AI SOLUTION TO ECONOMIC DISPATCH

NUR FARHANAH BT WAKIMAN

This thesis is submitted as partial fulfillment of the requirements for the award of the Bachelor of Electrical Engineering (Hons.) (Power System)

Faculty of Electrical & Electronics Engineering

Universiti Malaysia Pahang

NOVEMBER,2009

DECRALATION

"All the trademark and copyrights use herein are property of their respective owner. References of information from other sources are quoted accordingly; otherwise the information presented in this report is solely work of the author."

Signature	: ______________________________
Author	: NUR FARHANAH BT WAKIMAN
Date	: 23 NOVEMBER 2009

DEDICATION

Specially dedicated to

My beloved family,friends and those who have guided and inspired me

Thank you for the endless support and encouragement

ACKNOWLEDGMENT

Praise to Allah S.W.T for giving me time and ability to complete this BEE4724 (Engineering Project II) I wish to express my deepest and sincere gratitude to my project supervisor DR. AHMED N. ABD ALLA, l for his endless support, guidance, constructive and keen interest in supervising this project.

Special thanks to University Malaysia Pahang for supporting and providing equipment and information sources that assisted my studies and projects.

My heartiest thanks to my beloved family especially both of my parents Wakiman b. Sario and Rahana bt Zakaria who always pray for my success continuously, giving me all the guidance, support and love that I needs all the time.

To all my lovely friends who always willingly assist and support me throughout my journey of education, you all deserve my wholehearted appreciation. Many thanks

Thank you.

ABSTRACT

In this study, proposes an AI optimization method for solving the economic dispatch (ED) problem in power systems. Many nonlinear characteristics of the generator, such as ramp rate limits, prohibited operating zone, and non-smooth cost functions are considered using the proposed method in practical generator operation. The feasibility of the proposed method is demonstrated for two different systems, and it is compared with the Conventional method in terms of the solution quality and computation efficiency. The experimental results show that the proposed PSO method was indeed capable of obtaining higher quality solutions efficiently in ED problems.

ABSTRAK

Dalam kajian ini, mencadangkan satu kaedah pengoptimuman AI untuk penyelesaian masalah dalam penghantaran sistem-sistem tenaga ekonomi secara ekonomik. Banyak ciri penjana tak linear, seperti kadar tanjakan mengehadkan, zon beroperasi terlarang, dan fungsi kos tidak licin dianggap menggunakan cadangkan kaedah dalam operasi penjana praktikal. Kemungkinan mencadangkan kaedah adalah didemonstrasikan untuk dua sistem berbeza, dan ia dibandingkan dengan kaedah konvensional dalam syarat-syarat bagi kualiti penyelesaian dan kecekapan pengiraan. Pertunjukan hasil percubaan yang mencadangkan kaedah PSO memang mampu mendapatkan penyelesaian-penyelesaian berkualiti lebih tinggi dengan cekap dalam masalah-masalah penghantaran sistem-sistem tenaga ekonomi secara ekonomik.

CHAPTER

TABLE OF CONTENT

LIST OF TABLE

LIST OF FIGURE

LIST OF ABBREVIATION

PSO	Particle Swarm Optimization
GUI	Graphical User Interface
AI	Artificial Intelligent
ED	Economic Dispatch
UMP	Universiti Malaysia Pahang

"I hereby acknowledge that the scope and quality of this thesis is qualified for the award of the Bachelor Degree of Electrical Engineering (Electronics)"

Signature	: ______________________________
Name	: DR AHMED N. ABD ALLA
Date	: 23 NOVEMBER 2009

CHAPTER 1

INTRODUCTION

1.1 Background

In realities, power plants are not at same distance with the center of load, and the fuel costs are different for each distance. Under normal operating condition the generating capacities are more than total load demand and losses. Thus, there is one main option for scheduling generation is called optimal dispatch. The optimal dispatch of power generation are used to defined an effective real and reactive power scheduling to power plant and meet the load demand at the same time minimize the operating cost. The function cost may present economic cost, security system. In this study, the analysis will limited to the economic dispatch of real power generation. Economic dispatch analysis has been studied by many researchers using different method.[1] Previous efforts on solving economic dispatch problems have employed various mathematical programming methods and optimization techniques. The solution of the power system depends largely on the type of the algorithm used for optimization. The power system optimization problem need algorithm with faster rate of convergence, very high accuracy and capacity to handle very large complexity.[2] in order to make numerical methods more convenient for solving economic dispatch problem, artificial intelligent technique such as the Hopfield Neural networks, Genetic

Algorithms(GA) or Simulated Annealing(SA) and Particle Swarm Optimization (PSO) has been successfully used to solve power optimization problem.[2]

1.2 Objective

The objective of this project is to;

i. Find the optimum value that simultaneously minimize the generation cost rate
ii. To obtain simulation on economic dispatch of power generation using MATLAB
iii. To compare the result achieve by using conventional technique and PSO technique
iv. Build the user friendly software to solve the economic dispatch problem.

1.3 Scope of Project

In this project, there are several scopes that need to cover such as;

i. Study and analyze the use method Particle Swarm Optimization (PSO) to obtain the optimal dispatch power generation.
ii. Simulation and analysis for Economic Dispatch in MATLAB.
iii. Simulation using MATLAB GUI and this stage will be classified to two phases. Development of the GUI gone in two phases, the first phase is designing the layout of GUI and the second phase is the MATLAB GUI programming.

1.4 Thesis Organization

This thesis consists of five chapters including this chapter. The content of each chapter are outlined as follow:

Chapter 2	Contain a detailed description each part of project. It will explain about the MATLAB GUIDE and MATLAB programming.
Chapter 3	Include the project methodology. This will explain how the project is organized and the flow of the process in completing this project
Chapter4	Present the expected result of simulation run using MATLAB GUIDE

Finally the conclusion and the future recommendation of this project and presented are presented in chapter 5.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Literature reviews are very important as a reference for understanding before making the software package as good as possible. The author has studied many journal and article that had been done by previous researcher.

2.2 Economic Dispatch of Power Generation

Electric utility investment practices and operation have been designed to ensure affordable, reliable electricity service to consumers. Affordability and reliability require thoughtful, long-term investments in generation and transmission as well as sophisticated operation of these assets. Economic dispatch (ED) focuses on short-term operational decisions, specifically how to best use available resources to meet customers' electricity needs reliably and at lowest cost. [11] Before explaining further detail about ED let's look for the definition of ED, ED is The operation of generation facilities to produce energy at the lowest cost to reliably serve consumers, recognizing any operational limits of generation and transmission facilities. [12]. In order to solve the economic dispatch problem we have to used the conventional method that include lambda-iteration method, the base point and participation factors method and the gradient method.[3]. In these numerical methods for solution of economic dispatch problem, an essential assumption is that the incremental cost curves of the units are monotonically increasing piecewise-linear functions. Unfortunately, this assumption may render these methods infeasible because of its nonlinear characteristic in practical systems. Furthermore, for a large-scale mixed-generating system, the conventional method has oscillatory problem resulting in a longer solution time [4,5]. A dynamic programming then being introduces to solve the economic dispatch problem with valve-point modeling, but this method may cause the dimensions of economic dispatch problem to become extremely large, thus requiring enormous computational efforts. In this paper, the process to solve a constrained economic dispatch problem using a PSO algorithm was develop to obtain efficiently a high-quality solution within practical power system operation. The PSO algorithm was utilized mainly to determine the optimal generation power of each unit that was submitted to operation at the specific period, thus minimizing the total generation cost [2]

2.3 Artificial Intelligent

Artificial Intelligent (AI) is the ability of a computer or other machine to perform those activities that are normally thought to require intelligence. Many activities involve intelligent action such as problem solving, perception, learning, planning and other symbolic reasoning, creativity, and language.

The applications of AI are used to solve economic dispatch problems for units with piecewise quadratic fuel cost functions and prohibited zones constraint. There a few techniques are used to solve the economic dispatch problem such as Hopfield neural networks, genetic algorithms (GA), simulated annealing (SA) and Particle swarm optimization (PSO).

In this paper, a PSO method for solving the economic dispatch problem in power system is proposed. The process to solve a constrained ED problem using a PSO algorithm was developed to obtain efficiently a high-quality solution within practical power system operation. It was developed through simulation of a simplified social system, and has been found to be robust in solving continuous nonlinear optimization problems. The PSO technique can generate high-quality solutions within shorter calculation time and stable convergence characteristic than other stochastic methods.

2.3.1 Particle Swarm Optimization.

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling.

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

Each particle keeps track of its coordinates in the problem space which are associated with the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called *pbest*. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the neighbors of the particle. This location is called *lbest*. When a particle takes all the population as its topological neighbors, the best value is a global best and is called *gbest*.

The particle swarm optimization concept consists of, at each time step, changing the velocity of (accelerating) each particle toward its p*best* and *lbest* locations (local version of PSO). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward *pbest* and *lbest* locations.

In past several years, PSO has been successfully applied in many research and application areas. It is demonstrated that PSO gets better results in a faster, cheaper way compared with other methods. Although the PSO seems to be sensitive to the tuning of some weights or parameters, many researches are still in progress for proving its potential in solving complex power system problem. In this project, a PSO method for solving the economic dispatch problem in power generation is proposed. The proposed method considers the nonlinear characteristic of a generator such as ramp rate limits and prohibited operating zone for actual power generation operation. The feasibility of the proposed method was demonstrated for three different system, respectively, as compared with the real coded GA method in the solution quality and computation efficiency.

2.4 Problem Formulation

Figure 2.1 shows the one-line diagram of a simple 5-bus power system with generator at buses 1, 2 and 3 and load at buses 2, 3, 4 and 5.

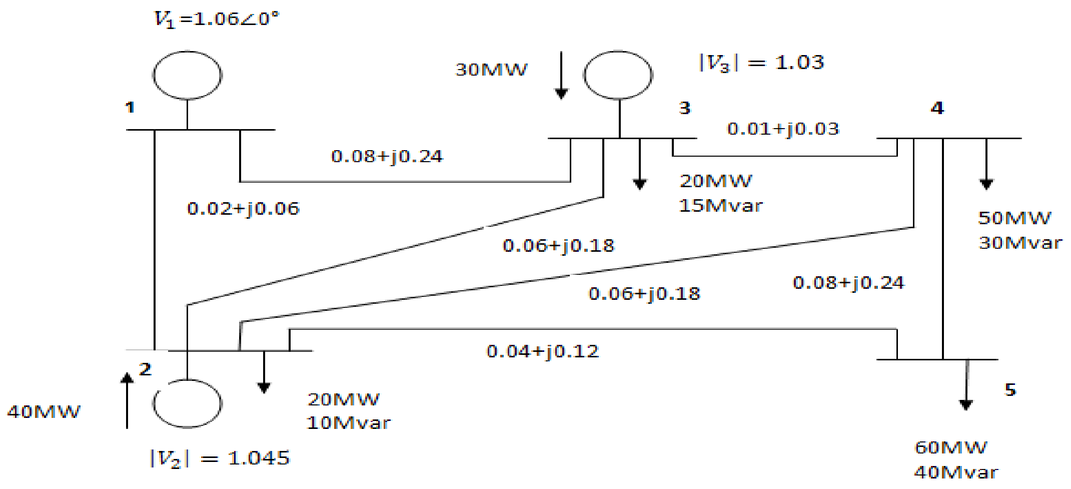


Figure 2.1 : One-line Diagram of the System

The basic economic dispatch problem can described mathematically as a minimization of problem of minimizing the total fuel cost of all committed plants subject to the constraints. The power output of any generator should not exceed its rating nor should it be below that necessary for stable boiler operation. Thus, the generations are restricted to lie within given minimum and maximum limits. The problem is to find the real power generation for each plant such that the objective function (i.e., total production cost) as define by (Equ .6)

$$C_t = \sum_{i=1}^{n_g} C_i = \sum_{i=1}^{n} \propto_i + \beta_i P_i + \gamma_i {P_i}^2 \qquad \text{(Equ. 6)}$$

is minimum, subject to the constraint given by (Equ. 7)

$$\sum_{i=1}^{n_g} P_i = P_D \qquad \text{(Equ. 7)}$$

and the inequality constraints given by

$$P_{i(min)} \leq P_i \leq P_{i(max)} \ , \ i = 1, \ . \ . \ . \ ., n_g \qquad \text{(Equ 8)}$$

Where C_t is the total production cost, C_i is the production cost of ith plant, P_i is the generation of ith plant, P_D is the total load demand, and $n_g$is the total number of dispatchable generating plants , $P_{i(min)}$ and $P_{i(max)}$ are the minimum and maximum generating limits respectively for plant i.

A typical approach is to augment the constraints into objective function by using the Lagrange multipliers

$$\mathcal{L} = C_t + \lambda(P_D - \sum_{i=1}^{ng} P_i) \qquad \text{(Equ. 8)}$$

The Khun-Tucker conditions complement the Lagrangian conditions to include the inequality constraints as additional terms. The necessary condition for the optimal dispatch with losses neglected becomes

$$\frac{dC_i}{dP_i} = \lambda \qquad \text{for} \qquad P_{i(min)} \leq P_i \leq P_{i(max)}$$

$$\frac{dC_i}{dP_i} \leq \lambda \qquad \text{for} \qquad P_i = P_{i(max)} \qquad \text{(equ 9)}$$

$$\frac{dC_i}{dP_i} \geq \lambda \qquad \text{for} \qquad P_i = P_{i(min)}$$

The numerical solution for an estimated λ, P_i are found from the coordination equation (Equ 11) and iteration is continued until $\sum P_i = P_D$.

$$P_i = \frac{\lambda - \beta_i}{2_{\gamma i}} \qquad \text{(Equ 11)}$$

As soon as any plant reaches a maximum or minimum, the plant becomes pegged at the limit. In effect, the plant output becomes a constant, and only the inviolate plants must operate at equal incremental cost.

2.5 MATLB GUI

A graphical user interface (GUI) is a graphical display that contains devices, or components, that enable a user to perform interactive tasks. A good GUI can make programs easier to use by providing them with a friendly appearance and with control icon like pushbuttons, list boxes, sliders, menus, radio button and so forth. To perform these tasks, the user of the GUI does not have to create a script or type commands at the command line. Often, the user does not have to know the details of the task at hand. The GUI should behave in an understandable and predictable manner, so that a user knows what to expect when he or she performs an action. For example, when a mouse click occurs on a pushbutton, the GUI should initiate the action described on the label of the button [8] [9].

Each component, and the GUI itself, are associated with one or more user-written routines known as callbacks. The execution of each callback is triggered by a particular user action such as a button push, mouse click, selection of a menu item, or the cursor passing over a component. The creator of the GUI will provide these callbacks. MATLAB enables the user to create GUIs programmatically or with GUIDE, an interactive GUI builder. It also provides functions that simplify the creation of standard dialog boxes. The technique had chosen depends on the creator experience, preferences, and the kind of GUI that want to create [8] [9].

GUIDE, the MATLAB graphical user interface development environment, provides a set of tools for creating graphical user interfaces (GUIs). These tools simplify the process of laying out and programming GUIs [8] [9].

2.5.1 A Brief Introduction of GUIDE

GUIDE, the MATLAB graphical user interface development environment, provides a set of tools for creating graphical user interfaces (GUIs). These tools simplify the process of laying out and programming GUIs [8].

GUIDE is a primarily a set of layout tools that also generates an M-file that contains code to handle the initialization and launching of the GUI [8]. This M-file also provides a framework for the implementation of the callbacks and the functions will execute when the users activate the component in the GUI [8].

2.5.2 Two Basic Tasks in Process of Implementing a GUI

There are two basic tasks in Process of Implementing a GUI. First, laying out a GUI where MATLAB implement GUIs as figure windows containing various styles of uicontrol

(User Interface) objects. Seconds, programming the GUI where each object must be program to perform the intended action when activated by the user of GUI.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter presents the methodology of this project. This methodology describes on how the project is organize and the flow of the steps in order to complete this project. The methodology is divided into two parts which is simulation and analysis of optimal dispatch in MATLAB. There are three main steps are needed for software development of this project.

Before the project being developed using MATLAB, it is needed to make sure to study and understand the method of economic dispatch of power generation analysis and how MATLAB GUIDE works. The flow chart in figure 3.1 illustrated the sequence of steps for work process. The first step is to study and understands about optimal dispatch analysis and MATLAB. Second step is to develop the programming of PSO method and running the simulation in MATLAB. Figure 3.2 show the flow chart that describing the Particle Swarm Optimization (PSO) algorithm, the first step of the algorithm is to randomly initialize the position and velocity of each particle in the swarm, dispersing them uniformly across the search space [6].The fundamental process of particle swarm optimization is how the particles move through solution space. The termination condition can be the number of iterations, the convergence of the swarm or the achievement of a particular goal fitness value [7].

