NEURAL NETWORK LEARNING ALGORITHMS

BASED ON LIMITED MEMORY

QUASI-NEWTON METHODS

By

YIJUN HUANG

Bachelor of Science

Shanghai Institute of Education

Shanghai, China

1987

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1997

NEURAL NETWORK LEARNING ALGORITHMS

BASED ON LIMITED MEMORY

QUASI-NEWTON METHODS

Thesis Approved:

J Chandler

Thesis Adviser

D. E. Heck

H. Lu

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my deep appreciation to my thesis advisor Dr. J. P. Chandler, for his guidance in choosing the topic of this thesis, his skillful supervision, constructive suggestions and unceasing encouragement in solving problems that I have encountered throughout my work.

I extend my sincere gratitude to my other committee members Dr. G. E. Hedrick and Dr. H. Lu as well. Their patience and guidance in reviewing this thesis were invaluable.

I am forever indebted to my husband Debao Chen and our daughter Xuejing Chen, whose wholehearted support made this work possible.

Throughout my life, my mother Zhuang Li had given me immeasurable love and encouragement. She and my father were extremely proud of my entrance into graduate school and both hoped to attend my Hooding Convocation. Unfortunately, my mother passed away last November at age eighty-four and consequently was unable to witness the completion of my degree.

To her memory I dedicate this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

Our brains are huge biological neural networks made up of individual neurons that are extensively interconnected with many synapses. Today's artificial neural networks have arisen from attempts to model this biological structure with computer software.

Generally speaking, an artificial neural networks is an information processing system. It consists of a large number of artificial neurons which are interconnected together in order to solve a desired computational task.

Currently, neural networks are used in many fields, such as aerospace, automotive, electronics, financial, insurance, medical, transportation, and so on. For details of such applications, the readers are referred to [15,21,23,28]. In general, the most important and the most successful applications of neural networks can be classified as function approximation and pattern classification [21].

One important aspect of such research is focused on standard numerical optimization techniques. In neural network design, one usually uses the sum of squares of other nonlinear functions as the error function, and the objective of a learning algorithm for a neural network is to minimize the error function, or objective function. The minimization of this type of objective function is referred to as nonlinear least squares, which is a very popular category of optimization problem [44,47]. When the form of the objective function is known, it is often possible to design more efficient algorithms. The Gauss-Newton method, modified Gauss-Newton methods, and Marquardt methods are designed specifically for nonlinear least squares problems [47]. All of those methods are very efficient for some problems. Other useful methods of

general numerical optimization applied to neural networks are Newton methods and variations of them [33,38,48,50], the conjugate gradient methods [12,18], and the quasi-Newton methods [20]. In general, the quasi-Newton methods are among the most efficient known general optimization methods. However, the storage requirements increase as the square of the number of variables of the objective function. Obviously, they are not suitable for very large neural networks. The limited memory quasi-Newton methods have been discussed in [28,29,35], which limits the memory requirements in the process of optimization. Liu and Nocedal proposed an efficient limited memory quasi-Newton algorithm [29].

The main purpose of this thesis is to design a supervised learning algorithm based on a limited memory quasi-Newton method to train fully-connected feed-forward neural networks.

To evaluate the efficiency of the limited memory quasi-Newton method, we test this algorithm on various functions from [5,39]. We have to mention that the primary purpose of the limited memory quasi-Newton method is to minimize high dimension functions, especially functions with more than one thousand independent variables. The dimensions of the test functions in [5,39] range from two to twenty. We see that the limited memory quasi-Newton method are very fast and robust for low dimension functions.

This thesis is organized into seven chapters.

In Chapter 2, we briefly explain fully-connected feed-forward neural networks and formulate the learning algorithm as an optimization problem. We only consider fully-

connected feed-forward neural networks in this thesis. Chapter 3 gives a brief review of unconstrained optimization. We concentrate our discussion on gradient methods, especially the Newton-like methods. Chapter 4 describes some details of the limited memory quasi-Newton method, and how it can be used in a neural network learning algorithm. In Chapter 5, we test the limited memory quasi-Newton method algorithm on various test functions from [5,39]. Chapter 6 explains how to implement and test our learning algorithm. The conclusions are given in Chapter 7.

In Proben1 [43], Prechelt collected a set of problems for neural network learning in the realm of pattern classification and function approximation. Proben1 contains 15 data sets from 12 different domains, and all of the data sets consist of real world data. We choose some data sets from Proben1 to test our algorithm. The rules and conventions of Proben1 are followed strictly in our implementation.

2. Neural Networks

The main purpose of this chapter is to formulate the learning problem within the context of nonlinear optimization.

Artificial neural networks are much simpler than the biological neural networks. However, there are at least two similarities between them. First, both networks are highly-interconnected simple computational devices. Second, the connections between neurons determine the functions of the network. In the remainder of this thesis, "neural network" always refers to an artificial neural network or ANN.

In the literature, neural network architectures are characterized into three basic categories: Feed-forward, Feed-back, and Self-organizing neural networks. Although there are some essential differences among these categories, the common characterization of neural networks is an ability to learn. Learning is the process by which a neural system acquires the ability to carry out certain tasks by adjusting its internal parameters according to some learning scheme [24].

2.1 Feed-forward Neural Networks

In this thesis, we concentrate on one particular neural network category: fully-connected, feed-forward neural networks.

A feed-forward neural network can be viewed as a system transforming a set of input patterns into a set of output patterns. It consists of an input layer, one or more hidden layers, and an output layer of neurons. Layers are connected by sets of weights. A neuron in any layer, except for the input layer, of the network is connected to all the neurons in the previous layer. A neural network connected in this way is referred to as

fully-connected. The input signal propagates through the network in a forward direction, from left to right, on a layer-by-layer basis. Such neural architectures are called *feed-forward* since the output of each layer feeds the next layer of units. The network can be trained to provide a desired response to a given input. The training of feed-forward neural network often requires the existence of a set of input and output patterns, called the *training set.* This kind of learning is called *supervised learning* [24].

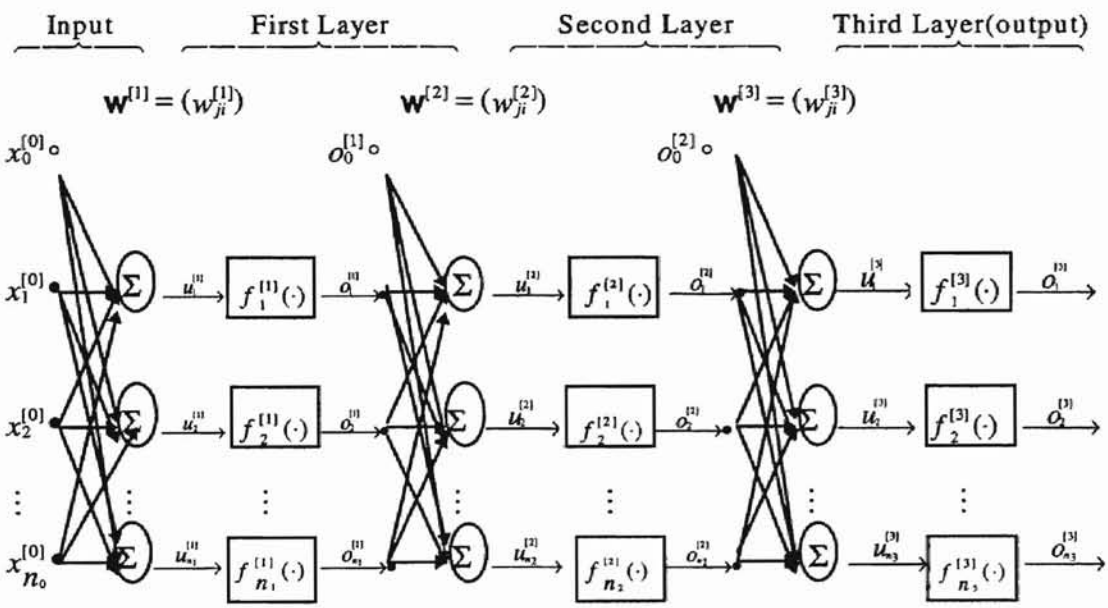

Figure 2.1: Architecture of a Three-Layer Neural Network

2.2 Problem Formulation

Figure 2.1 is a multilayer feed-forward neural network consisting of three layers, made of a number of neurons. The network is fully interconnected from one layer to the next layer and the connections are represented by lines which are characterized by their weights. Based on the weights of all the input connections, each neuron computes a weighted sum of all the inputs and evaluates a nonlinear activation function using the sum as the argument of the function. In our algorithm, we choose the following widely used sigmoid function as the activation function:

$$f(x) = \frac{1}{1+e^{-x}} .$$

The result of this function evaluation is the output of the neuron. The objective of a learning algorithm is to find the optimum weights, to minimize the discrepancy between the outputs of a neural network at output layer and the desired outputs, corresponding to a set of specific inputs. The input-output patterns used for the learning are termed "examples" or "exemplars".

We define:

$l \in [1, L]$ where L is the number of layers in the network.

$n_l \in [1, N_l]$ where N_l is the number of neurons in layer l.

$p \in [1, P]$ where P is the number of examples in the data set.

w^l_{nm} as the weight of the m-th input to neuron n in layer l.

o^l_{pn} as the output of neuron n in layer l for example p.

t^L_{pn} as the desired output of neuron n in layer L for example p.

Based on the above definitions, if we define our objective function as the sum of the squares of output errors in the output layer (layer L) of a neural network over a set of examples, then

$$E(w) = \sum_{p=1}^{P} \sum_{n_L=1}^{N_L} (o_{pn_L}^L - t_{pn_L}^L)^2 .$$

In the above optimization model, the learning corresponds to minimizing $E(w)$ with regard to the weight vector w. Thus, the training of a feed-forward neural network simply turns into a numerical optimization problem.

In our actual implementation, we use the squared error percentage which differs by a constant factor from the above error function. We explain the reason that we choose the squared error percentage in Chapter 6. However, this slight difference is not essential from the viewpoint of neural network training.

Our training is batch rather than incremental, that is the weights are updated only after the entire training set has been presented

3. Unconstrained Optimization

3.1 General Optimization

The nonlinear optimization problem can be stated as follows:

given a set $D \subseteq \Re^n$, and given a real function f: $D \rightarrow \Re$, find

$$\min\{ f(x): \ x \in D\}$$

and the vector $x^* \in D$ where the minimum is achieved. Here f refers to the objective function, and D is called the feasible region. If $D = \Re^n$, the optimization problem is said to be *unconstrained*. In this case, nothing else needs to be said about how to specify D, since every vector in $\Re^n$ is feasible [12,31]. The neural network learning algorithm can be treated within the context of nonlinear unconstrained optimization problems. This chapter gives a brief review of unconstrained optimization. Despite the diversity of both algorithms and problems, all of the algorithms that we discuss in any detail in this chapter and in Chapter 4 are all iterative processes which fit into the same general framework:

General Optimization Algorithm:

Specify some initial guess for the solution vector x_0.

For $k = 0, 1, 2, ...$

If x_k is optimal, stop.

Determine an improved estimate of the solution: $x_{k+1} = x_k + \alpha_k p_k$.

Here the vector p_k represents a search direction and the positive scalar α_k is a step length that determines the point x_{k+1}. For our purpose, the word "optimize" means finding the value of x that minimizes f. For an unconstrained problem of this form, we require that

the search direction p_k be a descent direction for the function f at the point x_k. This means that for a small enough step taken along p_k the function value is guaranteed to decrease, in other words,

$$f(x_k + \alpha_k p_k) < f(x_k), \qquad \text{for } 0 < \alpha_k < \varepsilon$$

for some $\varepsilon > 0$. With p_k is available, we would ideally like to determine the step length α_k so as to minimize the function in that direction:

$$\underset{\alpha_k > 0}{\text{minimize}}\, f(x_k + \alpha_k p_k).$$

This is a problem only involving one variable, the parameter α_k. The restriction $\alpha_k > 0$ is imposed because p_k is a descent direction. The calculation of α_k is called a *line search* since it corresponds to a search along the line $x_k + \alpha_k p_k$ defined by α_k. However, it is not always the best to minimize $f(x_k + \alpha_k p_k)$ with regard to α_k. We discuss the line search in Chapter 4. Here we first discuss how to choose the search direction p_k, although in optimization algorithms the choices of p_k and α_k cannot be separated in general.

3.2 Gradient Methods

There are various methods to determine the direction vector p_k. The most popular methods are gradient methods, which use first, and sometimes second, derivatives of the objective function to compute p_k. The derivatives may be available analytically or perhaps are approximated in some way. When we discuss their properties in the following, we assume that the objective function has continuous second derivatives, whether or not these are explicitly available. As we mentioned in previous

chapter, we choose the sigmoid function as the activation function, so that the error function chosen in our training algorithms is infinitely differentiable, and the assumption of differentiability is always satisfied. In this thesis we design a learning algorithm based on a limited memory quasi-Newton method, so we concentrate on Newton-like methods in this chapter. For other optimization methods the readers are referred to any good book on nonlinear optimization, for example [12,47].

For convenience of further discussion, we first give some terminology and notations. Let f be a smooth, nonlinear function from $\Re^n$ to $\Re$, the gradient of f is defined as:

$$g(x) = \nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n} \right)^T$$

The Hessian matrix of f is defined as:

$$B(x) = \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 (x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

By definition, a matrix A is positive definite if

$$x^T A x > 0$$

for any vector $x \in \Re^n$, $x \neq 0$. We require that the Hessian matrix B be positive definite at the minimum of f. Then the inverse of the Hessian matrix exists. We denote the inverse matrix of the Hessian matrix of f as H.

One may choose a search direction, in which the objective function decreases fastest. For this purpose we choose $p_k = -g_k$, which is the opposite direction of the gradient vector. The corresponding method is referred to as the steepest descent method. It seems that steepest descent method is not a very good method in neural network learning since it is too slow in converging. Wang [51] compared his damped learning algorithm and the steepest descent method. Another popular method is Newton's method, in which one chooses the search direction as

$$p_k = -H_k g_k \tag{3.1}$$

where $H_k = H(x_k)$, $g_k = g(x_k)$ and H is the inverse Hessian matrix. Newton's method is not always feasible, since the inverse Hessian matrix may not exist. Even if H_k is invertible, it may not be positive definite, hence p_k may not be a descent direction.

To avoid computing any second derivatives of f, the Gauss-Newton method and Levenberg-Marquardt method [27,30] were introduced. These methods need a particular form of the objective function. That is, the objective function is a sum of squares of some nonlinear functions. As we mentioned before, one often chooses such functions as the error function in neural network training, hence, the Levenberg-Marquardt method is usually considered as a good way to train neural networks [22,51]. However, we do not discuss these methods in detail in this thesis; their storage requirements are too great for very large networks.

3.3 Quasi-Newton Methods

Let us return to the general form of the Newton method with the search vector calculated at each iteration as in (3.1),

$$p_k = -H_k g_k$$

Here H_k is the precise inverse of the Hessian matrix at x_k. As was mentioned previously, H_k may not exist or may not be positive definite. To avoid such problems we use a BFGS update formula [47] to replace the precise inverse of Hessian matrix with a positive definite matrix, H_k, which is in some way an approximation of the inverse Hessian matrix.

Let

$$s_k = x_{k+1} - x_k, \qquad y_k = g_{k+1} - g_k.$$

We define

$$H_{k+1} = (I - s_k y_k^T / y_k^T s_k) H_k (I - y_k s_k^T / y_k^T s_k) + s_k s_k^T / y_k^T s_k. \tag{3.2}$$

Let

$$\rho_k = I / y_k^T s_k, \qquad v_k = I - \rho_k y_k s_k^T.$$

Then (3.2) can be expressed as

$$H_{k+1} = v_k^T H_k v_k + \rho_k s_k s_k^T,$$

where we store each H_k explicitly. It is easy to verify that if H_k is positive definite and $y_k^T s_k > 0$, then H_{k+1} is positive definite also. We assume that $y_k^T s_k > 0$ for all k. In

gradient-related methods this can always be done, provided the line search is sufficiently accurate. The search direction p_k is obtained from the matrix-vector product:

$$p_k = -H_k g_k$$

The quasi-Newton methods use an iterative process to approximate the inverse Hessian matrix, so that no explicit expression for the second derivatives is needed for carrying out a Newton-like search. Although the quasi-Newton algorithms require slightly more operations to calculate an iterate, and they require somewhat more storage than the conjugate gradient algorithms do, but in almost all cases, these additional costs are outweighed by the advantage of superior speed of convergence.

At first glance, quasi-Newton methods may seem unsuitable for large problems because the approximate inverse Hessian matrices are generally dense. In the next chapter, we discuss ways to cut down on storage in order to create limited memory quasi-Newton methods for large problems.

4. Limited Memory Quasi-Newton Methods

In Chapter 3 we gave a brief review of unconstrained optimization. In particular, we discussed a quasi-Newton method. When we use a quasi-Newton method, we construct a sequence of matrices which in some way approximate the inverse Hessian matrix instead of storing the precise inverse Hessian matrix in each iteration. In such a way we avoid using the second derivatives of the objective function so that we save a substantial fraction of the computing time. However, it is necessary to have $O(n^2)$ storage locations for each H_k. In neural network training, in some cases, for example, one may need to use a large number of weights. Sometimes the network has many thousands of weights. In such a case, storage becomes an issue since it will be impossible to retain the matrix in the high speed storage of a computer.

In this chapter, we describe an algorithm which uses a limited amount of storage and where the quasi-Newton matrix is updated continually. At every step the oldest information contained in the matrix is updated and replaced by the newest information. Recall that the BFGS update of H is:

$$H_{k+1} = v_k^T H_k v_k + \rho_k s_k s_k^T,$$

where

$$\rho_k = I / y_k^T s_k, \qquad v_k = I - \rho_k y_k s_k^T.$$

Let H_0 be a given positive definite matrix. Then the above BFGS update gives:

$$\begin{aligned} H_1 &= v_0^T H_0 v_0 + \rho_0 s_0 s_0^T \\ H_2 &= v_1^T H_1 v_1 + \rho_1 s_1 s_1^T \\ &= v_1^T v_0^T H_0 v_0 v_1 + v_1^T \rho_0 s_0 s_0^T v_1 + \rho_1 s_1 s_1^T \end{aligned}$$

$$
\begin{aligned}
H_3 &= v_2^T H_2 v_2 + \rho_2 s_2 s_2^T \\
&= v_2^T v_1^T v_0^T H_0 v_0 v_1 v_2 + v_2^T v_1^T \rho_0 s_0 s_0^T v_1 v_2 + v_2^T \rho_1 s_1 s_1^T v_2 + \rho_2 s_2 s_2^T \\
&\cdots \\
H_{k+1} &= v_k^T v_{k-1}^T \cdots v_0^T H_0 v_0 \cdots v_{k-1} v_k \\
&\quad + v_k^T \cdots v_1^T \rho_0 s_0 s_0^T \cdots v_k \\
&\quad \cdots \\
&\quad + v_k^T v_{k-1}^T \rho_{k-2} s_{k-2} s_{k-2}^T v_{k-1} v_k \\
&\quad + v_k^T \rho_{k-1} s_{k-1} s_{k-1}^T v_k \\
&\quad + \rho_k s_k s_k^T
\end{aligned}
$$

Instead of forming H_k explicitly, now we store previous values of y_j and s_j separately. Here m is a given integer that represents the maximum number of correction matrices that can be stored. Normally we choose $3 \le m \le 7$.

The following algorithm was given by Liu and Nocedal [28].

L-BFGS Algorithm:

Step 1. Choose x_0, m, $0 < \beta' < \frac{1}{2}$, $\beta' < \beta < 1$, and a symmetric and positive definite starting matrix H_0 (normally we choose a scaled diagonal matrix or I as the H_0). Set $k = 0$.

Step 2. Compute

$$
\begin{aligned}
p_k &= -H_k g_k, \\
x_{k+1} &= x_k + \alpha_k p_k,
\end{aligned}
$$

where α_k satisfies the Wolfe conditions [52]:

$$
\begin{aligned}
& f(x_k + \alpha_k p_k) \le f(x_k) + \beta' \alpha_k g_k^T p_k, \\
& |g(x_k + \alpha_k p_k)^T p_k| \le -\beta g_k^T p_k
\end{aligned}
$$

(We always try the steplength $\alpha_k = 1$ first)

Step 3. Let $\hat{m} = \min(k, m\text{-}1)$. Update H_0 $\hat{m}+1$ times using the pairs $(y_j, s_j)_{j=k-\hat{m}}^{k}$, i.e. let

$$
\begin{aligned}
H_k &= (v_k^T \cdots v_{k-\hat{m}}^T) H_0 (v_{k-\hat{m}} \cdots v_k) \\
&\quad + \rho_{k-\hat{m}} (v_k^T \cdots v_{k-\hat{m}+1}^T) s_{k-\hat{m}} s_{k-\hat{m}}^T (v_{k-\hat{m}+1} \cdots v_k)
\end{aligned}
$$

$$+\rho_{k-\hat{m}+1}(v_k^T \cdots v_{k-\hat{m}+2}^T)s_{k-\hat{m}+1}s_{k-\hat{m}+1}^T(v_{k-\hat{m}+2} \cdots v_k)$$
$$\vdots$$
$$+\rho_k s_k s_k^T.$$

(We do not calculate and store the H_k in this step, instead, we use the above formulas to calculate the direction vector $p_k = -H_k g_k$, directly).

Step 4. Set $k := k + 1$, go to Step 2.

The stopping criterion of L-BFGS is:

$$\|g_k\| < \varepsilon \times \max(1, \|x_k\|),$$

where ε is a small positive number supplied by the user.

The L-BFGS algorithm is almost identical in its implementation to the well-known BFGS method. The only differences that are the amount of storage required by the algorithm (and thus the cost of the iteration) can be controlled by the user and the later approximations H_k to the inverse Hessian deviate from the BFGS method. The user specifies the number m of BFGS corrections that are to be kept, and provides a sparse symmetric and positive definite matrix H_0 which approximates the inverse Hessian of f. During the first m iterations the method is the same as the BFGS method. When the available storage is used up, i.e., $k > m$, since the BFGS corrections are stored separately, we can delete the oldest one to make space for the new one. All subsequent iterations are of this form: one correction is deleted and a new one inserted. Hence, it requires only $O(mn)$ storage locations ($m << n$), contrast with usual BFGS algorithm which requires $O(n^2)$ storage locations as we discussed before. If there is no previous information, one can choose the identity matrix I or a scaled diagonal matrix as the H_0.

It is also known that simple scaling of the variables can improve the performance of quasi-Newton methods on small problems. For large problems, scaling becomes much more important. Several scaling methods for the matrix H_0 were introduced in [28]. We use the following strategy.

In Step 3 of the L-BFGS algorithm, instead of using a fixed H_0, we use $H_0^{(k)}$ which is a scale of the identity matrix I:

$$H_0^{(k)} = \gamma_k I$$

where

$$\gamma_0 = 1/\|y_0\|^2$$
$$\gamma_k = y_{k-1}^T s_{k-1} / \|y_{k-1}\|^2, \qquad k = 1, 2, 3, \cdots.$$

It was showed [28] that this is a simple and effective way of introducing a scale in the algorithm.

Since we do not store H_k explicitly, the product Hg must be computed. The following recursion performs this computation efficiently [35]. It is essentially the same as the formula for the usual BFGS method.

In the following algorithm, m is the number of corrections stored, and Iter is the iteration number.

Recursive Formula to Compute *Hg*:

1) If Iter $\leq m$, Set Incr = 0; Bound = Iter;
   else Set Incr = Iter - m; Bound = m;

2) $q_{\text{Bound}} = g_{\text{iter}}$;

3) For i = (Bound -1), …, 0
   $j = i + \text{Incr}$;
   $\alpha_i = \rho_j s_j^T q_{i+1}$; (store α_i)

$q_i = q_{i+1} - \alpha_i y_j;$

4) $\gamma_0 = H_0 q_0;$

5) For $i = 0, 1, \ldots,$ (Bound-1)
$j = i + \text{Incr};$
$\beta_i = \rho_j y_j^T r_i;$
$r_{i+1} = r_i + s_j(\alpha_i - \beta_i);$

This formula requires at most $4nm + 2m + n$ multiplications and $4nm + m$ additions.

