

# **Chapter 1**

## **Introduction**

The objective of this research is to develop new learning methods based upon optimization techniques. Two different but related areas are focused on. The first is to develop new learning algorithms for neural networks. These new learning techniques could later be used in a variety of applications such as control and pattern recognition applications. The second is to develop new learning controllers which improve in their performance each time they are requested to repeat a task.

### **1.1. Introduction to Neural Network Learning**

Neural net (NN) models have been studied for many years with the hope that the superior learning and recognition capability of the human brain could be emulated by man-made machines. Similar massive networks, in the human brain, make the complex pattern and speech recognition of humans possible. In contrast to the Von Neumann computers which compute sequentially, neural nets employ huge parallel networks of many densely interconnected computational elements called neurons. Neural networks have been used in many different applications such as Adaptive and Learning Control, Pattern Recognition, Image Processing, Signature Recognition, Signal Processing and Speech Recognition, Financial Problems, etc.

A neuron is the most elementary computational unit in a neural network which sums a number of weighted inputs and passes the result through a non-linear activation function. Multi-layer neural networks (figure 1.1) consist of a large number of neurons. Before a neural network can be used for any purpose, the weights connecting inputs to neurons and the parameters of the activation functions of neurons should be adjusted so that outputs of the network will match desired values for specific sets of inputs. The methods used for adjusting these weights and parameters are usually referred to as learning algorithms.

Rumelhart et al.<sup>[1]</sup> introduced a learning theory for multi-layer neural nets in 1986 and to some extent demonstrated that a general learning algorithm for multi-layer Neural Networks is possible. In a learning algorithm called the Back-Propagation technique, they used the so called generalized delta rule to calculate an approximate gradient vector which is then used by a steepest descent search to minimize the difference between the NN output and the desired output. However, as demonstrated by simulations obtained by Rumelhart et al., low rates of convergence were seen in practically every problem. Lippmann<sup>[2]</sup> states, "One difficulty noted by the Back-Propagation algorithm is that in many cases the number of presentations of training data required for convergence has been large (more than 100 passes through all the training data.)"

We have tried the usage of neural networks for developing learning control schemes for repetitive manufacturing processes. These processes include machining, milling, robotics, etc. In the neural networks approach to solving the learning control problem in for instance machining, a scrap piece has to be machined to provide the neural network with a training set of data. If a slow learning neural network is used, it would generate a great deal of waste in scrap pieces to achieve learning. This problem makes it imperative to develop new learning algorithms for neural networks with orders of magnitude of faster learning compared to the Back-Propagation technique. Also, in applications such as speech or hand writing recognition, the one percent error margin of the Back-Propagation learning is no longer acceptable and a more precise convergence would be desirable. Once new faster converging and more accurate learning algorithms are present, these problems and similar problems in

other applications of neural networks could be alleviated, thus making neural networks much more practical.

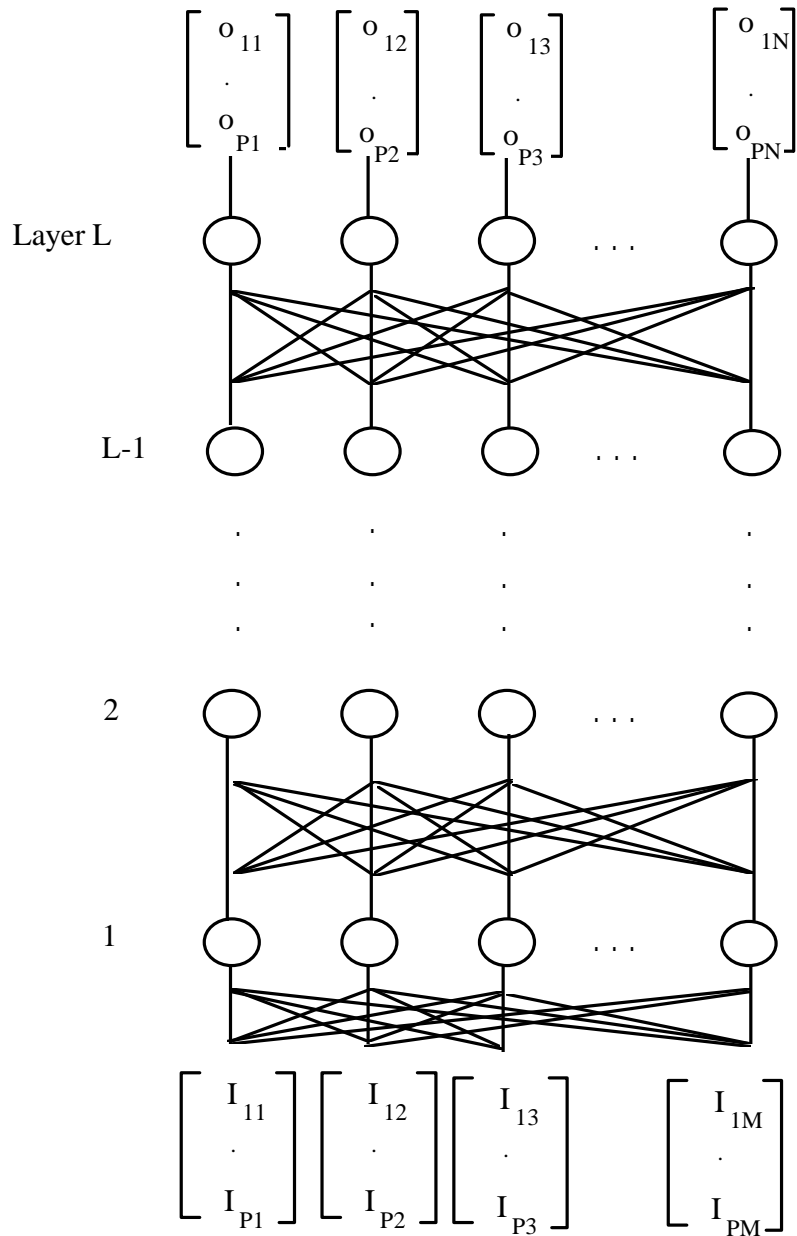
In the literature, several methods<sup>[2-3]</sup> have been proposed to increase the rate of convergence of learning by making strong assumptions such as linearity for multi-layer networks. In addition, other more practical methods have recently been proposed for speeding the convergence of the Back-Propagation technique.<sup>[4-7]</sup>

The Back-Propagation technique is a special case of the steepest descent technique with some additional assumptions. In general, steepest descent techniques perform well while away from local minima and require many iterations to converge when close to the minima. On the other hand, Newton's method usually converges fast in the vicinity of the minima. In addition, Newton's minimization technique handles functions with ill-conditioned Hessian matrices elegantly [8]. It would be desirable to take advantage of the properties of steepest descent when the state is far from minimal and then to use Newton's method in the vicinity of the minimum.

To use Newton's method, the first gradient and the matrix of second partial derivatives (Hessian) matrix should be evaluated. The moment one talks about evaluating the Hessian matrix, it becomes clear that a layer by layer adjustment of the weights is not possible because there are elements of the Hessian which are related to neurons in different layers. In general, two difficulties have prohibited the use of Newton's method for neural network learning: 1) the complexity of the evaluation of the Hessian, and 2) the inversion of the Hessian. One way to alleviate these difficulties is to use a momentum method which would approximate the diagonal

elements of the Hessian matrix and would stay ignorant of the off-diagonal elements.[9]

On the other hand, Quasi-Newton methods provide another solution to the problem, by providing an iterative estimate for the inverse of the Hessian matrix. If one



*figure 1.1 (General Multi-Layer Feed-Forward Neural Network)*

selects the initial estimate to be an identity matrix, initially, the method coincides with the steepest descent technique and gradually changes into Newton's method as the estimate approaches the inverse of the Hessian. This thesis will first review the state-of-the-art in Quasi-Newton and conjugate gradient methods and will develop new learning methods based on these minimization techniques, increasing the speed and accuracy of learning of feedforward neural networks drastically. Then, these newly developed learning methods will be evaluated through simulations.

In many cases such as in control problems, one might try to train a neural network which might include one or more neurons with unknown activation functions, such as the case when an unknown nonlinear plant to be controlled is assumed to be a neuron among several other neurons for which the activation functions are known. In such cases, the whole network should be trained without any knowledge of the functions and gradients of individual neurons. In gradient based learning techniques, the gradient evaluation requires knowledge of the activation functions of all neurons including the controlled plant. In some other cases, activation functions are desired which might not have continuous derivatives. One such activation function is the threshold function or the step function which is thought to be used in biological neurons. For these, functions, gradient based techniques will fail. In these cases, the step function is approximated with a logistic or similar smooth function. These and some other special and very important problems could be alleviated if fast gradient-free learning techniques were available.

A series of gradient-free minimization schemes are reviewed and used to develop new learning algorithms for neural networks to be applicable to the

aforementioned problems and to minimize the need for extra software/hardware. These methods do not require the evaluation of the gradient or Hessian of the objective function and only require the objective function evaluation which can be performed by the network itself. This property makes learning algorithms based on gradient-free techniques in some sense independent of the structure and connectivity of neurons in the hidden layers. An attractive consequence of this independence is that upon possible changes in the configuration, of the network, the network could still be trained in the same manner. As an example of this feature, consider a neural network with several neurons which uses some gradient learning technique. For this technique to be used, an algorithm should be developed which would include theoretical formulas for the gradient of the network. At once, however, a neuron becomes disabled or a connection is severed. The gradient based learning algorithm is no longer valid and cannot be used for learning. In this case, either the network should be thrown away or a new algorithm should be formulated for its new structure. The gradient-free techniques, however, would operate with either structure and no change is necessary to the network or to the learning algorithm. This is another practical feature which is sought by developing fast gradient-free learning techniques. The new learning methods will be evaluated through simulations.

## **1.2. Introduction to Discrete-Time Learning Control**

A large percentage of the practical applications of control systems involve cases in which a system is repeatedly asked to perform the same task. Examples include tracking problems for robots on assembly lines, as well as a large number of manufacturing applications. Standard controller design methods produce systems that repeat the same errors everytime the command is repeated. It is a bit primitive to persist in repeating the same errors. In the last few years several theorems of learning

control have been developed in the literature, generating controllers which can learn from previous experience at performing a specific task<sup>[10-15]</sup>. Two learning control algorithms are proposed here.

The first algorithm minimizes the output errors using a generalized secant method for recursive identification. In this approach, a time variant, discrete-time model of the control system is devised in the form of a system of linear algebraic equations relating the change in the state of the system to the change in the control action from one repetition of the task to another. This set of linear equations gives the transition between any two repetitions. This system is then solved for the appropriate control action that will minimize the tracking error of the controlled dynamic system, only requiring the availability of the order of the system, without any prior knowledge of the system parameters. This leads to a learning-adaptive controller, the performance of which is tested by applying it in simulation to controlling two different nonlinear plants.

In a second approach, a self-tuning (adaptive) controller based on the recursive least square parameter estimation is presented. The learning recursive least square parameter estimator applies the recursive least square parameter estimator, which is normally used in self-tuning regulators, in the repetition domain for each discrete time step. This will allow learning to be established. In applying this learning-adaptive controller, again, the order of the system should be known, but no prior knowledge of the system parameters is required. The performance of this control scheme is tested by simulation and experiment in controlling three different nonlinear plants.

### 1.3. Thesis Structure

This thesis consists of two major parts. In these two parts, the problems of learning in neural networks and learning control are treated, respectively.

Part I (chapters 2-6) presents several new learning schemes for neural networks. Chapter 2 formulates the neural network learning problem as a minimization problem. Chapter 3 applies gradient-based minimization techniques on this problem to develop fast new learning algorithms for the adjustment of neural network intercellular weights and activation function parameters. In chapter 4, more implementable NN learning algorithms are developed based on gradient-free minimization schemes. These methods are in general independent of the architecture of the neural networks. Chapters 5 and 6 discuss and conclude the simulation results conducted for these new learning algorithms. Simulation results show two to three orders of magnitude improvement on the rate of learning when compared with present state-of-the-art learning techniques.

Part II (chapters 7 and 8) presents two new learning control algorithms. In Chapter 7, an optimization approach is taken in solving the learning control problem. An objective function is formulated for the general time-variant learning control problem so that the tracking error of the controlled system can be minimized repetition after repetition. Using this objective function, a learning-adaptive control scheme is generated based on the Generalized Secant Method and is shown in simulation results, as applied to the control of two nonlinear dynamic systems performing highly nonlinear tasks, to perform very well. Chapter 8 uses a different approach for solving the learning control problem. A learning parameter estimator based on the recursive least squares algorithm is developed which in conjunction with

a suitable control law forms a learning self-tuning regulator. Simulations of this learning parameter estimator have also been conducted on controlling the same nonlinear plants as in chapter 7 and have shown to be very effective. In addition, very good experimental results have been obtained in the application of this learning control technique to the control of a Piezoelectric tool in a diamond cutting lathe with highly nonlinear dynamics including hysteresis.

Appendix A provides a summary of the mathematics required for the formulations of parts I and II. Finally, Appendix B provides a list of the abbreviations used in this thesis.

## Part I: LEARNING IN NEURAL NETWORKS

## **Chapter 2**

### **Objective Function and Gradient Vector Formulation for Learning in Feed-Forward Neural Networks**

The objective of learning, in the problem of learning in neural networks, is to minimize the output error of the top (output) layer (layer  $L$  in figure 1.1) of a neural network over a set of  $P$  input-output patterns.

Define,

$l \in [ 1, L ]$	(Layer number in the network)
$n_l \in [ 1, N_l ]$	(Neuron number in layer $l$ )
$p \in [1, P]$	(Pattern number)
$\omega_{nm}^l$	(Weighting factor between the $m^{\text{th}}$ input and neuron $n$ in layer $l$ )
$o_{pn}^l$	(Output of neuron $n$ of layer $l$ for input pattern $p$ )
$t_{pn}$	(Desired output of neuron $n$ in layer $L$ )
$i_{pm}$	(Input $m$ of pattern $p$ to the network)

Then, the objective of learning becomes,

$$\text{minimize } E = \sum_{p=1}^P \sum_{n_L=1}^{N_L} (o_{pn_L}^L - t_{pn_L})^2 \quad (2.1)$$

Define,

$$E_p = \sum_{n_L=1}^{N_L} (o_{pn_L}^L - t_{pn_L})^2 \quad (2.2)$$

then,

$$E = \sum_{p=1}^P E_p \quad (2.3)$$

Let us define a few variables and eventually a state vector which would include all the variables to be optimized for minimum  $E$ :

$$\phi^l = [\phi_1^l, \phi_2^l, \dots, \phi_{N_l}^l] \quad (\text{Activation function parameter vector for level } l)$$

$$\omega_{n_l}^l = [\omega_{n_l 1}^l, \omega_{n_l 2}^l, \dots, \omega_{n_l N_{l-1}}^l] \quad (\text{Vector of intercellular weights to neuron } n_l \text{ at layer } l)$$

$$\omega^l = [\omega_1^l, \omega_2^l, \dots, \omega_{N_l}^l] \quad (\text{Supervector of intercellular weights of level } l)$$

$$x^l = [\phi^l, \omega^l] \quad (\text{State vector for level } l)$$

$$x^T = [x^1, x^2, \dots, x^L] \quad (\text{Super state vector})$$

Let  $j$  denote any element of the state vector  $x$ , then,

$$\frac{\partial E}{\partial x_j} = \sum_{p=1}^P \frac{\partial E_p}{\partial x_j} \quad (2.4)$$

and by the chain rule,

$$\frac{\partial E_p}{\partial x_j} = 2 \sum_{n_L=1}^{N_L} (o_{pn_L}^L - t_{pn_L}^L) \frac{\partial o_{pn_L}^L}{\partial x_j} \quad (2.5)$$

where,

$$\frac{\bullet o_{pn_L}^L}{\bullet \phi_{n_l}^l} = \frac{\bullet o_{pn_L}^L}{\bullet o_{pn_l}^l} \frac{\bullet o_{pn_l}^l}{\bullet \phi_{n_l}^l} \quad (2.6a)$$

$$\frac{\bullet o_{pn_L}^L}{\bullet \omega_{n_l n_{l-1}}^l} = \frac{\bullet o_{pn_L}^L}{\bullet o_{pn_l}^l} \frac{\bullet o_{pn_l}^l}{\bullet \omega_{n_l n_{l-1}}^l} \quad (2.6b)$$

and,

$$\frac{\bullet o_{pn_L}^L}{\bullet o_{pn_l}^l} = \prod_{i=l+1}^L \frac{\bullet o_{pn_{(L+i)}}^{(L+i+1)}}{\bullet o_{pn_{(L+i)}}^{(L+i)}} \quad (\text{using indicial notation}) \quad (2.7)$$

In equation (2.7) the index notation has been employed, i.e.,

$$\frac{\overset{l+1}{\bullet}O_{pn_{l+1}} \overset{l}{\bullet}O_{pn_l}}{\overset{l}{\bullet}O_{pn_l} \overset{l-1}{\bullet}O_{pn_{l-1}}} \leftrightarrow \sum_{n_l=1}^{N_l} \frac{\overset{l+1}{\bullet}O_{pn_{l+1}} \overset{l}{\bullet}O_{pn_l}}{\overset{l}{\bullet}O_{pn_l} \overset{l-1}{\bullet}O_{pn_{l-1}}} \text{ (traditional notation)}$$

Take, for example, the logistic function,

$$O_{pn_l}^l = \frac{1}{1 + e^{-s_{pn_l}^l}} \quad (2.8)$$

where,

$$s_{pn_l}^l = \sum_{n_{l-1}=1}^{N_{l-1}} (O_{pn_{l-1}}^{l-1} \omega_{n_l n_{l-1}}^l) + \phi_{n_l}^l \quad (2.9)$$

then,

$$\frac{\overset{l}{\bullet}O_{pn_l}^l}{\overset{l}{\bullet}\phi_{n_l}^l} = d_{pn_l}^l \quad (2.10)$$

$$\frac{\overset{l}{\bullet}O_{pn_l}^l}{\overset{l}{\bullet}\omega_{n_l n_{l-1}}^l} = d_{pn_l}^l O_{pn_{l-1}}^{l-1} \quad (2.11)$$

and,

$$\frac{\overset{l}{\bullet}O_{pn_l}^l}{\overset{l-1}{\bullet}O_{pn_{l-1}}^{l-1}} = d_{pn_l}^l \omega_{n_l n_{l-1}}^l \quad (2.12)$$

where,

$$d_{pn_l}^l = \frac{e^{-s_{pn_l}^l}}{(1 + e^{-s_{pn_l}^l})^2} \quad (2.13)$$

However,

$$o_{pn_l}^l = \frac{1}{1 + e^{-s_{pn_l}^l}} \quad (2.14)$$

from which,

$$e^{-s_{pn_l}^l} = \frac{1}{o_{pn_l}^l} - 1 \quad (2.15)$$

Rewrite equation (2.13) using (2.15),

$$d_{pn_l}^l = o_{pn_l}^l (1 - o_{pn_l}^l) \quad (2.16)$$

From (2.16), (2.10), (2.11), and (2.12),

$$\frac{\bullet o_{pn_l}^l}{\bullet \phi_{n_l}^l} = o_{pn_l}^l (1 - o_{pn_l}^l) \quad (2.17)$$

$$\frac{\bullet o_{pn_l}^l}{\bullet \omega_{n_l n_{l-1}}^l} = o_{pn_{l-1}}^{l-1} o_{pn_l}^l (1 - o_{pn_l}^l) \quad (2.18)$$

and,

$$\frac{\bullet o_{pn_l}^l}{\bullet o_{pn_{l-1}}^{l-1}} = \omega_{n_l n_{l-1}}^l o_{pn_l}^l (1 - o_{pn_l}^l) \quad (2.19)$$

We can therefore use equations 2.17-2.19 and 2.4-2.7 to evaluate the elements of gradient of E with respect to the super state vector  $x$  ( $\nabla_x E$ ). Define,

$$g = \nabla_x E = \sum_{p=1}^P \nabla_x E_p \quad (\text{Gradient Vector}) \quad (2.20)$$

and,

$$G = \nabla_x^2 E = \sum_{p=1}^P \nabla_x^2 E_p \quad (\text{Hessian Matrix}) \quad (2.21)$$

Most minimization techniques require gradient evaluations of the objective function. The above formulation is used in most of the minimization techniques discussed in this thesis.

### Example: The Exclusive-OR (XOR) Problem

Take the example of a network of three neurons (figure 2.1) which is employed to simulate the exclusive-OR (XOR) logic pattern (table 2.1.)

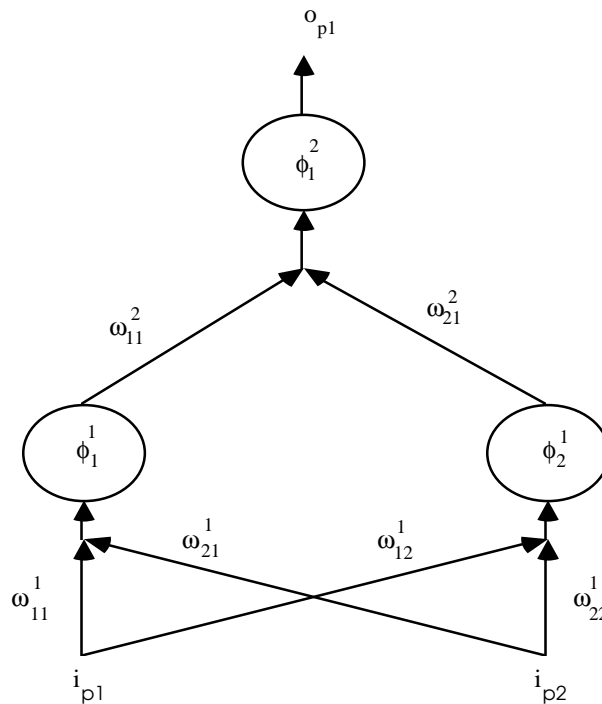


figure 2.1 (Neural Network Simulating the Exclusive-OR Logic)

Input Patterns		Output Patterns
0	0	0
1	0	1
0	1	1
1	1	0

table 2.1 (XOR Logic)

The objective function of minimization will then be,

$$E = \sum_{p=1}^4 (o_{p1}^2 - t_{p1})^2 \quad (2.22)$$

and the super state vector is defined by the following sequence of definitions,

$$\phi^{1T} = [\phi_1^1, \phi_2^1]$$

$$\phi^2 = [\phi_1^2]$$

$$\omega^{1T} = [\omega_{11}^1, \omega_{12}^1, \omega_{21}^1, \omega_{22}^1]$$

$$\omega^{2T} = [\omega_{11}^2, \omega_{12}^2]$$

$$x^T = [\phi_1^1, \phi_2^1, \omega_{11}^1, \omega_{12}^1, \omega_{21}^1, \omega_{22}^1, \phi_1^2, \omega_{11}^2, \omega_{12}^2]$$

From equation (2.5),

$$\frac{\bullet E_p}{\bullet x_j} = 2 (o_{p1}^2 - t_{p1}) \frac{\bullet o_{p1}^2}{\bullet x_j} \quad (2.23)$$

and from equations 2.17-2.19,

$$\frac{\bullet o_{p1}^2}{\bullet \phi_1^2} = o_{p1}^2 (1 - o_{p1}^2) \quad (2.24)$$

$$\frac{\bullet o_{p1}^2}{\bullet \omega_{11}^2} = o_{p1}^1 o_{p1}^2 (1 - o_{p1}^2) \quad (2.25)$$

$$\frac{\bullet o_{p1}^2}{\bullet \omega_{12}^2} = o_{p2}^1 o_{p1}^2 (1 - o_{p1}^2) \quad (2.26)$$

$$\frac{\bullet o_{p1}^2}{\bullet \phi_1^1} = \frac{\bullet o_{p1}^2}{\bullet o_{p1}^1} \frac{\bullet o_{p1}^1}{\bullet \phi_1^1} = \omega_{11}^2 o_{p1}^2 (1 - o_{p1}^2) o_{p1}^1 (1 - o_{p1}^1) \quad (2.27)$$

$$\frac{\bullet o_{p1}^2}{\bullet \phi_2^1} = \frac{\bullet o_{p1}^2}{\bullet o_{p2}^1} \frac{\bullet o_{p2}^1}{\bullet \phi_2^1} = \omega_{12}^2 o_{p1}^2 (1 - o_{p1}^2) o_{p2}^1 (1 - o_{p2}^1) \quad (2.28)$$

$$\frac{\bullet o_{p1}^2}{\bullet \omega_{11}^1} = \frac{\bullet o_{p1}^2}{\bullet o_{p1}^1} \frac{\bullet o_{p1}^1}{\bullet \omega_{11}^1} = \omega_{11}^2 o_{p1}^2 (1 - o_{p1}^2) i_{p1}^1 o_{p1}^1 (1 - o_{p1}^1) \quad (2.29)$$

$$\frac{\bullet o_{p1}^2}{\bullet \omega_{12}^1} = \frac{\bullet o_{p1}^2}{\bullet o_{p1}^1} \frac{\bullet o_{p1}^1}{\bullet \omega_{12}^1} = \omega_{11}^2 o_{p1}^2 (1 - o_{p1}^2) i_{p2}^1 o_{p1}^1 (1 - o_{p1}^1) \quad (2.30)$$

$$\frac{\bullet o_{p1}^2}{\bullet \omega_{21}^1} = \frac{\bullet o_{p1}^2}{\bullet o_{p2}^1} \frac{\bullet o_{p2}^1}{\bullet \omega_{21}^1} = \omega_{12}^2 o_{p1}^2 (1 - o_{p1}^2) i_{p1}^1 o_{p2}^1 (1 - o_{p2}^1) \quad (2.31)$$

$$\frac{\bullet o_{p1}^2}{\bullet \omega_{22}^1} = \frac{\bullet o_{p1}^2}{\bullet o_{p2}^1} \frac{\bullet o_{p2}^1}{\bullet \omega_{22}^1} = \omega_{12}^2 o_{p1}^2 (1 - o_{p1}^2) i_{p2}^1 o_{p2}^1 (1 - o_{p2}^1) \quad (2.32)$$

Using the above equations, the elements of  $\nabla_x E_p$  and using equation (2.4) the elements of  $\nabla_x E$  are found.

# **Chapter 3**

## **Gradient-Based Minimization Algorithms**

This Chapter describes a set of gradient based minimization techniques which require the evaluation of the objective function, its gradient and sometimes the matrix of its second partial derivatives (Hessian) for certain states. Another group of minimization techniques are described in chapter 4 which require evaluations of the objective function only. In the latter methods, no direct gradient evaluations are needed.

### 3.1. The Steepest Descent Technique

The steepest descent technique is a gradient-minimization method which uses a first order approximation to the objective function (E) to generate directions of descent with the knowledge of the gradient at each iteration  $k$ . The steepest descent method is very reliable since it always provides a descent direction. This technique is ideal for points which are far away from the local minima. However, close to the local minima, the steepest descent technique will generally require lots of iterations to converge due to its nature of approximating the objective function with a linear function.

Let  $x^*$  denote the state vector which results in a minimum objective function E. Further, assume that the current state is  $x_k$  and define  $\Delta x_k$  to be the difference between the states at  $k$  and  $k+1$  iterations, namely,

$$\Delta x_k = x_{k+1} - x_k \quad (3.1)$$

Decompose the step  $\Delta x_k$  into a direction  $s_k$  and a magnitude  $\lambda_k$ ,

$$\Delta x_k = \lambda_k s_k \quad (3.2)$$

Since the gradient of E points in the ascent direction, the steepest descent is given by the direction opposite to the gradient direction at every step  $k$ . Therefore for steepest descent minimization,

$$s_k = - \frac{\mathbf{g}_k}{\|\mathbf{g}_k\|} \quad (3.3)$$

However, a line search should be used to provide an optimum step size  $\lambda_k^*$  to minimize  $E$  in the  $s_k$  direction. This will produce the following recursive steepest descent technique for minimization of the sum of squares of errors of the output of a neural network.

$$x_{k+1} = x_k - \lambda_k^* \frac{\mathbf{g}_k}{\|\mathbf{g}_k\|} \quad (3.4)$$

The steepest descent technique of equation (3.4) is different from the Back-Propagation<sup>[1]</sup> technique in the sense that it uses the true gradient of  $E$  at step  $k$  and not an approximation. The Back-Propagation technique finds a descent direction based on an approximation to the gradient of the objective function by using only local data in a multi-layer network. Furthermore, the early implementations of the Back-Propagation technique took steps based on the gradient of  $E_p$  for each pattern  $p$  as soon as a new pattern was presented. The true steepest descent is done, however, by summing the gradients  $\nabla E_p$  over all the  $P$  patterns and taking a final step in the direction opposite to the gradient of  $E$ ,  $\nabla E$ . The steepest descent method only provides good minimizing directions if the condition number of the Hessian of the objective function is close to 1. As this condition number gets larger, the steepest descent becomes slower and more advanced minimization techniques (preferably with quadratic convergence) will be preferred. To further enhance the speed of learning of the network, let us consider the following more advanced minimization techniques.

