APPLICATION OF NEURAL NETWORKS IN ROBOTIC CONTROL AND DESIGN OF MECHANISMS

RAGHU BALASUBRAMANIAN

APPLICATION OF NEURAL NETWORKS IN ROBOTIC CONTROL AND DESIGN OF MECHANISMS

By

© RAGHU BALASUBRAMANIAN, B.E.

A thesis submitted to the School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Engineering
and Applied Science

Faculty of Engineering and Applied Sciences
Memorial University of Newfoundland
December 1993

St. John's Newfoundland Canada

National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

ISBN 0-315-91636-2

Abstract

Neural network has been widely used in various fields of robotics. In this work, the neural network analysis using backpropagation algorithm is applied to the inverse velocity analysis of robotic manipulators near the singularity points accounting for the tracking error and feasibility of joint velocities. The inverse computations using the pseudo-inverse of the Jacobian matrix are compared with those obtained by the neural network analysis. The results illustrated using examples of two well known manipulators show the advantages of using the present work. A new learning algorithm called LP-neuro method is then developed to solve neural network problems. In this algorithm, the weights are obtained by a combination of Linear Programming having a sparse coefficient matrix and a single variable non-linear optimization method. The results are illustrated by solving three different problems, two of which are useful in the on-line control of robotic manipulators.

The designs of a function generator and a four-bar mechanism whose coupler curve passes through nine specified points, have been carried out using neural network methods. The design problem has been solved using non-linear techniques which yield a weight matrix in each of the cases. The accuracy of the methods is also discussed. Finally, gain parameters required for the trajectory control are evaluated using non-linear optimization method. Neural network is then trained to evaluate the gain parameters based on error history of different trajectories.

Acknowledgements

I would like to express my appreciation and profound gratitude to my advisor Prof. A.M. Sharan for his patient guidance and constant support. I am grateful to Profs. M.J. Hinchey, A.S.J. Swamidas and K. Munaswamy for their valuable suggestions and guidance during the course. I am also thankful to the Dean, Faculty of Engineering and Applied Science for the financial support during the program. I would also like to thank our Associate Deans, Dr. J.J. Sharp and Dr. T.R. Chari for their encouragement extended during my stay in the campus. Finally, I thank all my fellow graduate students and friends for their help and moral support during the course of this work.

Contents

List of Figures

List of Tables

List of Symbols

$\{\ \}$	vector
$[\ ]$	matrix
$\bar{a}_p$	acceleration of the end effector in the radial direction
e	error in joint position
$\dot{e}$	error in joint velocity
$f(.)$	activation function
f_x^i, f_y^i	forces acting on link i
l_1, l_2	link lengths
v_i, a_i	Cartesian velocity and acceleration of *i*th joint respectively
v_t	velocity of the end effector in the tangential direction
x_i,y_i	precision points
$\{\dot{x}\}$	Cartesian velocity vector
$\{I\}$,$\{D\}$,$\{O\}$	input, desired, and output vectors respectively
$[J]$	Jacobian matrix
$[J^+]$	psuedo-inverse of Jacobian matrix
k_{p1},k_{p2},k_{v1},k_{v2}	gain parameters
$[K_p]$	proportional gain matrix
$[K_v]$	velocity gain matrix
L_0,L_1,L_2,L_3	link lengths of four-bar mechanism
$[W]$,$[V]$	weight matrices

$[W_1]$	weight matrix for acceleration analysis
$[W_2]$	weight matrix for torque analysis
$[W_{ij}]$	weight matrix connecting *i*th and *j*th layers
X_i, Y_i	coordinates of the nine-point path problem
δ_i	error in the *i*th layer
η	learning factor
θ_i	displacement of *i*th joint
$\dot{\theta}_i$, $\ddot{\theta}_i$	angular velocity and acceleration of *i*th joint respectively
λ	damping factor
σ_i, u_i and v_i	components of singular value decomposed matrix
τ^i	torque acting on link i
Θ_i	displacement of *i*th joint
$\{\dot{\Theta}\}$	angular velocity vector
$\dot{\Theta}_{imax}$	upper feasible limit of angular velocity
$\ddot{\Theta}$	joint acceleration

Chapter 1
Introduction and Literature Survey

1.1 Introduction

Artificial Intelligence (AI) is applied in diversified fields to achieve faster and better results. They are useful for achieving computationally fast and approximate solutions of certain decision problems that are based on information of diverse criteria. Expert systems, Artificial Neural Networks (ANN), Knowledge-based representations etc., are examples of different tools used in the application of AI. Robotics is a field that requires such techniques because robots are often employed to work in hazardous environments impossible for human interactions, and where the calculations are numerous and complicated. In the recent past, ANN have proved quite useful in robotics. Fig. 1.1 shows the various fields in robotics in which ANN is being widely used.

Singularity avoidance, synthesis of mechanisms, finer control of the trajectories of robotic manipulators are still the topics that require further research. A new technique which optimizes the efficiency and speed would be of great help because of on-line computational requirements in the robotics area.

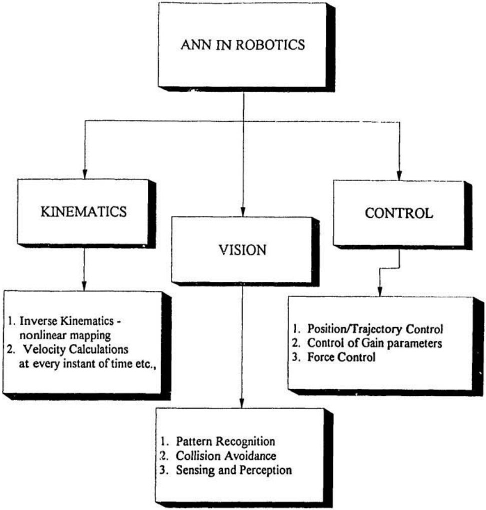


Figure 1.1 Application of Neural Network in Robotics

1.2 Literature Survey

1.2.1 Artificial Neural Network Methods

Artificial Neural Networks (ANN) have been studied for more than 30 years. Its use has increased tremendously in recent years because of the availability of faster and parallel processors and the basic learning algorithms (Grossberg, 1982; Hopfield, 1982; Rumelhart and McClelland, 1986; Kohonen, 1988). ANNs also referred as neural networks in this thesis are being used to accomplish complex functions such as generalization, error correction, information reconstruction, pattern analysis and learning. Neural network can learn mapping between the input and output space and synthesize an associative memory that retrieves the appropriate output when presented with an input, and has the ability to generalize with new inputs. Because of their massively parallel nature, neural networks can perform computations at very high speed (Fukuda and Shibata, 1992).

Neural networks have also been used to successfully solve complex problems like the Travelling salesman problem. It has been observed that neural networks have often been opportunistic, i.e. the network model is customized to serve the needs of the task at hand (Kulkarni, 1991). They represent a new approach that is robust and fault-tolerant.

Neural networks require basic algorithms for accomplishing the learning task. Several algorithms are functional in the present. One such algorithm which is widely used is backpropagation (BP) algorithm. In backpropagation algorithm, during the learning phase, the observed outputs are compared with the desired outputs, and the weights are optimized to minimize the error function. In competitive learning, the weights are updated with each new input (Rumelhart and McClelland, 1986). Barmann and Biegler-Kong (1992) discuss efficient learning algorithms for neural networks.

Neural networks can perform functional approximations that are beyond the scope of optimal linear techniques. Gulati et al., (1990) have introduced neural formalism to efficiently learn non-linear mapping using a mathematical construct called terminal attractors.

Neural networks have been found useful in the field of robotics in the recent times. Forward and inverse displacement analyses of robotic manipulators have been done by Nyugen et al. (1990) and Gulati et al., (1990). Neural networks seem to be a promising approach to solve non-linear control problems as well (Tabary and Salaun, 1992). Some other interesting applications in the control of robotic manipulators can be seen in Fukuda et al., 1991; and Akio et al., 1992.

1.2.2 Singularity Problems in Robotics

Inverse kinematics problems of robotic manipulators are always difficult to solve because of (a) the multiple solutions in the displacement analysis problems, or (b) the occurrence of singularity points along the trajectories in the case of velocity analysis. The singularity problems, which involve the rank deficiency in the Jacobian Matrix, have been dealt with by Chiaverni (1992). In this regard, general discussions on pseudo-inverse solutions can be seen in Lawson and Hanson, 1974. The pseudo-inverse solutions do not lead to satisfactory performance near the points of singularity because of abrupt changes in the elements of the joint velocity vector.

Damped-Least Squares method (DLS) approach has been used by many researchers (Wampler, 1986; Nakamura and Hanafusa, 1986; Maciejewski and Klein, 1989; Wampler and Leifer, 1988; Mayorga et al., 1992). The additional advantage with this method is that one can set the limit (achievable limit) on the norm of the joint velocity vector and find the corresponding damping factor, λ, which yields the minimum error. Maciejewski and Klein (1989) also proposed a truncated Singular Value Decomposition (SVD) solution method which could be used for on-line computations. However the resulting errors could be more in this method. So far, there has not been any method which takes into account factors such as the errors as well as the computational efficiency. Neural networks are known to perform well in those areas provided a relationship is established between the joint velocity vectors and Cartesian velocity vectors on an off-line basis. This circumvents the on-line computational requirements of the joint velocity vector, as was done by researchers mentioned earlier (Maciejewski and Klein, 1989).

1.2.3 Mechanism Synthesis

Synthesis of a mechanism is a means of finding the linkage that will produce the specified motion. The problem of approximate synthesis of a four-bar mechanism whose coupler curve is a planar trajectory was solved by Wampler et al., (1992). Solution of such problems date as early as 1923 and some of the important works are given in Freudenstein and Sandor, 1959; Shigley and Uicker, 1980; Erdman and Sandor, 1984; Morgan and Wampler, 1989; Subbian and Flugrad, 1989. The use of optimization technique has been made by Suh and Radcliffe (1978). Angeles et al., (1988), or Akhras and Angeles (1990) have applied a variable-separation technique and non-linear optimization scheme to solve the four-bar path generation problem. Tsai and Lu (1989) have solved the nine-point path problem using a new continuation method. Wampler et al., (1992) have solved this problem using a combination of analytical and numerical tools. Problems where the number of points is greater than nine result in an over-determined system whose exact solutions are not possible.

The four-bar mechanisms have also been used in the design of function-generators. Freudenstein (1955) proposed an algebraic formulation for the approximate synthesis of such a mechanism. Wilde (1982) applied error linearization techniques to solve this problem. Other interesting references on such problems can be seen in (Mohan Rao et al., 1973; Tinubu and Gupta, 1984; and Liu and Angeles, 1992).

1.2.4 Neural Network Control in Robotics

There has been recent trend within the robotics control literature to apply neural networks for the control of robotic systems. In many applications reported in the literature (Gu and Chan, 1989; Fukuda and Shibata, 1990; Helferty and Biswas, 1990; Jamshidi et al., 1990; Karakasoglu and Sundareshan, 1990; Yamamura et al., 1990) the process of neural network learning is conducted on-line (i.e. the dynamics of the neural network is embedded in the closed-loop with the dynamics of the robotic system), yet there appears to be a lack of studies focussing on the dynamic behavior of the neural network during learning and/or control when the neural network is used in such context.