At the k-th iteration of the L-BFGS algorithm, we need to find a positive scalar α_k as a step length, to determine a new point x_{k+1} that is a minimum in the direction p_k or that gives a sufficient reduction in function value. This process is called a *line search*, which is a univariate problem

$$\underset{\alpha>0}{\text{minimize}}\, F(\alpha) \equiv f(x_k + \alpha p_k).$$

As we mentioned in section 3.1, it is not always best to minimize $f(x_k + \alpha_k p_k)$ with regard to α_k Instead, we find an α_k that satisfies the conditions:

$$f(x_k + \alpha_k p_k) \le f(x_k) + \beta' \alpha_k g_k^T p_k, \tag{4.1}$$

$$|g(x_k + \alpha_k p_k)^T p_k| \le -\beta g_k^T p_k \tag{4.2}$$

We fix x_k and p_k and let

$$\Phi(\alpha) = f(x_k + \alpha p_k), \qquad \alpha \ge 0.$$

Then conditions (4.1) and (4.2) can be formulated as finding $\alpha > 0$ such that

$$\Phi(\alpha) \le \Phi(0) + \beta' \Phi'(0)\alpha \tag{4.3}$$

and

$$|\Phi'(\alpha)| \le \beta |\Phi'(0)| \tag{4.4}$$

The motivation for requiring conditions (4.3) and (4.4), or (4.1) and (4.2), in a line search method should be clear. If α is not too small, condition (4.3) forces a sufficient decrease in the function. However, this condition is not sufficient to guarantee convergence, because it allows arbitrarily small choices of $\alpha > 0$. Condition (4.4) rules out arbitrarily small choices of α and usually guarantees that α is near a local minimizer of Φ. Condition (4.4) is a curvature condition because it implies that

$$\Phi'(\alpha) - \Phi'(0) > (1-\beta)|\Phi'(0)|,$$

and thus the average curvature of Φ on $(0, \alpha)$ is positive. The curvature condition (4.4) is particularly important in a quasi-Newton method or a limited memory quasi-Newton method because it guarantees that a positive definite quasi-Newton update is possible [11,12].

As final motivation for (4.3) and (4.4), we mention that if a step satisfies these conditions, then the line search method is convergent for reasonable choices of direction [1,6,11,12,16,28,42]. In particular, in a quasi-Newton method, we choose $p_k = -H_k g_k$, and the line search method is convergent if conditions (4.3) and (4.4) are satisfied.

There are still many choices of α_k to satisfy the conditions (4.3) and (4.4). Moré and Thuente [32] designed a very efficient line search algorithm which was used by several authors, for example, Liu and Nocedal [28], O'Leary [38], and Gilbert and Nocedal [16]. It seems that the main idea is to combine quadratic and cubic interpolations to find a suitable α_k. For the detail of this search procedure and the associated convergence theory, the readers are referred to [32].

We design and implemented our neural network learning algorithm using Liu and Nocedal's L-BFGS algorithm. Moré and Thuente's line search method is also used. Our program works fine. Later on, we found Nocedal's L-BFGS FORTRAN program in the Internet [36], which works even better than ours. We modified our program. In the current version of our program, we treat Nocedal's L-BFGS as a subroutine and simply call it in our neural network learning process.

5. Comparisons among L-BFGS, PRAXIS and DFMCG

The main purpose of this thesis is to design and implement neural network learning algorithms based on limited memory quasi-Newton methods. In this chapter we compare the limited memory quasi-Newton methods and Brent's optimization method [5]. We also compare the limited memory quasi-Newton methods and one conjugate gradient method [14,31].

We have to mention that the primary purpose of the limited memory quasi-Newton method is to minimize the high dimension functions, especially functions with more than one thousand independent variables. The main concern is the storage. The dimensions of the test functions we use in this chapter range from two to twenty. For such low dimension functions, storage is not a problem at all. However, the main difference between the L-BFGS algorithm and the BFGS algorithm is how to store the inverse Hessian matrices in Step 3. The remaining of these two algorithms seems same. In fact, if the correction number m is sufficient large in L-BFGS, then L-BFGS is identical to BFGS. Hence, we still can see how good the algorithms are.

Powell [42] introduced an optimization algorithm without using the derivatives of the objective function. Brent [5] modified Powell's method and overcame some of the difficulties observed in the literature. Numerical tests suggested that Brent's proposed method is faster than Powell's original method and some other previous methods [5]. Brent [5] gave the ALGOL procedure PRAXIS to implement his algorithm. Chandler [7] gave the FORTRAN procedure PRAXIS, a direct translation of Brent's procedure.The conjugate gradient method [14,31] is another method to solve large optimization

problems. The storage complexity of the conjugate gradient method is $O(n)$, where n is the number of the variables of the cost function. We have tested a conjugate gradient FORTRAN procedure DFMCG from the IBM Scientific Subroutine Package [53].

In this chapter we mainly compare L-BFGS and PRAXIS on speed and accuracy. We also roughly compare L-BFGS and DFMCG. In Section 5.1 we compare these three algorithms on various test functions used in [5]. Section 5.2 summarizes the results of L-BFGS running on Osborne's functions [39]. In Section 5.3 we run L-BFGS on Osborne function 2 with different numbers of corrections m. Section 5.4 verifies that L-BFGS does not have the quadratic termination property.

Recall that the stopping criterion of L-BFGS is

$$\|g_k\| < \varepsilon \times \max(1, \|x_k\|).$$

Throughout this chapter, we choose $\varepsilon = 10^{-7}$, unless we mention otherwise.

5.1 Comparisons on Various Test Functions

Most of the functions tested in [5] are actually differentiable. We run Chandler's PRAXIS FORTRAN program [7], the DFMCG program [53], and Nocedal's L-BFGS program [36] on the UNIX System of the Oklahoma State University Computer Science Department, using the same test functions in [5], and compare the results. The results of the PRAXIS program are listed in Table 5.1, the results of DFMCG program are listed in Table 5.2, and the results of L-BFGS program are listed in Table 5.3.

Table 5.1 Result of PRAXIS Program

Function	n	x_0^T	n_f	$f(x)$
Rosenbrock	2	(-1.2, 1)	155	2.012E-24
Singular	4	(3, -1, 0, 1)	421	5.476E-19
Helix	3	(0.01,0.01, 0)	201	2.998E-24
Helix	3	(-1, 0, 0)	200	8.886E-25
Cube	2	(-1.2, -1)	234	1.599E-25
Beale	2	(0.1, 0.1)	80	1.595E-25
Watson	9	0^T	1869	1.400E-06
Powell	3	(0, 1, 2)	86	0.00E00
Wood	4	-(3, 1, 3, 1)	487	2.846E-21
Hilbert	10	(1,...,1)	2417	7.602E-17
Tridiag	20	0^T	941	-2.00E+1
Box	3	(0, 10, 20)	154	4.173E-25

Table 5.2 Results of DFMCG Program

Function	n	x_0^T	n_f	$f(x)$
Rosenbrock	2	(-1.2, 1)	73	4.123E-27
Singular	4	(3, -1, 0, 1)	305	2.198E-18
Helix	3	(0.01,0.01, 0)	47	5.518E-29
Cube	2	(-1.2, -1)	80	6.024E-27
Beale	2	(0.1, 0.1)	62	5.493E-1
Watson	9	0^T	713	3.479E+0
Powell	3	(0, 1, 2)	59	0.0E0
Wood	4	-(3, 1, 3, 1)	67	7.68E-26
Hilbert	10	(1,...,1)	2688	5.715E-2
Tridiag	20	0^T	111	-2.0E+1
Box	3	(0, 10, 20)	121	3.579E+0

Table 5.3 Results of L-BFGS Program

Function	n	x_0^T	n_f	$(n+1)n_f$	$f(x)$
Rosenbrock	2	(-1.2, 1)	49	147	1.947E-25
Singular	4	(3, -1, 0, 1)	76	380	7.614E-16
Helix	3	(0.01, 0.01, 0)	23	92	3.276E-19
Cubic	2	(-1.2, -1)	64	192	9.917E-16
Beale	2	(0.1, 0.1)	16	48	9.953E-17
Watson	9	0^T	1991	19910	6.527E-6
Powell	3	(0, 1, 2)	20	80	1.110E-15
Wood	4	-(3, 1, 3, 1)	122	610	2.053E-20
Hilbert	10	(1,...,1)	109	1199	1.236E-12
Tridiag	20	0^T	98	2058	-2.00E+1
Box	3	(0, 10, 20)	41	164	4.508E-14

In the above tables we use the following conventions:

n is the number of variables.

x_0^T is the starting point.

n_f is the number of function evaluations.

$f(x)$ is the approximated minimum.

In the PRAXIS program, we do not need to calculate the derivatives, while in L-BFGS the gradient must be calculated. In order to compare L-BFGS and PRAXIS, a proper weighting factor [5] must be used for the number of function evaluations in the L-BFGS program. As in [5], we define:

the weighted number of function evaluations = $(n+1)\ n_f$,

Note that the convergence criteria for PRAXIS are generally tighter than for L-BFGS, resulting in "better" minima in most cases, although not in all. It is not possible to use the convergence criterion of L-BFGS in PRAXIS, which does not have the gradient available.

The following are brief descriptions of each function and the comparison of L-BFSG and PRAXIS for each function. As in [5], to compare the speeds of the two programs, we simply compare n_f in Table 5.1 and $(n+1)\ n_f$ in Table 5.3.

1. **Rosenbrock** (Rosenbrock [45]):

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

This is a well-known function with a parabolic valley. Descent methods tend to fall into the valley and then follow it around to the minimum of 0 at $(1, 1)^T$.

The two programs perform similarly in both speed and accuracy.

2. **Singular** (Powell [40]):

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4.$$

This function is difficult to minimize, and provides a severe test of the stopping criterion, because the Hessian matrix at the minimum ($x = 0$) is doubly singular.

The function varies very slowly near 0 in the two-dimensional subspace $\{(10\lambda_1, -\lambda_1, \lambda_2, \lambda_2)^T\}$. For this function, PRAXIS is slightly slower, but slightly more accurate than L-BFGS.

3. **Helix** (Fletcher and Powell [13]):

$$f(x) = 100[(x_3 - 10\theta)^2 + (r-1)^2] + x_3^2,$$

where

$$r = (x_1^2 + x_2^2)^{1/2}$$

and

$$2\pi\theta = \left\{ \begin{array}{ll} \arctan(x_2/x_1) & \text{if } x_1 > 0, \\ \pi + \arctan(x_2/x_1) & \text{if } x_1 < 0. \end{array} \right\}$$

This function of three variables has a helical valley, and a minimum at $(1, 0, 0)^T$.

Originally, Brent used $(-1, 0, 0)^T$ as the starting point. However, since this function is not differentiable when $x_1 = 0$, L-BFGS does not work for this function when we use this starting point. Instead, we use $(0.01, 0.01, 0)^T$ as the starting point in both programs.

For this function, the situation is similar as for the **Singular** function, PRAXIS is slightly slower than L-BFGS, but more accurate than L-BFGS.

4. **Cube** (Leon [26]):

$$f(x) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2.$$

This function is similar to Rosenbrock's, and much the same remarks apply. Here the valley follows the curve $x_2 = x_1^3$.

For this function, the situation is also similar to the **Singular** function, and PRAXIS is slightly slower than L-BFGS, but more accurate than L-BFGS.

5. **Beale** (Beale [2]):

$$f(x) = \sum_{i=1}^{3} \left[c_i - x_1 (1 - x_2^i) \right]^2,$$

where $c_1 = 1.5,\ c_2 = 2.25,\ c_3 = 2.625.$ This function has a valley approaching the line $x_2 = 1,$ and has a minimum of 0 at $(3, 1/2)^T$

For this function, PRAXIS is slightly slower than L-BFGS, but more accurate than L-BFGS.

6. **Watson** (Kowalik and Osborne [25]):

$$f(x) = x_1^2 + (x_2 - x_1^2 - 1)^2 + \sum_{i=2}^{30} \left\{ \sum_{j=2}^{n} (j-1) x_j \left(\frac{i-1}{29} \right)^{j-2} - \left[\sum_{j=1}^{n} x_j \left(\frac{i-1}{29} \right)^{j-1} \right]^2 - 1 \right\}^2.$$

Here a polynomial

$$p(t) = x_1 + x_2 t + \ldots + x_n t^{n-1}$$

is fitted, by least squares, to approximate a solution of the differential equation

$$\frac{dz}{dt} = 1 + z^2,\ \mathrm{z}(0) = 0,$$

for $t \in [0, 1]$. (The exact solution is z = tan$t$.) The minimization problem is ill-conditioned, and rather difficult to solve, because of a bad choice of basis functions $\{1, t, \ldots, t^{n-1}\}$. We choose $n = 9$.

For this function, the two programs have similar accuracy, but PRAXIS is much faster than L-BFGS.

7. **Powell** (Powell [41]):

$$f(x) = 3 - \left(\frac{1}{1+(x_1-x_2)^2}\right) - \sin\left(\frac{\pi}{2}x_2x_3\right) - \exp\left\{-\left[\left(\frac{x_1+x_3}{x_2}\right)-2\right]^2\right\}$$

For a description of this function, see Powell [41].

For this function, the two programs have similar speeds, but PRAXIS is more accurate than L-BFGS.

8. **Wood** (Colville [10]):

$$\begin{aligned} f(x) = {} & 100(x_2 - x_1^2)^2 + (1-x_1)^2 + 90(x_4 - x_3^2)^2 + (1-x_3)^2 \\ & + 10.1[(x_2-1)^2 + (x_4-1)^2] + 19.8(x_2-1)(x_4-1). \end{aligned}$$

This function is rather like Rosenbrock's, but with four variables instead of two.

For this function, the two programs have similar accuracy, but PRAXIS is faster than L-BFGS.

9. **Hilbert**

$$f(x) = x^T A x,$$

where A is an n by n Hilbert matrix, i.e.,

$$a_{ij} = \frac{1}{i+j-1} \quad \text{for } 1 \le i,\ j \le n.$$

We choose $n = 10$.

For this function, PRAXIS is slower, but more accurate, than L-BFGS.

10. **Tridiag** (Gregory and Karney [19]):

$$f(x) = x^T A x - 2x_1,$$

where

$$A = \begin{bmatrix} 1 & -1 & & & & \\ -1 & 2 & -1 & & 0 & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & 0 & & \cdots & \cdots & \cdots \\ & & & & -1 & 2 \end{bmatrix}$$

This function is useful for testing the property of finite convergence on a quadratic function. The minimum $f(\mu) = -n$ occurs when μ is the first column of A^{-1}, i.e.,

$$\mu = (n, n\text{-}1, n\text{-}2, \ldots, 2, 1)^T.$$

we choose $n = 20$.

For this function, the two programs have similar accuracy, but PRAXIS is much faster than L-BFGS.

11.**Box** (Box [4]):

$$f(x) = \sum_{i=1}^{10} \left\{ \begin{array}{c} \left[\exp(-i x_1/10) - \exp(-i x_2/10)\right] \\ -x_3\left[\exp(-i/10) - \exp(-i)\right] \end{array} \right\}^2.$$

This function has minima of 0 at $(1, 10, 1)^T$ and also along the line $\{(\lambda, \lambda, 0)^T\}$.

Both programs find the first minimum, and have similar speeds. However, PRAXIS is more accurate than L-BFGS.

Summary: Brent's algorithm is considered to be a very good one. Overall, it is better than many other algorithms [5]. Our tests shows that the L-BFGS program is almost as good as PRAXIS, though for some test functions, PRAXIS is much better than L-BFGS.

PRAXIS was tuned extensively by Brent on his suite of test problems, unlike L-BFGS, which explains most of any superiority of PRAXIS.

Now we roughly compare L-BFGS and DFMCG. L-BFGS obtains satisfactory results for all eleven test functions, while DFMCG does not converge for some functions, such as those of Beale, Watson, Hilbert, and Box. For all other functions, it seems that L-BFGS is slightly faster and/or more accurate than DFMCG. We do not know why the conjugate gradient method is unstable. To answer a question posted in [54], Chandler [8] made the following comments:"At least one conjugate gradient (CG) method was shown to be unstable:[31], see page 383 in particular. No CG method has ever been shown to be stable, as far as I know. Instability means that small errors such as roundoff are magnified in each succeeding iteration, which can lead to unreliability. CG methods have great difficulty solving moderately ill-conditioned problems efficiently.

As far as I know, no CG method has solved either of the simple nonlinear least squares problems of M. R. Osborne [39] in a competitive time (fewer than 10,000 equivalent function evaluations). Any decent direct search method (such as my STEPIT

[7] or Richard Brent's PRAXIS) or quasi-Newton method will solve both of these problems much faster than this. Marquardt's method, developed specifically for least squares problems, also solves them efficiently.

If you want a low-storage method that is reasonably robust, I recommend the limited quasi-Newton method developed and programmed by Nocedal. It crunches both Osborne problems with no difficulty."

5.2 Test Using Osborne Functions

Osborne [39] studied a general method for minimizing a sum of squares which has the property that a linear least squares problem is solved at each stage, and which includes the Gauss-Newton, Levenberg, Marquardt, and Morrison methods as particular special cases. In this section we do not discuss the method which Osborne discussed in [39], but use his example functions to test L-BFGS.

The problem of minimizing a sum of squares arises naturally from the problem of determining parameters x_i, $i = 1, 2, ..., p$ in the model equation

$$y(t) = F(t, x)$$

from observations

$$y_i = y(t_i) + \varepsilon_i, \quad (i = 1, 2,...,n),$$

where the ε_i (the experimental errors) are independent, normally distributed random variables with mean zero and standard deviation σ. In the case $n > p$ the appropriate maximum likelihood analysis indicates that x should be estimated by minimizing $\|f(x)\|^2$, where

$$f_i(x) = y_i - F(t_i, x)$$

and

$$\|f(x)\|^2 = \sum_{i=1}^{n} f_i(x)^2$$

This problem will be referred to as the *model problem*, and it is stressed that we have offered a statistical justification for minimizing a sum of squares. Osborne's two test problems are classic practical nonlinear least squares problems.

1. **Osborne function 1**

In this example, the data values $\{(t_i, y_i),\ 1 \le i \le 33\}$, which are given in [36], are fitted by the model

$$F(t,x) = x_1 + x_2 \exp(-x_4 t) + x_3 \exp(-x_5 t).$$

The result of [39] is copied in the following Table 5.4.

Table 5.4 Original Result of Osborne Function 1

$\|f(x)\|^2$	x_1	x_2	x_3	x_4	x_5
0.546E-4	0.3754	1.9358	-1.4647	0.01287	0.02212

We run L-BFGS and PRAXIS using the above example and list the results in the following Table 5.5 and 5.6.

Table 5.5 L-BFGS Result of Osborne Function 1

$\|f(x)\|^2$	n_f	$(n+1)n_f$	x_1	x_2	x_3	x_4	x_5
5.465E-5	172	1032	0.3754	1.9358	-1.4647	0.01287	0.02212

Table 5.6 PRAXIS Result of Osborne Function 1

$\|f(x)\|^2$	n_f	x_1	x_2	x_3	x_4	x_5
5.465E-5	1268	0.3753	1.9203	-1.4490	0.01284	0.02186

We mention that in this section we set $\varepsilon = 10^{-5}$ as the stopping criterion in L-BFGS.

The three programs have similar accuracy. Since the number of function evaluations is not available in Table 5.4, we cannot compare the speed of the two algorithms. However, when we use L-BFGS on Osborne function 1, there are only 172 function and gradient evaluations, which is considered very fast. L-BFGS and PRAXIS have similar speed on Osborne function 1.

2. **Osborne function 2**

In this example, the model has the form

$$f(t,x) = x_1 \exp(-x_5 t) + x_2 \exp[-x_6 (t - x_9)^2] \\ + x_3 \exp[-x_7 (t - x_{10})^2] + x_4 \exp[-x_8 (t - x_{11})^2]$$

The data values $\{(t_i, y_i), \quad 1 \le i \le 65\}$ are also given in [39]. The result of [39] is copied in the Table 5.7.

Table 5.7 Original Result of Osborne Function 2

$\|f(x)\|^2$	x_1	x_2	x_3	x_4	x_5
0.0402	1.3100	0.4315	0.6336	0.5993	0.7539
x_6	x_7	x_8	x_9	x_{10}	x_{11}
0.9056	1.3651	4.8248	2.3988	4.5689	5.6754

we run the L-BFGS and PRAXIS using the above example and list the results in the Table 5.8 and Table 5.9.

Table 5.8 L-BFGS Result of Osborne Function 2

$\|f(x)\|^2$	$(n+1)n_f$	x_1	x_2	x_3	x_4	x_5
0.04014	2136	1.3100	0.4315	0.6337	0.5996	0.7543
n_f	x_6	x_7	x_8	x_9	x_{10}	x_{11}
178	0.9038	1.3666	4.8227	2.3988	4.5688	5.6753

Table 5.9 PRAXIS Result of Osborne Function 2

$\|f(x)\|^2$	x_1	x_2	x_3	x_4	x_5	
0.04014	1.3100	0.4316	0.6337	0.5994	0.7542	
n_f	x_6	x_7	x_8	x_9	x_{10}	x_{11}
857	0.9043	1.3658	4.8237	2.3987	4.5689	5.6753

Using L-BFGS, we only need 178 function and gradient evaluations to complete this problem. PRAXIS is faster than L-BFGS for this function. In addition, the final result 0.04014 of both L-BFGS and PRAXIS are better than Osborne's original result 0.0402.

5.3 Testing L-BFGS Using Different Numbers of Corrections

Previously, we set the number of corrections $m = 5$ in L-BFGS to test various functions. For a very large problem, one cannot set m too large, otherwise, it will take too much storage. Also, the larger the values of m, the more execution time for each iteration. Normally, we set $3 \leq m \leq 7$. However, the larger the m, the more information we can store. Hence, it seems that the larger the m. the smaller the number of iterations. We verify this by testing L-BFGS on Osborne function 2 with various m and list the results on Table 5.10.

Table 5.10 Results of Osborne Function 2 for Different m

m	Total Epochs	n_f	$\|f(x)\|^2$
2	344	379	4.014E-2
3	419	446	4.014E-2
4	311	345	4.014E-2
5	246	268	4.014E-2
6	227	253	4.014E-2
7	144	161	4.014E-2
8	117	132	4.014E-2
9	113	130	4.014E-2
10	83	99	4.014E-2
11	83	94	4.014E-2
12	79	91	4.014E-2
100	63	73	4.014E-2
1000	63	73	4.014E-2

We mention that in this section we set $\varepsilon = 10^{-7}$ as the stopping criterion.

From Table 5.10, we can see that when $3 \leq m \leq 13$, the larger the m, the smaller the number of iterations and the number of function evaluations. In fact, in the first m iterations, L-BFGS and BFGS are identical. Hence, if m is sufficiently large, for example, larger than the number of iterations, then executions of L-BFGS and BFGS are exactly the same.

5.4 Lack of Quadratic Termination Property

Many optimization algorithms possess a property called *quadratic termination* which means that they minimize a quadratic function exactly in a finite number of iterations [34]. For example, Newton's method and quasi-Newton methods have quadratic termination properties.

In this section, we verify that the limited memory quasi-Newton method does not have such property.

Let

$$f(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$$
$$+ 0.5x_1x_2 - 0.4x_2x_4 + 0.3x_3x_4 - 0.2x_4x_5 + 0.7x_5x_6 - 0.08x_6x_1.$$

It is not difficult to prove that this quadratic function is positive definite. We run this function using different m and list the results in Table 5.11.

Table 5.11 Results of a Quadratic Function

m	# of Iterations	n_f	$f(x)$
2	16	21	1.782E-15
3	14	19	1.677E-15
4	15	20	1.5E-15
5	14	19	2.511E-17
6	13	18	3.223E-16
7	13	18	1.805E-16
8	13	18	6.604E-17
9	12	17	1.450E-15
10	12	17	1.420E-15
11	12	17	1.421E-15
12	12	17	1.421E-15

From the above table we see that for small m, L-BFGS does not have quadratic termination in n or $(n+1)$ iterations. The authors of [28] pointed out that:"Our aim is that the limited memory method resemble BFGS as much as possible, and we disregard quadratic termination properties, which are not very meaningful, in general, for large dimensional problems."

6. Testing

In this chapter, we discuss the testing of our learning algorithm. We use the neural network data in Proben1 [43] to test our algorithm. The rules and conventions in Proben1 are followed strictly.

The scope of the Proben1 problems can be characterized as follows. All problems can be suited for supervised learning, since input and output values are separated. All examples within a problem are independent of each other. Some of the problems can be tackled by pattern classification algorithms, while others need the capability of continuous multivariate function approximation. All problems are presented as static problems in the sense that all data to learn from are present at once and do not change during learning. All problems consist of real data from real problem domains.

6.1 Some Aspects of Proben1

1 Training set, validation set, test set [43]

The data used for performing benchmarks on neural network learning algorithms must be split into at least two parts: one part on which the training is performed, called the *training data set*, and another part on which the performance of the resulting network is measured, called the *test data set*. The idea is that the performance of a network on the test set estimates its performance in real use. This means that absolutely no information about the test set examples or the test set performance of the network can be available during the training process; otherwise the benchmark is invalid.

In some cases the training data are further subdivided. Some examples are put into the actual training set, others into a so-called *validation set* [43]. The latter is used as

a pseudo test set in order to evaluate the quality of a network during training. Such an evaluation is called *cross validation* [43]. It is necessary due to the overfitting (overtraining) phenomenon: For two networks trained on the same problem, the one with larger training set error may actually be better, since the other has concentrated on peculiarities of the training set at the cost of losing much of the regularity needed for good generalization. This is a problem in particular when not very many training examples are available, or when too large a network is used.