3.2 Flow Chart of Project

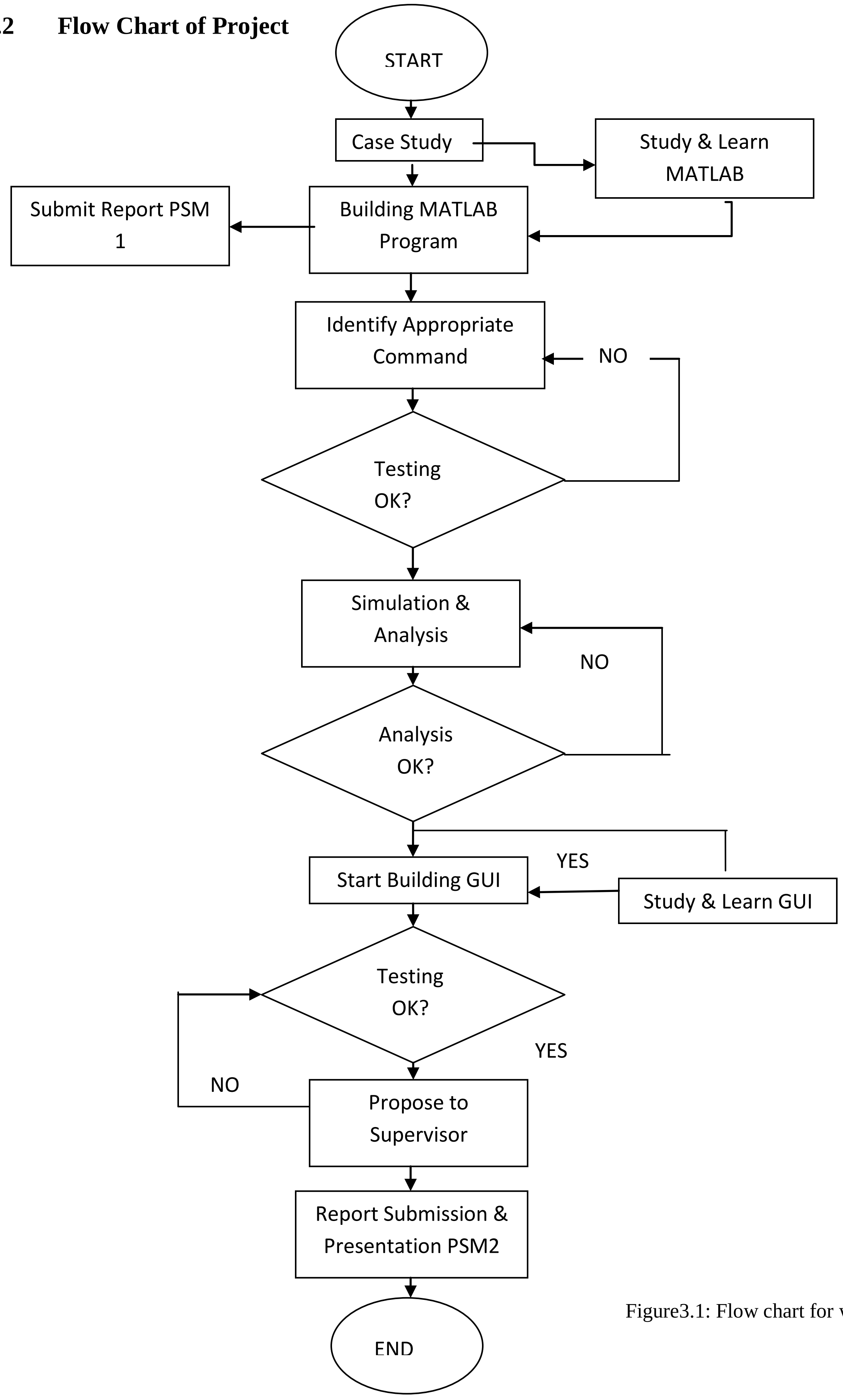


Figure3.1: Flow chart for work process

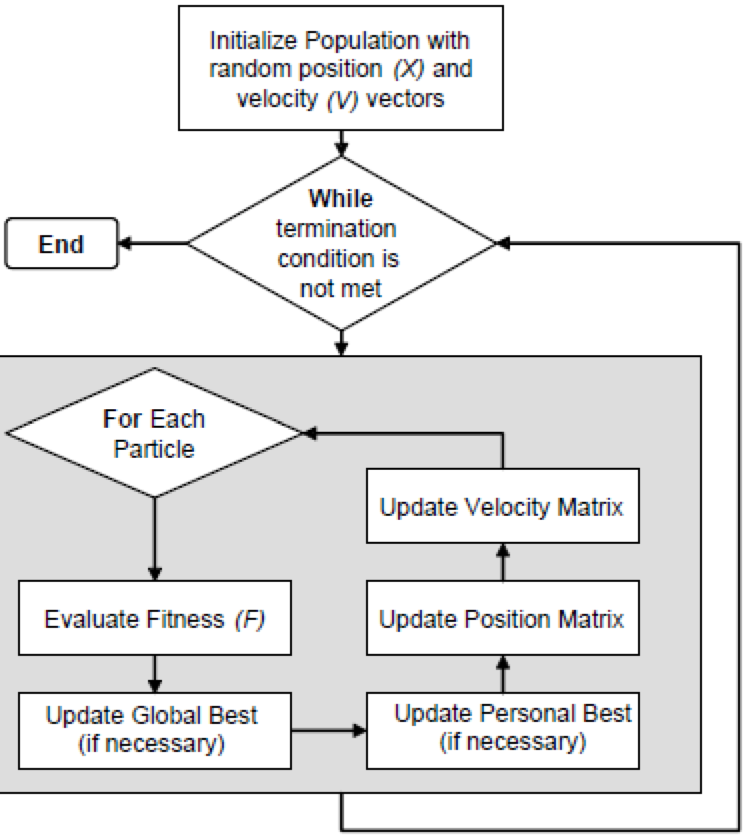

Initialize Population with random position (X) and velocity (V) vectors
While termination condition is not met
End
For Each Particle
Update Velocity Matrix
Evaluate Fitness (F)
Update Position Matrix
Update Global Best (if necessary)
Update Personal Best (if necessary)

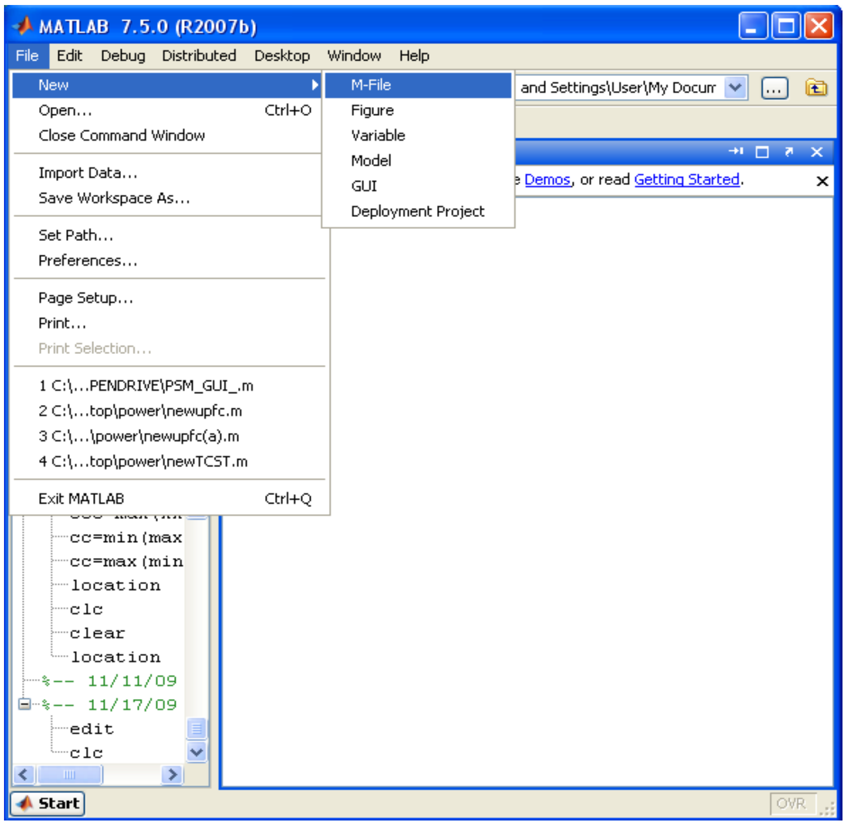
Figure 3.2: Flow chart describing the Particle Swarm Optimization (PSO) algorithm

3.3 Development Command

The MATLAB software provides a full programming language that enables user to write a series of MATLAB statements into a file and then execute them with a single command. User will write a program in an ordinary text file, giving the file a name of filename.m. The term use for filename becomes the new command that MATLAB associates with the program. The file extension of .m makes this a MATLAB M-file.[10] M-files can be "*scripts*" that simply execute a series of MATLAB statements, or they can be" *functions*" that also accept input arguments and produce output.

3.3.1 Creating MATLAB file

To create a new text file in the Editor/Debugger, either click the New M-file button on the MATLAB desktop toolbar, or select File > New > M-File from the MATLAB desktop as shown in figure 3.3.



```
cc=min(max
cc=max(min
location
clc
clear
location
%-- 11/11/09
%-- 11/17/09
edit
clc
```

Figure 3.3: Toolbar for M-file

After clicking the M-file toolbar, a new window will pop-up as shown in figure 3.4 and it is ready to use to make a programming.

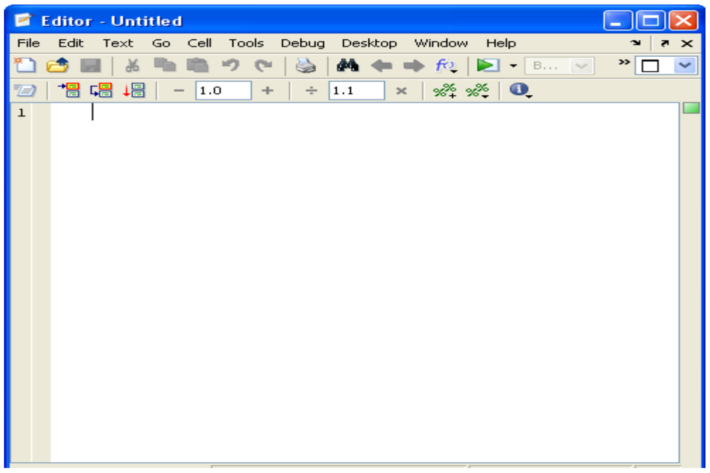
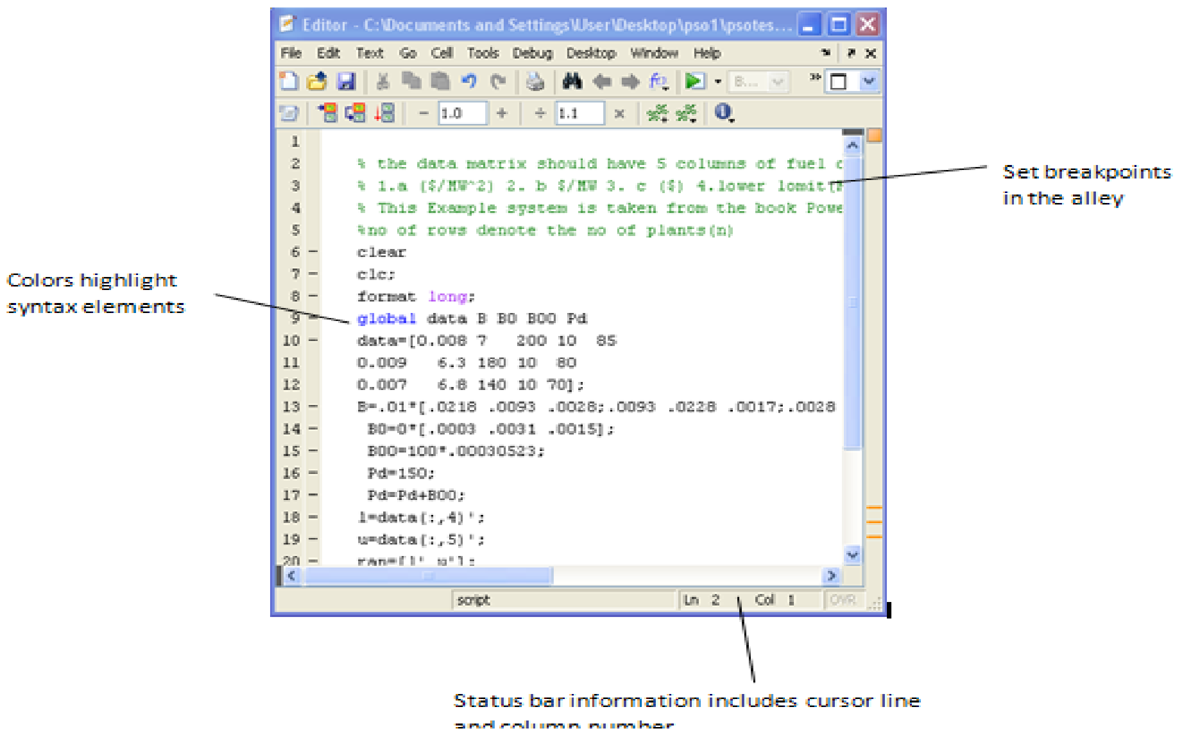


Figure 3.4: New Blank M-File

Now the M-file Editor is ready to use for programming. Figure 3.5 shows the example of the Editor/Debugger outside of the desktop opened to an existing M-file, and calls out some of the tool's useful features.


Editor - C:\Documents and Settings\User\Desktop\pso1\psotes...
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
Set breakpoints in the alley
Colors highlight syntax elements
script
Ln 2
Col 1
OVR
Status bar information includes cursor line and column number

```

% the data matrix should have 5 columns of fuel c
% 1.a ($/MW^2) 2. b $/MW 3. c ($) 4.lower lomit(M
% This Example system is taken from the book Powe
%no of rows denote the no of plants(n)
clear
clc;
format long;
global data B B0 B00 Pd
data=[0.008 7   200 10  85
0.009   6.3 180 10  80
0.007   6.8 140 10 70];
B=.01*[.0218 .0093 .0028;.0093 .0228 .0017;.0028
 B0=0*[.0003 .0031 .0015];
 B00=100*.00030523;
 Pd=150;
 Pd=Pd+B00;
l=data(:,4)';
u=data(:,5)';
ran=[l' u'];
```

Figure 3.5: Example of written Editor

3.4 A Brief Introduction of GUIDE

GUIDE, the MATLAB Graphical User Interface development environment, provides a set of tools for creating graphical user interfaces (GUIs). Usually, GUI will give the user to design the layout of a program. In this section, we will discuss about the development in GUI on how this project be conducted.

3.4.1 Creating Graphical User Interfaces (GUIs)

To start GUIDE, enter guide at MATLAB prompt. The display of GUIDE Quick Start dialog, is shown in figure 3.3

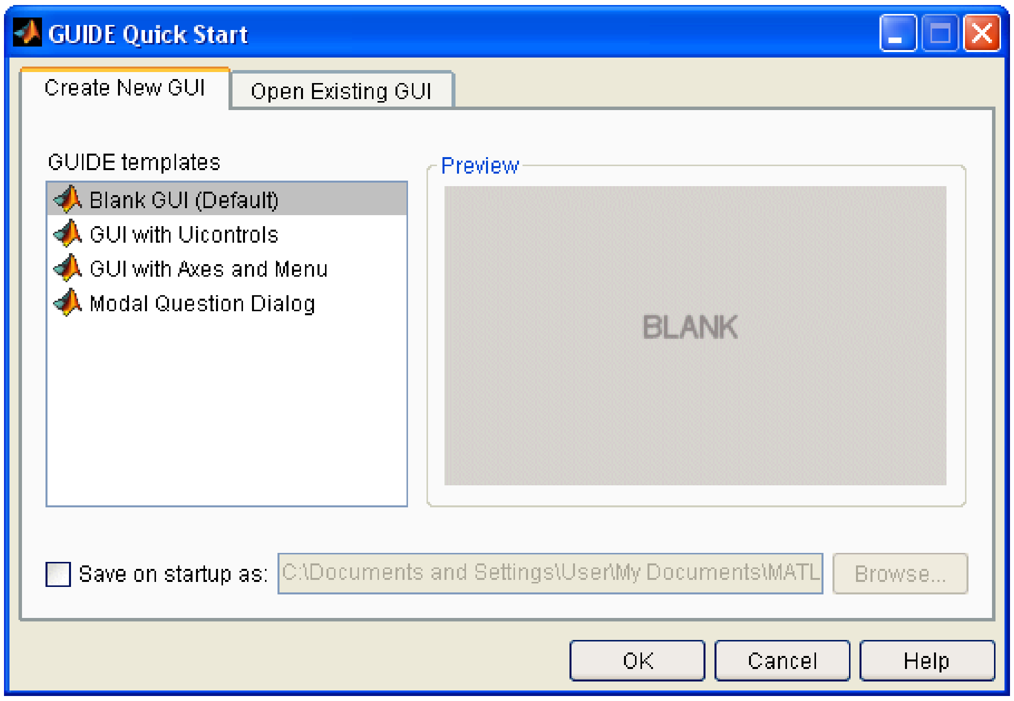


Figure 3.6: Main Page of GUI.

From the Quick Start dialog, user can create a new GUI from one of the GUIDE templates or open an existing GUI. The Create New Guide part will be used to create a new GUI program and after select the option, click OK. The result should be appearing as shown in figure 3.4.