Kawato (1990) used feedback error learning to compute the feedforward torques required for a manipulator to follow a path. The neural network implemented in this method uses the desired joint positions, velocities and accelerations as inputs and adjusts the network weights using the feedback torque as the error signal to a backpropagation parameter optimizing algorithm. Yuh (1992) also used a neural network for manipulator control. He used a "critic" equation, which is a function of the manipulator output error, to train the network to directly compute the manipulator input torques.

Asada (1990) used a multilayered feedforward network to learn a non-linear mapping for compliance control. From the measured forces and torques in an assembly task he used the network to compute the required velocities, which would allow the assembly task to be completed.

1.3 Thesis Objectives

We have seen in the last few sections that the neural networks are quite versatile tools to solve problems in a wide variety of areas. With this in mind, it was thought to apply this tool to solve problems in the areas of mechanism design and robotic control. Based on this, the following are the objectives of this thesis:

1) Development of a new neural network learning algorithm (LP-neuro method) which is fast and accurate.

2) Application of neural networks for inverse kinematics of robotic manipulators near singular configurations and comparison with damped-least squares and pseudo-inverse methods.

3) Velocity, acceleration and torque analysis of robotic manipulators using neural networks.

4) Synthesis of mechanisms using neural networks

5) Trajectory control of the robotic manipulators using neural networks.

Chapter 2, briefly reviews the basics of neural networks. Backpropagation algorithm is introduced here and various factors influencing a neural network are discussed in this chapter. The significance of solving for weight matrix in neural network problems using combination of LP and a single variable non-linear optimization routine is identified here. The validity of the application of backpropagation algorithm is checked by using them near singular configurations of robotic manipulators. An inverse kinematic relationship is established between the Cartesian and joint velocities on off-line basis which reduces on-line computation time. The relative merits and demerits of this method over conventional pseudo-inverse and damped-least squares method are discussed in this chapter. A new algorithm called LP-neuro method is developed to solve problems using neural networks.

In Chapter 3, the backpropagation method and the new algorithm called the LP-neuro method are then applied to solve various mechanism synthesis problems.

Chapter 4 deals with solution of non-linear or adaptive control problems. Here the non-linear control problem is solved using LP-neuro method developed in Chapter 2. Next, the gain values obtained by the non-linear method are then used in the neural control method where the methodology developed in sections 2.4.1 to 2.4.3 are used. In this way, the number of training sets required is a lot less than what many other researchers have used.

Finally, in Chapter 5, the contributions of the thesis and recommendations for future research are outlined.

Chapter 2
Neural Network Methods

2.1 Introduction

Neural network methods are widely used in many engineering applications. They can be thought of as a mathematical tool to solve common engineering problems such as optimization, pattern recognition etc. The *neural network* indicates the similarity of modelling network of neurons in the brain. Many linear and nonlinear neuron models are connected in the network and information is processed in a parallel distributed manner. This greatly reduces the computation time. Neural networks have learning and self-organization capabilities. They adapt to changes in data, learning the characteristics of the input signal.

Neural networks can be broadly classified into two types:

1) The neural networks that learn and adapt to changes are called recurrent networks or backpropagation networks. Multilayer perceptron neural nets, Hopfield nets, Adaptive Resonance Theory (ART) networks fall under this category.

2) Those that do not involve learning and sometimes called feedforward nets. Outer-product associative memories and multilayer nets without backward error

corrections belong to this type. The most popular neural networks used today are the Hopfield nets, Kohonen's self-organizing maps, multilayer perceptrons and ART nets.

Some of the operations that neural networks perform are shown in Fig. 2.1. They are advantageous in the following situations:

1) Decision-making from a massive amount of data

2) Non-linear mapping

3) Obtaining near-optimal solutions to optimization problem in less time.

2.2 Backpropagation Method

2.2.1 Multilayer Neural Network

A typical neural network is shown in Fig. 2.2. Basic components of a neural network are:

1) Input and output data sets

2) Weighed connections

3) Processing Elements (PE) or neurons

4) Activation function

The neural networks that need to be trained are supplied with predefined input and output data sets in a vector form. Each layer of a neural network consists of several processing elements. Each PE in a neural network sums all of its input values and performs a predefined operation and produces a single output value. PE's are connected with

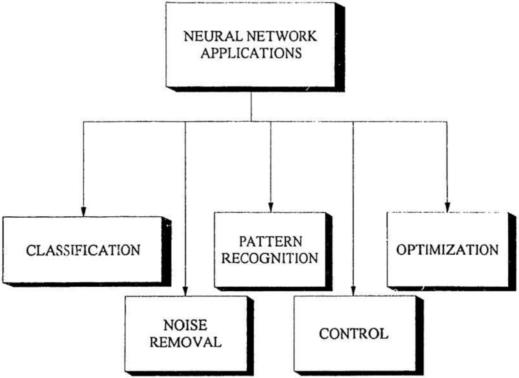


Figure 2.1 Applications of Neural Networks

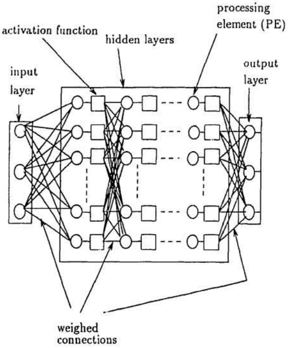


Figure 2.2 A Typical Neural Network

weighed connections. Information is stored in a network in the form of weights. In neural network method the weight matrix is obtained based on the learning process i.e., based on the input and output information used for that purpose.

Activation functions, also known as squashing functions, perform mapping of PE's infinite domain into a prespecified range. Commonly used activation functions (shown in Fig. 2.3) are:

1) Linear activation function

2) Step activation function

3) Ramp activation function

4) Sigmoidal activation function or squashing function

5) Gaussian function

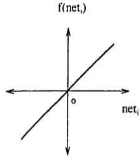
Neural networks are organized into several layers of PE's which include input layer, hidden layers and output layer as shown in Fig. 2.2. A feedforward network is one that has connections which feed information in one direction without any feedback path. If a network has feedback paths, then it is called feedback network. The training of multilayer neural networks depend on the following factors:

1) The number of layers

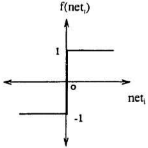
2) The number of PE in each layer

3) The amount of data needed for sufficient training.

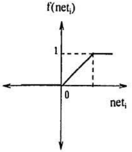
There are no predefined set of rules available for determining the above factors. Several techniques are available for the multilayer neural networks to have their connection



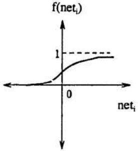
(a) Linear activation function



(b) Hard limiting function



(c) Threshold function



(d) Sigmoidal activation function

Figure 2.3 Activation Functions

weights adjusted to learn mapping. The most popular technique is the backpropagation algorithm (Werbos, 1974; Parker, 1982; Rumelhart, Hinton, and Williams 1986).

Learning process can be classified into two categories: supervised learning and unsupervised learning. Supervised learning monitors the duration of the training and the error performance etc., Unsupervised learning incorporates no monitoring process and relies only upon local information during the entire learning process. Most learning techniques are carried out off-line.

2.2.2 Feedforward Recall and Error Backpropagation Algorithm.

In neural network method, one establishes a relationship between the input and the desired output parameters. The matrix relationship between these two vectors are approximated by using several hidden layers as shown in Fig. 2.4. In this figure, the relationship between the input vector and the first hidden layer vector is at first expressed involving a weight matrix whose elements vary between -1 and 1 and are randomly generated. Similar procedure is adopted for the relationship between two adjacent hidden layers or the last hidden layer and the output layer. Mathematically, one of these typical relationships can be written as,

$$\{H\}_1 = [W]_1 \ \{I\} \tag{2.1}$$

where $\{I\}$ is the input vector and $\{H\}_1$ is the first hidden layer.

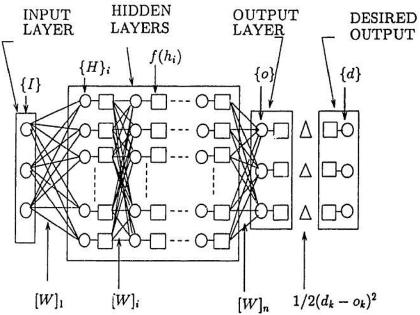


Figure 2.4 Representation of Neural Network Layers - Forward Computations

Next, values corresponding to sigmoidal function of each of the elements of the vector $\{H\}_1$ are computed and are symbolically represented by a square (□) in Fig. 2.4. For example, for a typical element it would be written as

$$f(h_i) = \frac{1}{1 + \exp(-\alpha h_i)} \tag{2.2}$$

where α is the steepness factor and h_i is one of the elements of vector $\{H\}_1$. This process is continued until the last hidden layer i.e., each layer is related to other by a matrix containing weights, and also, there is a similar relationship written between the last hidden layer and the output layer.

Defining two vectors $\{o\}$ and $\{d\}$ as the vector of output sigmoidal functions and desired values respectively, we wish to minimize the error E defined by

$$E = \frac{1}{2} \sum_{k=1}^{N} (d_k - o_k)^2 \tag{2.3}$$

Each of the summation terms (E_i) is represented by triangular (△) symbol in Fig. 2.4. This error has to be backpropagated using the same weights mentioned above. To do this, we first write the equation

$$\delta_{ok} = (d_k - o_k)\,(1 - o_k)\,o_k \tag{2.4}$$

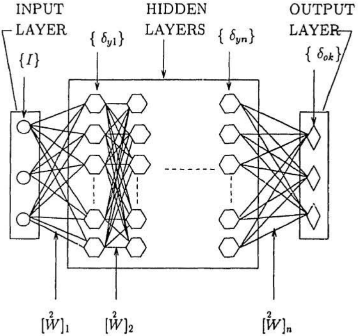


Figure 2.5 Representation of Neural Network Layers - Back-propagation of Errors

which is represented by a diamond symbol (◇) in Fig. 2.5. The error in the last hidden layer element wise is computed using

$$\delta_{yj} = y_j\,(1 - y_j) \sum_{k=1}^{K} \delta_{ok}\, w_{kj}, \quad j = 1,\dots,J \tag{2.5}$$

where y_j is the sigmoidal elemental output of the last hidden layer in Fig. 2.4 and w_{kj} is an element of the corresponding (to the right of y_j) weight matrix. This process is repeated until one computes all the elements of the first hidden layer. The weight matrix between the output layer and the last hidden layer to be used in the next cycle is recomputed as

$$[W_{kj}^2] = [W_{kj}^1] + \eta\,\{\delta_{ok}\}\{y_j\}^T \tag{2.6}$$

where the superscripts refer to the cycle number and η is the learning factor which is normally assumed between 10^{-3} to 10. The relationship for the weight matrix in other layers is given by

$$[W_{kj}^2] = [W_{kj}^1] + \eta\,\{\delta_{yj}\}\,\{y\}_{j-1}^T \tag{2.7}$$

Finally the weight matrix between the input and the first hidden layer is calculated using

$$[W_{kj}^2] = [W_{kj}^1] + \eta\,\{\delta_{yl}\}\,\{I\}^T \tag{2.8}$$

Once these weight matrices are obtained, then for any input vector one has to go through the forward computations as shown in Fig. 2.4 to obtain the output vector. This process is continued until the final set of weight matrices are obtained which yield the desired output values within the accuracy specified. Flowchart for the backpropagation method is shown in Fig.2.6.