A popular and very powerful form of cross validation used in neural networks is *early stopping*: Training proceeds not until a minimum of the error on the training set is reached, but only until a minimum of the error on the validation set is reached during training. Training is stopped at this point and the current network state is the result of the training run.

The sizes of the training, validation, and test sets in all Proben1 data files are 50%, 25%, and 25% of all examples, respectively.

Our primary goal is to design a learning algorithm for a problem, either a classification problem or a function approximation problem, with a large number of examples. For a problem with a large number of examples, overtraining is not a big problem, provided the network architecture is suitably chosen. In our implementation, we choose the data sets with large number of examples, which are much larger than the number of weights. Hence, we do not use the validation set. Instead, we combine the training set and the validation set as the training set.

2 Input and output representation

How to represent the input and output attributes of a learning problem in a neural network implementation of the problem is one of the key decisions influencing the quality of the solutions one can obtain.

In Proben1, the real-valued attributes are usually rescaled by some linear factors. The integer-valued attributes are most often handled as if they were real-valued. Each input in the data set is a real-valued attribute, and each output in the data set is either a real number for the function approximation problems or an integer 0 or 1 for the classification problems.

3 Error measures

Many different error measures (also called error functions, objective functions, cost functions, or loss functions) can be used for network training. The most commonly used is the *squared error*:

$$E(o,t) = \sum_i (o_i - t_i)^2 ,$$

where o_i and t_i are the actual output and target output at the i-th output node for one example. The above measure gives one error value per example — obviously there are too many data to report. Thus one usually reports either the sum or the average of these values over the set of all examples. The average is called the *mean squared error*. The author of [43] believed that the *mean squared error* may have the advantage of being independent of the size of the data set. Note that the *mean squared error* still depends on the number of output coefficients in the problem representation and on the range of output values used. We thus follow [43] to normalize for these factors as well, and report a *squared error percentage* as:

$$E = 100 \cdot \frac{o_{\max} - o_{\min}}{N \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2$$

where $o_{\max}$ and $o_{\min}$ are the maximum and minimum values of output coefficients in the problem representation, N is the number of outputs of the network, and P is the number of examples in the data set considered. However, all the data sets in Proben1 have been normalized such that $o_{\max} = 1$ and $o_{\min} = 0$. So the squared error percentage can be simplified as :

$$E = \frac{100}{N \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2 .$$

Note that this error function is never used in the field of optimization, but is specific to ANN training. In our algorithm, we use this squared error percentage as the error function, so that one may compare our training results with the results in [43].

4 Classification measure

The actual target function for classification problems is usually not the continuous error measure used during training but the classification performance. However, the classification performance is not the only measure we are interested in. We thus report the actual error values in addition to the classification performance. Classification performance is reported in terms of percent of incorrectly classified examples, the percent classification error. This is better than reporting the percentage of correctly classified examples, the classification accuracy, because the latter makes important differences insufficiently clear: an accuracy of 98% is actually twice as good as one of 96%, which is easier to see if the percent errors are reported (2% compared to 4%).

There are several possibilities to determine the classification a network has computed from the outputs of the network.

In our implementation, we use the following method to determine the classification. We calculate

$$d_i = |o_i - t_i|$$

where o_i and t_i are the actual output and target output at the i-th output node for one example. If there is at least one d_i, such that $d_i \geq 0.5$, then we reject this example, otherwise we accept it. We may use 0.4 or 0.3 (instead of 0.5) to determine the rejection region. However, in our implementation, the differences are not significant.

5 Networks used

Neural network structure is one of the most important things to be specified when we use a neural network. No one knows which particular structure is the best for any particular problem. Basically, we just try several different structures. Following [43], we mainly choose a neural network with zero, one, or two hidden layers.

To describe the topology, we try to refer to common topology models. For instance, for the common case of fully-connected layered feed-forward networks, the numbers of nodes in each layer from input to output can be given as a sequence. For example, a 14-50-3 network refers to a network with 14 input, 50 hidden, and 3 output nodes. We call this *a network with one hidden layer*.

6 Stopping criteria

We design a training algorithm based on a limited memory quasi-Newton method to solve both classification problems and approximation problems. Since we intend to

solve problems with a large number of examples, the validation set is not used, and the following stopping criteria are used:

1. The weight update is within tolerance:

$$\|w_{k+1} - w_k\|_2 < \text{tolerance}$$

The tolerance is chosen depending on the problem. We will specifically state the tolerance for our test problems in Section 6.4.

2. The number of function evaluations exceeds a pre-defined limit.

In our implementation, we set this pre-defined limit at 2000. Some authors use the number of iterations instead of the number of function evaluations. In our implementation, the difference between these two numbers is not large. Stopping on this criterion implies failure to converge, although the results might still be of some use.

In addition to the above two stopping criteria, there is another stopping criterion in the subroutine L-BFGS, as we mentioned in Chapter 4.

If a validation set is used in the implementation, besides the above criteria, the GL_α stopping criterion can be used. Although we do not use a validation set in our implementation, we still state the GL_α stopping criterion [43] in the following. Interested readers may use it in their implementation.

Let $E_{va}(t)$ be the squared error percentage over the validation set, measured during epoch t. The value $E_{opt}(t)$ is defined to be the lowest validation set error obtained in the epochs up to t:

$$E_{opt}(t) = \min_{t' \le t} E_{va}(t')$$

Now we define the generalization loss at epoch t to be the relative increase of the validation error over the minimum-so-far (in percent):

$$\mathrm{GL}(t) = 100 \cdot \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right)$$

A high generalization loss is one candidate reason to stop training. This leads us to a class of stopping criteria: Stop as soon as the generalization loss exceeds a certain threshold α. We define the class GL_α as

$$\mathrm{GL}_\alpha : \text{ stop after first epoch } t \text{ with } \mathrm{GL}(t) > \alpha.$$

Typical value used for α is 5 [43].

6.2 Language Implementation

We use the FORTRAN 77 language to implement our learning algorithm. The main purpose of a learning process is to train the neural network to have generalization ability. That is, the network should have small error on data as well which it has not learned.

We choose one-hidden-layer and two-hidden-layer feed-forward neural network architectures. Neural networks without hidden layer are also used. The propagated computations and notations are exactly the same as in Wang [51]. Here we do not repeat them.

Before executing our program, one must prepare an input file exactly named "INPUT.DAT". The format of "INPUT.DAT" is as follows:

The first line--TYPE, SEED

TYPE is an integer, which represents the type of training problem.

TYPE = 1: Function approximation problem.

TYPE = 2: Pattern classification problem.

SEED is an integer, which is used to generate a series of random numbers, which represent the initial weights of the network.

The second line--NTRAIN, NTEST, NLAYER

NTRAIN is an integer, which represents the number of training examples.

NTEST is an integer, which represents the number of test examples.

NLAYER is an integer, which represents the number of layers (including the input layer) of this network.

The third line--integers

These integers represent the number of nodes (excluding the bias) in each layer, starting from the input layer, and ending at the output layer.

In the remainder of the file, each line contains data for one example, data inputs followed by the desired outputs.

The major subroutines of the program and their functions are the following:

LBSET ( ) -- define the values of several parameters in common areas.

INPUT ( ) -- open and read the input data file.

INWEIT ( ) -- initialize all connection weights.

COMOU1 ( ) -- compute the outputs of the network for one example.

GRADIE ( ) -- compute the value and gradient of the error function.

MLBFGS ( ) -- implement the limited memory BFGS algorithm. This is a slight modification of Nocedal's L-BFGS program [36].

6.3 Test Problems.

As we mentioned before, the main purpose of this thesis is to train a neural network with a large number of weights. We choose three data sets with the largest number of examples in Proben1 [43]. All these three data sets have a relatively large number of inputs, so that the number of weights may be large, though it depends on the actual design of the network. If the number of inputs and the number of outputs are fixed, then the larger the number of hidden nodes, the larger the number of weights.

Two of the problems are function approximation problems, while the third is classification problem.

Table 6.1 Test Problems

Problem	Type of Problems	# of Inputs	# of Outputs	# of Training Examples	# of Test Examples
building2	Approximation	14	3	3156	1052
flare1	Approximation	24	3	800	266
thyroid1	Classification	21	3	5400	1800

For further comparison, seven different network topologies were used for each problem: zero-hidden-layer network, one-hidden-layer networks with 4, 16, or 32 hidden nodes and two-hidden-layer networks with 4+4, 8+8, or 16+8 hidden nodes on the first

and second hidden layer, respectively. For the two approximation problems, we also use some other network topologies. All of these networks have all possible feed-forward connections, including a bias connection.

6.4 Test Results

In optimization problems, one uses various methods to find an approximate minimum value of the objective function. Theoretically, the exact minimum of the objective function cannot be known in advance. However, some certain known results can serve as a reference.

In Proben1 [43], a few results of neural network learning runs on the data sets are given. The runs were made with linear networks, having only direct connections from inputs to the outputs, and with various fully connected multilayers with one or two layers of sigmoidal hidden nodes. Training was performed using the RPROP algorithm [43]. We list the average results of [43] in Table 6.2.

We notice that for different network topologies, the propagation from input layer to output layer are different. Hence, the minimum values may not be exactly the same. However, comparing with the results in Table 6-2 , we can get a rough idea how well our program works.

Table 6.2 Results in Proben1

Problem	Total epochs	Training set	Test set	Test set classification
building2	1183	0.23	0.26	-
flare1	71	0.39	0.74	-
thyroid1	491	0.60	1.31	2.32

Training set: minimum squared error percentage on the training set.

Test set: minimum squared error percentage on the test set.

Test set classification: percent of incorrectly classified examples on the test set.

Recall that we first need to set the stopping criteria. The first criterion is that the weight update is within tolerance:

$$\|w_{k+1} - w_k\|_2 < \text{tolerance}.$$

Since the number of weights are relatively large, it is not necessary to set the tolerance to be too small. In our implementation, we set the tolerance = 10^{-3}, with one exception. For the problem building2 with 16+16 hidden nodes, when tolerance = 10^{-3}, it stops too early and does not obtain the desired results. Hence, we set the tolerance = 10^{-5} instead of 10^{-3} for this problem.

The second stopping criterion is that the number of function evaluations cannot exceed a pre-defined limit, which we set as 2000.

The stopping criterion of L-BFGS is:

$$\|g_k\| < \varepsilon \times \max(1, \|w_k\|),$$

We set $\varepsilon = 10^{-4}$ with a few exceptions. With a similar reason as above, we set $\varepsilon = 10^{-5}$ for the following network topologies: flare1 without hidden layer, and thyroid1 with the hidden layers 8+8 and 16+8.

In our actual implementation, for the building2 with 16+16 hidden nodes, the program stops when the number of function evaluations achieves the limit 2000. For all other situations, it stops when either $\|\Delta w\|_2$ is too small or $\|g\|_2$ is too small.

The results of our tests are listed in the following tables.

Table 6.3 Results of building2 ($m = 5$)

Hidden nodes	# of weights	Total epochs	n_f	$\|\|\Delta w\|\|_2$	$\|\|g\|\|_2$	Training set	Test set
None	45	67	75	0.0006483	0.002806	0.3441	0.3427
4	75	759	820	0.008264	0.002349	0.2635	0.2643
8	147	1084	1140	0.0008241	0.01060	0.2472	0.2544
16	291	1331	1421	0.0008560	0.008237	0.2267	0.2443
32	579	1341	1425	0.0009306	0.04055	0.2176	0.2464
4+4	95	1328	1434	0.02195	0.002396	0.2543	0.2598
8+8	219	1631	1747	0.0006908	0.003420	0.2172	0.2402
16+16	563	1801	2000	0.02671	0.01257	0.2073	0.2363

Table 6.4 Results of flare1 ($m = 5$)

Hidden nodes	# of weights	Total epochs	n_f	$\|\|\Delta w\|\|_2$	$\|\|g\|\|_2$	Training set	Test set
None	75	138	150	0.004856	0.0002159	0.2915	0.5962
4	115	99	121	0.03944	0.003599	0.2470	0.7380
8	227	162	183	0.03414	0.003334	0.2118	0.7999
16	451	166	188	0.03917	0.004947	0.2209	0.9708
32	899	153	171	0.6937	0.007733	0.2229	0.8296
4+4	135	121	144	0.07549	0.005195	0.2508	0.7387
8+4	251	143	187	0.4097	0.005541	0.2462	0.6216
8+8	299	121	153	0.01822	0.004801	0.2407	0.7591
16+8	563	306	356	0.01908	0.004518	0.2183	1.026

Table 6.5 Results of thyroid1 ($m = 5$)

Hidden nodes	# of weights	Total epochs	n_f	$\|\|\Delta w\|\|_2$	$\|\|g\|\|_2$	Training set	Test set	Error rate
None	66	98	113	1.431	0.01917	2.885	3.179	6.500
4	103	236	280	0.05897	0.02529	0.9794	1.346	2.444
16	403	371	420	0.04310	0.02790	0.8786	1.246	2.500
32	803	647	735	0.02569	0.02054	1.445	1.751	4.111
4+4	123	35	41	21.94	0.02343	4.469	4.369	7.278
8+8	275	49	59	53.04	0.007671	4.477	4.391	7.278
16+8	515	611	730	0.003879	0.006455	2.696	2.652	4.944

n_f: total number of function evaluations.

Training set: minimum squared error percentage on training set.

Test set: minimum squared error percentage on test set.

Error rate: percent of incorrectly classified examples.

1. **Building2**

It seems that the iterations converge for all network topologies we chose. Except for the network without hidden nodes, the training set error and the test set error are similar in Table 6.2. The best training set error and the best test error are 0.2073 and 0.2363, respectively, which are better than the results in Table 6.2.

2. **Flare1**

It seems that the iterations converge for every network topology. The best training set error is 0.2118, which is obtained by the network topology with eight hidden nodes, while the best test set error is 0.5962 which is obtained by the network without hidden nodes. We notice that the network topology with the best training set error may not have the best test set error.

3. **Thyroid1**

It seems that the iterations converge for some network topologies but do not converge for some other network topologies (none, 4+4, 8+8). The best (smallest) training set error and the best test error are 1.445 and 1.246, respectively, and the best percent of incorrectly classified examples is 2.444%

In the following we summarize our results :

First of all, our program handles the large number of weights of a neural network very well. The largest number of weights in our tests is 899. Actually, we believe that our program can work on a neural network with several thousand weights without any problem.

Second, for most of the network topologies which we chose for the three problems, the optimization procedure converges. The other cases presumably would have converged eventually.

Third, compared with the results in Table 6.2, most of our results are acceptable. Some results are more accurate than the results in Table 6.2. In particular, we get very good generalization results on test sets.

Roughly speaking, the greater the number of hidden nodes, the greater the number of weights, and the greater the number of iterations and functions evaluations required. It seems that the average of the number of iterations of different topologies for each problem is similar in Table 6.2. However, we cannot claim that our program is very fast, since it may take more time for each iteration than the other methods. For example, it took about 30 hours for the problem Thyroid1 with 32 hidden notes when we run it on CSA: Sequent 24 Intel 80386 processors. However, our program can handle a very large number of weights, which the other methods may not be able to (a conjugate gradient method may handle a large problem, but it is unstable.) In addition, as we mentioned before, the main difference between a limited memory quasi-Newton method and a quasi-Newton method is the storage part. If we modify the storage part of our program and store the inverse Hessian matrixes explicitly, then our program can train a network with a moderate number of weights and run very fast.

6.5 Test on XOR Problem

In this section we test our program on an "exclusive or" function of two variables. This is perhaps the simplest learning problem that is not linearly separable. It therefore cannot be solved by a network with no hidden layer. For the detailed description and results see [55].

There are two main forms of learning architectures that have been used by others to solve this problem [55]. The first one is a 2-2-1 network, which has two hidden units, each connected to both inputs and to the output. The second one is a "2-1-1 shortcut network", which has only a single hidden unit, but also has "shortcut" connections from the inputs to the output unit. However, we do not consider the shortcut network here.

Some researchers have also investigated this problem with more than two hidden units [56]. In general, the more hidden units there are, the easier the problem becomes.

We first call our program directly with 2-2-1 and 2-3-1 networks. But it does not work very well. It seems to stop at a local minimum. We then test our program with a 2-4-1 network. The results are satisfactory. We list the results in the following table.

Table 6.6 Results of XOR Problem with 2-4-1 Network

# Of Weights	Total Epochs	n_f	$\|\Delta w\|_2$	$\|g\|_2$	Error	Error Rate
17	27	55	3.187	4.966D-4	3.292D-4	0.0

WEIGHT = (-1.621D-01, 1.400D-01, -4.693D-02, 2.231D-02, -4.414D-02,
6.472D-02, 9.630D-03, 2.710D-02, 8.472D-02, -8.488D-02,
8.082D-02, -5.366D-02, -4.286D-02, -5.266D-02, -9.787D-02,
6.157D-02, -8.823D-02)

error: minimum squared error percentage

error rate: percent of incorrectly classified examples

In our original program, we chose the initial weights between -0.5/fanin and 0.5/fanin, where fanin is the number of nodes in previous layer [51]. Chandler [8] suggested that we enlarge the range of the initial weights for this problem, that is the initial weights are chosen between [-50, 50]. In addition, we randomly choose the initial weights and calculate the value of the error function one thousand times. We save the weights with the smallest of squared error percentage, and use these weights as the initial weights to test the XOR problem with a 2-2-1 network. The results are satisfactory. We list the results in the following table.

Table 6.7 Results of XOR Problem with 2-2-1 Network

# Of Weights	Total Epochs	n_f	$\|\|\Delta w\|\|_2$	$\|\|g\|\|_2$	Error	Error Rate
9	1	2	1.000D00	1.576D-9	1.119D-9	0.0

WEIGHT = (2.598D+01, 2.987D+01, -1.286D+01, 8.498D+00, -3.505D+01, 3.404D+01, 1.236D+01, 2.570D+01, 4.382D+01)

7. Conclusions

In this thesis, we designed a neural network program based on limited memory quasi-Newton methods to train fully-connected feed-forward neural networks. Our program can train a neural network with a large number of weights and it has been tested on several real world problems in Proben1 [43]. Comparing with the results in Proben1, our results are satisfactory. In particular, we obtain very good generalization results on the test sets. Since we do not store the inverse Hessian matrix explicitly, it may take more time for each iteration than for other methods. However if we modify the storage part of our program, it will run very fast in training a neural network with a moderate number of weights.

In addition, we tested the subroutine L-BFGS on various functions and obtained very good test results.

Suggestion for further study:

Test our program on real data sets with a very large number of inputs and not very many examples. If necessary, use the validation set as a pseudo test set.

References

[1] M. Al-Baali, *Descent property and global convergence of the Fletcher-Reeves method with inexact line searches*, IMA J. Numer. Anal., **5** (1985) 121-124.

[2] E. M. L. Beale, *On an iterative method for finding a local minimum of a function of more than one variable*, Tech. Report No. 25, Statistical Techniques Research Group, Princeton Univ. (1958).

[3] H. S. M. Beigi and C. J. Li, *Learning algorithms for neural network based on quasi-Newton methods with self-scaling*, J. of Dynamics Systems, Measurement and Control, **115**(1993) 38-43.

[4] M. J. Box, *A comparison of several current optimization methods, and the use of transformations in constrained problems*, Comp. J. **9** (1966) 67-77.

[5] R. P. Brent, *Algorithms for Minimization without Derivatives*, Prentice-Hall, Inc., Englewood Cliffs, N. J. 1973.

[6] R. H. Byrd, J. Nocedal, and Y. Yuan, *Global convergence of a class of quasi-Newton methods on convex problems*, SIAM J. Numer. Anal., **24** (1987) 1171-1190.

[7] J. P. Chandler, Anonymous ftp://a.cs.okstate.edu/pub/jpc/praxis.f, praxis.txt, stepit.f.

[8] J. P. Chandler, Private communication.

[9] A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, John Wiley & Sons, Inc., New York, 1993.

[10] A. R. Colville, *A comparative study of nonlinear programming codes*, IBM New York Scientific Center Report, 320-2949 (1968).

[11] J. E. Dennis and R. E. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, 1983.

[12] R. Fletcher, *Practical Methods of Optimization*, Vol. 1, John Wiley & Sons, Inc., Chichester, 1980.

[13] R. Fletcher and M. J. D. Powell, *A rapidly convergent descent method for minimization*, Comp. J. **6** (1963) 163-168.

[14] R. Fletcher and C. M. Reeves, Function minimization by conjugate gradients, Computer J. 7 (1964) 149-154.

[15] J. A. Freeman and D. M. Skapura, *Neural Networks Algorithms, Applications, and Programming Techniques*, Addison-Wesley Publishing Co., New York, 1992.

[16] J. C. Gilbert and J. Nocedal, *Global convergence properties of conjugate gradient methods for optimization*, SIAM J. Optimization, **2** (1992) 21-42.

[17] P. E. Gill and W. Murray, *Quasi-Newton methods for unconstrained optimization*, J. Inst. Math. Applics. **9** (1972) 91-108.

[18] P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization*, Academic Press, London, 1981.

[19] R. T. Gregory and D. L. Karney, *A Collection of Matrices for Testing Computational Algorithms*, Interscience, New York, 1969.

[20] A. Griewank and Ph. L. Toint, *Partitioned variable metric updates for large structured optimization problems*, Numerische Mathematik **39** (1982) 119-137.

[21] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*, PWS Publishing Co., Boston, 1996.

[22] M. T. Hagan and M. B. Menhaj, *Training feedforward networks with the Marquardt algorithm*, IEEE Transactions On Neural Networks **5** (1994) 989-994.

[23] S. Haykin, *Neural Networks, A Comprehensive Foundation*, Macmillan College Publishing Co., 1994.

[24] N. B. Karayiannis and A. N. Venetsanopoulos, *Artificial Neural Networks, Learning Algorithms, Performance Evaluation, and Applications*, Kluwer Academic Publishers, Boston, 1993.

[25] J. S. Kowalik and M. R. Osborne, *Methods for Unconstrained Optimization Problems*, Elsevier, New York, 1968.

[26] A. Leon, *A comparison of eight known optimizing procedures*, Recent advances in optimization techniques, A. Lavi and T. P. Vogl eds.,Wiley & Sons, Inc., New York, (1966).

[27] K. Levenberg, *A Method For the Solution of Certain Non-Linear Problems in Least Squares*, Quart. Appl. Math. **2** (1944) 164-168.

[28] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Programming **45** (1989) 503-528.

[29] D. C. Liu and J. Nocedal, *Test results of two limited memory methods for large scale optimization*, Technical Report NAM 04, Department of Electrical Engineering and Computer Science, Northwestern University, 1988.

[30] D. W. Marquardt, *An Algorithm for Least-Squares Estimation of Nonlinear Parameters*, J. Soc. Indust. Appl. Math. **11** (1963) 525-553.

[31] W. Miller and D. Spooner, ACM Trans. Math. Software 4 (1978) 369-390.

[32] J. J. Moré and D. J. Thuente, *Line search algorithms with guaranteed sufficient decrease*, ACM Transaction on Mathematical Software **20** (1994) 287-307.

[33] S. G. Nash, *Preconditioning of truncated-Newton methods,* SIAM Journal on Scientific and Statistical Computing **6** (1985) 599-616.

[34] S. G. Nash and A. Sofer, *Linear and Nonlinear Programming*, McGraw-Hill Inc., New York, 1996.

[35] J. Nocedal, *Updating quasi-Newton matrices with limited storage*, Mathematics of Computation **35** (1980) 773-782.

[36] J. Nocedal, Anonymous ftp://eecs.nwu.edu/pub/lbfgs/lbfgs_um

[37] L. Nyhoff and S. Leestma, *FORTRAN 77 and Numerical Methods for Engineers and Scientists*, Prentice-Hall, New Jersey, 1995.

[38] D. P. O'Leary, *A discrete quasi-Newton algorithm for minimizing a function of many variables*, Mathematical Programming **23** (1982) 20-33.

[39] M. R. Osborne. *Some aspects of non-linear least squares calculations*, Numerical Methods for Non-linear Optimization, F. A. Lootsma ed., Academic Press, New York, (1971) 171-189.

[40] M. J. D. Powell, *An iterative method for finding stationary values of a function of several variables*, Comp. J. **5** (1962) 147-151.

[41] M. J. D. Powell, *An efficient method for finding the minimum of a function of several variables without calculating derivatives*, Comp. J. **7** (1964) 155-162.

[42] M. J. D. Powell, *Some global convergence properties of a variable metric method without line searches*, Nonlinear Programming, R. W. Cottle and C. E. Lemke, eds., SIAM-AMS Proceedings, American Mathematical Society, **9** (1976) 53-72.

[43] L. Prechelt, *Proben1 — A Set of Neural Network Benchmark Problems and Benchmarking Rules*, Technical Report 21/94, University Karlsruhe, 1994, Anonymous ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz.

[44] B. D. Ripley, *Pattern Recognition and Neural Networks,* Cambridge University Press, 1996.

[45] H. H. Rosenbrock, *An automatic method for finding the greatest or least value of a function*, Comp. J. **3** (1960)173-184.