### 3.2. Newton's Minimization Technique

Write the Taylor series expansion of  $E$  at  $x^*$  and about  $x_k$  assuming that every update of the state vector should drive the state vector to optimal value  $x^*$ , namely,  $x^* = x_k + \Delta x_k$ .

$$E(x^*) = E(x_k) + \nabla_x^T E|_{x_k}^T x_k + \frac{1}{2} x_k^T \nabla_x^2 E|_{x_k} x_k + O(x_k^3)$$

By using the previously defined symbols for the first and second gradients of  $E$  ( $g$  and  $G$ .) with a subscript  $k$  to denote their evaluation at  $x_k$ , we may write (3.5) as,

$$E(x^*) = E(x_k) + g_k^T x_k + \frac{1}{2} x_k^T G_k x_k + O(x_k^3) \quad (3.6)$$

If we disregard the higher than second order terms in (3.6) and thus approximate  $E$  with a quadratic function in the vicinity of  $x_k$  and  $x^*$ , then the quadratic approximation of (3.6) may be written as follows:

$$E(x^*) - E(x_k) + g_k^T x_k + \frac{1}{2} x_k^T G_k x_k \quad (3.7)$$

Note that for a minimum of  $E(x^*)$ , a necessary condition is that  $\nabla_{x^*} E$  be zero. However, keeping the current state  $x_k$  constant and then taking the gradients of both sides of (3.7), since  $x^* = x_k + \Delta x_k$ ,

$$\nabla_{x^*} E - g_k + G \Delta x_k \quad (3.8)$$

Setting the gradient of  $E$  at  $x^*$  equal to zero gives,

$$g_k + G \Delta x_k - 0 \quad (3.9)$$

or,

$$\Delta x_k = -G_k^{-1} g_k \quad (3.10)$$

However, since  $E$  is generally not a quadratic function in  $x_k$ , the following recursive update could be used for the state vector,

$$x_{k+1} = x_k + \lambda_k^* s_k \quad (3.11)$$

where  $\lambda_k^*$  is the optimum step size in the direction  $s_k$ , and  $s_k$  is given by the following,

$$s_k = -\frac{G_k^{-1} g_k}{\|G_k^{-1} g_k\|} \quad (3.12)$$

A line search method could be used to provide  $\lambda_k^*$  for direction  $s_k$ .

This method provides quadratic convergence and is very efficient in the vicinity of the minima. However, there are three problems that are faced when trying to use this algorithm. The first problem is that in order for  $E$  to always descend in value, the matrix  $G^{-1}$  should be positive definite. Since  $E$  is generally not quadratic,  $G^{-1}$  could become indefinite or even negative definite. There are many techniques developed to keep a positive definite approximation of the inverse Hessian matrix ( $G^{-1}$ ) such that the quadratic information in the Hessian matrix will be used. Using the quadratic information generally provides a better direction of descent than the steepest descent direction, especially in the vicinity of the minima. Among the methods for keeping a positive definite approximation of the inverse Hessian matrix are Greenstadt's method<sup>[16]</sup>, Marquardt<sup>[17]</sup>, Levenberg<sup>[18]</sup>, and Goldfeld, Quandt and Tratter's alternative<sup>[19]</sup>.

A second problem is that for networks with a small number of neurons it might be feasible to find the Hessian matrix, however, for larger networks it will become a very difficult task. In addition, it will be very hard to write general equations for the evaluation of the elements of the Hessian matrix as done for the elements of the gradient vector  $g$ .

Let us for the sake of argument say that a Hessian matrix is calculated at every iteration  $k$ . Then a still more serious problem occurs. For huge networks it is not practical to take the inverse of the Hessian matrix. Taking this inverse in most cases will require more time than taking more steps using a simpler method such as the steepest descent method.

The problem of retainment of positive definiteness of the inverse Hessian can be solved by the methods noted above. However, problems two and three make using Newton's method quite impractical. These limitations are reasons for looking at the following alternatives which in turn will solve the aforementioned problems and still keep a super-linear rate of convergence.

### 3.3. Quasi-Newton or Large Step Gradient Techniques

From equation (3.11), we can write the following generalized recursive algorithm to update the state vector such that a minimum  $E$  will be approached:

$$x_{k+1} = x_k - \lambda_k^* H_k \nabla_x E(k) \quad (3.13)$$

where  $\lambda^*$  is a weighting factor, and  $H_k$  is a square symmetric matrix. Depending on the choice of  $H_k$ , different optimization algorithms will be resulted. Therefore,  $H_k$  multiplied by the gradient of  $E$  will provide a direction of descent in the objective function  $E$  and  $\lambda_k^*$  is the optimal step in that descent direction as provided by some line search method. If  $H_k$  in equation (3.13) is made equivalent to the identity ( $I$ ) matrix, then the method reduces to the steepest descent technique which provides linear convergence. Making  $H_k$  equivalent to the inverse of the Hessian matrix ( $G^{-1}$ ) of the quadratic approximation of  $E$ , as previously defined, the method will reduce to the Newton minimization technique which provides quadratic convergence.

Instead of using the real inverse-Hessian, Quasi-Newton methods use an approximation to the inverse-Hessian provided by an iterative updating scheme. Quasi-Newton methods usually start with an approximation to the inverse-Hessian matrix such as the identity matrix. Different updates for  $H_k$  are then used, leading to different types of Quasi-Newton methods. Updates to matrix  $H_k$  are done recursively

in different directions of the inverse-Hessian space, based on the information obtained from the function and gradient behavior in that direction. Depending on whether these updates are done in one or two directions at a time, rank one or rank two methods are generated. Those Quasi-Newton methods which retain a positive definite  $H_k$  are called variable metric methods. Not all Quasi-Newton methods use variable metric updates. Newton-like methods in general try to keep Newton's condition (3.14) satisfied.

$$H^{-2} g_k = -x_k \quad (3.14)$$

Condition (3.14) is automatically satisfied for a quadratic function if  $H$  is the exact inverse Hessian. However, since in Quasi-Newton methods, the inverse Hessian is supposed to be approximated, instead of  $H_k^{-1} g_k = -x_k$ , the methods try to keep the following relation satisfied at each step  $k$ ,

$$H_{k+1}^{-2} g_k = -x_k \quad (3.15)$$

This relationship is referred to as the Quasi-Newton condition and it means that the inverse Hessian matrix should be updated such that relation (3.15) is satisfied.

In 1959, Davidon<sup>[20]</sup> introduced the idea of Quasi-Newton methods. In 1965, Barnes<sup>[21]</sup> and Broyden<sup>[22]</sup> independently introduced a method for solving a set of simultaneous linear equations of the same form as equation (3.14). Barnes' equation is a more general one and includes Broyden's method as a special case. Equation (3.16) gives this update,

$$H_{k+1}^{-2} = \frac{(x_k - H_k^{-2} g_k)^T z_k}{z_k^T g_k} \quad (3.16)$$

where  $z_k$  is a direction in which the update to  $H_k$  is done.

### 3.3.1. Rank-One Updates to Inverse Hessian

A rank-one update to the inverse Hessian matrix  $H_k$  would mean the following,

$$H_{k+1} = H_k + \alpha u u^T$$

where  $u$  is a direction of update. Setting  $z_k$  in equation (3.16) equal to the error of equation (3.14), when  $H$  is approximated by  $H_k$ , will produce Broyden's rank-one update (Eq. 3.17).

$${}^2 H_k = \frac{({}^2 x_k - H_k {}^2 g_k) ({}^2 x_k - H_k {}^2 g_k)^T}{{}^2 g_k^T ({}^2 x_k - H_k {}^2 g_k)} \quad (3.17)$$

This update as applied to Quasi-Newton problems was introduced by Broyden<sup>[23]</sup>, Davidon<sup>[24]</sup> and others<sup>[25, 26]</sup> independently. Let  $N$  denote the dimension of the state vector  $x$ . This update has the property that if  $\mathcal{A}x_1, \mathcal{A}x_2, \dots, \mathcal{A}x_N$  are linearly independent, then at  $k=N+1$ ,  $H_k = G^{-1}$  for quadratic functions.

Another important feature of Broyden's update is that  $\lambda_k^*$  of equation (3.13) does not necessarily have to minimize  $E$  in the  $s_k$  direction. As long as  $\lambda_k^*$  is such that  $H_{k+1}$  will not become singular and the denominator of (3.17) is not made zero, any  $\lambda_k^*$  could be used in conjunction with the Broyden update. However, some unattractive features also exist for this update. If the objective function is non-quadratic, as in the case of general neural networks, the following less than satisfactory aspects of Broyden update exist,

1.  $H_k$  may not retain its positive definiteness in which case it is necessary to use one of the methods in section 3.2, such as Greenstadt's method, to force this matrix to be positive definite.

2. The correction  $\Delta H_k$  of equation (3.17) may sometimes become unbounded (sometimes even for quadratic functions, due to round-off errors).

3. If  $\Delta x_k$  given by equation (3.13) is by chance in the same direction as  $\Delta x_{k-1}$ , then  $H_{k+1}$  becomes singular or undetermined. Therefore, if

$$H_k^2 g_k = \Delta x_k \quad (3.18)$$

or,

$$(H_k^2 g_k - \Delta x_k)^T g_k = 0 \quad (3.19)$$

then  $H_{k+1}$  should be made equal to  $H_k$ , namely  $\Delta H_k = 0$ .

### 3.3.2. Pearson's Updates

Pearson<sup>[27]</sup> introduced other directions of update for the projection of the error of equation (3.14). In his No.2 method he proposed  $z_k = \Delta x_k$  in equation (3.16), which in turn generated the following update for  $H_k$ :

$${}^2 H_k = \frac{(\Delta x_k - H_k^2 g_k)^T \Delta x_k}{\Delta x_k^T g_k} \quad (3.20)$$

His No.3 method used the other possibility which is  $z_k = H_k \Delta g_k$ . This gives the following update,

$${}^2 H_k = \frac{(\Delta x_k - H_k^2 g_k) (H_k^2 g_k)^T}{(H_k^2 g_k)^T g_k} \quad (3.21)$$

Pearson's methods do not guarantee positive definiteness of the inverse Hessian Matrix and usually lead to ill-conditioned matrices. Therefore, it is a good idea to reset the inverse Hessian approximation to Identity, every N iterations (i.e.  $H_{(Nt)} = I$  for  $t = 0, 1, 2, \dots$ ).

### 3.3.3. Rank-Two Updates

The rank-one update does not leave much of a freedom for choosing the directions of update. This motivated the formulation of rank-two updates such that the objective is still to satisfy relation (3.15) at every step  $k$ . The general rank two updates are of the following form,

$$H_{k+1} = H_k + \alpha u u^T + \beta v v^T \quad (3.22)$$

where, the directions  $u$  and  $v$  and scaling factors  $\alpha$  and  $\beta$  should be picked. It would be a good idea to update these directions relative to  $\Delta x_k$  and  $H_k \Delta g_k$ . This would give the following general update,

$$H_k = \alpha \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T \Delta g_k} + \beta \frac{H_k \Delta g_k \Delta g_k^T H_k}{\Delta g_k^T H_k \Delta g_k} \quad (3.23)$$

A natural choice to retain symmetry is to pick  $\alpha = 1$ ,  $\beta = -1$ ,  $y = \Delta x_k$ , and  $z = H_k \Delta g_k$ . This will generate the Davidon-Fletcher-Powell (DFP)<sup>[28]</sup> update given by equation (3.24).

$$\Delta H_k = \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T \Delta g_k} - \frac{H_k \Delta g_k \Delta g_k^T H_k}{\Delta g_k^T H_k \Delta g_k} \quad (3.24)$$

This algorithm works properly in general if  $g_k$  is calculated with minimal error and  $H_k$  does not become ill-conditioned. Define,

$$A_i = \frac{\Delta x_i \Delta x_i^T}{\Delta x_i^T \Delta g_i} \quad (3.25)$$

and,

$$B_i = \frac{H_i^T g_i^T g_i H_i}{g_i^T H_i^T g_i} \quad (3.26)$$

Then,

$$\sum_{i=0}^{k-1} A_i \rightarrow H \quad \text{as } k \rightarrow N \quad (3.27)$$

and,

$$\sum_{i=0}^{k-1} B_i \rightarrow H_0 \quad \text{as } k \rightarrow N \quad (3.28)$$

which result in,

$$H_k \rightarrow H \quad \text{as } k \rightarrow N \quad (3.29)$$

for a quadratic function. The proof of this statement is as follows.

Using (3.14), substitute for  $\mathcal{A}g_i$  in equation (3.25),

$$A_i = \frac{x_i^T x_i}{x_i^T G x_i} \quad (3.30)$$

Sum  $A_i$  over  $N$  consecutive steps,

$$\sum_{i=0}^{N-1} A_i = \sum_{i=0}^{N-1} \frac{x_i^T x_i}{x_i^T G x_i} \quad (3.31)$$

$$\sum_{i=0}^{N-1} A_i = \frac{\lambda_i^* s_i^T \lambda_i}{\lambda_i^* s_i^T G s_i \lambda_i} = \frac{s_i^T s_i}{s_i^T G s_i} \quad (3.32)$$

Therefore, for quadratic functions when  $s_i$  are conjugate about  $G$  (see (A.38)),

$$\sum_{i=0}^{N-1} A_i = H \quad (3.33)$$

The DFP method provides  $s_i$  which are conjugate about  $G$  and thus (3.33) holds. Similarly,  $\Sigma B_i$  can be shown to approach  $H_0$  and to keep  $H_k$  positive definite as  $k \rightarrow N$ . An important property of this update is that if  $\Delta x_k^T \Delta g_k > 0$  for all  $k$ , then the approximate inverse Hessian matrix will retain its positive definiteness. This condition can be imposed by using a line search method which satisfies the following relation,

$$\Delta x_{k+1}^T \Delta g_{k+1} = \sigma \Delta x_k^T \Delta g_k \quad (3.34)$$

Eventually this means that the curvature estimate should be positive where the updating is done. In equation (3.34),

$$\sigma \in [\tau, 1]$$

and,

$$\tau \in [0, \frac{1}{2}]$$

These are parameters of the line search termination.[28]

In 1970, Broyden<sup>[29]</sup>, Fletcher<sup>[30]</sup>, Goldfarb<sup>[31]</sup>, and Shanno<sup>[32]</sup> suggested the BFGS update which is dual with the DFP update. This means that if one applies the DFP method to updating the Hessian matrix from the following,

$$G_{k+1} \Delta x_k = \Delta g_k \quad (3.35)$$

rather than equation (3.15), and then apply the Sherman-Morrison inversion formula (A.32) to obtain an expression for  $\Delta H_k$ , then the BFGS formula given by equation (3.36) will be obtained.

$$\Delta H_k = \left( 1 + \frac{\Delta g_k^T H_k \Delta g_k}{\Delta x_k^T \Delta g_k} \right) \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T \Delta g_k} - \frac{\Delta x_k \Delta g_k^T H_k + H_k \Delta g_k \Delta x_k^T}{\Delta x_k^T \Delta g_k} \quad (3.36)$$

The BFGS update has all the qualities of the DFP method plus the fact that it has been noted to work exceptionally well with inexact line searches and a global convergence proof exists<sup>[28]</sup> for the BFGS update. No such proof has been done for the convergence of DFP yet.

### 3.3.4. Quasi-Newton Updates Through Variational Means

Greenstadt<sup>[33]</sup> developed a general updating scheme using variational means by minimizing the Euclidian norm of the update to the inverse Hessian. This generated the following general updating formula,

$${}^2 H_k = \frac{1}{{}^2 g_k^T M^2 g_k} ({}^2 x_k^T {}^2 g_k M + M^2 g_k^T {}^2 x_k - H_k^T {}^2 g_k^T {}^2 g_k H_k) - \frac{1}{{}^2 g_k^T M^2 g_k} ({}^2 g_k^T {}^2 x_k - ({}^2 g_k^T H_k^T {}^2 g_k)) M^2 g_k^T {}^2 g_k M) \quad (3.37)$$

where  $M$  is a positive definite matrix. Greenstadt in his paper proposed two possibilities for  $M$  as,

1.  $M = H_k \rightarrow$  (Equation 3.38)

2.  $M = I \rightarrow$  (Equation 3.39)

$${}^2 H_k = \frac{1}{{}^2 g_k^T H_k^T {}^2 g_k} ({}^2 x_k^T {}^2 g_k H_k + H_k^T {}^2 g_k^T {}^2 x_k - (1 + \frac{{}^2 g_k^T {}^2 x_k}{{}^2 g_k^T H_k^T {}^2 g_k}) H_k^T {}^2 g_k^T {}^2 g_k H_k) \quad (3.38)$$

$${}^2 H_k = \frac{1}{{}^2 g_k^T {}^2 g_k} ({}^2 x_k^T {}^2 g_k H_k + H_k^T {}^2 g_k^T {}^2 x_k - {}^2 g_k^T {}^2 g_k H_k) - \frac{1}{{}^2 g_k^T {}^2 g_k} ({}^2 g_k^T {}^2 x_k - {}^2 g_k^T H_k^T {}^2 g_k) {}^2 g_k^T {}^2 g_k \quad (3.39)$$

These methods do not retain positive definiteness in general. However, Goldfarb<sup>[31]</sup> proposed the use of  $M = H_{k+1}$  which does provide a positive definite approximation to the inverse Hessian matrix. This method is identical to the BFGS update.

### 3.3.5. Self-Scaling Quasi-Newton Methods

Suppose that the objective function  $E$  is scaled by a number  $c$  and results in a new objective function,

$$E' = cE \quad (3.40)$$

This objective function has the same minimizer as  $E$  and its gradient and inverse Hessian are given in terms of those of  $E$  by,

$$g' = cg \quad (3.41)$$

$$H' = \left(\frac{1}{c}\right) H \quad (3.42)$$

The Newton step for finding the minimizer of a quadratic function,  $x^*$ , is  $x^* = x_k - H^{-1} g_k$ . Similarly, the Newton update for  $E'$  is,

$$x^* = x_k - H'^{-1} g'_k = x_k - \left(\frac{1}{c}\right) H^{-1} c g_k = x_k - H^{-1} g_k$$

Therefore, the Newton step is invariant under scaling while Quasi-Newton methods are generally not invariant under such scaling and will give different results.

Take Broyden's single parameter class of updates described by equations (3.43), which includes the BFGS and DFP updates as special cases.

$$H_{k+1} = H_k - \frac{H_k^{-2} g_k^{-2} g_k^T H_k^{-1}}{g_k^T H_k^{-2} g_k} + \theta_k \frac{v_k v_k^T}{v_k^T v_k} + \frac{x_k^{-2} x_k^T}{x_k^T x_k} \quad (3.43a)$$

where,

$$v_k = \left( g_k^T H_k g_k \right)^{\frac{1}{2}} \left( \frac{g_k^T x_k}{g_k^T H_k g_k} - \frac{H_k g_k}{g_k^T H_k g_k} \right) \quad (3.43b)$$

(Setting  $\theta_k$  to 1 in equation (3.43a) produces the BFGS update and setting it to zero gives the DFP update.)

This class of updates is generally not invariant under scaling. This motivated Oren and Spedicato<sup>[34,35]</sup> to modify this Broyden's single parameter family (3.43) by introducing a new parameter  $\mu_k$  such that by the appropriate choice of  $\mu_k$  and  $\theta_k$ , they could have an update which is invariant under scaling of equation (3.40). This is the general update given by (3.44),

$$H_{k+1} = \mu_k \left( H_k - \frac{H_k g_k g_k^T H_k}{g_k^T H_k g_k} + \theta_k v_k v_k^T \right) + \frac{g_k^T x_k x_k^T}{g_k^T H_k g_k} \quad (3.44)$$

where  $v_k$  is given by (3.43b).

In addition, Shanno and Phua<sup>[36]</sup> proposed an initial scaling method which makes Broyden's single parameter class of update self-scaling. The two approaches to making updates invariant under scaling are discussed in more detail in the following two sections.

### 3.3.5.1. Self-Scaling Variable Metric (SSVM) Algorithms

Equation (3.44), when used with exact line searches, will become a member of Huang's family<sup>[37]</sup> where in his notation,

$$\rho_k = \frac{1}{\prod_{i=0}^k \mu_i} \quad (3.45)$$

Equation (3.44) leaves a lot of freedom in choosing the parameters  $\mu_k$  and  $\theta_k$  such that invariance under scaling is achieved. Oren suggested in [34] that  $\mu_k$  and  $\theta_k$  be picked in the following manner,

$$\mu_k = \phi_k \frac{\mathbf{g}_k^T \mathbf{x}_k}{\mathbf{g}_k^T \mathbf{H}_k \mathbf{g}_k} + (1 - \phi_k) \frac{\mathbf{x}_k^T \mathbf{g}_k}{\mathbf{x}_k^T \mathbf{H}_k \mathbf{g}_k} \quad (3.46)$$

and  $\phi_k, \theta_k \in [0,1]$ . This choice will provide a set of  $\mu_k$  such that,

$$\frac{\mathbf{x}_k^T \mathbf{g}_k}{\mathbf{g}_k^T \mathbf{H}_k \mathbf{g}_k} \leq \mu_k \leq \frac{\mathbf{x}_k^T \mathbf{H}_k^{-1} \mathbf{x}_k}{\mathbf{x}_k^T \mathbf{g}_k} \quad (3.47)$$

In another approach, Oren and Spedicato<sup>[35]</sup> tried picking  $\mu_k$  and  $\theta_k$  based on heuristics such that  $\mu_k$  is as close as possible to unity and  $\theta_k$  is such as to offset an estimated bias in  $\det(\mathbf{H}_k \mathbf{G})$ . In a third approach Oren and Spedicato<sup>[35]</sup> picked those  $\mu_k$  and  $\theta_k$  which minimize the condition number of  $(\mathbf{H}_k^{-1} \mathbf{H}_{k+1})$ . This choice will put a bound on the condition number of the inverse Hessian approximate and therefore will provide numerical stability. Minimizing this condition number, the following relationship is held between  $\mu_k$  and  $\theta_k$ ,

$$\theta_k = \frac{b(c - b\mu_k)}{\mu_k(ac - b^2)} \quad (3.48a)$$

where,

$$a = \mathbf{g}_k^T \mathbf{H}_k \mathbf{g}_k \quad (3.48b)$$

$$b = \mathbf{x}_k^T \mathbf{g}_k \quad (3.48c)$$

and,

$$\begin{aligned} c &= \mathbf{x}_k^T \mathbf{H}_k^{-1} \mathbf{x}_k \\ &= \lambda_k^* \mathbf{g}_k^T \mathbf{H}_k \mathbf{g}_k \end{aligned} \quad (3.48d)$$

Then, using Fletcher's duality concept<sup>[28]</sup>, Oren and Spedicato<sup>[35]</sup> found those  $\mu_k$  and  $\theta_k$  which would make their update self-dual. This set is given by,

$$\theta_k = \frac{1}{1 + \sqrt{\frac{ac}{b^2}}} \quad (3.49a)$$

and,

$$\mu_k = \sqrt{\frac{c}{a}} \quad (3.49b)$$

Reference [35] gives four sets of switching rules for picking  $\mu_k$  and  $\theta_k$ . Algorithms based on Oren and Spedicato's updates are called Self Scaling Variable Metric (SSVM) algorithms. SSVM methods maintain positive definiteness of the approximation to the inverse Hessian matrix provided that  $\mathbf{A} \mathbf{x}_k^T \mathbf{A} \mathbf{g}_k > 0$  for all  $k$ . Condition (3.34) is again used in the line searches to impose this inequality. For a general non-linear objective function, the SSVM algorithms provide a set of search directions which are invariant under scaling of the objective function. Also, for a quadratic function, these algorithms have the property that they monotonically reduce the condition number of the inverse Hessian approximate.

A draw-back of the SSVM algorithms is that they fail to converge to the inverse Hessian matrix for a quadratic function.<sup>[36]</sup> This convergence is especially desirable for methods employed for minimizing non-quadratic objective functions.<sup>[38]</sup>

The SSVM algorithms, in general, perform well with objective functions which depend on lots of variables. This makes them ideal for usage in neural network learning. These algorithms perform exceptionally well (better than all other updates in general) for homogeneous objective functions. A homogeneous objective function  $E(x)$  is such that,

$$E(x) = \tau^{-1} (x - x^*) g(x) + E(x^*) \quad (3.50)$$

where  $\tau$  is the degree of homogeneity and  $x^*$  is the minimizing state.<sup>[39]</sup> Differentiating (3.50) gives,

$$x^* = x - (\tau - 1) H(x) g(x) \quad (3.51)$$

Equation (3.51) suggests that the Newton step should be multiplied by  $(\tau - 1)$  in order to get to the minimum. This makes the switch 2 SSVM methods superior to all other Quasi-Newton methods, when used on homogeneous functions.<sup>[34-36]</sup>

The lack of convergence of the approximate inverse Hessian to its true value in SSVM updates motivated Shanno and Phua to investigate methods which would make the Broyden single parameter class self-scaling.<sup>[36]</sup>

### 3.3.5.2. Initial Scaling of the Inverse Hessian Approximate

In [40], Spedicato proposes the initialization such that  $H_0$  is the inverse of a diagonal matrix with its diagonal being the diagonal of the true Hessian matrix at  $x_0$ . This, however, is not practical since it is very hard to evaluate the Hessian for the objective function of a multi-layer neural network. Shanno and Phua <sup>[36]</sup> proposed an initial scaling such that,

$$H_0 = \lambda_0^* H'_0 \quad (3.52)$$

where  $\lambda_0^*$  is the initial linear step given by the line search algorithm and  $H'_0$  is the initial guess for the inverse Hessian (usually the identity matrix,  $I$ ). This makes Broyden's one parameter class of updates, given by equations (3.43), self scaling and invariant under scaling of the objective function. Initial scaling of (3.52) will therefore give,

$$H_0 = \lambda_0^* I \quad (3.53)$$

as the new initial guess for the inverse Hessian, if no better estimate of  $H$  is available.

Another initial scaling, proposed by Shanno and Phua<sup>[36]</sup>, uses Oren-Spedicato's SSVM algorithm and finds the  $\mu_0$  provided by that algorithm which minimizes the condition number of  $(H_k^{-1} H_{k+1})$ . Then it scales the initial estimate of the inverse Hessian by that value. For example, consider the BFGS update for which  $\theta_0 = 1$ .  $\mu_0$  is given by equation (3.48a) to be,

$$\mu_0 = \frac{b}{a} \quad (3.54)$$

Since  $\mu$  should be equal to one for the BFGS method, the initial estimate of  $H$  is scaled by  $\mu_0$ ,

$$H_0 = \frac{b}{a} H'_0 \quad (3.55)$$

This initial scaling can be evaluated in the same manner for all the members of the Broyden single parameter class of updates using (3.48a) and the appropriate  $\theta_0$  for that update.

These initial scalings were shown by Shanno and Phua<sup>[36]</sup> to improve the performance of the BFGS method over the SSVM methods of Oren and Spedicato in all the cases tested but the special case of homogeneous objective functions.

### 3.3.6. Quasi-Newton Methods with Inexact Line Searches

Quasi-Newton methods which work well with inexact line searches such as the BFGS and SSVM methods have become very popular due to their reduction of the computational burden associated with line searches. Hoshino<sup>[41]</sup> presented, with his Quasi-Newton algorithm, a correction term which would maintain the orthogonality of the search direction and gradient at the termination point of an inexact line search.

Davidon also presented a new algorithm which has drawn a lot of attention in the field of optimization<sup>[36,42]</sup>. His algorithm uses no line searches, optimally conditions the inverse Hessian approximate, and uses the square root of the inverse Hessian approximate which improves the numerical stability of his algorithm. The following two sections describe the theoretical details of these two approaches to weaken or eliminate line searches.