2.2.3 Properties and its Significance

Backpropagation algorithm uses gradient descent technique to adjust the weights so as to minimize the error

$$\Delta w_k = -\eta \frac{\partial E}{\partial W_k} \tag{2.9}$$

where η is the step value. The movement of the weight vector in two-dimensional space can be observed on the error surface shown in Fig. 2.7. The weights of the network to be trained are typically initialized at small random values. The initialization strongly affects the ultimate solution. Another factor that affects the convergence is the steepness factor α, in the sigmoidal activation function given in Eq.(2.2). The effectiveness and convergence of the error backpropagation learning algorithm depend significantly on the value of the learning constant η. In general, however, the optimum value of η depends upon the problem being solved and there is no single learning constant suitable for different training cases. Activation functions with larger steepness factor produces the same effect as increasing the learning factor. So, the steepness factor is usually taken as

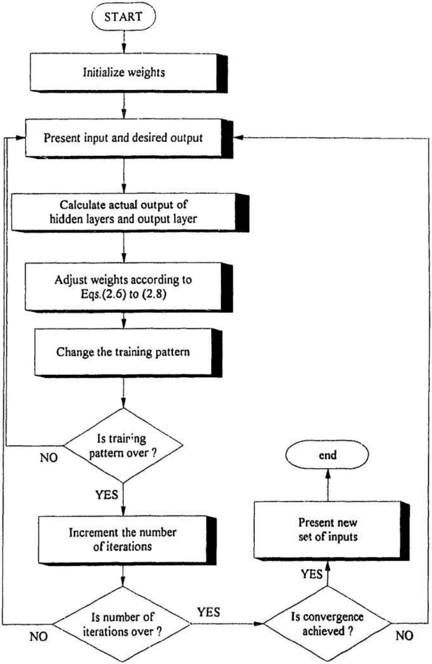


Figure 2.6 Flow Chart - Back-propagation Method

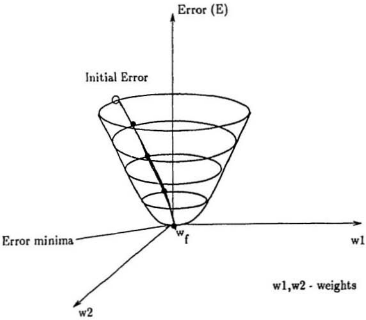


Figure 2.7 Movement of Weight Vector (2-D) on the Error Surface

1 and the learning factor is adjusted to control the convergence. However, gradient descent algorithm suffers from local minimum problem which is a common property of any nonlinear optimization algorithm.

2.2.4 Application - Singularity Problems in Velocity Analysis of Robots

When a manipulator is in singular configuration, it loses one or more degrees of freedom in the Cartesian space. Singularities in robotic manipulators may arise due to the geometrical limitations (constraints in the connecting links) of the manipulators. This problem can be handled by the use of redundant manipulators. There are two kinds of singularities:

1) Boundary singularities arise due to the geometrical limitations.

2) Interior singularities are due to two or more joint axes lining up.

Redundant manipulators also have singular configurations which have to be either avoided or handled. Near singular points, very high joint velocities result if the Cartesian velocities have components in the direction in which the arm loses mobility. These are the points at which the Jacobian matrix becomes rank-deficient.

While this problem can be handled using mathematical techniques like pseudo-inverse methods, yet it has certain limitations. The problems of singularities can be tackled at the task planning level itself by carefully designing the trajectory which avoids singular configuration. On the other hand, if due to wrong task planning or in situations where on-line computations are made and the singularity appears in the trajectory, the robot control system must be able to pass through them safely. Multiple solutions exist at singularity points.

2.2.4.1 Velocity Analysis Using Psuedo-Inverse Method

The inverse kinematics for robotic manipulators is given by (Craig, 1986)

$$\{\dot{x}\} = [J]\ \{\dot{\Theta}\} \tag{2.10}$$

where $\{\dot{\Theta}\}$ represents the joint velocity vector and $\{\dot{x}\}$ is the end-effector velocity vector and [J] is the Jacobian matrix. Therefore, the joint velocity corresponding to a given $\{\dot{x}\}$ is given by

$$\{\dot{\Theta}\} = [J]^{-1}\ \{\dot{x}\}$$

$$\{\dot{\Theta}\} = [J^{+}] \{\dot{x}\} \tag{2.12}$$

where $[J^{+}]$ is called the pseudo-inverse of the Jacobian matrix. The basic idea is to minimize the norm $\| \{\dot{x}\} - [J]\{\dot{\Theta}\} \|$ since $[J]^{-1}$ does not exist at singular points. $[J^{+}]$ gives an approximate solution satisfying the condition

$$\begin{aligned} &\min \ \| \{\dot{\Theta}\} \| \quad \text{and} \\ &\min \ \| \{\dot{x}\} - [J] \{\dot{\Theta}\} \| \end{aligned} \tag{2.13}$$

Near the singular points, $[J^{+}]$ is equivalent to $[J]^{-1}$ and pseudo-inverse finds out the exact solution. Though pseudo-inverse gives exact solution near singular points, they are not feasible because of very high values of $\{\dot{\Theta}\}$. Hence a compromise is required between feasibility and exactness in case of inverse kinematic solution near singular points. Otherwise, pseudo-inverse solutions result in undesirable continuity leading to high joint velocity which results in very high oscillations.

2.2.4.2 Velocity Analysis Using the Damped Least Squares Method

Damped Least Squares (DLS) method has been proposed by several researchers to solve inverse kinematics problems. In this method, one writes the relation between $\{\dot{\Theta}\}$ and $\{\dot{x}\}$ as

$$\{\dot{\Theta}\} = [\, [J]^{T} [J] + \lambda^{2} [I] \,]^{-1} [J]^{T} \{\dot{x}\} \tag{2.14}$$

In order to realistically achieve the desired joint velocity values, one must modify the above equation to suit the highest achievable limit of the manipulator in terms of angular velocities. In other words, we have to minimize the expression

$$\text{Min} \quad \|\{\dot{x}\} \; - \; [J] \; \{\dot{\Theta}\}\|^2 \; + \; \lambda^2 \; \| \; \{\dot{\Theta}\}\|^2 \tag{2.15}$$

where λ is known as the damping factor. $\|\{\dot{\Theta}\}\|$ is the norm of the joint velocity and the term $\|\{\dot{x}\} - [J]\{\dot{\Theta}\}\|$ accounts for the minimization of the tracking error or exactness of the solution and $\lambda^2\|\{\dot{\Theta}\}\|^2$ takes care of the feasibility of the solution. It is equivalent to solving a minimization problem,

$$\begin{gathered} \text{Min} \quad \| \; \{\dot{x}\} \; - \; [J] \; \{\dot{\Theta}\} \; \| \\ \text{subject to constraint} \\ \| \; \{\dot{\Theta}\} \; \| \; \leq \; \dot{\Theta}_{max} \end{gathered} \tag{2.16}$$

where $\dot{\Theta}_{max}$ is practical limit on manipulators joint velocity. An appropriate value of damping factor, λ, will give the desired solution. Damping factor, λ, is computed using (Maciejewski and Klein (1989))

$$\|\{\dot{\Theta}_{max}\}\|^2 \; = \; \|\{\dot{\Theta}\}^{\lambda}\|^2 \; = \; \sum_{i=1}^{r} [\frac{x_i^{'} \; \sigma_i}{\sigma_i^2 \; + \; \lambda^2}]^2 \tag{2.17}$$

where $x_i' = \{u_i\}^T \{\dot{x}\}$ and r is the rank of the matrix and σ_i, $\{v_i\}$ and $\{u_i\}$ are obtained from Singular Value Decomposition (SVD) of the Jacobian matrix [J]. To express Eq.(2.17) in a simple manner one can write

$$\{\dot{\Theta}_{max}\} = \begin{Bmatrix} \dot{\Theta}_1^* \\ \dot{\Theta}_2^* \\ \vdots \\ \dot{\Theta}_n^* \end{Bmatrix} \tag{2.18}$$

where superscript * represents the maximum allowable value for that particular joint. At first, one evaluates

$$\| \{\dot{\Theta}_{max}\} \|^2 = \dot{\Theta}_1^{*^2} + \dot{\Theta}_2^{*^2} + \dots + \dot{\Theta}_n^{*^2} \tag{2.19}$$

and then using Eqs.(2.18) and (2.19) and using a nonlinear optimization technique, finds the value of λ which would minimize the function

$$[\frac{x_1'\sigma_1}{\sigma_1^2 + \lambda^2}]^2 + [\frac{x_2'\sigma_2}{\sigma_2^2 + \lambda^2}]^2 + \dots + [\frac{x_r'\sigma_r}{\sigma_r^2 + \lambda^2}]^2 - \|\{\dot{\Theta}_{max}\}^2\| \tag{2.20}$$

The optimal value of λ is then substituted in the following equation to get the damped joint velocity vector

$$\{\dot{\Theta}^{(\lambda)}\} = \sum_{i=1}^{r} (\frac{\sigma_i}{\sigma_i^2 + \lambda^2})\{v_i\}\{u_i\}^T\{\dot{x}\} \tag{2.21}$$

Unfortunately, both these methods, i.e., the pseudo-inverse as well as the DLS are, expensive in terms of computations, and not suitable for on-line tasks. It is important to select an appropriate value of damping factor, λ. A low value of λ minimizes the tracking error and gives rise to undesirable high joint velocities. A high value of λ accounts for the robustness but leads to low tracking accuracy (Chiaverni, 1992). The term $\sigma_i / (\sigma_i^2 + \lambda^2)$ far away from singular points, becomes (as $\lambda \rightarrow 0$)

$$\frac{\sigma_i}{\sigma_i^2 + \lambda^2} \simeq \frac{1}{\sigma_i} \tag{2.22}$$

DLS solution overcomes two main limitations of pseudo-inverse solution near singular configurations namely the discontinuity and infeasible high joint velocities. But SVD calculations are computationally expensive and error compromise is high. In theory, it is possible to calculate the damping factor λ at each of the points along the trajectory (near singular points) but an optimal value of λ, if chosen for all the points would minimize the computational burden.

2.2.4.3 Velocity Analysis Using Neural Network Method

A single layer neural network is capable enough to learn the relationship between the Cartesian and joint velocities near singular configurations. This is a highly non-linear mapping where joint velocities increase at a higher rate.

Considering the fact that in the real-time control problems one has to keep in mind both, the errors (displacement, velocity, force etc.), as well as the computational efficiency (real-time computations); therefore, in the present work, the relationship between the Cartesian velocity and the joint velocity vectors was established on off-line basis using the neural networks over a segment of a trajectory. This circumvents the on-line computational requirements of the joint velocity vector, as was done by researchers (Maciejewski and Klein, 1989) mentioned earlier in Chapter 1. In their method, the calculations were required to be done on a point by point basis but which results in the slowing down of the actual task. The additional benefit of the neural network method is that one can achieve better accuracy also.

