps of 0.01 pu to
                elseif Qgc  > Qmax(n),   % bring the generator Mvar within
                Vm(n) = Vm(n) - 0.01;end % the specified limits.
              else, end
           else,end
         else,end
     end
   if kb(n) ~= 1
     A(nn,nn) = J11;  %diagonal elements of J1
     DC(nn) = P(n)-Pk;
   end
   if kb(n) == 0
     A(nn,lm) = 2*Vm(n)*Ym(n,n)*cos(t(n,n))+J22;  %diagonal elements of J2
     A(lm,nn)= J33;        %diagonal elements of J3
     A(lm,lm) =-2*Vm(n)*Ym(n,n)*sin(t(n,n))-J44;  %diagonal of elements of J4
     DC(lm) = Q(n)-Qk;
   end
end
DX=A\DC';
for n=1:nbus
  nn=n-nss(n);
  lm=nbus+n-ngs(n)-nss(n)-ns;
    if kb(n) ~= 1
    delta(n) = delta(n)+DX(nn); end
    if kb(n) == 0
    Vm(n)=Vm(n)+DX(lm); end
 end
  maxerror=max(abs(DC));
     if iter == maxiter & maxerror > accuracy
   fprintf('\nWARNING: Iterative solution did not converged after ')
   fprintf('%g', iter), fprintf(' iterations.\n\n')
   fprintf('Press Enter to terminate the iterations and print the results \n')
   converge = 0; pause, else, end

end

if converge ~= 1
   tech= ('                      ITERATIVE SOLUTION DID NOT CONVERGE'); else,
   tech=('                   Power Flow Solution by Newton-Raphson Method');
end
V = Vm.*cos(delta)+j*Vm.*sin(delta);
deltad=180/pi*delta;
i=sqrt(-1);
k=0;
for n = 1:nbus
     if kb(n) == 1
     k=k+1;
     S(n)= P(n)+j*Q(n);
```

```
     Pg(n) = P(n)*basemva + Pd(n);
     Qg(n) = Q(n)*basemva + Qd(n) - Qsh(n);
     Pgg(k)=Pg(n);
     Qgg(k)=Qg(n);
     elseif  kb(n) ==2
     k=k+1;
     S(n)=P(n)+j*Q(n);
     Qg(n) = Q(n)*basemva + Qd(n) - Qsh(n);
     Pgg(k)=Pg(n);
     Qgg(k)=Qg(n);
  end
yload(n) = (Pd(n)- j*Qd(n)+j*Qsh(n))/(basemva*Vm(n)^2);
end
busdata(:,3)=Vm'; busdata(:,4)=deltad';
Pgt = sum(Pg);  Qgt = sum(Qg); Pdt = sum(Pd); Qdt = sum(Qd); Qsht = sum(Qsh);

%clear A DC DX  J11 J22 J33 J44 Qk delta lk ll lm
%clear A DC DX  J11 J22 J33  Qk delta lk ll lm
```

APPENDIX C

PSO Algorithm Matlab Code

```
% pso_Trelea_vectorized.m
% a generic particle swarm optimizer
% to find the minimum or maximum of any
% MISO matlab function
%
% Implements Common, Trelea type 1 and 2, and Clerc's class 1". It will
% also automatically try to track to a changing environment (with varied
% success - BKB 3/18/05)
%
% This vectorized version removes the for loop associated with particle
% number. It also *requires* that the cost function have a single input
% that represents all dimensions of search (i.e., for a function that has 2
% inputs then make a wrapper that passes a matrix of ps x 2 as a single
% variable)
%
% Usage:
%  [optOUT]=PSO(functname,D)
% or:
%  [optOUT,tr,te]=...
%        PSO(functname,D,mv,VarRange,minmax,PSOparams,plotfcn,PSOseedValue)
%
% Inputs:
%    functname - string of matlab function to optimize
%    D - # of inputs to the function (dimension of problem)
%
% Optional Inputs:
%    mv - max particle velocity, either a scalar or a vector of length D
%           (this allows each component to have it's own max velocity),
%           default = 4, set if not input or input as NaN
%
%    VarRange - matrix of ranges for each input variable,
%      default -100 to 100, of form:
%       [ min1 max1
%         min2 max2
%            ...
%         minD maxD ]
%
%    minmax = 0, funct minimized (default)
%           = 1, funct maximized
%           = 2, funct is targeted to P(12) (minimizes distance to errgoal)
%    PSOparams - PSO parameters
%      P(1) - Epochs between updating display, default = 100. if 0,
%             no display
%      P(2) - Maximum number of iterations (epochs) to train, default = 2000.
%      P(3) - population size, default = 24
%
%      P(4) - acceleration const 1 (local best influence), default = 2
%      P(5) - acceleration const 2 (global best influence), default = 2
%      P(6) - Initial inertia weight, default = 0.9
%      P(7) - Final inertia weight, default = 0.4
%      P(8) - Epoch when inertial weight at final value, default = 1500
%      P(9)- minimum global error gradient,
%                  if abs(Gbest(i+1)-Gbest(i)) < gradient over
%                  certain length of epochs, terminate run, default = 1e-25
%      P(10)- epochs before error gradient criterion terminates run,
%                  default = 150, if the SSE does not change over 250 epochs
%                               then exit
%      P(11)- error goal, if NaN then unconstrained min or max, default=NaN
%      P(12)- type flag (which kind of PSO to use)
%                  0 = Common PSO w/intertia (default)
%                  1,2 = Trelea types 1,2
%                  3   = Clerc's Constricted PSO, Type 1"
```