### 3.3.6.1. Hoshino's Method

Hoshino<sup>[41]</sup> presented a new variable metric update which generally works well and has properties similar to those of the BFGS and the DFP methods. However, this method in general has shown to give updates with condition numbers larger than the BFGS and SSVM methods. Hoshino's update is given by equation (3.56).

$$H_{k+1} = H_k + \frac{1}{\frac{g_k^T g_k}{g_k^T H_k g_k} + \frac{g_k^T H_k g_k}{g_k^T g_k}} \left( \left[ 1 + \frac{g_k^T H_k g_k}{g_k^T g_k} \right] \frac{g_k g_k^T}{g_k^T g_k} - \frac{g_k g_k^T H_k - H_k g_k g_k^T - H_k g_k g_k^T H_k}{g_k^T g_k} \right) \quad (3.56)$$

An attractive feature of Hoshino's Quasi-Newton minimization is his theoretical approach to the use of his update with inexact line searches. Inexact line searches are desired to reduce the number of function evaluations. These inexact line searches will sometimes give gradients at their termination point which are not perpendicular to the search direction. This will slow down the convergence of the minimization scheme. To evaluate the Quasi-Newton direction of update at any step  $k+1$ , Hoshino uses a modified gradient which is forced to be perpendicular to the search direction. Consider the line search terminating at  $x_{k+1}$ . Also, consider  $x'_{k+1}$  to be the true

minimum in the direction of search. Furthermore, denote the step from  $x_k$  to the true minimum  $x'_{k+1}$  by  $\Delta x'_k$  and define the scalar  $\epsilon_k$  such that,

$$\Delta x'_k = \epsilon_k \Delta x_k \quad (3.57)$$

Therefore, the gradient at the true minimum  $g'_{k+1}$  will be given by,

$$\begin{aligned} g'_{k+1} &= g_{k+1} + (g_{k+1} - g_k) \epsilon_k \\ &= g_{k+1} + \epsilon_k \Delta g_k \end{aligned} \quad (3.58)$$

The true minimum would be at the point where the gradient is perpendicular to the direction of search or,

$$\Delta x_k^T g'_{k+1} = 0 \quad (3.59)$$

Solving for the  $\epsilon_k$  which satisfies this condition gives,

$$\epsilon_k = - \frac{\Delta x_k^T g_{k+1}}{\Delta x_k^T \Delta g_k} \quad (3.60)$$

Then, the expression for the modified gradient is given using this scalar factor by,

$$g'_{k+1} = g_{k+1} - \frac{\Delta x_k^T g_{k+1}}{\Delta x_k^T \Delta g_k} \Delta g_k \quad (3.61)$$

This new gradient gives the Quasi-Newton direction at step  $k+1$  to be,

$$s_{k+1} = - H_{k+1} g'_{k+1} \quad (3.62)$$

Hoshino does not use this modified gradient for his update to the inverse Hessian approximate. This gradient is only used to obtain the Quasi-Newton step. Reference [41] gives a stability analysis for this scheme.

### 3.3.6.2. Davidon's Optimally Conditioned Quasi-Newton Method with No Line Searches

In 1975, Davidon<sup>[42]</sup> made an important contribution to the improvement of Quasi-Newton methods by introducing his optimally conditioned method which is free of line searches. Schnabel<sup>[43]</sup> has devoted most of his PhD dissertation to evaluating Davidon's method. The method conducts updates to the inverse Hessian approximate which are optimally conditioned in the same sense as the optimal conditioning of Oren and Spedicato. This conditioning is done by minimizing the condition number of  $(H_k^{-1} H_{k+1})$  which has been obtained by minimizing,

$$\frac{\lambda_1}{\lambda_N} \text{ in the eigen value problem, } H_{k+1} u = \lambda H_k u$$

Previously, researchers had been trying to minimize the ratio of the condition number  $H_{k+1}$  to the condition number of  $H_k$ . However, doing this would generate invariance under orthogonal transformations only, while the optimal conditioning used by Davidon and Oren and Spedicato is invariant under all invertible linear transformations.

Davidon's update to the inverse Hessian approximate is given by equation (3.63). This general update includes some updates such as the DFP and BFGS updates as special cases. In equation (3.63), the value of  $\theta_k$  is chosen such that the condition number of  $(H_k^{-1} H_{k+1})$  is minimized. Davidon uses  $\nabla E x_0$  as the initial value of  $w$ , namely,

$$w_0 = \nabla E x_0$$

Following  $w$ 's are then obtained by (3.64) for quadratic functions. For non-quadratic functions, he sometimes uses,

$$w_k = \nabla E x_k$$

This update is part of Davidon's algorithm which does not use a line search. However, if sufficient reduction is not experienced by the objective function, a simple inexact line search is used to impose sufficient function reduction.

$$H_{k+1} = H_k + \frac{(\mathbf{x}_k - H_k \mathbf{g}_k)^T \mathbf{w}_k + \mathbf{w}_k^T (\mathbf{x}_k - H_k \mathbf{g}_k) - (\mathbf{x}_k - H_k \mathbf{g}_k)^T \mathbf{g}_k \mathbf{w}_k \mathbf{w}_k^T}{\mathbf{w}_k^T \mathbf{x}_k - (\mathbf{w}_k^T \mathbf{g}_k)^2} \quad (3.63)$$

$$\theta_k \left[ \frac{\mathbf{x}_k - H_k \mathbf{g}_k}{(\mathbf{x}_k - H_k \mathbf{g}_k)^T \mathbf{g}_k} - \frac{\mathbf{w}_k}{\mathbf{w}_k^T \mathbf{g}_k} \right] \left[ \frac{\mathbf{x}_k - H_k \mathbf{g}_k}{(\mathbf{x}_k - H_k \mathbf{g}_k)^T \mathbf{g}_k} - \frac{\mathbf{w}_k}{\mathbf{w}_k^T \mathbf{g}_k} \right]^T \quad (3.64)$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k (\mathbf{x}_k - H_k \mathbf{g}_k)^T \mathbf{g}_k - (\mathbf{x}_k - H_k \mathbf{g}_k) \mathbf{w}_k^T \mathbf{g}_k$$

This method has three important features. First, to improve the numerical stability and accuracy of his algorithm, Davidon updates a Jacobian matrix which is the square root of the inverse Hessian approximate.

$$H_k = J_k J_k^T \quad (3.65)$$

By this factorization, the condition number of the Jacobian matrix  $J_k$  is of the order of the square root of the condition number of the matrix  $H_k$ . This smaller condition number improves the stability of the method in practical applications. An update in the Jacobian matrix of the following form,

$$J_{k+1} = (1 + uv^T) J_k \quad (3.66)$$

translates to the following update in the inverse Hessian approximate,

$$H_{k+1} = (1 + uv^T) H_k (1 + vu^T) \quad (3.67)$$

This factorization produces positive definite inverse Hessian approximates. Some rank two updates in the Broyden family such as BFGS, DFP and optimally conditioned updates correspond to rank one updates in the Jacobian matrix. This results in fewer computational operations and less round-off error.

A second feature of this algorithm is motivated by the fact that one could approximate the gradient at the minimum of a quadratic function by a linear interpolation similar to the approach of Hoshino discussed in the previous section.<sup>[44]</sup> Davidon does not use the actual change in the gradient and the actual step size, he instead uses projections of these changes. This allows him to avoid line searches. Shanno and Phua have devoted a paper to the discussion of these projections.<sup>[45]</sup> Despite the use of these projections, Davidon's method still maintains a positive definite approximation to the inverse Hessian matrix for quadratic and non-quadratic functions. This ensures quadratic convergence for Davidon's algorithm.

In the real implementation of his algorithm, Davidon uses an equivalent form of equation (3.63) given by equation (3.68),

$$\begin{aligned}
 H_{k+1} = H_k + & \frac{(\mathbf{x}'_k - H_k \mathbf{g}'_k)^T \mathbf{x}'_k + \mathbf{x}'_k (\mathbf{x}'_k - H_k \mathbf{g}'_k)^T}{\mathbf{x}'_k \mathbf{x}'_k} \\
 & - \frac{(\mathbf{x}'_k - H_k \mathbf{g}'_k)^T \mathbf{g}'_k \mathbf{x}'_k \mathbf{x}'_k}{(\mathbf{x}'_k \mathbf{g}'_k)^2} \\
 \theta_k \mathbf{g}'_k \mathbf{g}'_k H_k \mathbf{g}'_k & \left[ \frac{\mathbf{x}'_k}{\mathbf{x}'_k \mathbf{g}'_k} - \frac{H_k \mathbf{g}'_k}{\mathbf{g}'_k H_k \mathbf{g}'_k} \right] \left[ \frac{\mathbf{x}'_k}{\mathbf{x}'_k \mathbf{g}'_k} - \frac{H_k \mathbf{g}'_k}{\mathbf{g}'_k H_k \mathbf{g}'_k} \right]^T \quad (3.68)
 \end{aligned}$$

where,

$$\mathbf{x}'_k = Q_k \mathbf{x}_k \quad (3.69)$$

and,

$$\mathbf{g}'_k = Q_k \mathbf{g}_k \quad (3.70)$$

$Q_k$  is the matrix projecting onto

$$\begin{bmatrix} \mathbf{T} \\ \mathbf{w}_k^T (\mathbf{x}_k - \mathbf{H}_k \mathbf{g}_k) \end{bmatrix}$$

in  $\mathbf{H}_k^{-1}$ .

The third feature of Davidon's algorithm is that the directions  $\mathbf{w}$  of (3.63) need not be orthogonal to the error of the Quasi-Newton step. These directions are chosen such that the updates and new directions satisfy the following conditions:

$$(\mathbf{H}_{k+1} - \mathbf{H}_k) \mathbf{z} = 0 \quad \forall \mathbf{z} \in \mathbf{Z}_H \quad (3.71)$$

$$\mathbf{w} \perp \mathbf{Z}_H \perp \mathbf{x}_k - \mathbf{H}_k \mathbf{g}_k$$

where,

$$\mathbf{Z}_H = \begin{bmatrix} \mathbf{g}_0, \dots, \mathbf{g}_{k-1} \end{bmatrix} \quad (3.72)$$

This ensures approximate inverse Hessian matrices which satisfy the following condition,

$$\mathbf{H}_{k+1} \mathbf{g}_j = \mathbf{x}_j \quad j \leq k \quad (3.73)$$

This condition ensures convergence to the minimum of a quadratic function after  $N$  updates to the inverse Hessian approximate when inexact line searches (or even no line searches) are used.

Figures 3.1 give a flow chart of Davidon's minimization algorithm without any line searches. Davidon's update, when no projections are used and the inverse Hessian approximate is updated directly, can be described by equations (3.43), (3.48), and

$$\theta = b(c - b) / (ac - b^2) \quad \text{when } b \leq 2ac/(a+c) \quad (3.74a)$$

$$\theta = b / (b - a) \quad \text{when} \quad b > 2ac / (a + c) \quad (3.74b)$$

Davidon's update can also be used in accordance with the scaling of the initial guess for the inverse Hessian to make the update invariant under scaling of the objective function. The first method of initial scaling discussed in 3.3.5.2, has produced much improvement on the scheme.<sup>[36]</sup>

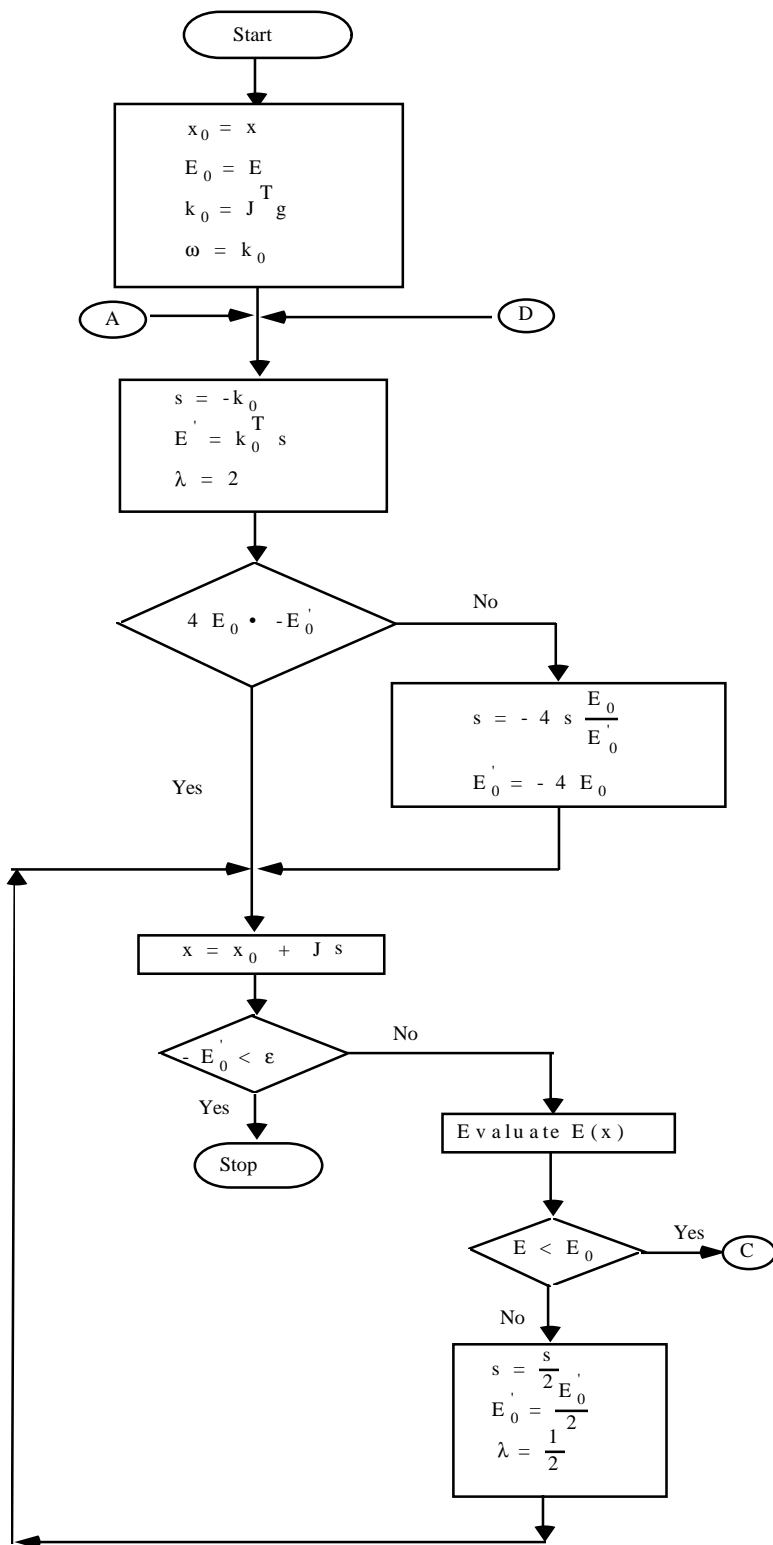


figure 3.1a (Flowchart of Davidon's Quasi-Newton Method)

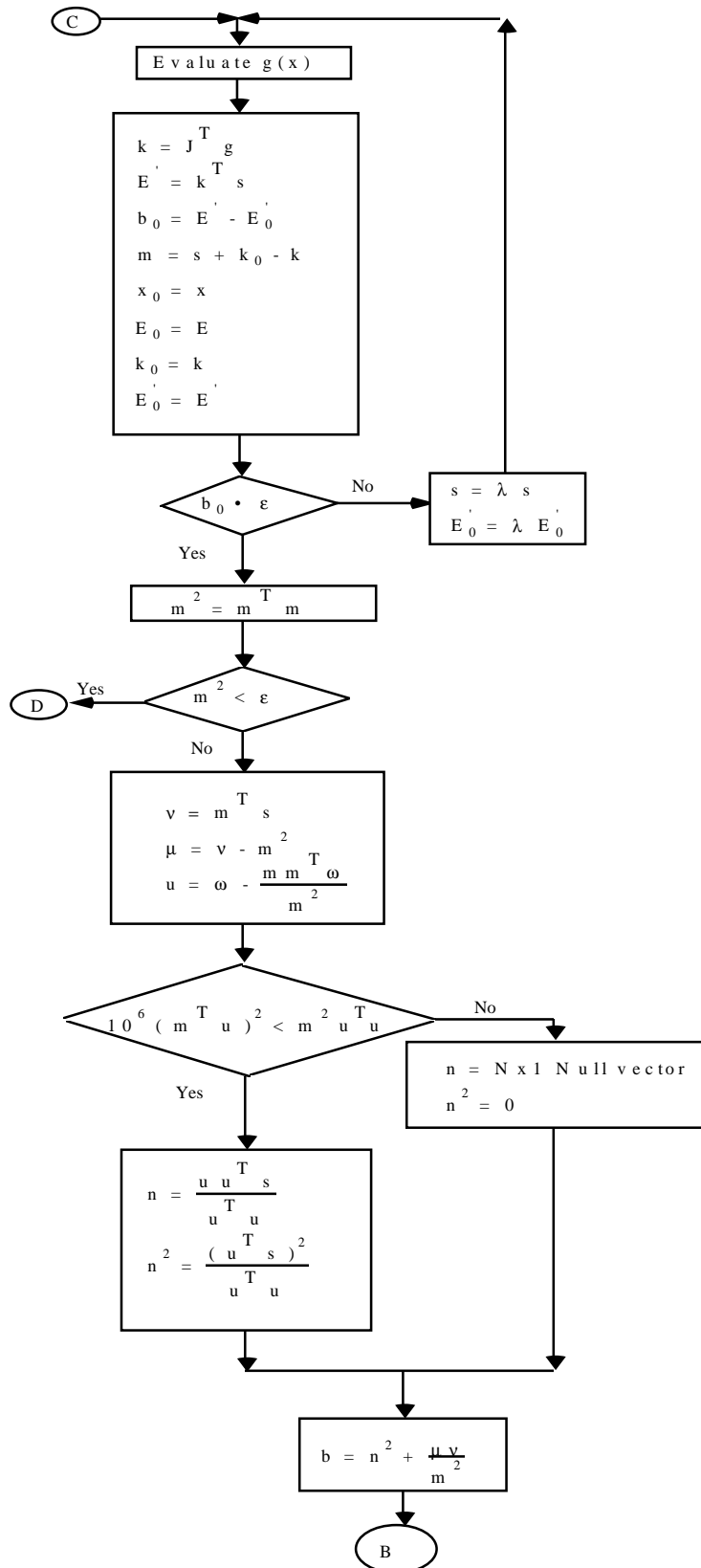


figure 3.1b (Flowchart of Davidon's Quasi-Newton Method)

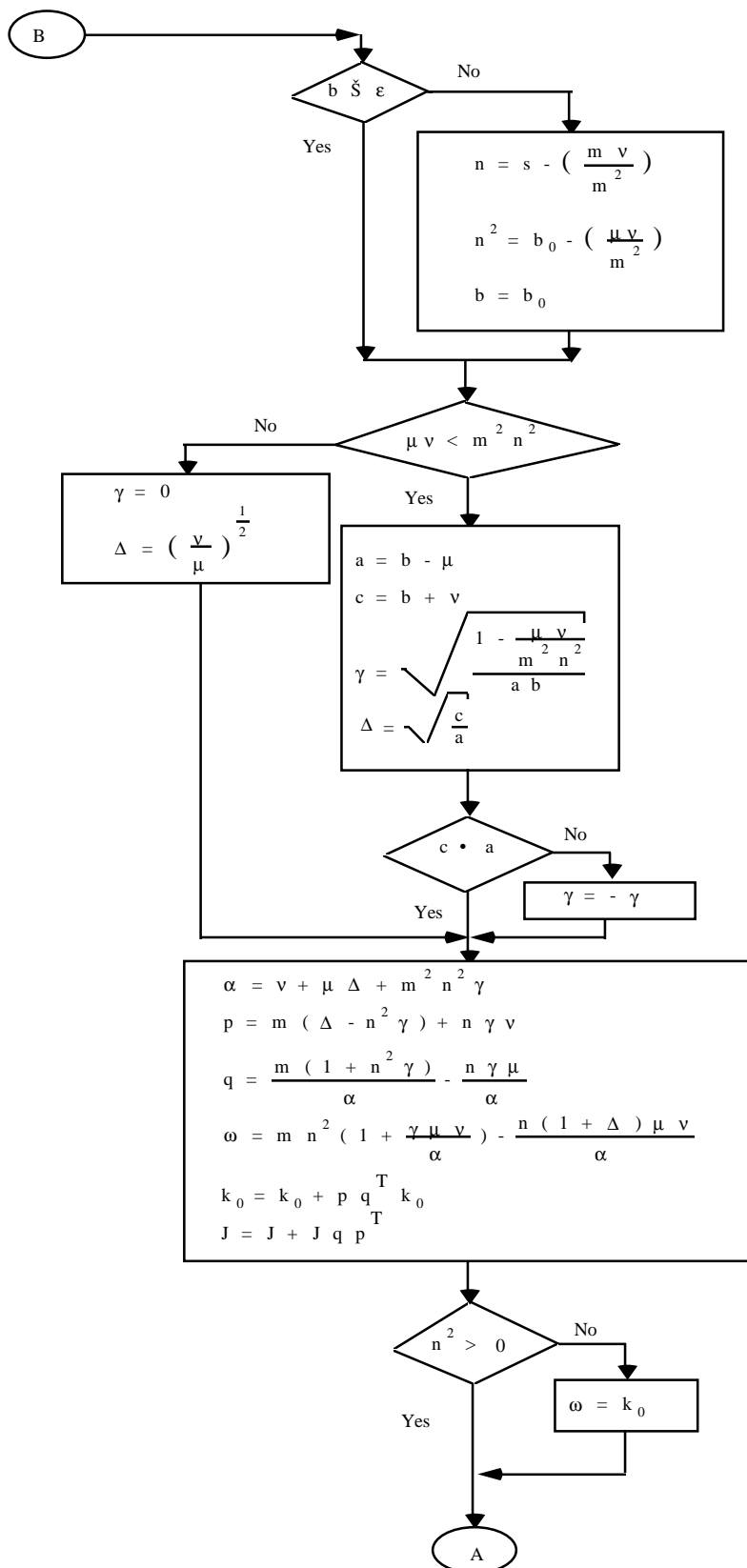


figure 3.1c (Flowchart of Davidon's Quasi-Newton Method)

### 3.4. Conjugate Gradient Methods

Consider a quadratic function  $E(x)$ ,  $x \in \mathbb{R}^N$ ,

$$E(x) = a + b^T x + \frac{1}{2} x^T G x \quad (3.75)$$

where  $G$  is positive definite and has full rank  $N$ .

Take a normalized starting direction  $s_0 \in \mathbb{R}^N$  such that  $\|s_0\|_E = 1$  and an initial vector  $x_0$ . Find the step size  $\lambda_0^*$  such that,

$$x_1 = x_0 + \lambda_0^* s_0 \quad (3.76)$$

minimizes  $E$  along the direction  $s_0$ .

Then,

$$E(x_1) = a + b^T (x_0 + \lambda_0^* s_0) + \frac{1}{2} (x_0 + \lambda_0^* s_0)^T G (x_0 + \lambda_0^* s_0) \quad (3.77)$$

To find the minimum of  $E$  in the  $s_0$  direction,

$$\frac{\bullet E}{\bullet \lambda_0^*} = 0 \quad (3.78)$$

or,

$$\nabla^T E(x_0) s_0 + s_0^T G \lambda_0^* s_0 = 0 \quad (3.79)$$

Solving for  $\lambda_0^*$  from (3.79),

$$\lambda_0^* = - \frac{s_0^T \nabla E(x_0)}{s_0^T G s_0} \quad (3.80)$$

If a unidirectional minimization is done for  $N$  times in  $N$  directions  $s_k$ ,  $k \in \{0, 1, \dots, N-1\}$  which are mutually conjugate about  $G$ , the Hessian (matrix of second partial derivatives) of  $E$ , then,

$$\begin{aligned}
x_N &= x_0 + \sum_{k=0}^{N-1} \lambda_k^* s_k \\
&= x_0 - \sum_{k=0}^{N-1} \frac{s_k^T \nabla E(x_k)}{s_k^T G s_k} s_k
\end{aligned} \tag{3.81}$$

However,

$$\nabla E(x_k) = G x_k + b \tag{3.82}$$

For simplification, define,

$$g_k = \nabla_x E(x_k) \tag{3.83}$$

then,

$$\begin{aligned}
s_k^T g_k &= s_k^T (G x_k + b) \\
&= s_k^T (G (x_0 + \sum_{i=0}^{k-1} \lambda_i^* s_i) + b) \\
&= s_k^T (G x_0 + b) \quad \text{since } s_k^T G s_i = 0 \text{ for all } i < k \text{ due to conjugacy}
\end{aligned} \tag{3.84}$$

The expression for  $x_N$  can be written as,

$$x_N = x_0 - \sum_{k=0}^{N-1} \frac{s_k^T (G x_0 + b) s_k}{s_k^T G s_k} \tag{3.85}$$

By theorem A.1,

$$x_0 = \sum_{k=0}^{N-1} \frac{s_k^T G x_0 s_k}{s_k^T G s_k} \tag{3.86}$$

Substituting (3.86) into (3.85),

$$\begin{aligned}
x_N &= - \sum_{k=0}^{N-1} \frac{s_k^T b s_k}{s_k^T G s_k} \\
&= - \sum_{k=0}^{N-1} \frac{s_k s_k^T b}{s_k^T G s_k}
\end{aligned} \tag{3.87}$$

However, from theorem A.2,

$$G^{-1} = H = \sum_{k=0}^{N-1} \frac{s_k s_k^T}{s_k^T G s_k}$$

which after substitution in (3.87) will give,

$$x_N = - G^{-1} b \tag{3.88}$$

Equation (3.88) is equivalent to the Newton step which minimizes a quadratic function. Therefore,  $x_N$  is the minimizer of  $E$ . This shows that the minimum of a quadratic function can be reached in at most  $N$  linear minimizations if these minimizations are done along a full set of directions mutually conjugate about the Hessian matrix of the objective function. In general, the method of conjugate directions provides quadratic convergence. A few different conjugate direction methods are discussed in the following sections.

### 3.4.1. Fletcher-Reeves Conjugate Gradient Method

The Fletcher-Reeves<sup>[46]</sup> conjugate gradient method generates a set of search directions  $s_i$ ,  $i \in \{ 0, 1, \dots, N-1 \}$  such that  $s_k$  is a linear combination of  $g_k$  and all  $s_j$ ,  $j \in \{ 0, 1, \dots, k-1 \}$  with the combination weights picked such that  $s_k$  is conjugate about the Hessian matrix  $G$  of a quadratic objective function to all  $s_j$ ,  $j \in \{ 0, 1, \dots, k-1 \}$ . These weights (coefficients) are chosen such that only the two most recent gradients

are needed for their evaluation at each iteration. This solution was reached by influences from Hestenes and Stiefel<sup>[47]</sup> and Beckman<sup>[48]</sup> as noted in [46].

Reference [46] provides a method for finding conjugate directions based on the two most recent gradients. Here a derivation is given for the calculation of these directions, leading to the method of conjugate gradients due to Fletcher and Reeves<sup>[46]</sup>.

Write the direction at iteration  $k+1$ ,  $s_{k+1}$ , such that  $s_{k+1}$  is a linear combination of the gradient  $g_{k+1}$  and direction  $s_k$ ,

$$s_{k+1} = -g_{k+1} + \omega_k s_k \quad (3.89)$$

where  $\omega_k$  is a weight to be chosen. Doing this for all  $k$ ,  $s_{k+1}$  becomes a linear combination of  $g_{k+1}$  and  $s_i$ ,  $i \in \{ 0, 1, \dots, k \}$ .