The input vector is the Cartesian velocity vector and the output vector is the joint velocity vector. The training is performed on either side of the singularity point (Sharan and Balasubramanian, 1993). The following points are kept in mind while performing the training:

1) Maintain the joint velocities close to the upper feasible limit near the singular point.

2) A smooth transition curve of joint velocities is required on either side of singularity points.

3) Minimize the errors between the actual and achievable joint velocities.

4) Have optimal number of training tasks to achieve the non-linear mapping.

2.2.4.4 Case Study

To illustrate the theory developed so far, the task of moving the end effector along a trajectory consisting of a segment of a circle and a radial line is shown in Fig. 2.8. The point of singularity was the point B in this figure. While performing the task a constant tangential velocity along the radial path was desired. This task was performed using (a) A planar two degrees of freedom (DOF) manipulator (b) PUMA-560 manipulator. These are typical manipulators widely used by various researchers in the field of robotics.

A Planar Two-Link Manipulator

A simple two-link manipulator is shown in Fig. 2.9. The velocity relationships between joint velocity and the Cartesian velocity for this manipulator is given by

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \begin{pmatrix} \dot{\Theta}_1 \\ \dot{\Theta}_2 \end{pmatrix} \tag{2.23}$$

where x and y are coordinates of the path followed by the end-effector expressed in universal frame. The inverse of the Jacobian is written as

$$[J]^{-1} = \frac{1}{l_1\, l_2\, s_2} \begin{bmatrix} l_2 c_{12} & l_2 s_{12} \\ -l_1 c_1 - l_2 c_{12} & -l_1 s_1 - l_2 s_{12} \end{bmatrix} \tag{2.24}$$

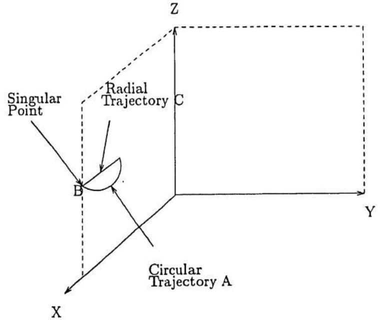


Figure 2.8 Trajectory Used for PUMA-560 Manipulator

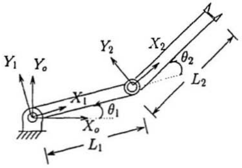


Figure 2.9 A Planar Two-Link Manipulator

where $c_1 = \cos\theta_1$; $s_1 = \sin\theta_1$; $s_2 = \sin\theta_2$; $s_{12} = \sin(\theta_1 + \theta_2)$; and $c_{12} = \cos(\theta_1 + \theta_2)$. Here, θ_1 and θ_2 are the joint angles of the manipulator and l_1 and l_2 are the link lengths. One can find from the above equation, the singularity arises when $s_2 = 0$ ($\theta_2 = 0$) i.e. when as the arm stretches outward and both joint rates go to infinity. The two-link manipulator is moving its tip at a constant tangential velocity of 0.03 m/s. The link lengths used were $l_1 = 0.4$ m and $l_2 = 0.2$ m; the radius of the circle was 0.07 m and the damping factor λ obtained from nonlinear optimization routine was 0.0077.

PUMA-560 Manipulator

The forward kinematic relationship between Cartesian coordinates and joint coordinates for a PUMA-560 manipulator (shown in Fig. 2.10) is given by

$$\begin{aligned} x_o &= a_3 c_1 c_{23} - d_4 c_1 s_{23} + a_2 c_1 c_2 - d_3 s_1 \\ y_o &= a_3 s_1 c_{23} - d_4 s_1 s_{23} + a_2 s_1 c_2 - d_3 c_1 \\ z_o &= -a_3 s_{23} - d_4 c_{23} - a_2 s_2 \end{aligned} \tag{2.25}$$

The link parameters for this manipulator are shown in Table 2.1. While performing the task, the desired tangential velocity along the circular path for PUMA-560 was 0.5 m/s and it was the same velocity along the radial path also. The maximum achievable limit $\dot{\theta}_{max}$ for each of the manipulators was taken to be 25 rad/s.

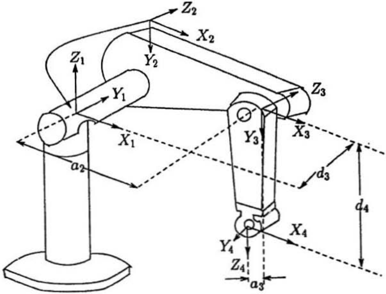


Figure 2.10 PUMA-560 Manipulator

Table 2.1 Link Parameters of PUMA-560 Manipulator

Link i	α_i	θ_i	H_i	D_i
	(degrees)	(degrees)	(m)	(m)
1	0	θ_1	0	0
2	-90	θ_2	0.4318	0
3	0	θ_3	0.02032	0.127
4	-90	θ_4	0	0.4318

2.2.4.5 Results and Discussion

At first a PUMA-560 manipulator is considered. The $\{\dot{\Theta}\}$ vector was obtained using Eqs. (2.11) or (2.12) depending upon the proximity of the point to the point of singularity. The results obtained are shown in Figs.2.11 to 2.14. Similarly, the results for damped least squares method using Eq.(2.21) are also shown in these figures. It is quite clear here that the required values near the point of singularity are high and not achievable because this manipulator has a maximum $\|\dot{\Theta}\|$ equal to 25 rad/s. For the neural network analysis, the input and the output values for the learning phase were specified in accordance with Eqs. (2.11) or the maximum limits over the trajectory. After this, the weight matrix [W] which relates $\{\dot{x}\}$ and $\{\dot{\Theta}\}$ as

$$\{\dot{x}\} = [W]\{\dot{\Theta}\} \tag{2.26}$$

was obtained using Eqs. (2.1) to (2.8). The results are shown in Figs. 2.11 to 2.14. In all these figures, the results obtained by neural network analysis are far more accurate than those obtained by the DLS method i.e. the neural network method gives the norm values much closer to the values given by Eqs. (2.11) and (2.12) than the DLS method. Secondly, the error in $\|\dot{x}\|$ (to the right of point B) in Fig.2.14 in the case of neural network method, is due to the maximum achievable limit and not due to the method itself. In addition, as mentioned earlier, the DLS method requires much more on-line computations. These facts were further confirmed in the case of two-link manipulator as shown in Fig.2.9. The results in this case are shown in Figs. 2.15 to 2.18. The trajectory in this case was the same as used earlier.

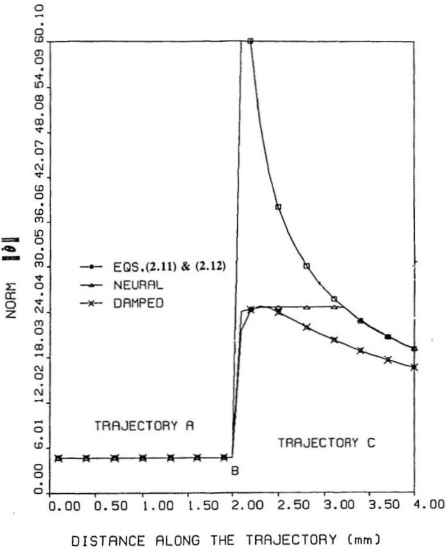


Figure 2.11 Variation of the Norm of the Angular Velocity Vector, $\|\dot{\theta}\|$, Along the Trajectory of a PUMA-560 Manipulator

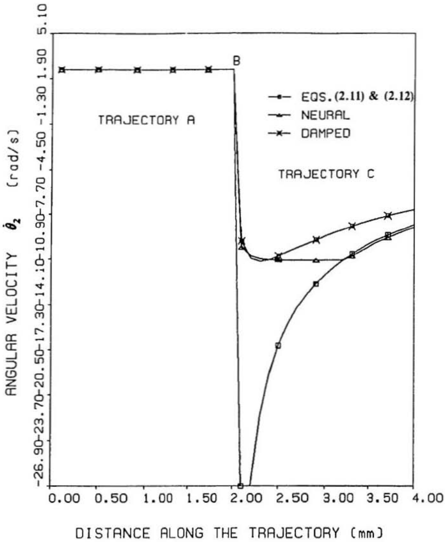


Figure 2.12 **Variation of the Angular Velocity, $\dot{\theta}_2$, Along the Trajectory of a PUMA-560 Manipulator**

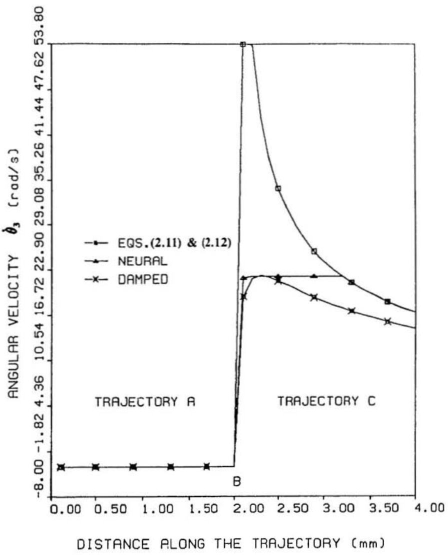


Figure 2.13 Variation of the Angular Velocity, $\dot{\theta}_3$, Along the Trajectory of a PUMA-560 Manipulator

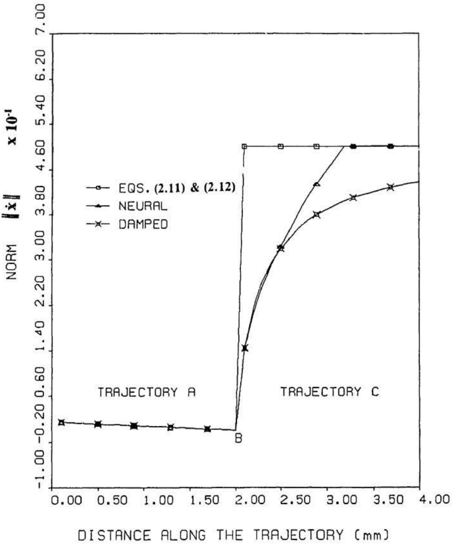


Figure 2.14 Variation of the Norm of the Cartesian Velocity Vector, $\|\dot{x}\|$, Along the Trajectory of a PUMA-560 Manipulator

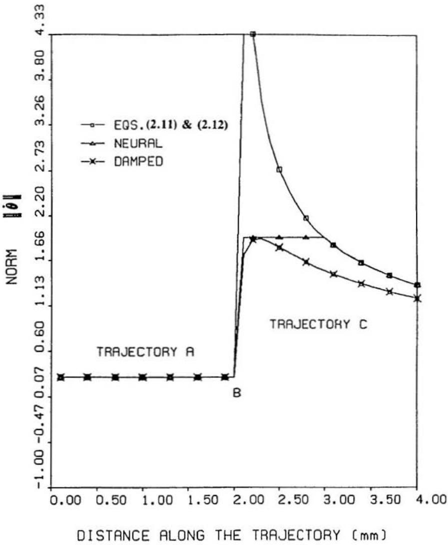


Figure 2.15 **Variation of the Norm of the Angular Velocity Vector, $\|\dot{\theta}\|$, Along the Trajectory of a Two-Link Manipulator**