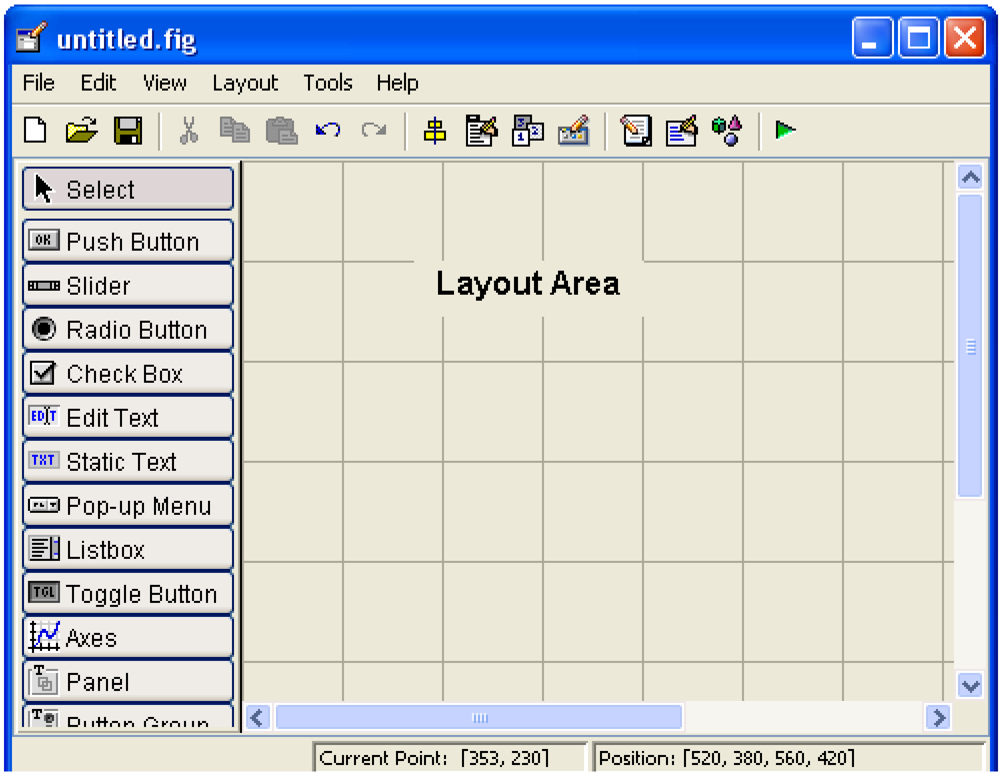


Figure3.7 : Layout Area of GUI

The Open Existing GUI used to callback the previous project that we have saved before.

3.4.2 Layout the GUI

Using the GUIDE Layout Editor, the user can lay out a GUI easily by clicking and dragging GUI component such as panels, buttons, text fields, sliders, menus, and so on into the layout area. All of these component palettes have their own function in GUI. Figure 3.5 shows the GUI page and the name for each button.

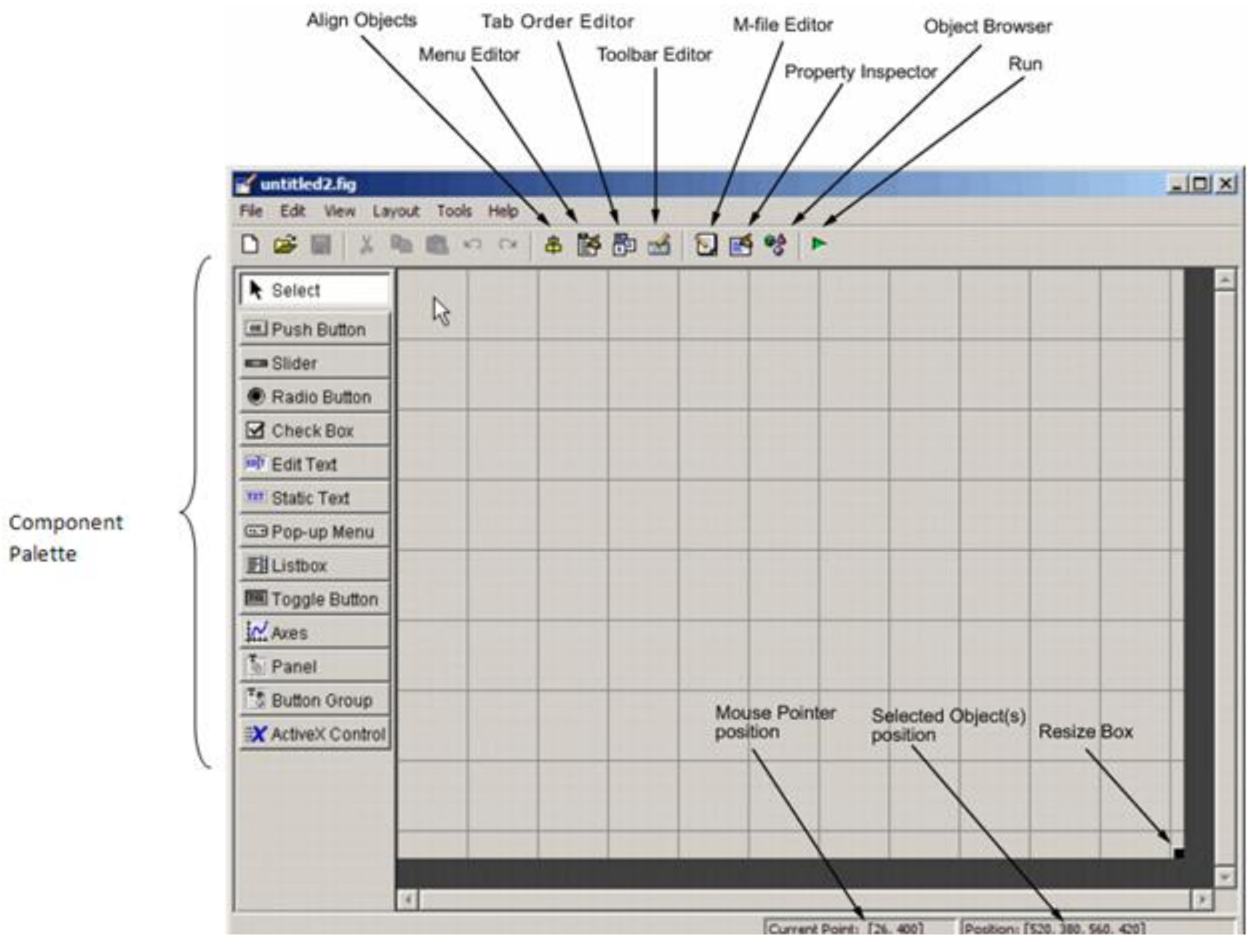


Figure 3.8: GUI Page

3.4.2.1 Component Palette

Table 3.1 below shows the type of components that include in the GUI page and the function for each component.

Component	Icon	Function
Push Button	OK	Push buttons generate an action when clicked. For example, an OK button might apply settings and close a dialog box. When you click a push button, it appears depressed and when you release the mouse button, the push button appears raised.
Toggle Button	TGL	Toggle buttons generate an action and indicate whether they are turned on or off. When you click a toggle button, it appears depressed, showing that it is on. When you release the mouse button, the toggle button remains depressed until you click it a second time. When you do so, the button returns to the raised state, showing that it is off. Use a button group to manage mutually exclusive toggle buttons.
Radio Button	◉	Radio buttons are similar to check boxes, but radio buttons are typically mutually exclusive within a group of related radio buttons. That is, when you select one button the previously selected button is deselected. To activate a radio button, click the mouse button on the object. The display indicates the state of the button. Use a button group to manage mutually exclusive radio buttons.
Check Box	☑	Check boxes can generate an action when checked and indicate their state as checked or not checked. Check boxes are useful when providing the user with a number of independent choices, for example, displaying a toolbar.

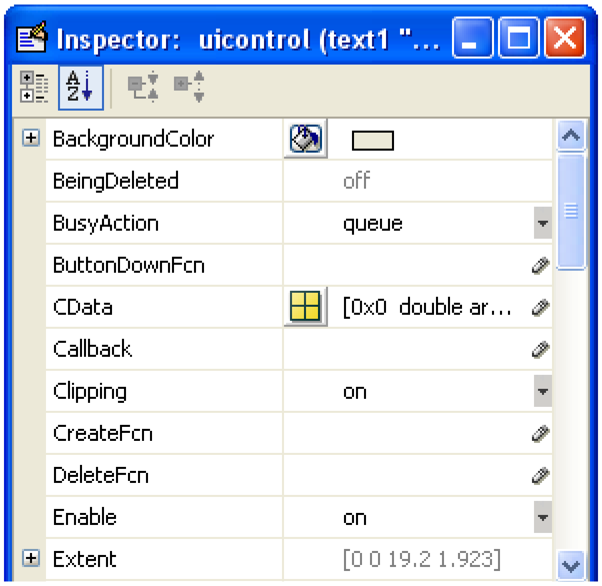
Edit Text		Edit text components are fields that enable users to enter or modify text strings. Use edit text when you want text as input. Users can enter numbers but you must convert them to their numeric equivalents.
Static Text		Static text controls display lines of text. Static text is typically used to label other controls, provide directions to the user, or indicate values associated with a slider. Users cannot change static text interactively.
Slider		Sliders accept numeric input within a specified range by enabling the user to move a sliding bar, which is called a slider or thumb. Users move the slider by clicking the slider and dragging it, by clicking in the trough, or by clicking an arrow. The location of the slider indicates the relative location within the specified range.
List Box		List boxes display a list of items and enable users to select one or more items.
Pop-Up Menu		Pop-up menus open to display a list of choices when users click the arrow.
Axes		Axes enable your GUI to display graphics such as graphs and images. Like all graphics objects, axes have properties that you can set to control many aspects of its behavior and appearance. See Axes Properties in the MATLAB Graphics documentation and commands such as the following for more information on axes objects: plot, surf, line, bar, polar, pie, contour, and mesh. See Functions by Category in the MATLAB Function Reference documentation for a complete list.
Panel		Panels arrange GUI components into groups. By visually grouping related controls, panels can make the user interface easier to

		understand. A panel can have a title and various borders. Panel children can be user interface controls and axes as well as button groups and other panels. The position of each component within a panel is interpreted relative to the panel. If you move the panel, its children move with it and maintain their positions on the panel.
Button Group		Button groups are like panels but are used to manage exclusive selection behavior for radio buttons and toggle buttons.
ActiveX Component		ActiveX components enable you to display ActiveX controls in your GUI. They are available only on the Microsoft Windows platform. An ActiveX control can be the child only of a figure, i.e., of the GUI itself. It cannot be the child of a panel or button group. See ActiveX Control in this document for an example. See MATLAB COM Client Support in the MATLAB External Interfaces documentation to learn more about ActiveX controls.

Table 3.1: Name and Function of Component Palette

3.4.3 Property Inspector

After placing the component in the layout area, the properties of each GUI component must be set. The *Property Inspector* from the *View* menu is chosen to display the property Inspector dialog box. When the component in the Layout Editor is chosen, the Property Inspector will display that component's properties. If no component is selected, the Property Inspector displays the properties of the GUI figure. Figure 3.6 shows the dialog box for Property Inspector


Inspector: uicontrol (text1 "...

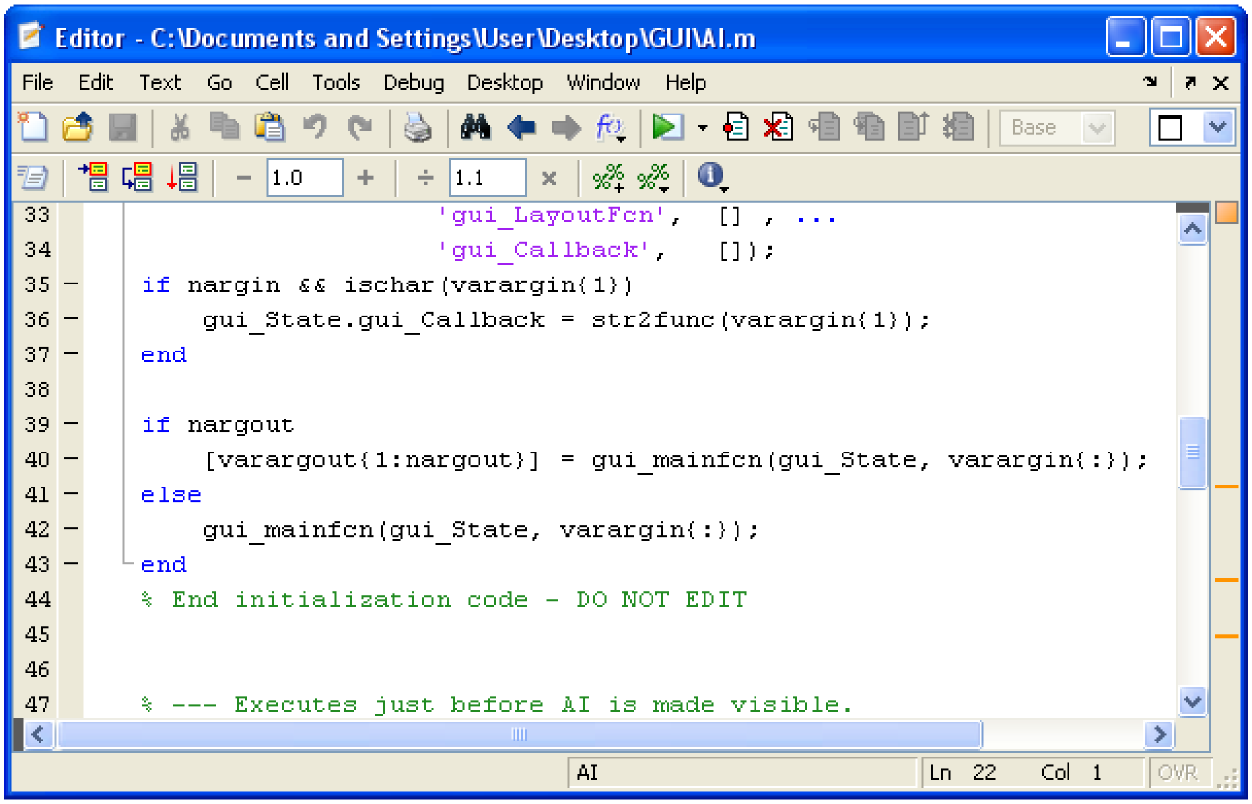
BackgroundColor	
BeingDeleted	off
BusyAction	queue
ButtonDownFcn	
CData	[0x0 double ar...
Callback	
Clipping	on
CreateFcn	
DeleteFcn	
Enable	on
Extent	[0 0 19.2 1.923]

Figure 3.9: Property Inspector

The Property Inspector is used to give each component a name and to set the characteristic of each component such as color, name, font size, tag and others.

3.5 Program the GUI

GUIDE automatically generates an M-file that controls how the GUI operates. The M-files initializes the GUI and contains a framework for all the GUI callbacks, the commands that are executed when a user clicks a GUI component.


Editor - C:\Documents and Settings\User\Desktop\GUI\AI.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Base
1.0
1.1
AI
Ln 22 Col 1
OVR

```
                          'gui_LayoutFcn',  [] , ...
                          'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before AI is made visible.
```

Figure 3.10: Example of M-file

Figure 3.7 was the example of M-file that is generated when the complete layout has been design and saved. By using M-file, the behavior of the GUI can be programmed by several code or function. This code will be programmed to give responded for component palette that has been designed in the Layout area.

The programming M-file was the most difficult part in developing GUI because it need an extra reading on the coding before GUI can performed.

CHAPTER 4

RESULT AND ANALYSIS

4.1 Introduction

The discussion of this chapter was categorized into two parts. The first parts consist of the discussions on the software package development using MATLAB Graphical User Interface Development Environment. Meanwhile the second part consists of the Economic Dispatch simulation result with MATLAB based on problem in Section 3.

4.2 Economic Dispatch of Power Generation Software Package

4.2.1 Introduction

Economic Dispatch of Power Generation Software Package is a MATLAB GUI files that has been developed to solve the economic dispatch problem. The whole software package file was installed in the C:\Program Files\MATLAB\R2007b\work\power. This is important because if this step is not allowed, the user will face difficulties to load the software. The first time user must add entire folder by click the 'Add with Subfolder…' as shown in figure 4.1. After the adding path process completed by clicking save button, users just have to type 'EDP' in the command window to load the software package.