In (3.89)  $\omega_k$  should be chosen such that  $s_{k+1}$  is conjugate to  $s_k$  about the Hessian matrix  $G$  of the objective function  $E(x)$ , or,

$$s_k^T G s_{k+1} = 0 \quad (3.90)$$

From (3.2),

$$s_k = \frac{\nabla^2 E(x_k)^T}{\lambda_k^*} \quad (3.91)$$

and from (3.91), (3.18) and the definition of  $H$  ( i.e.  $H = G^{-1}$  ),

$$s_k = \frac{\nabla^2 E(x_k)^T G^{-1}}{\lambda_k^*} \quad (3.92)$$

Substituting (3.92) and (3.89) into (3.90),

$$\frac{\mathbf{g}_k^T}{\lambda_k^*} \mathbf{G}^{-1} \mathbf{G} (-\mathbf{g}_{k+1} + \omega_k \mathbf{s}_k) = 0 \quad (3.93)$$

or,

$$\frac{\mathbf{g}_k^T}{\lambda_k^*} (-\mathbf{g}_{k+1} + \omega_k \mathbf{s}_k) = 0 \quad (3.94)$$

and for any arbitrary  $\lambda_k^*$ ,

$$\mathbf{g}_k^T (-\mathbf{g}_{k+1} + \omega_k \mathbf{s}_k) = 0 \quad (3.95)$$

Write out all the terms in (3.95) using the definition of  $\mathbf{A}\mathbf{g}_k$ ,

$$-\mathbf{g}_{k+1}^T \mathbf{g}_{k+1} + \mathbf{g}_k^T \mathbf{g}_{k+1} + \omega_k \mathbf{g}_{k+1}^T \mathbf{s}_k - \omega_k \mathbf{g}_k^T \mathbf{s}_k = 0 \quad (3.96)$$

The  $\lambda_k^*$ 's are chosen such that they minimize  $E(x)$  in direction  $\mathbf{s}_k$ . Writing (3.79)

using (3.82) and (3.83),

$$(\mathbf{G} \mathbf{x}_k + \mathbf{b})^T \mathbf{s}_k + \mathbf{s}_k^T \mathbf{G} \lambda_k \mathbf{s}_k = 0 \quad (3.97)$$

Using (3.2) in (3.97) and factoring out  $\mathbf{s}_k^T$ ,

$$\mathbf{s}_k^T (\mathbf{G} \mathbf{x}_k + \mathbf{b} + \mathbf{G}^2 \mathbf{x}_k) = 0 \quad (3.98)$$

or,

$$\begin{aligned} \mathbf{s}_k^T (\mathbf{b} + \mathbf{G} (\mathbf{x}_k + \mathbf{G} \mathbf{x}_k)) &= \mathbf{s}_k^T (\mathbf{b} + \mathbf{G} \mathbf{x}_{k+1}) \\ &= 0 \end{aligned} \quad (3.99)$$

However, from (3.82),

$$\mathbf{b} + \mathbf{G} \mathbf{x}_{k+1} = \mathbf{g}_{k+1} \quad (3.100)$$

Substituting (3.100) into (3.99),

$$\mathbf{s}_k^T \mathbf{g}_{k+1} = 0 \quad (3.101)$$

Writing (3.82) for any iteration  $l$ ,

$$\mathbf{g}_l = \mathbf{b} + \mathbf{G} \mathbf{x}_l \quad (3.102)$$

The transition from  $\mathbf{x}_k$  to  $\mathbf{x}_l$  using the argument in the previous section is given by,

$$\mathbf{x}_l = \mathbf{x}_k + \sum_{j=k}^{l-1} \lambda_j^* \mathbf{s}_j \quad (3.103)$$

Substituting (3.103) into (3.102),

$$\mathbf{g}_l = \mathbf{b} + \mathbf{G} \left( \mathbf{x}_k + \sum_{j=k}^{l-1} \lambda_j^* \mathbf{s}_j \right) \quad (3.104)$$

or,

$$\mathbf{g}_l = \mathbf{g}_k + \sum_{j=k}^{l-1} \lambda_j^* \mathbf{s}_j \quad (3.105)$$

Multiply (3.105) from the left by  $\mathbf{s}_{k-1}^T$ ,

$$\mathbf{s}_{k-1}^T \mathbf{g}_l = \mathbf{s}_{k-1}^T \mathbf{g}_k + \sum_{j=k}^{l-1} \lambda_j^* \mathbf{s}_{k-1}^T \mathbf{G} \mathbf{s}_j \quad (3.106)$$

For conjugate directions and using (3.101) in (3.106),

$$\mathbf{s}_k^T \mathbf{g}_l = 0 \quad 0 \leq k < l-1 \quad (3.107)$$

and combining (3.101) and (3.107),

$$\mathbf{s}_k^T \mathbf{g}_l = 0 \quad 0 \leq k \leq l-1 \quad (3.108)$$

Substituting for  $\mathbf{s}_k$  in (3.101) using (3.89),

$$\mathbf{s}_k^T \mathbf{g}_{k+1} = \left( -\mathbf{g}_k + \omega_{k-1} \mathbf{s}_{k-1} \right)^T \mathbf{g}_{k+1} \quad (3.109)$$

or,

$$\mathbf{s}_k^T \mathbf{g}_{k+1} = -\mathbf{g}_k^T \mathbf{g}_{k+1} + \omega_{k-1} \mathbf{s}_{k-1}^T \mathbf{g}_{k+1} \quad (3.110)$$

Using (3.108) in (3.110),

$$\mathbf{g}_k^T \mathbf{g}_{k+1} = 0 \quad (3.111)$$

Substituting for  $\mathbf{s}_k$  from (3.89) in (3.96),

$$\begin{aligned}
& -\mathbf{g}_{k+1}^T \mathbf{g}_{k+1} + \mathbf{g}_k^T \mathbf{g}_{k+1} - \omega_k \mathbf{g}_{k+1}^T \mathbf{g}_k + \omega_k \omega_{k-1} \mathbf{g}_{k+1}^T \mathbf{s}_{k-1} \\
& + \omega_k \mathbf{g}_k^T \mathbf{g}_k - \omega_k \omega_{k-1} \mathbf{g}_k^T \mathbf{s}_{k-1} = 0
\end{aligned} \tag{3.112}$$

Using (3.108) and (3.111) in (3.112),

$$-\mathbf{g}_{k+1}^T \mathbf{g}_{k+1} + \omega_k \mathbf{g}_k^T \mathbf{g}_k = 0 \tag{3.113}$$

Solving for  $\omega_k$  from (3.113),

$$\omega_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \tag{3.114}$$

Substituting for  $\omega_k$  from (3.114) into (3.89),

$$\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \mathbf{s}_k \tag{3.115}$$

Using (3.115), the Fletcher-Reeves conjugate gradient method could be summarized by the following steps:

1. Set  $k = 0$  and  $\mathbf{s}_0 = -\mathbf{g}_0$
2. At the  $k^{\text{th}}$  iteration, find  $\lambda_k^*$  such that  $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k^* \mathbf{s}_k$  minimizes  $E$  in direction  $\mathbf{s}_k$

$$3. \mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \mathbf{s}_k$$

4. If  $\|\mathbf{s}_k\|_E < \varepsilon$  then terminate minimization
5. If  $k = N$  then goto step 1, Else goto Step 2

### 3.4.2. Partan's Iterative Conjugate Gradient Method

Partan's minimization method<sup>[49]</sup> is another popular conjugate gradient minimization technique which has been used in developing many commercial

minimization software packages. The Iterative Partan method is described by the following procedure.

1. Set  $s_0 = -g_0$  at initial point  $x_0$
2. Find  $\lambda^*_0$  such that  $x_1 = x_0 + \lambda^*_0 s_0$  minimizes E along the direction  $s_0$
3. Set  $s_1 = -g_1$
4. Find  $\lambda^*_1$  such that  $x_2 = x_1 + \lambda^*_1 s_1$  minimizes E along the direction  $s_1$
5. Set  $s_2 = \frac{x_2 - x_0}{\|x_2 - x_0\|_E}$
6. Find  $\lambda^*_2$  such that  $x_3 = x_2 + \lambda^*_2 s_2$  minimizes E along the direction  $s_2$
7. If the termination criteria for the minimization are not met then set  $x_0 = x_3$  and goto step 1, Else, terminate minimization.

In practice the iterative method is not as effective as the following variation which is called the Continuous Partan method.

### 3.4.3. Continuous Partan Minimization Method

The Continuous Partan method<sup>[50]</sup> has added a few extra steps to the iterative Partan method which enhance the performance of the method in practice. These steps are as follows,

1. Do steps 1 through 6 of the iterative Partan method
2. Set  $s_3 = -g_3$
3. Find  $\lambda^*_3$  such that  $x_4 = x_3 + \lambda^*_3 s_3$  minimizes E along the direction  $s_3$
4. Set  $s_4 = \frac{x_4 - x_1}{\|x_4 - x_1\|_E}$
5. Find  $\lambda^*_4$  such that  $x_5 = x_4 + \lambda^*_4 s_4$  minimizes E along the direction  $s_4$

6. If the termination criteria for the minimization are not met then set  $x_3 = x_5$  and goto step 2, Else, terminate minimization.

### 3.4.4. The Projected Newton Algorithm

Zoutendijk<sup>[51]</sup> presented a gradient projection method which is summarized by the following steps:

1. Let  $P_0 = I$  and start from the initial state  $x_0$ .
2. 
$$P_k = I - G_k [ G_k^T G_k ]^{-1} G_k^T$$

$$= P_{k-1} - P_{k-1}^2 g_k [ ^2 g_k^T P_{k-1} ^2 g_k ]^{-1} ^2 g_k^T P_{k-1}$$
3. If  $P_k g_k = 0$  let  $s_k = - P_k g_k$
4. Minimize  $E(x)$  in direction  $s_k$
5. If  $P_k g_k = 0$  and  $g_k = 0$  then terminate
6. If (  $P_k g_k = 0$  and  $g_k \neq 0$  ) or  $k=N$ , then  $k=0$ ,  $x_0=x_N$  and goto step 1
7.  $k=k+1$ , goto step 2

For a derivation of the projection matrix of step 2 see reference [52]. The above procedure is equivalent to a Quasi-Newton approach with the initial inverse-Hessian approximation picked as  $H_0=I$  and then updated by equation (3.116).

$$H_{k+1} = H_k - \frac{^2 g_k^T H_k^T H_k ^2 g_k}{^2 g_k^T H_k ^2 g_k} \quad (3.116)$$

This update is equivalent to the method of Projected Newton-Raphson given by Pearson.<sup>[27]</sup>

## **Chapter 4**

### **Minimization Techniques Requiring No Gradient Evaluations**

All the minimization methods listed in the previous chapters required evaluation of the first gradient vector. In neural network learning, this requirement brings about the need for knowing the exact structure of the network and all of the connections. For large networks, it is very difficult to analytically evaluate these gradients. To evaluate the gradient of the objective function with respect to weights and activation function parameters in the lower (close to input) layers, chain rule requires many derivative evaluations. This difficulty becomes more serious when the number of layers grows. On the other hand, objective function evaluations can easily be performed by the neural network while gradient evaluations cannot be by the network alone, in general. This is the main motivation behind the use of optimization methods which require no gradient evaluations. The objective is to minimize the need for extra hardware/software for gradient evaluations and to have the neural network circuit perform most of the computations. A special advantage of gradient-free methods is that they do not require regularity and continuity of the objective function.

#### **4.1. Search Methods**

The optimization methods which were described in the previous chapters require analytic evaluation of the gradient of the objective function. In a family of optimization methods called search methods, the directions of minimization are evaluated solely based on the objective function values. In general, gradient based minimization methods converge faster than gradient-free methods. However, when many variables are involved, as in the case of neural-network learning problems, the analytical evaluation of the gradient becomes complicated. A popular gradient-free minimization scheme is discussed by the following.

#### **Hooke-Jeeves and Wood Direct Search Method**

The direct search method as implemented by Hooke-Jeeves<sup>[53]</sup> and Wood<sup>[54]</sup> is based on exploratory searches in the directions of independent variables one at a time, while keeping the rest of the variables constant. These methods are known to work poorly when there are terms in the objective function involving the product of a few design variables.<sup>[8]</sup> This makes the direct search method a poor choice for application to the neural-network learning problem.

## **4.2. Gradient-Free Conjugate Direction Methods**

There are a few methods which use only objective function evaluations to predict a search direction which is conjugate to one or more directions about the Hessian of the quadratic approximation to the objective function. Among these methods are Rosenbrock's method, the Davies-Swann-Campey method ( a modified version of Rosenbrock's method), Smith's method and Powell's first and second methods. These gradient-free minimization techniques are discussed in more detail in the following.

### **4.2.1. Rosenbrock's Method**

Rosenbrock's method<sup>[55]</sup> starts with a full set of orthonormal directions  $s_i$ ,  $i \in \{ 0, 1, \dots, N-1 \}$  which could be the directions corresponding to the  $N$  independent variables of the objective function. It searches along these directions ( i.e. starts out like the Hooke-Jeeves direct search method) for a sufficient reduction in the objective function. After completing  $N$  searches, it takes the final value of the state  $x$  and subtracts from it the initial value of the state. Let us denote the value of the state vector after these  $N$  searches by  $x_1$  and its initial value by  $x_0$ . Following the previous nomenclature,

$${}^2 x_0 = x_1 - x_0 \quad (4.1)$$

$\bar{A}x_0$  gives the new direction  $s_0$  upon normalization and then a Gram-Schmidt orthogonalization process as described in Appendix A is employed to obtain the rest of the directions  $s_i, i \in \{ 1, 2, \dots, N-1 \}$  which are made orthonormal to  $s_0$ . The search through all the new  $N$  directions is repeated again and a new direction is found. These  $s_0$  directions at every iteration  $k$  tend to line up with the principal axes of the Hessian of the quadratic approximation to the objective function ( eigenvectors of  $G$  ). This makes Rosenbrock's method similar to conjugate direction methods in convergence properties when applied to the minimization of a quadratic objective function.

This method has a very serious problem with its applicability to practical problems such as the neural network learning problem. The search directions generated by the method could sometimes become zero. In that case, the scheme fails. Davies, Swann and Campey made a modification to Rosenbrock's method to reduce the chances of this type of failure.

#### **4.2.2. Davies-Swann-Campey Method**

Davies, Swann and Campey presented a variation of the Rosenbrock gradient-free minimization method which makes it more practical. This method is named after them and abbreviated to the DSC method<sup>[56]</sup>. In the DSC method, a gradient-free linear minimization method is used to find the minimum of the objective function along the  $N$  directions in contrast to Rosenbrock's method that makes a mere reduction in the objective function. Another modification done to Rosenbrock's method is a reordering of the directions which allows the retainment of nonzero

directions separate from the zero directions and minimization is done in those nonzero directions until termination occurs or all the directions become zero.

### 4.2.3. Powell's Methods

In most minimization methods, the state vector is updated in a direction ( $s_k$ ) dictated by the method and a linear search is used to find the optimum step size of the update in that direction ( $\lambda_k^*$ ). This update will be of the following form at every step of the minimization,

$$x_{k+1} = x_k + \lambda_k^* s_k \quad (4.2)$$

The main purpose of the minimization algorithm is to find a sequence of directions in which to perform the linear searches. If the objective function is quadratic, then the best set of directions are in general the set which are mutually conjugate about the Hessian matrix of that function. However, since a gradient-free minimization scheme is intended, the Hessian matrix and even the first gradients are not evaluated. This makes the task of finding a mutually conjugate set of directions very difficult. Powell in [57] states a theorem on conjugacy that helps him develop an algorithm which tends to line up two consecutive new directions of search with conjugate directions.

Powell<sup>[57]</sup> devised two methods, in evolution from Smith's method<sup>[58]</sup>, which minimize the objective function  $E(x)$  by successive linear searches in directions which are generated by the methods and tend to become conjugate about the Hessian of the quadratic approximation to the objective function  $E$ .

These methods are based on two theorems stated by Powell<sup>[57]</sup> in the following manner:

*Powell's Theorem 1:*

" If  $q_1, q_2, \dots, q_m, m^2 \leq n$ , are mutually conjugate directions, then the minimum of the quadratic  $f(x)$ , where  $x$  is a general point in the  $m$ -dimensional space containing  $x_0$  and the directions  $q_1, q_2, \dots, q_m$ , may be found by searching along each of the directions once only."

*Powell's Theorem 2:*

" If  $x_0$  is the minimum in a space containing the direction  $q$ , and  $x_1$  is also the minimum in such a space, then the direction  $(x_1 - x_0)$  is conjugate to  $q$ ."

Powell's paper provides the proofs to these theorems. Using these two theorems, Powell presented his first method given by the following steps.

*Powell's first method:*

Set  $s_k = e_k$ .

1. Find  $\lambda_k$  which minimize  $E(x_{k-1} + \lambda_k s_k)$

and  $x_k = x_{k-1} + \lambda_k s_k$

$k=1, 2, 3, \dots, N$

2. Replace  $s_k$  by  $s_{k+1}$  for  $k=1, 2, 3, \dots, N-1$

3. Replace  $s_N$  by  $x_N - x_0$

4. Find  $\lambda_N$  which minimizes  $E(x_0 + \lambda_N s_N)$

5. Set  $x_0 = x_0 + \lambda_N s_N$  and goto step 1

This procedure finds points  $x_1, \dots, x_N$  which minimize the quadratic approximate of the objective function in the  $s_1, \dots, s_N$  directions and generates the new direction,

$$s_{N+1} = \frac{x_N - x_0}{\|x_N - x_0\|} \quad (4.3)$$

This procedure makes up one iteration of Powell's first method. Through renumbering, the procedure is repeated for the new directions  $s_1, \dots, s_N$ . Powell claims that after  $N$  iterations, the minimum of the quadratic function is reached. However, this claim has been shown by Zangwill<sup>[59]</sup> to be false in general. Zangwill provides a counter example in [59] for which Powell's method will never converge to the minimum. This lack of convergence is due to a mistake in the statement of Powell's first theorem. Powell's methods could sometimes generate linearly dependent directions which will not span the entire space. To correct for this mistake, Powell states that sometimes it is not wise to accept any new direction provided by his method. Zangwill gives a correction for Powell's first theorem which would solve this problem. He states that in theorem 1, the directions  $q_1, q_2, \dots, q_m$ , must be such that they span the entire  $m$ -dimensional space. This gives the motivation behind his second minimization method. This method as simplified by Zangwill<sup>[59]</sup> is as follows.

*Powell's second method:*

Set  $s_k^1 = e_k$ ,  $x_0^1$  is picked,  $\epsilon: 0 < \epsilon^2 < 1$  is given as the accuracy for accepting a direction

$$\delta^1 = 1 \text{ and } r=1$$

1. Find  $\lambda_k^r$  which minimize  $E(x_{k-1}^r + \lambda_k^r s_k^r)$

$$\text{and } x_k^r = x_{k-1}^r + \lambda_k^r s_k^r$$

$$k=1, 2, 3, \dots, N$$

$$2. \text{ Define } \alpha^r = \| x_N^r - x_0^r \| \text{ and } s_{N+1}^r = ( x_N^r - x_0^r ) / \alpha^r$$

$$3. \text{ Find } \lambda_{N+1}^r \text{ which minimizes } E(x_N^r + \lambda_{N+1}^r s_{N+1}^r)$$

$$4. \text{ Set } x_0^{r+1} = x_{N+1}^r = x_N^r + \lambda_{N+1}^r s_{N+1}^r$$

$$5. \lambda_s^r = \max \{ \lambda_k^r \mid k=1,2,3, \dots, N \}$$

$$6. \text{ If } \lambda_s^r \delta^r / \alpha^r \geq \epsilon \text{ then}$$

$$s_k^{r+1} = s_k^r \text{ for } k \neq s$$

$$s_s^{r+1} = s_{N+1}^r$$

$$\delta^{r+1} = \lambda_s^r \delta^r / \alpha^r$$

$$7. \text{ If } \lambda_s^r \delta^r / \alpha^r < \epsilon \text{ then}$$

$$s_k^{r+1} = s_k^r \text{ for } k = 1, 2, 3, \dots, N$$

$$\delta^{r+1} = \delta^r$$

$$8. r = r+1, \text{ goto step 1}$$

In the above algorithm,  $\delta^r$  is the determinant,

$$\delta^r = \det ( [ s_1, s_2, \dots, s_N ] )$$

For  $r=1$ , since the directions  $s_k, k \in \{ 1, 2, \dots, N \}$  coincide with the columns of the identity matrix, the determinant  $\delta^1 = 1$ . As the method proceeds in iterations, the objective is to find a set of directions which would have the largest determinant, for if the determinant approaches zero, then the set of directions approach a linearly dependent set. When a new direction  $s_{N+1}^r$  is found through step 3, if it replaces any direction  $s_s^r$ , then the new determinant of the direction set will be given by  $\delta^{r+1} = \lambda_s^r \delta^r / \alpha^r$ . Since the size of this determinant is largest if the largest  $\lambda_s^r$  is used, then the direction that should be replaced with the new direction should be the one that corresponds to the largest linear step  $\lambda_s^r$ . However, if this new determinant is seen to

be smaller than some computational tolerance  $\epsilon$ , then replacing any direction with this new direction will make the new set of directions linearly dependent. In this case, the new direction is rejected and minimization takes place again with the previous set of directions.

In addition, in [60], Fletcher gives a modified version of Smith's method which makes it a rival of Powell's methods.

# **Chapter 5**

## **Simulations and Results**

Computer simulations of learning have been done using Quasi-Newton, Conjugate Gradients, and gradient-free methods. In all the simulations, inexact line searches have been used unless indicated otherwise. Inexact line searches provide methods with less function evaluations if the minimization methods are robust enough. In the tables of this chapter, E is the final value of the objective function, P is the number of presentations of all input patterns which is equivalent to the number of evaluations of the objective function, and G is the number of new directions generated by the scheme which for Quasi-Newton methods is equivalent to the number of updates made to the inverse Hessian estimate and the number of gradient evaluations. Finally, F is the number of floating point operations. In these tables, for the cases in which local minima were reached, only E is given. In addition, a list of the abbreviations used in this chapter's tables is given in Appendix B.

### **5.1. Quasi-Newton Methods**

Computer simulations of the weight and activation function parameter adjustments have been done using all the described minimization schemes as applied to the classical XOR problem and an encoder. The input and desired output patterns for these two problems are given in tables 2.1 and 5.1 respectively. The architectures used for the XOR and encoder problems are also given in figures 2.1 and 5.3. Two important measures of the performance of the minimization algorithms are the number of times the input patterns have to be presented and the total number of floating point operations necessary to achieve convergence. Some Quasi-Newton methods such as BFGS, SSVM, and Davidon's updates work well with inexact line searches and there is no point in doing a lot of function evaluations within a line search subproblem. The performance of some methods such as the DFP method is however much better when an exact line search is done.

### 5.1.1. Classical Quasi-Newton Methods

The results of the computer simulation of the classical Quasi-Newton methods of the Broyden family with no scaling are given in table 5.2. Termination of the minimization took place when  $E < 1e-5$  or when  $|E_{k+1} - E_k| < 1e-6$ . In the tables, XOR I and XOR II correspond to the XOR problem when the first and second set of initial conditions were used respectively (see figures 5.1 and 5.2).

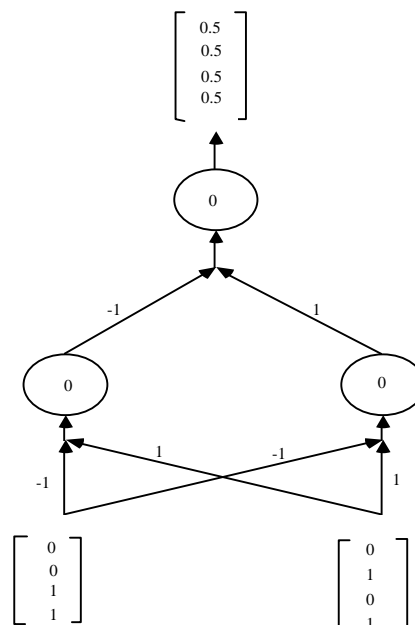


figure 5.1 (Initial State for the XOR I Problem)

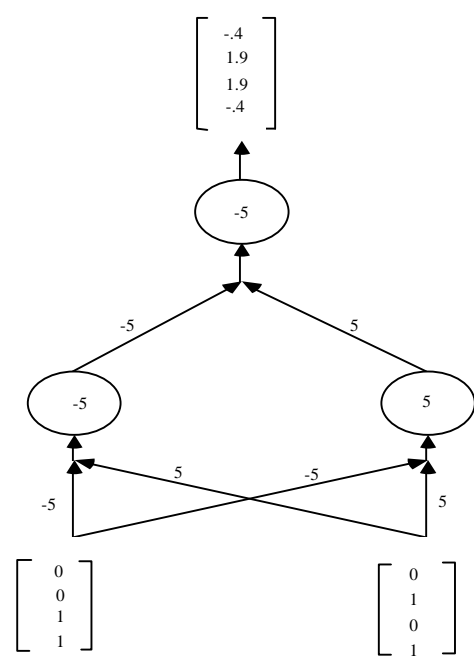


figure 5.2 (Initial State for the XOR II Problem)

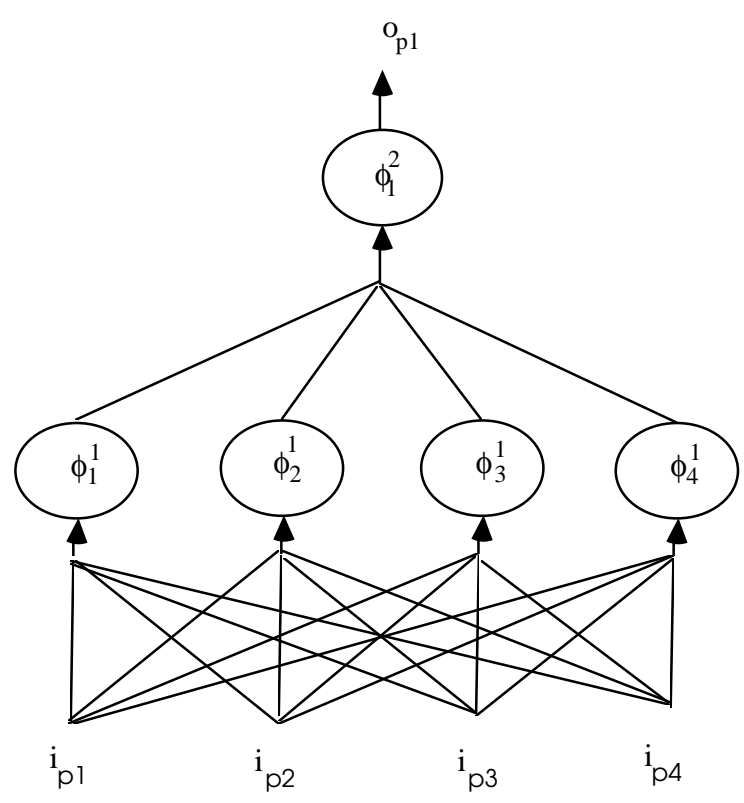


figure 5.3 (Neural Network Simulating the Encoder Logic of table 5.1)

The first set of initial conditions in the XOR problem were possibly close to a ridge in the objective function. This introduces a handicap for the Steepest Descent technique. However, most Quasi-Newton methods tested show a good performance in finding a local minimum (two orders of magnitude better than the Steepest Descent method.) Pearson II and BFGS find the global minimum in very few number of iterations. For the second set of initial conditions used in the XOR problem, Steepest Descent does an order of magnitude better than the first set. However, the Quasi-Newton methods still have a much better performance. Again, Pearson II and BFGS provided a near global minimum. However, the DFP, Broyden and Projected Newton-Raphson (Zoutendijk) methods also converged to the near global minimum much in the same speed as BFGS and Pearson II for XOR II. A performance similar to that in XOR I is shown in the encoder problem with Pearson II and BFGS in the leading positions. Also, it is seen from the final values of the objective function that Quasi-Newton methods converged to points much closer to the global minimum ( $E = 0$ .)

I1	I2	I3	I4	Desired Output
1	1	1	1	0
1	0	1	1	1
1	0	1	0	0
0	0	1	0	1
0	0	0	0	0

*table 5.1 (Encoder Logic)*

		BR.	DFP	BFGS	PR II	PR III	PNR	GR I	GR II	SD
XOR I	E	0.956	0.666	4e-16	6e-7	0.989	0.955	0.989	0.989	2.8e-5
	P	----	----	15	36	----	----	----	----	8553
	G	----	----	6	11	----	----	----	----	8465
	F	----	----	2.5e4	2.2e4	----	----	----	----	8e6
XOR II	E	8e-6	3.5e-6	9.6e-6	2e-4	0.952	0.014	0.226	0.226	7e-6
	P	84	58	53	108	----	31	----	----	163
	G	20	17	15	24	----	8	----	----	52
	F	6e4	7.7e4	7.3e4	6.5e4	----	6.5e4	----	----	10e5
Encoder	E	0.956	0.599	1e-17	1.5e-7	1.092	0.500	1.098	1.163	8e-5
	P	----	----	36	54	----	----	----	----	1352
	G	----	----	11	16	----	----	----	----	1194
	F	----	----	5.1e5	1.4e5	----	----	----	----	3.8e6

table 5.2 (Results of Classical Quasi-Newton Methods)

		BFGS	PRII	O(1,.1)	O(1,.2)	O(1,.3)	O(1,.5)	O(.5,.5)
XOR I	E	4e-16	6e-7	9e-9	0	0	----	8e-7
	P	15	36	12	8	11	----	13
	G	6	11	4	3	4	----	5
Encoder	E	1e-17	2e-7	7e-6	8e-6	3.7e-6	2e-11	1e-8
	P	36	54	30	42	33	20	28
	G	11	16	9	13	10	6	9

table 5.3 (Results of Oren's SSVM method)\*

\*  $O(\phi, \theta)$ 's denote the two parameters of equation 3.46.

### 5.1.2. Self-Scaling Quasi-Newton Methods

Table 5.3 presents the results of a set of simulations for the purpose of comparison between classical and SSVM Quasi-Newton techniques as applied to neural network learning. Two important measures of the performance of the algorithms in this

comparison are the number of times the input patterns had to be presented (P) and the number of updates which were made to the initial guess for the inverse Hessian matrix, to achieve convergence (G).

In picking the parameters  $\mu_k$  and  $\theta_k$ , a trial and error method was used. The switching method of [34] did not perform well compared to pre-chosen constant parameters. This result agrees with experiments done by Oren<sup>[34]</sup>. (e.g. the number of presentations and inverse-Hessian updates for the encoder problem were 77 and 30 respectively, when this switching was done.) Also, the switching methods of [35] which were demonstrated to have done well were not used here due to their impracticality for neural networks. In these switching methods, an update of the Hessian matrix should be kept as well as the inverse Hessian. This would require a lot of memory when a large network is involved.