```
%      P(13)- PSOseed, default=0
%               = 0 for initial positions all random
%               = 1 for initial particles as user input
%
%    plotfcn - optional name of plotting function, default 'goplotpso',
%              make your own and put here
%
%    PSOseedValue - initial particle position, depends on P(13), must be
%                   set if P(13) is 1 or 2, not used for P(13)=0, needs to
%                   be nXm where n<=ps, and m<=D
%                   If n<ps and/or m<D then remaining values are set random
%                   on Varrange
% Outputs:
%    optOUT - optimal inputs and associated min/max output of function, of form:
%        [ bestin1
%          bestin2
%            ...
%          bestinD
%          bestOUT ]
%
% Optional Outputs:
%    tr    - Gbest at every iteration, traces flight of swarm
%    te    - epochs to train, returned as a vector 1:endepoch
%
% Example:  out=pso_Trelea_vectorized('f6',2)

% Brian Birge
% Rev 3.3
% 2/18/06

function [OUT,varargout]=pso_Trelea_vectorized(functname,D,varargin)

rand('state',sum(100*clock));
if nargin < 2
   error('Not enough arguments.');
end

% PSO PARAMETERS
if nargin == 2      % only specified functname and D
   VRmin=ones(D,1)*-100;
   VRmax=ones(D,1)*100;
   VR=[VRmin,VRmax];
   minmax = 0;
   P = [];
   mv = 4;
   plotfcn='goplotpso';
elseif nargin == 3  % specified functname, D, and mv
   VRmin=ones(D,1)*-100;
   VRmax=ones(D,1)*100;
   VR=[VRmin,VRmax];
   minmax = 0;
   mv=varargin{1};
   if isnan(mv)
       mv=4;
   end
   P = [];
   plotfcn='goplotpso';
elseif nargin == 4  % specified functname, D, mv, Varrange
   mv=varargin{1};
   if isnan(mv)
       mv=4;
   end
   VR=varargin{2};
   minmax = 0;
   P = [];
   plotfcn='goplotpso';
elseif nargin == 5  % Functname, D, mv, Varrange, and minmax
   mv=varargin{1};
   if isnan(mv)
       mv=4;
   end
   VR=varargin{2};
   minmax=varargin{3};
   P = [];
```

```
   plotfcn='goplotpso';
elseif nargin == 6  % Functname, D, mv, Varrange, minmax, and psoparams
   mv=varargin{1};
   if isnan(mv)
       mv=4;
   end
   VR=varargin{2};
   minmax=varargin{3};
   P = varargin{4}; % psoparams
   plotfcn='goplotpso';
elseif nargin == 7  % Functname, D, mv, Varrange, minmax, and psoparams, plotfcn
   mv=varargin{1};
   if isnan(mv)
       mv=4;
   end
   VR=varargin{2};
   minmax=varargin{3};
   P = varargin{4}; % psoparams
   plotfcn = varargin{5};
elseif nargin == 8  % Functname, D, mv, Varrange, minmax, and psoparams, plotfcn,
PSOseedValue
   mv=varargin{1};
   if isnan(mv)
       mv=4;
   end
   VR=varargin{2};
   minmax=varargin{3};
   P = varargin{4}; % psoparams
   plotfcn = varargin{5};
   PSOseedValue = varargin{6};
else
   error('Wrong # of input arguments.');
end