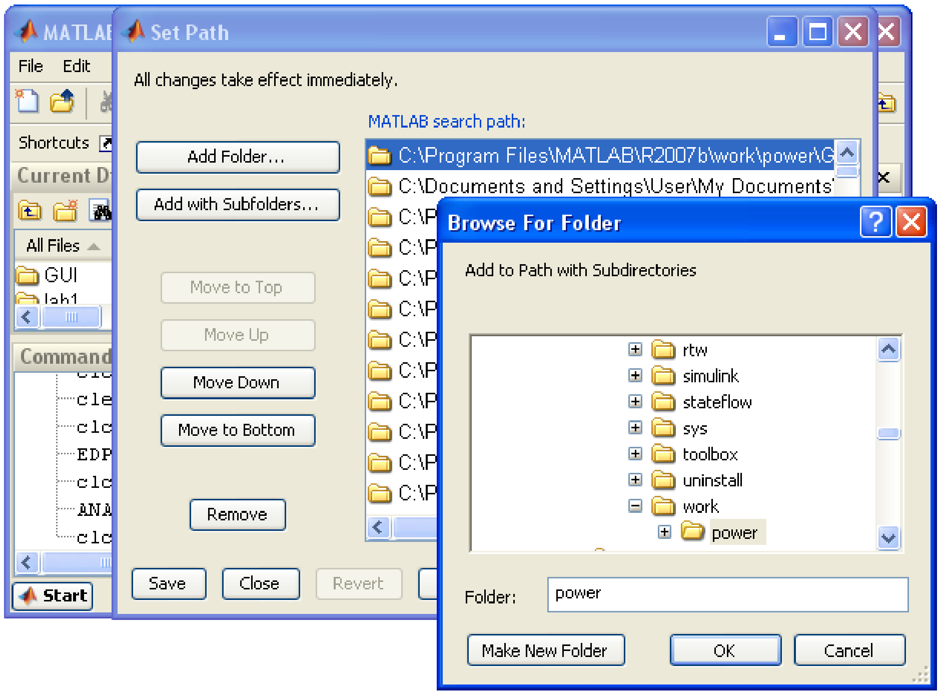
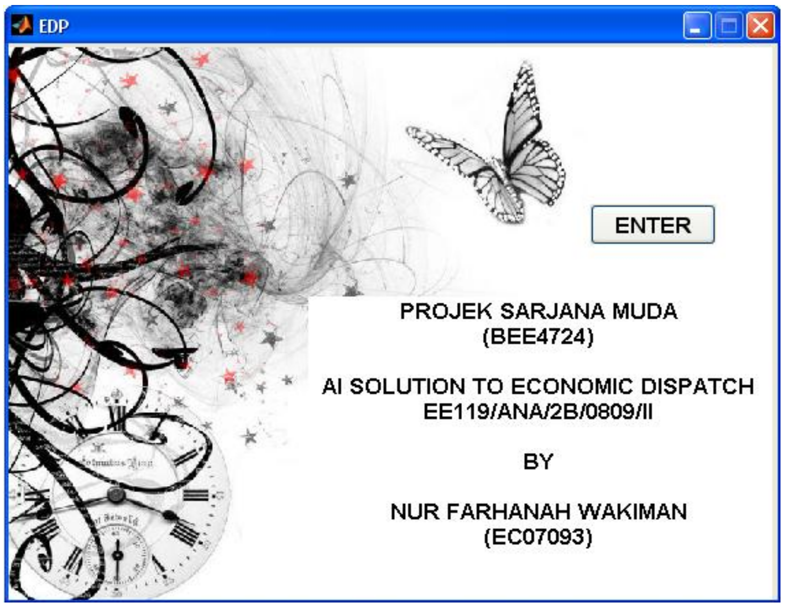


Figure 4.1: MATLAB Set Path

To view the list of folder in EDP, user can type 'cd(C:\Program Files\MATLAB\R2007b\work\power) and followed by 'ls'. If the users want to learn more about this command, user can type 'help cd' or 'help ls' in command window in MATLAB.

4.2.2 Detail of Software Package

After entered the MATLAB window by clicking twice at MATLAB icon, user must load the software package. In order to load the Economic Dispatch of Power Generation Software Package, user must type 'EDP' in MATLAB command window. The main page of the software package will appear as shown in figure 4.2



EDP

ENTER

PROJEK SARJANA MUDA
(BEE4724)

AI SOLUTION TO ECONOMIC DISPATCH
EE119/ANA/2B/0809/II

BY

NUR FARHANAH WAKIMAN
(EC07093)

Figure 4.2: Main Page of Software Package

This main page will give an introduction about the project such as the name of the subject, subject code, title of project, project code, the designer and the designer identity number.

In this page also include one push button name ‘ENTER’. This push button will open the new window named MENU. Figure 4.3 shows the MENU page.

This window contain four push button which are ‘ARTIFICIAL INTELLIGENT’, ‘ECONOMIC DISPATCH’, ‘ECONOMIC DISPATCH ANALYSIS’ and ‘CLOSE’. The first button which is the ‘ARTIFICIAL INTELLIGENCE’ button, it will give the user some information about the definition of Artificial Intelligence (AI) and Particle Swarm Optimization (PSO) as shown in figure 4.4. For the second button, ‘ECONOMIC DISPATCH’ button, same as the first button this window gives brief information about Economic Dispatch (ED) of power generation analysis as shown in figure 4.5. The third button, ‘ECONOMIC DISPATCH ANAYSIS’ button will open new window names ‘TYPE’ which is shown in figure 4.6. And the last button is ‘CLOSE’ button. When the users click this button, a pop-up window will appear shown in figure 4.7. The pop-up window will ask user whether they are very sure to close the entire windows that are opened. These functions are applied for ‘MENU’ window only. Another ‘CLOSE’ button that use on other window can just close the window only.

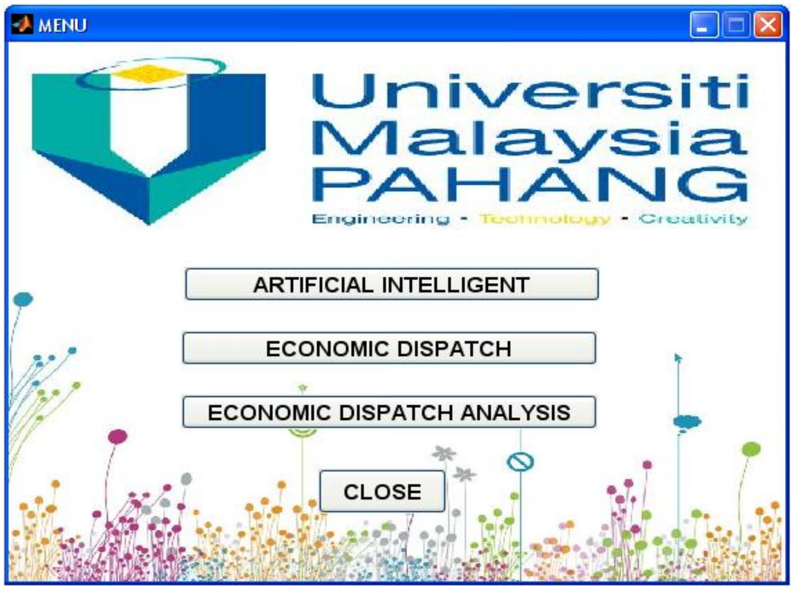
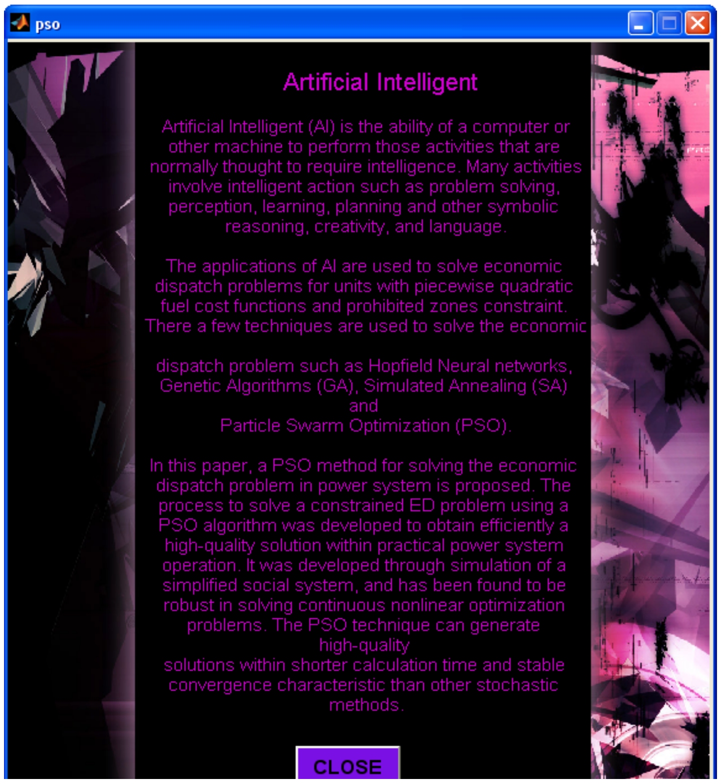

MENU
Universiti Malaysia PAHANG
Engineering • Technology • Creativity
ARTIFICIAL INTELLIGENT
ECONOMIC DISPATCH
ECONOMIC DISPATCH ANALYSIS
CLOSE

Figure 4.3: Menu Page



Artificial Intelligent

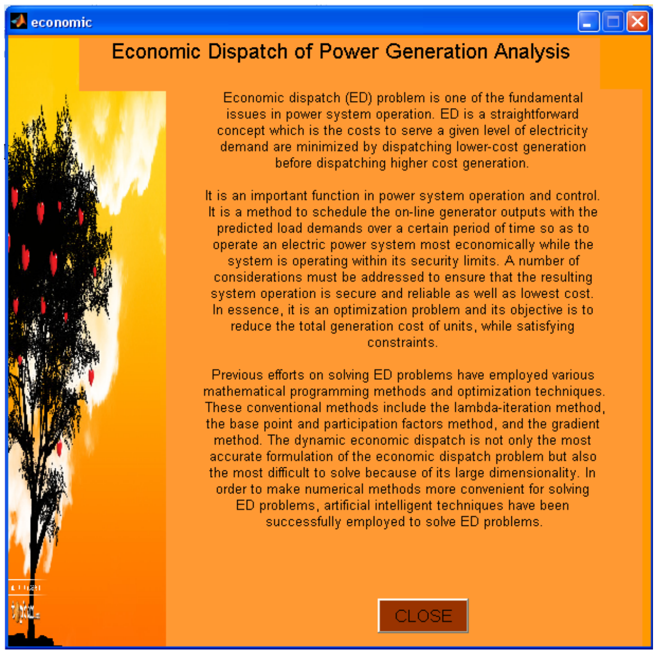
Artificial Intelligent (AI) is the ability of a computer or other machine to perform those activities that are normally thought to require intelligence. Many activities involve intelligent action such as problem solving, perception, learning, planning and other symbolic reasoning, creativity, and language.

The applications of AI are used to solve economic dispatch problems for units with piecewise quadratic fuel cost functions and prohibited zones constraint. There a few techniques are used to solve the economic

dispatch problem such as Hopfield Neural networks, Genetic Algorithms (GA), Simulated Annealing (SA) and
Particle Swarm Optimization (PSO).

In this paper, a PSO method for solving the economic dispatch problem in power system is proposed. The process to solve a constrained ED problem using a PSO algorithm was developed to obtain efficiently a high-quality solution within practical power system operation. It was developed through simulation of a simplified social system, and has been found to be robust in solving continuous nonlinear optimization problems. The PSO technique can generate high-quality
solutions within shorter calculation time and stable convergence characteristic than other stochastic methods.

Figure 4.4: Information of AI and PSO



Economic Dispatch of Power Generation Analysis

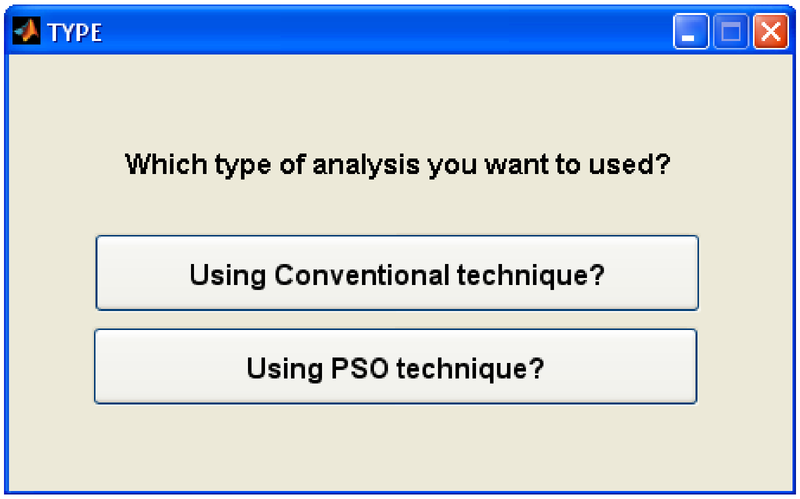
Economic dispatch (ED) problem is one of the fundamental issues in power system operation. ED is a straightforward concept which is the costs to serve a given level of electricity demand are minimized by dispatching lower-cost generation before dispatching higher cost generation.

It is an important function in power system operation and control. It is a method to schedule the on-line generator outputs with the predicted load demands over a certain period of time so as to operate an electric power system most economically while the system is operating within its security limits. A number of considerations must be addressed to ensure that the resulting system operation is secure and reliable as well as lowest cost. In essence, it is an optimization problem and its objective is to reduce the total generation cost of units, while satisfying constraints.

Previous efforts on solving ED problems have employed various mathematical programming methods and optimization techniques. These conventional methods include the lambda-iteration method, the base point and participation factors method, and the gradient method. The dynamic economic dispatch is not only the most accurate formulation of the economic dispatch problem but also the most difficult to solve because of its large dimensionality. In order to make numerical methods more convenient for solving ED problems, artificial intelligent techniques have been successfully employed to solve ED problems.

Figure 4.5: Information of ED

This figure is generated by economic.fig,economic.asv and economic.m file here the first two will generate the window. This window briefly gives the information to the user about economic dispatch of power generation analysis. The 'CLOSE' pushbutton will close this window and call back the menu page of the package software window.

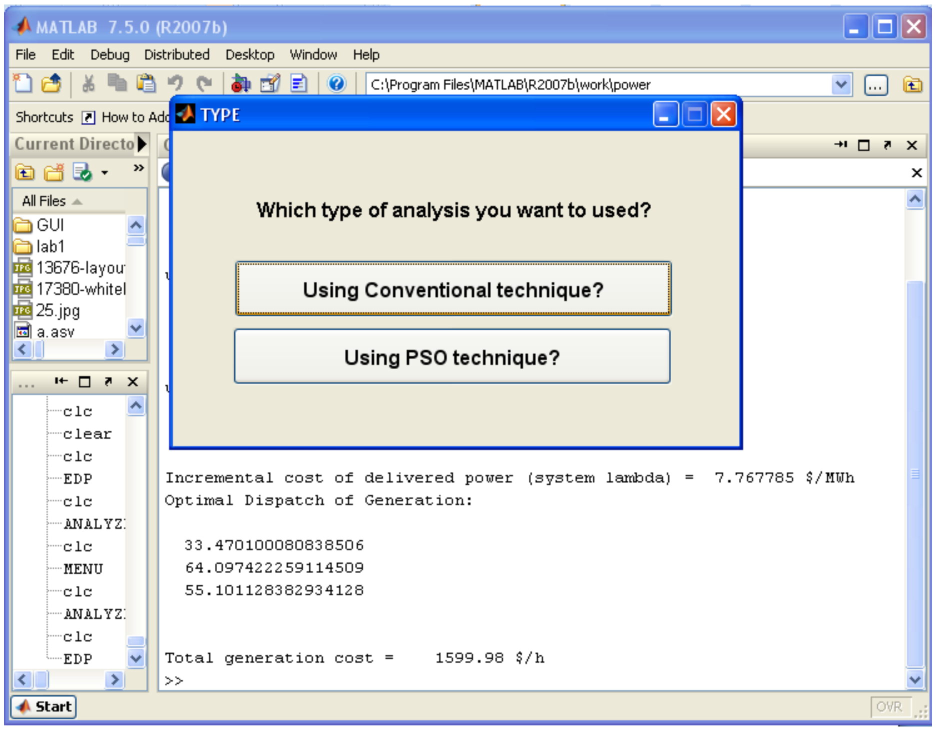


Which type of analysis you want to used?

Figure 4.6: Type of analysis window

This window will ask the user what type of analysis they want to use for the analysis. There are two types of analysis which is using conventional technique and PSO technique. This two buttons will run the programming and give different output as shown in figure 4.7 and 4.8. each time the user choose type of analysis there will be a pop-up window asking user whether they are sure to used this type of analysis.

Figure 4.8 shows the output of analysis when users choose to used conventional technique.


MATLAB 7.5.0 (R2007b)
File Edit Debug Distributed Desktop Window Help
C:\Program Files\MATLAB\R2007b\work\power
Current Directory
GUI
lab1
13676-layou
17380-whitel
25.jpg
a.asv
TYPE
Using Conventional technique?
Using PSO technique?
Start

Which type of analysis you want to used?

```
Incremental cost of delivered power (system lambda) =   7.767785 $/MWh
Optimal Dispatch of Generation:

  33.470100080838506
  64.097422259114509
  55.101128382934128


Total generation cost =    1599.98 $/h
>>
```

Figure 4.7: Result for conventional technique

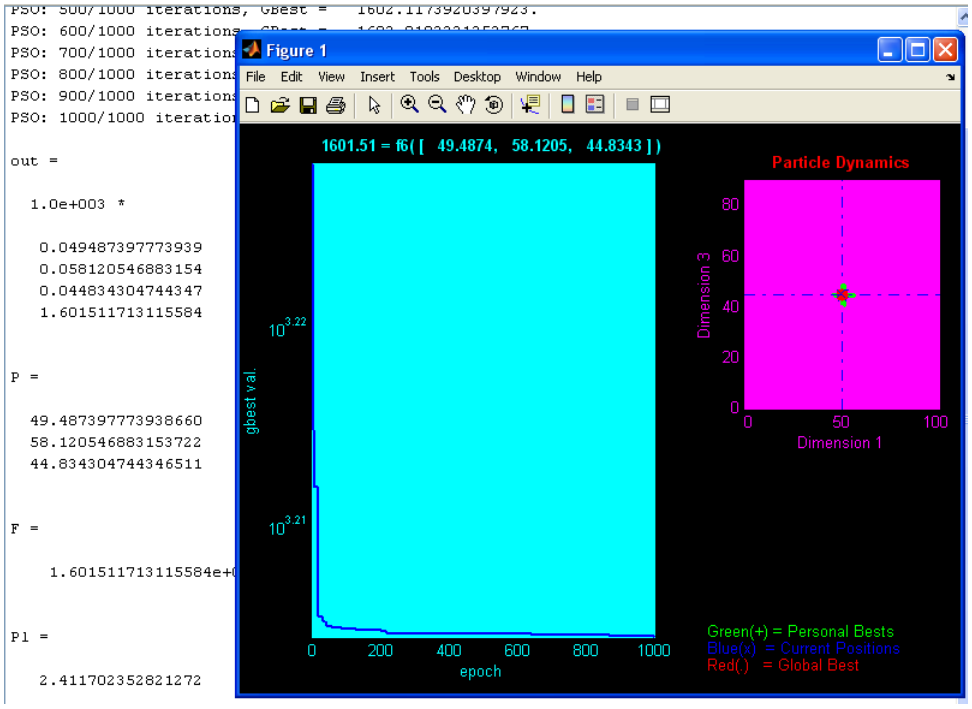


Figure 1
File Edit View Insert Tools Desktop Window Help
1601.51 = f6([49.4874, 58.1205, 44.8343])
$10^{3.22}$
$10^{3.21}$
gbest val.
0
200
400
600
800
1000
epoch
Particle Dynamics
Dimension 3
0
20
40
60
80
0
50
100
Dimension 1
Green(+) = Personal Bests
Blue(x) = Current Positions
Red( ) = Global Best

```
PSO: 500/1000 iterations, GBest =   1602.1173920397923.
PSO: 600/1000 iterations
PSO: 700/1000 iterations
PSO: 800/1000 iterations
PSO: 900/1000 iterations
PSO: 1000/1000 iteratio

out =

  1.0e+003 *

   0.049487397773939
   0.058120546883154
   0.044834304744347
   1.601511713115584


P =

  49.487397773938660
  58.120546883153722
  44.834304744346511


F =

     1.601511713115584e+0


Pl =

   2.411702352821272
```

Figure 4.8: Result for PSO technique

4.2.3 GUI Programming.

After designing the GUI and setting component properties, the need to be programmed. User can programmed the GUI by coding one or more callbacks for each of components. Callbacks are functions that used to execute in response to some action by the user.