Computer simulations of the (Quasi-Newton with initial scaling) learning have been done using 3.52 and 3.55 for initial scaling of the approximate inverse-Hessian matrix with the BFGS algorithm as applied to the classical XOR problem and an encoder. For the XOR problem, two different set of initial states were chosen. Results of the computer simulations are given in table 5.4. Results are compared to those obtained for the BFGS algorithm. BFGSa and BFGSb in table 5.4 correspond

to BFGS with initial scalings (3.52) and (3.55) respectively. Termination of the minimization took place when  $E < 1e-4$  or when  $|E_{k+1} - E_k| < 1e-5$ .

		BFGS	BFGSa	BFGSb
XOR I	E	4e-16	7.27e-5	0
	P	15	14	9
	G	6	6	4
	F	2.5e4	2.5e4	1.7e4
XOR II	E	9.6e-6	8.83e-5	3.7e-5
	P	53	37	49
	G	15	10	13
	F	7.3e4	5.0e4	6.49e4
Encoder	E	1e-17	1.65e-8	4.82e-6
	P	36	36	35
	G	11	12	12
	F	5.1e5	5.57e5	5.58e5

table 5.4 (Results of BFGS Method with Initial Scaling)

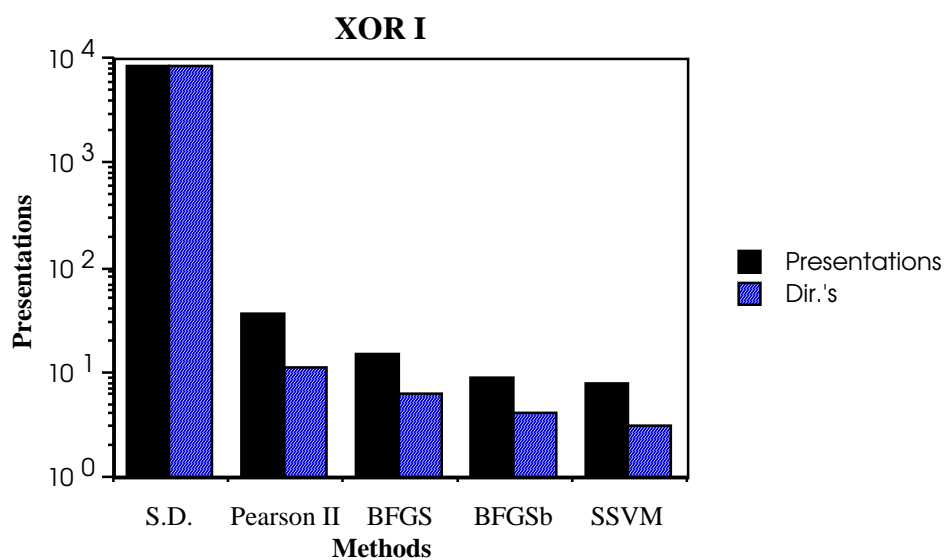


figure 5.4 (Comparison of the Performance of Quasi-Newton Methods for the XOR I Problem)

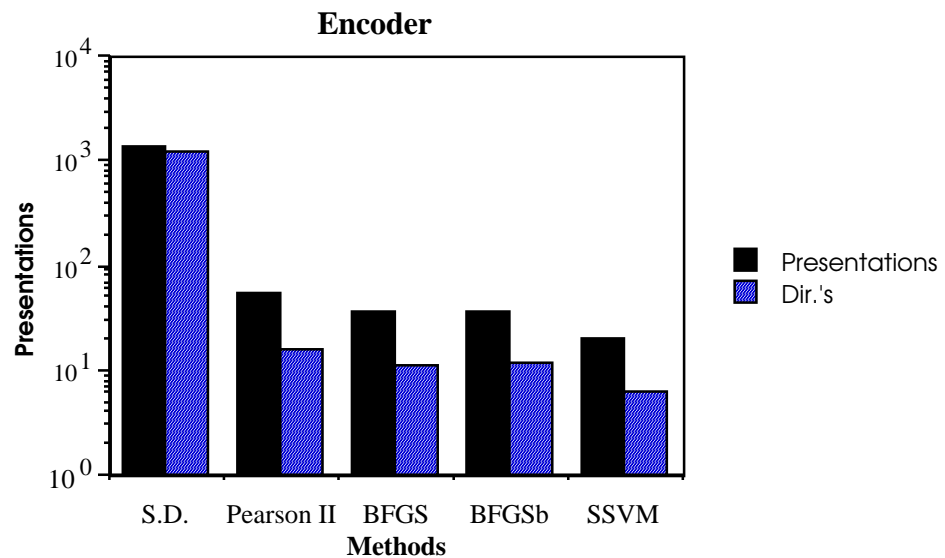


figure 5.5 (Comparison of the Performance of Quasi-Newton Methods for the Encode Problem)

## 5.2. Conjugate Gradient Techniques

Table 5.5 presents the results of the Fletcher-Reeves, Iterative Partan, and Continuous Partan conjugate gradient methods applied to the learning in the XOR I and XOR II problems. As shown by the table, the Fletcher-Reeves method is the most efficient technique among these conjugate gradient methods. These results have been obtained in accordance with exact and inexact line searches. Results show that these methods work much better when an exact line search is employed. When applied to the encoder problem, all the methods failed to reach a near global minimum and converged to a number of different local minima. Therefore, these results were not included in table 5.5. For simplicity of comparisons, in figure 5.6, the number of presentations necessary for convergence of learning have been graphed for the Fletcher-Reeves (F-R) and Continuous Partan methods as applied to the XOR I learning problem.

		Fletcher-Reeves		Iterative Partan		Continuous Partan	
		Ex/LS	Inex/LS	Ex/LS	Inex/LS	Ex/LS	Inex/LS
XORI	E	8.01e-6	0.7885	0.963	0.963	8.9e-6	0.985
	P	98	-----	-----	-----	154	-----
	G	18	-----	-----	-----	23	-----
	F	4e5	-----	-----	-----	5.6e4	-----
XORII	E	8.33e-6	9.97e-6	0.985	0.096	0.096	0.0167
	P	64	375	-----	-----	-----	-----
	G	19	369	-----	-----	-----	-----
	F	3.6e4	3e5	-----	-----	-----	-----

table 5.5 (Results of Conjugate Gradient Methods)

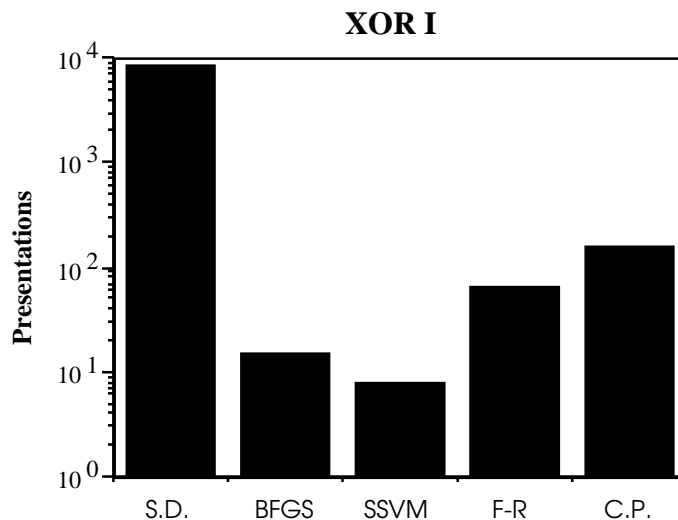


figure 5.6 (Comparison of the Performance of Conjugate Gradient Methods with Quasi-Newton and S.D. Methods)

### 5.3. Gradient-Free Techniques

Computer simulations of the learning have been done using Powell's first and second algorithms for minimizing the objective function  $E$  of the XOR problem. Results of learning based on Powell's methods are compared to those of table 5.2 for the steepest descent (S.D.) and the BFGS algorithms. These results are given in table 5.6. Termination of the minimization took place when  $E < 1e-4$  or  $|E_{k+1} - E_k| < 1e-5$ .

	S.D.	BFGS	Pwl1	Pwl2
E	7e-6	9.6e-6	6.1e-5	1e-4
P	163	53	411	337
G	52	15	2	1
F	1e5	7.3e4	7.7e4	6.2e4

table 5.6 (Results of Powell's Gradient-Free Methods)

The table shows that the second method of Powell had the best performance and it required less floating point operation than any other method , for convergence. However, in general, both methods of Powell performed satisfactorily, specially considering the fact that only function evaluations were required. Figures 5.7 and 5.8 show a bar chart comparison among the methods of table 5.6 for the number of new directions and number of floating point operations (FLOP's) required for convergence. Powell's second method required the least of both measures.

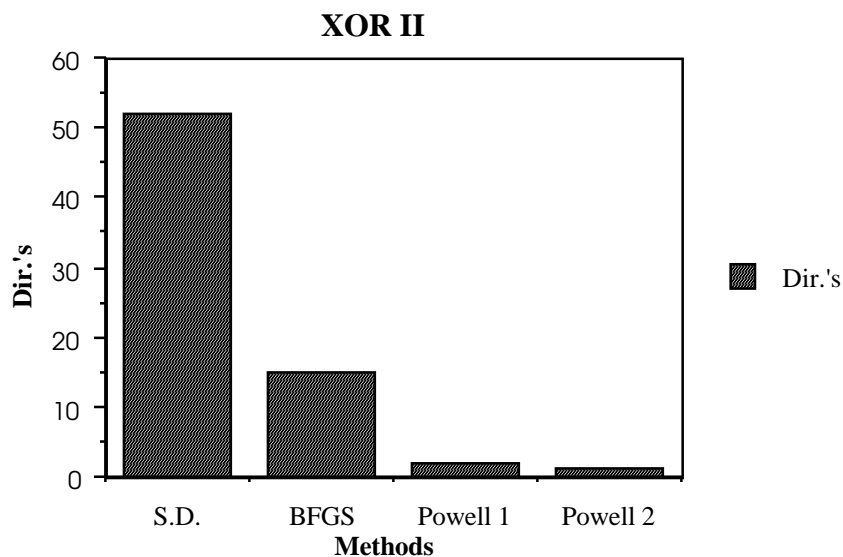
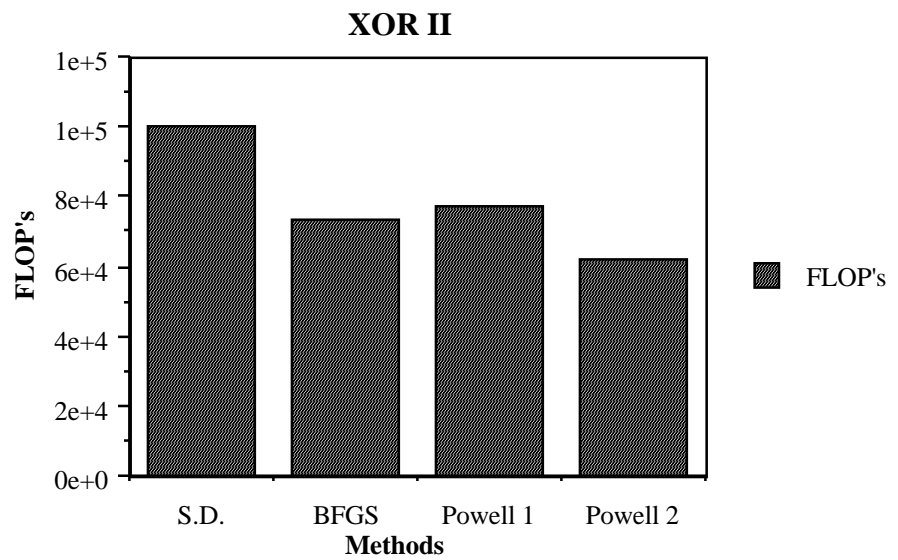


figure 5.7 (Comparison of Number of New Directions Found for the XOR II Problem Using Powell's Methods)



*figure 5.8 (Comparison of Number of FLOP's Required for the XOR II Problem Using Powell's Methods)*

## **Chapter 6**

### **Conclusion**

Looking at the state-of-the-art in neural network learning, a need was noted for faster learning algorithms. For general non-quadratic objective functions, steepest descent techniques are known to perform well, away from the minima and Newton's method works well, in the vicinity of the minima. Therefore, for a fast learning algorithm, one should take advantage of the best of the two methods. In Newton's method, evaluation and inversion of the Hessian matrix pose a difficult and time consuming problem.

### **6.1. Quasi-Newton Methods**

Classical Quasi-Newton methods start off with an estimate of the inverse Hessian matrix equal to the identity matrix. This matrix provides the Steepest descent direction. As patterns are presented, the Quasi-Newton methods provide an update to the inverse Hessian matrix. This will in the limit provide an optimal direction of descent based on the momentum of the inverse Hessian matrix.

Results show an increase in the rate of convergence of about two to three orders of magnitude from the Steepest Descent technique to Quasi-Newton methods. Pearson II and BFGS showed the best performance in all the classical Quasi-Newton methods simulated. Other Quasi-Newton methods converged to local minima which does not show any weakness on their part. The problem with local minima is faced by all learning algorithms specially with neural networks which contain lots of local minima by nature. The condition numbers of the inverse Hessian approximates given by the BFGS method were in average lower than those of the other classical methods such as DFP, etc. This explains in part the better convergence rate of the BFGS algorithm. The initial state of XORI and its final state given by the BFGS learning algorithm are provided in figures 5.1 and 6.1 respectively.

The Pearson II algorithm is a much simpler algorithm than BFGS and therefore uses less number of floating point operations ( though more iterations ) to converge. However, Pearson II does not guarantee a positive definite inverse Hessian approximation while BFGS does, provided the right line search is used (see sec. 3.3.3.) Therefore there is a trade-off between the number of floating point operations and the confidence on positive definiteness of the inverse Hessian approximation (descent in direction.)

Results show an increase in the rate of convergence of up to 100% in the case of SSVM methods over BFGS and Pearson II. SSVM updates are known to perform well especially for problems with a large number of variables.<sup>[28]</sup> This fact is illustrated by the outstanding performance of the SSVM method compared to BFGS in the encoder problem where the size of the state vector "x" is 25. The SSVM algorithm of Oren and Spedicato generated inverse Hessian approximates which had even lower condition numbers than those generated by the BFGS method. Again, this is a predominant reason for their superb performance.

Quasi-Newton methods with initial scaling of the approximate inverse Hessian matrix are known to perform well especially for problems with a large number of variables.<sup>[36]</sup> This fact is illustrated by the outstanding performance of the BFGSa in XOR II and BFGSb in XOR I compared to BFGS without initial scaling. However, no major change is seen in the application of the initial scalings to the encoder problem. This is due to the fact that the initial scaling factor comes out to be very close to "one" meaning that the identity matrix is the best choice for the initial guess of the inverse Hessian matrix. Optimal conditioning introduced by the initial scaling

is also an important reason for the acceleration of convergence of the BFGS algorithm.

Figures 5.4 and 5.5 show a comparison of the number of presentations and the number of FLOP's required for convergence among steepest descent, Pearson II, BFGS, BFGS with initial scaling, and SSVM algorithms for the XOR I and encoder problems. Note that the charts are provided on a logarithmic scale and show two to three orders of magnitude reduction in the number of presentations and FLOP's from the steepest descent technique to the Quasi-Newton methods.

## **6.2. Conjugate Gradient Methods**

Among the conjugate gradient methods tested, the Fletcher-Reeves method showed the best performance. However, as seen in the results of the simulations, all conjugate gradient methods worked well with exact line searches and took much longer to converge when an inexact line search was used. Conjugate gradient methods require less memory space for operation than both Quasi-Newton and gradient-free methods and a little more than the Steepest Descent technique. However, there is a trade-off because of the need for an exact line search. These methods showed to be much more superior to the Steepest Descent technique in their rate of convergence specially considering the fact that they do not require much more memory space compared to the Steepest Descent. The performance of the conjugate gradient methods is still much worse than those of Quasi-Newton methods and the fact that inexact line searches could be used with Quasi-Newton methods makes them much more attractive than conjugate gradient methods even with the higher requirements for memory space.

### 6.3. Gradient-Free Methods

A quantitative comparison between Powell's methods and the methods of table 5.2 is not very informative for the reason that the whole purpose behind gradient-free schemes is to avoid gradient evaluations. Steepest descent, Quasi-Newton, and conjugate gradient methods make first gradient evaluations and other costly computations such as updating the inverse-Hessian approximate. With these points in mind for learning in neural networks, the gradient-free methods could prove more efficient than the techniques using gradients. An important feature of these gradient-free methods is that as long as the weights and activation function parameters of the network are changeable, the learning schemes do not require any knowledge of the internal structure and connectivity of the neurons. This makes the learning algorithm very general in its usage and easy to implement for almost any network.

Figures 5.7 and 5.8 show comparisons of the number of new directions and the number of floating point operations (FLOP's) required for convergence among Steepest Descent, BFGS, and the Powell methods. Powell's second method required the least direction evaluations and FLOP's. It should also be noted that most of the FLOP's were required by objective function evaluations in Powell's methods and by search direction evaluations in gradient methods. In a neural network, the least costly operations are, by design, those of function evaluations. This makes Powell's methods much more efficient in practice than what is depicted by available figures from simulations.

For learning control of repetitive processes, which will be discussed in detail in the next part of this thesis, the gradient-free learning techniques make it possible to include the controlled plant in the network as an additional neuron (Figure 6.2).

Learning with this configuration is not possible when gradient based learning schemes are used, since the plant is considered to be unknown and its derivatives are therefore not available. However, with gradient-free learning techniques discussed here, the output error of the plant could be minimized where in figure 6.2, the  $y$ 's and  $y_d$ 's are supervectors of actual and desired outputs respectively, containing data for all sample points in a repetition (*see section 7.2*).

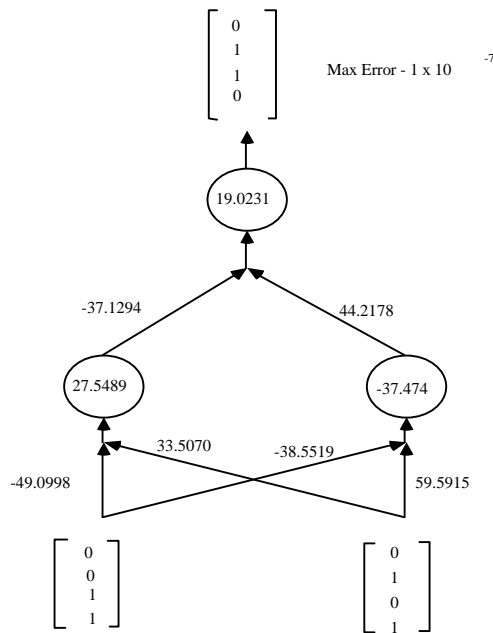


figure 6.1 (Final State of the Neural Network Simulating the XOR Logic as Produced by the BFGS Method)

#### 6.4. Choosing From Available Minimization Methods

SSVM Quasi-Newton methods seem to be the best choice when gradients are easily available and Powell's second method when gradient evaluations are expensive or impossible. However, a drawback of the proposed methods is the large memory size needed for retaining the inverse Hessian matrix or set of directions respectively. This problem can be solved in part, for Quasi-Newton methods, since the inverse Hessian is symmetric and not all the elements need to be kept in memory. However,

for problems with large number of neurons, this still poses a limitation. If memory storage is still a problem, then the best method would be the Fletcher-Reeves conjugate gradient method with an exact line search. Although, with the price of memory going down, one may still favor methods which result in greater speeds.

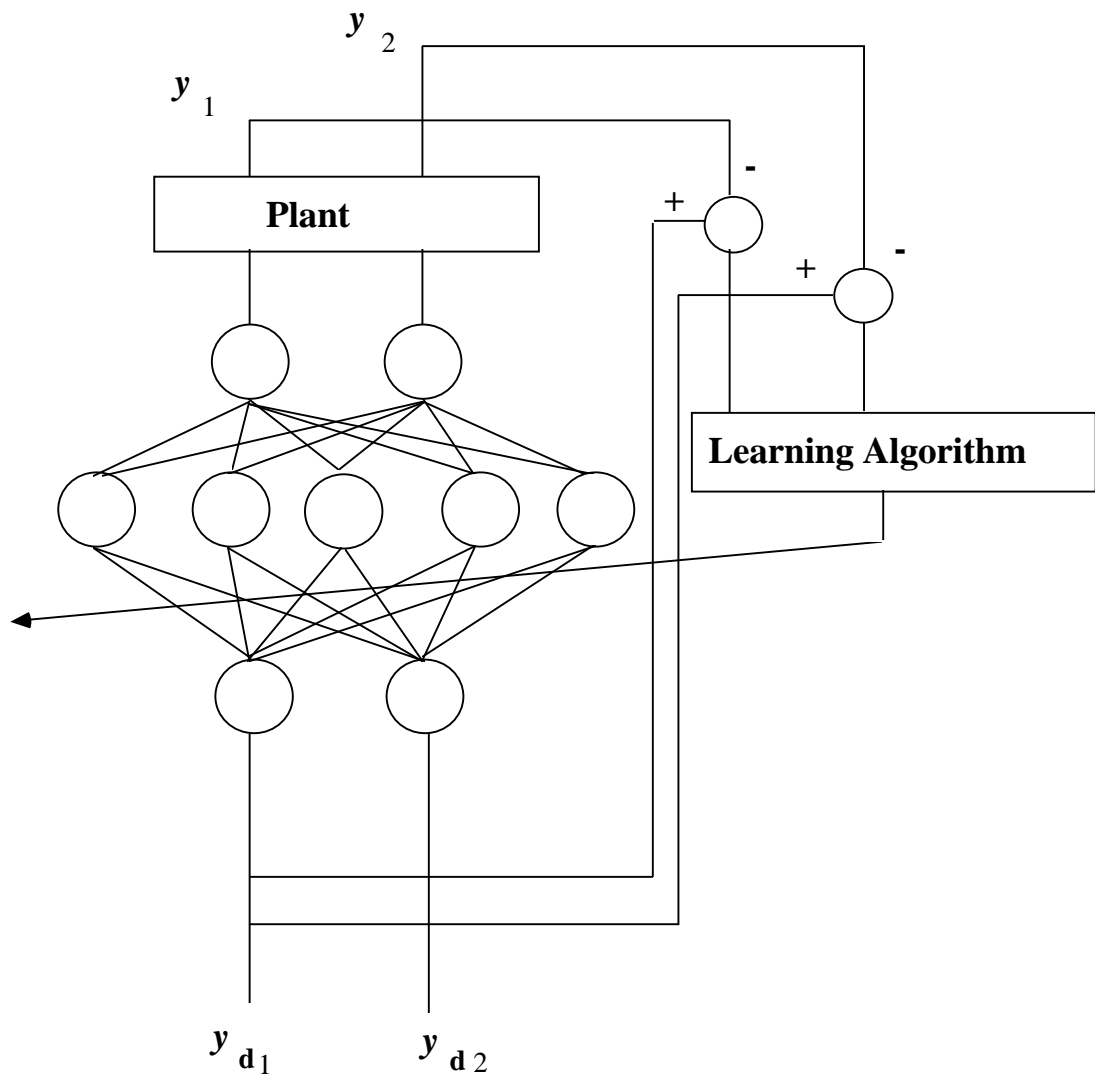


figure 6.2 (Control of a Repetitive System using Neural Networks)

## Part II: LEARNING CONTROL

## **Chapter 7**

### **Learning Control through Numerical Optimization**

## 7.1. Introduction

In this chapter, a time variant, discrete-time model of the learning control system is devised in the form of a system of linear algebraic equations relating the change in the state of the system to the change in the control action from one repetition of the task to another. This set of linear equations gives the transition between any two repetitions. This system is then solved for the appropriate control action that will minimize the tracking error of the controlled dynamic system, only requiring the availability of the order of the system, without any prior knowledge of the system parameters. This leads to a learning-adaptive controller.

## 7.2. Problem Formulation

We consider a general discrete-time linear time-varying or time-invariant system,

$$x^k(t+1) = A(t)x^k(t) + B(t)u^k(t) + w^k(t) \quad (7.1a)$$

$$y^k(t+1) = C(t+1)x^k(t+1) \quad t = 0, 1, 2, \dots, p-1 \quad k = 0, 1, 2, \dots \quad (7.1b)$$

where the state,  $x$ , is  $n$ -dimensional, the control,  $u$ , is  $m$ -dimensional, the output,  $y$ , is  $q$ -dimensional ( $q \geq m$ ),  $t$  is the time step in the  $p$ -step repetitive operation, and  $k$  is the repetition number. For simplicity, the dimension  $n$  of the system is assumed known, but generalization to just knowing an upper bound on  $n$  is easily considered. Also,  $A$ ,  $B$ ,  $C$ , and  $w^k$  are assumed unknown -- otherwise one could determine in advance what control to use to minimize tracking error, and there would be no need for learning control.

In the learning control problem (as contrasted with the repetitive control problem in [11]) the system is assumed to always start from the same initial state in each repetition of the task. Matrix  $A$  includes any state or output feedback control

present in the system and the symbol  $u^k$  is reserved for the signal added to the control for learning purposes. A time varying model is considered because many applications such as in robotics involve nonlinear dynamic systems which when linearized produce linear models with coefficients that vary with time step, and vary in the same manner each repetition. In such repetitive operations it is often the case that there will be disturbances  $w^k(t)$  that repeat with each repetition of the task, and the learning can be made to also correct for this source of errors in a natural way. One of the purposes of the feedback control is to handle any non-repetitive disturbances, and these will be ignored for purposes of designing the learning controller (some analysis of the effect of random noises in learning control can be found in [13]).

The solution to (7.1) can be written as,

$$x^k(t+1) = \left( \prod_{j=0}^t A(j) \right) x^k(0) + \sum_{j=0}^t \left( \left( \prod_{r=j+1}^t A(r) \right) B(j) u^k(j) + w^k(j) \right) \quad (2) \quad (7.2)$$

where the product symbol is taken to give the identity matrix if the lower limit is larger than the upper limit. Let  $p$  be the total number of time steps. Defining a difference operator  $\delta_k z(t) = z^k(t) - z^0(t)$  for any variable  $z$ , and using the fact that  $x^k(0)$  and  $w^k(t)$  are repetitive, one can write,

$$y^k = y^0 + P \delta_k u \quad (7.3)$$

where,

$$y^k = [y^kT(1) \ y^kT(2) \ \dots \ y^kT(p)]^T$$

$$u^k = [u^kT(0) \ u^kT(1) \ \dots \ u^kT(p-1)]^T$$

$$P = \begin{bmatrix} C(1)B(0) & 0 & \dots & 0 \\ C(2)A(1)B(0) & C(2)B(1) & & 0 \\ \cdot & & \dots & \cdot \\ C(p)A(p-1)A(p-2)\dots A(1)B(0) & \dots & \dots & C(p)B(p-1) \end{bmatrix}$$

The objective is to find a change in the control at each repetition  $k$  that will, as  $k$  progresses, minimize the quadratic error function

$$f(\delta_k u) = (y^k - y_D)^T (y^k - y_D) \quad (7.4)$$

where  $y_D$  is the desired output history for the  $p$ -step process. In previous work on learning control in discrete systems [13-15,61], as well as in the extensions of these papers for journal publication, attention had to be directed to the problem of ensuring that the special desired trajectory was in fact a feasible trajectory for the system to perform. For example, one method involved assuming knowledge of the system dimension  $n$ , and assuming the system is controllable, and then specifying the desired measured output variable history every  $n$  steps, which guaranteed to be a feasible specification by modern control theory. In the present approach these conditions can be relaxed if one is satisfied with minimizing (7.4) rather than insisting that (7.4) be driven to zero.

Substitute (7.3) in (7.4),

$$\begin{aligned} f(\delta_k u) &= (y^0 + P\delta_k u - y_D)^T (y^0 + P\delta_k u - y_D) \\ &= \delta_k^T (P^T P) \delta_k u + 2\delta_k^T [P^T (y^0 - y_D)] + (y^0 - y_D)^T (y^0 - y_D) \end{aligned} \quad (7.5)$$

Note that the gradient of this objective function and the Hessian matrix of second partials are,

$$\frac{df}{d(\delta_k u)} = 2 P^T P \delta_k u + 2 P^T (y^0 - y_D) \quad (7.6)$$

$$\left[ \frac{d^2 f}{d(\delta_k u)^2} \right] = 2 P^T P \quad (7.7)$$

It is now possible to characterize the nonlinear optimization problem represented by our learning control objective. We wish to

1. minimize a function which is known to be quadratic in the control change variable  $\delta_k u$ ,
2. but the first derivative of this function is not directly available, because we do not know the system matrices A, B, C, and hence do not know P,
3. and the second derivative is similarly unavailable.

Having characterized the problem, here are some preliminary assessments of the possible approaches to the learning control problem as formulated here.

