ARTIFICIAL IMMUNE SYSTEMS FOR OPTIMIZATION, CLUSTERING AND CLASSIFICATION APPLICATIONS IN CHEMICAL SCIENCES AND ENGINEERING

By

AASHTI HAMID

Submitted in partial fulfilment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

CHEMICAL ENGINEERING: ADVANCED MODELING AND SIMULATION

of

Academy of Scientific and Innovative Research

NATIONAL CHEMICAL LABORATORY
(Council of Scientific and Industrial Research)
PUNE

JULY, 2012

DECLARATION

I hereby declare that the work being presented by me in this thesis entitled "**Artificial Immune Systems for optimization, clustering and classification applications in Chemical Sciences and Engineering**" by in partial fulfilment of requirements for the award of degree of **MASTER OF TECHNOLOGY** in **CHEMICAL ENGINEERING: ADVANCED MODELING AND SIMULATION** is an authentic record of my own work carried out during January 2012 to July 2013 under the supervision of "**Dr. Sanjeev S. Tambe"** at CSIR-NCL, Pune.

The matter presented in this thesis has not been submitted to any other university/institute for the award of any degree.

_____

Aashti Hamid

This M Tech thesis submitted by (name of the candidate) is a record of work carried out under my guidance and may be accepted.

_____

Dr. Sanjeev S. Tambe

ACKNOWLEDGEMENTS

I take the opportunity to express my reverence to my supervisor Dr. Sanjeev S. Tambe (Principal Scientist, CEPD, NCL, Pune) for his guidance, inspiration and innovative technical discussions during the course of this work. He is not only a great teacher with deep vision but also a very kind person. His trust and support inspired me for taking right decisions and I am glad to work with him.

I thank all my teachers together with Dr. Chetan Gadgil and Dr. Ashish Orpe, co coordinators for their contribution in my studies and research work. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I am grateful and indebted to my lab mates and colleagues for infallible motivation and moral support.

Last but not the least I want to thank my parents and sisters for their immense support, love and inspiration which is instrumental in completing this task.

ABSTRACT

THIS *study presents a highly robust and stochastic nonlinear artificial intelligence based optimization technology for chemical engineering process optimization, which is inspired from human biological immune system called as artificial immune systems (AIS). Specifically, clonal selection algorithm (CLONAG) has been used for the development of unconstrained and constrained optimization methods. Further, process modeling and optimization of process operating variables and parameters of a batch reactor process for the abatement of a common pollutant, namely H_2S, to elemental sulfur using Fe^{3+}-malic acid chelate (Fe^{3+}-MA) catalyst has been conducted using artificial intelligence based hybrid strategies involving artificial neural networks (ANN) and artificial immune systems (AIS) and artificial neural networks (ANN) and genetic algorithms (GA). Prior to the modeling the process variables and parameters that significantly influence H_2S conversion were determined by conducting Sensitivity Analysis. The hybrid strategies first use an ANN to develop a data-driven model predicting H_2S conversion followed by the AIS formalism which optimizes inputs of the ANN model with a view to maximize the H_2S conversion. The AIS-optimized process conditions leading to high (≈ 97%) H_2S conversion were tested experimentally, and the results obtained thereby show an excellent match with the optimized conditions. Also, AIS was proved to give optimized results in lesser number of generations and function evaluations to reach convergence as compared to the GA-optimized results.*

Keywords: Batch reactor, artificial immune systems, artificial neural networks, clonal selection algorithm, genetic algorithms, sensitivity analysis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

$C_{Fe,0}$ Initial Fe^{3+}-MA concentration (ppm)

$C_{H2S,0}$ H_2S gas concentration (ppm)

M Memory antibodies in the initial population

N_c Number of clones produced by each antibody set

p Mutation operator

P Reactor Pressure (bar)

r Replaceable antibodies having poor affinity in the initial population

S Stirring Speed of Impeller (rpm)

T Reactor Temperature (^oC)

t Reaction time (min)

V_B Volume of buffer solution (cm^3)

V_G Volume of gas (cm^3)

X_{rate} Selection Rate

y Fitness scores for the corresponding candidate solutions

Greek Letters

η Learning Rate ($0 < \eta \leq 1$)

α Momentum Coefficient ($0 < \alpha \leq 1$)

β Cloning Factor (0 to 1)

1. INTRODUCTION

1.1 Process optimization

OPTIMIZATION is the process of making something optimal. It consists of determining the values of process inputs (operating variables & parameters) to be used in the process operation to obtain the optimal process performance. Some optimization goals in process engineering are: (i) maximization of conversion and or product yield, (ii) maximization of product selectivity, (iii) maximization of process profit, (iv) minimization of operating loss, (v) minimization of production cost and (vi) minimization of selectivity of the undesirable products.

Generalized optimization problem statement is given as:

$$\textit{Maximize/Minimize: } f = (X, W) \tag{1.1}$$

Subject to constraints…$C_1, C_2, \ldots, C_n$

Where, $X = \{\mathbf{x}_i\}$, $i = 1, 2, \ldots, n$; n-dimensional set of *decision variables* (to be optimized), f is the *objective function* and $W = \{w_j\}$, $j = 1, 2, \ldots, u$; u-dimensional set of *objective function parameters*.

The two methods of optimization are classical deterministic methods (e.g. gradient descent & its variants) and stochastic methods which include evolutionary techniques, genetic algorithms (GA), memetic algorithm (MA), ant colony optimization (ACO), particle swarm (PS), deterministic evolution (DE) and artificial immune systems (AIS).

Conventionally, chemical plant design consists of choosing and sizing appropriate process equipment, as well as fixing the nominal operating points. In this endeavor, deterministic gradient-based optimization techniques are utilized. The characteristics of them are as follows:

1. Computation of gradient of the objective function is an integral feature of these optimization methodologies.
2. Most gradient based optimization methods do not guarantee convergence to global optima.

3. Most of the deterministic methods require objective function to be smooth, continuous and differentiable.

The features of stochastic optimization methods are:

1. Random search of solution space.
2. Do not require gradient.
3. Good at finding deepest local optima.
4. And, are not constraint by the continuity, differentiability and smoothness of the objective function.

Due to these features stochastic approaches have found tremendous applications over last two decades in engineering design and plant operation.

The objective of this thesis is to develop artificial intelligence (AI) based optimization technologies for chemical process optimization. Specifically clonal selection algorithm (CLONALG) has been used for optimization of the batch reactor process for the abatement of H_2S gas.

1.2 Process modeling

Process optimization requires a mathematical model of the process. There are two types of models, phenomenological and empirical.

Phenomenological models can be constructed from the knowledge of mass, momentum, and energy balances, as well as from other chemical engineering principles. But due to the lack of a good understanding of the underlying physicochemical phenomena, development of phenomenological process models poses considerable difficulties. Moreover, nonlinear behavior being a common feature of chemical processes, it leads to complex nonlinear models, which in most cases are not amenable to analytical solutions; thus, computationally intensive numerical methods must be utilized for obtaining solutions. The difficulties associated with the construction and solutions of the phenomenological models necessitate exploration of alternative modeling formalisms.

Process modeling via empirical models (regression) is one such alternative. These linear models and their nonlinear counterparts are constructed exclusively from the input-output process data. A fundamental deficiency of the empirical modeling approach is that the model structure (form) must be specified a priori. Satisfying this requirement, especially for nonlinearly behaving processes is a cumbersome task, since it involves selecting heuristically an appropriate model structure from numerous alternatives.

In recent years, support vector machines which is a statistical learning theory based machine learning formalism has gained popularity due to its many attractive features and promising empirical performance. The salient features of SVR are:

1. It is an exclusively data-based nonlinear modelling paradigm which is based on the principle of structural risk minimization (SRM) and thus has a greater potential to generalize.
2. Parameters of an SVR model are obtained by solving a quadratic optimization problem.
3. The objective function in SVR being of quadratic form possesses a single minimum thus avoiding the heuristic procedure involved in locating the global or the deepest local minimum on the error surface.
4. And, in SVR, the inputs are first nonlinearly mapped into a high-dimensional feature space wherein they are correlated linearly with the output.

However, the biggest limitation of the support vector machines lie in choice of the kernel and its function parameters. Once the kernel is fixed, SVR classifiers have only one user- chosen parameter (the error penalty). Secondly, training for very large datasets (millions of support vectors) due to the heavy quadratic programming, is still an unsolved problem. Hence, a high algorithmic complexity and extensive memory is required.

Therefore, artificial neural networks (ANN) have been used for the process modeling because:

1. It can be constructed solely from the historic process input–output data (example set).
2. Detailed knowledge of the process phenomenology is unnecessary for the model development unlike phenomenological and empirical modelling methods.
3. A properly trained model possesses excellent generalization ability owing to which it can accurately predict outputs for a new input data set.

4. And even very large data sets having multiple input–multiple output (MIMO) nonlinear relationships can be approximated simultaneously and easily unlike support vector regression.

1.3 Need for a stochastic optimization algorithm

Whether it is ANN, SVR or genetic programming (GP), these are exclusively data driven models which guarantee differentiability and continuity of the objective function but not its smoothness. This is why we cannot use deterministic methods and for these cases stochastic methods are better suited. Accordingly, we have developed an ANN based model and the inputs of the ANN model are being optimized so as to improve the process performance.

The thesis is structured as follows: the performance of ANN- AIS based methodology is compared with ANN- GA based methodology.

1.4 Artificial Intelligence based Modeling and Optimization Strategies

1.4.1 Artificial Neural Networks

In the last two decades, an artificial intelligence paradigm viz. *artificial neural networks* (ANNs) has emerged as an attractive tool for developing non-linear empirical models when the development of phenomenological or conventional empirical models either becomes impractical or cumbersome. The most widely employed ANN paradigm is the *multi-layered perceptron* (MLP), which approximates non-linear relationships existing between multiple "causal (*input/prediction*)" process operating variables and parameters, and the corresponding "dependent (*response/output*)" variables (Nandi et al., 2001). Once an ANN-based process model with good generalization performance is constructed, its input space representing process operating variables and parameters can be optimized with a view to improve the process performance. Being exclusively data based, the ANN models require example data which are noise-free and statistically well-distributed. It may also be noted that the ANN-based models are not good at extrapolation. This weakness, which is shared by all of the data-based nonlinear models, can be overcome by collecting more example data in the regions where extrapolation is desired (Nandi et al., 2002).

1.4.2 Artificial Immune Systems

The vertebrate immune system, which defends our body from foreign substances, is one of the most complex and elaborate bodily systems. Its complexity is in fact, compared to that of the brain. With the advances in the technology, the curiosity about how the immune system functions increased very rapidly (de Castro and Timmis, 2002). This led to its study about including the development of mathematical models based on several of its main operative mechanisms. Similar to the study of the nervous system that led to the emergence of artificial neural networks (ANN), the study of immune system has lately inspired the development of artificial immune systems (AIS) as a novel computational/ artificial intelligence (CI/AI) paradigm (de Castro and Timmis, 2002).

AIS contains several methods based on the different theories of immune system like negative selection theory (*Negative Selection algorithm*, Forrest et al., 1994), clonal selection theory (*Clonal Selection algorithm*, De Castro et al. 2001), immune network theory (*opt-Ainet*, Farmer et al., 1986) and danger cell theory (Matzinger, 1994, 2001).

The important applications of artificial immune system include solving of non-linear multi-modal optimization problems (Khaled et al., 2010, Aragon et al., 2010 and Kilic et al., 2010), pattern recognition and classification (Zhong et al., 2008, Taylor et al., 2010 and Adraine et al., 2008), fault detection (Luther et al., 2007, Dasgupta et al., 2003) in the areas of Engineering and Science.

1.4.3 Genetic Algorithms

Genetic algorithms (GAs) (Goldburg, 1989; Davis, 1991) are artificial intelligence based stochastic non-linear optimization formalisms and have been used with a great success in solving problems involving very large search spaces (Ramanathan et al., 2001; Sumanwar et al., 2002). The GAs are based on the Darwinian principles of “survival-of-the-fittest” and “random exchange of memory during genetic propagation”, followed by biologically evolving species.

The Genetic algorithm is structured in the following way i.e., *Initialization* (Population of candidate Solutions or chromosomes), *Evaluation of Fitness Function*, *Selection of best solutions based on the fitness scores*, *Cross Over* and *Mutation* are performed on the parent solutions to generate new set of solutions which poses similar characteristics of parents solutions, and finally

evaluating the fitness scores of newly generated solutions and repeating the process till the termination criterion is satisfied. It can handle complex non-linear unconstrained and constrained optimization problems effectively, and can able to attain the solutions close to global optimum. Algorithm is easy to implement and no gradient information is required.

It has found several applications in Bio informatics, Phylogenetics, Computational Science, Engineering, Economics, Chemistry, Mathematics, Physics and other fields. Some of the applications in chemical engineering include PID controller tuning (Slavov et al., 2012), optimal control problems (Zhang et al., 2010) and optimization of distillation sequences (Özçelik, 2011).

1.5 Why AIS is chosen over other Evolutionary Techniques

The immune system is highly distributed, highly adaptive, self organizing, maintains a memory of past encounters and has the ability to continually learn about the new encounters. AIS are the systems developed around the current understanding of the immune system. We chose to study this method for optimization as compare to other evolutionary techniques namely, particle swarm, ant colony optimization, memetic algorithms and genetic algorithms because it is still an emerging field and has not been widely explored for solving optimization problems in chemical engineering and technology. Moreover, it may be described as somewhat of a cross between genetic algorithms (for optimization) and neural networks (for classification). AIS models based on immune networks resemble the structures and interactions of connectionist (neural network) models and population, genotype, phenotype mapping, and proliferation of the most fit like genetic algorithm.

An advantage of AIS over Particle Swarm Optimization (PSO) is that in PSO choice of parameters has a large impact on the optimization performance and selecting those PSO parameters which yield a good performance has therefore been the subject of much research up till now. In Ant colony Optimization (ACO) unlike AIS theoretical analysis is very difficult. Also in ACO time required to converge to the optimal solution is uncertain. According, using it to solve complex optimization problem was not cost effective.

Memetic algorithms (MA) are very promising for multi-objective optimization problems but their use in multi-objective optimization problems is not straight- forward (a discrete search

space is needed). Moreover, in order to improve its performance a considerably higher memory expense may be required.

1.6 Thesis Outline

Chapter 1 gives a brief introduction about the process modelling and process optimization techniques with their advantages and limitations. It also contains a short explanation about the artificial intelligence based hybrid strategies i.e. artificial neural networks-artificial immune systems (ANN-AIS) and artificial neural networks-genetic algorithms (ANN- GA), developed in this work and the need to use them.

Chapter 2 provides extensive literature survey on the artificial neural networks, artificial immune systems and genetic algorithms.

Chapter 3 presents in detail the work that we have carried out, it is given in the following steps:

1. Design and Development of Clonal Selection Algorithm (CLONALG) for unconstrained and constrained optimization using MATLAB.
2. Verification of the codes developed by solving 5 benchmark nonlinear unconstrained and 1 nonlinear constrained optimization problem.
3. Selection of the most influential process operating variables and parameters through sensitivity analysis for the optimization of the batch reactor process for the abatement of H_2S gas.
4. Development of ANN model using the influential process variables and parameters as model inputs and H_2S conversion (%) as the model output.
5. Optimization of the input space of the ANN model the unconstrained CLONALG with a view of maximizing the H_2S conversion to elemental sulphur.
6. Development of genetic algorithms code for unconstrained optimization using MATLAB.
7. Optimization the input space of the ANN model using GA, so as to maximize the H_2S conversion to elemental sulphur.

In Chapter 4 the results pertaining to the batch reactor process for the abatement of H_2S gas are presented and discussed.

This report is finally concluded in Chapter 5 along with the final remarks and future directions.

2. LITERATURE SURVEY

ARTIFICIAL INTELLIGENCE (termed by McCarthy, J., 1955) is a branch of computer science and engineering that deals with intelligent behavior, learning, and adaptation in machines. It is based on the concept that computers can be programmed to assume capabilities such as learning, reasoning, adaptation, and self-correction. Information processing in the case of artificial intelligence is by mimicking or simulation of the cerebral, nervous or cognitive processes.

One such type of process model where AI comes in is black box modeling. It is an exclusively data driven modeling method which uses a generic nonlinear function for data fitting thus obviating the need to pre-specify the data-fitting model like empirical or phenomenological modeling methods. Its development is relatively easy and it does require any knowledge of reaction mechanism, kinetics, heat and mass transfer processes and related parameter values.

2.1 What is Artificial Neural Networks

Derived from their biological counterparts, artificial neural networks (ANNs) are based on the concept that a highly interconnected system of simple processing elements can learn complex interrelationships between independent and dependent variables. ANNs offer an attractive approach to black box modeling of highly complex, non- linear systems having very large number of inputs and outputs (Nandi et al., 2002). ANNs are massively connected parallel structures containing processing elements called *neurons*.

In the fig. 2.1, three layers are shown, an *input layer*, an *output layer* and an intermediate layer called as *hidden layer*. The number of neurons and the number of hidden layers are selected heuristically, but the hidden layers cannot be greater than three. Each neuron is connected to all the neurons in the next layer by means of a "*connection weight*". The outputs from neurons can be calculated by suitable "*transform equations*" provided the inputs and the connection weights are known.

The problem of neural network modeling is to find a set of weights such that the prediction error is sufficiently small. The procedure is to assume a set of weights initially, compute the outputs and the prediction error, and then adjust the weights according to an *error minimization technique*, such as the gradient descent method, until the prediction error falls to an acceptable level. The activity of finding optimal weights is called *network training*. Once the network is trained, the black- box model is ready, and may be used to predict outputs for a new set of inputs, not originally part of those used in training. An advantage of neural network approach to modeling is that the structures of the model need not to be known. Also, multivariable problem can be handled with ease (Tambe et al., 1996).

Owing to their several attractive characteristics, ANNs have been widely used in chemical engineering applications such as steady state and dynamic process modeling, process identification, yield maximization, nonlinear control, and fault detection and diagnosis.

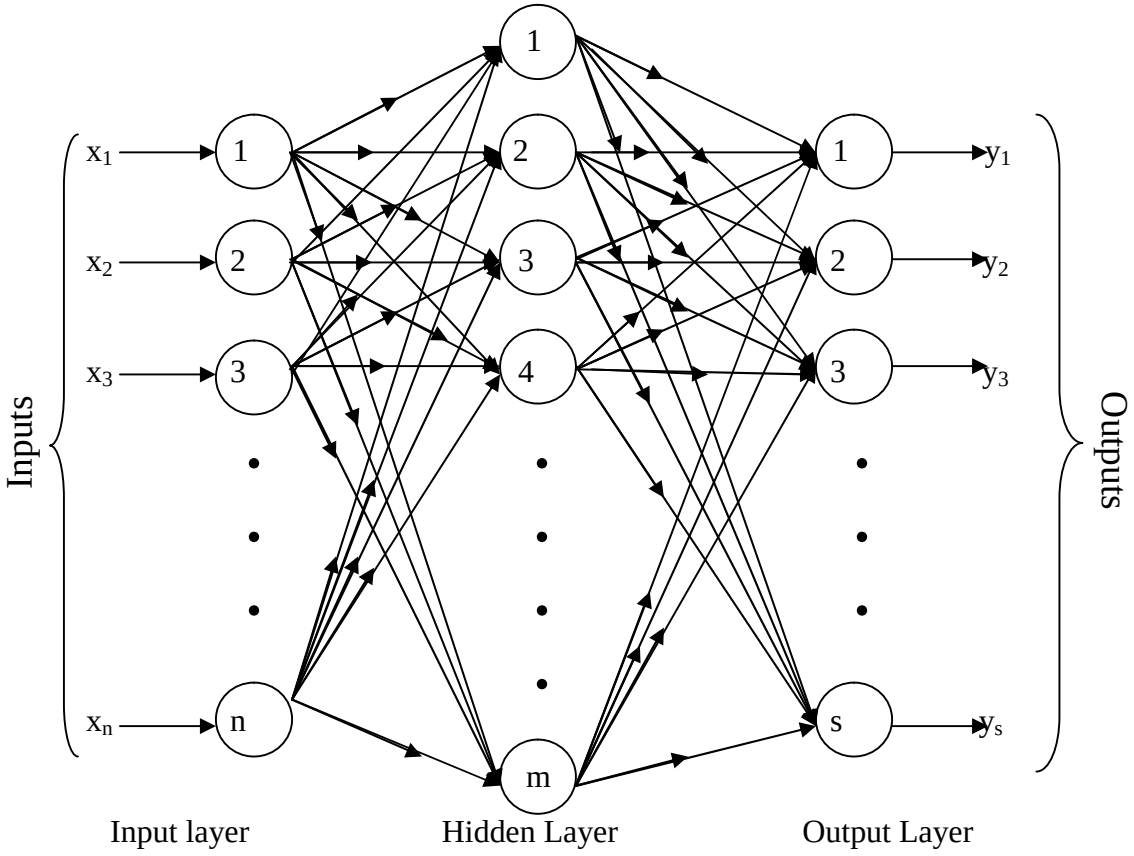


Fig.2. 1 Schematic of common Artificial Neural Network

A formal procedure for modeling with neural networks may be outlined as:

2.1.1 Selection of Inputs

The selection of suitable inputs has a direct bearing on the fidelity of the model. All inputs that have an influence on the output(s) must be included. The number of weights in the network depends directly on the number of inputs. As the number of inputs increases, the number of connection weights increases rapidly, the volume of training data required grows exponentially.

The input dimensionality can be reduced by finding combinations of inputs variables (usually non- linear) known as *features* (Maier et al., 2011). Another approach to reduce dimensionality is to evaluate the statistical properties of the data using *principal component analysis* (PCA)/ *partial least squares* (PLS). The simplest approach to reduce dimensionality is to determine the relative importance of different inputs on the output(s), and omit the least significant ones (Tambe et al., 1996) like *Sensitivity Analysis* (SA) method.

2.1.1.1 Sensitivity Analysis

SA involves varying each input variable across all the data points one by one and holding all the other variables constant at their minimum, 1st quartile, median, ¾th and at their maximum values (Lek et al., 1996). The median predicted response value across the 5 summary statistics is calculated and the relative importance of each variable is illustrated by the magnitude of its range of predicted response values (maximum- minimum) (Olden et al., 2004).

2.1.2 Selection of Outputs

The definition of the problem at hand usually determines what the system outputs are (Tambe et al., 1996).

2.1.3 Preprocessing of Data

The inputs and outputs may represent very different physical variables. For instance, one input may be in the range of 0 to 1 (e.g., conversion) while another may be of the order of 10^4 (activation energy). Such an order of magnitude difference between the two inputs or outputs causes difficulties like numerical overflows, during the network training. This problem may be overcome by scaling the input- output data (Nguyen et al., 2004).

Finally, a thorough examination of input- output data is necessary to identify and examine outliers which would otherwise corrupt the model fidelity (Tambe et al., 1996).

2.1.4 Selecting the network architecture and topology

Fig. 2.1 illustrates one particular neural network model. In addition to model selection, several other details would have to be considered. They include (Tambe et al., 1996):

1. Selecting the number of hidden layers and number of neurons in each hidden layer,
2. Volume of the training data
3. And, training algorithm

2.2 History and Development of Neural Networks

Although often considered a relatively new phenomenon initial attempts at artificial neural networks date back to the early 20th century. In 1943, Warren McCulloch and Walter Pitts co-authored a paper outlining a model of a simple neural network with electronic circuits. Later, as computers emerged in the 1950s, several researchers attempted to utilize the new technology to create better neural networks. Over the next decade Frank Rosenblatt, a neurobiologist at Cornell intrigued by the neural processing occurring within the eye of the fly formed the basis of his "*Perceptron*" neural network in 1958. The Perceptron and other models showed great promise with many initial successes. Unfortunately, this success of artificial neural networks caused a great deal of hype within the media and eventually led to disappointment as earlier claims were left unfulfilled. In part this was due to the very limited computing power available at the time. However, there were also conceptual limitations to progress.

Although funding was minimal several scientists continued to develop neural network models. Paul Werbos in 1975 worked to improve the earlier Perceptron model and created the now popular back-propagation network (though, it wasn't until it was rediscovered in 1986 by Rumelhart and McClelland that it became widely used). Other researchers such as Steve Grossberg, Teuvo Kohonen, and Henry Klopf also created new models. However, it was not until John Hopfield of Caltech presented a paper to the national Academy of Sciences that general interest in the field began to resurge. In 1985 the American Institute of Physics hosted the first annual meeting on Neural Networks for Computing and by 1987 the Institute of

Electrical and Electronic Engineers (IEEE) first International Conference on Neural Networks drew more than a thousand attendees. This interest has more or less continued through to the present day.