GUI callback can be found in an M-file generated by the GUIDE automatically. GUIDE add templates for the most commonly used callbacks to this M-files, but user may want to add other M-files used to edit the files.

The created GUI uses the dialog box and image to make the GUI more attractive. The coding used during programming can be referred on the appendix.

4.3 Analysis Using Conventional Technique

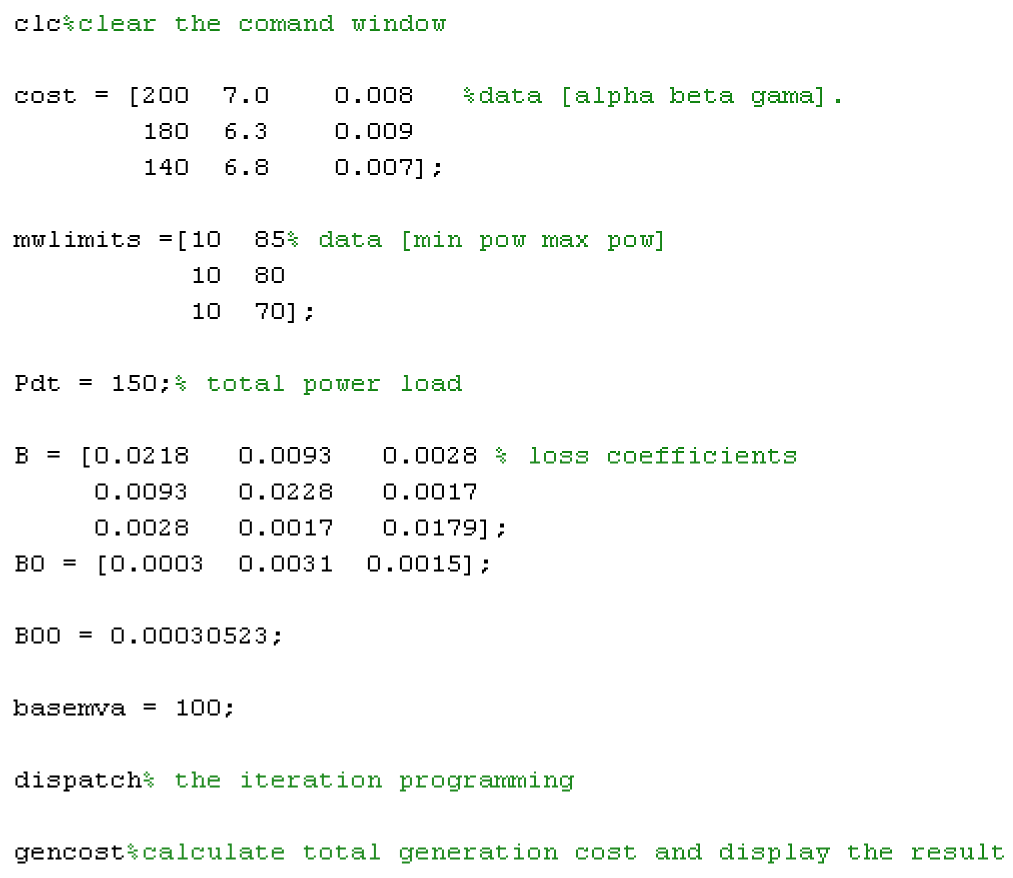
4.3.1 Introduction

ECONOMIC dispatch (ED) problem is one of the fundamental issues in power system operation. In essence, it is an optimization problem and its objective is to reduce the total generation cost of units, while satisfying constraints. Previous efforts on solving ED problems have employed various mathematical programming methods and optimization techniques. These conventional methods include the lambda-iteration method, the base point and participation factors method, and the gradient method. In this project, the lambda iteration is chosen as the conventional method.

4.3.2 Solve Programming

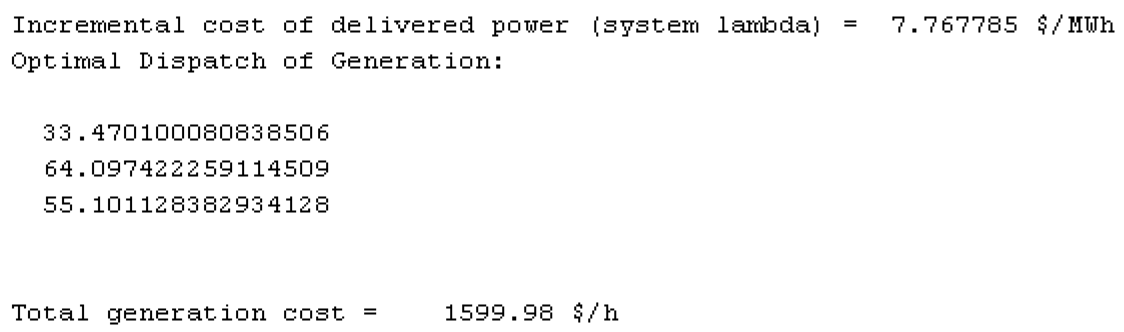
This conventional mathematic method are used to solve the one-line diagram of a simple 5-bus power system with generator at buses 1,2 and 3. Bus2 and 3 are 1.045pu, 40MW and 1.030pu 30MW respectively. The load and MW and Mvar values are shown on the diagram. Line impedance and one-half of the line capacitive susceptance are given in in per unit on aa 100MVA base.

In order to solve the problem, following command is used



```
clc%clear the comand window

cost = [200   7.0    0.008   %data [alpha beta gama].
         180   6.3    0.009
         140   6.8    0.007];

mwlimits =[10   85% data [min pow max pow]
            10   80
            10   70];

Pdt = 150;% total power load

B = [0.0218   0.0093   0.0028 % loss coefficients
     0.0093   0.0228   0.0017
     0.0028   0.0017   0.0179];
B0 = [0.0003   0.0031   0.0015];

B00 = 0.00030523;

basemva = 100;

dispatch% the iteration programming

gencost%calculate total generation cost and display the result
```

and the result obtain as shown below:-



```
Incremental cost of delivered power (system lambda) =  7.767785 $/MWh
Optimal Dispatch of Generation:

   33.470100080838506
   64.097422259114509
   55.101128382934128


Total generation cost =     1599.98 $/h
```

4.4 Analysis Using PSO Technique

4.4.1 Introduction

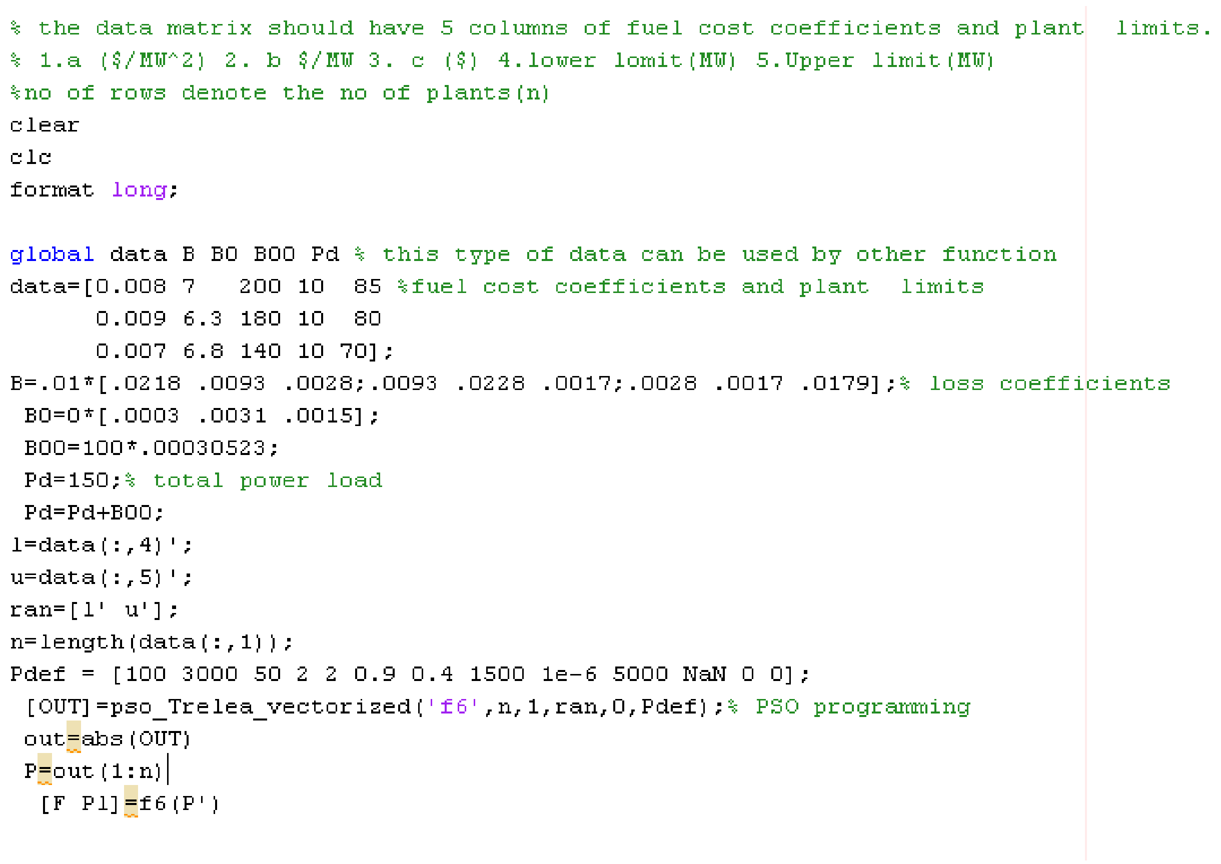
ECONOMIC dispatch (ED) problem is one of the fundamental issues in power system operation. In essence, it is an optimization problem and its objective is to reduce the total generation cost of units, while satisfying constraints. In these numerical methods for solution of ED problems, an essential assumption is that the incremental cost curves of the units are monotonically increasing piecewise-linear functions. Unfortunately, this assumption may render these methods infeasible because of its nonlinear characteristics in practical systems. These nonlinear characteristics of a generator include discontinuous prohibited zones, ramp rate limits, and cost functions which are not smooth or convex. Furthermore, for a large-scale mixed-generating system, the conventional method has oscillatory problem resulting in a longer solution time.

In order to make numerical methods more convenient for solving ED problems, artificial intelligent techniques, such as the Hopfield neural networks, have been successfully employed to solve ED problems for units with piecewise quadratic fuel cost functions and prohibited zones constraint. In this paper, a PSO method for solving the ED problem in power system is proposed. The proposed method considers the nonlinear characteristics of a generator such as ramp rate limits and prohibited operating zone for actual power system operation.

4.4.2 PSO Programming

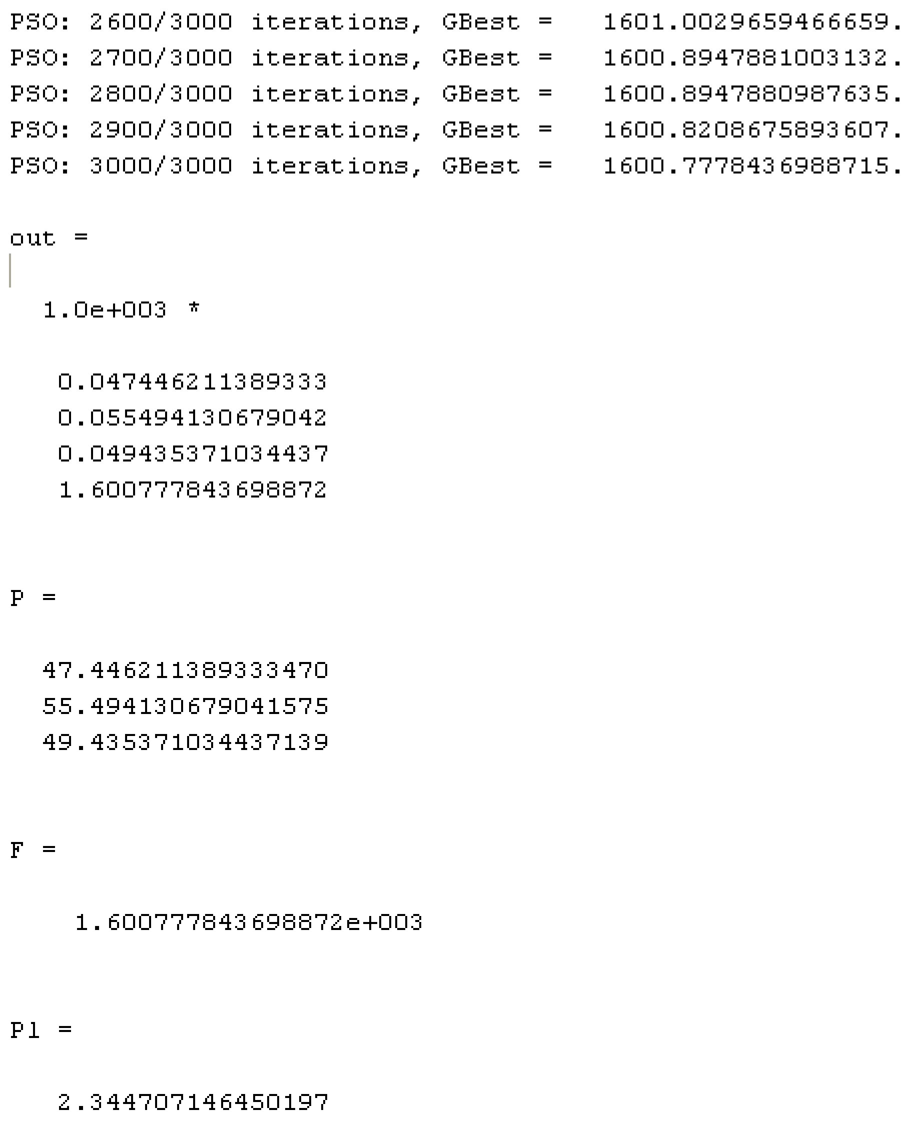
This programming are used to solve the one-line diagram of a simple 5-bus power system with generator at buses 1,2 and 3. Bus 2 and 3 are 1.045pu, 40MW and 1.030pu 30MW respectively. The load and MW and Mvar values are shown on the diagram. Line impedance and one-half of the line capacitive susceptance are given in per unit on a 100MVA base.

In order to solve the problem, following command is used



```
% the data matrix should have 5 columns of fuel cost coefficients and plant  limits.
% 1.a ($/MW^2) 2. b $/MW 3. c ($) 4.lower lomit(MW) 5.Upper limit(MW)
%no of rows denote the no of plants(n)
clear
clc
format long;

global data B B0 B00 Pd % this type of data can be used by other function
data=[0.008 7   200 10  85 %fuel cost coefficients and plant  limits
      0.009 6.3 180 10  80
      0.007 6.8 140 10 70];
B=.01*[.0218 .0093 .0028;.0093 .0228 .0017;.0028 .0017 .0179];% loss coefficients
 B0=0*[.0003 .0031 .0015];
 B00=100*.00030523;
 Pd=150;% total power load
 Pd=Pd+B00;
l=data(:,4)';
u=data(:,5)';
ran=[l' u'];
n=length(data(:,1));
Pdef = [100 3000 50 2 2 0.9 0.4 1500 1e-6 5000 NaN 0 0];
 [OUT]=pso_Trelea_vectorized('f6',n,1,ran,0,Pdef);% PSO programming
 out=abs(OUT)
 P=out(1:n)
  [F Pl]=f6(P')
```

and the result obtain as shown below:-



```
PSO: 2600/3000 iterations, GBest =   1601.0029659466659.
PSO: 2700/3000 iterations, GBest =   1600.8947881003132.
PSO: 2800/3000 iterations, GBest =   1600.8947880987635.
PSO: 2900/3000 iterations, GBest =   1600.8208675893607.
PSO: 3000/3000 iterations, GBest =   1600.7778436988715.

out =

  1.0e+003 *

   0.047446211389333
   0.055494130679042
   0.049435371034437
   1.600777843698872


P =

  47.446211389333470
  55.494130679041575
  49.435371034437139


F =

     1.600777843698872e+003


P1 =

   2.344707146450197
```

Figure 4.9 show the movement of particle data during the calculation.