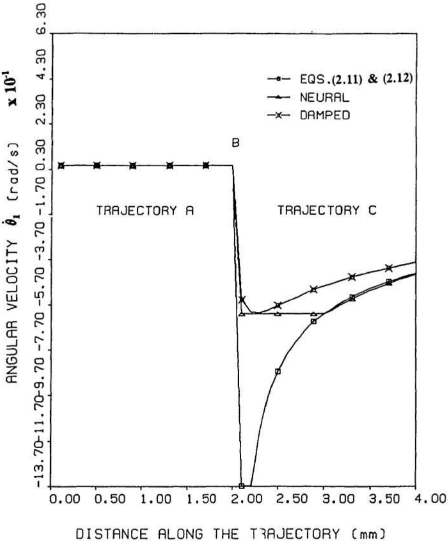


Figure 2.16 Variation of the Angular Velocity, $\dot{\Theta}_1$, Along the Trajectory of a Two-Link Manipulator

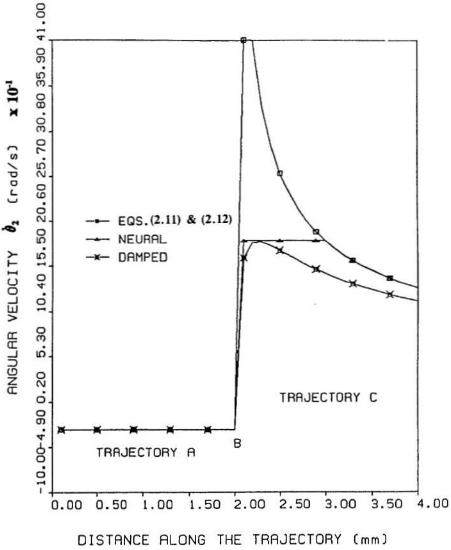


Figure 2.17 Variation of the Angular Velocity, $\dot{\theta}_2$, Along the Trajectory of a Two-Link Manipulator

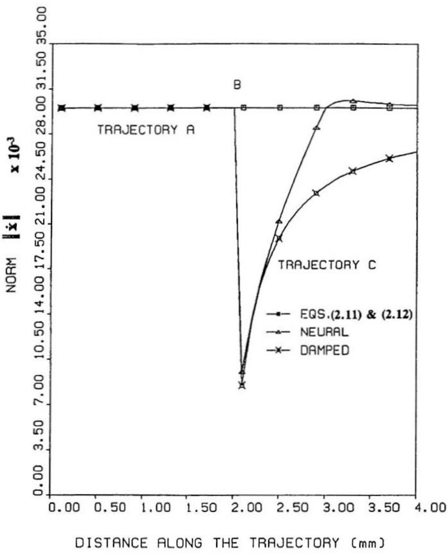


Figure 2.18 **Variation of the Norm of the Cartesian Velocity, $\|\dot{\mathbf{x}}\|$, Along the Trajectory of a Two-Link Manipulator**

2.3 LP-Neuro Method

2.3.1 A New Approach - Development of LP-Neuro Method

As discussed earlier, a method to trade-off the accuracy and computational efficiency, is sought. A new method called LP-neuro method (Balasubramanian and Sharan, 1993) is developed in this section which utilizes the faster convergence property of linear programming; this result in better error minimization. The architecture of this method is similar to the feed forward error backpropagation neural network except that a single layer is enough. The activation function used in this case is a linear activation function with slope m and intercept c. A nonlinear curve is approximated by several linear curves of different slopes and intercepts. The error minimization objective function has weights and intercepts as linear variables and the slope as non-linear variables which is solved using Hookes and Jeeves method.

2.3.1.1 LP-Neuro Method - Type 1

In the neural network method (as used in Sec. 2.2.4.3), the input {I} and desired output {D} vectors are related by the equation

$$\{D\} = [W]\{I\} \tag{2.27}$$

However, due to errors, one obtains a vector {O} instead of {D}. The weight matrix [W] which relates the input and output vector in that case is given by

$$\{H\} = \begin{Bmatrix} h_1 \\ h_2 \\ \vdots \\ h_j \end{Bmatrix} = \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{kl} \\ w_{21} & w_{22} & \cdots & w_{k2} \\ & \vdots & \vdots & \\ w_{jl} & w_{j2} & \cdots & w_{jk} \end{bmatrix} \begin{Bmatrix} i_1 \\ i_2 \\ \vdots \\ i_k \end{Bmatrix} \tag{2.28}$$

The LP-neuro method is diagrammatically explained in Fig. 2.19. The functional relationship between {H} and {O} can be written as

$$o_j = f(h_j) = h_j \tag{2.29}$$

and

$$\{O\} = [W]\{I\} \tag{2.30}$$

The element o_j is shown by a square symbol (□) in Fig. 2.19 and h_j are the elements of vector {H}. This is similar to the sigmoidal functional relationship used in backpropagation method, where one uses the equation

$$o_j = f(h_j) = \frac{1}{1 + \exp(-h_j)} \tag{2.31}$$

One of the ways to obtain the set of weight matrices with minimum error would be by

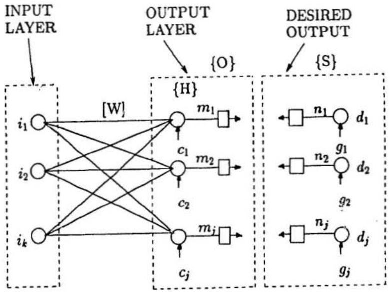


Figure 2.19 Diagrammatic Representation of the Network - LP-Neuro Method

writing a cost function E in the following form:

$$\text{Minimize } E = (d_1 - o_1) + (d_2 - o_2) + \ldots + (d_j - o_j)$$

or

$$\text{Minimize } E = (d_1 + d_2 + \ldots + d_j) - (w_{11}\, i_1 + \ldots + w_{jk} i_k)$$

Subject to

$$\begin{aligned} w_{11}\, i_1 + w_{12}\, i_2 + \ldots + w_{1k}\, i_k &= d_1 \\ \vdots \qquad \vdots \qquad \vdots & \\ w_{j1}\, i_1 + w_{j2}\, i_2 + \ldots + w_{jk}\, i_k &= d_j \end{aligned} \tag{2.32}$$

where d_i are the elements of the desired output vector {D} in Eq. (2.27), and w_{jk} are the weights. Eq. (2.32) has (j*k) weights and they can be collected in a single dimensional array or a vector as

$$\{W\} = \begin{Bmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{jk} \end{Bmatrix} = \begin{Bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{j*k} \end{Bmatrix} \tag{2.33}$$

This vector {W} contains j*k unknowns. Since these can take positive or negative values, each of these can be replaced by two positive variables (a requirement for solving linear programming). For example, one can write $w_1 = v_1 - v_2$, $w_2 = v_3 - v_4$ etc. Substituting w_i in terms of v_i, one can rewrite the Eq. (2.32) as

Minimize $E = (d_{1} + d_2 + \dots + d_j) - (i_1v_1 - i_1v_2 + i_2v_3 - i_2v_4 + \dots + i_1v_{2k+1} -$

$i_1v_{2k+2} + \dots + i_1v_{4k-1} - i_1v_{4k+2} + \dots + i_kv_{2jk-1} - i_kv_{2jk})$

Subject to

$$[A]\{V\} = \{D\} \tag{2.34}$$

The details of the coefficient matrix [A] can be shown as

$[A] =$

$$\begin{bmatrix} i_1 & -i_1 & i_2 & -i_2 & \cdots & i_k & -i_k & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & i_1 & -i_1 & i_2 & -i_2 & \cdots & i_k & -i_k & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & . & . & . & . & \vdots & . & \vdots & . & . & . & . & \vdots & \vdots & . & . & . & . & \vdots & . & . & . \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & i_1 & -i_1 & i_2 & -i_2 & \cdots & i_k & -i_k \end{bmatrix} \tag{2.35}$$

Just like {W}, one can also write

$$\{V\} = \begin{Bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{2k} \\ \vdots \\ v_{2jk} \end{Bmatrix} \tag{2.36}$$

The matrix [A] contains a number of zeroes in a given row. Here the non-zero elements occur together and only once in a given row. It is well known that the optimal cost function for such problems, involving sparsity, can be obtained much more quickly (McCormick, 1990) as compared to a case where [A] is a dense matrix.

2.3.1.2 LP-Neuro Method - Type 2:

Further refinements on the above method can be made by replacing the activation function given in Eq. (2.29) by another function given by

$$f(h_j) = m_j h_j + c_j \tag{2.37}$$

In Eq.(2.37) each variable h_i has a corresponding scalar m_j and a constant c_j. The new relationship corresponding to Eq. (2.30) will be

$$\{O\} = [M]\,[W]\,\{I\} + \{C\} \tag{2.38}$$

where [M] = diagonal slope matrix, which has scalar m_j as its diagonal elements.

A similar activation function for the output side can be written as

$$\{S\} = [N]\{D\} + \{G\} \tag{2.39}$$

where [N] is a diagonal matrix. The matrices [N] and {G} are analogous to [M] and {C} in Eq. (2.38). The new formulation using Eqs. (2.38) and (2.39) will be

Minimize $E_l = [N]\{D\} + \{G\} - [M][W]\{I\} - \{C\}$

subject to

$$[M][W]\{I\} + \{C\} = [N]\{D\} + \{G\}$$

or in the scalar form, it can be rewritten as

$$\text{Minimize } E_l = n_1d_1 + n_2d_2 + \dots + n_jd_j - m_1\{w_{11}i_1 + w_{12}i_2 + \dots + w_{1k}i_k\} - m_2\{ w_{21}i_1 + w_{22}i_2 + \dots + w_{2k}i_k\} - \dots -m_j\{w_{j1}i_1 + w_{j2}i_2 + \dots + w_{jk}i_k\} + g_1 + g_2 + \dots + g_j - c_1 - c_2 - \dots - c_j$$

subject to

$$\begin{aligned} m_1\{w_{11}\, i_1 + w_{12}\, i_2 + \dots + w_{1k}\, i_k\} + c_1 &= n_1d_1 + g_1 \\ \vdots \qquad \vdots \qquad \vdots& \\ m_j\{w_{j1}\, i_1 + w_{j2}\, i_2 + \dots + w_{jk}\, i_k\} + c_j &= n_jd_j + g_j \end{aligned} \tag{2.40}$$

Again here, the weights w_{jk}, c_j and g_j are replaced by two positive numbers as before in the following manner:

$$\begin{aligned} w_i &= v_l - v_{l+1} \\ c_i &= v_{cl} - v_{cl+1} \text{ and} \\ g_i &= v_{gl} - v_{gl} \qquad i = 1,2,\dots \\ & \qquad\qquad l = 1,3,\dots \end{aligned} \tag{2.41}$$