2.3 From Biological Neuron to Artificial Neural Networks

The human brain has a very complex and powerful architecture. It consists of approximately 10^{11} to 10^{14} nerve cells called as *neurons* (Tambe et al., 1996).

A neuron (see fig., 2.2) is an electrically excitable cell that carries out essentially two main functions: (i) addition of electrical incoming signals and producing chemical signals i.e. an output also known as *cell firing*, when the sum exceeds a threshold value. Thus the neuron behaves in an *all-or-nothing* manner indicating that it possesses two states, 'on' or 'off'.

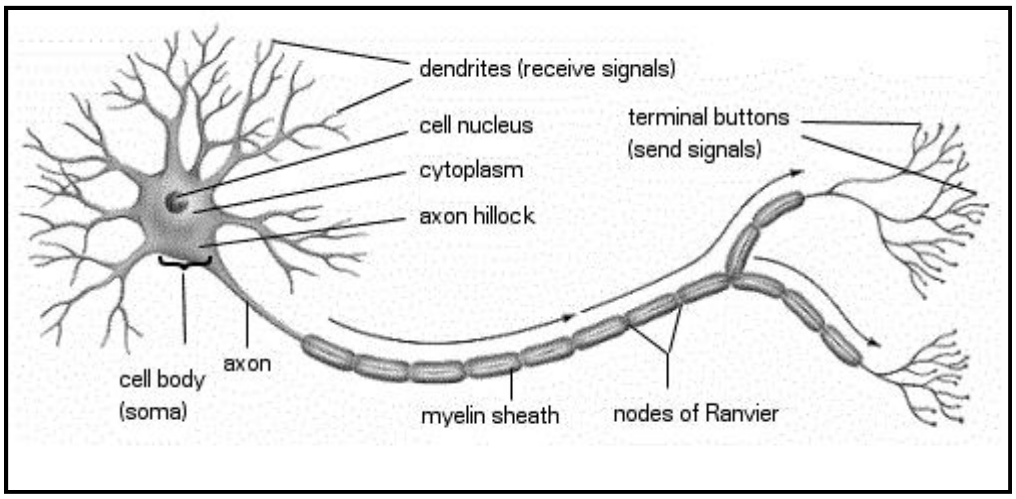


Fig.2. 2 Diagram of a typical nerve cell (neuron)

A typical neuron is divided into three parts: the soma or cell body, dendrites, and axon. The soma is usually compact; the axon and dendrites are filaments that extrude from it. Dendrites typically branch profusely, getting thinner with each branching, and extending their farthest branches a few hundred micro-meters from the soma. The axon leaves the soma at a swelling called the axon hillock, and can extend for great distances, giving rise to hundreds of branches. Unlike dendrites, an axon usually maintains the same diameter as it extends. Synaptic signals from other neurons are received by the soma and dendrites; signals to other neurons are transmitted by the axon. A typical synapse, then, is a contact between the axon of one neuron and

a dendrite or soma of another. Synaptic signals may be excitatory or inhibitory (Nowakowski, 2006).

An ANN can be defined mathematically as a parallel distributed structure having the following features:

1. The artificial analogue of biological neuron is variously referred to as a *node, processing element* (PE), *processing unit*, or simply *neuron*.
2. Similar to the biological neuron, all the information processing occurs at the nodes.
3. A real valued (positive or negative) weight is associated with each *connection*, which is the link between two nodes.
4. Also, the processing element can have multiple inputs, but only a single output which may be branched into several connections; however all the branched connections carry the same signal equal to the PE's output.
5. Each processing element using either activation and transfer function, or an activation function alone to compute its state (output) as a function of: (i) the bias, (ii) the weights of incoming connections, and (iii) the input signals. While the activation function computes the activation level of the processing element, the transfer function (generally non- linear) transforms the activation level to produce the output of the processing element. The transfer function also restricts the PE's output to a finite range.
6. The network acquires knowledge through a learning involving modification of connection weights in a systematic manner.
7. The acquired knowledge is stored in the form of weights.

Table 2. 1 Correspondence between Biological and Artificial Neuron

S. No.	Biological Terminology	Neural Network Terminology
1.	Neuron	Node/ processing element (PE), processing unit/ neuron.
2.	Synapse	Connection/ Edge/ Link
3.	Synaptic efficiency	Connection strength or weight
4.	Firing Frequency	Node output

2.4 Building Blocks of ANNs

2.4.1 Processing Elements (PEs)

The processing elements are the fundamental constituents of ANN. They perform simple mathematical operations on the information (data) received from their input connections with other PEs and pass on to the PEs located in the same or different layers. These simple manipulations consist of computing the overall (net) activation followed by its linear or non-linear transformation.

2.4.2 Connections

A processing element has two types of connections namely, *input* and *output*. It receives data from the input connection and after processing transmits the computed data via output connection. The connection types are further classified as *excitatory* or *inhibitory*, according to their resultant actions. The excitatory connection carries a positive signal, and enhances the activation level of the destination node. An inhibitory connection has a negative sign, and it reduces the destination PE's activation level. Also, the connections are classified according to their connectivity patterns. Such connection schemes are termed *intra-layer*, *inter-layer* and *recurrent*. The term intra- layer is synonymous with lateral, and suggests that PEs in the same layer are interconnected. Inter- layer connections indicate that PEs in two successive layers are linked. Lastly, the connections that originate from and terminate into the same PE are known as recurrent type. In this case, the output of a processing element serves as an additional input to the same PE. The inter-layer connections are further characterized as *feed forward* or *feedback* depending upon the direction of the information flow.

2.4.3 Weights

With each network connection is associated its strength – a real positive or negative quantity known as *connection strength* or *weight*. Weights are freely adjustable parameters for any ANN model since their number and magnitudes determine what the network represents. In the networks that carry out pattern recognition (classification), the pattern information is stored in the form of weights. For mapping networks, the knowledge of interrelationships among the input- output variables is encoded into weights. In order that a network performs the intended

task, its weight parameters must be properly chosen. This is achieved using a learning rule that defines a procedure for adapting the network weights.

Different learning rules are:

1. Hebb's Rule
2. Error Dependent Learning
3. Competitive Learning
4. Grossberg Learning

2.4.4 Activation and Transfer Functions

A processing unit in the network's active layer essentially performs three numerical operations. First, it combines all the input signals to compute the net input. It is then transformed into the net activation level (or simply activation) using an activation function. Lastly, the net activation is operated upon using a transfer function to yield the node's output. Since the action of transfer function produces the output of an active node, it is also termed as the output function.

If f_j represents the transfer function, then the output, y_j of the processing element is evaluated according to

$$y_j\,(t) = f_j\,(net_j\,(t)) \tag{2.1}$$

Once the choice of the activation and transfer function has been made, then all the nodes in the particular active layer use the same forms for evaluating their activation levels and outputs. Since activation is usually assumed to be equal to the net input, the need to distinguish between an activation function and the transfer function is not felt. As a result, the transfer function f_j is also referred to as the activation function. The transfer function can transform PE's activation in a linear or non- linear manner.

2.5 Modeling a Single Neuron

Fig. 2.3 shows a schematic representation of a typical processing element visualized by McCulloch and Pitts (1943).

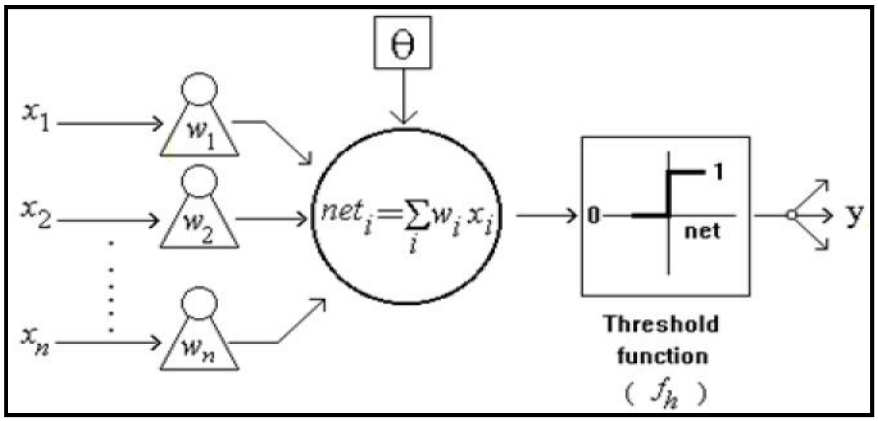


Fig.2. 3 Schematic Diagram of a Single Processing Element (PE) processing n inputs

It comprises of a single output net_j with *n* input connections x_i (*i= 1, 2,…, n*), which may be the outputs of neurons to which the particular neuron is connected. The inputs x_i can assume the values 1 or 0 depending upon whether the i^{th} neuron is firing or latent.

The output net_j of the j^{th} neuron is according to

$$net_j = f_h\left(\sum_{i=1}^{n} x_i\, w_{ji} - \theta\right) \tag{2.2}$$

Where f_h represents the threshold (Heaviside) function defined as:

$$f_h(net) = \begin{cases} 1 \text{ if } net \geq 0 \\ 0 \text{ otherwise} \end{cases} \tag{2.3}$$

The connection strength w_{ji} denotes the weight of the connection from i^{th} to j^{th} neuron and the weighted- sum $(\sum_{i=1}^{n} x_i\, w_{ji})$ represents the net input to the neuron. The parameter θ represents a threshold value which the weighted- sum must exceed for neuron to fire. The effect of threshold can be accounted by adding an input with a constant magnitude of +1 to the set of *n* inputs. The additional input, commonly referred to as the *zero*th input or *bias*, is assigned the weight equal to the negative of the threshold value. Under these circumstances the output net_j is given as:

$$net_j = f_h\left(\sum_{i=1}^{n} x_i\, w_{ji}\right) \tag{2.4}$$

It can be noted that eqs. (2.2) and (2.4) are numerically equivalent when $x_o = 1$ and $w_{jo} = -\theta$.

2.6 Well known models of ANN

Table 2. 2 Well- known ANN Model

ANN Models			
Feed forward		**Feedback**	
Supervised Learning	*Unsupervised Learning*	*Supervised Learning*	*Unsupervised Learning*
Adaline/ Madaline	Drive- Reinforcement Learning (DR)	Brain- State- in-a- Box (BSB)	Adaptive Bidirectional Associative Memory
Adaptive Heuristic Critic (AHC)	Fuzzy Associative Memory (FAM)	Fuzzy Cognitive Map(FCM)	Additive Gross berg (AG)
Associative- Reward- Penalty (ARP)	Learning Matrix (LM)		Analog Adaptive Resonance Theory (ART2)
Boltzmann machine (BM)	Learning Vector Quantizer (LVQ)		Binary Adaptive Resonance Theory (ART1)
Cauchy Machine (CM)	Linear Associative Memory (LAM)		Continuous Hopfield (CH)
Counter propagation (CP)			Discrete Hopfield (DH)
Multilayer Perceptron (MLP)			Shunting Gross berg (SG)
Perceptron			
Radial Basis Function (RBFN)			

2.6.1 Multilayer Feed forward Neural Networks

In feed forward neural networks, as the name suggests the information flow is in the forward direction only while in the feedback neural networks, the information is fed back to the nodes in the same layer and/or to those in the preceding layer(s). Feed forward neural networks (FFNs) are the most frequently used class of artificial neural networks for a wide spectrum of

applications, primarily involving non-linear function approximation (mapping) and classification. Most widely used MFFs are:

1. Multilayer Perceptron (MLP)
2. Radial Basis Function Network (RBFN)
3. Counter propagation Network (CP)

2.6.1.1 Architecture and Operation of Multilayer Perceptron (MLP)

A multilayer perceptron (MLP) is a feed-forward artificial neural network model that maps sets of input data onto a set of appropriate outputs utilizing a supervised learning technique called back-propagation for training the network (Rosenblatt, 1961 and Rumelhart et al., 1986). It is very useful in research in terms of its ability to solve problems stochastically which often provides approximate solutions for extremely complex problems like fitness approximation. It has therefore, become a standard algorithm for any supervised learning process and the subject of research in computational neuroscience and parallel distributed processing.

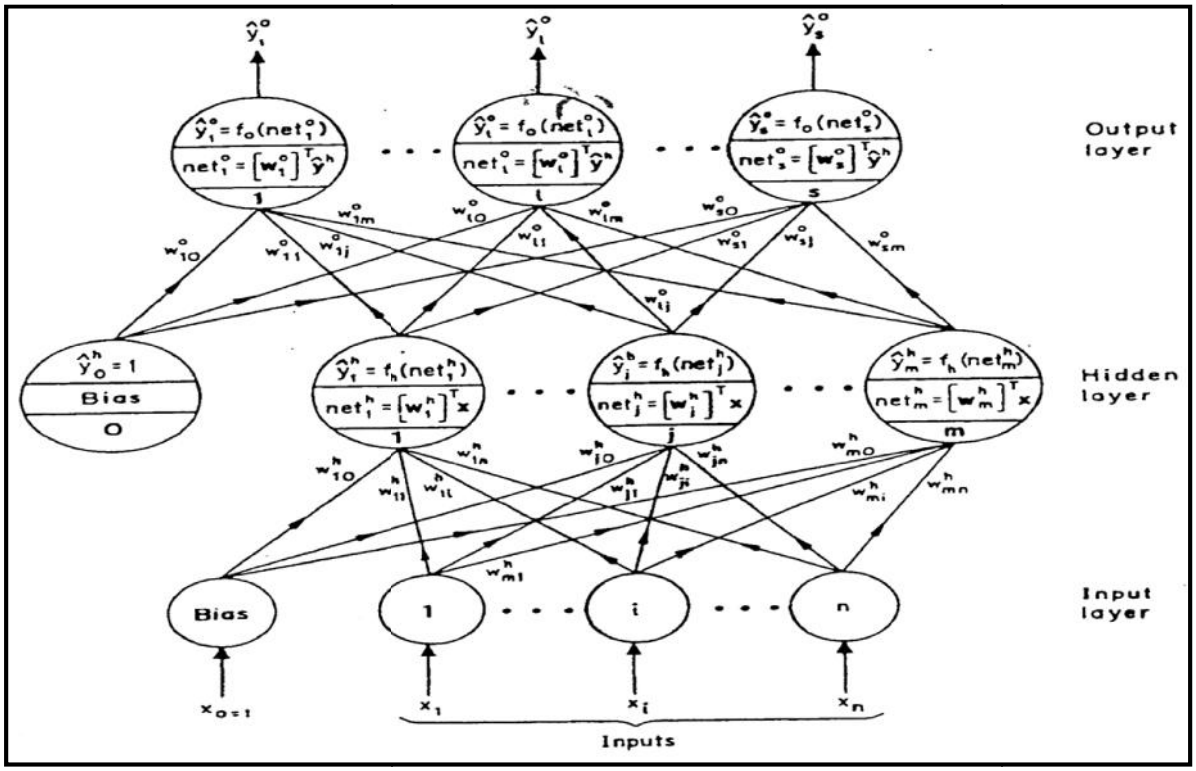


Fig.2. 4 Single hidden layered MLP network

2.6.1.1.1 Error- Back- Propagation

The *error- back- propagation* (EBP) or simply *back- propagation* (BP) is the most widely used algorithm for supervised training of multilayer perceptron network. Owing to its extensive use in the training of MLP networks, the network itself is often referred to as an EBP or BP network. The EBP algorithm employs a special kind of error- correction strategy which can be viewed as the generalization of the least- mean- squared (LMS) error minimization technique. LMS learning rule is targeted at single linearly- processing unit, whereas the EBP algorithm trains the weights associated with the non- linear processing elements of the MLP.

2.6.1.1.2 Implementation of EBP algorithm

1. Initialize hidden and output layer weights to small random values, e.g., between -1 and +1.
2. Apply k^{th} input pattern $x_k = (x_{k0}, x_{k1}, \ldots, x_{kn})^T$ to the network input layer.
3. Compute the weighted-sum of inputs (activation level) for the individual nodes in the hidden layer according to

   $net_{kj}^{h} = \sum_{i=0}^{n} w_{ji}^{h} \, x_{ki} \; ; j = 1, m$

   where net_{kj}^{h} denotes the weighted-sum of inputs for the hidden node j when k^{th} input pattern is presented to the network.
4. Transform the weighted-sum using the logistic sigmoid transfer function or hyperbolic tangent (tanh) function to compute the hidden node outputs as

   $\hat{y}_{kj}^{h} = \frac{1}{1+\exp(-net_{kj}^{h})} \; ; j = 1,m$

   $\hat{y}_{kj}^{h} = \frac{\exp(net_{kj}^{h}) - \exp(-net_{kj}^{h})}{\exp(net_{kj}^{h}) + \exp(-net_{kj}^{h})} \; ; j = 1,m$

   $\hat{y}_{kj}^{h}$, refers to the output of the j^{th} hidden- layer node.
5. Compute the weighted-sum of inputs for the individual output layer nodes (l= 1,s) as

   $net_{kl}^{o} = \sum_{j=0}^{m} w_{lj}^{o} \, \hat{y}_{kj}^{h} \; ; l = 1,s$

   Where, w_{lj}^{o} is the connection weight between node l in the output layer and node j in the hidden layer.

6. Transform the net activations of the output layer units using the logistic sigmoid transfer function or hyperbolic tangent (tanh) function. They form

$$\hat{y}_{kl}^{o} = \frac{1}{1+\exp(-net_{kl}^{o})} \; ; \; l = 1, s$$

$$\hat{y}_{kl}^{o} = \frac{\exp(net_{kl}^{o}) - \exp(-net_{kl}^{o})}{\exp(net_{kl}^{o}) + \exp(-net_{kl}^{o})} \; ; \; l = 1, s$$

7. Compute the scaled-error for the output layer units as

$$\delta_{kl}^{o} = (y_{kl} - \hat{y}_{kl}^{o})\, \hat{y}_{kl}^{o} (1 - \hat{y}_{kl}^{o}); \; l = 1,s$$

Where y_{kl} refers to the desired output of neuron l when input vector x_k is applied to the input nodes.

8. Compute the scaled-error for the neurons in the hidden layer according to

$$\delta_{kj}^{h} = \hat{y}_{kj}^{h} (1 - \hat{y}_{kj}^{h}) \sum_{l=1}^{s} \delta_{kl}^{o} w_{lj}^{o} \; ; \; j = 0,m$$

9. Update the weights between the output and hidden layer nodes as

$$w_{lj}^{o}(t+1) = w_{lj}^{o}(t) + \eta\, \delta_{kl}^{o}\, \hat{y}_{kj}^{h} + \alpha\, [w_{lj}^{o}(t) - w_{lj}^{o}(t-1)] \; ; \; j = 0, m; \; l = 1, s$$

Where the training iteration number is represented by t, and η, α denote the learning coefficient ($0 < \eta < 1$) and the momentum parameter ($0 \leq \alpha < 1$), respectively.

10. Update the hidden layer weights as given below

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta\, \delta_{kj}^{h}\, x_{ki} + \alpha\, [w_{ji}^{h}(t) - w_{ji}^{h}(t-1)] \; ; \; i = 0, n; \; j = 1, m$$

11. Repeat steps (2-6) with all the input patterns till the preselected convergence criterion is satisfied.
12. At convergence, the absolute rate of change in the average squared error per iteration must be sufficiently small. E is defined as $E = \frac{1}{p} \sum_{k=1}^{p} E_k$

For example, rate of change of E in the range 0.1-1% may be selected.

2.7 Artificial Immune Systems

Immunological computation (also called artificial immune systems) is a field of study devoted to the development of computational models based on the principles of the biological immune system (BIS). The BIS is a complex and highly distributive learning system with several mechanisms for defense against pathogenic organisms (Dasgupta et al., 2009). It learns through adaptation, to distinguish between dangerous foreign antigens and the body's own cells or

molecules. These powerful information-processing capabilities of the immune system provide rich metaphors for its artificial counterpart.

From 1990s until now there has been an increased research interest in immunity-based techniques and their applications. Some of these models, however, are intended to describe the processes that occur in the BIS so as to have a better understanding of the dynamic behavior of immunological processes and to simulate BIS's dynamic behavior in the presence of antigens/ pathogens. In contrast, immune-inspired models have been developed in an attempt to solve complex real-world problems such as anomaly detection, pattern recognition, data analysis (clustering), function optimization, and computer security.

2.8 Overview of Biological Immune System

Mammals have developed a robust defense system called the immune system to deal with foreign and potentially dangerous pathogens. The immune system consists of a set of organs, cells, and molecules; and their coordinated response in the presence of a pathogen is known as the immune response. In a broader sense, the physiological function of the immune system is to defend an organism against all kinds of harmful non-self invaders called as antigens such as fungi, bacteria, parasites, viruses, and other protozoa. The ability of an antigen to induce an immune response probably depends on four main factors: foreignness, molecular size, chemical composition and heterogeneity, and susceptibility to antigen processing and antigen presentation (Dasgupta et al., 2009).

The biological immune system (BIS) has the ability to detect foreign substances, and to respond adequately. It is inherently distributed and fault-tolerant, and exhibits a complex behavior while interacting with all its constituents. One of the main capabilities of the immune system is to distinguish own body cells from foreign substances, which is called self/ non-self discrimination. In general, the BIS are capable of recognizing the dangerous elements and deciding an appropriate response while tolerating self-molecules and ignoring many harmless substances.

2.8.1 Functional Elements of the Immune System

The immune system is a collection of organs, cells, and molecules responsible for dealing with potentially harmful invaders; it also has other functionalities in the body.

2.8.1.1 Bone Marrow

Naive immune cells are initially generated in the bone marrow and these derived stem cells divide into either mature immune cells (to perform immunological function) or precursors of cells that migrate out of the bone marrow to continue their maturation process elsewhere (thymus or germinal centre (GC)) (Murphy, 2008). In addition to red blood cells and platelets, the bone marrow produces B cells, natural killer cells, granulocytes, and immature thymocytes.

2.8.1.2 Thymus

The function of the thymus is to produce mature T cells. Some immature immune cells leave the bone marrow and migrate into the thymus. T cells that are beneficial to the immune system are kept through a maturation process, referred as 'thymic education', whereas those T cells that might cause a detrimental autoimmune response are eliminated; mature T cells are then released into the bloodstream for performing immunological functions (Murphy, 2008).

2.8.1.3 Lymphocytes or WBCs

White blood cells (WBCs), also called lymphocytes, are very important constituents of the immune system. These cells are produced in the bone marrow, circulate in the blood and lymph system, and reside in various lymphoid organs to get matured and to perform various immunological functions. B and T cells constitute the major population of lymphocytes.

2.8.1.3.1 T Cells

T cells are produced in the Bone Marrow and get matured in the thymus. The thymus produces five types of T cells which are as follows:

1. Delayed hypersensitivity T cells: This type of T cells produce cytokines (lymphokines + monokines) that direct the cellular- mediated immune response and also phagocytosis (A process through which specialized cells like macrophages, engulf large particles such as bacteria, yeast, and dying cells into it).
2. Helper T cells: Helps B cells in the recognition of the antigen by releasing cytokines.
3. Cytotoxic T cells or Killer T-cells: Bind themselves on the foreign invaders and inject poisonous chemicals into them causing their destruction. They also kill infected self-cells and tumour cells.
4. Memory T cells: Remembers earlier immune responses.

5. Suppressor T cells: Inhibits the action of other immune cells thus preventing allergic reactions and autoimmune diseases.

2.8.1.3.2 B Cells

B cells are specialized white blood cells produced in the bone marrow and are responsible for producing and secreting Y-shaped antibodies, which bind to antigens (see fig., 2.5). Each B cell secretes multiple copies of one kind of antibody for antigen match. Activated B cells become memory cells or plasma cells; the latter actively secret antibodies.