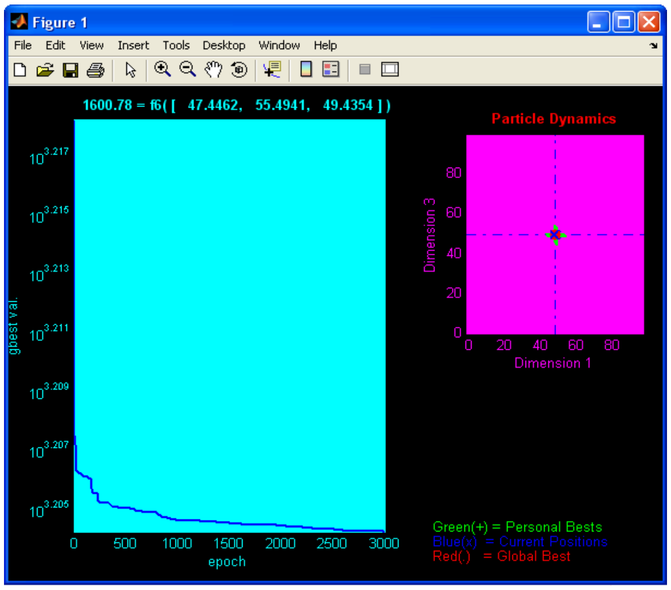


Figure 4.9: Figure of Movement Data

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

At the end of this project, we can identified the optimum value that simultaneously minimize the generation cost rate and to meet the load demand and experimental results show that the proposed PSO method was indeed capable of obtaining higher quality solutions efficiently in ED problems.

And by creating a friendly GUI modeling, user will be more interested to used this software because of the information that they will gain and the interesting design of GUI

5.2 Future Recommendation

This project is well functioning. For future improvement, several suggestions are proposed for more advanced and better application in future:

- Try to add more function in the software. Not only calculate the economic dispatch. Besides calculating the economic dispatch only, try to add other application in the software for an example, build other programming to solve economic dispatch and optimal dispatch.
- To make this software more users friendly and interesting, try to add function that allowed user to key in the input data and add some sound or music during the iteration session.
- Time taken for this software to gain output is depending on number of iteration, try to build a programming that can help user identify the time taken for every calculation.

5.3 Commercialization

This project and thesis is meant for the academic motivation of all electrical engineering students, especially in teaching and learning session in Universiti Malaysia Pahang (UMP). All the lecturer of UMP are allowed to use this project and thesis for teaching session. I give full authorities to my supervisor Dr Ahmed N. Abd Alla to handle the matter that come further about the usefulness of my project.

REFERENCES

[1] X.S. Han, H. B. Gooi., Dynamic Economic Dispatch: Feasible and Optimal Solutions,2001

[2] Zwe-Lee Gaing, ,“Particle Swarm Optimization to Solving the Economic Dispatch Considering the Generator Constraints” , IEEE TRANSACTIONS ON POWER SYSTEMS, VOL. 18, NO. 3, AUGUST 2003

[3] A.Bakirtzis,V.Petridis, and SKazarlis, “Genetic algorithm solution to the economic dispatch problem,”Proc.Inst. Elect. Eng.-Gen., Transm. Dist. Vol. 141, no.4, pp. 377-382,July1994

[4] C.C.Fung, S. Y. Chow,and K. P. Wong, “Solving the economic dispatch problem with an integrated parallel genetic algorithm,” in Proc PowerCon Int. Conf., vol. 3,2000,pp.1257-1262

[5] H.Saadat,Power System Analysis. New York:McGraw-Hill,1999

[6] J.M. Johnson and Y. Rahmat-Samii, “Genetic Algorithms in EngineeringElectromagnetics,” *IEEE Antennas and Propagation Magazine*, vol. 39, no. 4, pp. 7 -21,1997.

[7] Yahya Rahmat-Samii, Dennis Gies, and Jacob Robinson “Particle Swarm Optimization (PSO): A Novel Paradigm for Antenna Designs”

[8] Chapman, Stephen J., MATLAB Programming for Engineer, Brooks Cole, 2001.

[9] Creating Graphical User Interfaces, Version 7: The Math Work

[10] http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/f7-41453.html#f7-38070

[11] United States Department of Energy, “The Value Of Economic Dispatch”, A Report To Congress Pursuant To Section 1234 Of The Energy Policy Act Of 2005

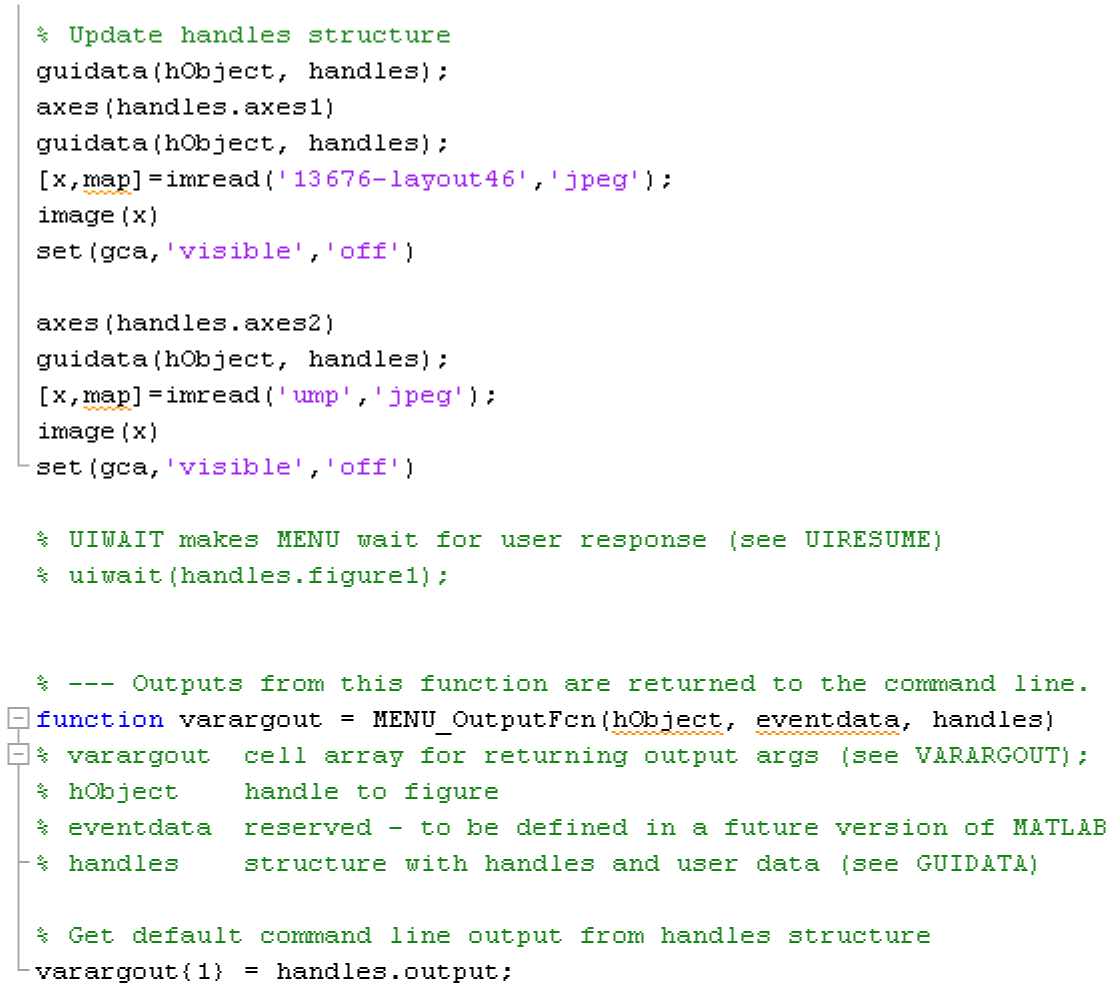
[12] FERC Staff Boston, Massachussetts, “**Economic Dispatch: Concepts, Practices and Issues”,** Presentation to the Joint Board for the Study of Economic Dispatch, November 29, 2005

APPENDIX A

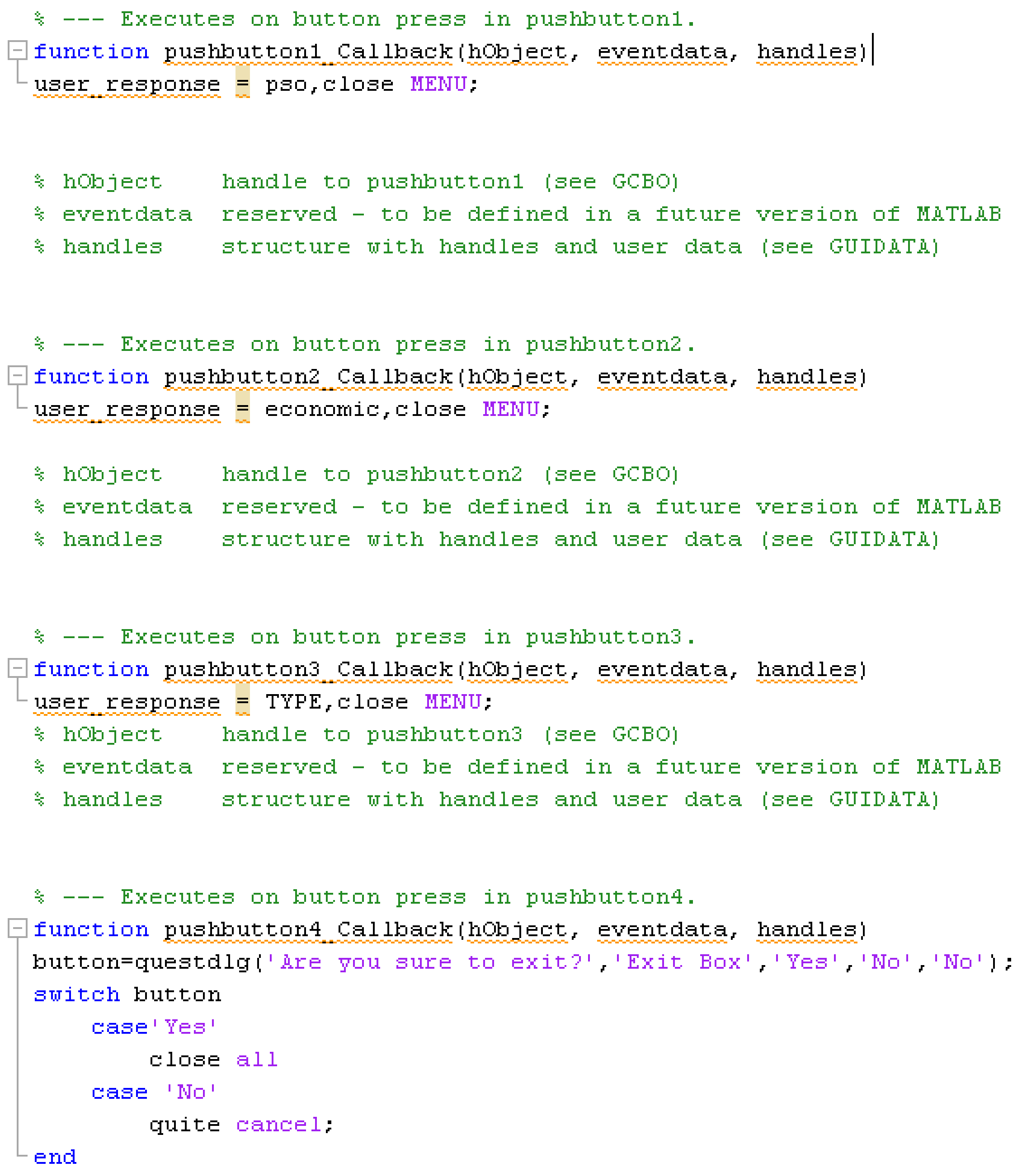
GUI PROGRAMMING

```
% Choose default command line output for EDP
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('17380-whiteblackbut','jpeg');
image(x)
set(gca,'visible','off')

% UIWAIT makes EDP wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = EDP_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
user_response=MENU,close EDP;
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

GUI PROGRAMMING FOR MENU

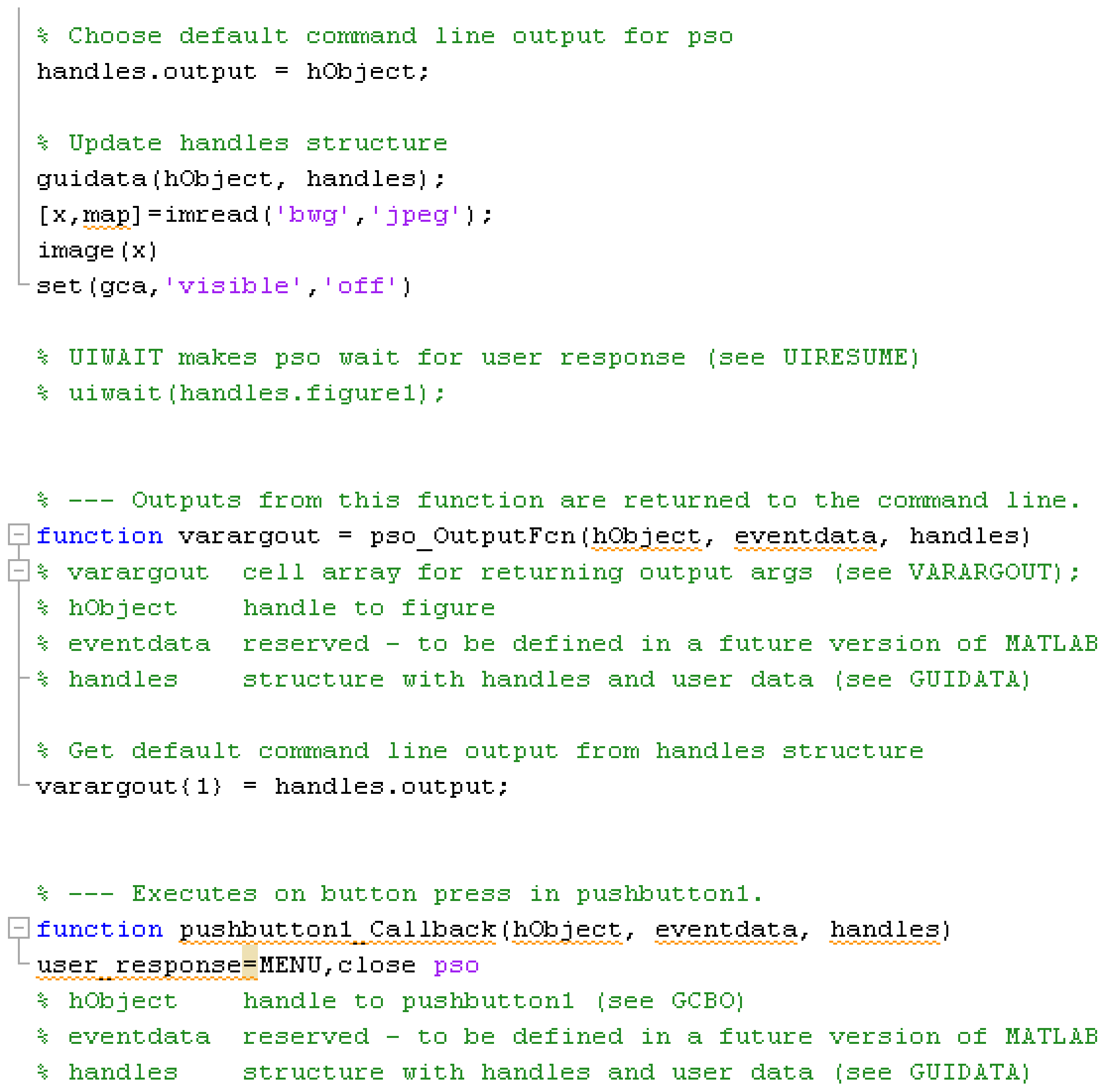


```
% Update handles structure
guidata(hObject, handles);
axes(handles.axes1)
guidata(hObject, handles);
[x,map]=imread('13676-layout46','jpeg');
image(x)
set(gca,'visible','off')

axes(handles.axes2)
guidata(hObject, handles);
[x,map]=imread('ump','jpeg');
image(x)
set(gca,'visible','off')

% UIWAIT makes MENU wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = MENU_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