[46] D. E. Rumelhart, G. E. Hinton and R. J. Williams, *Learning representations by back-propagation errors*, Nature **323** (1986) 533-536.

[47] L. E. Scales, *Introduction to Non-Linear Optimization*, Springer-Verlag, New York, 1985.

[48] T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM Journal on Numerical Analysis **20** (1983) 626-637.

[49] J. G. Taylor, *The Promise of Neural Networks*, Springer-Verlag, London, 1993.

[50] Ph. L. Toint, *Towards an efficient sparsity exploiting Newton method for minimization*, in: Sparse Matrices and Their Uses, I. S. Duff, ed., Academic Press, New York, 1981, 57-58.

[51] L. Wang, *The Damped Newton Method - An ANN Learning Algorithm*, M. S. Thesis, Computer Science Department, Oklahoma State University, 1995.

[52] M. A. Wolfe, *Numerical Methods for Unconstrained Optimization*, Van Nostrand Reinhold Company, Ltd., Molly Millars Lane, Wokingham, Berkshire, England, 1978.

[53] System/360 Scientific Subroutine Package (360A-CM-03X) Version III Programmers Manual, IBM Corporation, 1968.

[54] Internet: comp.ai.neural-nets.

[55] Anonymous FTP: ftp.cs.cmu.edu/afs/cs/project/connect/bench/xor.

APPENDIX A: PROGRAM LIST FOR DRIVEN.F

```
      PROGRAM DRIVEN
C-----------------------------------------------------------------------
C   WRITTEN BY JOHN P. CHANDLER AND YIJUN HUANG,
C   COMPUTER SCIENCE DEPARTMENT, OKLAHOMA STATE UNIVERSITY, 1997.
C-----------------------------------------------------------------------
C   THIS IS A DRIVER TO IMPLEMENT A NEURAL NETWORK LEARNING
C   ALGORITHM BASED ON THE LIMITED MEMORY QUASI-NEWTON METHOD.
C
C   THE ACTUAL LIMITED MEMORY BFGS METHOD IS IMPLEMENTED BY
C   MEANS OF THE SUBROUTINE MLBFGS, WHICH IS A SLIGHT
C   MODIFICATION OF THE ROUTINE LBFGS WRITTEN BY JORGE NOCEDAL.
C
C   BEFORE THE PROGRAM IS EXECUTED, THE INPUT FILE "INPUT.DAT"
C   MUST BE READY.
C
C   INPUT FILE FORMAT:
C
C   1ST LINE --  TYPE, SEED
C                TYPE IS AN INTEGER, WHICH REPRESENTS THE TYPE OF
C                     THE TRAINING PROBLEM.
C                     TYPE = 1:  FUNCTION APPROXIMATION PROBLEM.
C                     TYPE = 2:  PATTERN CLASSIFICATION PROBLEM.
C                SEED IS AN INTEGER, WHICH IS USED TO GENERATE
C                     RANDOM NUMBERS.
C
C   2ND LINE -- NTRAIN, NTEST, NLAYER  ARE ALL INTEGERS, WHICH
C               REPRESENT THE NUMBER OF TRAINING EXAMPLES, THE
C               NUMBER OF TEST EXAMPLES, AND THE NUMBER OF LAYERS
C               (INCLUDING THE INPUT LAYER) OF THE NETWORK,
C               RESPECTIVELY.
C
C   3RD LINE -- INTEGERS, WHICH REPRESENT THE NUMBER OF NODES (EXCLUDING
C               THE BIAS) IN EACH LAYER, START FROM THE INPUT LAYER.
C
C   THE REMAINDER OF THE FILE EACH CONTAINS DATA FOR ONE
C   EXAMPLE, DATA INPUT FOLLOWED BY THE DESIRED OUTPUT.
C   VARIABLES:
C
C   N       IS AN INTEGER, WHICH REPRESENTS THE NUMBER OF WEIGHTS
C
C   WEIGHT AND G ARE DOUBLE PRECISION ARRAYS, WHICH CONTAIN THE VALUE OF
C          WEIGHTS AND THE GRADIENT OF THE ERROR FUNCTION, RESPECTIVELY.
C
C   M       IS AN INTEGER, WHICH REPRESENTS THE NUMBER OF CORRECTION
C          USED IN THE BFGS UPDATE.
C
C   F       IS A DOUBLE PRECISION VARIABLE CONTAINS THE VALUE OF THE
C          ERROR FUNCTION IN THE TRAINING PROCESS.
C
C   ERTEST IS A DOUBLE PRECISION VARIABLE CONTAINS THE VALUE OF THE
C          ERROR FUNCTION IN THE GENERATION PROCESS WITH TEST DATA
C          SET.
C
C   LAYIFO IS A TWO DIMENSION INTEGER ARRAY USED TO STORE THE
C          NETWORK STRUCTURE.
C          THE FIRST INDEX OVER LAYERS (INCLUDING THE INPUT LAYER).
C          THE 2ND INDEX ARE DEFINED AS FOLLOWS:
C              (*,1)  CONTAINS THE NUMBER OF NODES IN EACH LAYER,
C                     EXCLUDING THE BIAS NODE.
C              (*,2)  CONTAINS THE STARTING INDEX OF NEURON FOR
C                     EACH LAYER.
```

```
C               (*,3)  CONTAINS THE STARTING INDEX OF WEIGHT FOR
C                      EACH LAYER.
C
C   NEURON IS TWO DIMENSION DOUBLE PRECISION ARRAY.
C          THE 1ST INDEX OVER ALL NEURONS, OR NODES, IN THE NETWORK.
C          THE 2ND INDEX ARE DEFINED AS FOLLOWS:
C               (*,1)  CONTAINS THE OUTPUT VALUE OF EACH NEURON.
C               (*,2)  CONTAINS THE FIRST DERIVATIVE OF THE
C                      ACTIVATION FUNCTION OF EACH NEURON.
C               (*,3)  STORES THE PARTIAL DERIVATIVE OF THE ERROR
C                      FUNCTION E(W) W.R.T. A NEURON OUTPUT.
C
C   INDATA IS A TWO DIMENSIONAL DOUBLE PRECISION ARRAY.
C          THE 1ST INDEX CONTAINS THE INPUTS FOR ONE TRAINING EXAMPLE.
C          THE 2ND INDEX OVER ALL TRAINING EXAMPLES IN THIS NETWORK.
C          I. E., EACH COLUMN OF THE ARRAY CONTAINS INPUTS DATA OF
C          ONE TRAINING EXAMPLE.
C
C   OUDATA IS A TWO DIMENSIONAL DOUBLE PRECISION ARRAY.
C          THE 1ST INDEX CONTAINS THE OUTPUTS FOR ONE TRAINING EXAMPLE.
C          THE 2ND INDEX OVER ALL TRAINING EXAMPLES IN THIS NETWORK.
C          I. E., EACH COLUMN OF THE ARRAY CONTAINS OUTPUT DATA FOR
C          ONE TRAINING EXAMPLE.
C
C   TESTIN IS A TWO DIMENSIONAL DOUBLE PRECISION ARRAY.
C          THE 1ST INDEX CONTAINS THE INPUTS FOR ONE TEST EXAMPLE.
C          THE 2ND INDEX OVER ALL TEST EXAMPLES IN THIS NETWORK.
C          I. E., EACH COLUMN OF THE ARRAY CONTAINS INPUTS DATA FOR
C          ONE TEST EXAMPLE.
C
C   TESOUT IS A TWO DIMENSIONAL DOUBLE PRECISION ARRAY.
C          THE 1ST INDEX CONTAINS THE OUTPUT FOR ONE TEST EXAMPLE.
C          THE 2ND INDEX OVER ALL TEST EXAMPLES IN THIS NETWORK.
C          I. E., EACH COLUMN OF THE ARRAY CONTAINS OUTPUT DATA FOR
C          ONE TEST EXAMPLE.
C
C   ICALL  IS A INTEGER, WHICH REPRESENTS THE NUMBER OF
C          EVALUATIONS OF F AND G.
C
C   ANN    IS A LOGICAL VARIABLE.  SINCE THE SUBROUTINE MLBFGS IS
C          CALLED BY TWO DRIVERS, DRIVEN USED TO IMPLEMENT THE
C          ARTIFICIAL NEURAL NETWORKS, THEREFORE SET THE ANN = TRUE
C          IN THIS DRIVER.
C
C   IN1, INTRAI, INTEST, NOUT, N1, N2, L1, L2   ARE ALL POSITIVE
C         INTEGERS, USED TO PASS THE DIMENSION INDICES WHEN CALLING
C         SUBROUTINES.
C
C   NTRAIN, NTEST, NLAYER    ARE THE NUMBER OF TRAINING EXAMPLES, THE
C         NUMBER OF TEST EXAMPLES AND THE NUMBER OF LAYERS RESPECTIVELY.
C
C   OTHER VARIABLES AND PARAMETERS ARE DESCRIBED IN THE SUBROUTINES
C   LBSET AND MLBFGS.
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      DOUBLE PRECISION WEIGHT(2000),G(2000),DIAG(2000),W(35000)
      DOUBLE PRECISION F,EPS,XTOL,GTOL,STPMIN,STPMAX,TOLERA,ER
      INTEGER IPRINT(2),IFLAG,ICALL,N,M,MP,LP,J,NFMAX,I,K,NOUT
      LOGICAL DIAGCO,ANN,GRD
C
      INTEGER    TYPE,NTRAIN,NTEST,NLAYER,LAYIFO(5,3),II,
     *   SEED, IN1,INTRAI,INTEST,NOUT1,N1,N2,L1,L2,NF05
      DOUBLE PRECISION  INDATA(25,5400),OUDATA(5,5400),
     *   TESTIN(25,1800),TESOUT(5,1800),
     *   NEURON(200,3),ERTEST,ERRATE
```

```
C
      COMMON /LB3/MP,LP,GTOL,STPMIN,STPMAX
      COMMON /IP/ NTRAIN, NLAYER
C
      XTOL=1.0D-16
      EPS=1.0D-4
      TOLERA = 1.0D-3
      N1=200
      N2=3
      L1=5
      L2=3
      IN1=25
      INTRAI=5400
      INTEST=1800
      NOUT=5
C
      M=5
      IPRINT(1)= 50
      IPRINT(2)= 0
C
C  WE DO NOT WISH TO PROVIDE THE DIAGONAL MATRICES HK0, AND
C  THEREFORE SET DIAGCO TO FALSE.
C
      DIAGCO= .FALSE.
      ANN = .TRUE.
      ICALL=0
      IFLAG=0
      NFMAX = 2000
      CALL LBSET
C
C  READ IN INPUT FILE AND INITIALIZE THE WEIGHTS.
C
      CALL INPUT(INDATA,OUDATA,TESTIN,TESOUT,LAYIFO,NTEST,TYPE,SEED,
     *   L1,L2,IN1,INTRAI,INTEST,NOUT)
      CALL INWEIT(WEIGHT,N,LAYIFO,SEED,L1,L2)
C
C  SET BIAS INPUT.
C
      DO 10 K=1, NLAYER
         II = LAYIFO(K, 2)
         NEURON(II, 1) = -1.0
   10 CONTINUE
C
C  --- MAIN LOOP ---
C
   20 CONTINUE
      F= 0.D0
      DO 30 I=1,N
         G(I)=0.0D0
   30 CONTINUE
C
C  COMPUTE VALUE AND GRADIENT OF ERROR FUNCTION .
C
      CALL GRADIE(F,WEIGHT,N,G,INDATA,OUDATA,LAYIFO, NEURON,
     *   N1,N2,L1,L2,IN1,INTRAI,INTEST,NOUT)
C
      CALL MLBFGS(N,M,WEIGHT,F,G,DIAGCO,DIAG,IPRINT,EPS,
     *   XTOL,W,IFLAG,ANN,TOLERA)
C
      IF(IFLAG.EQ.1)   THEN
C
C  IF IFLAG=1, EVALUATE THE FUNCTION F AND GRADIENT G.
C
         ICALL=ICALL + 1
C
```

```
C  WE ALLOW AT MOST NFMAX EVALUATIONS OF F AND G
C
         IF(ICALL.GT.NFMAX) GO TO 40
         GO TO 20
C
C  IFLAG = 0 INDICATES THE ROUTINE MLBFGS HAS TERMINATED SUCCESSFUL,
C  THEREFORE, EVALUATE THE TEST DATA; OTHERWISE, AN ERROR OCCURS,
C  THEN STOP THE EXECUTION OF THE PROGRAM.
C
      ELSE IF (IFLAG .EQ. 0)  THEN
   40    ERTEST = ER(TYPE,TESTIN,TESOUT,NTEST,NEURON,
     *       LAYIFO,WEIGHT,N,NF05,N1,N2,L1,L2,IN1,INTEST,NOUT)
         WRITE(MP,50) ERTEST,EPS
         IF(TYPE .EQ. 2)  THEN
            ERRATE = 1.0*NF05/NTEST
            WRITE(MP,60) NF05,ERRATE
         END IF
      END IF
   50 FORMAT(/'  ERTEST = ', 1PD10.3,2X,'EPS=',1PD10.3)
   60 FORMAT(/'  NF05 = ',I5,2X,'ERRATE=',1PD10.3)
C
      CONTINUE
C
      STOP
      END
      SUBROUTINE INPUT(INDATA, OUDATA, TESTIN, TESOUT, LAYIFO, NTEST,
     *   TYPE,SEED,L1,L2,IN1,INTRAI,INTEST,NOUT)
C-----------------------------------------------------------------------
C   OPEN AND READ IN THE "INPUT.DAT". STORE THE NETWORK INFORMATION
C   INTO THE ARRAY LAYIFO, AND ALL TRAINING DATA AND TESTING DATA
C   INTO INDATA, OUTADA, TESTIN, AND TESOUT, RESPECTIVELY.
C   THE FORMAT OF "INPUT.DAT" AND ALL VARIABLES ARE DESCRIBED
C   AT THE BEGINNING OF THE MAIN PROGRAM.
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER L1,L2,I,IN,J,IN1,INTRAI,INTEST,NOUT,NOUT1,NINPUT
      INTEGER  IOERR,TYPE,SEED,NTRAIN,NTEST,LAYIFO(L1,L2),NLAYER
      DOUBLE PRECISION   INDATA(IN1,INTRAI),OUDATA(NOUT,INTRAI),
     *   TESTIN(IN1,INTEST),TESOUT(NOUT,INTEST)
C
      COMMON /IP/NTRAIN, NLAYER
C
      IN = 5
C
      IOERR=0
      OPEN(UNIT=IN, FILE='INPUT.DAT', STATUS='OLD', IOSTAT=IOERR)
      IF(IOERR .NE. 0) THEN
         PRINT 10, IOERR
   10    FORMAT('CANNOT OPEN INPUT DATA FILE, IOERR=',I10)
         GOTO 120
      END IF
C
      READ(IN,20)  TYPE, SEED
   20 FORMAT(2I5)
      PRINT 30,TYPE,SEED
   30 FORMAT(/' TYPE =',I5,5X,'SEED =',I11)
C
      READ(IN,40)   NTRAIN, NTEST, NLAYER
   40 FORMAT(3I5)
      PRINT 50,NTRAIN,NTEST,NLAYER
   50 FORMAT(/' NTRAIN =',I9,8X,'NTEST =',I9,8X,'NLAYER =',I2)
C
      READ(IN,60)  (LAYIFO(I,1), I=1, NLAYER)
   60 FORMAT(5I5)
      PRINT 70,(LAYIFO(I,1), I=1, NLAYER)
```

```
   70 FORMAT(/' LAYIFO(I,1) =',8I5)
C
      LAYIFO(1,2) = 1
      LAYIFO(1,3) = 0
      LAYIFO(2,3) = 1
C
      DO 80 I=2, NLAYER
         LAYIFO(I,2) = LAYIFO(I-1,1) + LAYIFO(I-1,2) + 1
   80 CONTINUE
      DO 90 I=3, NLAYER
         LAYIFO(I,3)=LAYIFO(I-1,3)+LAYIFO(I-1,1)*(LAYIFO(I-2,1)+1)
   90 CONTINUE
C
      NOUT1 = LAYIFO(NLAYER,1)
      NINPUT = LAYIFO(1,1)
C
      DO 100 J=1, NTRAIN
         READ(IN,*) (INDATA(I,J),I=1,NINPUT),(OUDATA(I,J),I=1,NOUT1)
  100 CONTINUE
C
      DO 110 J=1, NTEST
         READ(IN,*) (TESTIN(I,J),I=1,NINPUT), (TESOUT(I,J),I=1,NOUT1)
  110 CONTINUE
C
      CLOSE(IN)
C
  120 RETURN
      END
      SUBROUTINE INWEIT(WT,N,LAYIFO,SEED,L1,L2)
C-----------------------------------------------------------------------
C     THIS SUBROUTINE IS USED TO INITIALIZE ALL THE CONNECTION WEIGHTS.
C     EACH WEIGHT IS INITIALIZED TO A RANDOM NUMBER BETWEEN -0.5/FAN-IN
C     AND 0.5/FAN-IN, WHERE FAN-IN IS THE NUMBER OF NODES(INCLUDING THE
C     BIAS NODE) IN THE PREVIOUS LAYER.
C
C     THE CALLING STATEMENT IS
C         CALL INWEIT(WT,N,LAYIFO,SEED,L1,L2)
C
C     WHERE
C
C     WT          IS A DOUBLE PRECISION ARRAY, WHICH IS USED TO CONTAIN
C                 THE INITIAL WEIGHTS.
C
C     LAYIFO      IS A TWO-DIMENSION ARRAY OF INTEGERS, WHICH CONTAINS
C                 SOME NETWORK INFORMATION, SUCH THAT THE NUMBER OF NODES
C                 IN EACH LAYER.
C
C     SEED        IS AN INTEGER VARIABLE THAT USED TO GENERATE RANDOM
C                 NUMBERS.
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER  L1,L2,LLL,I,K, NTRAIN, NLAYER
      INTEGER  SEED,TEMP, FANIN, LAYIFO(L1,L2), N
      DOUBLE PRECISION WT(1), RANDOM
C
      COMMON /IP/ NTRAIN, NLAYER
C
      TEMP = 0
C
C  ITERATE OVER ALL LAYERS (EXCLUDING THE INPUT LAYER).
C
      DO 20 K = 2, NLAYER
C