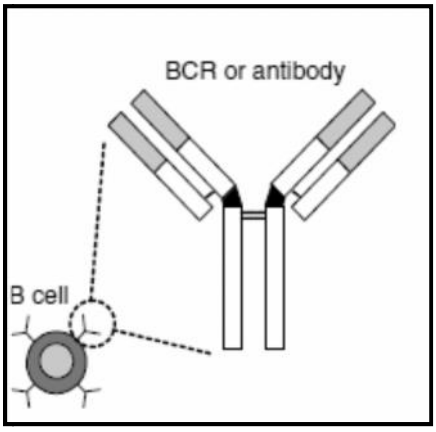


Fig.2. 5 Y-shaped antibody produced by the B-Cell binding with the antigen

2.8.1.3 Antibodies

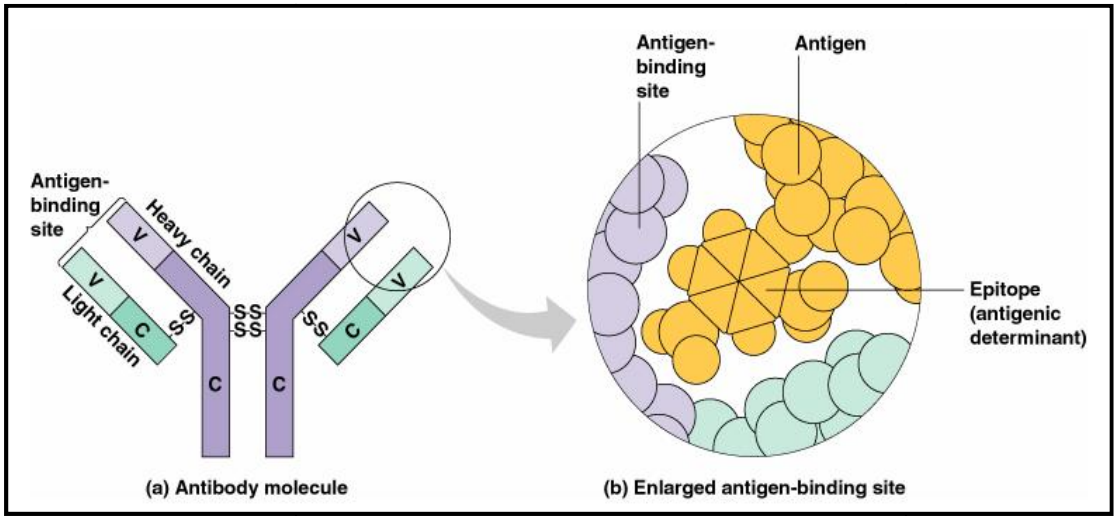


Fig.2. 6 Antibody Molecule

Antibodies (Abs) are a particular kind of molecules, called immunoglobulin found in the blood and produced by mature B cells, also known as plasma cells. An antibody contains four

polypeptide chains: two identical light chains and two identical heavy chains. Each chain comprises a variable region (V) and a constant region (C) (see fig., 2.6).

2.8.1.5 Antigens

Antigen is a foreign substance that, when introduced into the body, is capable of stimulating an immune response, specifically activating lymphocytes, which are the body's infection-fighting white blood cells. Virtually any large foreign molecule can act as an antigen, including those contained in bacteria, viruses, protozoa, helminthes, foods, serum components, red blood cells, and other cells and tissues of various species, including humans. An antigen that induces an immune response i.e., stimulates the lymphocytes to produce antibody or to attack the antigen directly—is called an immunogen.

2.8.2 Layers of Immune System

The human body is protected against foreign invaders by a multi-layered system. The immune system is composed of physical barriers such as the skin and respiratory system; physiological barriers such as destructive enzymes and stomach acids. The immune system, which has can be broadly divided under two heads – Innate (nonspecific) Immunity and Adaptive (specific) Immunity, which are inter-linked and influence each other. The Adaptive Immunity again is subdivided under two heads – Humoral Immunity and Cell Mediated Immunity (Burke and Kendall, 2005).

2.8.2.1 Innate Immunity

Innate immunity (Woods, 1991) refers to all defense mechanisms against foreign pathogens that individuals are born with. Innate immunity is mainly composed of the following mechanisms:

1. Phagocytic barriers: Some specialized cells (like macrophages, natural killer cells) are able to ingest foreign substances, including whole pathogenic microorganisms. This ingestion has two purposes: to kill the antigen and to present fragments of the invader's proteins to other immune cells and molecules.
2. Inflammatory response: Activated macrophages produce cytokines (hormone like protein messengers), which induce the inflammatory response characterized by vasodilation and rise in capillary permeability. These changes allow a large number of circulating immune cells to be recruited to the site where an infection occurs.

2.8.2.2 Adaptive Immunity

Adaptive immunity (Kuby et al., 2000; Stanley, 2002), also called acquired or specific immunity, represents the part of the immune mechanism that is able to specifically recognize and selectively eliminate foreign microorganisms and molecules. Adaptive immunity produces two types of responses in the presence of pathogens: humoral immunity and cellular immunity:

1. Humoral immunity: The humoral branch of the immune system involves interaction of B cells with antigen and their subsequent proliferation and differentiation into antibody-secreting plasma cells. Antibody functions as the effectors of the humoral response by binding to antigen and facilitating its elimination. This elimination can be done in several ways. For example, antibody can cross-link the antigen, forming clusters that are more readily ingested by phagocytic cells or this can activate the complement system which results in destruction of the foreign organism.
2. Cellular immunity: Cellular immunity is cell-mediated; thus, cytotoxic T cells participate in cell-mediated immune reactions by killing altered self-cells; virus infected and tumour cells. Cytokines secreted by TDH cells can mediate cellular immunity, and activate various phagocytic cells, enabling them to kill microorganisms more effectively. This type of cell-mediated immune response is especially important in host defence against intracellular bacteria and protozoa.

2.9 How Immune System protects the body

A schematic of how a biological immune system function is shown in Fig. 2.7. An important class of immune cells namely lymphocytes comprises B-cells, T-cells and antibodies. When a foreign substance (e.g., pathogen, virus, and parasites) invades an organism, delayed hypersensitivity T cells (TDH, a type of T cells) recognize the infection and start producing cytotoxic factor (hormone like protein messengers). This helps macrophages (phagocytes) to reach to the infection. Then TDH cells produce migration inhibitory factor to resist macrophages from leaving that site. Macrophages after engulfing the pathogenic microorganism within it break it into short chains of amino acids (also called a “peptides”). These are held on to the surface of macrophages where Helper T-cells recognize them.

During their maturation, T-cells are made to undergo a selection process to ensure that they are able to recognize the non-self peptides (pathogens) held on to the macrophages. This process has two main phases: *Positive Selection* and *Negative Selection*. In positive selection, the T-cells are tested for their recognition of macrophages (self cells). If a T-cell recognizes it, it is retained otherwise discarded. The algorithm simulating this behavior is termed "Positive Selection" algorithm. The purpose of negative selection is to provide tolerance for self cells. Here, those T-cells that react against the self-proteins are destroyed, and only those, which do not bind to the self-proteins, are allowed to circulate throughout the body to perform immunological functions and to protect the body against foreign antigens. The algorithm which simulates this behavior is known as "Negative Selection" algorithm (Forrest et al., 1994).

Following recognition of the antigen, the antigen-activated T-cells secrete lymphokines. When B-cells recognize an antigen in the presence of lymphokines secreted by the Helper T-cells, the receptors on the surface of the B-cells bind to the antigen. This stimulates B-cells and they start proliferating and differentiating. This process is called "clonal selection". The B-cells generated thereby become either memory cells or plasma cells.

Plasma cells secrete large amounts of Y shaped antigen-specific antibodies that bind themselves with the antigen, whereas Memory cells are there to ensure that the antigen receives a much faster response if it attacks again.

In the early stages of the immune response, the affinity between antibodies and antigens may be low. However, as B-cells undergo "clonal selection" they clone and mutate repetitively to improve the binding affinity between the antigen and B-cell. This mutation process is called *somatic hypermutation*. The entire process of creation of new B-cells with high affinity to an antigen (*clonal selection* + *somatic hypermutation*) is called *affinity maturation* (Dasgupta and Nino, 2009). Ultimately, affinity maturation leads to the production of a pool of antibody-secreting plasma cells and a pool of memory cells (Dasgupta and Nino, 2009). The B-cell response to antigens inspired development of *Clonal Selection Theory* (Burnet, 1959) based on which an algorithm has been proposed called *Clonal Selection* Algorithm (De-Castro et al.,

2001). The antigen coated with an antibody is ultimately killed by the Killer T cells (a type of T cells).

Artificial immune systems can incorporate many properties of natural immune systems, including diversity, distributed computation, error tolerance, dynamic learning and adaptation and self-monitoring. It is a general framework for a distributed adaptive system and could, in principle, be applied to many domains (Hofmeyr and Forrest, 2000) mainly comprising classification and optimization tasks.

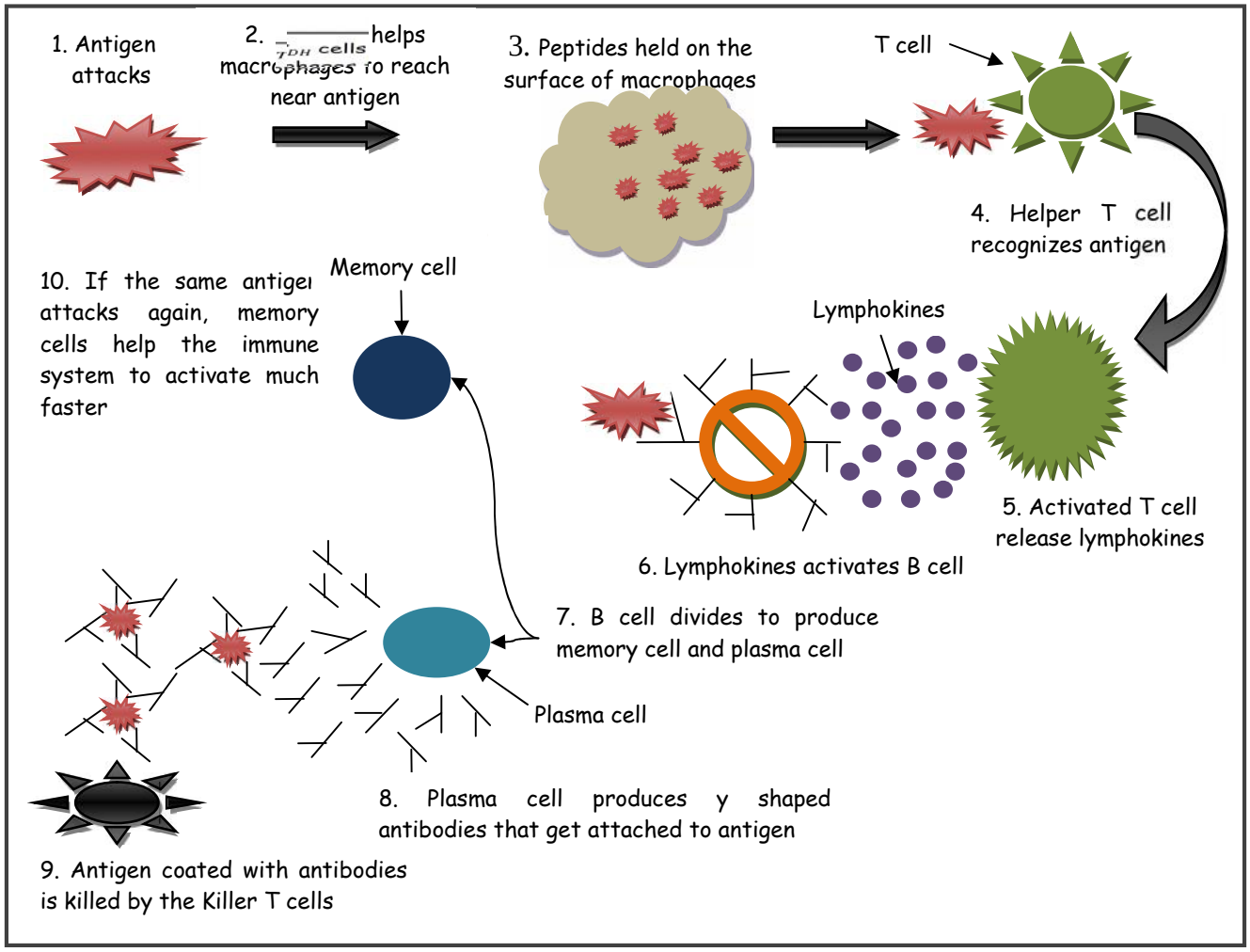


Fig.2. 7 Illustration of Immune Response

2.10 AIS based Computational Models

2.10.1 Negative Selection Algorithm

Negative Selection Process (NSA) models T-cell maturation process. Although several variants of NSA have been proposed, they all share the basic features of the original algorithm. The aim of NS is to discriminate between self and non-self patterns, while only self samples are available (Dasgupta and Nino, 2009). NSA has been applied for fault detection and diagnosis (Kaushik and Rajgopalan, 2010), detecting data manipulation caused by computer viruses (Forrest et al., 1994), and pattern recognition (Dasgupta and Nino, 2000; Malhotra et al., 2012).

2.10.2 Positive Selection Approach

Similar to the NSA, *Positive Selection Approach* (PSA), models the T-cell maturation process, however, it aims to generate set of detectors that match "self" patterns instead of the "non-self." A simple model of it can be built by using a nearest neighbour approach i.e., if a point lies in a neighbourhood of a sample self-point, then it is labelled as belonging to the self-set (Dasgupta and Nino, 2009) otherwise to the non-self set. Although not employed extensively, PSA can be applied for pattern recognition (Dasgupta and Nino, 2000) clustering, and in other domains (Dasgupta and Nino, 2009).

2.10.3 Clonal Selection Algorithm

Clonal selection algorithm models clonal selection, proliferation and affinity maturation process of B cell in the presence of an antigen. Important features of clonal selection relevant from the viewpoint of computation are:

Step 1: Random initialization of the population of antibodies.

Step 2: Antigen presentation.

Step 3: Computation of antibody affinity with antigen.

Step 4: A subset of highest affinity antibodies is selected and exact copies of it are generated. Higher the affinity of an antibody set with the antigen higher the number of clones is made.

Step 5: All the clones are mutated at a rate inversely proportional to their affinities with the antigen. Again, the affinity between the cloned mutated antibodies is computed with the antigen and the best ones are re selected.

Step 6: Antibodies with lowest affinities are replaced by randomly generated new antibodies.

Clonal Selection Algorithm (CLONALG) was initially proposed to perform pattern recognition (White and Garrett, 2003; Malhotra et al., 2012) and then was adapted to solve multi-modal optimization (Coello Coello et al., 2007) and clustering (Younsi and Wang, 2004) tasks.

2.11 Application of AIS in Real world Problems

Artificial immune systems have been used as a problem solver in a wide range of domains such as for detecting data manipulation caused by computer viruses (Forrest et al., 1994), Navigation Control and Path Mapping of a Mobile Robot (Rao et al., 2010), Bioinformatics data mining (Dixon and Yu, 2010), Web Mining (Secker et al., 2005), Software Reliability Estimation (Chitra and Rajaram, 2008) and Global Optimization problems (Coello Coello et al., 2007). They have also been used in conjunction with other methods (hybridized) such as genetic algorithms (GAs), neural networks, fuzzy logic, and swarm intelligence.

2.12 Genetic Algorithms

In the computer science field of artificial intelligence, genetic algorithms (GA) is a search heuristic that mimics the process of natural evolution such as inheritance, mutation, selection, and crossover to generate solutions to the optimization and search problems. It is a stochastic search and optimization methodology that performs global search and is not heavily constrained by the smoothness, continuity, differentiability properties of the objective function to be maximized/minimized. Therefore, it needs measurement of the objective function only and not its derivatives (as done in gradient-based methods).

GA has found many applications in the field of bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics, pharmacometrics etc.

2.12.1 Methodology

The evolution usually starts from a population of randomly generated individuals and is an iterative process where the population in each iteration called a *generation*. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome

is modified (through repetitive application of the mutation, crossover, inversion and selection operators) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

Pseudo Code:

BEGIN

INITIALISE population with random candidate solution

Evaluate fitness of each candidate solution

REPEAT UNTIL (termination condition) is satisfied DO

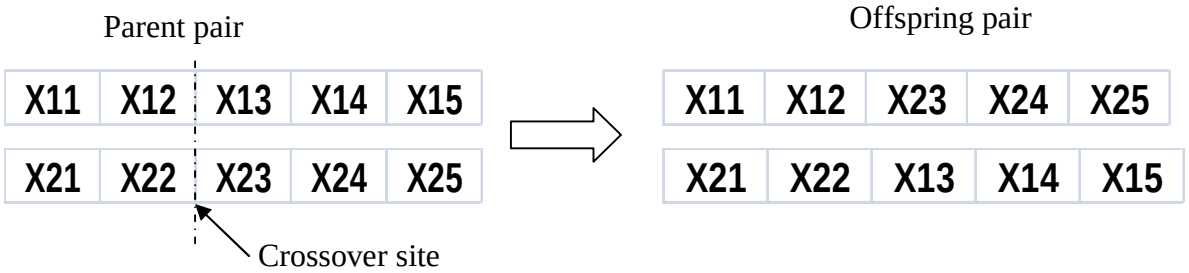
1. SELECT parents;
2. RECOMBINE (CROSSOVER) pair of parents;
3. MUTATE the resulting offspring;
4. SELECT individuals or the next generation;

END

Initialization: Many individual solutions (chromosome sets) are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions.

Selection of parents: During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a *fitness-based* process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected.

Crossover: A pair of "parent" solutions is selected for breeding from the pool selected previously to produce a new solution. For "*single point crossover*" new solutions (offspring's) are created by slicing the parent solutions and interchanging the sliced portions.



Mutation: is used to maintain genetic diversity from one generation of a population to the next. It is analogous to biological mutation. It alters one or more gene values in a chromosome from its initial state and this changes the solution entirely. Hence GA can come to a better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search.

All these four processes namely, initialization, selection of parent pairs, crossover and mutation ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness gets increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children.

Termination: This generational process is repeated until a termination condition has been reached. Common terminating conditions are: a solution is found that satisfies minimum criteria or fixed number of generations reached.

Summary of Chapter- 2

In the first part of the chapter artificial neural networks (ANN) is explained. It is a very highly interconnected system having three layers i.e. input, hidden and output which can be applied to almost all kinds of data to obtain a mathematical model. It does not require any knowledge of reaction mechanism, kinetics, heat and mass transfer processes and related parameter values of the process. Various models of ANN have been made since 1950’s but the most effective and widely used in all of them is multilayer perceptron (MLP). It trains the network using error back propogation (EBP) which uses generalized delta rule.

Second part comprises all about artificial Immune systems (AIS). It is an emerging field and not much has been done in it. It mimics the complex biological immune system and solves highly complicated optimization, classification and clustering problems. On T cell maturation and B cell proliferation, differentiation and affinity maturation processes various theories are

being made like the negative selection theory, positive selection theory and clonal selection theory and based on them negative selection algorithm, positive selection algorithm and clonal selection algorithm respectively, are made which are applied in various real world problems.

Genetic Algorithms is explained in the third part of the chapter. It is a stochastic search and optimization methodology which is based upon the Darwinian survival of the fittest theory and utilizes characteristics of genetics such as inheritance, selection, crossover and mutation, and to generate solutions.

3. APPLICATION OF AIS TO H_2S ABATEMENT PROCESS

HYDROGEN sulfide (H_2S) is a highly toxic and undesirable gaseous component of the natural gas and other waste gaseous streams. The catalytic conversion of H_2S to elemental sulfur can be achieved by various chemical and biological methods. The gas desulfurization can also be carried out using liquid redox chemistry, where iron complexes are alternately reduced according to the following reactions.

$$H_2S_{(g)} + H_2O_{(l)} \leftrightarrow H_2S_{(l)} \tag{3.1}$$

$$H_2S_{(l)} + 2Fe^{3+}L^{n-}_{(l)} \rightarrow S^0 \downarrow + 2H^+ + 2Fe^{2+}\,L^{n-}_{(l)} \tag{3.2}$$

Since the active ferric chelate is converted to inactive ferrous chelate, the later component has to be regenerated by oxidation according to the following reactions;

$$\tfrac{1}{2}\,O_{2\,(g)} + H_2O_{(l)} \leftrightarrow \tfrac{1}{2}\,O_{2\,(l)} \tag{3.3}$$

$$2\,Fe^{2+}\,L^{n-}_{(l)} + \tfrac{1}{2}\,O_{2\,(l)} + H_2O_{(l)} \rightarrow 2\,Fe^{3+}\,L^{n-}_{(l)} + OH^- \tag{3.4}$$

where, ‘n’ denotes the charge of an organic ligand ‘L’, which is usually a polyaminocarboxylic acid. Here, no iron chelate is consumed in the overall reaction; it may therefore be considered as a pseudocatalyst.

The experimental records of the oxidative scrubbing of H_2S by ferric chelates are available in the open literature. The Fe^{3+} redox couple process using hexacyanoferrate known as “Staatsmijnen-otto” process was reported by Pieters (Pieters et al., 1946). Problems with this process are usage of toxic solutions containing $Fe(CN)_6^{n-}$ (n = 3 or 4) and ammonia, and occurrence of side reactions.

For the first time a detailed study involving screening of biodegradable Fe^{3+}-chelates for the catalytic conversion of H_2S gas was reported by Deshpande (Deshpande, 2009). Among various chelates that were screened by them the Fe^{3+}-MA chelate was found to exhibit maximum sulfur recovery and purity. Thus, they studied mechanically agitated batch reactor (MABR) using Fe^{3+}-MA chelate as a catalyst extensively. The process data generated by them has been used in this work to model the same process and maximize the H_2S conversion by using AI based hybrid strategies, ANN- AIS, ANN-GA and to compare their performances.

3.1 Sensitivity Analysis

A number of process operating variables and parameters are known to influence the performance of a MABR and thereby H_2S conversion. To ascertain these influences we scrutinized the effects of a number of variables and parameters on the catalytic oxidative absorption of H_2S monitored in terms of H_2S conversion (%). The variables and parameters considered for scrutiny were: (i) initial Fe^{3+}-MA concentration ($C_{Fe,0}$), (ii) H_2S gas concentration ($C_{H2S,0}$), (iii) reactor pressure (P), (iv) reactor temperature (T), (v) stirring speed of impeller (S) and (vi) reaction time *(t).* The variables which may influence the H_2S abatement but were set constant throughout the investigation and thus excluded from the scrutiny were: pH of Fe^{+3}-MA chelate, reactor dimensions, reactor volume, liquid volume, gas volume, impeller dimensions and its position.

The result of the sensitivity analysis carried out on the 73 experimentally obtained data points (see Appendix I) using the IBM SPSS Statistics 19 package shows that the initial Fe^{3+}-MA concentration ($C_{Fe,0}$) has exerted least influence on the oxidative absorption of H_2S (see Table 3.1). For the sake of experimental convenience most (65) of the experiments were conducted at Fe^{3+}-MA concentration of 5000 ppm. This could be reason underlying Fe^{+3} concentration as the least influential among the six process variables and parameters, subjected to the sensitivity analysis. Owing to its low influence on the oxidative absorption of H_2S, the initial Fe^{3+}-MA concentration variable was not considered as an input during process modeling and optimization.

Table 3.1 Relative importance and normalized importance of the input variables

Independent Variable Importance		
Process Variables	**Relative Importance**	**Normalized Importance**
Initial H_2S Conc.	.358	100.0%
Pressure	.152	42.4%
Stirring speed	.151	42.2%
Temp	.148	41.3%
Time	.123	34.2%
Fe+3 Conc.	.068	19.0%

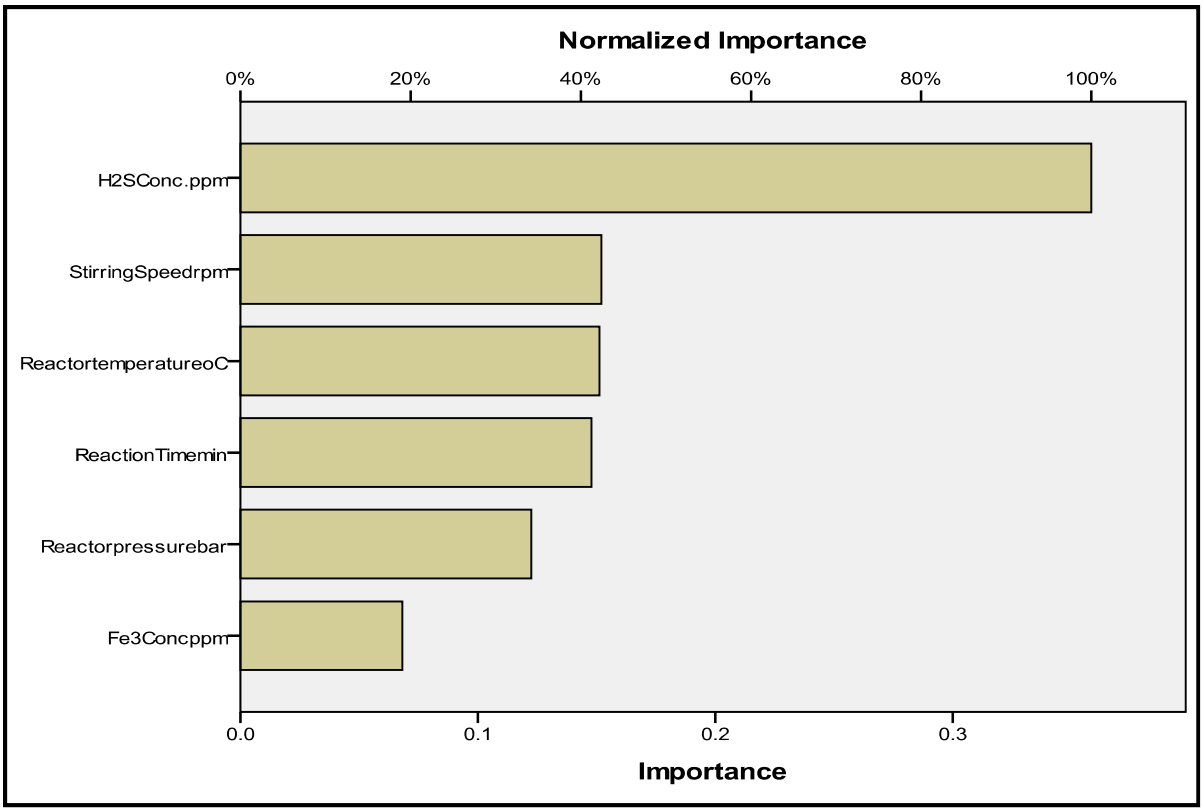


Fig. 3.1 Result of Sensitivity Analysis showing the significance of input variables and parameters on the conversion of H2S

3.2 ANN-based modeling for H_2S Conversion

The MLP network architecture considered in modeling of the H_2S abatement process has five input nodes (n = 5) representing as many significant process operating variables namely *H_2S concentration* (ppm) (x_1), *reactor pressure* (bar) (x_2), *reactor temperature* (^oC) (x_3), *stirring speed of the impeller* (rpm) (x_4) and *reaction time* (min) (x_5), and a single output node (s = 1) representing *H_2S conversion* (%) at the end of the batch run.