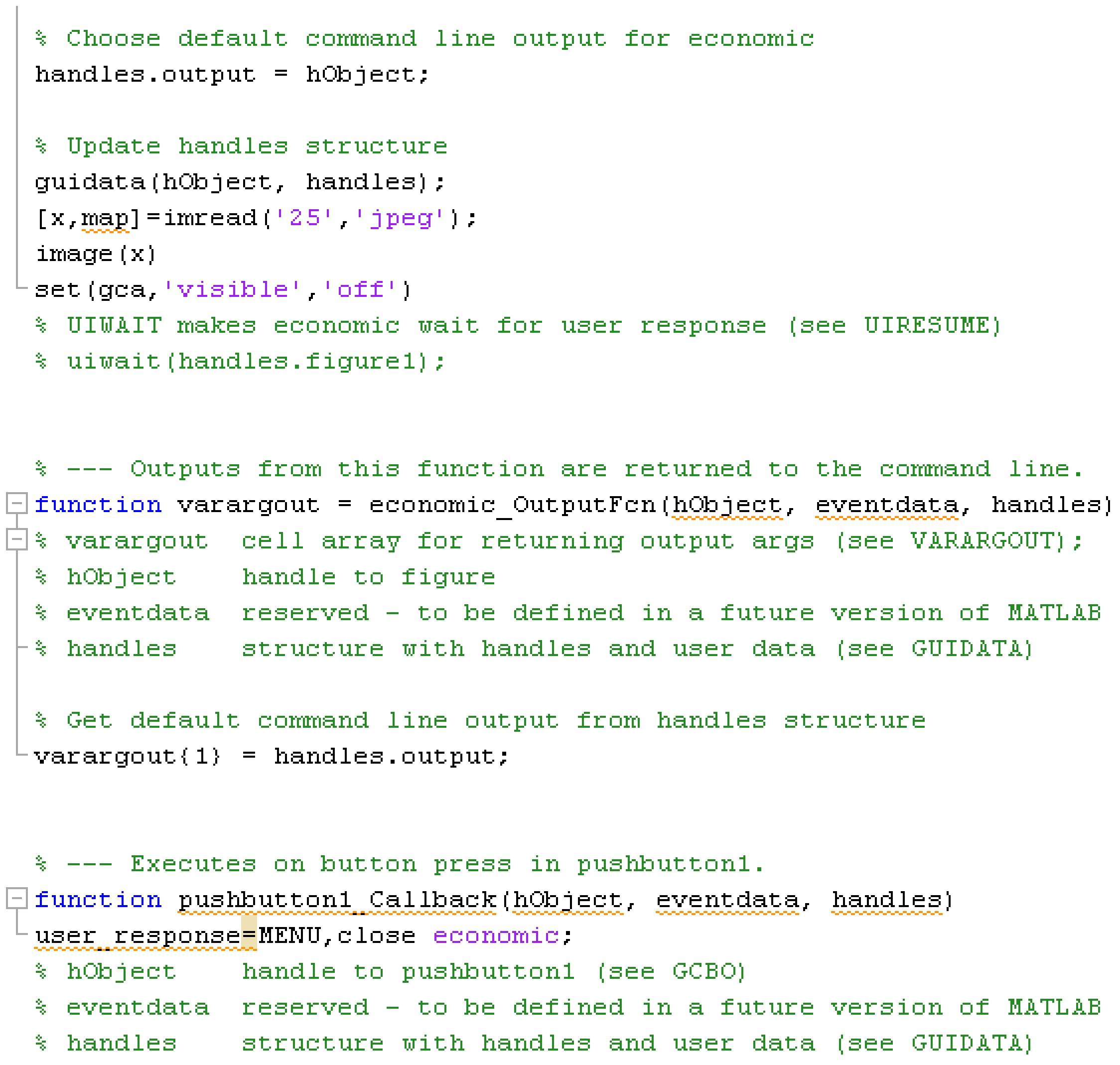


```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
user_response = pso,close MENU;


% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
user_response = economic,close MENU;

% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
user_response = TYPE,close MENU;
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
button=questdlg('Are you sure to exit?','Exit Box','Yes','No','No');
switch button
    case'Yes'
        close all
    case 'No'
        quite cancel;
end
```

GUI PROGRAMMING FOR pso

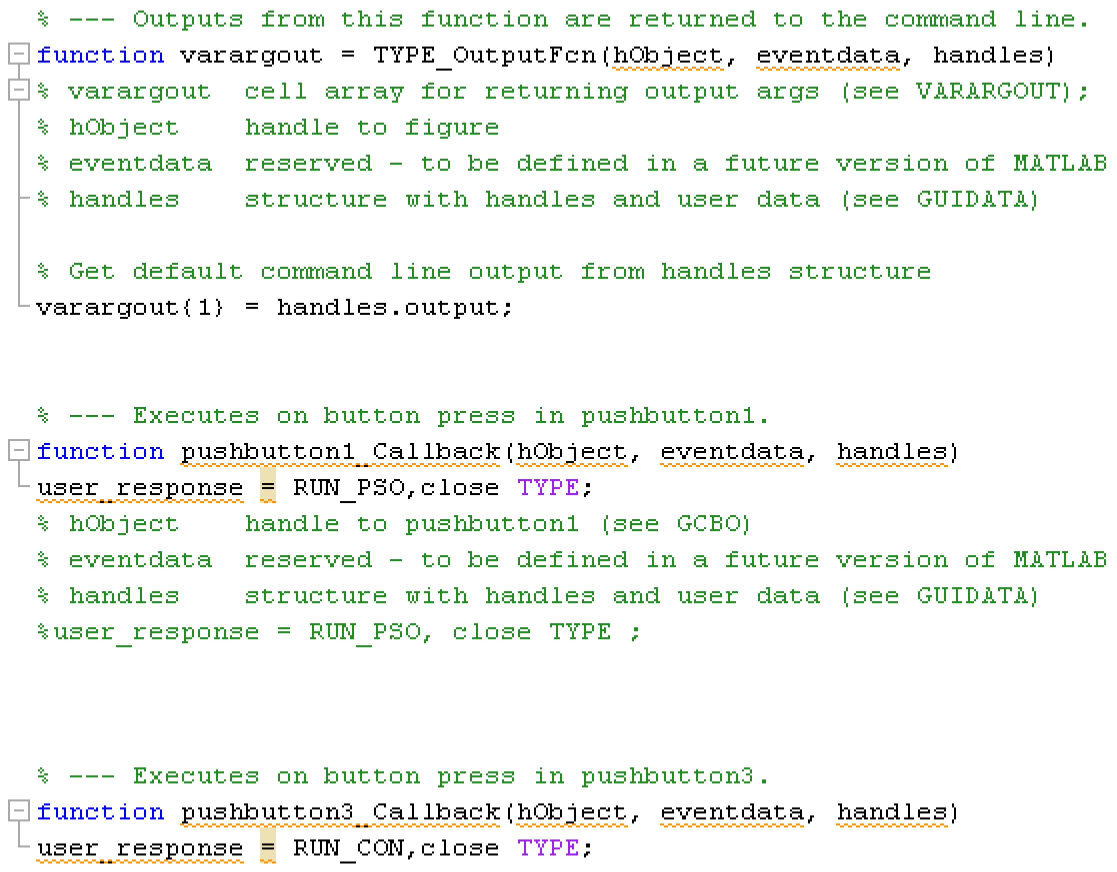


```
% Choose default command line output for pso
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('bwg','jpeg');
image(x)
set(gca,'visible','off')

% UIWAIT makes pso wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = pso_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
user_response=MENU,close pso
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```



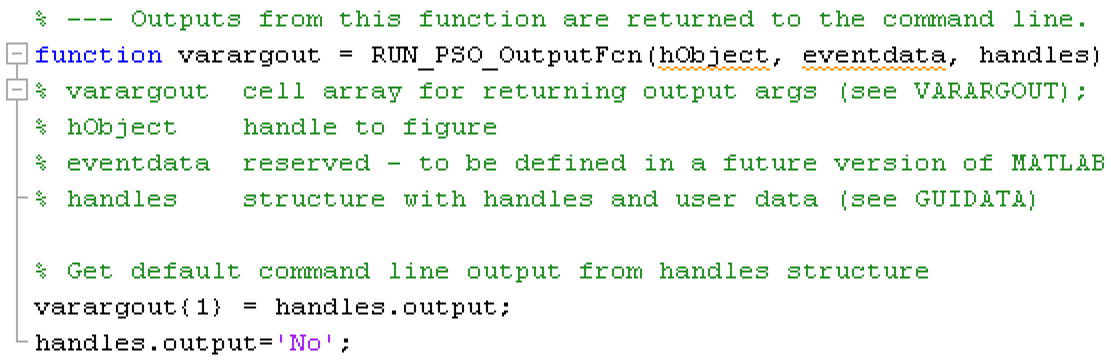
```
% Choose default command line output for economic
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
[x,map]=imread('25','jpeg');
image(x)
set(gca,'visible','off')
% UIWAIT makes economic wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = economic_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
user_response=MENU,close economic;
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

GUI PROGRAMMING FOR TYPE

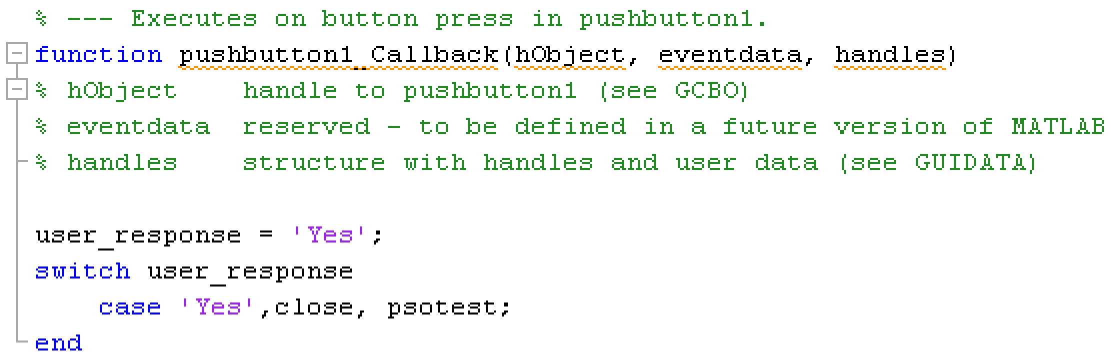


```
% --- Outputs from this function are returned to the command line.
function varargout = TYPE_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
user_response = RUN_PSO,close TYPE;
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%user_response = RUN_PSO, close TYPE ;



% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
user_response = RUN_CON,close TYPE;
```

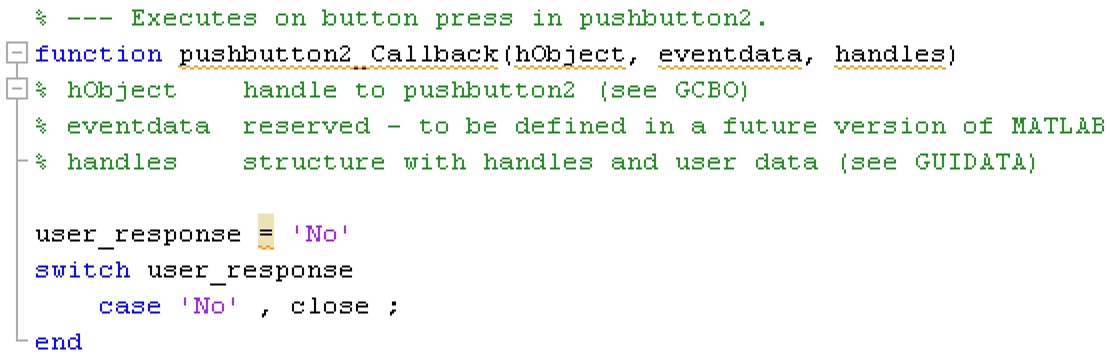
GUI PROGRAMMING FOR RUN_PSO



```
% --- Outputs from this function are returned to the command line.
function varargout = RUN_PSO_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
handles.output='No';
```



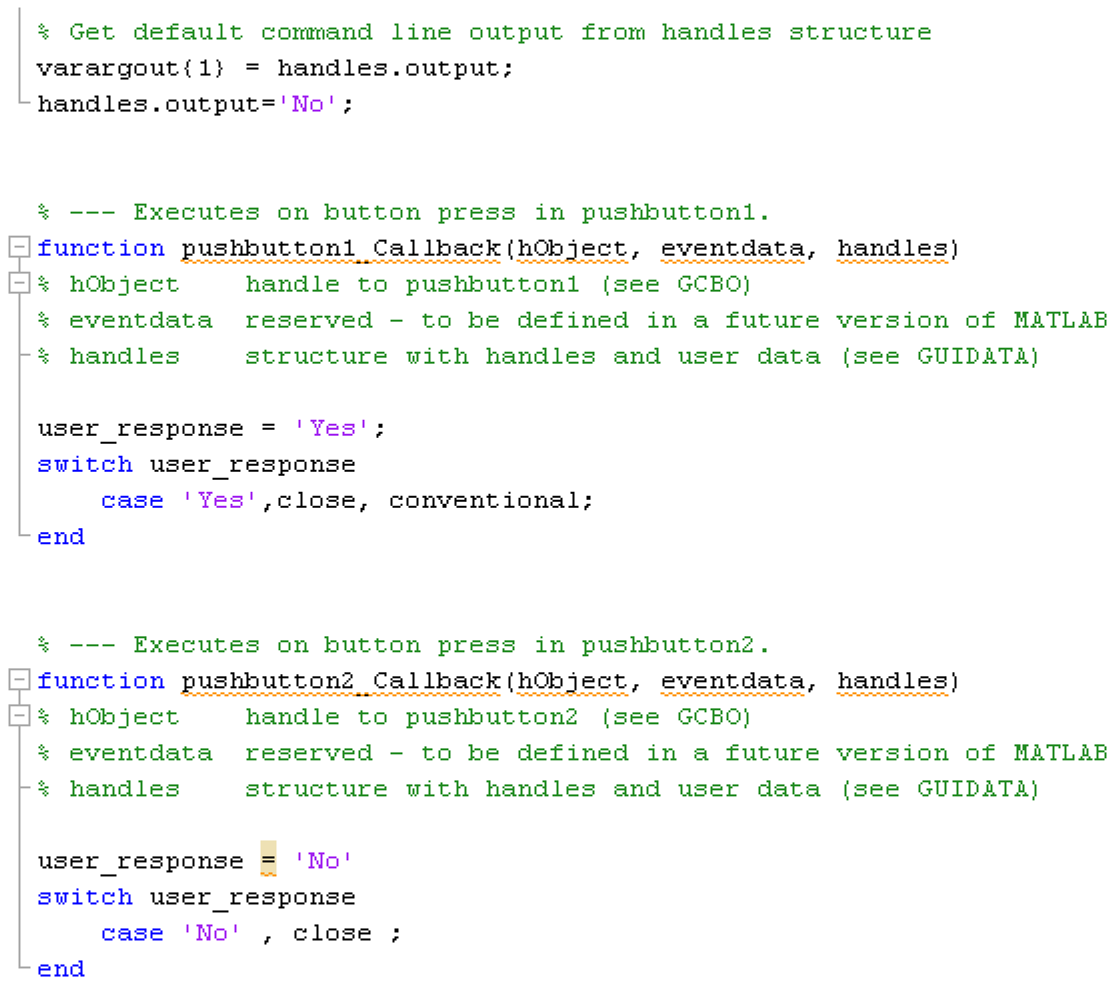
```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

user_response = 'Yes';
switch user_response
    case 'Yes',close, psotest;
end
```



```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

user_response = 'No'
switch user_response
    case 'No' , close ;
end
```

GUI PROGRAMMING FOR RUN_CON

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RUN_CON_OpeningFcn, ...
                   'gui_OutputFcn',  @RUN_CON_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before RUN_CON is made visible.
function RUN_CON_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to RUN_CON (see VARARGIN)
movegui ('center')
handles.output = 'Yes'

% Choose default command line output for RUN_CON
handles.output = hObject;
```



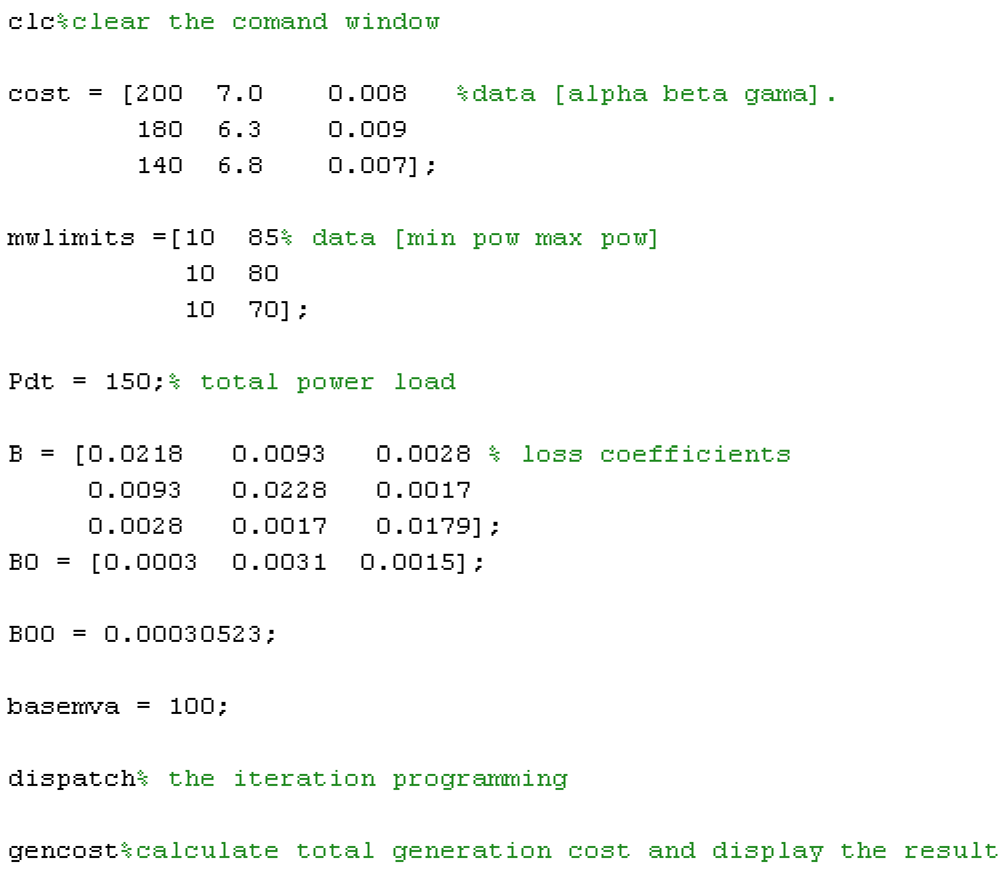
```
% Get default command line output from handles structure
varargout{1} = handles.output;
handles.output='No';