C  FANIN = THE NUMBER OF NODES IN PREVIOUS LAYER + 1(BIAS NODE)
C
```

```
         FANIN = LAYIFO(K-1,1)+1
         LLL=LAYIFO(K,1)*FANIN
         DO 10 I = 1,LLL
            WT(TEMP+I) = RANDOM(SEED)/FANIN
   10    CONTINUE
         TEMP = TEMP+I-1
   20 CONTINUE
C
C  SET THE TOTAL NUMBER OF WEIGHTS.
C
      N = TEMP
      RETURN
      END
      DOUBLE PRECISION FUNCTION RANDOM( SEED )
C-----------------------------------------------------------------------
C     THIS FUNCTION IS USED TO GENERATE A RANDOM NUMBER BETWEEN -0.5
C     TO 0.5.
C     REFERENCE: "A PORTABLE RANDOM NUMBER GENERATOR FOR USE IN SIGNAL
C                PROCESSING', SANDIA NATIONAL LABORATORIES TECHNICAL
C                REPORT, BY S.D.STEARNS.
C     INPUT:     SEED
C     RETURN:    A DOUBLE PRECISION RANDOM NUMBER.
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER SEED
C
      SEED = 2045*SEED + 1
      SEED  = SEED - (SEED/1048576)*1048576
      RANDOM = (SEED+1)/1048577.0 - 0.5
C
      RETURN
      END
      DOUBLE PRECISION FUNCTION INPROD (VEC1, VEC2, DIM )
C-----------------------------------------------------------------------
C     THIS FUNCTION RETURNS THE INNER PRODUCT OF TWO VECTORS.
C     IT IS USED IN SUBROUTINE COMOU1 TO CALCULATE THE
C     OUTPUT OF THE NETWORK.
C-----------------------------------------------------------------------
      INTEGER DIM,I
      DOUBLE PRECISION VEC1(DIM), VEC2(DIM)
C
      INPROD = 0.0
C
C     ITERATE OVER ALL ELEMENTS OF THE VECTORS.
C
      DO 10 I=1, DIM
         INPROD = INPROD + VEC1(I)*VEC2(I)
   10 CONTINUE
C
      RETURN
      END
      DOUBLE PRECISION FUNCTION SIGMOD(X)
C-----------------------------------------------------------------------
C     THIS FUNCTION RETURNS THE VALUE OF SINGMOID ACTIVATION FUNCTION.
C     IT IS USED IN SUBROUTINE COMOU1 TO CALCULATE THE
C     OUTPUT OF THE NETWORK.
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      DOUBLE PRECISION X
C
      SIGMOD = 1.0/(1.0 + DEXP(-X))
C
      RETURN
      END
      SUBROUTINE COMOU1(A,NEURON,LAYIFO,WT,N,N1,N2,L1,L2,IN1,GRD)
```

```
C-----------------------------------------------------------------------
C     THIS SUBROUTINE IS USED TO COMPUTE THE OUTPUT OF THE NETWORK FOR
C     EACH NEURON, AND STORE THEM IN ARRAY NEURON(*,1).
C     IT IS CALLED BY THE FUNCTION ER TO CALCULATE THE VALUE OF
C     THE ERROR FUNCTION ON THE TESTING DATA SET.
C     IN ADDITION, IT COMPUTES THE DERIVATIVE OF THE ACTIVATION FUNCTION
C     AND STORES THEM IN NEURON(*,2) FOR FURTHER REFERENCE.
C     IT IS CALLED BY SUBROUTINE GRADIE TO CALCULATE THE VALUE AND
C     GRADIENT OF THE ERROR FUNCTION ON TRAINING DATA SET.
C
C     THE CALLING STATEMENT IS
C         CALL  SUBROUTINE COMOU1(A,NEURON,LAYIFO,WT,N,N1,N2,L1,L2,
C                                 IN1,GRD)
C     WHERE
C
C     A       IS A DOUBLE PRECISION ARRAY THAT CONTAINS INPUT DATA.
C
C     NEURON IS A TWO-DIMENSIONAL DOUBLE PRECISION ARRAY THAT CONTAINS
C            OUTPUT OF THE NETWORK AND THE DERIVATIVE OF THE ACTIVATION
C            FUNCTION.
C
C     LAYIFO IS A TWO-DIMENSIONAL DOUBLE PRECISION ARRAY WHICH CONTAINS
C            SOME NETWORK INFORMATION.
C
C     WT     IS A DOUBLE PRECISION ARRAY THAT CONTAINS CURRENT
C            WEIGHTS.
C
C     UP AND LOW     ARE DOUBLE PRECISION VARIABLES. SINCE THE SIGMOID
C            FUNCTION F(X) WILL OVERFLOW WHEN THE ABSOLUTE OF X IS
C            TOO LARGE. WE SET THE FUNCTION VALUE TO BE 1.0
C            OR 0.0 WHEN X IS LARGER THAN UP OR SMALLER THAN LOW.
C
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER L1,L2,LLL,I,INDX,K,NDEX,NLAYER,NTRAIN
      INTEGER  TEMP, WIDX, N, LAYIFO(L1,L2),N1,N2,IN1,JJ
      DOUBLE PRECISION   SUM,WT(N),NEURON(N1,N2),INPROD,SIGMOD,
     *   A(IN1),UP,LOW
      LOGICAL  GRD
      COMMON /IP/ NTRAIN, NLAYER
C
C  SET UPPER AND LOWER BOUND FOR SIGMOID FUNCTION.
C
      UP = 7.0D+2
      LOW = -7.0D+2
C
C  COPY ALL INPUT DATA INTO NUERON(*,1), LAYIFO(1,1)=NUMBER OF INPUTS
C
      LLL=LAYIFO(1,1)+1
      DO 10 I=2, LLL
         NEURON(I,1) = A(I-1)
   10 CONTINUE
C
C  FORWARD PROPAGATED COMPUTATION OVER ALL LAYERS.
C
      DO 30 K=2, NLAYER
         INDX = LAYIFO(K,2)
         TEMP = LAYIFO(K,3)
         NDEX = LAYIFO(K-1,2)
         JJ = LAYIFO(K-1,1) + 1
C
C  ITERATE OVER ALL NEURONS IN LAYER K.
C  NOTE: LAYIFO(K,1) = NUMBER OF NEURONS IN K-TH LAYER.
C
         LLL=LAYIFO(K,1)
```

```
         DO 20 I=1,LLL
C
C  LOCATE CORRESPONDING INDEX FOR NEURONS AND WEIGHTS
C
            INDX = INDX+1
            WIDX = TEMP + JJ * (I-1)
C
C  COMPUTE WEIGHTED SUM WITH FUNCTION INPROD
C
            SUM = INPROD(NEURON(NDEX,1), WT(WIDX), JJ)
C
C  THE VALUE OF SIGMOID FUNCTION APPROACHES 1.0 OR 0 WHEN
C  THE ABSOLUTE VALUE OF SUM IS SUFFICIENT LARGE.
C
            IF(SUM .LE. UP .AND. SUM .GE. LOW ) THEN
               NEURON(INDX,1) = SIGMOD(SUM)
            ELSE IF (SUM .GT. UP) THEN
               NEURON(INDX,1) = 1.0
            ELSE
               NEURON(INDX,1)  = 0.0
            END IF
C
C  COMPUTE DERIVATIVE OF THE SIGMOID ACTIVATION FUNCTION F(X) WHEN IT
C  IS CALLED BY SUBROUTINE GRADIE (GRD=TRUE).
C  NOTE: F'(X)=F(X)*(1-F(X))
C
            IF(GRD)
     *         NEURON(INDX,2) = NEURON(INDX,1)*(1.0-NEURON(INDX,1))
C
   20    CONTINUE
   30 CONTINUE
C
      RETURN
      END
      DOUBLE PRECISION FUNCTION ER(TYPE,INARY,OUTARY,NUMBER,NRON,
     *   LAYIFO,WT,N,NF05,N1,N2,L1,L2,IN1,INTEST,NOUT)
C-----------------------------------------------------------------
C     THIS FUNCTION IS USED TO EVALUATE THE TRAINING RESULT WITH
C     THE TEST DATA SET WHEN TRAINING PROCESS HAS COMPLETED.
C     IT CALCULATES THE NUMBER OF INCORRECTLY CLASSIFICATION EXAMPLES
C     AND RETURN THE VALUE OF THE ERROR FUNCTION.
C
C     TYPE     IS AN INTEGER VARIABLE THAT SPECIFIES THE TRAINING
C              PROBLEM TYPE. FUNCTION APPROXIMATION TYPE = 1, WHILE
C              PATTERN CLASSIFICATION PROBLEM IF TYPE =2.
C
C     INARY    IS A TWO-DIMENSIONAL ARRAY. EACH COLUMN CONTAINS INPUT DAT
C              FOR AN EXAMPLE.
C
C     OUTARY   IS A TWO-DIMENSIONAL ARRAY. EACH COLUMN CONTAINS DESIRED
C              OUTPUT FOR AN EXAMPLE.
C
C     NUMBER   IS AN INTEGER VARIABLE, INDICATE THE NUMBER OF TOTAL
C              EXAMPLES IN THE TEST DATA SET.
C
C     NRON     IS A DOUBLE PRECISION ARRAY THAT CONTAINS ACTUAL
C              OUTPUT OF THE NETWORK.
C
C     LAYIFO   IS A TWO-DIMENSIONAL INTEGER ARRAY WHICH CONTAINS
C              SOME NETWORK INFORMATION.
C
C     WT       IS A DOUBLE PRECISION ARRAY THAT CONTAINS CURRENT
C              WEIGHTS.
C
C     NF05     IS THE NUMBER OF INCORRECTLY CLASSIFIED EXAMPLES.
```

```
C             IF THERE IS ANY ONE OF ABSOLUTE VALUE OF DIFFERENCE
C             BETWEEN THE ACTUAL OUTPUT AND THE DESIRED OUTPUT
C             OF A EXAMPLE IS GREATER THAN 0.5, THEN THIS EXAMPLE
C             REFERS TO INCORRECTLY CLASSIFIED, AND NF05
C             INCREASE BY 1.
C             NOTE: IT IS USED ONLY FOR PATTERN CLASSIFICATION
C                   PROBLEMS (TYPE = 2) AND WITH THE TEST
C                   DATA SET WHEN TRAINING PROCESS FINISHED .
C
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER     L1,L2,N1,N2,IN1,INTEST,NOUT
      INTEGER     TYPE,NUMOUT,INDX,LAYIFO(L1,L2),N,NUMBER,NF05
      DOUBLE PRECISION    WT(N), NRON(N1,N2), TEMP,
     *   INARY(IN1,INTEST),OUTARY(NOUT,INTEST)
      LOGICAL             ACCEPT,GRD
C
      COMMON /IP/ NTRAIN, NLAYER
C
      GRD = .FALSE.
      ER = 0
      NF05 = 0
      NUMOUT = LAYIFO(NLAYER, 1)
C
C  ITERATE OVER ALL EXAMPLES.
C
      DO 20 M=1, NUMBER
         ACCEPT = .TRUE.
C
C  CALL COMOU1 TO COMPUTE THE OUTPUT FOR M-TH EXAMPLE
C
         CALL COMOU1(INARY(1,M),NRON,LAYIFO,WT,N,N1,N2,L1,L2,IN1,GRD)
C
C  ITERATE OVER ALL OUTPUT OF M-TH EXAMPLE.
C
         DO 10 I=1, NUMOUT
            INDX = LAYIFO(NLAYER, 2) + I
C
C  COMPUTE THE DIFFERENCE BETWEEN THE ACTUAL OUTPUT AND THE DESIRED
C  OUTPUT
C
            TEMP = NRON(INDX,1) - OUTARY(I,M)
            ER = ER + TEMP**2
C
C  COMPUTE NF05, IF NECESSARY.
C
            IF(TYPE.EQ.2 .AND. ACCEPT .AND. DABS(TEMP).GE.0.5D0) THEN
               NF05 = NF05+1
               ACCEPT = .FALSE.
            END IF
   10    CONTINUE
   20 CONTINUE
C
C  COMPUTE THE SQUARED ERROR PERCENTAGE.
C
      ER = ER * 100/( NUMBER * NUMOUT )
C
      RETURN
      END
      SUBROUTINE GRADIE (F,X,N,G,INDATA,OUDATA,LAYIFO,NEURON,
     *   N1,N2,L1,L2,IN1,INTRAI,INTEST,NOUT)
C-----------------------------------------------------------------------
C     THIS SUBROUTINE IS USED TO COMPUTE THE VALUE AND GRADIENT OF THE
C     ERROR FUNCTION E(W).
C
```

```
C     THE CALLING STATEMENT IS
C         CALL GRADIE (F,X,N,G,INDATA,OUDATA,LAYIFO, NEURON,
C                      N1,N2,L1,L2,IN1,INTRAI,INTEST,NOUT)
C     WHERE
C
C     F        IS A DOUBLE PRECISION VARIABLE THAT CONTAINS THE VALUE
C              OF THE ERROR FUNCTION.
C
C     X        IS A DOUBLE PRECISION ARRAY THAT CONTAINS CORRECT WEIGHTS.
C
C     N        IS AN INTEGER, WHICH IS NUMBER OF WEIGHTS.
C
C     G        IS A DOUBLE PRECISION ARRAY THAT CONTAINS THE
C              GRADIENTS AT THE POINT X.
C
C     INDATA   IS A TWO-DIMENSIONAL ARRAY. EACH COLUMN CONTAINS INPUT
C              DATA OF AN EXAMPLE.
C
C     OUDATA   IS A TWO-DIMENSIONAL ARRAY. EACH COLUMN CONTAINS OUTPUT DA
C              OF AN EXAMPLE.
C
C     LAYIFO   IS A TWO-DIMENSIONAL INTEGER ARRAY THAT CONTAINS SOME
C              NETWORK INFORMATION.
C
C     NEURON   IS A TWO-DIMENSIONAL DOUBLE PRECISION ARRAY, THE FIRST COL
C              OF WHICH CONTAINS OUTPUT OF EACH NEURON, THE SECOND COLUMN
C              CONTAINS THE DERIVATIVE OF THE ACTIVATION FUNCTION, AND TH
C              THIRD COLUMN CONTAINS PARTIAL OF E(W) W.R.T. U(K,J).
C
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER L1,L2,LLL,LLLL,LLLLL,I,J,K,KK,M,NDX1,NOUT1
      INTEGER    LAYIFO(L1,L2), WDX, NDX,N, N1,N2,IN1,
     *   INTRAI,INTEST,NOUT,NTRAIN, NLAYER
      DOUBLE PRECISION    NEURON(N1,N2), OUDATA(NOUT,INTRAI),COEF,
     *   INDATA(IN1,INTRAI),X(N),G(N), F
      LOGICAL    GRD
C
      COMMON /IP/ NTRAIN, NLAYER
C
      NOUT1 = LAYIFO(NLAYER, 1)
C
C  ITERATE OVER ALL TRAINING EXAMPLES
C
      GRD = .TRUE.
      DO 80 M=1, NTRAIN
C
C  COMPUTE THE OUTPUT OF M-TH EXAMPLE.
C
         CALL  COMOU1(INDATA(1,M),NEURON,LAYIFO,X,N,N1,N2,L1,L2,IN1,GRD)
C
         DO 10 I=1,  NOUT1
            NDX = LAYIFO(NLAYER, 2) + I
            NEURON(NDX,3) = NEURON(NDX,1) - OUDATA(I,M)
            F = F + NEURON(NDX, 3 ) ** 2
   10    CONTINUE
C
C  BACKWARD PROPAGATION COMPUTATION OVER ALL LAYERS, STARTS FROM
C  THE LAST HIDDEN LAYER.
C
         LLL=NLAYER-1
         DO 40 KK=2, LLL
            K=LLL+2-KK
C
C  OVER ALL NEURONS IN LAYER K.
```

```
C
            LLLL=LAYIFO(K,1)
            DO 30 J=1, LLLL
               NDX =LAYIFO(K,2) + J
               NEURON(NDX,3) = 0
C
C  OVER ALL NEURONS IN THE NEXT LAYER.
C
               LLLLL=LAYIFO(K+1,1)
               DO 20 I = 1,  LLLLL
                  NDX1 = LAYIFO(K+1,2)+I
                  WDX = LAYIFO(K+1,3) + (I-1)*(LAYIFO(K,1)+1) + J
C
C  COMPUTE PARTIAL OF E(W) W.R.T. U(K,J)
C
                  NEURON(NDX,3) = NEURON(NDX,3) +
     *               NEURON(NDX1,2)*NEURON(NDX1,3)*X(WDX)
   20          CONTINUE
   30       CONTINUE
   40    CONTINUE
C
C  BACKWARD PROPAGATION COMPUTATION OVER ALL LAYERS AGAIN, STARTS
C  FROM OUTPUT LAYER.
C
         DO 70 KK=2,NLAYER
            K=NLAYER+2-KK
C
C  OVER ALL NEURONS IN K-TH LAYER.
C
            LLLL=LAYIFO(K,1)
            DO 60 J=1, LLLL
               NDX = LAYIFO(K,2) + J
C
C  OVER ALL NEURONS IN THE PREVIOUS LAYER.
C
               LLL=LAYIFO(K-1,1)+1
               DO 50 I=1, LLL
                  WDX = LAYIFO(K,3) + (J-1)*(LAYIFO(K-1,1)+1) + I-1
                  NDX1 = LAYIFO(K-1,2) + I-1
C
C  COMPUTE THE PARTIAL OF E(W) W.R.T. W(I,J,K)
C
                  G(WDX)= G(WDX) +
     *               NEURON(NDX,3)*NEURON(NDX,2)*NEURON(NDX1,1)
   50          CONTINUE
   60       CONTINUE
   70    CONTINUE
   80 CONTINUE
C
C  COMPUTE THE SQUARED ERROR PERCENTAGE AND GRADIENT.
C
      COEF = 200.0 / (NTRAIN * NOUT1)
      F = 0.5 * F * COEF
      DO 90 I=1,   N
         G(I) = G(I) * COEF
   90 CONTINUE
C
      RETURN
      END
```