For MLP-based modeling, process data from 65 experiments with a constant Fe^{3+}-MA concentration of 5000 ppm were employed and their ranges are as follows: H_2S gas concentration: 5000-50000 ppm, reactor pressure: 1.01- 5.2053 bar, reactor temperature: 25-85.22 ^oC, stirring speed of impeller: 100-500 rpm, reaction time: 15-60 min, and H_2S conversion: 2.96- 91.5 %. Each of the five inputs ($x_1 - x_5$) and the single output (y) in the example set comprising data from 65 experiments was normalized to lie between 0.1 and 0.9.

To ensure that the ANN model is capable of generalization, the example input-output set was partitioned randomly into a *training* set (49 patterns) and a *test* set (16 patterns). While the training set was utilized for adjusting the weights (W) of the MLP model during its training, the test set was used for gauging the network's generalization performance after each training iteration.

The weights resulting in the least and comparable training and test sets RMSE (*Root Mean Square Error*) magnitudes were chosen as the optimum weights. In the MLP training procedure, the hyperbolic tangent activation function was used for computing outputs of the two hidden layer nodes and linear identity function was used for the output layer nodes. It was observed that an MLP architecture with seven nodes each in hidden-1 and hidden-2 layers resulted in the least values for the training set RMSE (i.e., E_{trn} = 0.04933%) and test set RMSE (i.e., E_{tst} = 0.0437%). The value of the correlation coefficient (CC) between the model-predicted and desired (experimental) H_2S conversion (%) for training and test data were 0.953 and 0.994 respectively. The small and comparable magnitudes of the RSME and the high value of CC suggest that the MLP-based process model possesses excellent prediction and generalization characteristics.

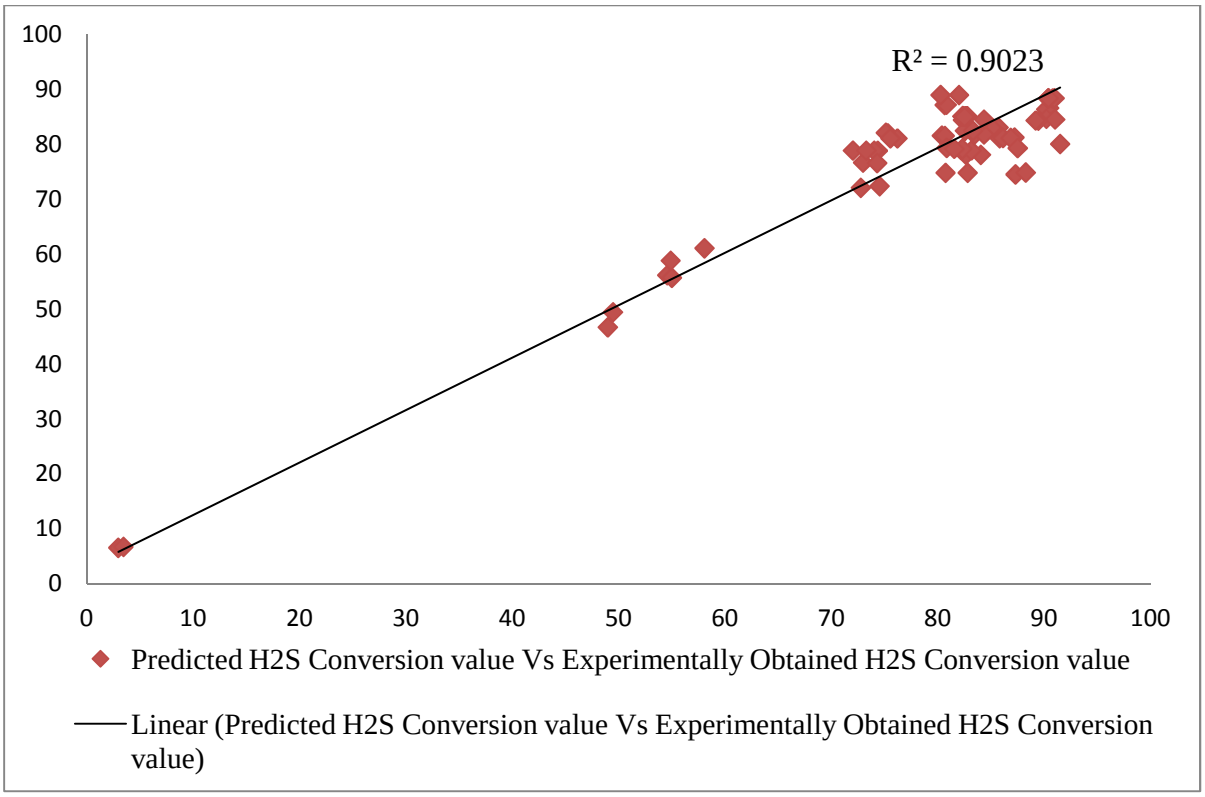


Fig. 3.2 A comparison of the MLP model predicted and the corresponding experimental values of the H_2S conversion.

3.3 AIS-Based Optimization of inputs of ANN- Model

In this study the *Clonal selection algorithm* (i.e. CLONALG) was used for the optimization of the five inputs of the ANN model representing process condition variables and parameters. The single objective of the optimization was to maximize the H_2S conversion (%). The basic tenet of the clonal selection is that only those antibodies that recognize the antigens are selected to proliferate. The selected antibodies are then subjected to an affinity maturation process, which improves their affinity to the selective antigens (Coello Coello et al., 2007).

In CLONALG, an antibody represents a candidate solution to the optimization problem. Consider a population of antibodies represented by a matrix, pop with dimensions $m \times n$ where m denotes the number of antibodies (population size) and n denotes dimensionality of the decision variable (candidate solution) vector x_i^T where i^{th} ($i = 1, 2, \ldots, m$) is denoted as [x_{i1}, x_{i2}, …., x_{in}]. The pop matrix is generated randomly wherein j^{th} decision variable lies between $x_{j,\ min} \leq x_{ij} \leq x_{j,\ max}$.

Optimization problems are typically of two kinds “constrained” and “un-constrained” optimization. All the nature inspired algorithms and also CLONALG are well suitable for unconstrained optimization problems.

3.3.1 Implementation of unconstrained CLONALG algorithm for H_2S abatement process

Maximize $f(\bar{x})$, such that $x_{min} \leq x_{ij} \leq x_{max}$, $i = 1, 2, \ldots, m$; $j = 1, 2 \ldots, n$ (3.5)

*Step*1. *Initialization*: The population of anti-bodies (65×5 matrix) representing candidate solutions to the optimization problem under consideration is randomly initialized. This population of antibodies can be coded in binary or real numbers as in genetic algorithm. It is randomly assumed that this population contains two parts *memory antibodies* (*M*) and *replaceable antibodies* (*r*); and is following 80:20 ratios. Accordingly first 49 rows of the 65×5 matrix represent memory antibodies (candidate solutions) and the remaining 16 rows represent replaceable antibodies.

Step 2 (*Fitness evaluation and ranking)*: The antibody affinity (fitness score) for each antibody (candidate solution) in the population is evaluated by using the MLP model. Here, each antibody (5-dimensional row vector of the population matrix) is used as an input to the ANN model and its output i.e., H_2S conversion (%) is evaluated. Fitness score indicates how well a candidate solution fulfills the optimization objective. Based on their fitness scores the antibodies are ranked in the decreasing order of fitness scores.

Step 3 (*Selection and cloning*): Select the top *M* best antibodies and clone (copy) them in proportion to their individual affinities or fitness scores. The candidate solutions with higher fitness scores produce more copies than their inferior counterparts as given by following equation:

$$NC = \left(\frac{\beta \times pop}{i}\right) \quad (3.6)$$

where *NC* is the number of clones generated, which should be an integer, $i = 1, 2, \ldots, M$, and β is cloning factor (generally equal to 1) (Coello Coello et al., 2007).

Step 4 (*Hyper mutation*): This process is unique to artificial immune systems. Here, the cloned antibodies are mutated in inverse proportion to affinity, wherein each antibody in the population is subjected to some variation. In binary coded antibody population hyper-mutation may be as

simple as flipping a bit from 0 to 1 or 1 to 0 in each antibody. The hyper-mutation method for a real coded population was given by Coello Coello et al. (2007) however its drawback is that hyper- mutation moves the candidate solution far away from the global optimum, then a large number of iterations would be required to reach the optimum since only one variable is changed at a time. The hyper mutation operator used in this study is particularly suited for the real coded antibody population:

$$x_i^{'} = x_i + rand\,() \times \left(\frac{i^2}{10^p}\right) \times (x_{max} - x_{min}) \tag{3.7}$$

where *rand*() generates random number with uniform distribution between 0 and 1, p is user-defined parameter usually taken as 2, 3, or 4 based on scale of the problem. In this study the mutation parameter, i.e. p is taken as 4, $i = 1, 2, \ldots, M$ and x_{max} and x_{min} are 0.9 and 0.1, respectively.

Step 5 (*Fitness score reevaluation*): Fitness scores of hyper-mutated antibodies are evaluated using the ANN model and antibodies are ranked in the descending order of on their fitness scores. Next, fitness scores of the ranked antibodies are compared with those of non-hyper-mutated antibodies and if the new fitness score of an antibody is lesser (for maximization problems) than the original, then the corresponding solutions are replaced by the original solutions else new (hyper-mutated) ones are retained.

Step 6 (*Diversity introduction)*: Replaceable antibodies are added in the hyper-mutated antibodies. This helps in avoiding the local optima and widening the solution search space. The resulting population is the newly generated population of candidate solutions.

Step 7 (*Convergence check*): Repeat steps 3 to 6 until convergence is reached. The convergence criterion could be: (i) the algorithm has evolved over pre-specified maximum number of generations, or (b) the fitness score of the best antibody (solution) remains constant or changes negligibly over a large number of successive generations.

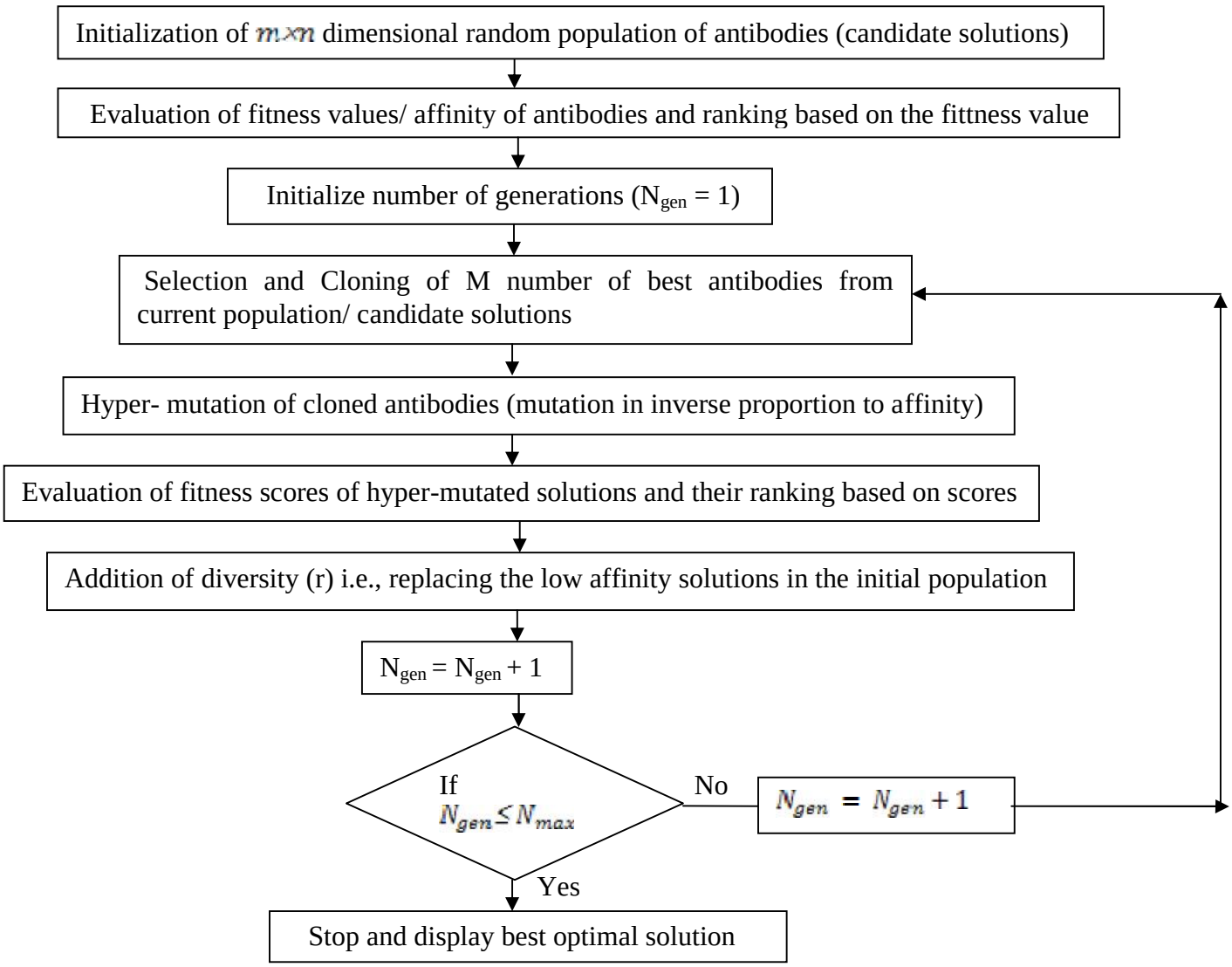


Fig. 3.3 Flowchart of CLONALG algorithm for unconstrained optimization problems

3.3.2 Stepwise procedure for modified CLONALG algorithm for constrained optimization

Most of the optimization problems in engineering and science involve constraints which need to be satisfied to get an optimal solution. The solutions which satisfy all the constraints are called feasible solutions and which violate the constraints are called infeasible solutions.

Constrained optimization problem is of the form:

$$Max/Min\, f(\bar{x}), \text{ such that } x_{min} \leq x_{ij} \leq x_{max}\ ,\ i = 1, 2, \ldots, m;\ j = 1, 2 \ldots, n \qquad (3.8)$$

$g_i(\bar{x}) \leq 0$ $\quad i = 1, 2 \ldots z$ $\quad z$ correspond to number of inequality constraints.

$h_i(\bar{x}) = 0$ $\quad i = 1, 2 \ldots q$ $\quad q$ correspond to number of equality constraints.

In our present work we have used constraint violation method from which the penalty factor values are obtained and the fitness function is evaluated by using static penalty method. Then a feasibility check is performed and the solutions are separated as feasible and infeasible solutions (Coello Coello et. al., 2007). Based on this approach we developed modified CLONALG algorithm for handling constrained optimization problems.

*Step*1 (*Initialization*): The population of anti-bodies (m×n) representing candidate solutions is randomly initialized. This population contains two parts *memory* antibodies (*M*) and *replaceable* antibodies (*r*) in 80:20 ratios.

*Step*2 (*Finding penalty values*): By using the static penalty function method penalty factor values are evaluated. These values are assigned based on the degree of constraint violation, if the degree of violation is higher, then a higher penalty value is imposed and if violation is lower than a low value of penalty is used. Too high value of penalty or too low value of penalty can't be used because a large penalty discourages the exploration of the infeasible region, and by giving a very low penalty value a lot of search time will be spent exploring the infeasible region.

*Step*3 (*Feasibility check*): After the fitness scores are evaluated the feasibility of candidate solutions or anti-bodies is checked, based on that they are separated as feasible and infeasible solutions and arranged in descending order (for maximization problem) based on their fitness scores or affinity.

*Step*4 (*Addition of infeasible solutions*): Top 80% of feasible solutions are selected and remaining infeasible solution is added to them. This is because in highly non-linear problems the solution lies in the boundary of feasible and infeasible solutions. By doing this the search space for each generation includes both feasible and infeasible solutions, which may result solution near to optima. Selecting the size of feasible solutions is user defined.

Step5 (Selection, Cloning & Hyper-mutation): Selected Antibody sets are then cloned and hyper mutated.

Step6 (Fitness Score Evaluation): Fitness scores of hyper-mutated antibodies are evaluated using the ANN model and antibodies are ranked in the descending order of on their fitness scores. Next, fitness scores of the ranked antibodies are compared with those of non-hyper- mutated antibodies and if the new fitness score of an antibody is lesser (for maximization problems) than the original, then the corresponding solutions are replaced by the original solutions else new (hyper-mutated) ones are retained.

Step7 (*Addition of Diversity)*: replaceable antibodies are added to the hyper-mutated candidate solutions.

*Step*8 (*Convergence criterion*): Repeat steps 5 to 7 until convergence is reached. The convergence criterion could be: (i) the algorithm has evolved over pre specified maximum number of generations or (b) the fitness score of the best antibody (solution) remains constant or changes negligibly over a large number of successive generations.

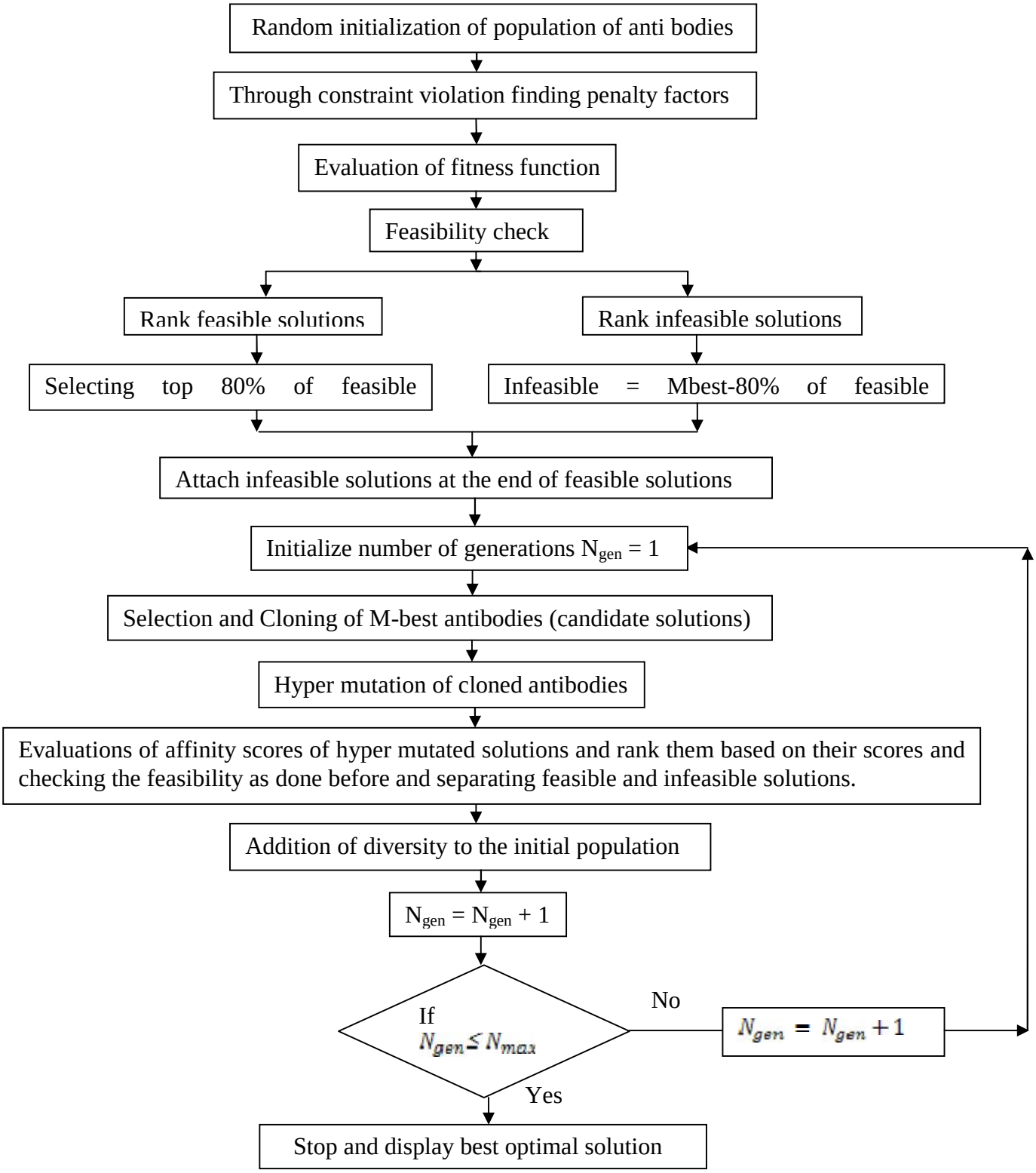


Fig. 3.4 Flowchart of CLONALG algorithm for constrained optimization problems

3.4 GA-Based Optimization of inputs of ANN-Model

3.4.1 Stepwise procedure for Genetic Algorithms

*Step*1 (*Initialization)*: Group of chromosomes also called as population (pop), which is $m \times n$ dimensional are randomly initialized within their boundary limits (0.1-0.9). Where, m is the number of the data values and n is the number of the variables (Haupt and Haupt, 2004).

*Step*2 *(Fitness function evaluation and ranking)*: Fitnees score for each chromosome set of the total population is evaluated with the help of the ANN model and based upon the fitness scores the chromosome sets are arranged in descending order (for maximization).

*Step*3 (*Selection*): The selection rate, X_{rate}, is the fraction of N_{pop} that survives for the next step of mating and the bottom N_{pop} - N_{keep} is discarded to make room for the new offspring. Deciding how many chromosomes to keep is somewhat arbitrary. Letting only a few chromosomes survive to the next generation limits the available genes in the offspring. And keeping too many chromosomes allows bad performers a chance to contribute their traits to the next generation. Often 50% ($X_{rate} = 0.5$) is kept in the natural selection process (Haupt and Haupt, 2004).

$$N_{keep} = X_{rate} \times N_{pop} \tag{3.9}$$

where N_{keep} is the number of chromosomes that are kept each generation.

*Step*4 (*Rank weighting of the Chromosomes):* From the rank (a) of the N_{keep} chromosome set probabilities are assigned to them in the mating pool along with their cumulative, which are directly proportional to their fitness score. A chromosome with the highest fitness score has the highest probability of mating, while the chromosome with the lowest fitness score has the lowest probability of mating. A random number determines which chromosome is to be selected. For instance, if the random number is $r = 0.577$, then $0.4 < r < 0.7$, so $chromosome_1$ is selected. This type of weighting is often referred to as *roulette wheel weighting* (Haupt and Haupt, 2004).

$$P_a = \frac{N_{keep} - a + 1}{\sum_{a=1}^{N_{keep}} a} \tag{3.10}$$

*Step*5 (*Pairing from top to bottom)*: Start at the top of the list pairing is done such that he mother has row numbers in the population matrix given by *ma* = 1, 3, 5, . . . and the father has the row numbers pa = 2, 4, 6, . . .