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

user_response = 'Yes';
switch user_response
    case 'Yes',close, conventional;
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

user_response = 'No'
switch user_response
    case 'No' , close ;
end
```

APPENDIX B

CONVENTIONAL

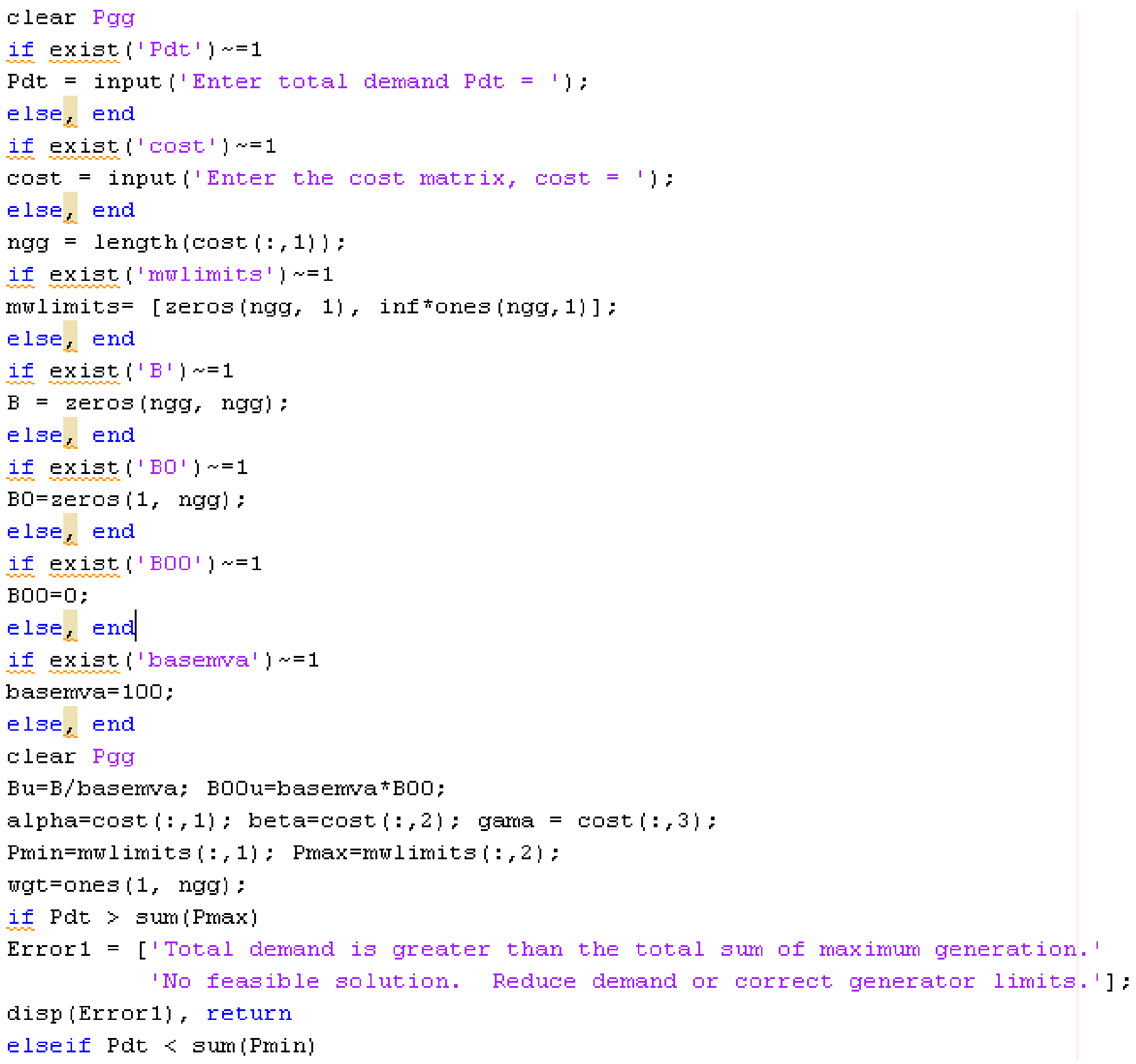
PROGRAMMING

PROGRAMMING FOR CONVENTIONAL

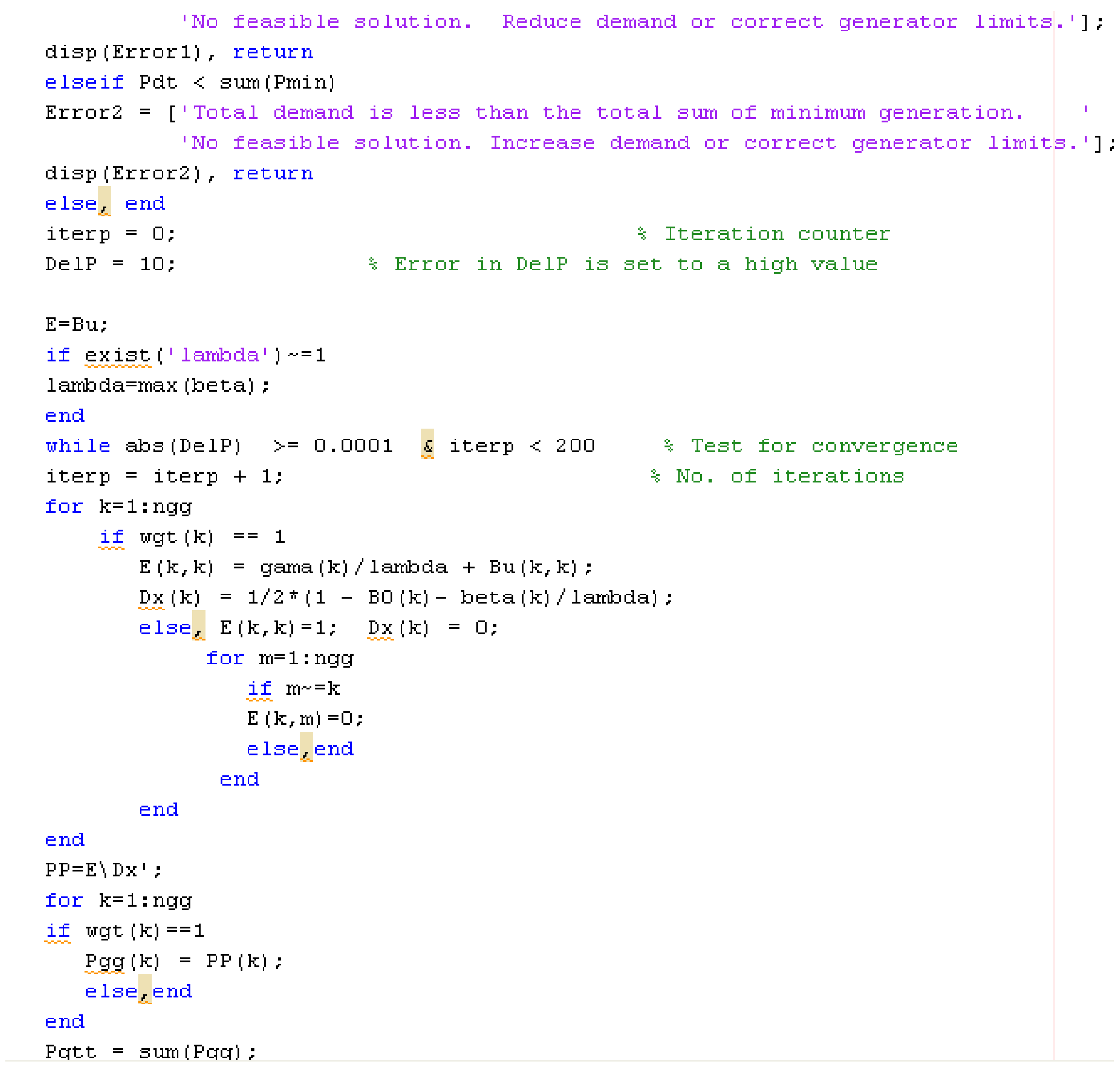


```
clc%clear the comand window

cost = [200  7.0    0.008   %data [alpha beta gama].
        180  6.3    0.009
        140  6.8    0.007];

mwlimits =[10  85% data [min pow max pow]
           10  80
           10  70];

Pdt = 150;% total power load

B = [0.0218   0.0093   0.0028 % loss coefficients
     0.0093   0.0228   0.0017
     0.0028   0.0017   0.0179];
B0 = [0.0003  0.0031  0.0015];

B00 = 0.00030523;

basemva = 100;

dispatch% the iteration programming

gencost%calculate total generation cost and display the result
```

PROGRAMMING FOR Dispatch



```
clear Pgg
if exist('Pdt')~=1
Pdt = input('Enter total demand Pdt = ');
else, end
if exist('cost')~=1
cost = input('Enter the cost matrix, cost = ');
else, end
ngg = length(cost(:,1));
if exist('mwlimits')~=1
mwlimits= [zeros(ngg, 1), inf*ones(ngg,1)];
else, end
if exist('B')~=1
B = zeros(ngg, ngg);
else, end
if exist('B0')~=1
B0=zeros(1, ngg);
else, end
if exist('B00')~=1
B00=0;
else, end
if exist('basemva')~=1
basemva=100;
else, end
clear Pgg
Bu=B/basemva; B00u=basemva*B00;
alpha=cost(:,1); beta=cost(:,2); gama = cost(:,3);
Pmin=mwlimits(:,1); Pmax=mwlimits(:,2);
wgt=ones(1, ngg);
if Pdt > sum(Pmax)
Error1 = ['Total demand is greater than the total sum of maximum generation.'
          'No feasible solution.  Reduce demand or correct generator limits.'];
disp(Error1), return
elseif Pdt < sum(Pmin)
```



```
          'No feasible solution.  Reduce demand or correct generator limits.'];
disp(Error1), return
elseif Pdt < sum(Pmin)
Error2 = ['Total demand is less than the total sum of minimum generation.    '
          'No feasible solution. Increase demand or correct generator limits.'];
disp(Error2), return
else, end
iterp = 0;                                    % Iteration counter
DelP = 10;              % Error in DelP is set to a high value

E=Bu;
if exist('lambda')~=1
lambda=max(beta);
end
while abs(DelP)  >= 0.0001  & iterp < 200     % Test for convergence
iterp = iterp + 1;                           % No. of iterations
for k=1:ngg
    if wgt(k) == 1
       E(k,k) = gama(k)/lambda + Bu(k,k);
       Dx(k) = 1/2*(1 - B0(k)- beta(k)/lambda);
       else, E(k,k)=1;  Dx(k) = 0;
            for m=1:ngg
               if m~=k
               E(k,m)=0;
               else,end
             end
       end
end
PP=E\Dx';
for k=1:ngg
if wgt(k)==1
   Pgg(k) = PP(k);
   else,end
end
Pgtt = sum(Pgg);
```

PROGRAMMING FOR Gencost

```
%function [totalcost]=gencost(Pgg, cost)
if exist('Pgg')~=1
Pgg=input('Enter the scheduled real power gen. in row matrix ');
else,end
if exist('cost')~=1
cost = input('Enter the cost function matrix ');
else, end
ngg = length(cost(:,1));
Pmt = [ones(1,ngg); Pgg; Pgg.^2];
for i = 1:ngg
costv(i) = cost(i,:)*Pmt(:,i);
end
totalcost=sum(costv);
fprintf('\nTotal generation cost = % 10.2f $/h \n', totalcost)
```

APPENDIX C

PSO

PROGRAMMING

PROGRAMMING FOR psotest

```
% the data matrix should have 5 columns of fuel cost coefficients and plant
limits.
% 1.a ($/MW^2) 2. b $/MW 3. c ($) 4.lower lomit(MW) 5.Upper limit(MW)
%no of rows denote the no of plants(n)
clear
clc
format long;

global data B B0 B00 Pd % this type of data can be used by other function
data=[0.008 7   200 10  85 %fuel cost coefficients and plant  limits
      0.009 6.3 180 10  80
      0.007 6.8 140 10 70];
B=.01*[.0218 .0093 .0028;.0093 .0228 .0017;.0028 .0017 .0179];% loss
coefficients
 B0=0*[.0003 .0031 .0015];
 B00=100*.00030523;
 Pd=150;% total power load
 Pd=Pd+B00;
l=data(:,4)';
u=data(:,5)';
ran=[l' u'];
n=length(data(:,1));
Pdef = [100 3000 50 2 2 0.9 0.4 1500 1e-6 5000 NaN 0 0];
 [OUT]=pso_Trelea_vectorized('f6',n,1,ran,0,Pdef);% PSO programming
 out=abs(OUT)
 P=out(1:n)
  [F Pl]=f6(P')
```

PROGRAMMING FOR pso_Trelea_vectorized

```
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
```

```
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);
```

```
 %   % 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


%PSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSOPSO
% 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
```

PROGRAMMING FOR gplotpso

```
clf
 set(gcf,'Position',[651    50   600   474]); % this is the computer
dependent part
 %set(gcf,'Position',[743    33   853   492]);
 set(gcf,'Doublebuffer','on');

% particle plot, upper right
 subplot('position',[.7,.5,.27,.4]);
 set(gcf,'color','k')

 plot3(pos(:,1),pos(:,D),out,'bx','Markersize',7)

 hold on
 plot3(pbest(:,1),pbest(:,D),pbestval,'g+','Markersize',12);
 plot3(gbest(1),gbest(D),gbestval,'r.','Markersize',25);

 % crosshairs
 offx = max(abs(min(min(pbest(:,1)),min(pos(:,1)))),...
            abs(max(max(pbest(:,1)),max(pos(:,1)))));

 offy = max(abs(min(min(pbest(:,D)),min(pos(:,D)))),...
            abs(min(max(pbest(:,D)),max(pos(:,D)))));
 plot3([gbest(1)-offx;gbest(1)+offx],...
       [gbest(D);gbest(D)],...
       [gbestval;gbestval],...
       'b-.');
 plot3([gbest(1);gbest(1)],...
       [gbest(D)-offy;gbest(D)+offy],...
       [gbestval;gbestval],...
       'b-.');

 hold off

 xlabel('Dimension 1','color','m')
 ylabel(['Dimension ',num2str(D)],'color','m')

 title('Particle Dynamics','color','r','fontweight','bold')

 set(gca,'Xcolor','m')
 set(gca,'Ycolor','m')

 set(gca,'color','m')

 % camera control
 view(2)
 try
   axis([gbest(1)-offx,gbest(1)+offx,gbest(D)-offy,gbest(D)+offy]);
 catch
   axis([VR(1,1),VR(1,2),VR(D,1),VR(D,2)]);
 end

% error plot, left side
```

```
 subplot('position',[0.1,0.1,.475,.830]);
  semilogy(tr(find(~isnan(tr))),'color','b','linewidth',2)
  %plot(tr(find(~isnan(tr))),'color','m','linewidth',2)
  xlabel('epoch','color','c')
  ylabel('gbest val.','color','c')

  if D==1
     titstr1=sprintf(['%11.6g = %s( [ %9.6g ] )'],...
                gbestval,strrep(functname,'_','\_'),gbest(1));
  elseif D==2
     titstr1=sprintf(['%11.6g = %s( [ %9.6g, %9.6g ] )'],...
                gbestval,strrep(functname,'_','\_'),gbest(1),gbest(2));
  elseif D==3
     titstr1=sprintf(['%11.6g = %s( [ %9.6g, %9.6g, %9.6g ] )'],...

gbestval,strrep(functname,'_','\_'),gbest(1),gbest(2),gbest(3));
  else
     titstr1=sprintf(['%11.6g = %s( [ %g inputs ] )'],...
                gbestval,strrep(functname,'_','\_'),D);
  end
 title(titstr1,'color','c','fontweight','bold');

  grid on
%  axis tight

  set(gca,'Xcolor','c')
  set(gca,'Ycolor','c')

  set(gca,'color','c')

  set(gca,'YMinorGrid','off')

% text box in lower right
% doing it this way so I can format each line any way I want
subplot('position',[.62,.1,.29,.4]);
  clear titstr


  legstr = {'Green(+)  = Personal Bests';...
            'Blue(x)   = Current Positions';...
            'Red(.)    = Global Best'};
  text(.1,0.025,legstr{1},'color','g');
  text(.1,-.05,legstr{2},'color','b');
  text(.1,-.125,legstr{3},'color','r');

  hold off

  set(gca,'color','w');
  set(gca,'visible','off');

  drawnow
```