APPENDIX B: PROGRAM LIST FOR DRIVEF.F

```
      PROGRAM DRIVEF
C-----------------------------------------------------------------------
C   THIS IS THE DRIVE TO SOLVE THE FUNCTIONS FROM BRENT'S SUITE OF TEST
C   PROBLEMS AND OSBORNE'S FUNCTIONS BY USING THE LIMITED MEMORY BFGS
C   METHOD.
C
C   REFERENCES:
C         "ALGORITHMS FOR MINIMIZATION WITHOUT DERIVATIVES",
C      RICHARD P. BRENT, PRENTICE-HALL 1973, PAGE 138.
C
C         "SOME ASPECTS OF NON-LINEAR LEAST SQUARES CALCULATIONS"
C      M. R. OSBORNE, NUMERICAL METHODS FOR NON-LINEAR OPTIMIZATION,
C      F. A. LOOTSMA ED., ACADEMIC PRESS, NEW YORK, 1971,171-189.
C
C   THE SUBROUTINE TESTIN AND FUNCTION FTEST WERE WRITTEN BY
C   DR. J. P. CHANDLER, COMPUTER SCIENCE DEPARTMENT,
C   OKLAHOMA STATE UNIVERSITY.
C
C   THE LIMITED MEMORY BFGS METHOD IS IMPLEMENTED BY MEANS OF THE
C   SUBROUTINE MLBFGS, WHICH IS A SLIGHT MODIFICATION OF THE ROUTINE
C   LBFGS BY JORGE NOCEDAL.
C
C   VARIABLES:
c
C     N        IS THE NUMBER OF VARIABLES.
C
C     F        IS THE VALUE OF THE FUNCTION.
C
C     X AND G  ARE ARRAYS OF LENGTH N, WHICH CONTAIN THE VALUE OF THE
C              VARIABLES AND THE GRADIENT AT THE POINT X, RESPECTIVELY.
C
C     Y1,Y2,AND T ARE ARRAYS USED TO TEST OSBORNE FUNCTIONS.
C
C     EPS      TOLERANCE OF THE STOPPING CRITERIA OF SUBROUTINE MLBFGS.
C              THE MLBFGS TERMINATES WHEN ||G|| < EPS * MAX(1, ||X||).
C
C     M        IS THE NUMBER OF CORRECTIONS USED IN THE BFGS UPDATE.
C
C     ICALL    IS THE NUMBER OF EVALUATIONS OF F AND G.
C
C     ANN      IS A LOGICAL VARIABLE. SINCE THE SUBROUTINE MLBFGS
C              IS CALLED BY TWO DRIVERS, DRIVE IS NOT USED TO IMPLEMENT
C              THE ARTIFICIAL NEURAL NETWORKS, THEREFORE SET
C              ANN = FALSE; OTHERWISE SET ANN = TRUE.
C
C     OTHER VARIABLES AND PARAMETERS ARE DESCRIBED IN THE
C     SUBROUTINES LBSET AND MLBFGS.
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      DOUBLE PRECISION X(2000),G(2000),DIAG(2000),W(35000),
     *                 YY1(33),YY2(65),Y1,Y2,T,TOLERA
      DOUBLE PRECISION F,EPS,XTOL,GTOL,STPMIN,STPMAX,PI,FTEST
      INTEGER IPRINT(2),IFLAG,ICALL,N,M,MP,LP,J,JFUNC,MAXJF,JFF,NFMAX
C
      DATA YY1/0.844D0,0.908D0,0.932D0,0.936D0,0.925D0,0.908D0,0.881D0,
     *        0.850D0,0.818D0,0.784D0,0.751D0,0.718D0,0.685D0,0.658D0,
     *        0.628D0,0.603D0,0.580D0,0.558D0,0.538D0,0.522D0,0.506D0,
     *        0.490D0,0.478D0,0.467D0,0.457D0,0.448D0,0.438D0,0.431D0,
     *        0.424D0,0.420D0,0.414D0,0.411D0,0.406D0/
      DATA YY2/1.366D0,1.191D0,1.112D0,1.013D0,0.991D0,0.885D0,0.831D0,
     *        0.847D0,0.786D0,0.725D0,0.746D0,0.679D0,0.608D0,0.655D0,
```

```
     *        0.616D0,0.606D0,0.602D0,0.626D0,0.651D0,0.724D0,0.649D0,
     *        0.649D0,0.694D0,0.644D0,0.624D0,0.661D0,0.612D0,0.558D0,
     *        0.533D0,0.495D0,0.500D0,0.423D0,0.395D0,0.375D0,0.372D0,
     *        0.391D0,0.396D0,0.405D0,0.428D0,0.429D0,0.523D0,0.562D0,
     *        0.607D0,0.653D0,0.672D0,0.708D0,0.633D0,0.668D0,0.645D0,
     *        0.632D0,0.591D0,0.559D0,0.597D0,0.625D0,0.739D0,0.710D0,
     *        0.729D0,0.720D0,0.636D0,0.581D0,0.428D0,0.292D0,0.162D0,
     *        0.098D0,0.054D0/
      LOGICAL DIAGCO, ANN
      DATA XTOL, EPS/1.0D-16,1.0D-7/
C
      COMMON /LB3/MP,LP,GTOL,STPMIN,STPMAX
      COMMON /PRTEST/ PI,JFUNC,MAXJF
      COMMON /OSB/Y1(33),Y2(65),T(65)
C
      M=5
      IPRINT(1)= 50
      IPRINT(2)= 0
C
C  WE DO NOT WISH TO PROVIDE THE DIAGONAL MATRICES HK0, AND
C  THEREFORE SET DIAGCO TO FALSE.
C
      DIAGCO= .FALSE.
C
C  WE ARE NOT IMPLEMENTING THE ARTIFICIAL NEURAL NETWORKS LEARNING
C  ALGORITHM, THEREFORE SET ANN TO FALSE.
C
      ANN = .FALSE.
      NFMAX = 2000
      TOLERA = 1.0D-3
C
C  DEFINE THE DEFAULT VALUES OF SEVERAL PARAMETERS IN COMMON SECTIONS.
C
      CALL LBSET
      DO 5 I=1, 33
         Y1(I)=YY1(I)
   5  CONTINUE
      DO 6 I=1, 65
         Y2(I)=YY2(I)
   6  CONTINUE
C
      DO 50 JFF=1,13
         ICALL=0
         IFLAG=0
         JFUNC=JFF
         IF(JFUNC .LT. 12)  THEN
            WRITE(LP,10)JFUNC
  10        FORMAT(/' SOLVE PROBLEM NUMBER',I3,
     *             ' FROM THE TEST SUITE USED BY BRENT.')
         END IF
C
         IF(JFF.EQ.6) N=9
         IF(JFF.EQ.9) N=10
         IF(JFF.EQ.10) N=20
C
C  INITIALIZATION
C
         CALL TESTIN(X,N)
C
  20     CONTINUE
         F= 0.D0
         DO 30 J=1,N
            G(J)=0.D0
  30     CONTINUE
C
```

```
C  CALCULATE THE FUNCTION F AND GRADIENT G.
C
         IF(JFUNC .LT. 12)  THEN
            F = FTEST(X,N)
            CALL FGRAD(N,X,G)
         ELSE
            CALL OSBORNE(F,X,N,G)
         END IF
C
         CALL MLBFGS(N,M,X,F,G,DIAGCO,DIAG,IPRINT,EPS,XTOL,W,IFLAG,
     *                ANN,TOLERA)
C
         IF(IFLAG.LE.0) GO TO 35
         ICALL=ICALL + 1
C
C  WE ALLOW AT MOST NFMAX EVALUATIONS OF F AND G
C
         IF(ICALL.GT.NFMAX) GO TO 35
         GO TO 20
C
  35     WRITE(LP,40)
         WRITE(LP,45)  (X(I), I=1,N)
  40     FORMAT(' VACTOR X= ')
  45     FORMAT(6(2X,1PD10.3))
  50  CONTINUE
C
      END
C
      SUBROUTINE TESTIN(X,N)
C-----------------------------------------------------------------------
C  TESTIN 1.2            SEPTEMBER 1995
C
C  J. P. CHANDLER, COMPUTER SCIENCE DEPARTMENT,
C  OKLAHOMA STATE UNIVERSITY
C
C  INITILIZE FOR PROBLEM NUMBER JFUNC FROM BRENT'S TEST SUITE.
C
C  "ALGORITHMS FOR MINIMIZATION WITHOUT DERIVATIVES",
C  RICHARD P. BRENT, PRENTICE-HALL 1973, PAGE 138
C
C  CALL SUBROUTINE LBSET BEFORE CALLING TESTIN.
C
C  NOTE THAT FOR JFUNC = 6, 9, OR 10,
C  THE VALUE OF N MUST BE SET BEFORE CALLING TESTIN.
C
C-----------------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER N,LP,MP,JFUNC,MAXJF,J
      DOUBLE PRECISION X(N),PI,DATAN,GTOL,STPMIN,STPMAX,Y1,Y2,T
C
      COMMON /PRTEST/ PI,JFUNC,MAXJF
      COMMON /LB3/MP,LP,GTOL,STPMIN,STPMAX
      COMMON /OSB/Y1(33),Y2(65),T(65)
C
      MAXJF=13
C
      PI=4.0D0*DATAN(1.0D0)
C
      IF(JFUNC.LT.1 .OR. JFUNC.GT.MAXJF) STOP
C
      GO TO (10,30,50,70,90,110,170,190,210,240,270,300,330),JFUNC
C
C  JFUNC=1
C  ROSENBROCK'S TEST FUNCTION
C
```

```
C     THE MINIMUM IS F(1.0,1.0)=0.0 .
C
   10 N=2
      X(1)=-1.2D0
      X(2)=1.0D0
      WRITE(LP,20)
   20 FORMAT(' ROSENBROCK TEST FUNCTION')
      RETURN
C
C  JFUNC=2
C  POWELL'S SINGULAR TEST FUNCTION
C
C     THE MINIMUM IS F(0.0,0.0,0.0,0.0)=0.0 .
C
   30 N=4
      X(1)=3.0D0
      X(2)=-1.0D0
      X(3)=0.0D0
      X(4)=1.0D0
      WRITE(LP,40)
   40 FORMAT(' SINGULAR TEST FUNCTION OF POWELL')
      RETURN
C
C  JFUNC=3
C  HELICAL VALLEY TEST FUNCTION OF FLETCHER AND POWELL
C  SINCE THE GRADIENT DOES NOT EXIST AT THE POINT (0,0,0), THEREFORE
C  WE CHANGE THE INITIAL VALUE FROM (-1,0,0) TO (0.01,0.01,0)
C     THE MINIMUM IS F(1.0,0.0,0.0)=0.0 .
C
   50 N=3
      X(1)=0.01D0
      X(2)=0.01D0
      X(3)=0.0D0
      WRITE(LP,60)
   60 FORMAT(' HELICAL VALLEY TEST FUNCTION OF FLETCHER AND POWELL')
      RETURN
C
C  JFUNC=4
C  LEON'S CUBIC TEST FUNCTION
C
C     THE MINIMUM IS F(1.0,1.0)=0.0 .
C
   70 N=2
      X(1)=-1.2D0
      X(2)=-1.0D0
      WRITE(LP,80)
   80 FORMAT(' CUBIC TEST FUNCTION OF LEON')
      RETURN
C
C  JFUNC=5
C  BEALE'S TEST FUNCTION
C
C     THE MINIMUM IS F(3.0,0.5)=0.0 .
C
   90 N=2
      X(1)=0.1D0
      X(2)=0.1D0
      WRITE(LP,100)
  100 FORMAT(' BEALE TEST FUNCTION')
      RETURN
C
C  JFUNC=6
C  WATSON'S TEST FUNCTION (SEE KOWALIK AND OSBORNE)
C
C     FOR N=6, THE MINIMUM IS NEAR
```

```
C        F(-0.015725,1.012435,-0.232992,1.260430,-1.513729,0.992996)=
C           2.28767005355D-3 .
C
C     FOR N=9, THE MINIMUM IS NEAR
C       F(-0.000015,0.999790,0.014764,0.146342,1.00081,-2.617731,
C          4.104403,-3.143612,1.052627)=1.399760138D-6 .
C
  110 DO 120 J=1,N
         X(J)=0.0D0
  120 CONTINUE
      WRITE(LP,130)N
  130 FORMAT(' WATSON TEST FUNCTION WITH N =',I3)
      RETURN
C
C  JFUNC=7
C  POWELL'S 1964 TEST FUNCTION
C
C     THE MINIMUM IS F(1.0,1.0,1.0)=0.0 .
C
  170 N=3
      X(1)=0.0D0
      X(2)=1.0D0
      X(3)=2.0D0
      WRITE(LP,180)
  180 FORMAT(' POWELL (1964) TEST FUNCTION')
      RETURN
C
C  JFUNC=8
C  WOOD'S TEST FUNCTION
C
C     THE MINIMUM IS F(1.0,1.0,1.0,1.0)=0.0 .
C
  190 N=4
      X(1)=-3.0D0
      X(2)=-1.0D0
      X(3)=-3.0D0
      X(4)=-1.0D0
      WRITE(LP,200)
  200 FORMAT(' TEST FUNCTION OF WOOD')
      RETURN
C
C  JFUNC=9
C  BRENT'S HILBERT MATRIX TEST FUNCTION
C
C     THE MINIMUM IS F(0.0,0.0,0.0,...,0.0)=0.0 .
C
C     FOR N.GT.10, PRAXIS MAY RUN FOR A VERY, VERY LONG TIME
C        AND TERMINATE WITH SOME COMPONENTS OF X(*) FAR FROM ZERO,
C        BECAUSE OF THE EXTREME ILL-CONDITIONING OF THIS PROBLEM.
C
C  SHOULDN'T ILLCIN BE SET TO 1 FOR THESE FUNCTIONS,
C  AT LEAST FOR LARGE VALUES OF N?
C
  210 DO 220 J=1,N
         X(J)=1.0D0
  220 CONTINUE
      WRITE(LP,230)N
  230 FORMAT(' HILBERT MATRIX TEST FUNCTION WITH N =',I3)
      RETURN
C
C  JFUNC=10
C  TRIDIAGONAL MATRIX TEST FUNCTION
C
C     THE MINIMUM IS F(N,N-1,N-2,...,2,1) = -N .
C
```

```
  240 DO 250 J=1,N
         X(J)=0.0D0
  250 CONTINUE
      WRITE(LP,260)N
  260 FORMAT(' TRIDIAGONAL MATRIX TEST FUNCTION WITH N =',I3)
      RETURN
C
C  JFUNC=11
C  BOX'S TEST FUNCTION
C
C     THE MINIMUM IS F(1.0,10.0,1.0)=0.0 .
C
  270 N=3
      X(1)=0.0D0
      X(2)=10.0D0
      X(3)=20.0D0
      WRITE(LP,280)
  280 FORMAT(' TEST FUNCTION OF BOX')
      RETURN
C
C  JFUNC=12
C  OSBORNE 1 FUNCTION
C
C     THE MINIMUM IS F(0.3754,1.9358,-1.4647,0.01287,
C                      0.02212)=0.546D-4
C
  300 N=5
      X(1)=0.5D0
      X(2)=1.5D0
      X(3)=-1.0D0
      X(4)=1.0D-2
      X(5)=2.0D-2
      DO 310 J=1, 33
         T(J)=10.0D0*(J-1)
  310 CONTINUE
      WRITE(LP,320)
  320 FORMAT(/' TEST FUNCTION OF OSBORNE 1')
      RETURN
C
C  JFUNC=13
C  OSBORNE 2 FUNCTION
C
C     THE MINIMUM IS F(1.3100,0.4315,0.6336,0.5993,0.7539,0.9056,
C                      1.3651,4.8248,2.3988,4.5689,5.6754)=0.0402
C
  330 N=11
      X(1)=1.3D0
      X(2)=6.5D-1
      X(3)=6.5D-1
      X(4)=7.0D-1
      X(5)=6.0D-1
      X(6)=3.0D0
      X(7)=5.0D0
      X(8)=7.0D0
      X(9)=2.0D0
      X(10)=4.5D0
      X(11)=5.5D0
      DO 340 J=1, 65
         T(J)=0.1D0*(J-1)
  340 CONTINUE
      WRITE(LP,350)
  350 FORMAT(/' TEST FUNCTION OF OSBORNE 2')
      RETURN
      END
C
```

```
      DOUBLE PRECISION FUNCTION FTEST(X,N)
C-----------------------------------------------------------------
C FTEST 1.2            SEPTEMBER 1995
C
C  J. P. CHANDLER, Computer Science Department,
C  Oklahoma State University
C
C  COMPUTE THE VALUE OF FUNCTION NUMBER JFUNC
C  FROM BRENT'S SUITE OF TEST PROBLEMS.
C
C  "ALGORITHMS FOR MINIMIZATION WITHOUT DERIVATIVES",
C  RICHARD P. BRENT, PRENTICE-HALL 1973, PAGES 137-154, 164-166
C-----------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER N,JFUNC,MAXJF, I,IMAX,J,JEVEN,JJ,JJMAX
      DOUBLE PRECISION X(20),Y,PI,DATAN,DEXP,
     *                 DSIN,DSQRT,F,P,R,S,T,TERM,U,YY
C
      COMMON /PRTEST/ PI,JFUNC,MAXJF
C
      IF(JFUNC.LT.1 .OR. JFUNC.GT.MAXJF) STOP
      GO TO (10,20,30,40,50,60,140,150,160,190,210),JFUNC
C
C  JFUNC=1
C  ROSENBROCK'S TEST FUNCTION
C
   10 FTEST=100.0D0*(X(2)-X(1)**2)**2+(1.0D0-X(1))**2
      RETURN
C
C  JFUNC=2
C  POWELL'S SINGULAR TEST FUNCTION
C
   20 FTEST=(X(1)+10.0D0*X(2))**2+5.0D0*(X(3)-X(4))**2+
     *      (X(2)-2.0D0*X(3))**4+10.0D0*(X(1)-X(4))**4
      RETURN
C
C  JFUNC=3
C  HELICAL VALLEY TEST FUNCTION OF FLETCHER AND POWELL
C
   30 R=DSQRT(X(1)**2+X(2)**2)
C
      IF(X(1).EQ.0.0D0) THEN
         T=0.25D0
      ELSE
         T=DATAN(X(2)/X(1))/(2.0D0*PI)
      ENDIF
C
      IF(X(1).LT.0.0D0) T=T+0.5D0
      FTEST=100.0D0*((X(3)-10.0D0*T)**2+(R-1.0D0)**2)+X(3)**2
      RETURN
C
C  JFUNC=4
C  LEON'S CUBIC TEST FUNCTION
C
   40 FTEST=100.0D0*(X(2)-X(1)**3)**2+(1.0D0-X(1))**2
      RETURN
C
C  JFUNC=5
C  BEALE'S TEST FUNCTION
C
   50 FTEST=(1.5D0-X(1)*(1.0D0-X(2)))**2+
     *      (2.25D0-X(1)*(1.0D0-X(2)**2))**2+
     *      (2.625D0-X(1)*(1.0D0-X(2)**3))**2
      RETURN
C
```

```
C  JFUNC=6
C  WATSON'S TEST FUNCTION (SEE KOWALIK AND OSBORNE)
C
   60 S=X(1)**2+(X(2)-X(1)**2-1.0D0)**2
      DO 90 I=2,30
         YY=(I-1)/29.0D0
         T=X(N)
         JJMAX=N-1
         DO 70 JJ=1,JJMAX
            J=JJMAX+1-JJ
            T=X(J)+YY*T
   70    CONTINUE
         U=(N-1)*X(N)
C
         DO 80 JJ=2,JJMAX
            J=JJMAX+2-JJ
            U=(J-1)*X(J)+YY*U
   80    CONTINUE
         S=S+(U-T*T-1.0D0)**2
   90 CONTINUE
      FTEST=S
      RETURN
C
C  JFUNC=7
C  POWELL'S 1964 TEST FUNCTION
C
  140 IF(X(2).EQ.0.0D0) THEN
         TERM=0.0D0
      ELSE
         TERM=DEXP(-((X(1)+X(3))/X(2)-2.0D0)**2)
      ENDIF
      FTEST=3.0D0-1.0D0/(1.0D0+(X(1)-X(2))**2)-
     *   DSIN(0.5D0*PI*X(2)*X(3))-TERM
      RETURN
C
C  JFUNC=8
C  WOOD'S TEST FUNCTION
C
  150 FTEST=100.0D0*(X(2)-X(1)**2)**2+(1.0D0-X(1))**2+
     *   90.0D0*(X(4)-X(3)**2)**2+(1.0D0-X(3))**2+
     *   10.1D0*((X(2)-1.0D0)**2+(X(4)-1.0D0)**2)+
     *   19.8D0*(X(2)-1.0D0)*(X(4)-1.0D0)
      RETURN
C
C  JFUNC=9
C  BRENT'S HILBERT MATRIX TEST FUNCTION
C
  160 S=0.0D0
      DO 180 I=1,N
         T=0.0D0
         DO 170 J=1,N
            T=T+X(J)/(I+J-1.0D0)
  170    CONTINUE
         S=S+T*X(I)
  180 CONTINUE
      FTEST=S
      RETURN
C
C  JFUNC=10
C  TRIDIAGONAL MATRIX TEST FUNCTION
C
  190 S=X(1)*(X(1)-X(2))
      IMAX=N-1
      DO 200 I=2,IMAX
         S=S+X(I)*((X(I)-X(I-1))+(X(I)-X(I+1)))
```

```
  200 CONTINUE
      FTEST=S+X(N)*(2.0D0*X(N)-X(N-1))-2.0D0*X(1)
      RETURN
C
C  JFUNC=11
C  BOX'S TEST FUNCTION
C
  210 S=0.0D0
      DO 220 I=1,10
         P=-I/10.0D0
         IF(P*X(2).LT.-40.0D0) THEN
            TERM=0.0D0
         ELSE
            TERM=DEXP(P*X(2))
         ENDIF
         S=S+(DEXP(P*X(1))-TERM-
     *      X(3)*(DEXP(P)-DEXP(10.0D0*P)))**2
  220 CONTINUE
      FTEST=S
      RETURN
      END
C
      SUBROUTINE FGRAD(N,X,G)
C-----------------------------------------------------------------
C  COMPUTE THE GRADIENT OF FUNCTION NUMBER JFUNC
C  FROM BRENT'S SUITE OF TEST PROBLEMS.
C
C  "ALGORITHMS FOR MINIMIZATION WITHOUT DERIVATIVES",
C  RICHARD P. BRENT, PRENTICE-HALL 1973, PAGES 137-154, 164-166
C-----------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER N,JFUNC,MAXJF, I,IMAX,J,JEVEN,JJ,JJMAX

      DOUBLE PRECISION X(N),G(N),V,PI,DATAN,DEXP,
     *                 DCOS,DSQRT,F,P,R,S,T(4),C(3),TERM,U,YY
C
      COMMON /PRTEST/ PI,JFUNC,MAXJF
C
      IF(JFUNC.LT.1 .OR. JFUNC.GT.MAXJF) STOP
      GO TO (10,20,30,40,50,60,140,150,160,190,210),JFUNC
C
C  JFUNC=1
C  ROSENBROCK'S TEST FUNCTION
  10  G(2)=2.0D2*(X(2)-X(1)**2)
      G(1)=-2.0D0*(X(1)*G(2)+1.0D0-X(1))
      RETURN
C
C  JFUNC=2
C  POWELL'S SINGULAR TEST FUNCTION
C
  20  T(1)=2*(X(1)+10*X(2))
      T(2)=40*(X(1)-X(4))**3
      T(3)=4*(X(2)-2*X(3))**3
      T(4)=10*(X(3)-X(4))
      G(1)=T(1)+T(2)
      G(2)=10*T(1)+T(3)
      G(3)=T(4)-2*T(3)
      G(4)=-T(2)-T(4)
      RETURN
C
C  JFUNC=3
C  HELICAL VALLEY TEST FUNCTION OF FLETCHER AND POWELL
C
  30  U=X(1)**2+X(2)**2
      R=DSQRT(U)
```

```
      R=2.0D2*(R-1)/R
      S=DATAN(X(2)/X(1))
      S=2.0D2*(X(3)-5.0D0*S/PI)
      G(3)=S+2.0D0*X(3)
      S=5.0D0*S/(U*PI)
      G(2)=-X(1)*S+R*X(2)
      G(1)=X(2)*S+R*X(1)
      RETURN
C
C  JFUNC=4
C  LEON'S CUBIC TEST FUNCTION
C
  40  G(2)=2.0D2*(X(2)-X(1)**3)
      G(1)=-3*G(2)*X(1)**2-2*(1-X(1))
      RETURN
C
C  JFUNC=5
C  BEALE'S TEST FUNCTION
C
  50  C(1)=1.5D0
      C(2)=2.25D0
      C(3)=2.625D0
      DO 52 I=1,3
         T(I)=1-X(2)**I
  52  CONTINUE
      DO 55 I=1, 3
         G(1)=G(1)-(C(I)-X(1)*T(I))*T(I)
         G(2)=G(2)+(C(I)-X(1)*T(I))*I*X(1)*X(2)**(I-1)
  55  CONTINUE
      G(1)=2*G(1)
      G(2)=2*G(2)
      RETURN
C
C  JFUNC=6
C  WATSON'S TEST FUNCTION (SEE KOWALIK AND OSBORNE)
C
  60  DO 100 I=2, 30
         YY=(I-1)/29.0D0
         V=X(N)
         JJMAX=N-1
         DO 70 JJ=1,JJMAX
            J=JJMAX+1-JJ
            V=X(J)+YY*V
  70     CONTINUE
         U=(N-1)*X(N)
C
         DO 80 JJ=2,JJMAX
            J=JJMAX+2-JJ
            U=(J-1)*X(J)+YY*U
  80     CONTINUE
C
         U=2.0D0*(U-V*V-1.0D0)
         R=1.0D0
C
         DO 90 JJ=2, N
            G(JJ)=G(JJ)+U*((JJ-1)*R-2.0D0*V*R*YY)
            R=R*YY
  90     CONTINUE
         G(1)=G(1)-U*2.0D0*V
 100  CONTINUE
C
      R=X(2)-X(1)*X(1)-1.0D0
      G(1)=G(1)+2.0D0*X(1)*(1.0D0-2.0D0*R)
      G(2)=G(2)+2.0D0*R
      RETURN
```

```
C
C  JFUNC=7
C  POWELL'S 1964 TEST FUNCTION
C
 140  S=(X(1)+X(3))/X(2)-2.0D0
      R=(X(1)-X(2))/((1.0D0+(X(1)-X(2))**2)**2)
      U=S*DEXP(-S*S)/X(2)
      P=0.5D0*PI
      V=P*DCOS(P*X(2)*X(3))
      G(1)=2.0D0*(R+U)
      G(2)=-2.0D0*(R+U*(X(1)+X(3))/X(2))-X(3)*V
      G(3)=2.0D0*U-X(2)*V
      RETURN
C
C  JFUNC=8
C  WOOD'S TEST FUNCTION
C
 150  T(1)=X(2)-X(1)*X(1)
      T(2)=X(4)-X(3)*X(3)
      G(1)=-4.0D2*T(1)*X(1)-2.0D0*(1.0D0-X(1))
      G(2)=2.0D2*T(1)+2.0D0*10.1D0*(X(2)-1.0D0)+19.8D0*(X(4)-1.0D0)
      G(3)=-3.6D2*T(2)*X(3)-2.0D0*(1.0D0-X(3))
      G(4)=1.8D2*T(2)+2.0D0*10.1D0*(X(4)-1.0D0)+19.8D0*(X(2)-1.0D0)
      RETURN
C
C  JFUNC=9
C  BRENT'S HILBERT MATRIX TEST FUNCTION
C
 160  DO 180 I=1, N
         DO 170 J=1, N
            G(I)=G(I)+2.0D0*X(J)/(I+J-1)
 170     CONTINUE
 180  CONTINUE
      RETURN
C
C  JFUNC=10
C  TRIDIAGONAL MATRIX TEST FUNCTION
C
 190  G(1)=2*(X(1)-X(2)-1)
      DO 200 I=2, N-1
         G(I)=2.0D0*(2.0D0*X(I)-X(I-1)-X(I+1))
 200  CONTINUE
      G(N)=2*(2*X(N)-X(N-1))
      RETURN
C
C  JFUNC=11
C  BOX'S TEST FUNCTION
C
 210  DO 240 I=1, 10
         P=-I/10.0D0
         T(1)=DEXP(X(1)*P)
         T(2)=-DEXP(X(2)*P)
         T(3)=DEXP(1.0D1*P)-DEXP(P)
         S=2*(T(1)+T(2)+X(3)*T(3))
         G(1)=G(1)+S*T(1)*P
         G(2)=G(2)+S*T(2)*P
         G(3)=G(3)+S*T(3)
 240  CONTINUE
      RETURN
      END
C
      SUBROUTINE OSBORNE(F,X,N,G)
C
C  COMPUTE THE VALUES AND THE GRADIENTS FOR OSBORNE FUNCTIONS 1 AND 2.
C
```

```
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER N,JFUNC,MAXJF,J,I
      DOUBLE PRECISION X(N),G(N),DEXP,FTX,F,R,S,TEMP(4),PI,Y1,Y2,T
C
      COMMON /PRTEST/ PI,JFUNC,MAXJF
      COMMON /OSB/Y1(33),Y2(65),T(65)
C
      IF(JFUNC.LT.12 .OR. JFUNC.GT.13) STOP
      F=0.0D0
      DO 10 I=1, N
         G(I)=0.0D0
   10 CONTINUE
C
C  JFUNC=12
C  OSBORNE FUNCTION 1
C
      IF(JFUNC .EQ. 12) THEN
         DO 35 J=1,33
            R=DEXP((-1)*X(4)*T(J))
            S=DEXP((-1)*X(5)*T(J))
            FTX=X(1)+X(2)*R+X(3)*S-Y1(J)
            F = F+FTX**2
            G(1)=G(1)+FTX
            G(2)=G(2)+FTX*R
            G(3)=G(3) + FTX*S
            G(4)=G(4) - FTX*X(2)*T(J)*R
            G(5)=G(5) - FTX*X(3)*T(J)*S
   35    CONTINUE
C
C  JFUNC=13
C  OSBORNE FUNCTION 2
C
      ELSE
         DO 45 J=1, 65
            TEMP(1)=DEXP(-X(5)*T(J))
            TEMP(2)=DEXP(-X(6)*(T(J)-X(9))**2)
            TEMP(3)=DEXP(-X(7)*(T(J)-X(10))**2)
            TEMP(4)=DEXP(-X(8)*(T(J)-X(11))**2)
            FTX=0.D0
            DO 40 I=1, 4
               FTX=FTX+X(I)*TEMP(I)
   40       CONTINUE
            FTX=FTX-Y2(J)
            F = F+FTX**2
            DO 42 I=1, 4
               G(I)=G(I)+FTX*TEMP(I)
   42       CONTINUE
            G(5)=G(5)-FTX*T(J)*X(1)*TEMP(1)
            G(6)=G(6)-FTX*X(2)*TEMP(2)*((T(J)-X(9))**2)
            G(7)=G(7)-FTX*X(3)*TEMP(3)*((T(J)-X(10))**2)
            G(8)=G(8)-FTX*X(4)*TEMP(4)*((T(J)-X(11))**2)
            G(9)=G(9)+FTX*X(2)*TEMP(2)*X(6)*2*(T(J)-X(9))
            G(10)=G(10)+FTX*X(3)*TEMP(3)*X(7)*2*(T(J)-X(10))
            G(11)=G(11)+FTX*X(4)*TEMP(4)*X(8)*2*(T(J)-X(11))
   45    CONTINUE
      END IF
      DO 49 J=1, N
         G(J)=2*G(J)
   49 CONTINUE
C
      END
```