Step6 (Single Point Crossover Mating): A crossover point is randomly selected and then the variables across that point are merely swapped. For example purposes, consider the two parents to be:

$Parent_1 = [p_{m1}, p_{m2}, p_{m3}, p_{m4}, p_{m5,}\ldots\ldots, p_{mNvar}]$

$Parent_2 = [p_{d1}, p_{d2}, p_{d3}, p_{d4}, p_{d5,}\ldots\ldots, p_{dNvar}]$

Crossover point is randomly selected, and then the variables across that are exchanged:

$Offspring_1 = [p_{m1}, p_{m2}, p_{m3}, p_{d4}, p_{d5,}\ldots\ldots, p_{dNvar}]$

$Offspring_2 = [p_{d1}, p_{d2}, p_{d3}, p_{m4}, p_{m5}\ldots\ldots, p_{mNvar}]$

Step7 (Mutation): Mutation rate is taken as 0.2 i.e. 20% of the population is to be mutated. Mutation rate is multiplied by the total number of variables that can be mutated in the population giving total number of mutations. Next random numbers are chosen to select the row and columns of the variables to be mutated. A mutated variable is replaced by a new random variable in between 0.1 and 0.9.

*Step*8 (*Re-selection)*: Again, the fitness score or the output, for each chromosome set of the total newly generated population is evaluated with the help of the ANN model and are sorted in descending order based on their fitness score.

Step9 (convergence): With this newly generated population as initial population steps from 3 to 8 are repeated until maximum number of generations is up to fixed tolerance, or the conversion has reached up to the maximum conversion taken. After maximum number of generations best solutions are displayed, which are the best optimal solutions.

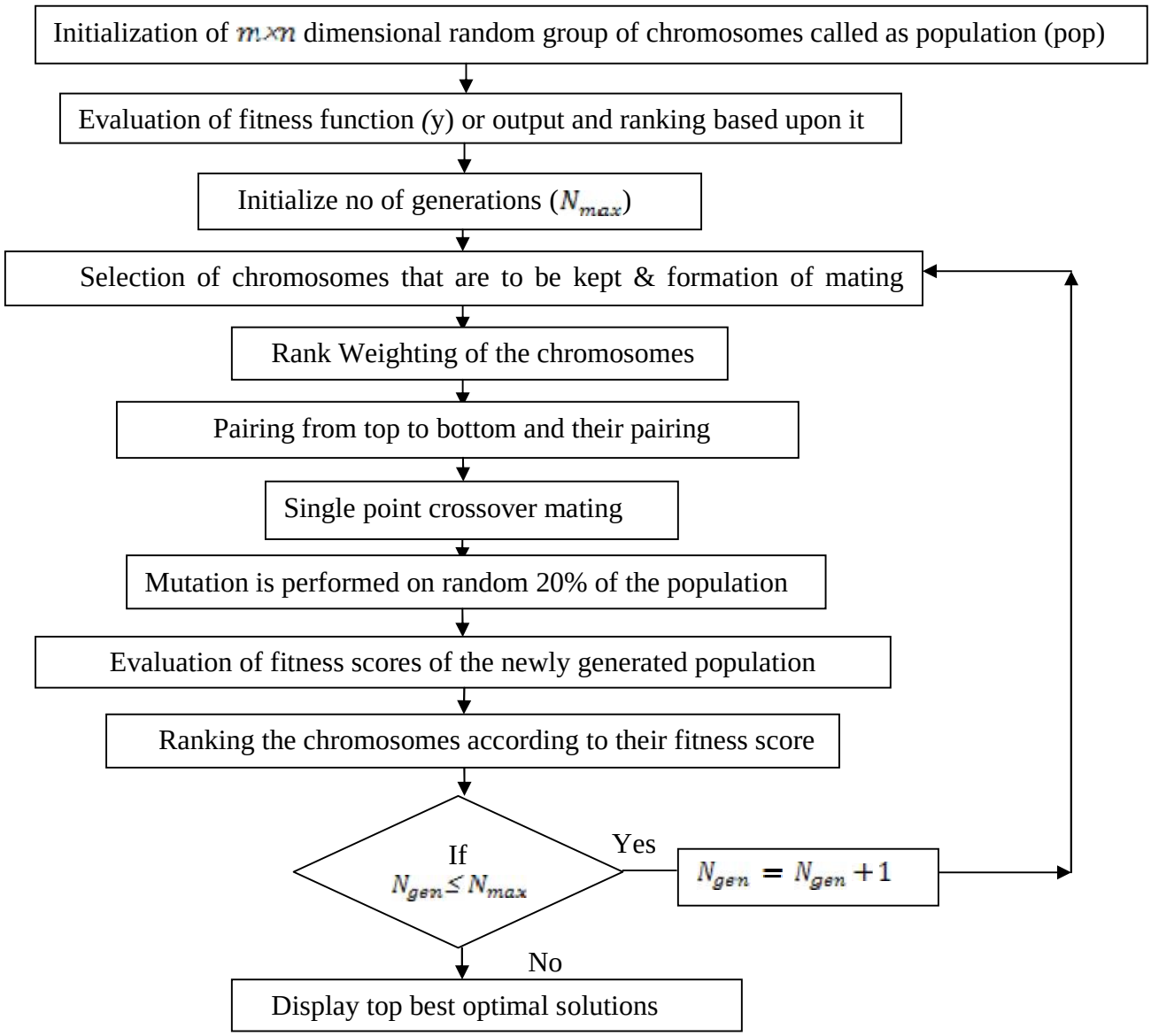


Fig. 3.5 Flowchart of genetic algorithms for unconstrained optimization problems

Summary of Chapter- 3

Section 1 of this chapter shows the results of the sensitivity analysis when it was carried out on all the 73 data points. It was found out that Fe^{3+} concentration was the least significant variable among all other process variables and parameters and was thus excluded from the further analysis part.

For the ANN Model in section 2 five most influential parameters were taken as inputs and H_2S conversion was taken as output. Four layered ANN model was made which consists of one input layer, two hidden layer and one output layer. The error for the training and test section came out to be nearly equal and correlation coefficient for the training and test set were also very high showing that a generalized model is built.

In section 3 based upon the principles of clonal selection theory, clonal selection algorithm (CLONALG) was made for the unconstrained optimization problems and with slight modification for the constrained optimization problems. And it was applied to maximize the H_2S conversion (%) value.

The last section genetic algorithms for the unconstrained optimization problems is developed and it was also used to maximize the value of H_2S conversion (%).

4. RESULTS OBTAINED AND DISCUSSION

USING the MLP-based model, process optimization was conducted by utilizing CLONALG and GA separately. Specifically, the optimized values of five process variables and parameters namely H_2S concentration (ppm) (x_1), reactor pressure (bar) (x_2), reactor temperature (^oC) (x_3), stirring speed of the impeller (rpm) (x_4) and reaction time (min) (x_5) were obtained such that H_2S conversion (%) is maximized. The results of AIS and GA-based optimizations are shown in Table 4.1. Also, Figures 4.1 and 4.2 respectively display the generation-wise evolution of the best candidate solutions found by the CLONALG and GA.

Table 4.1 AIS and GA based optimized results with their experimental validation

	Experimentally Verified Results	**AIS Optimized Solutions**	**GA Optimized Solutions**
H_2S Conc. (x1) (ppm)	33267.71	33390	27865.625
Pressure (x2) (bar)	4.07	4.10085	4.355
Temp (x3) (^oC)	22.80	25.015	25.65
Stirring Speed (x4) (rpm)	104.00	104.45	118.2
Reaction Time (x5) (min)	19.00	16.83	16.59
Fe^{+3} Conc. (ppm)	5000	5000	5000
H_2S conversion (%)	97.13	97.69	97.30
Number of Generations to reach convergence		1080	2432
Number of Function Evaluations till convergence		3240	4864

The optimal process conditions searched by the CLONALG and GA correspond to the Fe^{+3} concentration of 5000 ppm. It can be observed from Table 4.1, Fig. 4.1 and Fig. 4.2 that both CLONALG and GA have converged to the optimal solutions leading to closely matching maximized values of 97.69% and 97.30% of H_2S conversion, respectively. A comparison of the AIS and GA-based optimal solutions shows that except *H_2S concentration*, the optimal values of the remaining four process condition variables and parameters are nearly same. The optimal values of AIS and GA-based H_2S concentration however differ by ≈ 1690. This indicates that there exist two sets of process conditions that lead to nearly identical H_2S conversion magnitudes. A comparison of the AIS and GA-based optimal solutions suggest that the AIS-based solution is better (albeit marginally) in terms of lower magnitudes of the pressure and temperature and therefore it was chosen for the experimental validation. This validation resulted in 97.1% H_2S conversion thus exhibiting good match vis-à-vis AIS-based optimal solutions. In so far as the efficiency of the two algorithms is concerned, the number of generations required by the CLONALG to reach convergence to the optimal solution is significantly lower (1080) than that required by the GA (2432). The number of function evaluations to convergence is also lower in the case of CLONALG (3240), when compared with the GA (4864). It is thus seen that application of ANN-AIS hybrid modeling- optimization has resulted in nearly 6.12% improvement over the best H_2S conversion of 91.5% obtained in non-optimized experiments.

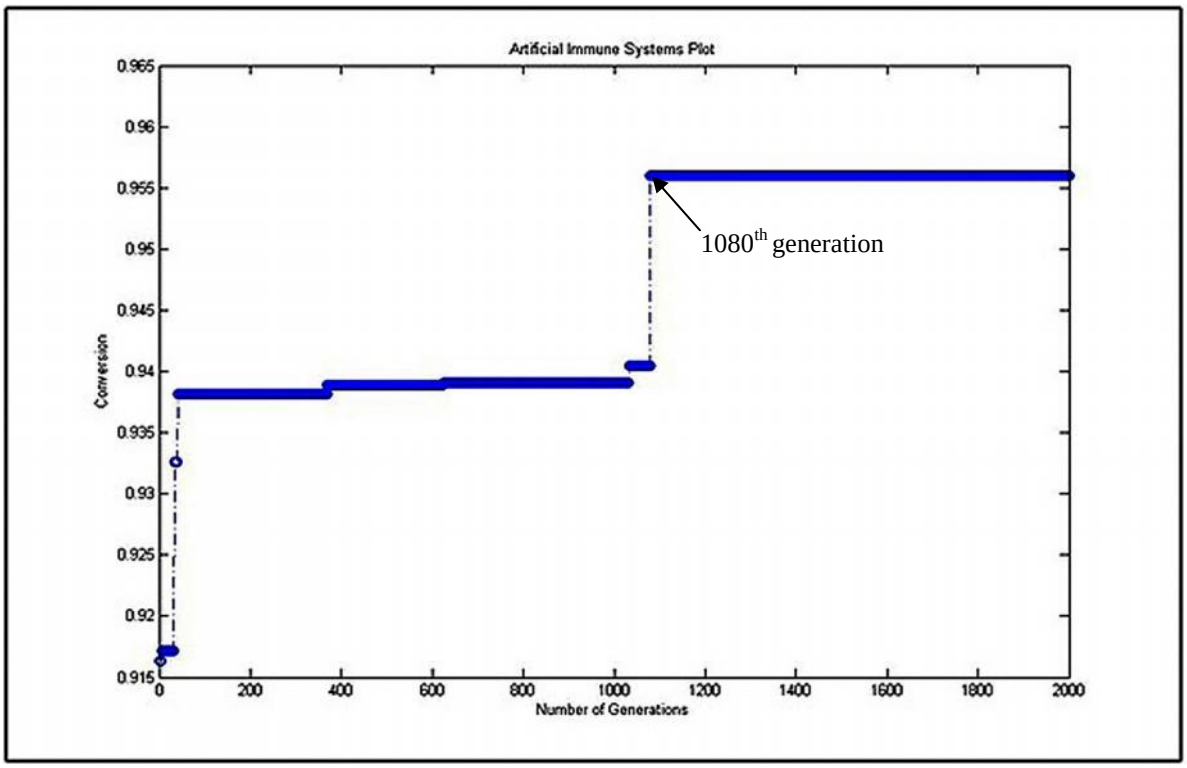


Fig. 4.1 Generatio-wise evolution of the best candidate solution (antibody) found by the CLONALG algorithm

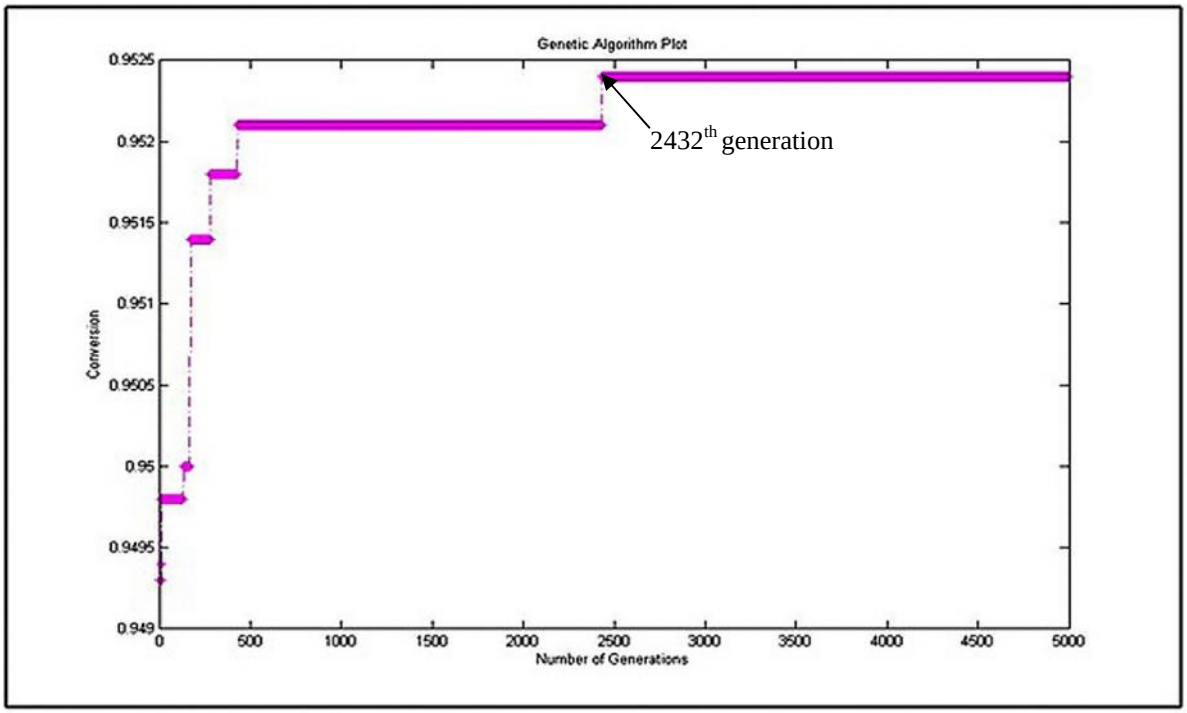


Fig. 4.2 Generation-wise evolution of the best candidate solution (antibody) found by the genetic algorithms

5. CONCLUSION & FUTURE WORK

5.1 Conclusion

THIS thesis reports a new artificial intelligence based robust and nonlinear stochastic optimization approach called as artificial immune systems which is inspired from the biological immune system. Particularly clonal selection algorithm which is a computational model of AIS has been developed as an unconstrained and constrained optimization tool. Further, for the modeling and optimization of the batch reactor process for the abatement of H_2S gas, the most influential process operating variables and parameters were evaluated through sensitivity analysis and process modeling, optimization were performed using artificial intelligence based hybrid strategies in grating ANN-AIS and ANN-GA methods with a view of maximizing H_2S conversion (%). In the implementation of this strategy, a process model was developed using an ANN and following which the input space of that model — representing significant process operating variables and parameters were optimized using both AIS and GA. The optimized sets of process conditions obtained using the hybrid ANN-AIS strategy were found to be more accurate, took significantly less number of generations and function evaluations to converge to the best solution. The obtained set of optimized process conditions through AIS has resulted in 6.12% improvement in the H_2S conversion over the best set of non-optimized process conditions.

5.2 Future Work

A number of strategies are available for process optimization. These strategies fall under two main categories namely, deterministic and stochastic. Most of the deterministic optimization strategies are gradient based and require objective function to be smooth, differentiable and continuous. In many instances these conditions are not satisfied for example when the objective function is represented in the form of ANN or other types of data driven models. In such a situation stochastic optimization methods are well suited. In the recent years a number of AI based stochastic optimization methods are proposed and successfully employed for industrial processes. These AI based strategies include genetic algorithms, evolutionary algorithms, memetic algorithms, ant colony and particle swarm. Recently, a novel AI based stochastic method namely artificial immune systems has been proposed. It is formed on the principles of biological immune system. However it has not been widely explored for solving optimization problems in chemical engineering and technology. Despite of its novelty AIS has many attractive features as demonstrated in this thesis. It is therefore important to exploit AIS for a wide variety of problems encountered in chemical engineering and technology. In the present study we have demonstrated CLONALG for optimizing chemical engineering problem. The other two AI based methods namely negative selection algorithm (NSA) and positive selection algorithm (PSA) are still needed to be explored for solving various problems in chemical engineering and technology. NSA can be used for solving fault detection and diagnosis and pattern recognition problems whereas PSA can be applied for clustering, and in other domains.

REFERENCES

[1] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Washington DC, 1961.

[2] D. Rumelhart, G. Hinton and R. Williams, "*Learning representations by back propagating errors*", Nature, pp 323, 1986.

[3] G. L . Ada and G. J. V. Nossal, "*The Clonal Selection Theory*", Scientific American, vol. 2, pp 50-57, 1987.

[4] D. Goldberg, *Genetic Algorithms in search, optimization, and machine Learning*, 1st ed. Addison-Wesley, New York, 1989.

[5] L. Davis, *Handbook of genetic algorithms*, 1st ed. Van Nostrand Reinhold, New York, 1991.

[6] K. Deb and D. E. Goldberg, *A comparison of selection schemes used in genetic algorithms. Foundations of genetic algorithms*, 1st ed. Morgan Kuafmann Publications, San Mateo, California, pp 69-93, 1991.

[7] J. A. Freeman and D. M. Skapura, "*Neural networks: algorithms, applications, and programming techniques*", Addison-Wesley, Reading, MA, 1991.

[8] C. B. Lucasius and G. Kateman, "*Understanding and using genetic algorithms. Part I. Concepts, properties and context*", Chem Intell Lab Syst., vol 19, pp 1, 1993.

[9] C. B. Lucasius and G. Kateman, "*Understanding and using genetic algorithms. Part II. Representation, configuration and hybridization*", Chem Intell Lab Syst., vol 25, pp 99, 1994.

[10] S. Forrest, A. S. Perelson, L. Allen and R. Cherukuri, *Self-nonself discrimination in a computer*, Proceedings of IEEE symposium on research in security and privacy, pp 202-212, 1994.

[11] C. M., Bishop, *Neural Networks for pattern recognition*, 1st ed. Oxford University Press. New York, 1995.

[12] S. Lek, A. Belaud, P. Baran, I. Dimopoulos and M. Delacoste, "*Role of some environmental variables in trout abundance models using neural networks*", Aquat. Living Resour., vol 9, pp 23-29, 1996.

[13] S. S. Tambe, B. D. Kulkarni and P. B. Deshpande, "*Elements of artificial neural network with selective applications in chemical and biological sciences*", Simulation and Advance Control Inc, 1996.

[14] V. Venkatasubramanian and A. Sundaram, Genetic *algorithms: introduction and applications*, In: Encyclopaedia of computational chemistry, Wiley, Chichester, UK, 1998.

[15] S. Haykin, *Neural networks and learning machines*, 3rd ed. Pearson Prentice Hall, New York, 1999.

[16] D. Dasgupta and L. F. Nino, *A comparison of negative and positive selection algorithms in novel pattern detection*, Proceedings of IEEE international conference on systems, man & cyberbetics, pp 125- 130, 2000.

[17] S. A. Hofmeyr and S. Forrest, "*Architecture for an artificial immune system", Evolutionary Computation*, vol 7(1), 45-68, 2000.

[18] L. N. de Castro and F. J. Zuben, *Learning and optimization using clonal selection principle. IEEE transactions on evolutionary computation*, special issue on artificial immune systems, vol 6, pp 239-251, 2001.

[19] S. Nandi, S. Ghosh, S. S. Tambe and B. D. Kulkarni, "*Artificial neural-network assisted stochastic process optimization strategies*", AIChE J, vol 47, pp 126-141 , 2001.

[20] S. P. Ramanathan, S. Mukherjee, R. K. Dahule, S. Ghosh, I. Rahman and S. S. Tambe, "*Optimization of continuous distillation columns using stochastic optimization approaches*", Trans Inst Chem Eng., vol 79, pp 310, 2001.

[21] S. Nandi, P. Mukherjee, S. S. Tambe, R. Kumar and B. D. Kulkarni, "*Reaction Modeling and Optimization Using Neural Networks and Genetic Algorithms: Case Study Involving TS-1-Catalyzed Hydroxylation of Benzene*", Ind. Chem. Eng., vol 41, pp 2159- 2169, 2002.

[22] L. N. de Castro and J. Timmis, *Artificial immune systems: A new computational intelligence approach*, 1st ed. Spinger. 67- 84, 2002.

[23] J. J. S. Cheema, N. V. Sankpal, S. S. Tambe, and B. D. Kulkarni, "*Genetic programming assisted stochastic optimization strategies for optimization of glucose to gluconic acid fermentation*", Biotechnol Prog., vol 18, pp 1356, 2002.

[24] V.S. Sumanwar, V. K. Jayaraman, B. D. Kulkarni, H. S. Kusumakar, K. Gupta and J. Rajesh, "*Solution of constrained optimization problems by multi objective genetic algorithms*", Comp Chem Eng., vol 26, pp 1481, 2002.

[25] F. Baishan, C. Hongwen, X. Xiaolan, W. Ning and H. Zongding, "*Using genetic algorithms coupling neural networks in a study of xylitol production: medium optimization*", Process Biochem., pp 979, 2003.

[26] A. W. Jennifer and S. M. Garrett, *Improved pattern recognition with artificial clonal selection ?*. Proceedings of artificial immune systems: Second international conference, ICARIS, pp 181- 193, 2003.

[27] J. Kelsey and J. Timmis, *Immune Inspired Somatic Contiguous Hyper mutation for Function Optimization*, In: Genetic and Evolutionary Computation Conference - GECCO 2003, JUL 12-16, Chicago, Illinois, 2003.

[28] R. Collobert and S. Bengio, *Links between Perceptrons, MLPs and SVMs*, Proceedings Int'l Conf. on Machine Learning (ICML), 2004.

[29] R. L. Haupt and S. E. Haupt, *Practical genetic algorithms*, 2nd ed. John Wiley & Sons, Inc., New Jersey, 2004.

[30] H. H. Nguyan and C. W. Chan, *A Comparison of Data Preprocessing Strategies for Neural Network Modeling of Oil Production Prediction*, In proceeding: ICCI '04 Proceedings of the Third IEEE International Conference on Cognitive Informatics, 2004.

[31] J. D. Olden, M. K. Joy and R. G. Death, "*An accurate comparison of methods for quantifying variable importance in artificial neural networks using stimulated data*", Ecological Modeling, vol. 178, pp 387-389, 2004.

[32] R. Younsi and W. Wang, *A new artificial immune system algorithm for clustering*, Proceedings of 5th international conference, Exeter, UK, pp 58- 64, August 25-27 2004.

[33] T. P. Runarsson and Xin Yao,"*Stochastic Ranking for Constrained Evolutionary Optimization ",* IEEE Transactions on Evolutionary Computation, Vol 4, No 3, 2005.

[34] J. Brownlee, "*Clonal selection theory & CLONALG*", Technical Report No 2-02, 2005.

[35] E. K . Burke and G. Kendall, *Artificial immune systems by Aickelin, U., Dasgupta, D. search methodologies: Introductory tutorials in optimization and decision support techniques*, 1st ed. Springer, 2005.

[36] A. Secker, A. A. Freitas and J. Timmis, "*Towards a danger theory inspired artificial immune system for web mining*", Web Mining: applications and techniques, Idea Group, pp. 145-168, 2005.

[37] R. S. Nowakowski, *Stable neuron numbers from cradle to grave, Proceedings of the National Academy of Sciences*, 103 (33): 12219, 2006.

[38] C. A. Coello Coello, S. C. Esquivel and V. S. Argon, "*Artificial immune system for solving constrained optimization problems*", Intelligencia Artificial. Revista lberoamericana de Inteligencia Artificial, 14, 46, pp 55-66, 2007.

[39] J. Brownlee, "*Clonal selection algorithms*", CIS Technical Report 070209A, 2007.

[40] S. Chitra and R. Rajaram, *A Software Reliability Estimation Tool Using Artificial Immune Recognition System*, Proceedings of the International Multi Conference of Engineers and Computer Scientists, Hong Kong, Vol 1, 2008.

[41] K. Murphy, *The Development and Survival of Lymphocytes*, Immuno Biology, 8th ed. New York, 2008.