After these substitutions, one arrives at

$$\text{Minimize } E_l = n_1d_1 + n_2d_2 + \dots + n_jd_j - m_1(i_1v_1 - i_1v_2 + i_2v_3 - i_2v_4 + \dots) + m_2(i_1v_{2k+1} - i_1v_{2k+2} + \dots) + m_3(i_1v_{4k+1} - i_1v_{4k+2} + \dots) + m_j(\dots + i_kv_{2jk-1} - i_kv_{2jk}) + v_{g1} - v_{g2} + \dots -$$

$v_{c1} + v_{c2} - \ldots$

Subject to

$$
\begin{aligned}
m_1\{i_1 v_1 - i_1 v_2 + \ldots - i_k v_{2k}\} \quad + v_{c1} - v_{c2} \quad &= n_1 d_1 + v_{g1} - v_{g2} \\
m_2\{i_1 v_{2k+1} - i_1 v_{2k+2} + \ldots - i_k v_{4k}\} + v_{c3} - v_{c4} \quad &= n_2 d_2 + v_{g3} - v_{g4} \\
\vdots \qquad\qquad \vdots \qquad\qquad &\vdots \\
m_j\{i_1 v_{2k(j-1)+1} - \ldots + i_k v_{2jk}\} + v_{c2j-1} - v_{c2j} \quad &= n_j d_j + v_{g2j-1} - v_{g2j}
\end{aligned}
\tag{2.42}
$$

This equation can be written in the standard LP notation as,

$$
\begin{aligned}
\text{Minimize } E_1 = {} & n_1 d_1 + n_2 d_2 + \ldots + n_j d_j - m_1(i_1 v_1 - i_1 v_2 + i_2 v_3 - i_2 v_4 + \ldots) + \\
& m_2(i_1 v_{2k+1} - i_1 v_{2k+2} + \ldots) + m_3(i_1 v_{4k+1} - i_1 v_{4k+2} + \ldots) + m_j(\ldots + i_k v_{2jk-1} - i_k v_{2jk}) \\
& + v_{g1} - v_{g2} + \ldots - v_{c1} + v_{c2} - \ldots
\end{aligned}
$$

subject to

$$[A]\{V\} = \{D\}$$

where the coefficient matrix [A] is given by

$$
\begin{bmatrix}
n_1 i_1 & -m_1 i_1 & m_1 i_k & -m_1 i_k & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & m_2 i_1 & -m_2 i_1 & m_2 i_k & -m_1 i_k & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & m_{j} & -m_{j} & m_{j} & -m_{j} & 1 & -1 & -1 & 1 & &
\end{bmatrix}
\tag{2.43}
$$

In the above equation, d_j are the desired values and i_k are the input values; v_i, v_{ci} and v_{gi} are the unknown variables and so are m_j and n_j. The problem shown in Eq.

(2.42) is non-linear because of the occurrence of product terms such as $m_i v_i$ ... etc. However, if this problem is combined with another multi-variable optimization problem containing all m_i only, then the problem involving the remaining variables can be solved by the linear programming method. Since, the number of variables far exceed the number of constraints, it would be better to solve for m_i using non-linear optimization and the remaining variables which include weights, by linear method. This method clearly differs from others because, for the majority of the variables (other than m_i), the linear method yields faster convergence as compared to totally non-linear method. The additional advantage in the linear method is that one can exploit the sparsity in $|A|$ matrix in Eq. (2.35). For example, if the iterative values of m_i are obtained from the non-linear method, and substituted in Eq. (2.42), then the resulting problem becomes linear and can be solved using the Revised Simplex Method (Siddal, 1982). The actual flow chart of the combined method is shown in Fig.2.20. In fact, one can attempt to solve using a single m value instead of j different m_j values and check for convergence. If results are satisfactory, then the problem can be reduced to single variable non-linear optimization problem followed by linear programming.

2.4 Applications of the LP-Neuro Method

2.4.1 Function Generation

Approximating a sine curve has been a test for non-linear mapping carried out by several researchers. The non-linear mapping of a sine curve using backpropagation is

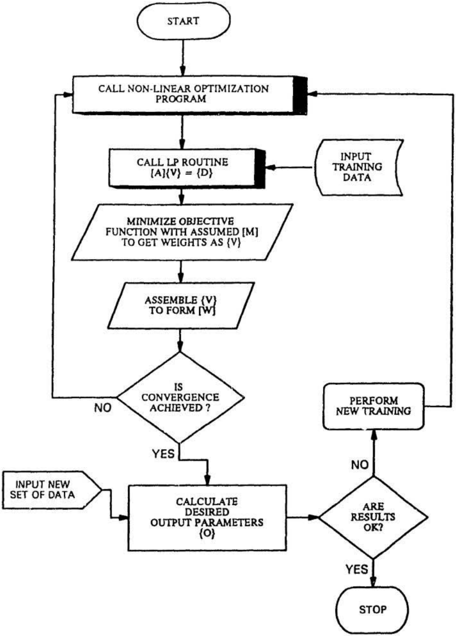


Figure 2.20 Flow Chart - LP-Neuro Method

described in (Zurada, 1992). The sine curve taken is

$$y = a\ Sin(b\,x) \tag{2.44}$$

where $a = 0.8$ and $b = \pi$. The same example was taken here for the case study. Instead of using several bias terms as done in Zurada (1992), a different approach was followed in the present work. To do this, 21 points along the sine curve in a period were taken for training. In order to identify this curve, the training was performed on different sine curves having different values of a and b. In all cases, 21 points were used. After this, the same number of points for this particular curve was provided as input and corresponding output was checked on the sine curve. The results using Eq. (2.34) and Eq. (2.44) are shown in Fig. 2.21. The results in this figure show that in the first quarter period, the BP method yields slightly better results than the LP method (Eq. 2.33) but not all through. On the other hand, the LP-neuro method (Eq. 2.41) is always accurate and decidedly the method to be used. In view of the above, only the LP-neuro method and BP method were used in the next two examples. Furthermore, a single value of m yielded results which were sufficiently accurate. Hence, the same procedure is followed in solving the next two examples.

2.4.2 Acceleration Analysis of a Two-link Planar Manipulator

A two-link planar manipulator having revolute joints is shown in Fig.2.22. The end-effector, P, is made to follow a circular trajectory at a constant tangential velocity, v_t, of magnitude equal to 0.15 m/s. (X_o, Y_o) represents the global coordinate system and

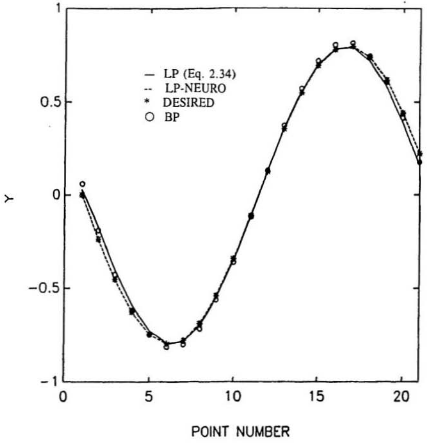


Figure 2.21 Comparison of Values for the Sine Curve (LP, LP-Neuro Method and BP Method and the Desired Values)

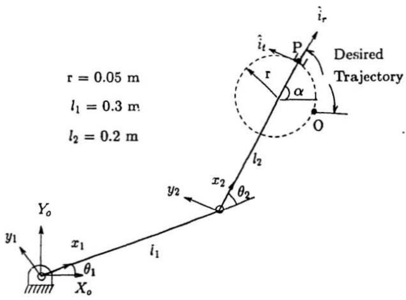


Figure 2.22 **A Planar Two-Link Manipulator and the Trajectory used for Acceleration Analysis**

(x_i, y_i) represent the local coordinate frame of the link i and the joint variables. θ_1 and θ_2 represent the rotational displacements.

The joint variables, θ_1 and θ_2 are related to the position of the end effector (X_P, Y_P) in Cartesian space through the following equations:

$$\theta_1 = Atan2(r_j, r_i) + Atan2(\sqrt{t}, r_k) \tag{2.45}$$

where $r_i = 2\, Y_P\, l_1$; $r_j = 2\, X_P\, l_1$; $r_k = Y_P^2 + X_P^2 + l_1^2 - l_2^2$; $t = r_j^2 + r_i^2 - r_k^2$;

and

$$\theta_2 = Atan2(Y_P - l_1 \sin\theta_1\,,\, X_P - l_1 \cos\theta_1\,) - \theta_1 \tag{2.46}$$

Differentiating Eqs. (2.45) and (2.46) with respect to time, we get

$$\begin{Bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{Bmatrix} = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{bmatrix}^{-1} \begin{Bmatrix} \dot{X}_P \\ \dot{Y}_P \end{Bmatrix} \tag{2.47}$$

where l_1, l_2 are the lengths of links 1 and 2 respectively; $c_1 = \cos\theta_1$; and $c_{12} = \cos(\theta_1 + \theta_2)$ etc. The acceleration of the tip moving along the circular path in the radial direction is given by

$$\bar{a}_p = -\omega^2\, r\, \hat{i}_r = \frac{-v_t^2}{r}\, \hat{i}_r \tag{2.48}$$

Resolving the tip acceleration in global coordinate system we get

$$\begin{Bmatrix} \ddot{X}_P \\ \ddot{Y}_P \end{Bmatrix} = \begin{Bmatrix} \dfrac{-v_t^2}{r} \cos(\alpha) \\ \dfrac{-v_t^2}{r} \sin(\alpha) \end{Bmatrix} \tag{2.49}$$

Here, one can obtain by differentiating the Eq. (2.49)

$$\begin{Bmatrix} \ddot{X}_p \\ \ddot{Y}_p \end{Bmatrix} = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{bmatrix} \begin{Bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{Bmatrix}$$

$$+ \begin{bmatrix} -l_1 c_1 \dot{\theta}_1 - l_2 c_{12}(\dot{\theta}_1 + \dot{\theta}_2) & -l_2 c_{12}(\dot{\theta}_1 + \dot{\theta}_2) \\ -l_1 s_1 \dot{\theta}_1 - l_2 s_{12}(\dot{\theta}_1 + \dot{\theta}_2) & -l_2 s_{12}(\dot{\theta}_1 + \dot{\theta}_2) \end{bmatrix} \begin{Bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{Bmatrix} \tag{2.50}$$

In robotic control, there is a great need for minimization of on-line computations. Rather than performing numerous computations as shown by Eqs. (2.45) to (2.50), it is desirable to find a linear relationship between the vectors given by

$$
\begin{Bmatrix} \ddot{\theta}_{11} \\ \ddot{\theta}_{21} \\ \ddot{\theta}_{12} \\ \ddot{\theta}_{22} \\ \vdots \\ \ddot{\theta}_{1n} \\ \ddot{\theta}_{2n} \end{Bmatrix} = [\,W_1\,] \begin{Bmatrix} r \\ \ddot{X}_{p1} \\ \ddot{Y}_{p1} \\ \vdots \\ \ddot{X}_{pn} \\ \ddot{Y}_{pn} \end{Bmatrix} \tag{2.51}
$$

on the off-line basis first. The first subscript, i, in θ_{ij} represents angular acceleration of a particular link, and the second one, j, the point along the trajectory. The acceleration elements $\ddot{X}_{pi}$, $\ddot{Y}_{pi}$ etc., are computed using Eq. (2.49).