APPENDIX C: PROGRAM LIST FOR MLBFGS.F

```
C     ------------------------------------------------------------------
C     THIS SUBROUTINE IS USED TO IMPLEMENT THE LIMITED MEMORY
C     BFGS METHOD, WHICH IS VERY SLIGHT MODIFICATION OF THE ROUTINE
C     LBFGS WRITTEN BY JORGE NOCEDAL.
C     THE MODIFICATIONS ARE:
C     1. REPLACE THE BLOCK DATA LB2 BY THE SUBROUTINE LBSET.
C     2. ADD A LOGICAL VARIABLE ANN IN THE CALLING STATEMENT, SO THAT
C        IT CAN BE CALLED BY DIFFERENT DRIVERS, AND USES DIFFERENT
C        STOPPING CRITERIA.
C     3. IF THE SUBROUTINE IS USED TO IMPLEMENT THE NEURAL NETWORK
C        LEARNING ALGORITHM, THEN ANN = TRUE, WE ADD A NEW STOPING
C        CRITEIRIA,I.E., THE SUBROUTINE TERMINATES
C        WHEN ||WK+1 - WK||<TOLERA.
C
C     ****************
C     LBFGS SUBROUTINE
C     ****************
C
      SUBROUTINE MLBFGS(N,M,X,F,G,DIAGCO,DIAG,IPRINT,EPS,XTOL,
     *                  W,IFLAG,ANN,TOLERA)
C
      INTEGER N,M,IPRINT(2),IFLAG
      DOUBLE PRECISION X(N),G(N),DIAG(N),W(1),F,EPS,XTOL
      LOGICAL DIAGCO,ANN
C
C        LIMITED MEMORY BFGS METHOD FOR LARGE SCALE OPTIMIZATION
C                          JORGE NOCEDAL
C                        *** JULY 1990 ***
C
C
C     THIS SUBROUTINE SOLVES THE UNCONSTRAINED MINIMIZATION PROBLEM
C
C                      MIN F(X),    X= (X1,X2,...,XN),
C
C      USING THE LIMITED MEMORY BFGS METHOD. THE ROUTINE IS ESPECIALLY
C      EFFECTIVE ON PROBLEMS INVOLVING A LARGE NUMBER OF VARIABLES. IN
C      A TYPICAL ITERATION OF THIS METHOD AN APPROXIMATION HK TO THE
C      INVERSE OF THE HESSIAN IS OBTAINED BY APPLYING M BFGS UPDATES TO
C      A DIAGONAL MATRIX HK0, USING INFORMATION FROM THE PREVIOUS M STEP
C      THE USER SPECIFIES THE NUMBER M, WHICH DETERMINES THE AMOUNT OF
C      STORAGE REQUIRED BY THE ROUTINE. THE USER MAY ALSO PROVIDE THE
C      DIAGONAL MATRICES HK0 IF NOT SATISFIED WITH THE DEFAULT CHOICE.
C      THE ALGORITHM IS DESCRIBED IN "ON THE LIMITED MEMORY BFGS METHOD
C      FOR LARGE SCALE OPTIMIZATION", BY D. LIU AND J. NOCEDAL,
C      MATHEMATICAL PROGRAMMING B 45 (1989) 503-528.
C
C      THE USER IS REQUIRED TO CALCULATE THE FUNCTION VALUE F AND ITS
C      GRADIENT G. IN ORDER TO ALLOW THE USER COMPLETE CONTROL OVER
C      THESE COMPUTATIONS, REVERSE  COMMUNICATION IS USED. THE ROUTINE
C      MUST BE CALLED REPEATEDLY UNDER THE CONTROL OF THE PARAMETER
C      IFLAG.
C
C      THE STEPLENGTH IS DETERMINED AT EACH ITERATION BY MEANS OF THE
C      LINE SEARCH ROUTINE MCVSRCH, WHICH IS A SLIGHT MODIFICATION OF
C      THE ROUTINE CSRCH WRITTEN BY MORE' AND THUENTE.
C
C      THE CALLING STATEMENT IS
C
C          CALL LBFGS(N,M,X,F,G,DIAGCO,DIAG,IPRINT,EPS,XTOL,W,IFLAG,
C                     ANN,TOLERA)
```

```
C
C      WHERE
C
C     N        IS AN INTEGER VARIABLE THAT MUST BE SET BY THE USER TO THE
C              NUMBER OF VARIABLES. IT IS NOT ALTERED BY THE ROUTINE.
C              RESTRICTION: N>0.
C
C     M        IS AN INTEGER VARIABLE THAT MUST BE SET BY THE USER TO
C              THE NUMBER OF CORRECTIONS USED IN THE BFGS UPDATE. IT
C              IS NOT ALTERED BY THE ROUTINE. VALUES OF M LESS THAN 3 ARE
C              NOT RECOMMENDED; LARGE VALUES OF M WILL RESULT IN EXCESSIV
C              COMPUTING TIME. 3<= M <=7 IS RECOMMENDED. RESTRICTION: M>0
C
C     X        IS A DOUBLE PRECISION ARRAY OF LENGTH N. ON INITIAL ENTRY
C              IT MUST BE SET BY THE USER TO THE VALUES OF THE INITIAL
C              ESTIMATE OF THE SOLUTION VECTOR. ON EXIT WITH IFLAG=0, IT
C              CONTAINS THE VALUES OF THE VARIABLES AT THE BEST POINT
C              FOUND (USUALLY A SOLUTION).
C
C     F        IS A DOUBLE PRECISION VARIABLE. BEFORE INITIAL ENTRY AND O
C              A RE-ENTRY WITH IFLAG=1, IT MUST BE SET BY THE USER TO
C              CONTAIN THE VALUE OF THE FUNCTION F AT THE POINT X.
C
C     G        IS A DOUBLE PRECISION ARRAY OF LENGTH N. BEFORE INITIAL
C              ENTRY AND ON A RE-ENTRY WITH IFLAG=1, IT MUST BE SET BY
C              THE USER TO CONTAIN THE COMPONENTS OF THE GRADIENT G AT
C              THE POINT X.
C
C     DIAGCO   IS A LOGICAL VARIABLE THAT MUST BE SET TO .TRUE. IF THE
C              USER  WISHES TO PROVIDE THE DIAGONAL MATRIX HK0 AT EACH
C              ITERATION. OTHERWISE IT SHOULD BE SET TO .FALSE., IN WHICH
C              CASE  LBFGS WILL USE A DEFAULT VALUE DESCRIBED BELOW. IF
C              DIAGCO IS SET TO .TRUE. THE ROUTINE WILL RETURN AT EACH
C              ITERATION OF THE ALGORITHM WITH IFLAG=2, AND THE DIAGONAL
C               MATRIX HK0  MUST BE PROVIDED IN THE ARRAY DIAG.
C
C     DIAG     IS A DOUBLE PRECISION ARRAY OF LENGTH N. IF DIAGCO=.TRUE.,
C              THEN ON INITIAL ENTRY OR ON RE-ENTRY WITH IFLAG=2, DIAG
C              IT MUST BE SET BY THE USER TO CONTAIN THE VALUES OF THE
C              DIAGONAL MATRIX HK0.  RESTRICTION: ALL ELEMENTS OF DIAG
C              MUST BE POSITIVE.
C
C     IPRINT   IS AN INTEGER ARRAY OF LENGTH TWO WHICH MUST BE SET BY THE
C              USER.
C
C              IPRINT(1) SPECIFIES THE FREQUENCY OF THE OUTPUT:
C                 IPRINT(1) < 0 : NO OUTPUT IS GENERATED,
C                 IPRINT(1) = 0 : OUTPUT ONLY AT FIRST AND LAST ITERATION
C                 IPRINT(1) > 0 : OUTPUT EVERY IPRINT(1) ITERATIONS.
C
C              IPRINT(2) SPECIFIES THE TYPE OF OUTPUT GENERATED:
C                 IPRINT(2) = 0 : ITERATION COUNT, NUMBER OF FUNCTION
C                                 EVALUATIONS, FUNCTION VALUE, NORM OF TH
C                                 GRADIENT, AND STEPLENGTH,
C                 IPRINT(2) = 1 : SAME AS IPRINT(2)=0, PLUS VECTOR OF
C                                 VARIABLES AND  GRADIENT VECTOR AT THE
C                                 INITIAL POINT,
C                 IPRINT(2) = 2 : SAME AS IPRINT(2)=1, PLUS VECTOR OF
C                                 VARIABLES,
C                 IPRINT(2) = 3 : SAME AS IPRINT(2)=2, PLUS GRADIENT VECT
C
C
C     EPS      IS A POSITIVE DOUBLE PRECISION VARIABLE THAT MUST BE SET B
C              THE USER, AND DETERMINES THE ACCURACY WITH WHICH THE SOLUT
C              IS TO BE FOUND. THE SUBROUTINE TERMINATES WHEN
```

```
C
C                          ||G|| < EPS MAX(1,||X||),
C
C              WHERE ||.|| DENOTES THE EUCLIDEAN NORM.
C
C     XTOL     IS A  POSITIVE DOUBLE PRECISION VARIABLE THAT MUST BE SET
C              THE USER TO AN ESTIMATE OF THE MACHINE PRECISION (E.G.
C              10**(-16) ON A SUN STATION 3/60). THE LINE SEARCH ROUTINE
C              TERMINATE IF THE RELATIVE WIDTH OF THE INTERVAL OF UNCERTA
C              IS LESS THAN XTOL.
C
C     W        IS A DOUBLE PRECISION ARRAY OF LENGTH N(2M+1)+2M USED AS
C              WORKSPACE FOR LBFGS. THIS ARRAY MUST NOT BE ALTERED BY THE
C              USER.
C
C     IFLAG    IS AN INTEGER VARIABLE THAT MUST BE SET TO 0 ON INITIAL EN
C              TO THE SUBROUTINE. A RETURN WITH IFLAG<0 INDICATES AN ERRO
C              AND IFLAG=0 INDICATES THAT THE ROUTINE HAS TERMINATED WITH
C              DETECTING ERRORS. ON A RETURN WITH IFLAG=1, THE USER MUST
C              EVALUATE THE FUNCTION F AND GRADIENT G. ON A RETURN WITH
C              IFLAG=2, THE USER MUST PROVIDE THE DIAGONAL MATRIX HK0.
C
C              THE FOLLOWING NEGATIVE VALUES OF IFLAG, DETECTING AN ERROR
C              ARE POSSIBLE:
C
C               IFLAG=-1  THE LINE SEARCH ROUTINE MCSRCH FAILED. THE
C                         PARAMETER INFO PROVIDES MORE DETAILED INFORMATI
C                         (SEE ALSO THE DOCUMENTATION OF MCSRCH):
C
C                        INFO = 0  IMPROPER INPUT PARAMETERS.
C
C                        INFO = 2  RELATIVE WIDTH OF THE INTERVAL OF
C                                  UNCERTAINTY IS AT MOST XTOL.
C
C                        INFO = 3  MORE THAN 20 FUNCTION EVALUATIONS WERE
C                                  REQUIRED AT THE PRESENT ITERATION.
C
C                        INFO = 4  THE STEP IS TOO SMALL.
C
C                        INFO = 5  THE STEP IS TOO LARGE.
C
C                        INFO = 6  ROUNDING ERRORS PREVENT FURTHER PROGRE
C                                  THERE MAY NOT BE A STEP WHICH SATISFIE
C                                  THE SUFFICIENT DECREASE AND CURVATURE
C                                  CONDITIONS. TOLERANCES MAY BE TOO SMAL
C
C               IFLAG=-2  THE I-TH DIAGONAL ELEMENT OF THE DIAGONAL INVER
C                         HESSIAN APPROXIMATION, GIVEN IN DIAG, IS NOT
C                         POSITIVE.
C
C               IFLAG=-3  IMPROPER INPUT PARAMETERS FOR LBFGS (N OR M ARE
C                         NOT POSITIVE).
C
C  MACHINE DEPENDENCIES
C
C        THE ONLY VARIABLES THAT ARE MACHINE-DEPENDENT ARE XTOL,
C        STPMIN AND STPMAX.
C
C  GENERAL INFORMATION
C
C    OTHER ROUTINES CALLED DIRECTLY:  DAXPY, DDOT, LB1, MCSRCH
C
C    INPUT/OUTPUT  :  NO INPUT; DIAGNOSTIC MESSAGES ON UNIT MP AND
C                     ERROR MESSAGES ON UNIT LP.
C
```

```
C     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
      DOUBLE PRECISION GTOL,ONE,ZERO,GNORM,DDOT,STP1,FTOL,STPMIN,
     .                 STPMAX,STP,YS,YY,SQ,YR,BETA,XNORM,DFNORM,TOLERA
      INTEGER MP,LP,ITER,NFUN,POINT,ISPT,IYPT,MAXFEV,INFO,
     .        BOUND,NPT,CP,I,NFEV,INMC,IYCN,ISCN
      LOGICAL FINISH
C
      COMMON /LB3/MP,LP,GTOL,STPMIN,STPMAX
C
      ONE = 1.0D+0
      ZERO = 0.0D+0
C
C     INITIALIZE
C     ----------
C
      IF(IFLAG.EQ.0) GO TO 10
      GO TO (172,100), IFLAG
   10 ITER= 0
      IF(N.LE.0.OR.M.LE.0) GO TO 196
      IF(GTOL.LE.1.D-04) THEN
        IF(LP.GT.0) WRITE(LP,245)
        GTOL=9.D-01
      ENDIF
C
C     PARAMETERS FOR LINE SEARCH ROUTINE
C
      FTOL= 1.0D-4
      MAXFEV= 20
      NFUN= 1
      POINT= 0
      FINISH= .FALSE.
      IF(DIAGCO) THEN
         DO 30 I=1,N
            IF (DIAG(I).LE.ZERO) GO TO 195
 30      CONTINUE
      ELSE
         DO 40 I=1,N
            DIAG(I)= 1.0D0
 40      CONTINUE
      ENDIF
C
C     THE WORK VECTOR W IS DIVIDED AS FOLLOWS:
C     ----------------------------------------
C     THE FIRST N LOCATIONS ARE USED TO STORE THE GRADIENT AND
C         OTHER TEMPORARY INFORMATION.
C     LOCATIONS (N+1)...(N+M) STORE THE SCALARS RHO.
C     LOCATIONS (N+M+1)...(N+2M) STORE THE NUMBERS ALPHA USED
C         IN THE FORMULA THAT COMPUTES H*G.
C     LOCATIONS (N+2M+1)...(N+2M+NM) STORE THE LAST M SEARCH
C         STEPS.
C     LOCATIONS (N+2M+NM+1)...(N+2M+2NM) STORE THE LAST M
C         GRADIENT DIFFERENCES.
C
C     THE SEARCH STEPS AND GRADIENT DIFFERENCES ARE STORED IN A
C     CIRCULAR ORDER CONTROLLED BY THE PARAMETER POINT.
C
      ISPT= N+2*M
      IYPT= ISPT+N*M
      DO 50 I=1,N
         W(ISPT+I)= -G(I)*DIAG(I)
 50   CONTINUE
      GNORM= DSQRT(DDOT(N,G,1,G,1))
      STP1= ONE/GNORM
C
```

```
      IF(IPRINT(1).GE.0) CALL LB1(IPRINT,ITER,NFUN,
     *                     GNORM,N,M,X,F,G,STP,FINISH,DFNORM)
C
C     --------------------
C      MAIN ITERATION LOOP
C     --------------------
C
 80    ITER= ITER+1
       INFO=0
       BOUND=ITER-1
       IF(ITER.EQ.1) GO TO 165
       IF (ITER .GT. M)BOUND=M
C
       YS= DDOT(N,W(IYPT+NPT+1),1,W(ISPT+NPT+1),1)
       IF(.NOT.DIAGCO) THEN
          YY= DDOT(N,W(IYPT+NPT+1),1,W(IYPT+NPT+1),1)
          DO 90 I=1,N
             DIAG(I)= YS/YY
 90       CONTINUE
       ELSE
          IFLAG=2
          RETURN
       ENDIF
 100   CONTINUE
       IF(DIAGCO) THEN
         DO 110 I=1,N
            IF (DIAG(I).LE.ZERO) GO TO 195
 110     CONTINUE
       ENDIF
C
C      COMPUTE -H*G USING THE FORMULA GIVEN IN: NOCEDAL, J. 1980,
C      "UPDATING QUASI-NEWTON MATRICES WITH LIMITED STORAGE",
C      MATHEMATICS OF COMPUTATION, VOL.24, NO.151, PP. 773-782.
C      ---------------------------------------------------------
C
       CP= POINT
       IF (POINT.EQ.0) CP=M
       W(N+CP)= ONE/YS
       DO 112 I=1,N
          W(I)= -G(I)
 112   CONTINUE
       CP= POINT
       DO 125 I= 1,BOUND
          CP=CP-1
          IF (CP.EQ. -1)CP=M-1
          SQ= DDOT(N,W(ISPT+CP*N+1),1,W,1)
          INMC=N+M+CP+1
          IYCN=IYPT+CP*N
          W(INMC)= W(N+CP+1)*SQ
          CALL DAXPY(N,-W(INMC),W(IYCN+1),1,W,1)
 125   CONTINUE
C
       DO 130 I=1,N
          W(I)=DIAG(I)*W(I)
 130   CONTINUE
C
       DO 145 I=1,BOUND
          YR= DDOT(N,W(IYPT+CP*N+1),1,W,1)
          BETA= W(N+CP+1)*YR
          INMC=N+M+CP+1
          BETA= W(INMC)-BETA
          ISCN=ISPT+CP*N
          CALL DAXPY(N,BETA,W(ISCN+1),1,W,1)
          CP=CP+1
          IF (CP.EQ.M)CP=0
```

```
 145  CONTINUE
C
C     STORE THE NEW SEARCH DIRECTION
C     ------------------------------
C
       DO 160 I=1,N
          W(ISPT+POINT*N+I)= W(I)
 160   CONTINUE
C
C     OBTAIN THE ONE-DIMENSIONAL MINIMIZER OF THE FUNCTION
C     BY USING THE LINE SEARCH ROUTINE MCSRCH
C     ---------------------------------------------------
 165  NFEV=0
      STP=ONE
      IF (ITER.EQ.1) STP=STP1
      DO 170 I=1,N
         W(I)=G(I)
 170  CONTINUE
 172  CONTINUE
C
      CALL MCSRCH(N,X,F,G,W(ISPT+POINT*N+1),STP,FTOL,
     *            XTOL,MAXFEV,INFO,NFEV,DIAG)
      IF (INFO .EQ. -1) THEN
        IFLAG=1
        RETURN
      ENDIF
      IF (INFO .NE. 1) GO TO 190
      NFUN= NFUN + NFEV
C
C     COMPUTE THE NEW STEP AND GRADIENT CHANGE
C     ----------------------------------------
C
      NPT=POINT*N
      DO 175 I=1,N
         W(ISPT+NPT+I)= STP*W(ISPT+NPT+I)
         W(IYPT+NPT+I)= G(I)-W(I)
  175 CONTINUE
C
      POINT=POINT+1
      IF (POINT.EQ.M)POINT=0
C
C     TERMINATION TEST
C     ----------------
C
      DFNORM=DSQRT(DDOT(N,W(ISPT+NPT+1),1,W(ISPT+NPT+1),1))
C
C     ADD A STOPING CRITERION FOR NEURAL NETWORKS LEARNING ALGORITHM.
C     THE SUBROUTINE TERMINATES WHEN ||XK+1-XK||< TOLERA.
C
      IF(ANN) THEN
         IF(DFNORM .LE. TOLERA) THEN
            FINISH = .TRUE.
            GO TO 180
         END IF
      END IF
C
      GNORM= DSQRT(DDOT(N,G,1,G,1))
      XNORM= DSQRT(DDOT(N,X,1,X,1))
      XNORM= DMAX1(1.0D0,XNORM)
      IF (GNORM/XNORM .LE. EPS) FINISH=.TRUE.
C
  180 IF(IPRINT(1).GE.0) CALL LB1(IPRINT,ITER,NFUN,GNORM,N,M,X,
     *                            F,G,STP,FINISH, DFNORM)
      IF (FINISH) THEN
         IFLAG=0
```

```
         RETURN
      ENDIF
      GO TO 80
C
C     ------------------------------------------------------------
C     END OF MAIN ITERATION LOOP. ERROR EXITS.
C     ------------------------------------------------------------
C
 190  IFLAG=-1
      IF(IPRINT(1).GE.0) CALL LB1(IPRINT,ITER,NFUN,
     *                GNORM,N,M,X,F,G,STP,FINISH,DFNORM)
      IF(LP.GT.0) WRITE(LP,200) INFO
      RETURN
 195  IFLAG=-2
      IF(LP.GT.0) WRITE(LP,235) I
      RETURN
 196  IFLAG= -3
      IF(LP.GT.0) WRITE(LP,240)
C
C     FORMATS
C     -------
C
 200  FORMAT(/' IFLAG= -1 '/' LINE SEARCH FAILED. SEE',
     .          ' DOCUMENTATION OF ROUTINE MCSRCH'/' ERROR RETURN',
     .          ' OF LINE SEARCH: INFO= ',I2/
     .          ' POSSIBLE CAUSES: FUNCTION OR GRADIENT ARE INCORRECT'/
     .          ' OR INCORRECT TOLERANCES')
 235  FORMAT(/' IFLAG= -2'/' THE',I5,'-TH DIAGONAL ELEMENT OF THE'/
     .       ' INVERSE HESSIAN APPROXIMATION IS NOT POSITIVE')
 240  FORMAT(/' IFLAG= -3'/' IMPROPER INPUT PARAMETERS (N OR M',
     .       ' ARE NOT POSITIVE)')
 245  FORMAT(/'  GTOL IS LESS THAN OR EQUAL TO 1.D-04'
     .       / ' IT HAS BEEN RESET TO 9.D-01')
      RETURN
      END
      SUBROUTINE LB1(IPRINT,ITER,NFUN,
     *                    GNORM,N,M,X,F,G,STP,FINISH,DFNORM)
C
C  ***  CHANGE PRINT STP TO DFNORM
C     ------------------------------------------------------------
C     THIS ROUTINE PRINTS MONITORING INFORMATION. THE FREQUENCY AND
C     AMOUNT OF OUTPUT ARE CONTROLLED BY IPRINT.
C     ------------------------------------------------------------
C
      INTEGER IPRINT(2),ITER,NFUN,LP,MP,N,M,    I
      DOUBLE PRECISION X(N),G(N),F,GNORM,DFNORM,STP,GTOL,STPMIN,STPMAX
      LOGICAL FINISH
      COMMON /LB3/MP,LP,GTOL,STPMIN,STPMAX
C
      IF (ITER.EQ.0)THEN
           WRITE(MP,10)
           WRITE(MP,20) N,M
           WRITE(MP,30)F,GNORM
                 IF (IPRINT(2).GE.1)THEN
                     WRITE(MP,40)
                     WRITE(MP,50) (X(I),I=1,N)
                     WRITE(MP,60)
                     WRITE(MP,50) (G(I),I=1,N)
                  ENDIF
           WRITE(MP,10)
           WRITE(MP,70)
      ELSE
          IF ((IPRINT(1).EQ.0).AND.(ITER.NE.1.AND..NOT.FINISH))RETURN
              IF (IPRINT(1).NE.0)THEN
                   IF(MOD(ITER-1,IPRINT(1)).EQ.0.OR.FINISH)THEN
```

```
                    IF(IPRINT(2).GT.1.AND.ITER.GT.1) WRITE(MP,70)
                    WRITE(MP,80)ITER,NFUN,F,GNORM,DFNORM
                ELSE
                    RETURN
                ENDIF
            ELSE
                IF( IPRINT(2).GT.1.AND.FINISH) WRITE(MP,70)
                WRITE(MP,80)ITER,NFUN,F,GNORM,DFNORM
            ENDIF
            IF (IPRINT(2).EQ.2.OR.IPRINT(2).EQ.3)THEN
                IF (FINISH)THEN
                    WRITE(MP,90)
                ELSE
                    WRITE(MP,40)
                ENDIF
                  WRITE(MP,50)(X(I),I=1,N)
              IF (IPRINT(2).EQ.3)THEN
                  WRITE(MP,60)
                  WRITE(MP,50)(G(I),I=1,N)
              ENDIF
            ENDIF
          IF (FINISH) WRITE(MP,100)
      ENDIF
C
 10   FORMAT('***************************************************')
 20   FORMAT('  N=',I5,'   NUMBER OF CORRECTIONS=',I2
     .       /  '        INITIAL VALUES')
 30   FORMAT(' F= ',1PD10.3,'   GNORM= ',1PD10.3)
 40   FORMAT(' VECTOR X= ')
 50   FORMAT(6(2X,1PD10.3))
 60   FORMAT(' GRADIENT VECTOR G= ')
 70   FORMAT(/'   I   NFN',4X,'FUNC',8X,'GNORM',7X,'DFNORM'/)
 80   FORMAT(2(I4,1X),3X,3(1PD10.3,2X))
 90   FORMAT(' FINAL POINT X= ')
 100  FORMAT(/' THE MINIMIZATION TERMINATED WITHOUT DETECTING ERRORS.'
     .       /' IFLAG = 0')
C
      RETURN
      END
      SUBROUTINE LBSET
C-----------------------------------------------------------------------
C     THIS SUBROUTINE CONTAINS ONE COMMON AREA, WHICH DEFINED
C     THE VALUES OF SEVERAL PARAMETERS DESCRIBED AS FOLLOWS:
C
C     MP    IS AN INTEGER VARIABLE WITH DEFAULT VALUE 6. IT IS USED AS T
C            UNIT NUMBER FOR THE PRINTING OF THE MONITORING INFORMATION
C            CONTROLLED BY PRI.
C
C     LP    IS AN INTEGER VARIABLE WITH DEFAULT VALUE 6. IT IS USED AS T
C            UNIT NUMBER FOR THE PRINTING OF ERROR MESSAGES.
C
C     GTOL  IS A DOUBLE PRECISION VARIABLE WITH DEFAULT VALUE 0.9, WHICH
C           CONTROLS THE ACCURACY OF THE LINE SEARCH ROUTINE MCSRCH. IF
C           FUNCTION AND GRADIENT EVALUATIONS ARE INEXPENSIVE WITH RESPE
C           TO THE COST OF THE ITERATION (WHICH IS SOMETIMES THE CASE WH
C           SOLVING VERY LARGE PROBLEMS) IT MAY BE ADVANTAGEOUS TO SET G
C           TO A SMALL VALUE. A TYPICAL SMALL VALUE IS 0.1.  RESTRICTION
C           GTOL SHOULD BE GREATER THAN 1.D-04.
C
C     STPMIN AND STPMAX ARE NON-NEGATIVE DOUBLE PRECISION VARIABLES WHIC
C           SPECIFY LOWER AND UPPER BOUNDS FOR THE STEP IN THE LINE SEAR
C           THEIR DEFAULT VALUES ARE 1.D-20 AND 1.D+20, RESPECTIVELY.
C-----------------------------------------------------------------------
      INTEGER LP,MP
      DOUBLE PRECISION   GTOL,STPMIN,STPMAX
```

```
      COMMON /LB3/MP,LP,GTOL,STPMIN,STPMAX
C
      MP=6
      LP=6
      GTOL=9.0D-1
      STPMIN=1.0D-20
      STPMAX=1.0D+20
C
      RETURN
      END
      SUBROUTINE DAXPY(N,DA,DX,INCX,DY,INCY)
C
C     CONSTANT TIMES A VECTOR PLUS A VECTOR.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION DX(1),DY(1),DA
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      IF(N.LE.0)RETURN
      IF (DA .EQ. 0.0D0) RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        DY(IY) = DY(IY) + DA*DX(IX)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,4)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DY(I) = DY(I) + DA*DX(I)
   30 CONTINUE
      IF( N .LT. 4 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,4
        DY(I) = DY(I) + DA*DX(I)
        DY(I + 1) = DY(I + 1) + DA*DX(I + 1)
        DY(I + 2) = DY(I + 2) + DA*DX(I + 2)