[42] Y. Zhong, L. Zhang, P. Li and X. Huang, "*An Artificial Immune Algorithm For Spectrum Recognition of Hyper spectral Data*", The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Beijing, Vol.37, B7, 2008.

[43] A. S. Deshpande, "In- situ chemo- bioremediation for gas clean up", Ph.D. thesis, University of Pune, Pune, India, Mar. 2009.

[44] D. Dasgupta and L. F. Nino, *Immunological computation*, 1st ed. Auerbach Publications, Taylor & Francis Group, Boca Raton, 2009.

[45] A. Secker, M. N. Davies, A. A. Freitas, J. Timmis, E. Clark and D. R. Flower, "*An Artificial Immune System for Clustering Amino Acids in the Context of Protein Function Classification*", J Math Model Algor., vol 8, pp 103–123, 2009.

[46] V. Selvi and R. Umarani, "*Comparative study of ant colony and particle swarm optimization techniques*", International journal of computer applications, vol 5, pp 975- 981, 2010.

[47] K. A. Al-Sheshtawi, H.M. Abdul- Kader and N. A. Ismail, "*Artificial immune clonal selection algorithms: A comparative study of CLONALG, opt-IA, and BCA with numerical optimization problems*", International journal of computer science and network security, vol 10, pp 4-10, 2010.

[48] H. Chen, S. Gao, S. Li and T. Zeng, "*A Probabilistic modeling clonal selection algorithm and its application to travelling salesman problems*", International journal of computer science and network security, vol 10, pp 6, 2010.

[49] S. Dixon and X. Yu, *Bioinformatics Data Mining using Artificial Immune Systems and Artificial Neural Networks,* Information and Automation (ICIA), 2010 IEEE International Conference, Harbin, pp 440- 445, 2010.

[50] G. Kaushik and S. Rajagopalan, "*Fault detection and diagnosis of chemical process by an Immune system inspired approach*", AICHEJ, 2010.

[51] O. Kilic and Q. M. Nguyen, *Application of Artificial Immune System Algorithm to Electro magnetics Problem*, Progress In Electromagnetics Research, vol B- 20, pp 1-17, 2010.

[52] P. Rao, R. Challoo, S. Ozcelik and L. Challoo, "*Navigation control and Path Mapping of a Mobile Robot using Artificial Immune Systems*", International Journal of Robotics and Automation, Vol 1, Issue 1, 2010.

[53] H. Zhang, Ze Zhang and L.H. Lan, "*Evolutionary optimization of fed batch penicillin fermentation process*", International Symposium on Computer, Communication, Control and Automation, Taiwan, pp 403, 2010.

[54] H. Maier, R. May and G. Dandy, "*Review of Input Variable Selection Methods for Artificial Neural Networks, Artificial Neural Networks - Methodological Advances and Biomedical Application*", ISBN: 978-953-307-243-2, 2011.

[55] P. Matzinger, *The Danger Model in Its Historical Context. Scandinavian Journal of Immunology*, vol 54, pp 4-9, 2011.

[56] Y. Özçelik, "*Exergetic Optimization of Distillation Sequences using a Genetic Based Algorithm*", Journal of Thermal Science and Technology, 2011.

[57] A. Malhotra, A. Baheti, S. Gupta, "*Pattern recognition approaches inspired by artificial immune system*", International journal of computer applications, vol 44, Issue 20, pp 12-16, 2012.

[58] T. Slavov and R. Olympia, "*Application of Genetic Algorithm to tuning a PID controller for Glucose Concentration Control*", Wseas Transactions on systems, Issue 7, Vol 11, 2012.

APPENDIX I

H_2S Reaction Data used in Sensitivity Analysis

Run No.	H_2S Conc. (ppm)	Reactor pressure (bar)	Reactor temperature (oC)	Stirring Speed (rpm)	Fe^{3+} Conc. (ppm)	Reaction Time (min)	H_2S conversion
1	16212.87	2.09	40.71	200	5000	15	87.31
2	16212.87	2.12	40.11	200	5000	15	88.27
3	38727.27	2.06	39.94	200	5000	15	90.46
4	38727.27	2.03	40.01	200	5000	15	90.2
5	16212.87	4.07	40.25	200	5000	15	90.21
6	16212.87	4.01	40.23	200	5000	15	91.03
7	16212.87	2.1	69.2	200	5000	15	74.53
8	16212.87	2.07	69.26	200	5000	15	72.77
9	38727.27	2.11	69.35	200	5000	15	85.71
10	38727.27	2.08	69.32	200	5000	15	85.16
11	16212.87	4.1	69.96	200	5000	15	75.32
12	16212.87	4.16	69.77	200	5000	15	75.12
13	38727.27	4.05	69.37	200	5000	15	90.38
14	38727.27	4	69.45	200	5000	15	90.79
15	16296.3	2.12	39.57	400	5000	15	83.46
16	16296.3	2.15	39.52	400	5000	15	84.35
17	38740.46	2.05	39.17	400	5000	15	80.65
18	38740.46	2.03	39.12	400	5000	15	80.83
19	16296.3	4.32	40.46	400	5000	15	76.2
20	16296.3	4.26	40.47	400	5000	15	75.54
21	38745.39	4.05	39.63	400	5000	15	82.36
22	38745.39	4	39.65	400	5000	15	84.35
23	16296.3	2.08	69.38	400	5000	15	82.64
24	16296.3	2.05	69.33	400	5000	15	84.05
25	38740.46	2.1	69.48	400	5000	15	82.55
26	38740.46	2.07	69.52	400	5000	15	83.32
27	16296.3	4.36	69.74	400	5000	15	74.36
28	16296.3	4.3	69.82	400	5000	15	74.02
29	38740.46	4.14	70.16	400	5000	15	85.82

30	38740.46	4.09	70.1	400	5000	15	86.13
31	5000	3.08	54.66	300	5000	15	72.98
32	5000	3.04	54.6	300	5000	15	74.3
33	50000	3.03	54.94	300	5000	15	80.27
34	50000	3	55.07	300	5000	15	82
35	27772	1.03	54.86	300	5000	15	82.2
36	27772	1.02	54.96	300	5000	15	83.15
37	27428.57	5.19	55.17	300	5000	15	89.46
38	27428.57	5.12	55.11	300	5000	15	89.2
39	27500	3.02	30.17	300	5000	15	90.97
40	27500	2.98	30.11	300	5000	15	90.41
41	27428.57	3.09	85.17	300	5000	15	80.39
42	27428.57	3.05	85.22	300	5000	15	80.67
43	27428.57	3.03	54.73	100	5000	15	91.5
44	27428.57	3.08	54.83	500	5000	15	72.03
45	27428.57	3.04	55.17	500	5000	15	73.28
46	27428.57	3.05	54.71	300	5000	15	82.74
47	27428.57	3.01	54.75	300	5000	15	82.35
48	27428.57	3.08	55.15	300	5000	15	82.66
49	27428.57	3.17	55.38	300	5000	15	82.62
50	27428.57	3.05	54.71	300	5000	15	82.74
51	27428.57	3.01	54.75	300	5000	15	82.35
52	27428.57	3.05	54.71	300	5000	15	82.74
53	27428.57	3.01	54.75	300	5000	15	82.35
54	27428.57	3.08	55.15	300	5000	15	82.66
55	27428.57	3.17	55.38	300	5000	15	82.62
56	27428.57	3.1	55.08	300	5000	15	82.44
57	27428.57	3.06	55.87	300	5000	15	82.52
58	50000	5.0217	25	500	5000	15	87.22
59	50000	5.08755	25	500	5000	15	86.86
60	50000	5.06055	85	100	1000	60	82.3
61	50000	5.1057	85	100	1000	60	80.84
62	5000	5.05185	85	500	1000	15	82.81
63	5000	5.04945	85	500	1000	15	80.73
64	5000	1.09605	85	500	5000	60	54.56
65	5000	1.01	85	500	5000	60	55
66	50000	1.13445	25	500	1000	60	87.51

67	50000	1.08585	25	500	1000	60	81.55
68	5000	5.2053	25	100	5000	60	49.47
69	5000	5.001	25	100	5000	60	48.98
70	50000	1.2135	85	100	5000	15	58.07
71	50000	1.10145	85	100	5000	15	54.89
72	5000	1.0506	25	100	1000	15	2.96
73	5000	1.05495	25	100	1000	15	3.45

MATLAB code for optimization of the batch reactor process for the abatement of H_2S gas

I. Main program

```
function main program
global pop M r B xmax xmin n p beta net1h1 net1h11 net1h12 net1h13 net1h14 net1h15 net1h16 out1h1 out2h1 out3h1
out4h1 out5h1 out6h1 net2h2 net2h21 net2h22 net2h23 net2h24 net2h25 net2h26 g out1h2 out2h2 out3h2 out4h2
out5h2 out6h2 net1h1r net1h16r net2h26r  net1h11r net1h12r net1h13r net1h14r net1h15r out1h1r out2h1r out3h1r
out4h1r out5h1r out6h1r net2h2r net2h21r net2h22r net2h23r net2h24r net2h25r gr  out1h2r out2h2r out3h2r out4h2r
out5h2r out6h2r net1h17 out7h2 out7h1 net1h17r out7h2r out7h1r net2h27r Z Y S K U V X XY

pop= 65; M= 49; r= 16; % In the given population 80% are taken as memory cells and rest as replaceable cells
xmax=0.9; xmin= 0.1; n= 5; p= 4; beta= 1;
Nmax= 2000;
W= [-0.181 -0.474 -0.355 -0.433 0.108; 0.117 0.138 -0.334 0.036 -0.459;  1.108 0.435 0.178 1.022 -0.357;   0.241 -0.163 -0.385
0.702 -0.080; -0.353 0.865 0.345 0.823 -0.282; -0.202 0.522 0.047 -0.025 -0.268; -0.166 -0.549 0.483 -0.234 -0.230];

net1h11= zeros(65,1); net1h12= zeros(65,1); net1h13= zeros(65,1); net1h14= zeros(65,1); net1h15= zeros(65,1); net1h16=
zeros(65,1); net1h17= zeros(65,1);

%%% Initializing the antibodies %%%
 rand('seed',20); % It is used so that it generates the same random solution when iterated several times.
x = xmin + (xmax-xmin)*rand(pop,n);

%%% Evaluation of antibodies %%%
for k= 1:pop       %Elements present in the rows of x matrix
   net1h1(k,1)= W(1,1)*x(k,1) + W(1,2)*x(k,2) + W(1,3)*x(k,3) + W(1,4)*x(k,4) + W(1,5)*x(k,5)+ 0.125;
   net1h11(k,1) = net1h11(k,1) + net1h1(k,1);
   net1h11(k,1) = net1h1(k,1);
end

for k= 1:pop       %Elements present in the rows of x matrix
   net1h1(k,1)= W(2,1)*x(k,1) + W(2,2)*x(k,2) + W(2,3)*x(k,3) + W(2,4)*x(k,4) + W(2,5)*x(k,5)+ 0.174;
   net1h12(k,1) = net1h12(k,1) + net1h1(k,1);
   net1h12(k,1) = net1h1(k,1);
end

for k= 1:pop       %Elements present in the rows of x matrix
   net1h1(k,1)= W(3,1)*x(k,1) + W(3,2)*x(k,2) + W(3,3)*x(k,3) + W(3,4)*x(k,4) + W(3,5)*x(k,5)+ 0.063;
   net1h13(k,1) = net1h13(k,1) + net1h1(k,1);
   net1h13(k,1) = net1h1(k,1);
```

```
end

for k= 1:pop       %Elements present in the rows of x matrix
   net1h1(k,1)= W(4,1)*x(k,1) + W(4,2)*x(k,2) + W(4,3)*x(k,3) + W(4,4)*x(k,4) + W(4,5)*x(k,5)+ 0.332;
   net1h14(k,1) = net1h14(k,1) + net1h1(k,1);
   net1h14(k,1) = net1h1(k,1);
end

for k= 1:pop       %Elements present in the rows of x matrix
   net1h1(k,1)= W(5,1)*x(k,1) + W(5,2)*x(k,2) + W(5,3)*x(k,3) + W(5,4)*x(k,4) + W(5,5)*x(k,5)- 0.328;
   net1h15(k,1) = net1h15(k,1) + net1h1(k,1);
   net1h15(k,1) = net1h1(k,1);
end

for k= 1:pop       %Elements present in the rows of x matrix
   net1h1(k,1)= W(6,1)*x(k,1) + W(6,2)*x(k,2) + W(6,3)*x(k,3) + W(6,4)*x(k,4) + W(6,5)*x(k,5)+ 0.124;
   net1h16(k,1) = net1h16(k,1) + net1h1(k,1);
   net1h16(k,1) = net1h1(k,1);
end
for k= 1:pop       %Elements present in the rows of x matrix
   net1h1(k,1)= W(7,1)*x(k,1) + W(7,2)*x(k,2) + W(7,3)*x(k,3) + W(7,4)*x(k,4) + W(7,5)*x(k,5) - 0.100;
   net1h17(k,1) = net1h17(k,1) + net1h1(k,1);
   net1h17(k,1) = net1h1(k,1);
end

for i = 1:pop
   out1h1(i,1)= tanh(net1h11(i,1));  out2h1(i,1)= tanh(net1h12(i,1));  out3h1(i,1)= tanh(net1h13(i,1)); out4h1(i,1)= tanh(net1h14(i,1)); out5h1(i,1)= tanh(net1h15(i,1)); out6h1(i,1)= tanh(net1h16(i,1)); (i,1)= tanh(net1h17(i,1));
end
q= [out1h1 out2h1 out3h1 out4h1 out5h1 out6h1 out7h1];

net2h21= zeros(65,1); net2h22= zeros(65,1); net2h23= zeros(65,1); net2h24= zeros(65,1); net2h25= zeros(65,1); net2h26= zeros(65,1); net2h27= zeros(65,1);

W1= [0.143 -0.199 0.184 -0.163 -0.158 -0.223 -0.125; -0.366 -0.085 0.608 -0.474 -0.437 0.407 -0.026; -0.163 -0.204 0.198 -0.409 0.160 -0.214 -0.127; -0.142 0.463 0.287 -0.060 -0.045 -0.008 -0.196; 0.579 0.138 0.309 -0.429 -0.193 -0.264 -0.009; 0.034 0.419 0.506 0.238 0.191 0.246 0.205; -0.731 0.203 0.563 0.613 1.036 0.362 -0.565];

for k= 1:pop       %Elements present in the rows of x matrix
 net2h2(k,1)= W1(1,1)*q(k,1) + W1(1,2)*q(k,2) + W1(1,3)*q(k,3)+ W1(1,4)*q(k,4)+ W1(1,5)*q(k,5)+ W1(1,6)*q(k,6)+ W1(1,7)*q(k,7)- 0.333;
   net2h21(k,1) = net2h21(k,1) + net2h2(k,1);
   net2h21(k,1) = net2h2(k,1);
end

for k= 1:pop       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(2,1)*q(k,1) + W1(2,2)*q(k,2) + W1(2,3)*q(k,3)+ W1(2,4)*q(k,4)+ W1(2,5)*q(k,5)+ W1(2,6)*q(k,6)+ W1(2,7)*q(k,7)+ 0.062;
   net2h22(k,1) = net2h22(k,1) + net2h2(k,1);
   net2h22(k,1) = net2h2(k,1);
end

for k= 1:pop       %Elements present in the rows of x matrix
```

```
   net2h2(k,1)= W1(3,1)*q(k,1) + W1(3,2)*q(k,2) + W1(3,3)*q(k,3)+ W1(3,4)*q(k,4)+ W1(3,5)*q(k,5)+ W1(3,6)*q(k,6)+
W1(3,7)*q(k,7)- 0.159;
   net2h23(k,1) = net2h23(k,1) + net2h2(k,1);
   net2h23(k,1) = net2h2(k,1);
end
for k= 1:pop       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(4,1)*q(k,1) + W1(4,2)*q(k,2) + W1(4,3)*q(k,3)+ W1(4,4)*q(k,4)+ W1(4,5)*q(k,5)+ W1(4,6)*q(k,6)+
W1(4,7)*q(k,7)+ 0.035;
   net2h24(k,1) = net2h24(k,1) + net2h2(k,1);
   net2h24(k,1) = net2h2(k,1);
end

for k= 1:pop       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(5,1)*q(k,1) + W1(5,2)*q(k,2) + W1(5,3)*q(k,3)+ W1(5,4)*q(k,4)+ W1(5,5)*q(k,5)+ W1(5,6)*q(k,6)+
W1(5,7)*q(k,7)- 0.105;
   net2h25(k,1) = net2h25(k,1) + net2h2(k,1);
   net2h25(k,1) = net2h2(k,1);
end

for k= 1:pop       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(6,1)*q(k,1) + W1(6,2)*q(k,2) + W1(6,3)*q(k,3)+ W1(6,4)*q(k,4)+ W1(6,5)*q(k,5)+ W1(6,6)*q(k,6)+
W1(6,7)*q(k,7)- 0.096;
   net2h26(k,1) = net2h26(k,1) + net2h2(k,1);
   net2h26(k,1) = net2h2(k,1);
end

for k= 1:pop       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(7,1)*q(k,1) + W1(7,2)*q(k,2) + W1(7,3)*q(k,3)+ W1(7,4)*q(k,4)+ W1(7,5)*q(k,5)+ W1(7,6)*q(k,6)+
W1(7,7)*q(k,7)- 0.235;
   net2h27(k,1) = net2h27(k,1) + net2h2(k,1);
   net2h27(k,1) = net2h2(k,1);
end

for i = 1:pop
   out1h2(i,1)= tanh(net2h21(i,1)); out2h2(i,1)= tanh(net2h22(i,1)); out3h2(i,1)= tanh(net2h23(i,1)); out4h2(i,1)=
tanh(net2h24(i,1)); out5h2(i,1)= tanh(net2h25(i,1)); out6h2(i,1)= tanh(net2h26(i,1)); out7h2(i,1)= tanh(net2h27(i,1));
end
g= [out1h2 out2h2 out3h2 out4h2 out5h2 out6h2 out7h2];

% Evaluation of Fitness Function %

 f = equation1(x);  %it is a user defined function where the objective function value is evaluated for the entire
population of anti bodies , program II called here

 c = [f x];          % concatinating scores(fitness) and solutions(antibodies) to a single matrix
z = sortrows(c, -1);    % arranging the solutions based on the fitness scores. As it is a maximization problem the one
%which is having high fitness score will have low affinity and is placed at the top. We used matlab operator to
%arrange the solutions.

tic;   % this particular function evaluates the total time for executing algorithm

 %%% Iteration starts %%%
```

```
for gen= 1:1:Nmax
   if gen > 1
      z(1,:)= XY1(1,:);
   end
z;

 o = z(:,2:end); % It gives a Matrix of only solutions or Antibodies.

% Now we need to select the N best solutions from the above set for cloning and hyper-mutation.
  B= o(1:M,:);

for i= 1:M  %program III called here
   s = ['T',int2str(i),'=matrices1(i)'];
   eval(s);
end

% after cloning and mutation, the output we are getting is in the form T1,T2...TM and the size of each Ti is the
%number of clones generated for that particular i.

c2=vertcat(T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21,T22,T23,T24,T25,T26,T27,T2
8,T29,T30,T31,T32,T33,T34,T35,T36,T37,T38,T39,T40,T41,T42,T43,T44,T45,T46,T47,T48,T49,T50,T51,T52,T53,T54,T55,T
56);   % vercat is used to vertically concatenate all the solutions coming from the matrices.m file

c2new = sortrows(c2,-1);  % To arrange the solutions in the descending order according to their scores.

% Next step is to select the top best M solutions from the matrix c2new
B2 = c2new(1:M,:);
B2;  %This gives  the top M best solutions

%Recreating the initial population by replacing the best solutions in the initial poplation, the replacement is done
%based on the fitness scores.
for i = 1:1:M
     if B2(i,1) > z(i,1)
     B2(i,1:end) = z(i,1:end);
     end
end
 z;     %represents the newly generated  population after replacing the best M antibodies in the initial population

% Now to replace the highest affinty solutions (lowest fitness score) from the initial population by generating
%replacable cells.
Wr= [-0.181 -0.474 -0.355 -0.433 0.108; 0.117 0.138 -0.334 0.036 -0.459; 1.108 0.435 0.178 1.022 -0.357; 0.241 -0.163 -0.385
0.702 -0.080; -0.353 0.865 0.345 0.823 -0.282; -0.202 0.522 0.047 -0.025 -0.268;-0.166 -0.549 0.483 -0.234 -0.230];

net1h11r= zeros(65,1); net1h12r= zeros(65,1); net1h13r= zeros(65,1); net1h14r= zeros(65,1); net1h15r= zeros(65,1);
net1h16r= zeros(65,1); net1h17r= zeros(65,1);

%%% Initializing the antibodies %%%
h = xmin + (xmax-xmin)*rand(r,n);

%%% Evaluation of antibodies %%%
for k= 1:r       %Elements present in the rows of x matrix
   net1h1r(k,1)= Wr(1,1)*h(k,1) + Wr(1,2)*h(k,2) + Wr(1,3)*h(k,3) + Wr(1,4)*h(k,4) + Wr(1,5)*h(k,5) + 0.125;
   net1h11r(k,1) = net1h11r(k,1) + net1h1r(k,1);
```

```
   net1h11r(k,1) = net1h1r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net1h1r(k,1)= Wr(2,1)*h(k,1) + Wr(2,2)*h(k,2) + Wr(2,3)*h(k,3) + Wr(2,4)*h(k,4) + Wr(2,5)*h(k,5) + 0.174;
   net1h12r(k,1) = net1h12r(k,1) + net1h1r(k,1);
   net1h12r(k,1) = net1h1r(k,1);
end

for k= 1:r      %Elements present in the rows of x matrix
   net1h1r(k,1)= Wr(3,1)*h(k,1) + Wr(3,2)*h(k,2) + Wr(3,3)*h(k,3) + Wr(3,4)*h(k,4) + Wr(3,5)*h(k,5) + 0.063;
   net1h13r(k,1) = net1h13r(k,1) + net1h1r(k,1);
   net1h13r(k,1) = net1h1r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net1h1r(k,1)= Wr(4,1)*h(k,1) + Wr(4,2)*h(k,2) + Wr(4,3)*h(k,3) + Wr(4,4)*h(k,4) + Wr(4,5)*h(k,5)+ 0.332;
   net1h14r(k,1) = net1h14r(k,1) + net1h1r(k,1);
   net1h14r(k,1) = net1h1r(k,1);
end

for k= 1:r      %Elements present in the rows of x matrix
   net1h1r(k,1)= Wr(5,1)*h(k,1) + Wr(5,2)*h(k,2) + Wr(5,3)*h(k,3) + Wr(5,4)*h(k,4) + Wr(5,5)*h(k,5)- 0.328;
   net1h15r(k,1) = net1h15r(k,1) + net1h1r(k,1);
   net1h15r(k,1) = net1h1r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net1h1r(k,1)= Wr(6,1)*h(k,1) + Wr(6,2)*h(k,2) + Wr(6,3)*h(k,3) + Wr(6,4)*h(k,4) + Wr(6,5)*h(k,5)+ 0.124;
   net1h16r(k,1) = net1h16r(k,1) + net1h1r(k,1);
   net1h16r(k,1) = net1h1r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net1h1r(k,1)= Wr(7,1)*h(k,1) + Wr(7,2)*h(k,2) + Wr(7,3)*h(k,3) + Wr(7,4)*h(k,4) + Wr(7,5)*h(k,5)- 0.100;
   net1h17r(k,1) = net1h17r(k,1) + net1h1r(k,1);
   net1h17r(k,1) = net1h1r(k,1);
end

for i = 1:r
  out1h1r(i,1)= tanh(net1h11r(i,1)); out2h1r(i,1)= tanh(net1h12r(i,1)); out3h1r(i,1)= tanh(net1h13r(i,1)); out4h1r(i,1)= tanh(net1h14r(i,1)); out5h1r(i,1)= tanh(net1h15r(i,1)); out6h1r(i,1)= tanh(net1h16r(i,1)); out7h1r(i,1)= tanh(net1h17r(i,1));
end

qr= [out1h1r out2h1r out3h1r out4h1r out5h1r out6h1r out7h1r];

net2h21r= zeros(65,1); net2h22r= zeros(65,1); net2h23r= zeros(65,1); net2h24r= zeros(65,1); net2h25r= zeros(65,1); net2h26r= zeros(65,1); net2h27r= zeros(65,1);
W1r= [0.143 -0.199 0.184 -0.163 -0.158 -0.223 -0.125; -0.366 -0.085 0.608 -0.474 -0.437 0.407 -0.026; -0.163 -0.204 0.198 -0.409 0.160 -0.214 -0.127; -0.142 0.463 0.287 -0.060 -0.045 -0.008 -0.196; 0.579 0.138 0.309 -0.429 -0.193 -0.264 -0.009; 0.034 0.419 0.506 0.238 0.191 0.246 0.205; -0.731 0.203 0.563 0.613 1.036 0.362 -0.565];
```

```
for k= 1:r       %Elements present in the rows of x matrix
   net2h2r(k,1)= W1r(1,1)*qr(k,1) + W1r(1,2)*qr(k,2) + W1r(1,3)*qr(k,3)+ W1r(1,4)*qr(k,4)+ W1r(1,5)*qr(k,5)+
W1r(1,6)*qr(k,6)+ W1r(1,7)*qr(k,7) -0.333;
   net2h21r(k,1) = net2h21r(k,1) + net2h2r(k,1);
   net2h21r(k,1) = net2h2r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net2h2r(k,1)= W1r(2,1)*qr(k,1) + W1r(2,2)*qr(k,2) + W1r(2,3)*qr(k,3)+ W1r(2,4)*qr(k,4)+ W1r(2,5)*qr(k,5)+
W1r(2,6)*qr(k,6)+ W1r(2,7)*qr(k,7)+ 0.062;
   net2h22r(k,1) = net2h22r(k,1) + net2h2r(k,1);
   net2h22r(k,1) = net2h2r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net2h2r(k,1)= W1r(3,1)*qr(k,1) + W1r(3,2)*qr(k,2) + W1r(3,3)*qr(k,3)+ W1r(3,4)*qr(k,4)+ W1r(3,5)*qr(k,5)+
W1r(3,6)*qr(k,6)+ W1r(3,7)*qr(k,7)-0.159;
   net2h23r(k,1) = net2h23r(k,1) + net2h2r(k,1);
   net2h23r(k,1) = net2h2r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net2h2r(k,1)= W1r(4,1)*qr(k,1) + W1r(4,2)*qr(k,2) + W1r(4,3)*qr(k,3)+ W1r(4,4)*qr(k,4)+ W1r(4,5)*qr(k,5)+
W1r(4,6)*qr(k,6)+ W1r(4,7)*qr(k,7)+ 0.035;
   net2h24r(k,1) = net2h24r(k,1) + net2h2r(k,1);
   net2h24r(k,1) = net2h2r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net2h2r(k,1)= W1r(5,1)*qr(k,1) + W1r(5,2)*qr(k,2) + W1r(5,3)*qr(k,3)+ W1r(5,4)*qr(k,4)+ W1r(5,5)*qr(k,5)+
W1r(5,6)*qr(k,6)+ W1r(5,7)*qr(k,7)- 0.105;
   net2h25r(k,1) = net2h25r(k,1) + net2h2r(k,1);
   net2h25r(k,1) = net2h2r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net2h2r(k,1)= W1r(6,1)*qr(k,1) + W1r(6,2)*qr(k,2) + W1r(6,3)*qr(k,3)+ W1r(6,4)*qr(k,4)+ W1r(6,5)*qr(k,5)+
W1r(6,6)*qr(k,6)+ W1r(6,7)*qr(k,7)- 0.096;
   net2h26r(k,1) = net2h26r(k,1) + net2h2r(k,1);
   net2h26r(k,1) = net2h2r(k,1);
end

for k= 1:r       %Elements present in the rows of x matrix
   net2h2r(k,1)= W1r(7,1)*qr(k,1) + W1r(7,2)*qr(k,2) + W1r(7,3)*qr(k,3)+ W1r(7,4)*qr(k,4)+ W1r(7,5)*qr(k,5)+
W1r(7,6)*qr(k,6)+ W1r(7,7)*qr(k,7)- 0.235;
   net2h27r(k,1) = net2h27r(k,1) + net2h2r(k,1);
   net2h27r(k,1) = net2h2r(k,1);
end

for i = 1:r
   out1h2r(i,1)= tanh(net2h21r(i,1)); out2h2r(i,1)= tanh(net2h22r(i,1)); out3h2r(i,1)= tanh(net2h23r(i,1)); out4h2r(i,1)=
tanh(net2h24r(i,1)); out5h2r(i,1)= tanh(net2h25r(i,1)); out6h2r(i,1)= tanh(net2h26r(i,1)); out7h2r(i,1)=
tanh(net2h27r(i,1));
```

```
end
gr= [out1h2r out2h2r out3h2r out4h2r out5h2r out6h2r out7h2r];

% Evaluation of Fitness Function %
 f1 = equation3(h);  %it is a user defined function where the objective function value is evaluated for the entire population of anti bodies, program V called here
 c3= [f1 h];

% replacing the newly generated r random solutions in the initial population
for i =1:r
   for j = 1:n+1
      z(M+i,j) = c3(i,j);
   end
end
z;
Z = sortrows(z, -1);%copying the newly generated population to the initial populations and perform the iterations until generations
Y(gen,1) = Z(1,1); % copying the newly generated population to the initial populations and perform the iterations %until the termination criterion is satisfied

S(gen,1) = Z(1,2);
K(gen,1)= Z(1,3);
U(gen,1)= Z(1,4);
V(gen,1)= Z(1,5);
X(gen,1) = Z(1,6);
XY = [Y S K U V X];
XY1= sortrows(XY, -1);

 end
Z; % gives the best anti bodies solutions
Y; % gives the fitness scores of top best solutions for all generations , and it is used to plot a graph between fitness %scores and numberof generations, which shows how the fitness score is varying for each generation
S;
K;
U;
V;
X;
XY = [Y S K U V X];

XY1= sortrows(XY, -1)
[r1 r2] = size(Y);
gen = 1:r1;
plot(gen,Y,'-.ob')
title('Artificial Immune Systems Plot');
xlabel('Number of Generations');
ylabel('Conversion');
toc
```