In control problems, one needs to know $\{\ddot{\theta}\}$, as the end effector traverses the trajectory. Usually, on-line computations are done on a point by point basis i.e., one has to carry out computations given by Eqs. (2.45) to (2.50) at every point. In Eq. (2.51) above, if we obtain the $[W_1]$ on an off-line basis then, one can compute $\{\ddot{\theta}\}$ on an on-line basis for any set of points along the trajectory, much more rapidly. For training, circles of different radii were used and in all cases 20 points were selected on different concentric circles. Figs. (2.23) and (2.24) show the results obtained by Eqs. (2.45) to (2.50). The same problem was also done using the BP method and shown in these figures. These figures clearly show that one can very successfully use neural network concept in general, and LP-neuro method in particular, in arriving at a better control strategy for robotic manipulators. The constant velocity requirement of the end effector is present in many industrial applications such as welding, painting etc.,

2.4.3 Solution of Torque and Reaction Forces of the Two-link Manipulator

The iterative Newton-Euler dynamics algorithm (see Table 2.2) has been used very extensively by various researchers. The link parameters used in this case are shown in Table 2.3. Here too, the number of computations is quite large to be performed on an on-line basis. In this method, kinematic solutions are carried out on a link by link basis starting from the base (refer to Fig. 2.22). When all the kinematic computations are completed, then the dynamic computations start from the outer link to the inner link. The details can be seen in (Craig, 1986) and are not mentioned here. Even in this case it would be better to have the following relationship on an off-line basis:

Table 2.2 The Iterative Newton-Euler Dynamics Algorithm

FORWARD RECURSION

Step 1: $\{\omega\}_i = [R]_i^T \{\omega\}_{i-1} + \{z\} \dot{\Theta}_i$

Step 2: $\{\alpha\}_i = [R]_i^T \{\alpha\}_{i-1} + \{z\} \ddot{\Theta}_i + [R]_i^T \{\omega\}_i \times \{z\} \dot{\Theta}_i$

Step 3: $\{a\}_i = [R]_i^T (\{\alpha\}_{i-1} + \{\alpha\}_{i-1} \times \{p\}_{i-1} + \{\omega\}_{i-1} \times \{p\}_{i-1})$

$\qquad + \{z\} \ddot{\Theta}_i + 2 \times [R]_i^T \{\omega\}_{i_i} \times \{z\} \dot{\Theta}_i$

Step 4: $\{a\}_{a} = \{a\}_i + \{\alpha\}_i \times \{s\}_i + \{\omega\}_i \times \{\omega\}_i \times \{s\}_i$

Step 5: $\{F\}_i = m_i \{a\}_a$

Step 6: $\{N\}_i = [I]_i \{\alpha\}_i + \{\omega\}_i \times ([I]_i \{\omega\}_i)$

BACKWARD RECURSION

Step 7: $\{f\}_i = \{F\}_i + [R]_{i+1} \{f\}_{i+1}$

Step 8: $\{n\}_i = [R]_{i+1} \{n\}_{i+1} + \{N\}_i + \{s\}_i \times \{F\}_i + \{p\}_i \times ([R]_{i+1} \{f\}_{i+1})$

Step 9: $\tau_i = \{z\} \{n\}_i^T = n_{iz}$

Table 2.3 Link Parameters of the Two-Link Manipulator

DETAILS	LINK 1	LINK 2	UNITS
LINK LENGTH	0.25	0.16	m
LINK CENTER OF GRAVITY	0.20	0.14	m
MASS	9.50	5.00	kg

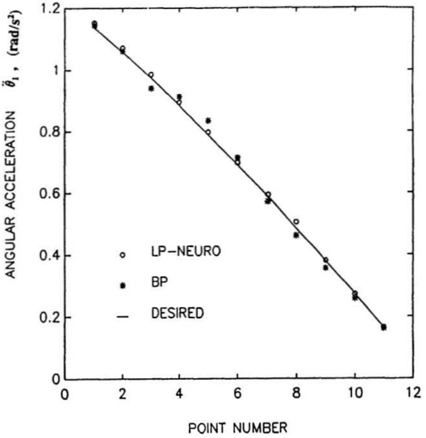


Figure 2.23 Variation of $\ddot{\Theta}_1$, Along the Trajectory

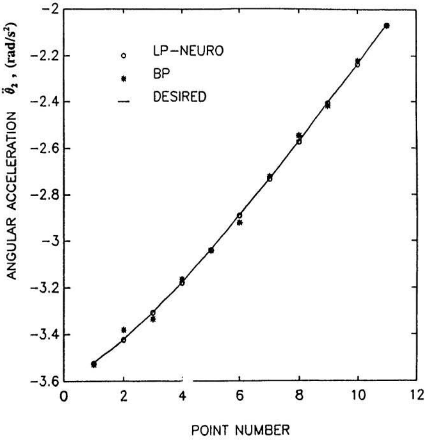


Figure 2.24 Variation of $\ddot{\Theta}_2$, Along the Trajectory

$$
\begin{Bmatrix} f_x^1 \\ f_y^1 \\ \tau^1 \\ f_x^2 \\ f_y^2 \\ \tau^2 \end{Bmatrix} = [\,W_2\,] \begin{Bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{Bmatrix} \tag{2.52}
$$

In Fig. 2.22, as the end effector P moves along the trajectory, due to the applied torques (τ^1 and τ^2) by the motors on the respective links, the reactions forces (f_x^1, f_y^1 etc.,) are produced. One has to know not only the torques but also these reactions forces at every point along the trajectory. The computations were carried out for the two-link manipulator along the shown trajectory. Twenty points were used here to obtain [W_2]. Results are shown in Figs. 2.25 to 2.30. The results clearly show that the overall error is quite small. It is less than 0.2 % in all the cases obtained by LP-neuro method. Such an accurate relationship would be of great help for on-line control of such systems. One can also see in these figures that LP-neuro method yields better results than BP method.

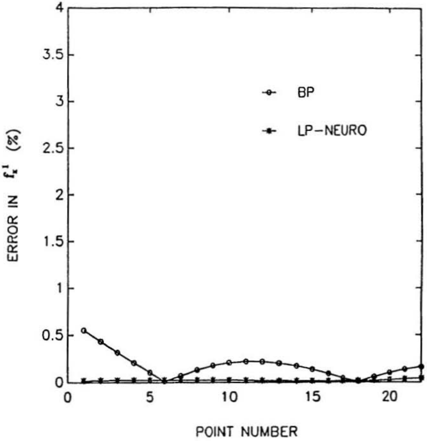


Figure 2.25 Error Values of f_x Acting on Link 1

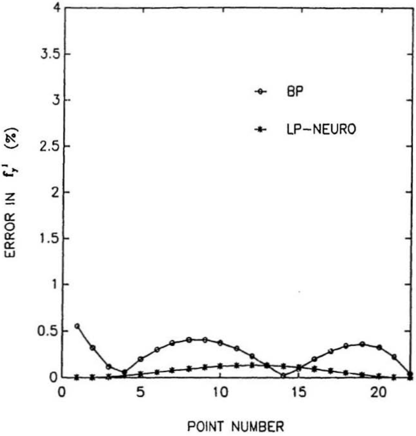


Figure 2.26 Error Values of f_y Acting on Link 1

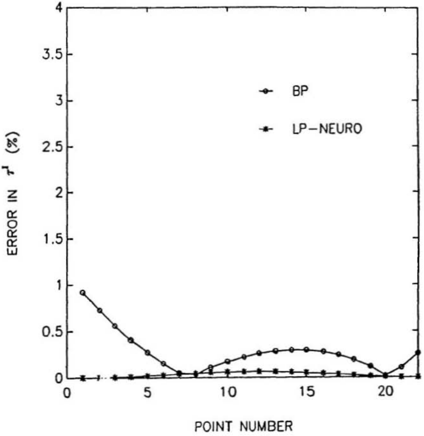


Figure 2.27 Error Values of τ_1 Acting on Link 1

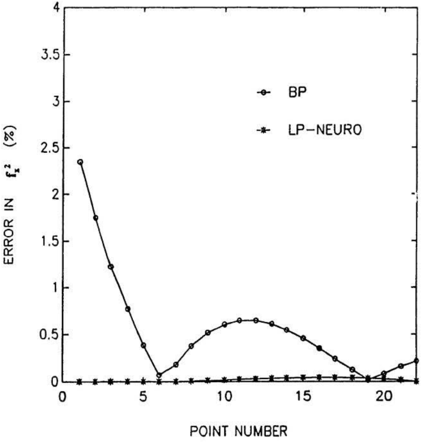


Figure 2.28 Error Values of f_x Acting on Link 2

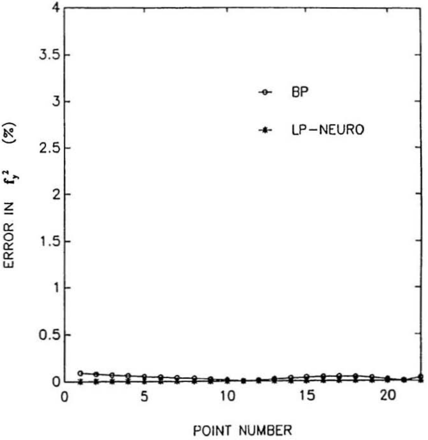


Figure 2.29 Error Values of f_y Acting on Link 2

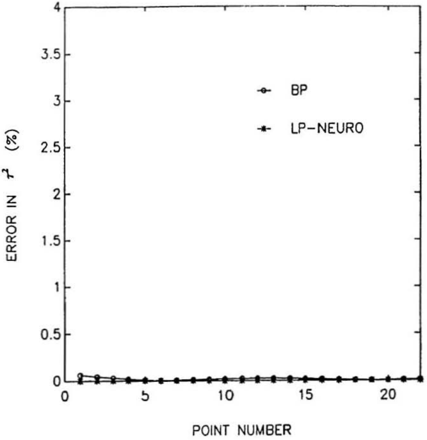


Figure 2.30 Error Values of τ_2 Acting on Link 2

2.5 Conclusions

In the velocity analysis near singular configurations, a mathematical relationship between the angular velocity and Cartesian velocity vectors was established using the DLS method and the neural network over a segment of a trajectory. The validity of this relationship was verified using two numerical examples.

Next in the LP-neuro method, the elements of the weight matrix were formulated as the unknown variables of the LP problem with equality constraints. These equations were then modified such that the coefficient matrix [A] was sparse. In another case, a general linear relationship for the activation function was used and the resulting problem was solved using a combination of LP and a single variable non-linear optimization method. The utility of the algorithm developed was illustrated using three case studies, two of which had applications in the on-line control of robotic manipulators.

Based on the work in this chapter, the following conclusions can be drawn:

1. The neural network method yields more accurate results than the DLS method.
2. The neural network method established relationship over a segment of a trajectory rather than a point as in the case of the DLS method.
3. The neural network method is more suitable for on-line computations due to fewer computations required.
4. The results in all cases of the LP-neuro method showed that this method

yielded more accurate results than the BP method.

5. The use of LP-neuro method results in faster convergence as compared to BP method in all cases.

Chapter 3
Neural Networks in Mechanism Design

3.1 Introduction

In the last chapter, neural network methods were used to solve velocity analysis problems of robotic manipulators near the points of singularities and of the nonlinear acceleration relationships of such manipulators in the Cartesian space. Furthermore, neural networks were also used to establish the relationships for the torque calculations which could be used on on-line basis.