        DY(I + 3) = DY(I + 3) + DA*DX(I + 3)
   50 CONTINUE
      RETURN
      END
      DOUBLE PRECISION FUNCTION DDOT(N,DX,INCX,DY,INCY)
C
C     FORMS THE DOT PRODUCT OF TWO VECTORS.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DOUBLE PRECISION DX(1),DY(1),DTEMP
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
```

```
C
      DDOT = 0.0D0
      DTEMP = 0.0D0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        DTEMP = DTEMP + DX(IX)*DY(IY)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      DDOT = DTEMP
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        DTEMP = DTEMP + DX(I)*DY(I)
   30 CONTINUE
      IF( N .LT. 5 ) GO TO 60
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
        DTEMP = DTEMP + DX(I)*DY(I) + DX(I + 1)*DY(I + 1) +
     *  DX(I + 2)*DY(I + 2) + DX(I + 3)*DY(I + 3) + DX(I + 4)*DY(I + 4)
   50 CONTINUE
   60 DDOT = DTEMP
      RETURN
      END
      SUBROUTINE MCSRCH(N,X,F,G,S,STP,FTOL,XTOL,MAXFEV,INFO,NFEV,WA)
C     LINE SEARCH ROUTINE MCSRCH
      INTEGER N,MAXFEV,INFO,NFEV
      INTEGER LP,MP
      DOUBLE PRECISION F,STP,FTOL,GTOL,XTOL,STPMIN,STPMAX
      DOUBLE PRECISION X(N),G(N),S(N),WA(N)
      COMMON /LB3/MP,LP,GTOL,STPMIN,STPMAX
C
C                        SUBROUTINE MCSRCH
C
C     A SLIGHT MODIFICATION OF THE SUBROUTINE CSRCH OF MORE' AND THUENTE
C     THE CHANGES ARE TO ALLOW REVERSE COMMUNICATION, AND DO NOT AFFECT
C     THE PERFORMANCE OF THE ROUTINE.
C
C     THE PURPOSE OF MCSRCH IS TO FIND A STEP WHICH SATISFIES
C     A SUFFICIENT DECREASE CONDITION AND A CURVATURE CONDITION.
C
C     AT EACH STAGE THE SUBROUTINE UPDATES AN INTERVAL OF
C     UNCERTAINTY WITH ENDPOINTS STX AND STY. THE INTERVAL OF
C     UNCERTAINTY IS INITIALLY CHOSEN SO THAT IT CONTAINS A
C     MINIMIZER OF THE MODIFIED FUNCTION
C
C          F(X+STP*S) - F(X) - FTOL*STP*(GRADF(X)'S).
C
C     IF A STEP IS OBTAINED FOR WHICH THE MODIFIED FUNCTION
```

```
C     HAS A NONPOSITIVE FUNCTION VALUE AND NONNEGATIVE DERIVATIVE,
C     THEN THE INTERVAL OF UNCERTAINTY IS CHOSEN SO THAT IT
C     CONTAINS A MINIMIZER OF F(X+STP*S).
C
C     THE ALGORITHM IS DESIGNED TO FIND A STEP WHICH SATISFIES
C     THE SUFFICIENT DECREASE CONDITION
C
C           F(X+STP*S) .LE. F(X) + FTOL*STP*(GRADF(X)'S),
C
C     AND THE CURVATURE CONDITION
C
C           ABS(GRADF(X+STP*S)'S)) .LE. GTOL*ABS(GRADF(X)'S).
C
C     IF FTOL IS LESS THAN GTOL AND IF, FOR EXAMPLE, THE FUNCTION
C     IS BOUNDED BELOW, THEN THERE IS ALWAYS A STEP WHICH SATISFIES
C     BOTH CONDITIONS. IF NO STEP CAN BE FOUND WHICH SATISFIES BOTH
C     CONDITIONS, THEN THE ALGORITHM USUALLY STOPS WHEN ROUNDING
C     ERRORS PREVENT FURTHER PROGRESS. IN THIS CASE STP ONLY
C     SATISFIES THE SUFFICIENT DECREASE CONDITION.
C
C     THE SUBROUTINE STATEMENT IS
C
C        SUBROUTINE MCSRCH(N,X,F,G,S,STP,FTOL,XTOL, MAXFEV,INFO,NFEV,WA)
C     WHERE
C
C       N IS A POSITIVE INTEGER INPUT VARIABLE SET TO THE NUMBER
C         OF VARIABLES.
C
C       X IS AN ARRAY OF LENGTH N. ON INPUT IT MUST CONTAIN THE
C         BASE POINT FOR THE LINE SEARCH. ON OUTPUT IT CONTAINS
C         X + STP*S.
C
C       F IS A VARIABLE. ON INPUT IT MUST CONTAIN THE VALUE OF F
C         AT X. ON OUTPUT IT CONTAINS THE VALUE OF F AT X + STP*S.
C
C       G IS AN ARRAY OF LENGTH N. ON INPUT IT MUST CONTAIN THE
C         GRADIENT OF F AT X. ON OUTPUT IT CONTAINS THE GRADIENT
C         OF F AT X + STP*S.
C
C       S IS AN INPUT ARRAY OF LENGTH N WHICH SPECIFIES THE
C         SEARCH DIRECTION.
C
C       STP IS A NONNEGATIVE VARIABLE. ON INPUT STP CONTAINS AN
C         INITIAL ESTIMATE OF A SATISFACTORY STEP. ON OUTPUT
C         STP CONTAINS THE FINAL ESTIMATE.
C
C       FTOL AND GTOL ARE NONNEGATIVE INPUT VARIABLES. (IN THIS REVERSE
C         COMMUNICATION IMPLEMENTATION GTOL IS DEFINED IN A COMMON
C         STATEMENT.) TERMINATION OCCURS WHEN THE SUFFICIENT DECREASE
C         CONDITION AND THE DIRECTIONAL DERIVATIVE CONDITION ARE
C         SATISFIED.
C
C       XTOL IS A NONNEGATIVE INPUT VARIABLE. TERMINATION OCCURS
C         WHEN THE RELATIVE WIDTH OF THE INTERVAL OF UNCERTAINTY
C         IS AT MOST XTOL.
C
C       STPMIN AND STPMAX ARE NONNEGATIVE INPUT VARIABLES WHICH
C         SPECIFY LOWER AND UPPER BOUNDS FOR THE STEP. (IN THIS REVERSE
C         COMMUNICATION IMPLEMENTATIN THEY ARE DEFINED IN A COMMON
C         STATEMENT).
C
C       MAXFEV IS A POSITIVE INTEGER INPUT VARIABLE. TERMINATION
C         OCCURS WHEN THE NUMBER OF CALLS TO FCN IS AT LEAST
C         MAXFEV BY THE END OF AN ITERATION.
C
```

```
C       INFO IS AN INTEGER OUTPUT VARIABLE SET AS FOLLOWS:
C
C         INFO = 0  IMPROPER INPUT PARAMETERS.
C
C         INFO =-1  A RETURN IS MADE TO COMPUTE THE FUNCTION AND GRADIEN
C
C         INFO = 1  THE SUFFICIENT DECREASE CONDITION AND THE
C                   DIRECTIONAL DERIVATIVE CONDITION HOLD.
C
C         INFO = 2  RELATIVE WIDTH OF THE INTERVAL OF UNCERTAINTY
C                   IS AT MOST XTOL.
C
C         INFO = 3  NUMBER OF CALLS TO FCN HAS REACHED MAXFEV.
C
C         INFO = 4  THE STEP IS AT THE LOWER BOUND STPMIN.
C
C         INFO = 5  THE STEP IS AT THE UPPER BOUND STPMAX.
C
C         INFO = 6  ROUNDING ERRORS PREVENT FURTHER PROGRESS.
C                   THERE MAY NOT BE A STEP WHICH SATISFIES THE
C                   SUFFICIENT DECREASE AND CURVATURE CONDITIONS.
C                   TOLERANCES MAY BE TOO SMALL.
C
C       NFEV IS AN INTEGER OUTPUT VARIABLE SET TO THE NUMBER OF
C         CALLS TO FCN.
C
C       WA IS A WORK ARRAY OF LENGTH N.
C
C     SUBPROGRAMS CALLED
C
C       MCSTEP
C
C       FORTRAN-SUPPLIED...ABS,MAX,MIN
C
C     ARGONNE NATIONAL LABORATORY. MINPACK PROJECT. JUNE 1983
C     JORGE J. MORE', DAVID J. THUENTE
C
C     **********
      INTEGER INFOC,J
      LOGICAL BRACKT,STAGE1
      DOUBLE PRECISION DG,DGM,DGINIT,DGTEST,DGX,DGXM,DGY,DGYM,
     *       FINIT,FTEST1,FM,FX,FXM,FY,FYM,P5,P66,STX,STY,
     *       STMIN,STMAX,WIDTH,WIDTH1,XTRAPF,ZERO
C
C     DATA P5,P66,XTRAPF,ZERO /0.5D0,0.66D0,4.0D0,0.0D0/
C
      P5=0.5D0
      P66=0.66D0
      XTRAPF=4.0D0
      ZERO=0.0D0
C
      IF(INFO.EQ.-1) GO TO 45
      INFOC = 1
C
C     CHECK THE INPUT PARAMETERS FOR ERRORS.
C
      IF (N .LE. 0 .OR. STP .LE. ZERO .OR. FTOL .LT. ZERO .OR.
     *    GTOL .LT. ZERO .OR. XTOL .LT. ZERO .OR. STPMIN .LT. ZERO
     *    .OR. STPMAX .LT. STPMIN .OR. MAXFEV .LE. 0) RETURN
C
C     COMPUTE THE INITIAL GRADIENT IN THE SEARCH DIRECTION
C     AND CHECK THAT S IS A DESCENT DIRECTION.
C
      DGINIT = ZERO
      DO 10 J = 1, N
```

```
         DGINIT = DGINIT + G(J)*S(J)
   10    CONTINUE
      IF (DGINIT .GE. ZERO) THEN
         WRITE(LP,15)
   15    FORMAT(/'  THE SEARCH DIRECTION IS NOT A DESCENT DIRECTION')
         RETURN
         ENDIF
C
C     INITIALIZE LOCAL VARIABLES.
C
      BRACKT = .FALSE.
      STAGE1 = .TRUE.
      NFEV = 0
      FINIT = F
      DGTEST = FTOL*DGINIT
      WIDTH = STPMAX - STPMIN
      WIDTH1 = WIDTH/P5
      DO 20 J = 1, N
         WA(J) = X(J)
   20    CONTINUE
C
C     THE VARIABLES STX, FX, DGX CONTAIN THE VALUES OF THE STEP,
C     FUNCTION, AND DIRECTIONAL DERIVATIVE AT THE BEST STEP.
C     THE VARIABLES STY, FY, DGY CONTAIN THE VALUE OF THE STEP,
C     FUNCTION, AND DERIVATIVE AT THE OTHER ENDPOINT OF
C     THE INTERVAL OF UNCERTAINTY.
C     THE VARIABLES STP, F, DG CONTAIN THE VALUES OF THE STEP,
C     FUNCTION, AND DERIVATIVE AT THE CURRENT STEP.
C
      STX = ZERO
      FX = FINIT
      DGX = DGINIT
      STY = ZERO
      FY = FINIT
      DGY = DGINIT
C
C     START OF ITERATION.
C
   30 CONTINUE
C
C        SET THE MINIMUM AND MAXIMUM STEPS TO CORRESPOND
C        TO THE PRESENT INTERVAL OF UNCERTAINTY.
C
         IF (BRACKT) THEN
            STMIN = DMIN1(STX,STY)
            STMAX = DMAX1(STX,STY)
         ELSE
            STMIN = STX
            STMAX = STP + XTRAPF*(STP - STX)
            END IF
C
C        FORCE THE STEP TO BE WITHIN THE BOUNDS STPMAX AND STPMIN.
C
         STP = DMAX1(STP,STPMIN)
         STP = DMIN1(STP,STPMAX)
C
C        IF AN UNUSUAL TERMINATION IS TO OCCUR THEN LET
C        STP BE THE LOWEST POINT OBTAINED SO FAR.
C
         IF ((BRACKT .AND. (STP .LE. STMIN .OR. STP .GE. STMAX))
     *      .OR. NFEV .GE. MAXFEV-1 .OR. INFOC .EQ. 0
     *      .OR. (BRACKT .AND. STMAX-STMIN .LE. XTOL*STMAX)) STP = STX
C
C        EVALUATE THE FUNCTION AND GRADIENT AT STP
C        AND COMPUTE THE DIRECTIONAL DERIVATIVE.
```

```
C         WE RETURN TO MAIN PROGRAM TO OBTAIN F AND G.
C
          DO 40 J = 1, N
             X(J) = WA(J) + STP*S(J)
   40        CONTINUE
          INFO=-1
          RETURN
C
   45     INFO=0
          NFEV = NFEV + 1
          DG = ZERO
          DO 50 J = 1, N
             DG = DG + G(J)*S(J)
   50        CONTINUE
          FTEST1 = FINIT + STP*DGTEST
C
C         TEST FOR CONVERGENCE.
C
          IF ((BRACKT .AND. (STP .LE. STMIN .OR. STP .GE. STMAX))
     *       .OR. INFOC .EQ. 0) INFO = 6
          IF (STP .EQ. STPMAX .AND.
     *        F .LE. FTEST1 .AND. DG .LE. DGTEST) INFO = 5
          IF (STP .EQ. STPMIN .AND.
     *        (F .GT. FTEST1 .OR. DG .GE. DGTEST)) INFO = 4
          IF (NFEV .GE. MAXFEV) INFO = 3
          IF (BRACKT .AND. STMAX-STMIN .LE. XTOL*STMAX) INFO = 2
          IF (F .LE. FTEST1 .AND. DABS(DG) .LE. GTOL*(-DGINIT)) INFO = 1
C
C         CHECK FOR TERMINATION.
C
          IF (INFO .NE. 0) RETURN
C
C         IN THE FIRST STAGE WE SEEK A STEP FOR WHICH THE MODIFIED
C         FUNCTION HAS A NONPOSITIVE VALUE AND NONNEGATIVE DERIVATIVE.
C
          IF (STAGE1 .AND. F .LE. FTEST1 .AND.
     *        DG .GE. DMIN1(FTOL,GTOL)*DGINIT) STAGE1 = .FALSE.
C
C         A MODIFIED FUNCTION IS USED TO PREDICT THE STEP ONLY IF
C         WE HAVE NOT OBTAINED A STEP FOR WHICH THE MODIFIED
C         FUNCTION HAS A NONPOSITIVE FUNCTION VALUE AND NONNEGATIVE
C         DERIVATIVE, AND IF A LOWER FUNCTION VALUE HAS BEEN
C         OBTAINED BUT THE DECREASE IS NOT SUFFICIENT.
C
          IF (STAGE1 .AND. F .LE. FX .AND. F .GT. FTEST1) THEN
C
C            DEFINE THE MODIFIED FUNCTION AND DERIVATIVE VALUES.
C
             FM = F - STP*DGTEST
             FXM = FX - STX*DGTEST
             FYM = FY - STY*DGTEST
             DGM = DG - DGTEST
             DGXM = DGX - DGTEST
             DGYM = DGY - DGTEST
C
C            CALL CSTEP TO UPDATE THE INTERVAL OF UNCERTAINTY
C            AND TO COMPUTE THE NEW STEP.
C
             CALL MCSTEP(STX,FXM,DGXM,STY,FYM,DGYM,STP,FM,DGM,
     *                   BRACKT,STMIN,STMAX,INFOC)
C
C            RESET THE FUNCTION AND GRADIENT VALUES FOR F.
C
             FX = FXM + STX*DGTEST
             FY = FYM + STY*DGTEST
```

```
            DGX = DGXM + DGTEST
            DGY = DGYM + DGTEST
         ELSE
C
C           CALL MCSTEP TO UPDATE THE INTERVAL OF UNCERTAINTY
C           AND TO COMPUTE THE NEW STEP.
C
            CALL MCSTEP(STX,FX,DGX,STY,FY,DGY,STP,F,DG,
     *                  BRACKT,STMIN,STMAX,INFOC)
            END IF
C
C        FORCE A SUFFICIENT DECREASE IN THE SIZE OF THE
C        INTERVAL OF UNCERTAINTY.
C
         IF (BRACKT) THEN
            IF (DABS(STY-STX) .GE. P66*WIDTH1)
     *         STP = STX + P5*(STY - STX)
            WIDTH1 = WIDTH
            WIDTH = DABS(STY-STX)
            END IF
C
C        END OF ITERATION.
C
         GO TO 30
C
C     LAST LINE OF SUBROUTINE MCSRCH.
C
      END
      SUBROUTINE MCSTEP(STX,FX,DX,STY,FY,DY,STP,FP,DP,BRACKT,
     *                  STPMIN,STPMAX,INFO)
      INTEGER INFO
      DOUBLE PRECISION STX,FX,DX,STY,FY,DY,STP,FP,DP,STPMIN,STPMAX
      LOGICAL BRACKT,BOUND
C
C     SUBROUTINE MCSTEP
C
C     THE PURPOSE OF MCSTEP IS TO COMPUTE A SAFEGUARDED STEP FOR
C     A LINESEARCH AND TO UPDATE AN INTERVAL OF UNCERTAINTY FOR
C     A MINIMIZER OF THE FUNCTION.
C
C     THE PARAMETER STX CONTAINS THE STEP WITH THE LEAST FUNCTION
C     VALUE. THE PARAMETER STP CONTAINS THE CURRENT STEP. IT IS
C     ASSUMED THAT THE DERIVATIVE AT STX IS NEGATIVE IN THE
C     DIRECTION OF THE STEP. IF BRACKT IS SET TRUE THEN A
C     MINIMIZER HAS BEEN BRACKETED IN AN INTERVAL OF UNCERTAINTY
C     WITH ENDPOINTS STX AND STY.
C
C     THE SUBROUTINE STATEMENT IS
C
C       SUBROUTINE MCSTEP(STX,FX,DX,STY,FY,DY,STP,FP,DP,BRACKT,
C                         STPMIN,STPMAX,INFO)
C
C     WHERE
C
C       STX, FX, AND DX ARE VARIABLES WHICH SPECIFY THE STEP,
C         THE FUNCTION, AND THE DERIVATIVE AT THE BEST STEP OBTAINED
C         SO FAR. THE DERIVATIVE MUST BE NEGATIVE IN THE DIRECTION
C         OF THE STEP, THAT IS, DX AND STP-STX MUST HAVE OPPOSITE
C         SIGNS. ON OUTPUT THESE PARAMETERS ARE UPDATED APPROPRIATELY.
C
C       STY, FY, AND DY ARE VARIABLES WHICH SPECIFY THE STEP,
C         THE FUNCTION, AND THE DERIVATIVE AT THE OTHER ENDPOINT OF
C         THE INTERVAL OF UNCERTAINTY. ON OUTPUT THESE PARAMETERS ARE
C         UPDATED APPROPRIATELY.
C
```

```
C       STP, FP, AND DP ARE VARIABLES WHICH SPECIFY THE STEP,
C         THE FUNCTION, AND THE DERIVATIVE AT THE CURRENT STEP.
C         IF BRACKT IS SET TRUE THEN ON INPUT STP MUST BE
C         BETWEEN STX AND STY. ON OUTPUT STP IS SET TO THE NEW STEP.
C
C       BRACKT IS A LOGICAL VARIABLE WHICH SPECIFIES IF A MINIMIZER
C         HAS BEEN BRACKETED. IF THE MINIMIZER HAS NOT BEEN BRACKETED
C         THEN ON INPUT BRACKT MUST BE SET FALSE. IF THE MINIMIZER
C         IS BRACKETED THEN ON OUTPUT BRACKT IS SET TRUE.
C
C       STPMIN AND STPMAX ARE INPUT VARIABLES WHICH SPECIFY LOWER
C         AND UPPER BOUNDS FOR THE STEP.
C
C       INFO IS AN INTEGER OUTPUT VARIABLE SET AS FOLLOWS:
C         IF INFO = 1,2,3,4,5, THEN THE STEP HAS BEEN COMPUTED
C         ACCORDING TO ONE OF THE FIVE CASES BELOW. OTHERWISE
C         INFO = 0, AND THIS INDICATES IMPROPER INPUT PARAMETERS.
C
C     SUBPROGRAMS CALLED
C
C       FORTRAN-SUPPLIED ... ABS,MAX,MIN,SQRT
C
C     ARGONNE NATIONAL LABORATORY. MINPACK PROJECT. JUNE 1983
C     JORGE J. MORE', DAVID J. THUENTE
C
      DOUBLE PRECISION GAMMA,P,Q,R,S,SGND,STPC,STPF,STPQ,THETA
      INFO = 0
C
C     CHECK THE INPUT PARAMETERS FOR ERRORS.
C
      IF ((BRACKT .AND. (STP .LE. DMIN1(STX,STY) .OR.
     *    STP .GE. DMAX1(STX,STY))) .OR.
     *    DX*(STP-STX) .GE. 0.0 .OR. STPMAX .LT. STPMIN) RETURN
C
C     DETERMINE IF THE DERIVATIVES HAVE OPPOSITE SIGN.
C
      SGND = DP*(DX/DABS(DX))
C
C     FIRST CASE. A HIGHER FUNCTION VALUE.
C     THE MINIMUM IS BRACKETED. IF THE CUBIC STEP IS CLOSER
C     TO STX THAN THE QUADRATIC STEP, THE CUBIC STEP IS TAKEN,
C     ELSE THE AVERAGE OF THE CUBIC AND QUADRATIC STEPS IS TAKEN.
C
      IF (FP .GT. FX) THEN
         INFO = 1
         BOUND = .TRUE.
         THETA = 3*(FX - FP)/(STP - STX) + DX + DP
         S = DMAX1(DABS(THETA),DABS(DX),DABS(DP))
         GAMMA = S*DSQRT((THETA/S)**2 - (DX/S)*(DP/S))
         IF (STP .LT. STX) GAMMA = -GAMMA
         P = (GAMMA - DX) + THETA
         Q = ((GAMMA - DX) + GAMMA) + DP
         R = P/Q
         STPC = STX + R*(STP - STX)
         STPQ = STX + ((DX/((FX-FP)/(STP-STX)+DX))/2)*(STP - STX)
         IF (DABS(STPC-STX) .LT. DABS(STPQ-STX)) THEN
            STPF = STPC
         ELSE
           STPF = STPC + (STPQ - STPC)/2
           END IF
         BRACKT = .TRUE.
C
C     SECOND CASE. A LOWER FUNCTION VALUE AND DERIVATIVES OF
C     OPPOSITE SIGN. THE MINIMUM IS BRACKETED. IF THE CUBIC
C     STEP IS CLOSER TO STX THAN THE QUADRATIC (SECANT) STEP,
```

```
C     THE CUBIC STEP IS TAKEN, ELSE THE QUADRATIC STEP IS TAKEN.
C
      ELSE IF (SGND .LT. 0.0) THEN
         INFO = 2
         BOUND = .FALSE.
         THETA = 3*(FX - FP)/(STP - STX) + DX + DP
         S = DMAX1(DABS(THETA),DABS(DX),DABS(DP))
         GAMMA = S*DSQRT((THETA/S)**2 - (DX/S)*(DP/S))
         IF (STP .GT. STX) GAMMA = -GAMMA
         P = (GAMMA - DP) + THETA
         Q = ((GAMMA - DP) + GAMMA) + DX
         R = P/Q
         STPC = STP + R*(STX - STP)
         STPQ = STP + (DP/(DP-DX))*(STX - STP)
         IF (DABS(STPC-STP) .GT. DABS(STPQ-STP)) THEN
            STPF = STPC
         ELSE
            STPF = STPQ
            END IF
         BRACKT = .TRUE.
C
C     THIRD CASE. A LOWER FUNCTION VALUE, DERIVATIVES OF THE
C     SAME SIGN, AND THE MAGNITUDE OF THE DERIVATIVE DECREASES.
C     THE CUBIC STEP IS ONLY USED IF THE CUBIC TENDS TO INFINITY
C     IN THE DIRECTION OF THE STEP OR IF THE MINIMUM OF THE CUBIC
C     IS BEYOND STP. OTHERWISE THE CUBIC STEP IS DEFINED TO BE
C     EITHER STPMIN OR STPMAX. THE QUADRATIC (SECANT) STEP IS ALSO
C     COMPUTED AND IF THE MINIMUM IS BRACKETED THEN THE THE STEP
C     CLOSEST TO STX IS TAKEN, ELSE THE STEP FARTHEST AWAY IS TAKEN.
C
      ELSE IF (DABS(DP) .LT. DABS(DX)) THEN
         INFO = 3
         BOUND = .TRUE.
         THETA = 3*(FX - FP)/(STP - STX) + DX + DP
         S = DMAX1(DABS(THETA),DABS(DX),DABS(DP))
C
C        THE CASE GAMMA = 0 ONLY ARISES IF THE CUBIC DOES NOT TEND
C        TO INFINITY IN THE DIRECTION OF THE STEP.
C
         GAMMA = S*DSQRT(DMAX1(0.0D0,(THETA/S)**2 - (DX/S)*(DP/S)))
         IF (STP .GT. STX) GAMMA = -GAMMA
         P = (GAMMA - DP) + THETA
         Q = (GAMMA + (DX - DP)) + GAMMA
         R = P/Q
         IF (R .LT. 0.0 .AND. GAMMA .NE. 0.0) THEN
            STPC = STP + R*(STX - STP)
         ELSE IF (STP .GT. STX) THEN
            STPC = STPMAX
         ELSE
            STPC = STPMIN
            END IF
         STPQ = STP + (DP/(DP-DX))*(STX - STP)
         IF (BRACKT) THEN
            IF (DABS(STP-STPC) .LT. DABS(STP-STPQ)) THEN
               STPF = STPC
            ELSE
               STPF = STPQ
               END IF
         ELSE
            IF (DABS(STP-STPC) .GT. DABS(STP-STPQ)) THEN
               STPF = STPC
            ELSE
               STPF = STPQ
               END IF
            END IF
```

```
C
C     FOURTH CASE. A LOWER FUNCTION VALUE, DERIVATIVES OF THE
C     SAME SIGN, AND THE MAGNITUDE OF THE DERIVATIVE DOES
C     NOT DECREASE. IF THE MINIMUM IS NOT BRACKETED, THE STEP
C     IS EITHER STPMIN OR STPMAX, ELSE THE CUBIC STEP IS TAKEN.
C
      ELSE
         INFO = 4
         BOUND = .FALSE.
         IF (BRACKT) THEN
            THETA = 3*(FP - FY)/(STY - STP) + DY + DP
            S = DMAX1(DABS(THETA),DABS(DY),DABS(DP))
            GAMMA = S*DSQRT((THETA/S)**2 - (DY/S)*(DP/S))
            IF (STP .GT. STY) GAMMA = -GAMMA
            P = (GAMMA - DP) + THETA
            Q = ((GAMMA - DP) + GAMMA) + DY
            R = P/Q
            STPC = STP + R*(STY - STP)
            STPF = STPC
         ELSE IF (STP .GT. STX) THEN
            STPF = STPMAX
         ELSE
            STPF = STPMIN
            END IF
         END IF
C
C     UPDATE THE INTERVAL OF UNCERTAINTY. THIS UPDATE DOES NOT
C     DEPEND ON THE NEW STEP OR THE CASE ANALYSIS ABOVE.
C
      IF (FP .GT. FX) THEN
         STY = STP
         FY = FP
         DY = DP
      ELSE
         IF (SGND .LT. 0.0) THEN
            STY = STX
            FY = FX
            DY = DX
            END IF
         STX = STP
         FX = FP
         DX = DP
         END IF
C
C     COMPUTE THE NEW STEP AND SAFEGUARD IT.
C
      STPF = DMIN1(STPMAX,STPF)
      STPF = DMAX1(STPMIN,STPF)
      STP = STPF
      IF (BRACKT .AND. BOUND) THEN
         IF (STY .GT. STX) THEN
            STP = DMIN1(STX+0.66D0*(STY-STX),STP)
         ELSE
            STP = DMAX1(STX+0.66D0*(STY-STX),STP)
            END IF
         END IF
      RETURN
C
C     LAST LINE OF SUBROUTINE MCSTEP.
C
      END
```

VITA

Yijun (Grace) Huang

Candidate for the Degree of

Master of Science

Thesis: NEURAL NETWORK LEARNING ALGORITHMS BASED ON LIMITED MEMORY QUASI-NEWTON METHODS

Major Field: Computer Science

Biographical:

Education: Graduated from Shanghai Normal University and Shanghai Institute of Education, Shanghai, China in July 1980 and July 1987, respectively. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in May 1997.

Professional Experience: Shanghai ChaoYang 6th High School and Technical School of 15th Cotton Mill of Shanghai, as a teacher of mathematics, Shanghai, China, September 1980 to September 1989; employed by Oklahoma State University, Department of Computer Science as a teaching assistant, January 1997 to May 1997.