### 7.3. Quasi-Newton Methods

Over the last couple of decades various finely tuned quasi-newton based methods have been generated for the minimization of unconstrained nonlinear functions. Among these methods are the rank 1 Broyden method, DFP, BFGS, and other members of the Broyden family of methods as well as SSVM methods ( see chapter 3 ). An important characteristic of these methods is the use of the iterative process to approximate the Hessian matrix so that no explicit expression for the second derivative is needed. If we could evaluate the first derivative  $df/d(\delta_k u)$ , then the BFGS method would converge to the optimal  $\delta_k u$  in approximately mp

repetitions. One might note that if one actually knows the  $df/d(\delta_k u)$  of (7.6), then one could equate it to zero and solve for  $\delta_k u$  and obtain the optimal  $\delta_k u$  in one step -- something which could be classified as a Newton method in numerical optimization.

Since we do not know  $df/d(\delta_k u)$ , one can consider the use of Quasi-Newton methods with approximation of the gradient obtained by a finite difference method. In order to obtain these differences for all of the  $mp$  elements of the gradient vector it would require approximately  $mp$  repetitions of the task to make one evaluation of the gradient vector. The total number of repetitions required for convergence would exceed  $(mp)^2$ . This makes such methods a poor choice in learning control. Methods that do not require the use of a gradient in picking the search direction are called for. Note that the same difficulty eliminates the use of steepest descent and conjugate gradient methods.

#### **7.4. A Direct Search Method**

A method which does not require knowledge of gradients, is the direct search method of optimization as expressed by Wood [54] and later by Hooke and Jeeves [53]. The direct search method takes advantage of the quadratic nature of the objective function is as follows:

1. Pick  $mp$  orthogonal directions in the  $\delta_k u$  space. For each of these directions in succession, perform the line search as in the next steps.
2. Take a step along the chosen direction in  $\delta_k u$  space, and apply the resulting control in the next repetition. Evaluate  $f$  from the data of this repetition.

3. If  $f$  decreased pick another step in the same direction, if it increased pick a step in the opposite direction, and apply to the system.

4. Since the quadratic objective function surface in the plane of the chosen direction is a parabola, the data from 2 and 3 determines this parabola, and can be used to find the minimizing  $\delta_k u$  for this direction. This completes the line search.

5. Return to 2 with the next direction.

This algorithm will improve the tracking every three repetitions, provided there is no noise in the measurements. There is no guarantee for a finite convergence unless the directions of search are mutually conjugate about the Hessian matrix of the quadratic objective function. Care must be taken to avoid large disturbances to the repetitive process during the line search.

### **7.5. Conjugate Direction Methods**

If the unidimensional search of the last section were used along  $m$  mutually conjugate directions, then in the absence of noise, the system would converge after minimizing the quadratic objective function in all these directions. There are a few conjugate direction methods available in the literature.<sup>[55,57-60]</sup> These methods start with a set of  $m$  orthonormal directions which could be coincident with the columns of an  $m \times m$  identity matrix. Then, by doing unidimensional searches in these directions, a new set of directions is generated.

Rosenbrock's<sup>[55]</sup> method provides a direction coincident with one of the eigenvectors of the Hessian ( $H$ ) matrix. Eigenvectors of  $H$  are mutually conjugate

and point in the direction of the minimum of the quadratic objective function. One such eigenvector is reached after  $mp$  unidimensional searches along the orthonormal directions. The rest of the  $mp-1$  directions are found by using the Gram-Schmidt [62] orthogonalization procedure. This cycle is then repeated to provide another eigenvector of  $H$ . There is no guarantee that the eigenvectors are not repeated. Therefore, there is no general finite convergence proof available.

Rosenbrock's method might converge to a non-minimum point in which case the method breaks down. To avoid or delay the failure of the minimization procedure, the method due to Davies, Swann, and Campey [56] (DSC) uses a renumbering system for the directions of search. Another method due to Powell (see 4.2.3) provides a direction set with one of the directions conjugate to the complementary  $mp-1$  directions. This set is obtained after a cycle similar to that of Rosenbrock and DSC. However, Powell's method does not require any orthogonalization procedure after every cycle and it never breaks down as Rosenbrock and DSC methods do.

None of the above methods guarantee a finite convergence since the main direction after each cycle might be linearly dependent on directions obtained in previous cycles. However, if the set of directions are linearly independent, then the function minimization will converge within  $mp(2mp+1)$  repetitions. Powell's second method provides this rate of convergence.

## **7.6. Generalized Secant Method**

### **7.6.1. Formulation**

Writing equation (7.2) for successive repetitions  $k$  and  $k+1$  and subtracting them from each other and assuming that disturbances  $w^k$  are repetitive with a period of  $p$  and that the initial conditions are repetitive from repetition  $k$  to  $k+1$ , we can write,

$$P (u^{k+1} - u^k) = y^{k+1} - y^k \quad (7.8)$$

and define,

$$e^k = y^k - y_d \quad (7.9)$$

$$v^k = u^{k+1} - u^k \quad (7.10)$$

Then equation (7.8) could be rewritten as,

$$Pv^k = e^{k+1} - e^k \quad (7.11)$$

Suppose that there exists some change in our input vector which would lead to a zero tracking error at the next repetition,  $e^{k+1}=0$ , then equation (7.11) for that input change could be written as follows,

$$P\tilde{v}^k = -e^k \quad (7.12)$$

If  $P^k$  is an approximation to  $P$  at the  $k^{\text{th}}$  repetition, then one can write,

$$P^k v^k = -e^k \quad (7.13)$$

where  $v^k$  is a change in control vector at repetition  $k$  that would use the information in the approximation to  $P$ ,  $P^k$ , and result in an  $e^{k+1}$  which is zero (or is minimum  $F$  norm as will be discussed later.)

Let us say that we are provided with an initial guess of the  $P$  matrix at the zeroth repetition,

$$P^0 v^0 = -e^0 \quad (7.14)$$

then,  $v^0$  could be solved for, by using the error vector  $e^0$  provided by the real system using control vector  $u^0$ .

Generally, an exact  $v^0$  might not exist, but a minimum error solution could be obtained for  $v^0$  by using the Moore-Penrose pseudo-inverse in the sense that  $v^0$  would minimize,

$$\| P^0 v^0 + e^0 \|_F$$

namely,

$$v^0 = -P^{0\dagger} e^0 \quad (7.15)$$

If we keep (7.13) satisfied for all  $k$ , then we could write,

$$v^k = -P^{k\dagger} e^k \quad \text{for all } k \quad (7.16)$$

minimizing  $\| P^k v^k + e^k \|_F$ .

At any repetition  $k$ , the actual system parameters  $P$  could be written as,

$$P = P^k + D^k \quad (7.17)$$

where  $D^k$  is a matrix of corrections for  $P^k$  at each repetition  $k$ .

Substituting for  $P$  in equation (7.11) from (7.17),

$$(P^k + D^k) v^k = e^{k+1} - e^k \quad (7.18)$$

or,

$$P^k v^k + D^k v^k = e^{k+1} - e^k \quad (7.19)$$

Solve  $D^k v^k$  from (7.19),

$$D^k v^k = e^{k+1} - e^k - P^k v^k \quad (7.20)$$

Since  $e^{k+1}$  is the error through the introduction of  $u^{k+1} = u^k + v^k$ , then the only unknown in equation (7.20) is the correction matrix  $D^k$ .

One solution to (7.20) would be,

$$D^k = \frac{(e^{k+1} - e^k - P^k v^k) z^k}{z^k T v^k} \quad (7.21)$$

where  $z^k$  are chosen in the following way<sup>[21]</sup>:

- If  $k \geq mp-1$  then  $z^k$  is chosen orthogonal to the previous  $mp-1$  control steps  $v^{k-mp+1}, \dots, v^{k-1}$ .

- If  $k < mp-1$  then  $z^k$  is chosen orthogonal to the available  $k$  steps,  $v^0, \dots, v^{k-1}$ .

One possible choice is to pick  $z^k$  as a linear combination of  $v^0, \dots, v^k$  which would be orthogonal to all  $v^0, \dots, v^{k-1}$ .

A few orthogonalization methods are available in the literature which may be used for the actual evaluation of the  $z$  vectors. These include the well-known Gram-Schmidt orthogonalization process<sup>[62]</sup> and a more advanced technique due to Fletcher<sup>[63]</sup> in which the number of vectors in the set are likely to be increased or decreased.

Then using an orthogonalization method and equations (7.16), (7.17), and (7.21) the following recursive algorithm will be generated,

$$v^k = -P^{k+1} e^k \quad (7.22)$$

$$P^{k+1} = P^k + \frac{(e^{k+1} - e^k - P^k v^k) z^k}{z^k T v^k} \quad (7.23)$$

### 7.6.2. Convergence

The set of  $z^k$  picked in equation (7.23) has the property that

$$D^k v^j = 0 \quad 0 \leq k-j < mp-1 \quad (7.24)$$

therefore,

$$P^{k+1} v^j = [P^{j+1} + D^{j+1} + \dots + D^k] v^j$$

$$= P^{j+1} v^j \quad 0 \leq k-j < mp-1 \quad (7.25)$$

Now assume that some  $n \leq mp$  is the maximum number of linearly independent  $v$ 's which could be found and thus  $v^n$  is a linear combination of all previous  $n$  linearly independent control steps. Then,

$$v^n = \alpha_0 v^0 + \dots + \alpha_{n-1} v^{n-1} \quad (7.26)$$

From equation (7.26),

$$P^n v^n = \sum_{j=0}^{n-1} \alpha_j P^n v^j \quad (7.27)$$

Also,

$$\begin{aligned} P^{i+1} v^j &= P^{j+1} v^j && 0 \leq j \leq n-1 \text{ (from eq. 7.25)} \\ &= e^{j+1} - e^j && \text{(from eq.'s 7.19 and 7.23)} \\ &= P v^j && \text{(from eq. 7.11)} \end{aligned} \quad (7.28)$$

Substituting (7.28) into (7.27),

$$P^n v^n = P \sum_{j=0}^{n-1} \alpha_j v^j$$

or,

$$P^n v^n = P v^n \quad (7.29)$$

Using equations (7.11), (7.19), and (7.29),

$$e^{n+1} - e^n = P^n v^n \quad (7.30)$$

$$e^{n+1} - e^n = P v^n \quad (7.31)$$

If  $v^n$  satisfies (7.13), then,

$$e^{n+1} - e^n = -e^n \quad (7.32)$$

which means that  $e^{n+1}$  must be zero. However, if  $v^n$  only satisfies (7.13) in a least F norm sense, then  $e^{n+1}$  would have a minimum F norm, or,

$$e^{n+1} = P v^n + e^n \quad (7.33)$$

where,  $e^{n+1}$  has minimum F norm.

If  $n$  (the rank of  $P^0-P$ ) is equal to  $mp$ , then after  $mp+1$  repetitions of the task, a minimum F norm tracking error will be achieved. Of course,  $n$ , the rank of  $P^0-P$  may be less than  $mp$  in which case if the directions in  $P^0$  containing the correct portions of  $P$  are given, then  $n$  would be less than  $mp$  by that number of correct guesses. However, if one is not sure of his initial guess for the  $P$  matrix, then  $mp+1$  is the maximum number of repetitions before convergence.

## 7.7. Simulation and Results

The learning control algorithm based on the Generalized Secant method was tested on two nonlinear dynamical systems. These systems are a Non-linear Mass-Spring-Dashpot and a One-Degree-of-Freedom Pendulum with damping. The details of these systems are given in the following two sections.

### 7.7.1. Non-Linear Mass-Spring-Dashpot (NLMSD)

The simulated NLMSD dynamics is given by the following differential equation:

$$m \ddot{\alpha} + c (1 + \alpha^2) \dot{\alpha} + (k |\alpha|) \alpha = T \quad (7.34)$$

where,

*Mass:*

$$\mathbf{m} = 1.0 \text{ kgr}$$

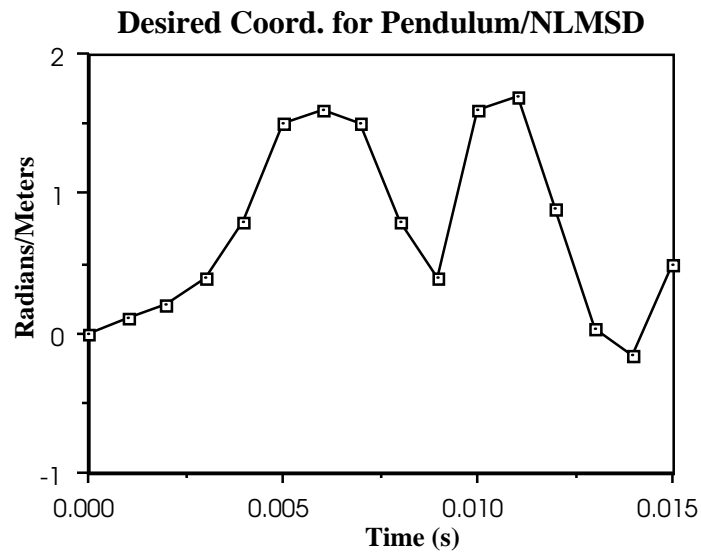
*Spring Constant:*

$$\kappa = 1.0 \text{ N/m}$$

*Damping Constant:*

$$c = 1.0 \text{ Ns/m}$$

The mass was asked to follow the trajectory of figure 7.1.

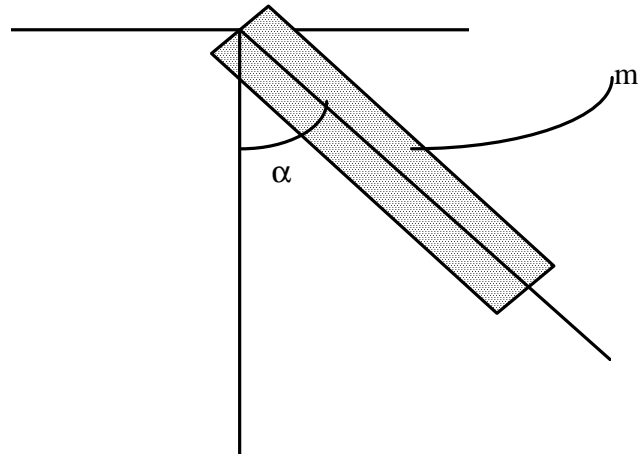


*figure 7.1 (Nonlinear Desired Trajectory Used for the Systems of Chapters 7 and 8)*

### 7.7.2. Pendulum

The dynamics of the simulated pendulum (*figure 7.2*) obeyed the following differential equation:

$$m l^2 \ddot{\alpha} + c \dot{\alpha} + m g l \sin \alpha = T \quad (7.35)$$



*figure 7.2 (The Pendulum Used for the Control Simulations of Chapter 7 and 8)*

where,

*mass:*

$$\mathbf{m} = 1.0 \text{ kgr}$$

*link dimension:*

$$l = 0.1 \text{ m}$$

*Damping Coefficient:*

$$\mathbf{c} = 1.0 \text{ Ns/m}$$

The robot was asked to swing across two radians so that the nonlinearity of the system cannot be neglected (*figure 7.1*).

### **7.7.3. Simulation Results**

A linear Mass-Spring-Dashpot (LMSD) was first simulated using the Generalized Secant Controller. This system is a linearization of the NLMSD system of equation 7.34 and is described by the following differential equation.

$$m \ddot{\alpha} + c \dot{\alpha} + k \alpha = T \quad (7.36)$$

The results of this simulation are given by figure 7.3 through plotting the sum of squares of the errors versus the repetition number. The first repetition was done

using a self-tuning regulator with forgetting factor 1 (a complete theory of the self tuning regulator is given in Chapter 8.) The sum of squares of errors in this simulation converges to 1.08 and did not go to zero.

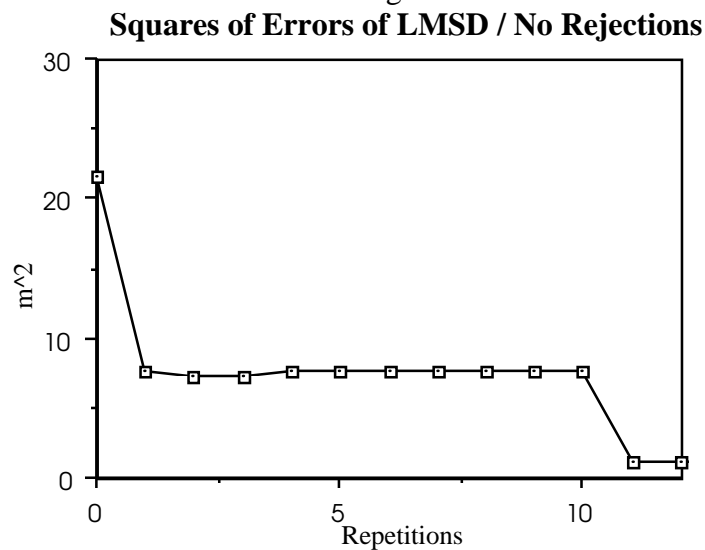
Barnes<sup>[21]</sup> proposes a practical stability criterion which translates to imposing a small positive lower limit to the inner product of the direction in which the P matrix is updated and the control step taken. Namely, the steps which do not meet the criterion of the inequality 7.37 should be rejected.

$$\frac{|z_k^T v_k|}{\|z_k\|_E \|v_k\|_E} > \rho \quad 0 < \rho < 1 \quad (7.37)$$

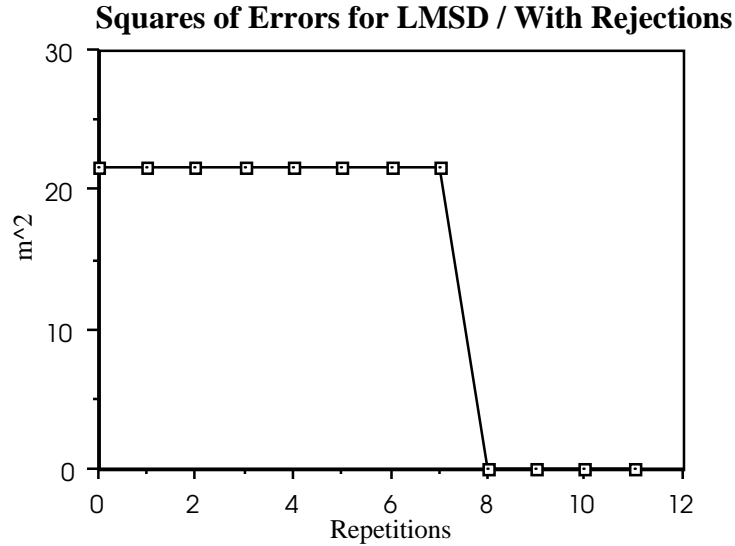
This ensures a numerically stable update to the P matrix. If a step is rejected for not meeting (7.37), then a new step  $v_k$  should be taken. If this step is made equivalent in the same direction as the P update direction,  $z_k$ , then the inequality of 7.37 is always satisfied. Note that 7.37 could be tested before the control step is inputted to the dynamic system. Therefore, if the step is rejected, for practical purposes there is no mathematical burden created. Barnes<sup>[21]</sup> suggests taking the magnitude of the new step to be equal to that of the rejected step. This suggestion was shown to trigger instabilities in practice when applied to simulated systems in this thesis. Therefore, to avoid instability, the size of the new steps were taken to be a small number which is greater than the accuracy of the computer and yet is small enough compared to the magnitude of the input vector in repetition 0, not to cause much deviation in the system dynamics. This choice turned out to be very practical when applied to the LMSD system (see figure 7.4.) To see the robustness of this controller, it was applied to the nonlinear systems of equation 7.34 and 7.35. figures 7.5 and 7.6 show the plots of the sums of squares of errors for the NLMSD and pendulum systems. In both of these simulations the rejection of control steps was done with  $\rho=1e-4$ .

In application to the LMSD and NLMSD systems, zero trajectory error was achieved after 8 repetitions which is less than the ceiling provided by the theory (10 repetitions.) Similarly, the pendulum reached a zero trajectory error after only 9 repetitions which is again one repetition less than the ceiling provided by the theory.

It should be noted that the theory in this chapter was derived for control of a time-variant dynamic system where the simulations were done on highly nonlinear systems performing a highly nonlinear task. These simulations provide practical evidence on the robustness of this learning controller.



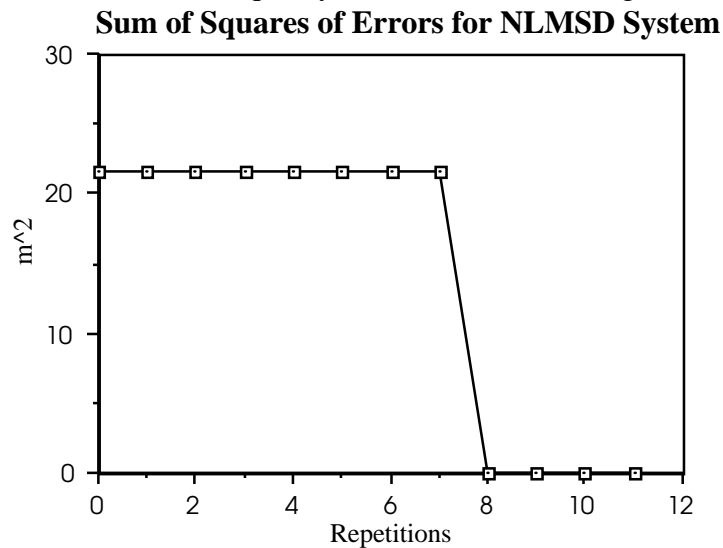
*figure 7.3 (Squares of Errors for the LMSD System  
Using the Generalized Secant Learning Controller without Rejections)*



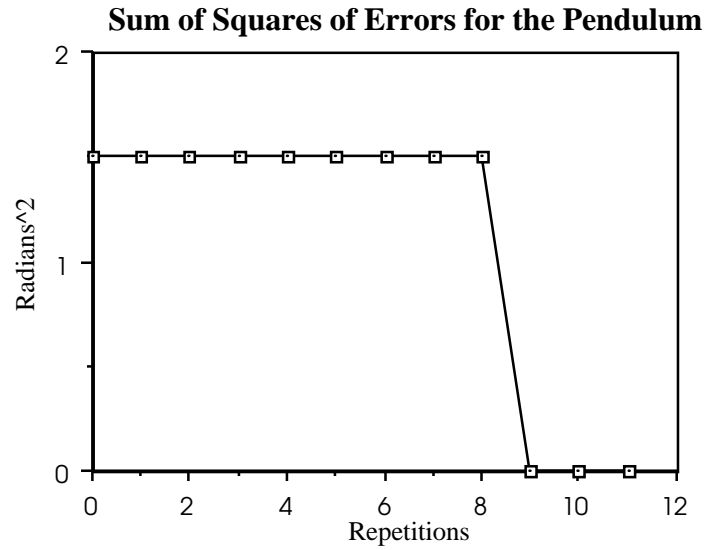
*figure 7.4 (Squares of Errors for the LMSD System  
Using the Generalized Secant Learning Controller with Rejections)*

### 7.8. Conclusion

Theoretical evidence shows that the Generalized Secant method requires the least number of repetitions for convergence. The Generalized Secant method was therefore used in the simulations of this chapter. Results show that without rejection of noninformative control steps, the system could become unstable and that the use of a  $\rho$  which is too small in inequality 7.37, could result a big reduction in the rate of



*figure 7.5 (Squares of Errors for the NLMSD System Using the  
Generalized Secant Learning Controller with Rejections)*



*figure 7.6 (Squares of Errors for the Pendulum Using the Generalized Secant Learning Controller with Rejections)*

convergence. Therefore, there is a trade-off in the magnitude of  $\rho$ . Also, in practice the magnitude of steps which replace rejected control steps should be made small enough not to cause instabilities in the system. In general, the Generalized Secant Learning Controller has shown to be very stable and robust in a practical sense when applied in the simulations of this chapter.

## **Chapter 8**

### **Self-Tuning Regulator with Learning Parameter Estimation**

## 8.1. Introduction

One of the major factors contributing to performance limitations of today's industrial automated machines/processes is the restricted capabilities of their control systems. A wide class of these controllers employ constant pre-defined gains and do not take into consideration the nonlinear dynamics in these machines. The result is that these machines are not being utilized to their full potential in terms of their speed and precision. With a more sophisticated control strategy, it is possible to compensate for the complicated effects of nonlinearities which have in the past been considered as mere disturbances in most systems.

Several advanced schemes have been proposed for an improved performance, which would generate control actions to compensate for the aforementioned nonlinear dynamics. These include nonlinear feedback control [64], feedforward control [65], resolved motion control [66-67], sliding mode control [68], repetitive control [11,69] and learning control [10,70]. In addition, adaptive controls have drawn a lot of attention in various applications [71-73]. One class of these adaptive controllers is the self-tuning regulator [74].

A self-tuning regulator consists of a parameter estimator and a controller. The parameter estimator estimates the parameters of an approximated model of the controlled system by utilizing a recursive estimation scheme. Based on the approximate model and the estimated parameters, the controller adjusts its actions to maintain its performance. Therefore, the performance of a self-tuning regulator depends greatly on that of the parameter estimator employed.

One common characteristic of the existing estimators is that they start their estimation process with each new task and do not use the information acquired from their past experiences even when performing the same task over and over again, as in many manufacturing applications. The implication is that the system will keep making the same errors at corresponding times in the duration of each repetition.

Another drawback of these estimators is that they have to keep the changes of estimations small between neighboring sample instants to maintain their immunity to noise and disturbances. Consequently, the estimators cannot respond to sudden changes of parameters quickly.

The objective is to present a learning adaptive control scheme based on a learning estimator which utilizes the information from past performances of a repetitive task to make a more accurate estimate of the system parameters repetition after repetition. Consequently, the learning adaptive controller improves its performance throughout the repetitions. Due to the addition of the repetition domain, the estimator is allowed to respond to sudden changes of parameters along the time axis.

## 8.2. Theory of the Self-Tuning Regulator

The general equations of motion of most rigid-body systems, with the consideration of dynamics such as inertial, gravity, Coriolis, centrifugal, and other forces will be as follows:

$$\mathbf{T} = \mathbf{M}(\boldsymbol{\alpha}) \ddot{\boldsymbol{\alpha}} + \mathbf{C}(\boldsymbol{\alpha}, \dot{\boldsymbol{\alpha}}) + \mathbf{F}(\dot{\boldsymbol{\alpha}}) + \mathbf{G}(\boldsymbol{\alpha}) + \mathbf{T}_d \quad (8.1)$$

where,  $\mathbf{T}$  is the  $n \times 1$  vector of generalized forces supplied by the actuators,  $\boldsymbol{\alpha}$  is the  $n \times 1$  vector of generalized coordinates,  $\mathbf{M}(\boldsymbol{\alpha})$  is the  $n \times n$  equivalent mass matrix,

$\mathbf{C}(\boldsymbol{\alpha}, \dot{\boldsymbol{\alpha}})$  is the  $n \times 1$  vector of generalized forces due to Coriolis and centrifugal forces,  $\mathbf{F}(\boldsymbol{\alpha}, \dot{\boldsymbol{\alpha}})$  is the  $n \times 1$  vector of generalized forces due to viscous friction,  $\mathbf{G}(\boldsymbol{\alpha})$  is the  $n \times 1$  vector of generalized forces due to gravity and  $\mathbf{T}_d$  is the  $n \times 1$  vector of disturbances, friction, and other unmodeled forces [73].

If equation (8.1) is discretized and linearized about a set of reference coordinates at each time instant, the nonlinear equations of motion for each of the generalized coordinates can be approximated by a set of second order linear difference equations.

Such a nominal linear difference equation can be written as:

$$\alpha(t) + \theta_1 \alpha(t-1) + \theta_2 \alpha(t-2) = q^{-d} (\theta_3 u(t) + \theta_4 u(t-1)) + v(t) \quad (8.2a)$$

or,

$$A(q^{-1}) \alpha(t) = q^{-d} B(q^{-1}) u(t) + v(t) \quad (8.2b)$$

where  $q^{-1}$  is the backward shift (or delay) operator such that,  $q^{-1} \alpha(t) = \alpha(t-1)$ ;  $d$  is the delay,  $u(t)$  is the input at time step  $t$ ,  $\alpha(t)$  is the output at time step  $t$ , and  $v(t)$  denotes the equation error.  $A(q^{-1})$  and  $B(q^{-1})$  are the system matrices which could change from one instant to the next.

Throughout this formulation the following assumptions are made:

1. the delay  $d$  is known,
2. all zeros of  $B(q^{-1})$  lie strictly inside the unit circle.

Taking  $d=1$  and solving for  $\alpha(t)$  in equations (8.2):

$$\alpha(t) = \boldsymbol{\theta}^T(t) \boldsymbol{\phi}(t) + v(t) \quad (8.3)$$

where,

$\boldsymbol{\theta}^T(t) = (\theta_1(t), \theta_2(t), \theta_3(t), \theta_4(t))$  is the parameter vector at time  $t$  and

$\phi^T(t) = ( -\alpha(t-1) , -\alpha(t-2) , u(t-1) , u(t-2) )$  is the linear regression vector.