% sets up default pso params
Pdef = [100 2000 24 2 2 0.9 0.4 1500 1e-25 250 NaN 0 0];
Plen = length(P);
P    = [P,Pdef(Plen+1:end)];

df      = P(1);
me      = P(2);
ps      = P(3);
ac1     = P(4);
ac2     = P(5);
iw1     = P(6);
iw2     = P(7);
iwe     = P(8);
ergrd   = P(9);
ergrdep = P(10);
errgoal = P(11);
trelea  = P(12);
PSOseed = P(13);

% used with trainpso, for neural net training
if strcmp(functname,'pso_neteval')
   net = evalin('caller','net');
    Pd = evalin('caller','Pd');
    Tl = evalin('caller','Tl');
    Ai = evalin('caller','Ai');
     Q = evalin('caller','Q');
    TS = evalin('caller','TS');
end


% error checking
 if ((minmax==2) & isnan(errgoal))
     error('minmax= 2, errgoal= NaN: choose an error goal or set minmax to 0 or 1');
 end

 if ( (PSOseed==1) & ~exist('PSOseedValue') )
     error('PSOseed flag set but no PSOseedValue was input');
 end

 if exist('PSOseedValue')
```

```
     tmpsz=size(PSOseedValue);
     if D < tmpsz(2)
         error('PSOseedValue column size must be D or less');
     end
     if ps < tmpsz(1)
         error('PSOseedValue row length must be # of particles or less');
     end
 end

% set plotting flag
if (P(1))~=0
  plotflg=1;
else
  plotflg=0;
end

% preallocate variables for speed up
 tr = ones(1,me)*NaN;

% take care of setting max velocity and position params here
if length(mv)==1
 velmaskmin = -mv*ones(ps,D);     % min vel, psXD matrix
 velmaskmax = mv*ones(ps,D);      % max vel
elseif length(mv)==D
 velmaskmin = repmat(forcerow(-mv),ps,1); % min vel
 velmaskmax = repmat(forcerow( mv),ps,1); % max vel
else
 error('Max vel must be either a scalar or same length as prob dimension D');
end
posmaskmin  = repmat(VR(1:D,1)',ps,1);  % min pos, psXD matrix
posmaskmax  = repmat(VR(1:D,2)',ps,1);  % max pos
posmaskmeth = 3; % 3=bounce method (see comments below inside epoch loop)

% PLOTTING
 message = sprintf('PSO: %%g/%g iterations, GBest = %%20.20g.\n',me);

% INITIALIZE INITIALIZE INITIALIZE INITIALIZE INITIALIZE INITIALIZE

% initialize population of particles and their velocities at time zero,
% format of pos= (particle#, dimension)
 % construct random population positions bounded by VR
  pos(1:ps,1:D) = normmat(rand([ps,D]),VR',1);

  if PSOseed == 1         % initial positions user input, see comments above
    tmpsz                     = size(PSOseedValue);
    pos(1:tmpsz(1),1:tmpsz(2)) = PSOseedValue;
  end

 % construct initial random velocities between -mv,mv
  vel(1:ps,1:D) = normmat(rand([ps,D]),...
      [forcecol(-mv),forcecol(mv)]',1);

% initial pbest positions vals
 pbest = pos;

% VECTORIZE THIS, or at least vectorize cost funct call
 out = feval(functname,pos);  % returns column of cost values (1 for each particle)
%---------------------------

 pbestval=out;   % initially, pbest is same as pos

% assign initial gbest here also (gbest and gbestval)
 if minmax==1
   % this picks gbestval when we want to maximize the function
    [gbestval,idx1] = max(pbestval);
 elseif minmax==0
   % this works for straight minimization
    [gbestval,idx1] = min(pbestval);
 elseif minmax==2
   % this works when you know target but not direction you need to go
   % good for a cost function that returns distance to target that can be either
   % negative or positive (direction info)
    [temp,idx1] = min((pbestval-ones(size(pbestval))*errgoal).^2);
    gbestval    = pbestval(idx1);
```

```
 end

 % preallocate a variable to keep track of gbest for all iters
 bestpos        = zeros(me,D+1)*NaN;
 gbest          = pbest(idx1,:);  % this is gbest position
   % used with trainpso, for neural net training
   % assign gbest to net at each iteration, these interim assignments
   % are for plotting mostly
    if strcmp(functname,'pso_neteval')
        net=setx(net,gbest);
    end
 %tr(1)          = gbestval;       % save for output
 bestpos(1,1:D) = gbest;