II. Program

```
function f = equation1(x)
global net W2  g output pop
pop= 65;
```

```
output= zeros(65,1);
W2= [0.255 0.329 -0.164 0.086 0.659 0.392 0.905];

for k= 1:pop       %Elements present in the rows of x matrix
   net(k,1)= W2(1,1)*g(k,1) + W2(1,2)*g(k,2)+ W2(1,3)*g(k,3)+ W2(1,4)*g(k,4)+ W2(1,5)*g(k,5)+ W2(1,6)*g(k,6)+
W2(1,7)*g(k,7);
   output(k,1) = output(k,1) + net(k,1);
   output(k,1) = net(k,1);
end
 for i = 1:pop
   f(i,1)= output(i,1);
end
```

III. Program

```
function Tnew= matrices1(i)

global pop n B T xmin xmax beta p M Nc Jnew qs net1h1s net1h11s net1h12s net1h13s net1h14s net1h15s net1h16s
out1h1s out2h1s out3h1s out4h1s out5h1s out6h1s net2h2s net2h21s net2h22s net2h23s net2h24s net2h25s net2h26s gs
out1h2s out2h2s out3h2s out4h2s out5h2s out6h2s net1h17s out7h2s out7h1s net2h27s
 pop= 65;
 n = 5 ;   %no of variables involved in objective function
 M = 57;   %No of best antibody solutions
 beta = 1 ;  %  cloning parameter
p =4;              %Hyper mutation operator
Jnew = [];    %a matrix of hyper mutated solutions. Jnew[] this is put because when cloning and mutation is
%performedon each population of solutions there is a chance that over riding of solutions takes place to avoid that
%we used empty matrix of Jnew.

 %%% Cloning starts %%
   Nc = ceil((beta*pop)/i); % This expression generates number of clones, and based on the value of number of clones
%the parent solutions are compared that many times
  for m= 1:Nc
       for j= 1:n
          T(m,j)= B(i,j); % here based on the value of clones the copy will be made i.e., the solutions from B will get
copied into the matrix T
       end
   end         % The solutions which are copied into T matrix are mutated

   %%%% Hypermutation Starts%%%%

  for m= 1:Nc
     Jnew(m,1)= T(m,1) + (rand()*(i^2)*(xmax-xmin))/10^p; % This will give hyper-mutated solution for the first
variable.
     Jnew(m,2)= T(m,2) + (rand()*(i^2)*(xmax-xmin))/10^p
 Jnew(m,3)= T(m,3) + (rand()*(i^2)*(xmax-xmin))/10^p;
     Jnew(m,4)= T(m,4) + (rand()*(i^2)*(xmax-xmin))/10^p;
     Jnew(m,5)= T(m,5) + (rand()*(i^2)*(xmax-xmin))/10^p;
end

Ws= [-0.181 -0.474 -0.355 -0.433 0.108; 0.117 0.138 -0.334 0.036 -0.459; 1.108 0.435 0.178 1.022 -0.357; 0.241 -0.163 -0.385
0.702 -0.080; -0.353 0.865 0.345 0.823 -0.282; 0.202 0.522 0.047 -0.025 -0.268; -0.166 -0.549 0.483 -0.234 -0.230];
```

```
net1h11s= zeros(65,1); net1h12s= zeros(65,1); net1h13s= zeros(65,1); net1h14s= zeros(65,1); net1h15s= zeros(65,1);
net1h16s= zeros(65,1); net1h17s= zeros(65,1);

for k= 1:Nc       %Elements present in the rows of x matrix
net1h1s(k,1)= Ws(1,1)*Jnew(k,1) + Ws(1,2)*Jnew(k,2) + Ws(1,3)*Jnew(k,3) + Ws(1,4)*Jnew(k,4) + Ws(1,5)*Jnew(k,5)+
0.125;
   net1h11s(k,1) = net1h11s(k,1) + net1h1s(k,1);
   net1h11s(k,1) = net1h1s(k,1);
end

for k= 1:Nc       %Elements present in the rows of x matrix
   net1h1s(k,1)= Ws(2,1)*Jnew(k,1) + Ws(2,2)*Jnew(k,2) + Ws(2,3)*Jnew(k,3) + Ws(2,4)*Jnew(k,4) + Ws(2,5)*Jnew(k,5)+
0.174;
   net1h12s(k,1) = net1h12s(k,1) + net1h1s(k,1);
   net1h12s(k,1) = net1h1s(k,1);
end

for k= 1:Nc       %Elements present in the rows of x matrix
   net1h1s(k,1)= Ws(3,1)*Jnew(k,1) + Ws(3,2)*Jnew(k,2) + Ws(3,3)*Jnew(k,3) + Ws(3,4)*Jnew(k,4) + Ws(3,5)*Jnew(k,5)+
0.063;
   net1h13s(k,1) = net1h13s(k,1) + net1h1s(k,1);
   net1h13s(k,1) = net1h1s(k,1);
end

for k= 1:Nc       %Elements present in the rows of x matrix
   net1h1s(k,1)= Ws(4,1)*Jnew(k,1) + Ws(4,2)*Jnew(k,2) + Ws(4,3)*Jnew(k,3) + Ws(4,4)*Jnew(k,4) + Ws(4,5)*Jnew(k,5)+
0.332;
   net1h14s(k,1) = net1h14s(k,1) + net1h1s(k,1);
   net1h14s(k,1) = net1h1s(k,1);
end

for k= 1:Nc       %Elements present in the rows of x matrix
   net1h1s(k,1)= Ws(5,1)*Jnew(k,1) + Ws(5,2)*Jnew(k,2) + Ws(5,3)*Jnew(k,3) + Ws(5,4)*Jnew(k,4) + Ws(5,5)*Jnew(k,5)-
0.328;
   net1h15s(k,1) = net1h15s(k,1) + net1h1s(k,1);
   net1h15s(k,1) = net1h1s(k,1);
end

for k= 1:Nc      %Elements present in the rows of x matrix
   net1h1s(k,1)= Ws(6,1)*Jnew(k,1) + Ws(6,2)*Jnew(k,2) + Ws(6,3)*Jnew(k,3) + Ws(6,4)*Jnew(k,4) + Ws(6,5)*Jnew(k,5)+
0.124;
   net1h16s(k,1) = net1h16s(k,1) + net1h1s(k,1);
   net1h16s(k,1) = net1h1s(k,1);
end

for k= 1:Nc       %Elements present in the rows of x matrix
   net1h1s(k,1)= Ws(7,1)*Jnew(k,1) + Ws(7,2)*Jnew(k,2) + Ws(7,3)*Jnew(k,3) + Ws(7,4)*Jnew(k,4) + Ws(7,5)*Jnew(k,5) -
0.100;
   net1h17s(k,1) = net1h17s(k,1) + net1h1s(k,1);
   net1h17s(k,1) = net1h1s(k,1);
end

for i = 1:Nc
```

```
   out1h1s(i,1)= tanh(net1h11s(i,1)); out2h1s(i,1)= tanh(net1h12s(i,1)); out3h1s(i,1)= tanh(net1h13s(i,1)); out4h1s(i,1)= tanh(net1h14s(i,1)); out5h1s(i,1)= tanh(net1h15s(i,1)); (i,1)= tanh(net1h16s(i,1)); out7h1s(i,1)= tanh(net1h16s(i,1));
end
qs= [out1h1s out2h1s out3h1s out4h1s out5h1s out6h1s out7h1s];

net2h21s= zeros(65,1); net2h22s= zeros(65,1); net2h23s= zeros(65,1); net2h24s= zeros(65,1); net2h25s= zeros(65,1); net2h26s= zeros(65,1); net2h27s= zeros(65,1);
W1s= [0.143 -0.199 0.184 -0.163 -0.158 -0.223 -0.125; -0.366 -0.085 0.608 -0.474 -0.437 0.407 -0.026; -0.163 -0.204 0.198 -0.409 0.160 -0.214 -0.127; -0.142 0.463 0.287 -0.060 -0.045 -0.008 -0.196; 0.579 0.138 0.309 -0.429 -0.193 -0.264 -0.009; 0.034 0.419 0.506 0.238 0.191 0.246 0.205; -0.731 0.203 0.563 0.613 1.036 0.362 -0.565];

for k= 1:Nc       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1s(1,1)*qs(k,1) + W1s(1,2)*qs(k,2) + W1s(1,3)*qs(k,3)+ W1s(1,4)*qs(k,4)+ W1s(1,5)*qs(k,5)+ W1s(1,6)*qs(k,6)+ W1s(1,7)*qs(k,7)- 0.333;
   net2h21s(k,1) = net2h21s(k,1) + net2h2s(k,1);
   net2h21s(k,1) = net2h2s(k,1);
end
for k= 1:Nc       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1s(2,1)*qs(k,1) + W1s(2,2)*qs(k,2) + W1s(2,3)*qs(k,3)+ W1s(2,4)*qs(k,4)+ W1s(2,5)*qs(k,5)+ W1s(2,6)*qs(k,6)+ W1s(2,7)*qs(k,7)+ 0.062;
   net2h22s(k,1) = net2h22s(k,1) + net2h2s(k,1);
   net2h22s(k,1) = net2h2s(k,1);
end
for k= 1:Nc       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1s(3,1)*qs(k,1) + W1s(3,2)*qs(k,2) + W1s(3,3)*qs(k,3)+ W1s(3,4)*qs(k,4)+ W1s(3,5)*qs(k,5)+ W1s(3,6)*qs(k,6)+ W1s(3,7)*qs(k,7)- 0.159;
   net2h23s(k,1) = net2h23s(k,1) + net2h2s(k,1);
   net2h23s(k,1) = net2h2s(k,1);
end
for k= 1:Nc       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1s(4,1)*qs(k,1) + W1s(4,2)*qs(k,2) + W1s(4,3)*qs(k,3)+ W1s(4,4)*qs(k,4)+ W1s(4,5)*qs(k,5)+ W1s(4,6)*qs(k,6)+ W1s(4,7)*qs(k,7)+ 0.035;
   net2h24s(k,1) = net2h24s(k,1) + net2h2s(k,1);
   net2h24s(k,1) = net2h2s(k,1);
end
for k= 1:Nc       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1s(5,1)*qs(k,1) + W1s(5,2)*qs(k,2) + W1s(5,3)*qs(k,3)+ W1s(5,4)*qs(k,4)+ W1s(5,5)*qs(k,5)+ W1s(5,6)*qs(k,6)+ W1s(5,7)*qs(k,7)- 0.105;
   net2h25s(k,1) = net2h25s(k,1) + net2h2s(k,1);
   net2h25s(k,1) = net2h2s(k,1);
end
for k= 1:Nc       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1s(6,1)*qs(k,1) + W1s(6,2)*qs(k,2) + W1s(6,3)*qs(k,3)+ W1s(6,4)*qs(k,4)+ W1s(6,5)*qs(k,5)+ W1s(6,6)*qs(k,6)+ W1s(6,7)*qs(k,7)- 0.096;
   net2h26s(k,1) = net2h26s(k,1) + net2h2s(k,1);
   net2h26s(k,1) = net2h2s(k,1);
end

for k= 1:Nc       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1s(7,1)*qs(k,1) + W1s(7,2)*qs(k,2) + W1s(7,3)*qs(k,3)+ W1s(7,4)*qs(k,4)+ W1s(7,5)*qs(k,5)+ W1s(7,6)*qs(k,6)+ W1s(7,7)*qs(k,7)- 0.235;
   net2h27s(k,1) = net2h27s(k,1) + net2h2s(k,1);
   net2h27s(k,1) = net2h2s(k,1);
```

```
end
for i = 1:Nc
   out1h2s(i,1)= tanh(net2h21s(i,1)); out2h2s(i,1)= tanh(net2h22s(i,1)); out3h2s(i,1)= tanh(net2h23s(i,1)); out4h2s(i,1)=
tanh(net2h24s(i,1)); out5h2s(i,1)= tanh(net2h25s(i,1)); out6h2s(i,1)= tanh(net2h26s(i,1));
out7h2s(i,1)= tanh(net2h27s(i,1));
end

gs= [out1h2s out2h2s out3h2s out4h2s out5h2s out6h2s out7h2s];
% Evaluating the fitness function for hypermutated solution
  Znew = equation2(Jnew); %program IV is called here
  Tnew = [Znew Jnew];
```

IV. Program

```
function Znew = equation2(x)
global nets W2s gs outputs Nc
outputs= zeros(65,1);
W2s= [0.255 0.329 -0.164 0.086 0.659 0.392 0.905];

for k= 1:Nc      %Elements present in the rows of x matrix
   nets(k,1)=    W2s(1,1)*gs(k,1)    +    W2s(1,2)*gs(k,2)+    W2s(1,3)*gs(k,3)+    W2s(1,4)*gs(k,4)+    W2s(1,5)*gs(k,5)+
W2s(1,6)*gs(k,6)+ W2s(1,7)*gs(k,7);
   outputs(k,1) = outputs(k,1) + nets(k,1);
   outputs(k,1) = nets(k,1);
end
for i = 1:Nc
   Znew(i,1)= outputs(i,1);
end
```

V. Program

```
function f1 = equation3(h)
global netr W2r  gr outputr r
r= 16;
outputr= zeros(73,1);
W2r= [0.255 0.329 -0.164 0.086 0.659 0.392 0.905];
for k= 1:r       %Elements present in the rows of x matrix
   netr(k,1)=    W2r(1,1)*gr(k,1)    +    W2r(1,2)*gr(k,2)+    W2r(1,3)*gr(k,3)+    W2r(1,4)*gr(k,4)+    W2r(1,5)*gr(k,5)+
W2r(1,6)*gr(k,6)+ W2r(1,7)*gr(k,7);
   outputr(k,1) = outputr(k,1) + netr(k,1);
   outputr(k,1) = netr(k,1);
end
outputr;
 for i = 1:r
f1(i,1)= outputr(i,1);
end
f1;
```

MATLAB code for constrained optimization

```
function constarinedais

% the problem which is considered for constrained optimization is
% Min f(x) = (x (1) – 10) ^3 + (x (2) – 20) ^3
```

```
%   Subject to  g1(x) = - (x (1) – 5) ^2 - (x (2) – 5) ^2 + 100 <=0
%   g2(x) = (x (1) – 6) ^2 + (x (2) – 5) ^2 – 82.81<=0,
  %  Where   13<= x (1) <=100    and      0<= x (2) <=100

global pop n M r Nmax xmin1 xmin2 xmax beta p L D G xnew
 pop= 400; M= 300; r= 100; xmax= 100; xmin1= 3; xmin2= 0; beta= 1; p= 4; n= 2; Nmax= 20;

%%% Initializing the antibodies %%%
rand('seed',10);        % This is used so that it generates same random solution everytime.

x1 = xmin1 + (xmax-xmin1)*rand(pop,1); %% Initializing the vector of initial population between the respective limits
x2 = xmin2 + (xmax-xmin2)*rand(pop,1);
x = [x1 x2];

%%% Using Penalty Constrained Optimisation method %%%
%%% Finding the most fit penalty value and then evaluating the fitness
%%% function.

% fixing the values of r1 and r2 by trial and error method
for i= 1:1:pop
   if -(x(i,1)-5)^2 - (x(i,2)-5)^2 + 100 <=0
      r1=0;
   elseif -(x(i,1)-5)^2 - (x(i,2)-5)^2 + 100 >0 && -(x(i,1)-5)^2 - (x(i,2)-5)^2 + 100 <=10
      r1 =0.5;
   elseif -(x(i,1)-5)^2 - (x(i,2)-5)^2 + 100 >10 && -(x(i,1)-5)^2 - (x(i,2)-5)^2 + 100 <=20
      r1 =1;
   else r1=1.5;
   end

   if (x(i,1)-6)^2 + (x(i,2)-5)^2 - 82.81 <=0
      r2=0;
   elseif (x(i,1)-6)^2 + (x(i,2)-5)^2 - 82.81 >0 && (x(i,1)-6)^2 + (x(i,2)-5)^2 - 82.81 <=10
      r2=0.5;
   elseif (x(i,1)-6)^2 + (x(i,2)-5)^2 - 82.81 > 10 && (x(i,1)-6)^2 + (x(i,2)-5)^2 - 82.81 <=20
      r2=1;
   else r2= 1.5;
   end
f(i,1)= (x(i,1)-10)^3 + (x(i,2)-20)^3 + r1*(max(0,-(x(i,1)-5)^2 - (x(i,2)-5)^2 + 100))^2 + r2*(max(0,(x(i,1)-6)^2 + (x(i,2)-5)^2 -
82.81))^2;
end
f;
z=[f x]; %% Set of objective function values for all solutions and the solutions generated

%%% Checking the feasibility of the solution and sorting as feasible and infeasible sets
for i=1:1:pop
   if (-(z(i,2)-5)^2 - (z(i,3)-5)^2 + 100 <=0) && ((z(i,2)-6)^2 + (z(i,3)-5)^2 - 82.81 <=0)
      A(i,1)= z(i,1); %% The feasible solution will get copied in the matrix A
      A(i,2)= z(i,2);
      A(i,3)= z(i,3);
   else
      B(i,1)= z(i,1); % The infeasible solution will be copied here in the matrix B
      B(i,2)= z(i,2);
      B(i,3)= z(i,3);
```

```
      A(i,1)= 0;
      A(i,2)= 0;
      A(i,3)= 0;
   end
end
newmatA= A(A(:,1)~=0,:); newmatB= B(B(:,1)~=0,:);
newmatA= sort(newmatA); %% Sorting feasible solutions in ascending order
newmatB= sort(newmatB); %% Sorting infeasible solution in ascending order

%%% Now vertically concatenating the overall solution, this can be done in two ways: 1. by adding the infeasible
%solution at the end of the feasible solution and 2. by taking the feasible solutions and then infeasible solutions and
%selecting the M best solutions out of them.

C1= vertcat(newmatA,newmatB);% Adding the feasible solution with the infeasible solution and now it will become
the initial population

%%% generation starts
for gen= 1:1:Nmax
for i= 1:1:M
   for j= 1:2 %% Specific for this problem only will change as the equation changes
      L(i,j)= C1(i,j+1);
   end
end
L; % Gives the M best solutions

% Cloning and Hypermutating and then evaluating the fitness score
for i= 1:M
   s = ['T',int2str(i),'=matrices1(i)'];
   eval(s);
end
%%% To vertically concatenate all the solutions coming from the matrices1.m
%%% file
C2= vertcat(T1,T2,T3,… T300);%concatenating all the solutions vertically
C2;
[D G]= size(C2);

%Finding the Feasibility Solution after hypermutation %
A=[]; B=[];
for i=1:1:D
if (-(C2(i,1)-5)^2 - (C2(i,2)-5)^2 + 100 <=0) && ((C2(i,1)-6)^2 + (C2(i,2)-5)^2 - 82.81 <=0)
      A(i,1)= C2(i,1); %% The feasible solution will get copied in the matrix A
      A(i,2)= C2(i,2);
    else
      B(i,1)= C2(i,1); % The infeasible solution will be copied here in the matrix B
      B(i,2)= C2(i,2);
       A(i,1)= 0;
      A(i,2)= 0;
   end
end
newmatA= A(A(:,1)~=0,:); newmatB= B(B(:,1)~=0,:);
newmatA= sort(newmatA); %% Sorting feasible solutions in ascending order
newmatB= sort(newmatB); %% Sorting infeasible solution in ascending order
```

```
C3 =vertcat(newmatA, newmatB);

for i= 1:1:M
   for j= 1:2
      C4(i,j)= C3(i,j);
   end
end
C4; %% are the top M best solutions

% Now comapring the fitness scores between the initial population C3 and the new created C4 and the one having
%the lower fitness score is replaced by the other.
for i=1:1:M
   for j=1
     if C4(i,j)<C3(i,j)
        C3(i,1:end)= C4(i,1:end);
     end
   end
end
C4; %% It is the new replaced initial population.

%%% Now to genarate r random antibodies so as to replace at the end of antibodies
x1= xmax + (xmax-xmin1)*rand(r,1);
x2= xmax + (xmax-xmin2)*rand(r,2);
xnew= [x1 x2];
% Now to evaluate the fitness score of the r randomly generated solutions
  fnew= equation1(xnew);
z2= [fnew xnew];
 for i = 1:1:r
      for j = 1:1:4
         C4(M+i,j) = z2(i,j);
      end
   end
   C4 %new generated population consists of best solutions and the poor with their fitness scores

%now the best N solutions are separated as feasible and infeasible solutions
A1 = []; B1 = [];
 for i = 1:1:pop
if ((-(C4(i,2)-5)^2-(C4(i,3)-5)^2+100 <=0) && ((C4(i,2)-6)^2 + (C4(i,3)-5)^2-82.81 <=0))
           A1(i,1) = C4(i,1);
           A1(i,2) = C4(i,2);
           A1(i,3) = C4(i,3);
else
           B1(i,1) = C4(i,1);
          B1(i,2) = C4(i,2);
          B1(i,3) = C4(i,3);
           A1(i,1) = 0;
          A1(i,2) = 0;
          A1(i,3) = 0;
end
 end
A1 %set of only feasible solutions
B1 % a set of infeasible solutions which contains the solutions and fitness scores
newmatA1 = A1(A1(:,1)~=0,:)  %set of only  feasible solutions
```

```
newmatB1 = B1(B1(:,1)~=0,:)  %set of  only infeasible solutions
newmatA1 = sort(A1)     % sorting out set of feasible solutions
newmatB1 = sort(newmatB1)     % sorting out set of infeasible solutions
C5 = vertcat(newmatA1,newmatB1) %is the new population of feasible and in feasible solutions
V(gen ,1) = C5(1,1);
end
C5
V    % gives the best fitness scores of all generations
plot(1:1:gen,V,'-.or')
```