A self-tuning regulator models a given dynamic system with a difference equation in the form of (8.3) and uses a recursive estimator to identify the parameter vector  $\theta(t)$  at each sampling instant and adjusts its control action accordingly. Therefore, the overall performance of the control system is very much dependent on how close these estimates are to the real system parameters.

One popular parameter estimator is the Recursive Least Squares (RLS) parameter estimator which minimizes the sum of squares of the equation errors  $v(t)$  over the time. Therefore, at time step  $k$ , the function,

$$V_k(\theta) = \sum_{t=0}^k \lambda^{k-t} v^2(t) \quad (8.4)$$

is minimized, where  $\lambda$  is a forgetting factor which is a measure of how fast the old data is forgotten. The range of  $\lambda$  is between 0 and 1 and as it becomes smaller the old data is forgotten more quickly. Using equation (8.3), the expression for  $V_k$  can be written:

$$V_k(\theta) = \sum_{t=0}^k \lambda^{k-t} [\alpha(t) - \theta^T(t) \phi(t)]^2 \quad (8.5)$$

Equation (8.4) is then minimized with respect to  $\theta$  and a parameter ( $\theta$ ) vector is obtained for time step  $k$ .

To recursively find a  $\theta$  vector which would minimize (8.5), the following algorithm can be used:[75-76]

$$L(k-1) = \frac{P(k-1) \phi(k)}{\lambda + \phi^T(k) P(k-1) \phi(k)} \quad (8.6a)$$

$$\theta(k) = \theta(k-1) + L(k-1) [ \alpha(k) - \theta^T(k-1) \phi(k) ] \quad (8.6b)$$

$$P(k) = \frac{[ I - L(k-1) \phi^T(k) ] P(k-1)}{\lambda} \quad (8.6c)$$

where,  $L$  is an intermediate vector which is calculated each time to avoid matrix inversions and  $P$  is the covariance matrix which signifies the degree of confidence in the accuracy of the parameter estimates and is started off as a diagonal matrix with large diagonal elements to minimize the effect of the initial guess for the parameter vector.

By nature, least squares estimators have to provide a new estimate of parameters close to the previous estimate. If the changes of the controlled plant are large between two consecutive steps and they continue to change, the parameter estimator may not keep up with the changes and poor estimates will be provided. These poor estimates will then lead to a poor control action on the part of the controller. One solution that comes to mind is increasing the sampling frequency which due to limited computational capabilities and quantization errors is usually not a practical solution.

If the task being executed is repetitive, then everytime the task is started, the same procedure is repeated and the controller starts everything fresh without looking back at how it had performed previously. Therefore, the controller will make the same errors each time it repeats the task. The following proposition provides a solution which will not only use the knowledge acquired from previous repetitions, but it will also allow greater parameter changes in neighboring time steps.

### 8.3. Learning Recursive Least Squares Estimator (LRLS):

Let us look at any sample instant  $k$  in the time domain. If the task being performed is repetitive, there is a nominal difference equation (8.7) which defines an equivalent linear time-variant system for the non-linear system at time step  $k$  and repetition  $r$ .

$$\alpha^r(k) = \boldsymbol{\theta}^{rT}(k) \boldsymbol{\phi}^r(k) + v^r(k) \quad (8.7)$$

where,

$$\boldsymbol{\theta}^{rT}(k) = [ \theta^{r_1}(k) , \theta^{r_2}(k) , \theta^{r_3}(k) , \theta^{r_4}(k) ], \quad (8.8a)$$

$$\boldsymbol{\phi}^{rT}(k) = [ -\alpha^r(k-1) , -\alpha^r(k-2) , u^r(k-1) , u^r(k-2) ], \quad (8.8b)$$

and  $v^r(k)$  denotes the equation errors.

Minimizing the sum of squares of the equation errors  $v^r(k)$  in the repetition domain, for time step  $k$ , will minimize,

$$V_k^r(\boldsymbol{\theta}) = \sum_{j=0}^r \gamma^{rj} [\alpha^j(k) - \boldsymbol{\theta}^{jT}(k) \boldsymbol{\phi}^j(k)]^2 \quad (8.9)$$

where,  $\gamma$  is the forgetting factor in the repetition domain, similar to  $\lambda$  in equation (8.5).

Minimizing equation (8.9) for  $\theta$ , will give a new estimate of the parameters ( $\theta$ ) for time step  $k$  and repetition  $r$  as presented in equation (8.7). To minimize (8.9) and solve for  $\theta$  in a recursive fashion, the following algorithm can be used:

$$L^{r-1}(k) = \frac{P^{r-1}(k) \phi^r(k)}{\gamma + \phi^{rT}(k) P^{r-1}(k) \phi^r(k)} \quad (8.10a)$$

$$\theta^r(k) = \theta^{r-1}(k) + L^{r-1}(k) [\alpha^r(k) - \theta^{r-1T}(k) \phi^r(k)] \quad (8.10b)$$

$$P^r(k) = \frac{[I - L^{r-1}(k) \phi^{rT}(k)] P^{r-1}(k)}{\gamma} \quad (8.10c)$$

These  $\theta^r(k)$  ( $k=0,1,2,\dots,N$ ) can be evaluated and stored after repetition  $r$  has been completed and before repetition  $r+1$  starts. When repetition  $r+1$  starts, corresponding values of  $\theta^r$  could be used at each time step  $k$  to apply an appropriate compensation through the control law used. As seen by equations (8.10), since  $\theta^j(k)$ 's ( $k=0,1,2,\dots,N$ ) are updated in the repetition domain, their difference is not limited to that allowed by the recursive least square estimator shown in (8.6).

#### 8.4. Control law:

Since the control law is not the emphasis of this research, the One-Step-Ahead control law was selected for its simplicity [71]. In general, the self-tuning regulator is very flexible with respect to the employment of control laws. Virtually any technique (pole-placement, minimum variance, etc.) can be accommodated. The One-Step-Ahead control law is the application of an inverse system while an estimate to the system parameters is known. The second order linear time-variant difference equation approximating the non-linear dynamics at any time instant  $k$  is:

$$\alpha(k) + \theta_1(k) \alpha(k-1) + \theta_2(k) \alpha(k-2) = \theta_3(k) u(k-1) + \theta_4(k) u(k-2) \quad (8.11)$$

If  $\alpha_d(k)$  denotes the desired coordinate at time step  $k$ , then at time step  $k-1$ , the control,

$$u(k-1) = \frac{\alpha_d(k) + \theta_1(k) \alpha(k-1) + \theta_2(k) \alpha(k-2) - \theta_4(k) u(k-2)}{\theta_3(k)} \quad (8.12)$$

should be applied.

For the first time of executing a repetitive task ( $r=0$ ), any common controller (PID, PD, adaptive, etc.) can be used. All inputs and outputs are recorded at every sampling instant. After completing the task for the first time, the parameters ( $\mathbf{P}$ 's and  $\boldsymbol{\theta}$ 's) of all time steps will be updated using equations (8.10) and stored in memory. Then, the task is repeated using the stored parameter estimates for determining the control at each time step using equation (8.12). The process of recording inputs and outputs, updating the parameters and executing the task is repeated from one repetition to the next.

In this study, among all the available control schemes, a self-tuning regulator using an RLS (eq. 8.6) estimator and a One-Step-Ahead control law was chosen to execute the task for the first time. This is done to demonstrate the advantage of learning. During the first run, since  $\boldsymbol{\theta}(k)$  is not known at time step  $k-1$ ,  $\boldsymbol{\theta}(k-1)$  should be used. Therefore, the expression used for  $u(k-1)$  in the first time of conducting the repetitive task is:

$$u(k-1) = \frac{\alpha_d(k) + \theta_1(k-1) \alpha(k-1) + \theta_2(k-1) \alpha(k-2) - \theta_4(k-1) u(k-2)}{\theta_3(k-1)} \quad (13) \quad (8.13)$$

However, in the following repetitions the above restriction does not exist and equation (8.12) can be used.

### **8.5. Convergence, Simulations, and Experimental Results:**

A self-tuning regulator, utilizing a stable controller, converges if the parameter estimates converge. This requires that the model structure used in the estimator be correct and that the input signal be sufficiently rich in frequencies. Since a least squares method is used, it is necessary that there be no correlation in the disturbances.<sup>[75]</sup>

The learning self-tuning regulator was applied, through computer simulation, to the control of the one degree-of-freedom robot manipulator and the nonlinear mass-spring-dashpot (NLMSD) system described in chapter 7. Forgetting factors used in the parameter estimator were,

$$\lambda = 0.98$$

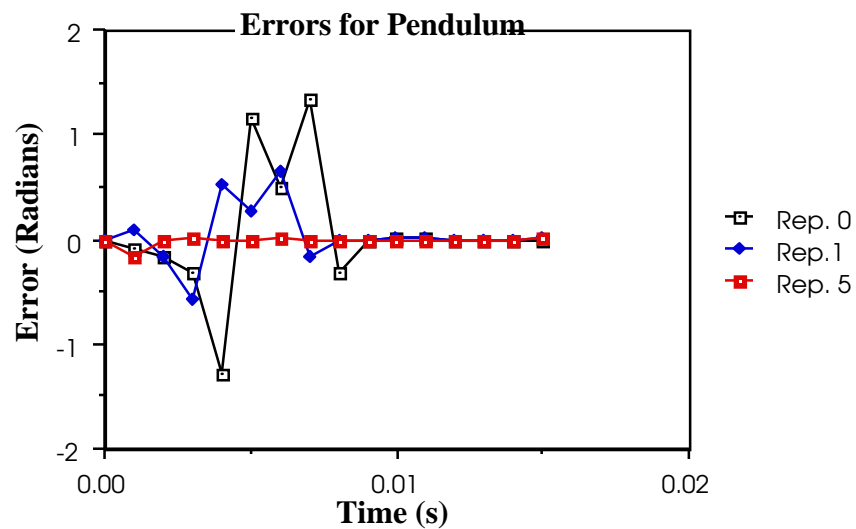
and,

$$\gamma = 0.98$$

Figures 8.1 and 8.2 present the simulation results with a plot of the tracking error versus time steps for different repetition cycles in the pendulum and NLMSD problems respectively. Figures 8.3 and 8.4 show that the sum of squares of errors decreases drastically toward zero as the number of repetitions increases.

These results show a near perfect regulation through the introduction of learning adaptive control. A perfect (zero) tracking error was not achieved due to the usage of a PD controller for the first two time steps of each repetition. Therefore, there is a

lower bound due to the error introduced by the PD controller in the first two steps, for the reduction of tracking error. The reason for using a PD in the first two time steps is the unavailability of data for the -1 and -2 time steps. However, if the time increment is small enough, then the parameters at time step 2 could be used to generate the control for the first two time steps. This will bring the limit closer to zero. Figures 8.3 and 8.4 show that an almost perfect tracking could be achieved after 4 repetitions of the task. One should remember that in real implementation of the self-tuning regulator, a rich persistent excitation to the dynamical system should be present such that all the states of the system are excited at all time even if there is no need for controlling them at certain instances.



*figure 8.1 (Errors for the Pendulum Problem Using the Learning Self-Tuning Regulator)*

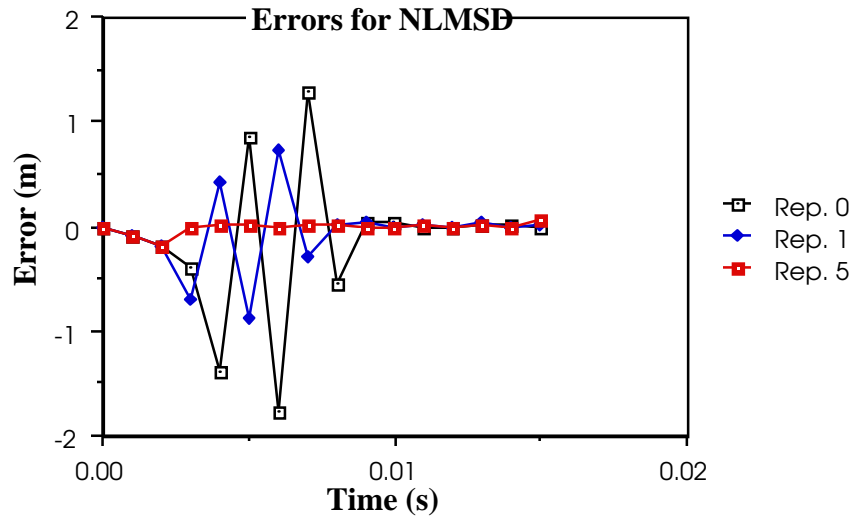


figure 8.2 (Errors for the NLMSD System Using the Learning Self-Tuning Regulator)

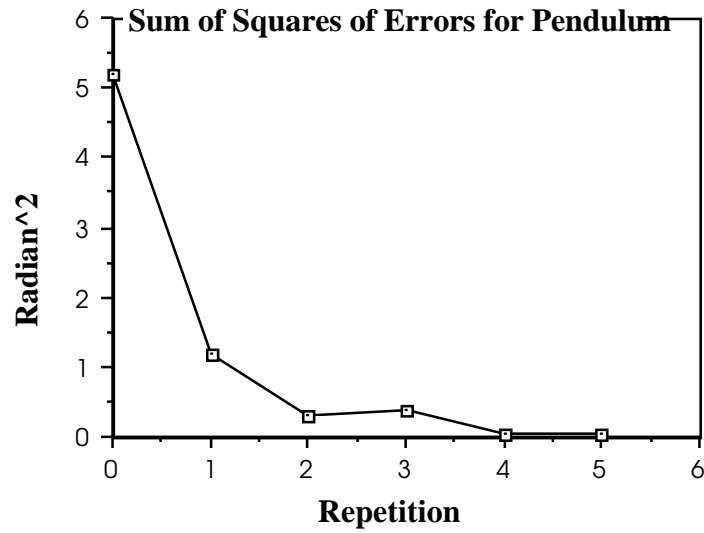
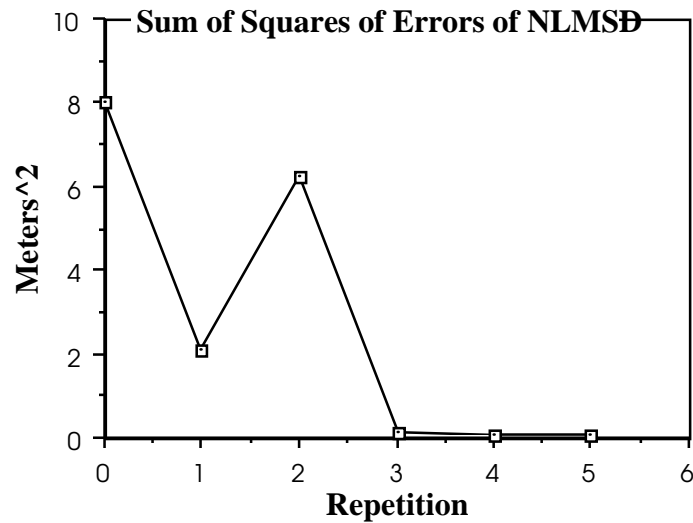


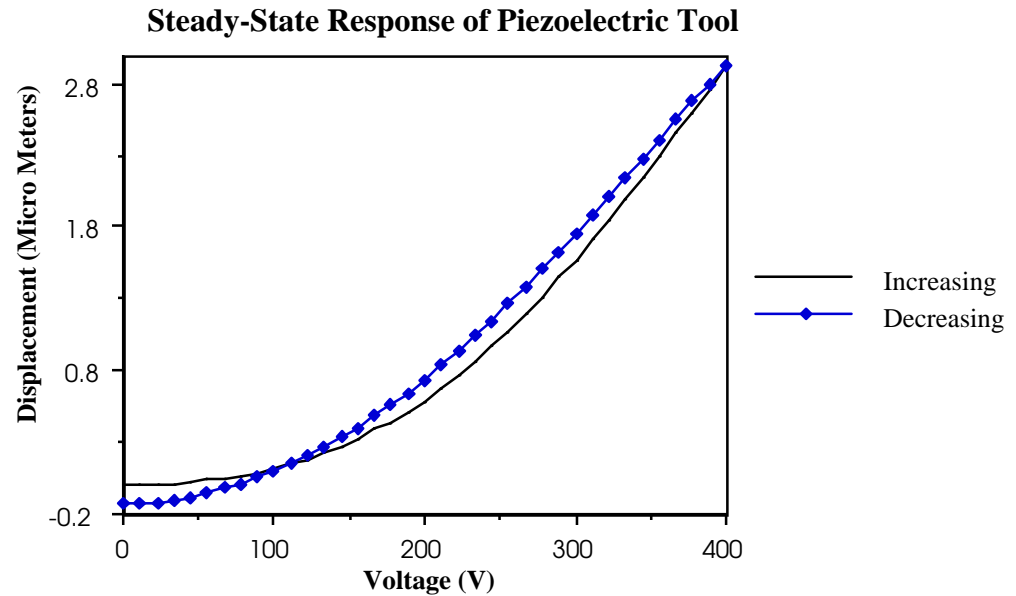
figure 8.3 (Sum of Squares of Errors For the Pendulum Using the Learning Self-Tuning Regulator)



*figure 8.4 (Sum of Squares of Errors for the NLMSD System Using the Learning Self-Tuning Regulator)*

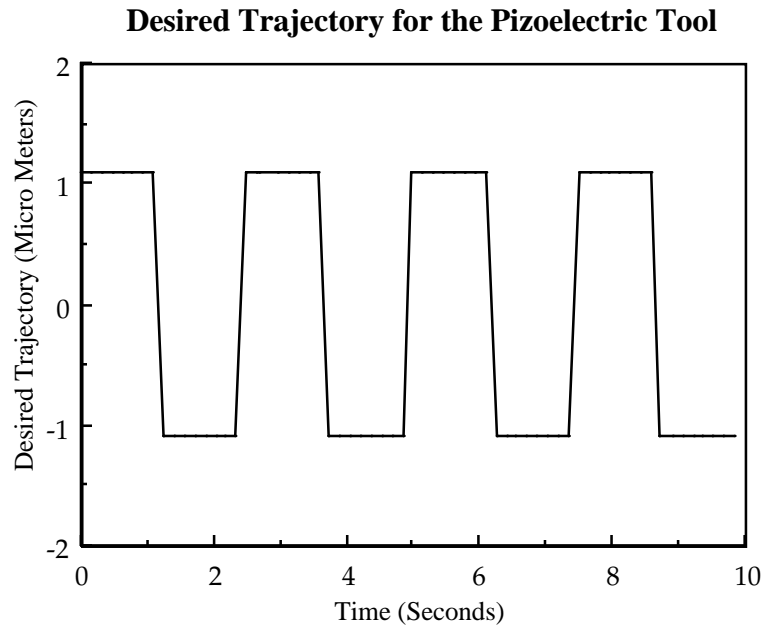
Simulations were also conducted using a PD controller for the first execution of the repetitive task and similar learning curves were observed.

In an experiment conducted by Mr. S. Y. Li who is an Associate Professor in Department of Precision Machinery of ChangSha Institute of Technology in the People's Republic of China, a Piezoelectric tool in a diamond cutting lathe was controlled used this learning control scheme. The dynamics of this tool is highly nonlinear and it features hysteresis. This highly nonlinear dynamics can be noted by looking at figure 8.5 which is a graph of the steady-state response of the Piezoelectric tool to the range of input voltage from 0 to 400 volts.

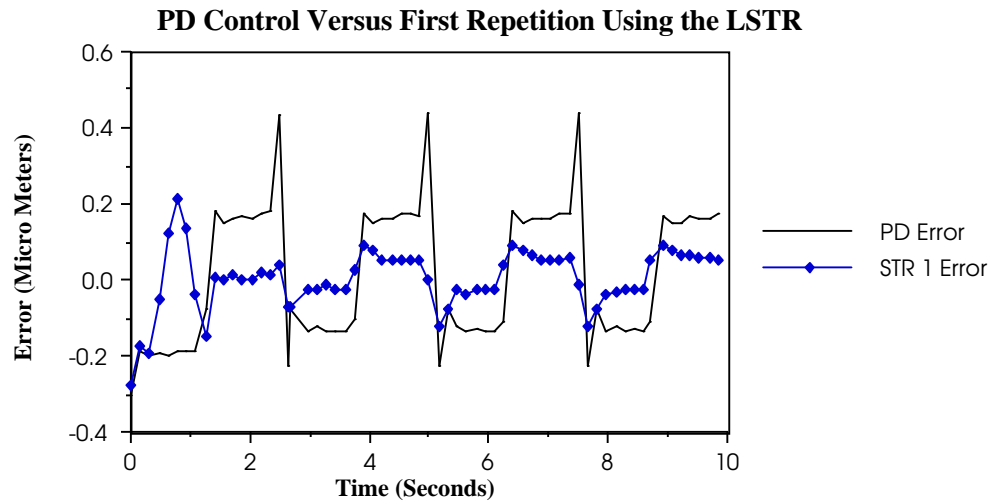


*figure 8.5 (Steady State response of the piezoelectric tool to input voltage 0-400V, signifying the hysteresis in its dynamics)*

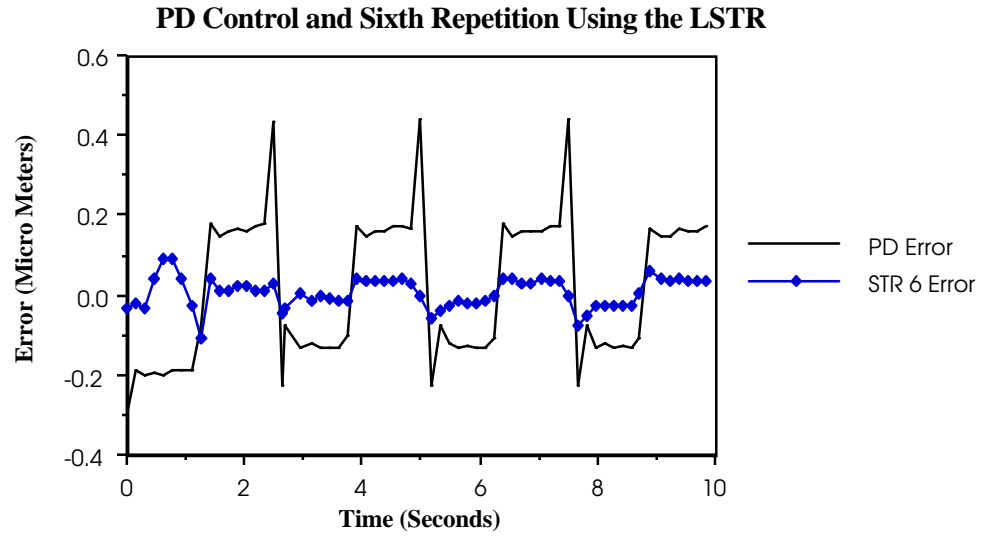
The Piezoelectric tool was asked to perform the demanding desired trajectory given in figure 8.6 having a Period of 2.3435 Seconds and a sampling interval of 0.15625 Seconds for a period of 10 Seconds in each repetition of the task. The computer used with this setup to do the calculations was a 25 MHz Intel-80386 machine with a math coprocessor. The computational speed of this computer does not allow the usage of the traditional Self-tuning Regulator for the first performance. For this reason, a PD controller was used to perform the control at the first time. Then, the Learning Self-tuning Regulator was used to repeat the task 10 times. Figures 8.7-8.9 show a comparison of the trajectory error of the PD controller versus the first, sixth and tenth repetition using the LSTR.



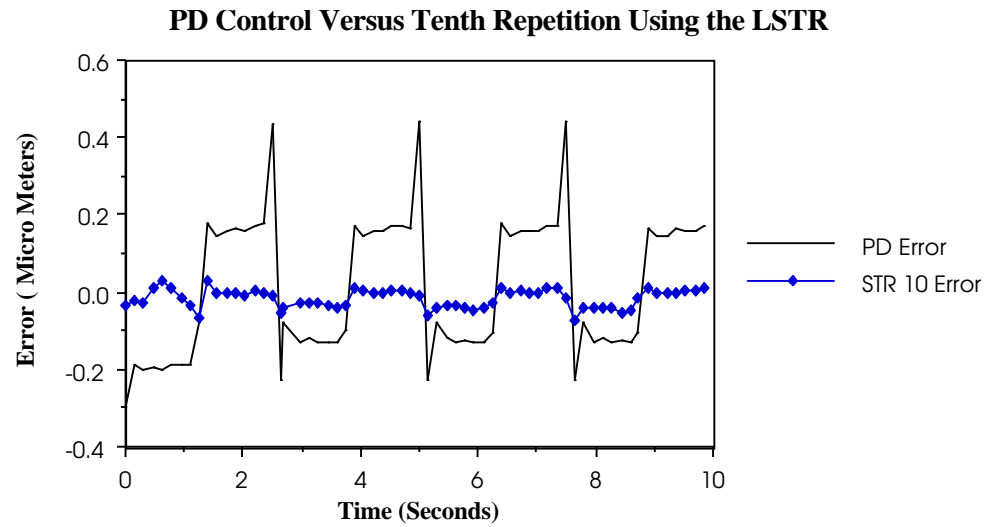
*figure 8.6 (Desired output for the Piezoelectric tool in each repetition)*



*figure 8.7 (Output error of the Piezoelectric tool for the first execution of the task using PD Control and the first repetition using the LSTR)*



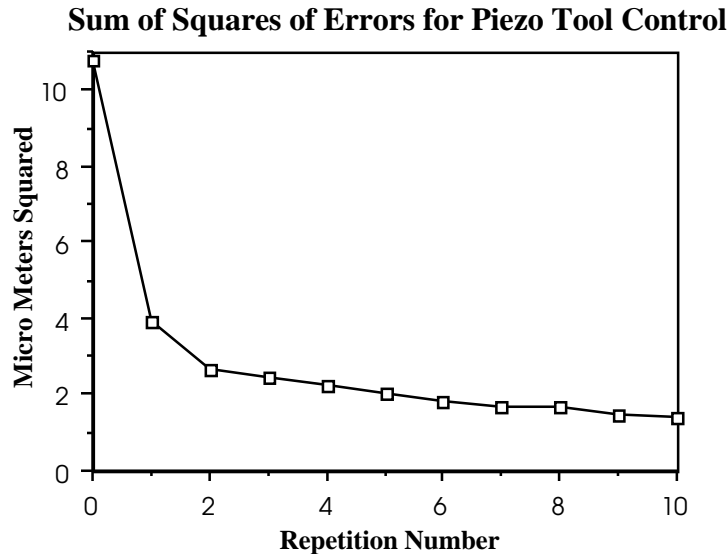
*figure 8.8 (Output error of the Piezoelectric tool for the first execution of the task using PD Control and the sixth repetition using the LSTR)*



*figure 8.9 (Output error of the Piezoelectric tool for the first execution of the task using PD Control and the tenth repetition using the LSTR)*

As seen by through the graphs, with only one repetition of the task, the overshoot has decreased drastically and it continues to decrease as repetitions increase. Figure 8.9 shows an almost perfect regulation of the tool as compared with the performance of the PD controller. Figure 8.10 gives a more informative graph of the tend of the sum

of squares of errors in different executions of the task. This value has fallen by about 60% upon the first repetition and by close to 90% at the tenth repetition, using the LSTR.



*figure 8.10 (Sum of squares of output errors of the Piezoelectric tool for the first execution of the task using PD Control and the ten repetition using the LSTR)*

## 8.6. Conclusion

Simulation results show that the learning self-tuning regulator is a quick converging controller which improves the performance of a robot manipulator and a nonlinear mass-spring-dashpot system executing repetitive tasks. Also, the fact that most calculations could be done off-line, makes this control algorithm very feasible. In fact, compared with a PD (proportional plus derivative) controller, only three more additions and three more multiplications have to be done while the mechanical system is in motion for each coordinate at each time step. This makes the use of the learning self-tuning regulator quite feasible with today's technology. Also, the amount of memory needed for the implementation of this controller in an  $n$  degree of freedom system is,

memory locations needed =  $22 * \# \text{ of time steps in each repetition} * n$ .

This amount of memory, considering today's low cost memory chips, is feasible for most applications, using even a personal computer. Therefore, it is proposed for future research to apply the learning algorithms developed in this and the previous chapter and study the performance of these algorithms as the number of degrees of freedom are increased and nonlinearities such as coriolis and centrifugal forces are introduced into the dynamic systems.

The proposed learning self-tuning regulator can be applied to the control of general nonlinear dynamic systems as demonstrated in the simulations. It is seen from the simulations and experimental results to be very robust when applied to controlling highly nonlinear plants which feature nonlinearities such as hysteresis, generating parameter drifts in the learning domain. This controller could be applied to many other manufacturing applications such as robotics, machining, process controls and other manufacturing processes. Also, the notion of a learning parameter estimator could be applied on other parameter estimators than the RLS estimator. These are some paths for future research.