% this part used for implementing Carlisle and Dozier's APSO idea
% slightly modified, this tracks the global best as the sentry whereas
% their's chooses a different point to act as sentry
% see "Tracking Changing Extremea with Adaptive Particle Swarm Optimizer",
% part of the WAC 2002 Proceedings, June 9-13, http://wacong.com
 sentryval = gbestval;
 sentry    = gbest;

if (trelea == 3)
% calculate Clerc's constriction coefficient chi to use in his form
 kappa   = 1; % standard val = 1, change for more or less constriction
 if ( (ac1+ac2) <=4 )
     chi = kappa;
 else
     psi     = ac1 + ac2;
     chi_den = abs(2-psi-sqrt(psi^2 - 4*psi));
     chi_num = 2*kappa;
     chi     = chi_num/chi_den;
 end
end

% INITIALIZE END INITIALIZE END INITIALIZE END INITIALIZE END
rstflg = 0; % for dynamic environment checking
% start PSO iterative procedures
 cnt    = 0; % counter used for updating display according to df in the options
 cnt2   = 0; % counter used for the stopping subroutine based on error convergence
 iwt(1) = iw1;
for i=1:me  % start epoch loop (iterations)

     out        = feval(functname,[pos;gbest]);
     outbestval = out(end,:);
     out        = out(1:end-1,:);

     tr(i+1)          = gbestval; % keep track of global best val
     te               = i; % returns epoch number to calling program when done
     bestpos(i,1:D+1) = [gbest,gbestval];

     %assignin('base','bestpos',bestpos(i,1:D+1));
   %-------------------------------------------------------------------------
   % this section does the plots during iterations
    if plotflg==1
      if (rem(i,df) == 0 ) | (i==me) | (i==1)
         fprintf(message,i,gbestval);
         cnt = cnt+1; % count how many times we display (useful for movies)

         eval(plotfcn); % defined at top of script

      end  % end update display every df if statement
    end % end plotflg if statement

    % check for an error space that changes wrt time/iter
    % threshold value that determines dynamic environment
    % sees if the value of gbest changes more than some threshold value
    % for the same location
    chkdyn = 1;
    rstflg = 0; % for dynamic environment checking

    if chkdyn==1
     threshld = 0.05;  % percent current best is allowed to change, .05 = 5% etc
```

```
     letiter  = 5; % # of iterations before checking environment, leave at least 3 so
PSO has time to converge
     outorng  = abs( 1- (outbestval/gbestval) ) >= threshld;
     samepos  = (max( sentry == gbest ));

     if (outorng & samepos) & rem(i,letiter)==0
         rstflg=1;
       % disp('New Environment: reset pbest, gbest, and vel');
       %% reset pbest and pbestval if warranted
%        outpbestval = feval( functname,[pbest] );
%        Poutorng    = abs( 1-(outpbestval./pbestval) ) > threshld;
%        pbestval    = pbestval.*~Poutorng + outpbestval.*Poutorng;
%        pbest       = pbest.*repmat(~Poutorng,1,D) + pos.*repmat(Poutorng,1,D);

        pbest     = pos; % reset personal bests to current positions
        pbestval  = out;
        vel       = vel*10; % agitate particles a little (or a lot)

       % recalculate best vals
        if minmax == 1
           [gbestval,idx1] = max(pbestval);
        elseif minmax==0
           [gbestval,idx1] = min(pbestval);
        elseif minmax==2 % this section needs work
           [temp,idx1] = min((pbestval-ones(size(pbestval))*errgoal).^2);
           gbestval    = pbestval(idx1);
        end

        gbest  = pbest(idx1,:);

        % used with trainpso, for neural net training
        % assign gbest to net at each iteration, these interim assignments
        % are for plotting mostly
        if strcmp(functname,'pso_neteval')
           net=setx(net,gbest);
        end
     end  % end if outorng

     sentryval = gbestval;
     sentry    = gbest;

    end % end if chkdyn

    % find particles where we have new pbest, depending on minmax choice
    % then find gbest and gbestval
     %[size(out),size(pbestval)]
    if rstflg == 0
     if minmax == 0
        [tempi]            = find(pbestval>=out); % new min pbestvals
        pbestval(tempi,1)  = out(tempi);   % update pbestvals
        pbest(tempi,:)     = pos(tempi,:); % update pbest positions