MATLAB code for genetic algorithms

```
function G_Alg
global npar varhi varlo maxit maxconv popsize mutrate selection Nt M nmut par maxc meanc iga net1h1 net1h11 net1h12 net1h13 net1h14 net1h15 net1h16 out1h1 out2h1 out3h1 out4h1 out5h1 out6h1 net2h2 net2h21 net2h22 net2h23 net2h24 net2h25 net2h26 g out1h2 out2h2 out3h2 out4h2 out5h2 out6h2 conv odds pick1 pick2 ma pa conv1 xy mrow mcol par1 net1h1s net1h11s net1h12s net1h13s net1h14s net1h15s net1h16s out1h1s out2h1s out3h1s out4h1s out5h1s out6h1s out7h1s net2h2s net2h21s net2h22s net2h23s net2h24s net2h25s net2h26s gs out1h2s out2h2s out3h2s out4h2s out5h2s out6h2s out7h2s...

npar=5; % number of optimization variables
varhi=0.9; varlo=0.1; % variable limits
% Stopping criteria
maxit= 5000; % max number of iterations
maxconv= 1.000; % maximum conv

% III GA parameters
popsize=65; % set population size
mutrate=.2; % set mutation rate
selection=0.5; % fraction of population kept
Nt=npar; % continuous parameter GA Nt=#variables
keep=floor(selection*popsize); % #population members that survive
nmut=ceil((popsize-1)*Nt*mutrate); % total number of mutations
M=ceil((popsize-keep)/2); % number of matings

% Create the initial population
iga=0; % generation counter initialized
rand('seed',20); % It is used so that it generates the same random solution when iterated several times.
par=(varhi-varlo)*rand(popsize,npar)+ varlo; % random
 W= [-0.181 -0.474 -0.355 -0.433 0.108; 0.117 0.138 -0.334 0.036 -0.459; 1.108 0.435 0.178 1.022 -0.357; 0.241 -0.163 -0.385 0.702 -0.080; -0.353 0.865 0.345 0.823 -0.282;-0.202 0.522 0.047 -0.025 -0.268;    -0.166 -0.549 0.483 -0.234 -0.230];
net1h11= zeros(65,1); net1h12= zeros(65,1); net1h13= zeros(65,1); net1h14= zeros(65,1); net1h15= zeros(65,1); net1h16= zeros(65,1); net1h17= zeros(65,1);

%%% Initializing the antibodies %%%
 %%% Evaluation of antibodies %%%
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1(k,1)= W(1,1)*par(k,1) + W(1,2)*par(k,2) + W(1,3)*par(k,3) + W(1,4)*par(k,4) + W(1,5)*par(k,5)+ 0.125;
   net1h11(k,1) = net1h11(k,1) + net1h1(k,1);
   net1h11(k,1) = net1h1(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1(k,1)= W(2,1)*par(k,1) + W(2,2)*par(k,2) + W(2,3)*par(k,3) + W(2,4)*par(k,4) + W(2,5)*par(k,5)+ 0.174;
```

```
   net1h12(k,1) = net1h12(k,1) + net1h1(k,1);
   net1h12(k,1) = net1h1(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1(k,1)= W(3,1)*par(k,1) + W(3,2)*par(k,2) + W(3,3)*par(k,3) + W(3,4)*par(k,4) + W(3,5)*par(k,5)+ 0.063;
   net1h13(k,1) = net1h13(k,1) + net1h1(k,1);
   net1h13(k,1) = net1h1(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1(k,1)= W(4,1)*par(k,1) + W(4,2)*par(k,2) + W(4,3)*par(k,3) + W(4,4)*par(k,4) + W(4,5)*par(k,5)+ 0.332;
   net1h14(k,1) = net1h14(k,1) + net1h1(k,1);
   net1h14(k,1) = net1h1(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1(k,1)= W(5,1)*par(k,1) + W(5,2)*par(k,2) + W(5,3)*par(k,3) + W(5,4)*par(k,4) + W(5,5)*par(k,5)- 0.328;
   net1h15(k,1) = net1h15(k,1) + net1h1(k,1);
   net1h15(k,1) = net1h1(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1(k,1)= W(6,1)*par(k,1) + W(6,2)*par(k,2) + W(6,3)*par(k,3) + W(6,4)*par(k,4) + W(6,5)*par(k,5)+ 0.124;
   net1h16(k,1) = net1h16(k,1) + net1h1(k,1);
   net1h16(k,1) = net1h1(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1(k,1)= W(7,1)*par(k,1) + W(7,2)*par(k,2) + W(7,3)*par(k,3) + W(7,4)*par(k,4) + W(7,5)*par(k,5) - 0.100;
   net1h17(k,1) = net1h17(k,1) + net1h1(k,1);
   net1h17(k,1) = net1h1(k,1);
end
for i = 1:popsize
   out1h1(i,1)= tanh(net1h11(i,1)); out2h1(i,1)= tanh(net1h12(i,1)); out3h1(i,1)= tanh(net1h13(i,1)); out4h1(i,1)= tanh(net1h14(i,1)); out5h1(i,1)= tanh(net1h15(i,1)); out6h1(i,1)= tanh(net1h16(i,1)); out7h1(i,1)= tanh(net1h17(i,1));
end
q= [out1h1 out2h1 out3h1 out4h1 out5h1 out6h1 out7h1];
 net2h21= zeros(65,1);net2h22= zeros(65,1);net2h23= zeros(65,1);net2h24= zeros(65,1);net2h25= zeros(65,1);net2h26= zeros(65,1);net2h27= zeros(65,1);

W1= [0.143 -0.199 0.184 -0.163 -0.158 -0.223 -0.125;-0.366 -0.085 0.608 -0.474 -0.437 0.407 -0.026; -0.163 -0.204 0.198 -0.409 0.160 -0.214 -0.127; -0.142 0.463 0.287 -0.060 -0.045 -0.008 -0.196; 0.579 0.138 0.309 -0.429 -0.193 -0.264 -0.009; 0.034 0.419 0.506 0.238 0.191 0.246 0.205; 0.731 0.203 0.563 0.613 1.036 0.362 -0.565];

for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(1,1)*q(k,1) + W1(1,2)*q(k,2) + W1(1,3)*q(k,3)+ W1(1,4)*q(k,4)+ W1(1,5)*q(k,5)+ W1(1,6)*q(k,6)+ W1(1,7)*q(k,7)- 0.333;
   net2h21(k,1) = net2h21(k,1) + net2h2(k,1);
   net2h21(k,1) = net2h2(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(2,1)*q(k,1) + W1(2,2)*q(k,2) + W1(2,3)*q(k,3)+ W1(2,4)*q(k,4)+ W1(2,5)*q(k,5)+ W1(2,6)*q(k,6)+ W1(2,7)*q(k,7)+ 0.062;
   net2h22(k,1) = net2h22(k,1) + net2h2(k,1);
   net2h22(k,1) = net2h2(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
```

```
   net2h2(k,1)= W1(3,1)*q(k,1) + W1(3,2)*q(k,2) + W1(3,3)*q(k,3)+ W1(3,4)*q(k,4)+ W1(3,5)*q(k,5)+ W1(3,6)*q(k,6)+
W1(3,7)*q(k,7)- 0.159;
   net2h23(k,1) = net2h23(k,1) + net2h2(k,1);
   net2h23(k,1) = net2h2(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(4,1)*q(k,1) + W1(4,2)*q(k,2) + W1(4,3)*q(k,3)+ W1(4,4)*q(k,4)+ W1(4,5)*q(k,5)+ W1(4,6)*q(k,6)+
W1(4,7)*q(k,7)+ 0.035;
   net2h24(k,1) = net2h24(k,1) + net2h2(k,1);
   net2h24(k,1) = net2h2(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(5,1)*q(k,1) + W1(5,2)*q(k,2) + W1(5,3)*q(k,3)+ W1(5,4)*q(k,4)+ W1(5,5)*q(k,5)+ W1(5,6)*q(k,6)+
W1(5,7)*q(k,7)- 0.105;
   net2h25(k,1) = net2h25(k,1) + net2h2(k,1);
   net2h25(k,1) = net2h2(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(6,1)*q(k,1) + W1(6,2)*q(k,2) + W1(6,3)*q(k,3)+ W1(6,4)*q(k,4)+ W1(6,5)*q(k,5)+ W1(6,6)*q(k,6)+
W1(6,7)*q(k,7)- 0.096;
   net2h26(k,1) = net2h26(k,1) + net2h2(k,1);
   net2h26(k,1) = net2h2(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2(k,1)= W1(7,1)*q(k,1) + W1(7,2)*q(k,2) + W1(7,3)*q(k,3)+ W1(7,4)*q(k,4)+ W1(7,5)*q(k,5)+ W1(7,6)*q(k,6)+
W1(7,7)*q(k,7)- 0.235;
   net2h27(k,1) = net2h27(k,1) + net2h2(k,1);
   net2h27(k,1) = net2h2(k,1);
end
for i = 1:popsize
   out1h2(i,1)= tanh(net2h21(i,1));out2h2(i,1)= tanh(net2h22(i,1)); out3h2(i,1)= tanh(net2h23(i,1)); (i,1)=
tanh(net2h24(i,1)); out5h2(i,1)= tanh(net2h25(i,1));out6h2(i,1)= tanh(net2h26(i,1)); out7h2(i,1)= tanh(net2h27(i,1));
end
g= [out1h2 out2h2 out3h2 out4h2 out5h2 out6h2 out7h2];

% Evaluation of Fitness Function %
 f = equation(par);  %it is a user defined function where the objective function value is evaluated for the entire
population of anti bodies, program VI called here
 f;
conv =sortrows(f, -1);
maxc(1)= max(f); % minc contains max of
meanc(1)=mean(f); % meanc contains mean of population
for i= 1:keep
   conv1(i,1)= conv(i,1);
end
conv1;
% % Iterate through generations
while iga<maxit
iga=iga+1; % increments generation counter
tic;

% Pair and mate
M=ceil((popsize-keep)/2); % number of matings
```

```
prob=flipud((1:keep)/sum((1:keep))); % weights chromosomes
odds=[0 cumsum(prob(1:keep))]; % probability distribution function
pick1=rand(1,M); % mate #1
pick2=rand(1,M); % mate #2 ma and pa contain the indicies of the chromosomes that will mate
ic=1;
while ic<=M
for id=2:keep+1
if pick1(ic)<=odds(id) && pick1(ic)>odds(id-1)
ma(ic)=id-1;
end
if pick2(ic)<=odds(id) && pick2(ic)>odds(id-1)
pa(ic)=id-1;
end
 end
 ic=ic+1;
end
 for i= 1: M
    mating(i,1)= ma(1,i);
    pairing(i,1) = pa(1,i);
 end
 mating;
 pairing;
%_____________________________________________________________
% Performs mating using single point crossover
ix=1:2:37; % index of mate #1
xp=ceil(rand(1,M)*Nt); % crossover point
r=rand(1,M); % mixing parameter

for ic=1:M
xy(ic)=par(ma(ic),xp(ic))-par(pa(ic),xp(ic)); % ma and pa mate
par(keep+ix(ic),:)=par(ma(ic),:); % 1st offspring
par(keep+ix(ic)+1,:)=par(pa(ic),:);  % 2nd offspring

par(keep+ix(ic),xp(ic))=par(ma(ic),xp(ic))-r(ic).*xy(ic); % 1st
par(keep+ix(ic)+1,xp(ic))=par(pa(ic),xp(ic))+r(ic).*xy(ic); % 2nd
end

% Mutate the population
a= ceil(rand(1,nmut)*(popsize-1))+1;
mrow=sort(a, 'descend');
mcol=ceil(rand(1,nmut)*Nt);
for ii=1:nmut
par(mrow(ii),mcol(ii))=(varhi-varlo)*rand+varlo; %mutation
end % ii
par1= par;
% The new offspring and mutated chromosomes are
% evaluated
W= [-0.181 -0.474 -0.355 -0.433 0.108; 0.117 0.138 -0.334 0.036 -0.459;.108 0.435 0.178 1.022 -0.357;     0.241 -0.163 -0.385
0.702 -0.080; -0.353 0.865 0.345 0.823 -0.282;- 0.202 0.522 0.047 -0.025 -0.268; -0.166 -0.549 0.483 -0.234 -0.230];
net1h11s=zeros(65,1);net1h12s=zeros(65,1);net1h13s=  zeros(65,1);net1h14s=  zeros(65,1);net1h15s=zeros(65,1);net1h16=
zeros(65,1);net1h17s= zeros(65,1);

%%% Initializing the antibodies %%%
```

```
 %%% Evaluation of antibodies %%%
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1s(k,1)= W(1,1)*par1(k,1) + W(1,2)*par1(k,2) + W(1,3)*par1(k,3) + W(1,4)*par1(k,4) + W(1,5)*par1(k,5)+ 0.125;
   net1h11s(k,1) = net1h11s(k,1) + net1h1s(k,1);
   net1h11s(k,1) = net1h1s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1s(k,1)= W(2,1)*par1(k,1) + W(2,2)*par1(k,2) + W(2,3)*par1(k,3) + W(2,4)*par1(k,4) + W(2,5)*par1(k,5)+ 0.174;
   net1h12s(k,1) = net1h12s(k,1) + net1h1s(k,1);
   net1h12s(k,1) = net1h1s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1s(k,1)= W(3,1)*par1(k,1) + W(3,2)*par1(k,2) + W(3,3)*par1(k,3) + W(3,4)*par1(k,4) + W(3,5)*par1(k,5)+ 0.063;
   net1h13s(k,1) = net1h13s(k,1) + net1h1s(k,1);
   net1h13s(k,1) = net1h1s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1s(k,1)= W(4,1)*par1(k,1) + W(4,2)*par1(k,2) + W(4,3)*par1(k,3) + W(4,4)*par1(k,4) + W(4,5)*par1(k,5)+ 0.332;
   net1h14s(k,1) = net1h14s(k,1) + net1h1s(k,1);
   net1h14s(k,1) = net1h1s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1s(k,1)= W(5,1)*par1(k,1) + W(5,2)*par1(k,2) + W(5,3)*par1(k,3) + W(5,4)*par1(k,4) + W(5,5)*par1(k,5)- 0.328;
   net1h15s(k,1) = net1h15s(k,1) + net1h1s(k,1);
   net1h15s(k,1) = net1h1s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1s(k,1)= W(6,1)*par1(k,1) + W(6,2)*par1(k,2) + W(6,3)*par1(k,3) + W(6,4)*par1(k,4) + W(6,5)*par1(k,5)+ 0.124;
   net1h16s(k,1) = net1h16s(k,1) + net1h1s(k,1);
   net1h16s(k,1) = net1h1s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net1h1s(k,1)= W(7,1)*par1(k,1) + W(7,2)*par1(k,2) + W(7,3)*par1(k,3) + W(7,4)*par1(k,4) + W(7,5)*par1(k,5) - 0.100;
   net1h17s(k,1) = net1h17s(k,1) + net1h1s(k,1);
   net1h17s(k,1) = net1h1s(k,1);
end
net1h17s;

for i = 1:popsize
   out1h1s(i,1)= tanh(net1h11s(i,1));out2h1s(i,1)= tanh(net1h12s(i,1));out3h1s(i,1)= tanh(net1h13s(i,1));out4h1s(i,1)=
tanh(net1h14s(i,1));out5h1s(i,1)=tanh(net1h15s(i,1));out6h1s(i,1)= tanh(net1h16s(i,1)); out7h1s(i,1)= tanh(net1h17s(i,1));
end
qs= [out1h1s out2h1s out3h1s out4h1s out5h1s out6h1s out7h1s];

net2h21s=zeros(65,1);net2h22s=zeros(65,1);net2h23s=zeros(65,1);net2h24s=zeros(65,1); net2h25s= zeros(65,1);net2h26s=
zeros(65,1);net2h27s= zeros(65,1);

W1= [0.143 -0.199 0.184 -0.163 -0.158 -0.223 -0.125; -0.366 -0.085 0.608 -0.474 -0.437 0.407 -0.026; -0.163 -0.204 0.198 -
0.409 0.160 -0.214 -0.127; -0.142 0.463 0.287 -0.060 -0.045 -0.008 -0.196; 0.579 0.138 0.309 -0.429 -0.193 -0.264 -0.009; 0.034
0.419 0.506 0.238 0.191 0.246 0.205; -0.731 0.203 0.563 0.613 1.036 0.362 -0.565];

for k= 1:popsize       %Elements present in the rows of x matrix
```

```
   net2h2s(k,1)= W1(1,1)*qs(k,1) + W1(1,2)*qs(k,2) + W1(1,3)*qs(k,3)+ W1(1,4)*qs(k,4)+ W1(1,5)*qs(k,5)+
W1(1,6)*qs(k,6)+ W1(1,7)*qs(k,7)- 0.333;
   net2h21s(k,1) = net2h21s(k,1) + net2h2s(k,1);
   net2h21s(k,1) = net2h2s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1(2,1)*qs(k,1) + W1(2,2)*qs(k,2) + W1(2,3)*qs(k,3)+ W1(2,4)*qs(k,4)+ W1(2,5)*qs(k,5)+
W1(2,6)*qs(k,6)+ W1(2,7)*qs(k,7)+ 0.062;
   net2h22s(k,1) = net2h22s(k,1) + net2h2s(k,1);
   net2h22s(k,1) = net2h2s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1(3,1)*qs(k,1) + W1(3,2)*qs(k,2) + W1(3,3)*qs(k,3)+ W1(3,4)*qs(k,4)+ W1(3,5)*qs(k,5)+
W1(3,6)*qs(k,6)+ W1(3,7)*qs(k,7)- 0.159;
   net2h23s(k,1) = net2h23s(k,1) + net2h2s(k,1);
   net2h23s(k,1) = net2h2s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1(4,1)*qs(k,1) + W1(4,2)*qs(k,2) + W1(4,3)*qs(k,3)+ W1(4,4)*qs(k,4)+ W1(4,5)*qs(k,5)+
W1(4,6)*qs(k,6)+ W1(4,7)*qs(k,7)+ 0.035;
   net2h24s(k,1) = net2h24s(k,1) + net2h2s(k,1);
   net2h24s(k,1) = net2h2s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1(5,1)*qs(k,1) + W1(5,2)*qs(k,2) + W1(5,3)*qs(k,3)+ W1(5,4)*qs(k,4)+ W1(5,5)*qs(k,5)+
W1(5,6)*qs(k,6)+ W1(5,7)*qs(k,7)- 0.105;
   net2h25s(k,1) = net2h25s(k,1) + net2h2s(k,1);
   net2h25s(k,1) = net2h2s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1(6,1)*qs(k,1) + W1(6,2)*qs(k,2) + W1(6,3)*qs(k,3)+ W1(6,4)*qs(k,4)+ W1(6,5)*qs(k,5)+
W1(6,6)*qs(k,6)+ W1(6,7)*qs(k,7)- 0.096;
   net2h26s(k,1) = net2h26s(k,1) + net2h2s(k,1);
   net2h26s(k,1) = net2h2s(k,1);
end
for k= 1:popsize       %Elements present in the rows of x matrix
   net2h2s(k,1)= W1(7,1)*qs(k,1) + W1(7,2)*qs(k,2) + W1(7,3)*qs(k,3)+ W1(7,4)*qs(k,4)+ W1(7,5)*qs(k,5)+
W1(7,6)*qs(k,6)+ W1(7,7)*qs(k,7)- 0.235;
   net2h27s(k,1) = net2h27s(k,1) + net2h2s(k,1);
   net2h27s(k,1) = net2h2s(k,1);
end
for i = 1:popsize
   out1h2s(i,1)= tanh(net2h21s(i,1));out2h2s(i,1)= tanh(net2h22s(i,1));out3h2s(i,1)= tanh(net2h23s(i,1));out4h2s(i,1)=
tanh(net2h24s(i,1));out5h2s(i,1)= tanh(net2h25s(i,1));out6h2s(i,1)= tanh(net2h26s(i,1));out7h2s(i,1)= tanh(net2h27s(i,1));
end
gs= [out1h2s out2h2s out3h2s out4h2s out5h2s out6h2s out7h2s];

% Evaluation of Fitness Function %
 f1 = equation4(par1);  %it is a user defined function where the objective function value is evaluated for the entire
population of anti bodies, program VII called here
par= par1;
conv =sortrows(f1, -1);
```

```
% % Stopping criteria
if or(iga>maxit,conv(1)> maxconv)
break
end
k1= iga;
k2(k1,1)= conv(1);

end %iga
k2
[r1 r2] = size(k2);
gen = 1:r1;
plot(gen,k2,'-.r*');
 [n1 n2]= size(Y);
gen1= 1:n1;
plot(gen1,Y,'-.magenta*')
title('Genetic Algorithm Plot');
xlabel('Number of Generations');
ylabel('Conversion');
toc;
```

VI. Program

```
function f = equation(par)
global net W2  g output popsize
popsize= 73;
output= zeros(73,1);
W2= [0.255 0.329 -0.164 0.086 0.659 0.392 0.905];
for k= 1:popsize       %Elements present in the rows of x matrix
   net(k,1)= W2(1,1)*g(k,1) + W2(1,2)*g(k,2)+ W2(1,3)*g(k,3)+ W2(1,4)*g(k,4)+ W2(1,5)*g(k,5)+ W2(1,6)*g(k,6)+
W2(1,7)*g(k,7);
   output(k,1) = output(k,1) + net(k,1);
   output(k,1) = net(k,1);
end
i = 1:popsize
   f(i,1)= output(i,1);
end
```

VII. Program

```
function f1 = equation4(par1)
global nets W2  gs  popsize f2
popsize= 73;
f2= zeros(73,1);
W2= [0.255 0.329 -0.164 0.086 0.659 0.392 0.905];
for k= 1:popsize       %Elements present in the rows of x matrix
   nets(k,1)=  W2(1,1)*gs(k,1)  +  W2(1,2)*gs(k,2)+  W2(1,3)*gs(k,3)+  W2(1,4)*gs(k,4)+  W2(1,5)*gs(k,5)+  W2(1,6)*gs(k,6)+
W2(1,7)*gs(k,7);
   f2(k,1) = f2(k,1) + nets(k,1);
   f2(k,1) = nets(k,1);
end
 for k= 1: popsize
    f1(k,1)= tanh(f2(k,1));
 end
```