## **References / Appendices**

## REFERENCES

- [1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation", in D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, MIT Press, 1986, pp. 675-695.
- [2] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, April 1987, pp. 4-22.
- [3] D. B. Parker, "A Comparison of Algorithms for Neuron-Like Cells," *Neural Networks for Computing*, AIP conference Proceeding 151, Snowbird, Utah, 1986, pp. 327-332.
- [4] S. E. Fahlman, "Faster-Learning Variations on Back-Propagation: *An Empirical Study*," 1988 Connectionist Models Summer School, 1988, pp.38-51.
- [5] T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, "Accelerating the Convergence of the Back-Propagation Method," *Biological Cybernetics*, Vol. 59, 1988, pp. 257-263.
- [6] S. A. Solla, E. Levin, M. Fleisher, "Accelerated Learning in Layered Neural Networks," *Complex Systems*, Vol. 2, 1988, pp. 625-640.
- [7] A. J. Owens and D. L. Filkin, "Efficient Training of the Back Propagation Network by Solving a System of Stiff Ordinary Differential Equations," *submitted for publication at IEEE/INNS International Conference on Neural Networks*, Washington D.C., June 19-22, 1989.
- [8] David M. Himmelblau, *Applied Nonlinear Programming*, McGraw-Hill Book Company, New York, 1972, p.87.
- [9] S. Becker, Y. Le Cun, "Improving the Convergence of Back-Propagation Learning with Second Order Methods," 1988 Connectionist Models Summer School, 1988, pp. 29-37.
- [10] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering Operations of Robots by Learning," *Journal of Robotic Systems*, Vol. 1, No. 2, 1984, pp. 123-140.
- [11] R. H. Middleton, G. C. Goodwin, and R. W. Longman, "A Method for improving the Dynamic Accuracy of a Robot Performing a Repetitive Task," *the International Journal of Robotics Research*, Vol. 8, No. 5, October 1989, pp. 67-74.
- [12] E. G. Harokopos and R. W. Longman, "A Learning Controller for Tracking Arbitrary Inputs in Robot Motions," *Proceeding of the Tenth IASTED International*

Symposium on Robotics and Automation, Lugano, Switzerland, June 29 - July 2, 1987.

[13] M. Phan and R. W. Longman, "A Mathematical Theory of Learning Control for Linear Discrete Multivariable Systems," Proceedings of the AIAA/AAS Astrodynamics Specialist Conference, Minneapolis, Minnesota, August 1988.

[14] M. Phan and R. W. Longman, "Liapunov Based Model Reference Learning Control," Proceedings of the Twenty-Sixth Annual Allerton Conference on Communication, Control, and Computing, Allerton House, Monticello, Illinois, 1988.

[15] M. Phan and R. W. Longman, "Indirect Learning Control with Guaranteed Stability," Proceedings of the 1989 Conference on Information Sciences and Systems, Johns Hopkins University, Baltimore, MD, 1989.

[16] J. Greenstadt, "On the Relative Effectiveness of Gradient Methods," Mathematics of Computation, Vol. 21, 1967, pp. 360-367.

[17] D. W. Marquardt, "An Algorithm of Least Squares Estimation of Nonlinear Parameters," SIAM Journal, Vol. 11, 1963, p.431.

[18] K. L. Levenberg, Quart. Appl. Math., Vol. 2, 1944, P.164.

[19] S. M. Goldfeld, R. E. Quandt, and H. F. Trotter, "Maximization by Quadratic Hill Climbing," Econometrica, Vol.34, 1966, p.541.

[20] W. C. Davidon, "Variable Metric Method for Minimization," AEC Res. and Dev. Report ANL-5990, 1959.

[21] J. G. P. Barnes, "An Algorithm for Solving Nonlinear Equations Based on the Secant Method," Computer Journal, Vol. 8, 1965, pp. 66-72.

[22] C. G. Broyden, "A Class of Methods for Solving Nonlinear Simultaneous Equations," Maths. Comp., Vol. 19, 1965, pp. 577-593.

[23] C. G. Broyden, "Quasi-Newton Methods and Their Application to Function Minimization," Maths. Comp., Vol. 21, 1967, pp. 368-381.

[24] W. C. Davidon, "Variance Algorithms for Minimization," Computer Journal, Vol. 10, 1968, pp.406-410.

[25] A. V. Fiacco and G. P. McCormick, Nonlinear Programming, John Wiley, New York, 1968.

- [26] B. A. Murtagh and E. W. H. Sargent, "A Constrained Minimization Method with Quadratic Convergence," Optimization (Ed. R. Fletcher), Academic Press, London, 1969, Ch. 14.
- [27] J. D. Pearson, "Variable Metric Methods of Minimization," Computer Journal, Vol. 12, 1969, p. 171-178.
- [28] R. Fletcher, Practical Methods of Optimization, John Wiley & Sons, New York, 1987, pp. 54-56.
- [29] C. G. Broyden, "The Convergence of a Class of Double Rank Minimization Algorithms," Parts I and II, J. Inst. Maths. Applns., Vol. 6, 1970, pp. 76-90, 222-231.
- [30] R. Fletcher, "A New Approach to Variable Metric Algorithms," Computer Journal, Vol. 8, 1970, pp.317-322.
- [31] D. Goldfarb, "A Family of Variable Metric Methods Derived by Variational Means, Maths. Comp., Vol. 24, 1970, pp.23-26.
- [32] D. F. Shanno, "Conditioning of Quasi-Newton Method for Function Minimization," Maths. Comp., Vol. 24, 1970, pp.647-656.
- [33] J. Greenstadt, "Variations on Variable-Metric Methods," Maths. Comp., Vol. 24, 1970, pp.1-22.
- [34] S. S. Oren, "On the Selection of Parameters in Self Scaling Variable Metric Algorithms," Mathematical Programming, Vol.7, 1974, pp.351-367.
- [35] S. S. Oren and E. Spedicato, "Optimal Conditioning of Self-Scaling Variable Metric Algorithms," Mathematical Programming, Vol. 10, 1976, pp.70-90.
- [36] D. F. Shanno and Kang-Hoh Phua, "Matrix Conditioning and Nonlinear Optimization," Mathematical Programming, Vol. 14, 1978, pp. 149-160.
- [37] H. Y. Huang, "Unified Approach to Quadratically Convergent Algorithms for Function Minimization," Journal of Optimization Theory and Applications, Vol. 5, No. 6, 1970, pp.405-422.
- [38] G. P. McCormick and K. Ritter, "Methods of Conjugate Directions Versus Quasi-Newton Methods," Mathematical Programming, Vol. 3, 1972, pp. 101-116.
- [39] D. H. Jacobson and W. Oksman, "An Algorithm that Minimizes Homogeneous Functions of  $n$  Variables in  $n+2$  Iterations and Rapidly Minimizes General Functions," Technical Report 618, Division of Engineering and Applied Physics, Harvard University, Cambridge, MA, 1970.

- [40] E. Spedicato, "Computational Experience with Quasi-Newton Algorithm for Minimization Problems of Moderately Large Size," Report CISE-N-175, CISE Documentation Service, Segrate (Milano), 1975.
- [41] S. Hoshino, "A Formulation of Variable Metric Methods," J. Inst. Maths Applics, Vol. 10, 1972, pp. 394-403.
- [42] W. C. Davidon, "Optimally Conditioned Optimization Algorithms Without Line Searches," Mathematical Programming, Vol. 9, 1975, pp. 1-30.
- [43] Robert B. Schnabel, "Analyzing and Improving Quasi-Newton Methods for Unconstrained Optimization," Ph.D. Thesis, Cornell University, August 1977.
- [44] M. J. Box, "A Comparison of Several Current Optimization Methods," The Computer Journal, Vol. 9, 1966, pp. 67-77.
- [45] D. F. Shanno and K. H. Phua, "Numerical Comparison of Several Variable Metric Algorithms," MIS Technical Report 21, University of Arizona, 1977.
- [46] R. Fletcher and C. M. Reeves, "Function Minimization by Conjugate Gradients," The Computer Journal, Vol. 7, 1964, p. 149-154.
- [47] Hestenes, M. R. and Stiefel, E., "Methods of Conjugate Gradients for Solving Linear Systems," J. Res. N. B. S., Vol. 49, 1952, p.409.
- [48] Beckman, F. S., "The Solution of Linear Equations by the Conjugate Gradient Method," Mathematical Methods for Digital Computers, Ralston, A. and Wolf, H. S. (Eds.), Wiley, New York, 1960.
- [49] David M. Himmelblau, Applied Nonlinear Programming, McGraw-Hill Book Company, New York, 1972, p.98.
- [50] B. V. Shah, R. J. Buehler and O. Kempthorne, "Some Algorithms for Minimizing Function of Several Variables," SIAM Journal, Vol. 12, No. 1, 1964, pp. 74-92.
- [51] G. Zoutendijk, Methods of Feasible Directions, American Elsevire Publishing Company, New York, 1960.
- [52] A. S. Householder, The Theory of Matrices in Numerical Analysis, Blaisdell Publishing Company, Mass., 1964, p.81.
- [53] R. Hooke and T. Jeeves, "Direct Search Solution of Numerical and Statistical Problems," J. Assoc. Computer Mach., Vol. 8, No. 2, April 1961, pp. 212-229.

- [54] C. F. Wood, "Application of Direct Search to the Solution of Engineering Problems," Westinghouse Research Laboratory Scientific Paper 6-41210-1-p1, 1960.
- [55] H. H. Rosenbrock, "Automatic Method for Finding Greatest or Least Value of Function," *The Computer Journal*, Vol. 3, No. 3, Oct. 1960, pp. 175-184.
- [56] W. H. Swann, "Report on the Development of a New Direct Search Method of Optimization," Imperial Chemical Industries, Ltd. Central Instr. Lab. Res. Note 6413, 1964.
- [57] M. J. D. Powell, "An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives," *The Computer Journal*, Vol. 7, 1964, pp. 155-162.
- [58] C. S. Smith, "The Automatic Computation of Maximum Likelihood Estimates," N. C. B. Scientific Department Report S.C. 846/MR/40, 1962.
- [59] W. Zangwill, "Minimizing a Function Without Calculating Derivatives," *The Computer Journal*, Vol. 10, 1967, pp. 293-296.
- [60] R. Fletcher, "Function Minimization Without Evaluating Derivatives - A Review," *The Computer Journal*, Vol. 8, 1965, pp. 33-41.
- [61] C. James Li and Homayoon S. M. Beigi, "A Self-tuning Regulator with Learning Parameter Estimation," Presented at the ASME Winter Annual Meeting, Dallas, November 25-30, 1990.
- [62] B. Noble and J. W. Daniel, *Applied Linear Algebra*, Second Edition, Prentice-Hall, NJ, 1977, pp. 138-143.
- [63] R. Fletcher, "A Technique for Orthogonalization," *J. Inst. Maths Applics*, Vol. 5, 1969, pp. 162-166.
- [64] A. K. Bejczy, T. J. Tarn, Y. L. Chen, "Computer Control of Robot Arms," Proceedings of first IEEE International Conference on Robotics and Automation, St. Louis, Missouri, March 1985.
- [65] J. J. Craig, *Introduction to Robotics Mechanics and Control*, Addison-Wesley, Reading, Massachusetts, 1986.
- [66] D. E. Whitney, "Resolved Motion Rate Control of Manipulators and Human Prostheses," *IEEE Transactions of Man, Machines, and Systems*, Vol. MMS-10, No. 2, June 1969, pp. 47-53.

- [67] J. Y. S. Luh, M. W. Walker, and R. P. Paul, "Resolved Acceleration Control of Mechanical Manipulators," IEEE Transactions on Automatic Control, Vol. AC-25, No.3, June 1980.
- [68] J. E. Slotine, "Sliding Controller Design for Non-linear Systems," International Journal of Control, Vol. 40, No. 2, 1984, pp. 421-434.
- [69] M. C. Tsai, G. Anwar, M. Tomizuka, "Discrete Time Repetitive Control for Robots", Proceedings to IEEE Conference, 1988.
- [70] S. Kawamura, F. Miyazaki and S. Arimoto, "Applications of Learning Methods for Dynamic Controls of Robot Manipulators," Proceeding of the 24th IEEE CDC, Fort Lauderdale, Florida, Dec. 1985.
- [71] G. C. Goodwin and K. S. Sin, Adaptive Filtering Prediction and Control, Prentice-Hall, New Jersey, 1984.
- [72] S. Dubowsky, and D. T. DeForges, "The Application of Model Referenced Adaptive Control to Robot Manipulators," ASME Journal of Dynamic Systems, Measurement, and Control, Vol. 101, Sep. 1979, pp. 193-200.
- [73] J. J. Craig, Adaptive Control of Mechanical Manipulators, Addison-Wesley Publishing Company, New York, 1987.
- [74] K. J. Åström, "Adaptive Feedback Control," Proceedings of the IEEE, Vol. 75, No. 2, February 1987.
- [75] K. J. Åström, B. Wittenmark, Computer Controlled Systems Theory and Design, Prentice-Hall, New Jersey, 1984.
- [76] L. Ljung and T. Soderstrom, Theory and Practice of Recursive Identification, The MIT Press, Cambridge, Massachusetts, 1986.

# **Appendix A**

## **A Summary of the Pertinent Mathematics**

### A.1. Definitions

#### *Real / Complex Space:*

$\mathbb{R}^N$  and  $\mathbb{C}^N$  denote the  $N$  dimensional real and complex spaces respectively.

#### *The Identity Matrix:*

The  $N$  dimensional identity matrix is denoted by  $I_N$  ( or sometimes  $I$  ) and is defined as follows,

$I_N : \mathbb{R}^N \rightarrow \mathbb{R}^N$  is the matrix such that

$$I_{ij} = \begin{cases} 1 & i=j \\ 0 & i \neq j \end{cases}$$

where  $i, j \in \{ 1, 2, \dots, N \}$  are the row number and column number of the corresponding element of matrix  $I_N$ .

#### *Transpose / Hermitian Transpose:*

##### *Transpose of a Matrix:*

The transpose of any matrix  $A : \mathbb{R}^N \rightarrow \mathbb{R}^M$  is given by  $A^T : \mathbb{R}^M \rightarrow \mathbb{R}^N$  such that,

$$A_{ji}^T = A_{ij} \tag{A.1}$$

where indices  $i \in \{ 1, 2, \dots, M \}$  and  $j \in \{ 1, 2, \dots, N \}$  denote the location of elements of the matrix such that the first index corresponds to the row number and the second index corresponds to the column number.

##### *Hermitian Transpose of a Matrix:*

The Hermitian transpose of a matrix  $A : \mathbb{C}^N \rightarrow \mathbb{C}^M$  is given by  $A^H : \mathbb{C}^M \rightarrow \mathbb{C}^N$  such that,

$$A = A_R + i A_I, \quad A_R, A_I : \mathbb{R}^N \rightarrow \mathbb{R}^M \quad (\text{A.2})$$

and,

$$A^H = A_R^T - i A_I^T \quad (\text{A.3})$$

*Hermitian Matrices:*

A Hermitian matrix  $A : \mathbb{C}^N \rightarrow \mathbb{C}^N$  is the matrix for which,

$$A = A^H \quad (\text{A.4})$$

*Inverse of a Square matrix:*

The inverse of a square matrix  $A : \mathbb{R}^N \rightarrow \mathbb{R}^N$  ( if it exists ) is denoted by  $A^{-1} : \mathbb{R}^N \rightarrow \mathbb{R}^N$  and is that unique matrix such that,

$$A^{-1} A = A A^{-1} = I_N \quad (\text{A.5})$$

*Norms:*

To have a notion of the magnitude of matrices, we shall use the Euclidian norm throughout this thesis. This norm is defined as follows,

*Euclidian Norm of a Vector:*

The Euclidian norm of a vector  $x \in \mathbb{R}^N$  is denoted by  $\|x\|_E$  and defined as,

$$\|x\|_E = \left( \sum_{i=1}^N x_i^2 \right)^{\frac{1}{2}} \quad \text{where } x_i, i \in \{1, 2, \dots, N\} \text{ is the } i^{\text{th}} \text{ element of vector } x \quad (\text{A.6})$$

*Euclidian ( Frobenius ) Norm of a Matrix:*

The Euclidian ( Frobenius ) norm of a matrix  $A : \mathbb{R}^N \rightarrow \mathbb{R}^M$  is denoted by  $\| A \|_E$  or  $\| A \|_F$  and is defined as,

$$\| A \|_E = \| A \|_F = \left( \sum_{i=1}^M \sum_{j=1}^N A_{ij}^2 \right)^{\frac{1}{2}} \quad (\text{A.7})$$

where  $A_{ij}$ , ( $i \in \{ 1, 2, \dots, M \}; j \in \{ 1, 2, \dots, N \}$ ) is the  $(i,j)^{\text{th}}$  element of matrix  $A$ .

The Euclidian Norm of a matrix can also be written in the following forms,

$$\begin{aligned} \| A \|_E = \| A \|_F &= \left( \sum_{i=1}^M \| A u_i \|_E^2 \right)^{\frac{1}{2}} \\ &= \sqrt{\text{tr} ( A^T A )} \end{aligned} \quad (\text{A.8})$$

where  $u_i$ ,  $i \in \{ 1, 2, \dots, M \}$  is any orthonormal basis and  $\text{tr} ( A^T A )$  denotes the trace of  $(A^T A)$  which is equivalent to the sum of all its diagonal elements.

In general, all matrix norms satisfy the following four conditions:

For  $A, B : \mathbb{R}^N \rightarrow \mathbb{R}^M$  and  $C : \mathbb{R}^M \rightarrow \mathbb{R}^N$ ,

1.  $\| A \| \geq 0$  and  $\| A \| = 0$  iff  $A = 0$
2.  $\| k A \| = |k| \| A \|$  where  $k$  is any scalar
3.  $\| A + B \| \leq \| A \| + \| B \|$
4.  $\| A C \| \leq \| A \| \| C \|$

### ***Linear Dependence / Independence:***

A set of vectors  $s_i \in \mathbb{R}^N$ ,  $i \in \{ 1, 2, \dots, N \}$  is said to be a linearly dependent set if there exist numbers  $\lambda_i$ ,  $i \in \{ 1, 2, \dots, N \}$ , not all zero, such that,

$$\sum_{i=1}^N \lambda_i s_i = 0 \quad (\text{A.9})$$

If the set is not linearly dependent, then it is said to be linearly independent.

***Unitary / Orthogonal Matrices:***

A matrix  $U : \mathbb{C}^N \rightarrow \mathbb{C}^N$  is said to be unitary if,

$$U^H U = U U^H = I_N \quad (\text{A.10})$$

A special case of unitary matrices is  $V : \mathbb{R}^N \rightarrow \mathbb{R}^N$  in which case,

$$V^T V = V V^T = I_N \quad (\text{A.11})$$

Matrices falling under this special case are called orthogonal matrices.

***Conjugacy / Orthogonality / Orthonormality:***

Any set of linearly independent vectors,

$$v_i : v_i \in \mathbb{R}^N, \quad i \in \{ 1, 2, \dots, M \} \quad (M \leq N)$$

is said to be mutually conjugate about a positive definite, full rank matrix  $Q : \mathbb{R}^N \rightarrow \mathbb{R}^N$  such that the product,

$$v_i^T Q v_j = \begin{cases} a > 0 & \text{if } i=j \\ 0 & \text{if } i \neq j \end{cases} \quad (\text{A.12})$$

If  $Q = I_N$  then the set is a mutually orthogonal set of vectors. If in addition  $a = 1$ , then the set is mutually orthonormal ( i.e. For an orthonormal set of vectors,  $\| v_i \|_E = 1$  ).

***Singular Values of a Matrix:***

If  $A : \mathbb{C}^N \rightarrow \mathbb{C}^M$ , then the strictly positive square roots  $\sigma_i$  of the nonzero eigenvalues of  $A^H A$  ( or  $A A^H$  ) are called the singular values of matrix  $A$ .

***Rank of a Matrix:***

Matrix  $A : \mathbb{C}^N \rightarrow \mathbb{C}^M$  has rank  $k$  if it has  $k$  singular values.

***Singular Value Decomposition:***

If  $A : \mathbb{C}^N \rightarrow \mathbb{C}^M$  has rank  $k$  and its singular values are denoted by  $\sigma_1 \sigma_2 \dots \sigma_k > 0$ , then there exist two unitary matrices,

$$U = [ u_1, u_2, \dots, u_M ] : \mathbb{C}^M \rightarrow \mathbb{C}^M \quad (\text{A.13})$$

and,

$$V = [ v_1, v_2, \dots, v_N ] : \mathbb{C}^N \rightarrow \mathbb{C}^N \quad (\text{A.14})$$

such that,

$$\Sigma = U^H A V \text{ and } A = U \Sigma V^H \quad (\text{A.15})$$

where,

$$\Sigma = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix} : \mathbb{C}^N \rightarrow \mathbb{C}^M \quad (\text{A.16})$$

and,

$$D_{ij} = \begin{cases} \sigma_i > 0 & \text{for } i=j \\ 0 & \text{for } i \neq j \end{cases} \quad (\text{A.17})$$

Then,

$$A = U \Sigma V^H \quad (\text{A.18})$$

is the singular value decomposition of matrix  $A$ , where,

$$\text{for } 1 \leq i \leq k, u_i = \frac{A v_i}{\sigma_i} \text{ and } v_i = \frac{A^H u_i}{\sigma_i} \text{ are eigenvectors of } A A^H \text{ and } A^H A \text{ respectively,}$$

associated with the  $k$  eigenvalues  $\sigma_i^2 > 0$  and the vectors  $u_i, k+1 \leq i \leq M$  and  $v_i, k+1 \leq i \leq N$  are the eigenvectors associated with the zero eigenvalues. If  $A$  is real, then  $U$  and  $V$  will also be real and are therefore orthogonal matrices.

***Pseudo-Inverse ( Moore-Penrose Generalized Inverse ):***

If  $A : C^N \rightarrow C^M$  and  $A^\dagger : C^M \rightarrow C^N$ , then  $A^\dagger$  is the pseudo-inverse (Moore-Penrose generalized inverse of  $A$  iff,

1.  $A A^\dagger A = A$
2.  $A^\dagger A A^\dagger = A^\dagger$
3.  $A A^\dagger$  and  $A^\dagger A$  are hermitian

Furthermore, if the singular value decomposition of  $A$  is given by,

$$A = U \Sigma V^H \quad (\text{A.19})$$

then the pseudo-inverse of  $A$ ,  $A^\dagger$  is given by,

$$A^\dagger = V \Sigma^\dagger U^H \quad (\text{A.20})$$

where,

$$\Sigma^\dagger = \begin{bmatrix} E & 0 \\ 0 & 0 \end{bmatrix} : C^M \rightarrow C^N \quad (\text{A.21})$$

$E$  is the  $k \times k$  diagonal matrix such that,

$$E_{ij} = \begin{cases} \frac{1}{\sigma_i} & \text{for } i=j \\ 0 & \text{for } i \neq j \end{cases} \quad (\text{A.22})$$

and  $k$  is the rank of  $A$ .

***Gram-Schmidt Orthogonalization:***

*Ordinary Gram-Schmidt Orthogonalization Procedure:*

Suppose,  $v_i : v_i \in \mathbb{R}^N, i \in \{1, 2, \dots, M\}$ ,  $M^2N$  are a set of unit vectors. Then, the following is the Gram-Schmidt procedure which generates the set of vectors  $z_i, i \in \{1, 2, \dots, M\}$  which form an orthonormal set spanning the same space as vectors  $v_i$ .

$$u_1 = v_1 \quad (\text{A.23})$$

$$u_i = v_i - \sum_{j=1}^{i-1} (v_i^T z_j) z_j \quad i \in \{2, 3, \dots, M\} \quad (\text{A.24})$$

and,

$$z_i = \frac{u_i}{\|u_i\|_E} \quad i \in \{1, 2, \dots, M\} \quad (\text{A.25})$$

*Modified (Numerically Accurate) Gram-Schmidt Orthogonalization:*

The following pseudo-code presents a modified Gram-Schmidt orthogonalization method which theoretically gives the same set of vectors as the original procedure but it is more accurate in actual numerical implementation.

$$1. u_1 = v_1 \quad (\text{A.26})$$

$$z_1 = \frac{u_1}{\|u_1\|_E} \quad (\text{A.27})$$

2. For  $i = 2, 3, \dots, M$

$$\text{Compute } v_i^{(1)} = v_i - (v_i^T z_1) z_1$$

3. For  $j = 2, 3, \dots, M$

$$u_j = v_j^{(j-1)} \tag{A.28}$$

$$z_j = \frac{u_j}{\|u_j\|_E} \tag{A.29}$$

For  $i = j+1, \dots, M$

$$v_i^{(j)} = v_i^{(j-1)} - (v_i^{(j-1)T} z_j) z_j \tag{A.30}$$

***Sherman-Morrison Inversion Formula:***

If  $G_{k+1}, G_k : \mathbb{R}^N \rightarrow \mathbb{R}^N$ , then the rank  $M$  ( $M^2N$ ) update to  $G_k$  for obtaining  $G_{k+1}$  is,

$$G_{k+1} = G_k + R S^T \tag{A.31}$$

where  $R, T : \mathbb{R}^M \rightarrow \mathbb{R}^N$  and  $S : \mathbb{R}^M \rightarrow \mathbb{R}^M$ , then the inverse of  $G_{k+1}$  is given by the following,

$$G_{k+1}^{-1} = G_k^{-1} - G_k^{-1} R U^{-1} T^T G_k^{-1} \tag{A.32}$$

where,

$$U = S^{-1} + T^T G_k^{-1} R \tag{A.33}$$

***Positive Definiteness:***

Let  $s$  be any vector such that  $s \in \mathbb{R}^N$ . A matrix  $G : \mathbb{R}^N \rightarrow \mathbb{R}^N$  is said to be positive definite if,

$$s^T G s > 0 \quad \forall s \neq 0$$

## A.2. Vector Representation Under a Set of Normal Conjugate Directions:

*Theorem A.1:*

Since conjugate directions are linearly independent, any vector  $v \in \mathbb{R}^N$  can be represented in terms of a set of directions  $s_i$ ,  $i \in \{ 0, 1, \dots, N-1 \}$  conjugate about a positive definite full rank matrix  $G : \mathbb{R}^N \rightarrow \mathbb{R}^N$  as follows,

$$v = \sum_{i=0}^{N-1} \lambda_i s_i \quad (\text{A.34})$$

where,

$$\lambda_i = \frac{s_i^T G v}{s_i^T G s_i} \quad (\text{A.35})$$

Furthermore, there always exists a full set of  $N$  directions  $s_i$  conjugate about  $G$  since the eigenvectors of  $G$  form such a set.

*Theorem A.2:*

Consider the matrix,

$$H = \sum_{i=0}^{N-1} \frac{s_i s_i^T}{s_i^T G s_i} \quad (\text{A.36})$$

where  $s_i \in \mathbb{R}^N$ ,  $i \in \{0, 1, \dots, N-1\}$  are a set of directions mutually conjugate about the positive definite full rank matrix  $G : \mathbb{R}^N \rightarrow \mathbb{R}^N$ . Post multiplication of H by  $Gs_k$  gives,

$$\begin{aligned} \sum_{i=0}^{N-1} \frac{1}{s_i^T G s_i} s_i s_i^T G s_k &= \frac{s_k s_k^T G s_k}{s_k^T G s_k} \\ &= s_k \quad \text{since } s_i^T G s_k = \begin{cases} = 1 & \text{when } i=k \\ = 0 & \text{when } i \neq k \end{cases} \end{aligned} \quad (\text{A.37})$$

Therefore, H is the representation of the inverse of G,

$$G^{-1} = H = \sum_{i=0}^{N-1} \frac{s_i s_i^T}{s_i^T G s_i} \quad (\text{A.38})$$

# **Appendix B**

## **Abbreviations**

<b><u>Abbreviation</u></b>	<b><u>Description</u></b>
BFGS	Broyden-Fletcher-Goldfarb-Shanno
BR.	Broyden
C.P.	Continuous Partan
DFP	Davidon-Fletcher-Powell
Ex/LS	Exact Line Search
FLOP	Floating Point Operation
F-R	Fletcher-Reeves
GRI	Greenstadt I
GRII	Greenstadt II
Inex/LS	Inexact Line Search
LRLS	Learning Recursive Least Squares
NN	Neural Network
O	Oren
PNR	Projected-Newton-Raphson
PRII	Pearson II
PRIII	Pearson III
Pw11	Powell 1
Pw12	Powell 2
RLS	Recursive Least Squares
SD	Steepest Descent
SSVM	Self-Scaling Variable Metric
XOR	Exclusive-OR