        [iterbestval,idx1] = min(pbestval);

        if gbestval >= iterbestval
            gbestval = iterbestval;
            gbest    = pbest(idx1,:);
            % used with trainpso, for neural net training
            % assign gbest to net at each iteration, these interim assignments
            % are for plotting mostly
             if strcmp(functname,'pso_neteval')
                net=setx(net,gbest);
             end
        end
     elseif minmax == 1
        [tempi,dum]        = find(pbestval<=out); % new max pbestvals
        pbestval(tempi,1)  = out(tempi,1); % update pbestvals
        pbest(tempi,:)     = pos(tempi,:); % update pbest positions

        [iterbestval,idx1] = max(pbestval);
        if gbestval <= iterbestval
            gbestval = iterbestval;
            gbest    = pbest(idx1,:);
            % used with trainpso, for neural net training
```

```
            % assign gbest to net at each iteration, these interim assignments
            % are for plotting mostly
             if strcmp(functname,'pso_neteval')
                net=setx(net,gbest);
             end
        end
     elseif minmax == 2  % this won't work as it is, fix it later
        egones            = errgoal*ones(ps,1); % vector of errgoals
        sqrerr2           = ((pbestval-egones).^2);
        sqrerr1           = ((out-egones).^2);
        [tempi,dum]       = find(sqerr1 <= sqrerr2); % find particles closest to targ
        pbestval(tempi,1) = out(tempi,1); % update pbestvals
        pbest(tempi,:)    = pos(tempi,:); % update pbest positions

        sqrerr            = ((pbestval-egones).^2); % need to do this to reflect new
pbests
        [temp,idx1]       = min(sqrerr);
        iterbestval       = pbestval(idx1);

        if (iterbestval-errgoal)^2 <= (gbestval-errgoal)^2
           gbestval = iterbestval;
           gbest    = pbest(idx1,:);
           % used with trainpso, for neural net training
            % assign gbest to net at each iteration, these interim assignments
            % are for plotting mostly
             if strcmp(functname,'pso_neteval')
                net=setx(net,gbest);
             end
        end
     end
    end


 %   % build a simple predictor 10th order, for gbest trajectory
 %   if i>500
 %    for dimcnt=1:D
 %      pred_coef  = polyfit(i-250:i,(bestpos(i-250:i,dimcnt))',20);
 %     % pred_coef  = polyfit(200:i,(bestpos(200:i,dimcnt))',20);
 %      gbest_pred(i,dimcnt) = polyval(pred_coef,i+1);
 %    end
 %    else
%       gbest_pred(i,:) = zeros(size(gbest));
%    end

   %gbest_pred(i,:)=gbest;
   %assignin('base','gbest_pred',gbest_pred);

 %   % convert to non-inertial frame
 %    gbestoffset = gbest - gbest_pred(i,:);
 %    gbest = gbest - gbestoffset;
 %    pos   = pos + repmat(gbestoffset,ps,1);
 %    pbest = pbest + repmat(gbestoffset,ps,1);

     %PSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSO

      % get new velocities, positions (this is the heart of the PSO algorithm)
      % each epoch get new set of random numbers
       rannum1 = rand([ps,D]); % for Trelea and Clerc types
       rannum2 = rand([ps,D]);
       if     trelea == 2
        % from Trelea's paper, parameter set 2
         vel = 0.729.*vel...                                % prev vel
               +1.494.*rannum1.*(pbest-pos)...              % independent
               +1.494.*rannum2.*(repmat(gbest,ps,1)-pos); % social
       elseif trelea == 1
        % from Trelea's paper, parameter set 1
         vel = 0.600.*vel...                                % prev vel
               +1.700.*rannum1.*(pbest-pos)...              % independent
               +1.700.*rannum2.*(repmat(gbest,ps,1)-pos); % social
       elseif trelea ==3
        % Clerc's Type 1" PSO
         vel = chi*(vel...                                  % prev vel
               +ac1.*rannum1.*(pbest-pos)...                % independent
               +ac2.*rannum2.*(repmat(gbest,ps,1)-pos)) ; % social
```

```
        else
         % common PSO algo with inertia wt
         % get inertia weight, just a linear funct w.r.t. epoch parameter iwe
          if i<=iwe
             iwt(i) = ((iw2-iw1)/(iwe-1))*(i-1)+iw1;
          else
             iwt(i) = iw2;
          end
         % random number including acceleration constants
          ac11 = rannum1.*ac1;    % for common PSO w/inertia
          ac22 = rannum2.*ac2;

          vel = iwt(i).*vel...                              % prev vel
                +ac11.*(pbest-pos)...                       % independent
                +ac22.*(repmat(gbest,ps,1)-pos);            % social
        end

        % limit velocities here using masking
         vel = ( (vel <= velmaskmin).*velmaskmin ) + ( (vel > velmaskmin).*vel );
         vel = ( (vel >= velmaskmax).*velmaskmax ) + ( (vel < velmaskmax).*vel );

        % update new position (PSO algo)
         pos = pos + vel;

        % position masking, limits positions to desired search space
        % method: 0) no position limiting, 1) saturation at limit,
        %         2) wraparound at limit , 3) bounce off limit
         minposmask_throwaway = pos <= posmaskmin;  % these are psXD matrices
         minposmask_keep      = pos >  posmaskmin;
         maxposmask_throwaway = pos >= posmaskmax;
         maxposmask_keep      = pos <  posmaskmax;

         if     posmaskmeth == 1
          % this is the saturation method
           pos = ( minposmask_throwaway.*posmaskmin ) + ( minposmask_keep.*pos );
           pos = ( maxposmask_throwaway.*posmaskmax ) + ( maxposmask_keep.*pos );
         elseif posmaskmeth == 2
          % this is the wraparound method
           pos = ( minposmask_throwaway.*posmaskmax ) + ( minposmask_keep.*pos );
           pos = ( maxposmask_throwaway.*posmaskmin ) + ( maxposmask_keep.*pos );
         elseif posmaskmeth == 3
          % this is the bounce method, particles bounce off the boundaries with -vel
           pos = ( minposmask_throwaway.*posmaskmin ) + ( minposmask_keep.*pos );
           pos = ( maxposmask_throwaway.*posmaskmax ) + ( maxposmask_keep.*pos );

           vel = (vel.*minposmask_keep) + (-vel.*minposmask_throwaway);
           vel = (vel.*maxposmask_keep) + (-vel.*maxposmask_throwaway);
         else
          % no change, this is the original Eberhart, Kennedy method,
          % it lets the particles grow beyond bounds if psoparams (P)
          % especially Vmax, aren't set correctly, see the literature
         end

      %PSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSO
% check for stopping criterion based on speed of convergence to desired
    % error
     tmp1 = abs(tr(i) - gbestval);
     if tmp1 > ergrd
        cnt2 = 0;
     elseif tmp1 <= ergrd
        cnt2 = cnt2+1;
        if cnt2 >= ergrdep
          if plotflg == 1
           fprintf(message,i,gbestval);
           disp(' ');
           disp(['--> Solution likely, GBest hasn''t changed by at least ',...
               num2str(ergrd),' for ',...
                   num2str(cnt2),' epochs.']);
           eval(plotfcn);
          end
          break
        end
     end
```

```
   % this stops if using constrained optimization and goal is reached
    if ~isnan(errgoal)
     if ((gbestval<=errgoal) & (minmax==0)) | ((gbestval>=errgoal) & (minmax==1))

         if plotflg == 1
             fprintf(message,i,gbestval);
             disp(' ');
             disp(['--> Error Goal reached, successful termination!']);

             eval(plotfcn);
         end
         break
     end

    % this is stopping criterion for constrained from both sides
     if minmax == 2
       if ((tr(i)<errgoal) & (gbestval>=errgoal)) | ((tr(i)>errgoal) ...
               & (gbestval <= errgoal))
         if plotflg == 1
             fprintf(message,i,gbestval);
             disp(' ');
             disp(['--> Error Goal reached, successful termination!']);

             eval(plotfcn);
         end
         break
       end
     end % end if minmax==2
    end  % end ~isnan if

 %    % convert back to inertial frame
 %     pos = pos - repmat(gbestoffset,ps,1);
 %     pbest = pbest - repmat(gbestoffset,ps,1);
 %     gbest = gbest + gbestoffset;


end  % end epoch loop

%% clear temp outputs
% evalin('base','clear temp_pso_out temp_te temp_tr;');

% output & return
 OUT=[gbest';gbestval];
 varargout{1}=[1:te];
 varargout{2}=[tr(find(~isnan(tr)))];

 return
```