In this chapter, neural network techniques are used in the design of mechanisms namely the function generators as well as rigid body guidance mechanism involving coupler curves.

3.2 Implementation of Neural Network in Mechanism Design

3.3.1 Nine-Point Path Problem

A four-bar mechanism with an added coupler P is shown in Fig. 3.1. L_0, L_1, L_2, L_3 and L_4 are the link lengths and θ_1 and α are the angles shown in Fig. 3.1. The

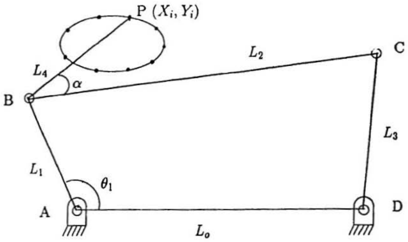


Figure 3.1 A Four-Bar Mechanism - Nine-Point Path Generation

objective is to find the necessary link parameters for the mechanism whose coupler point P passes through a given set of nine points. The constraint imposed here is that the link parameters should satisfy the Grashof's criterion i.e., the sum of the smallest and the longest link lengths cannot be greater than the sum of the remaining two link lengths if there is to be continuous relative rotation between two members.

The input consists of the co-ordinates of a set consisting of nine points and the output, the link parameters. The coupler point P goes through the nine points when θ_1 is varied in steps of 40° as shown in Fig. 3.1. The length of the fixed link L_0 was chosen arbitrarily.

The mechanism is obtained using neural networks by first training it by providing the input data set

$$\{I\} = \begin{Bmatrix} X_1 \\ Y_1 \\ X_2 \\ Y_2 \\ \vdots \\ X_9 \\ Y_9 \end{Bmatrix} \tag{3.1}$$

and the output

$$\{O\} = \begin{Bmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \\ \theta_1 \\ \alpha_1 \end{Bmatrix}^1 \tag{3.2}$$

The superscript 1 refers to the first training set and the input and output relationship is governed by the displacement equation of the mechanism

$$\vec{R}_{PA} = \vec{R}_{BA} + \vec{R}_{PB} \tag{3.3}$$

By providing different number of data sets, one obtains the convergent weight matrix [W].

The results obtained by both methods are shown in Tables 3.1 and 3.2. It is quite clear that the LP-neuro method yields better results but BP method also leads to good results.

TABLE 3.1: LINK PARAMETERS OF FOUR-BAR MECHANISM - NINE-POINT PATH GENERATION

LINK PARAMETERS	LP-NEURO METHOD	BP METHOD
L_0	106.00 mm	106.00 mm
L_1	21.01 mm	20.54 mm
L_2	104.28 mm	105.62 mm
L_3	110.79 mm	111.95 mm
L_4	43.04 mm	42.01 mm
θ_1	53.49°	54.00°
α	15.58°	15.62°

TABLE 3.2: COORDINATES OF THE NINE-POINT PATH PROBLEM - COMPARISON BETWEEN THE LP-NEURO METHOD AND BP METHOD

NO.	DESIRED		LP-NEURO METHOD		BP METHOD		LP-NEURO	BP
	X (mm)	Y (mm)	X (mm)	Y (mm)	X (mm)	Y (mm)	ERROR NORM	ERROR NORM
1	25.37	57.89	25.29	57.98	24.31	56.80	0.1204	1.5133
2	15.60	60.52	15.56	60.58	14.60	59.32	0.0721	1.5609
3	2.65	54.70	2.64	54.73	1.94	53.53	0.0316	1.3680
4	-6.70	43.00	-6.69	43.01	-7.09	41.98	0.0141	1.0927
5	-8.74	30.52	-8.73	30.53	-8.87	29.72	0.0141	0.8104
6	-3.45	22.79	-3.47	22.82	-3.47	22.20	0.0360	0.5903
7	6.51	23.69	6.44	23.72	6.44	23.19	0.0761	0.5048
8	17.70	33.59	17.59	33.64	17.38	32.98	0.1208	0.6888
9	25.88	47.53	25.77	47.61	25.12	46.65	0.1360	1.1627

3.3.2 Four-bar function generator

A typical four-bar function generator is shown in Fig. 3.2. The neural network method was used to design a four-bar function generator corresponding to

$$y = x^{1.5} \tag{3.4}$$

At first, the solutions were obtained for three precision points whose exact solutions were known (Wilson et al., 1983) and then for eight precision points. The precision points x_i were calculated by using Chebyshev spacing as follows:

$$x_i = x_o + \frac{\Delta x}{2}\left[1 - \cos(j\alpha - \frac{\alpha}{2})\right] \qquad j = 1,2,\ldots,n \tag{3.5}$$

where

$\Delta x = x_f - x_o$

n = number of precision points

α = (180/n) degrees.

The linear relationship between θ and x is given by

$$\theta_j = \theta_o + \frac{\Delta\theta}{\Delta x}(x - x_o) \tag{3.6}$$

Similarly, we also have

$$\phi_j = \phi_o + \frac{\Delta\phi}{\Delta y}(y - y_o) \tag{3.7}$$

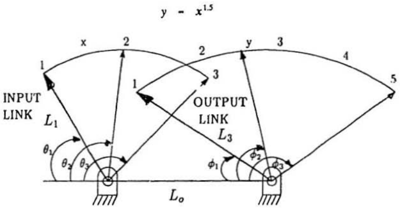


Figure 3.2 A Four-Bar Function Generator

The initial values x_o, θ_o, ϕ_o, as well as $\Delta\phi$, $\Delta\theta$, Δy and Δx are preselected. The objective was to find the necessary link parameters (L_o, L_1, L_2, L_3) which would yield accurate values at those precision points selected by using Eq.(3.5).

The network was trained using the input vector

$$\{I\} = \begin{Bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_n \\ y_n \end{Bmatrix}^1 \tag{3.8}$$

and the output

$$\{O\} = \begin{Bmatrix} L_o \\ L_1 \\ L_2 \\ L_3 \\ \theta_1 \\ \vdots \\ \theta_n \end{Bmatrix}^1 \tag{3.9}$$

The method consisted of the following steps:

1. Select three arbitrary values of x_j and a set of values for the link lengths.
2. Obtain θ_j corresponding to x_j using Eq.(3.6).

TABLE 3.3: LINK LENGTHS FOR THE FUNCTION-GENERATOR MECHANISM - THREE PRECISION POINTS

LINKS	EXACT SOLUTION*	LP-NEURO METHOD
L_0 **(mm)**	50.80	50.80
L_1 **(mm)**	264.16	263.83
L_2 **(mm)**	65.76	65.43
L_3 **(mm)**	255.52	255.78

* Refer Wilson et al. (1983)

TABLE 3.4: COMPARISON OF y VALUES (THEORETICAL AND LP-NEURO METHOD) - THREE PRECISION POINTS

x	y_{th} (THEO.)	y_{lp} (LP-NEURO)	% ERROR (LP-NEURO) $= (y_{th}-y_{lp})/y_d * 100$
1.00	1.000	0.999	0.100
2.00	2.828	2.828	0.000
3.00	5.196	5.195	0.019

TABLE 3.5: LINK LENGTHS FOR THE FUNCTION-GENERATOR MECHANISM - EIGHT PRECISION POINTS

LINKS	LP-NEURO METHOD
L_0 (mm)	50.80
L_1 (mm)	230.50
L_2 (mm)	56.642
L_3 (mm)	222.50

TABLE 3.6: COMPARISON OF y VALUES (THEORETICAL AND LP-NEURO METHOD) - EIGHT PRECISION POINTS

NO.	x	y_{th} (THEO.)	y_{lp} (LP-NEURO)	% ERROR (LP-NEURO) $= (y_{th}-y_{lp})/y_d$ *100
1	1.019	1.029	1.029	0.000
2	1.169	1.264	1.261	0.237
3	1.445	1.737	1.715	1.266
4	1.805	2.425	2.368	2.350
5	2.195	3.252	3.170	2.521
6	2.555	4.084	4.011	1.811
7	2.831	4.763	4.719	0.923
8	2.981	5.144	5.122	0.427

3. Solve for ϕ_i corresponding to θ_i using displacement analysis of a four bar mechanism.

4. Obtain y_i using Eq.(3.7).

The results obtained for the link lengths of four-bar function generator for the three and eight precision points are given in Tables 3.3 to 3.6. Tables 3.4 and 3.6 show the errors in y values in function generators. It is quite clear from these tables, that the neural network can be successfully used to design function generators. Only the LP-neuro method was used here because it yielded better results than the BP method earlier.

3.4 Conclusions

The present work deals with the use of new techniques in the solution of design problems of different mechanisms. The set of weights which establishes the linear relationship were obtained using two non-linear methods. Based on the work in this chapter the following conclusions can be drawn:

1. The LP-neuro method yields better results than the BP method.
2. One can successfully use the neural network technique to solve the mechanism design problems.

Chapter 4
Neural Network Control in Robotics

4.1 Introduction

Much effort has been devoted to develop efficient procedures for real-time computation of manipulator dynamic equations. Recursive algorithms like Newton-Euler algorithm are now being used to achieve substantial improvement in terms of computational efficiency. In inverse dynamic calculations, the joint accelerations are affected not only by the computed torques but also by the disturbances such as Coulomb and viscous friction and modelling errors. The dynamic equations of a robotic manipulator form a complex, non-linear multivariable system. Computations are done at each point along the trajectory, which in turn reduce the overall speed of the movement of the manipulator.

In this chapter, an effort has been made to improve the computational need, by providing the gain parameters required to control the desired trajectory of a simple planar two-link manipulator using neural networks. Learning is based on input parameters like positional parameters $\{\Theta\}$, error in position values $\{e\}$, error in joint velocity values $\{\dot{e}\}$ etc. A set of gain parameters namely position gain values k_p and velocity gain values k_v is identified using the LP-neuro method. Then the weights obtained are used in the on-line trajectory control.

4.2 Trajectory Control

An independent Proportional-Plus-Derivative (PD) Control scheme is used to control the movement of the manipulator. While PD schemes are adequate in most control applications, there is overshooting i.e. the end-effector could go beyond the specified position before actually settling down. Overshooting is quite undesirable, because in order to eliminate overshooting, an integrator is used which introduces damping and causes the end-effector to move slowly through a number of intermediate set points, thus considerably delaying the completion of the task, and the quality of the displacement etc. The controller design can thus become more sophisticated on account of the involvement of non-linear system dynamics.

4.2.1 Inverse Dynamics of a n-Link Manipulator

The dynamic equation of an n-link manipulator in matrix form is written as

$$\{\tau\} = [M(\Theta)]\{\ddot{\Theta}\} + \{V(\Theta,\dot{\Theta})\} + \{G(\Theta)\} \tag{4.1}$$

where

$[M(\Theta)]$ is the mass matrix

$\{V(\Theta,\dot{\Theta})\}$ is the vector containing centrifugal and coriolis terms

$\{G(\Theta)\}$ is the vector containing gravity terms

In the case of a simple planar two-link manipulator shown in Fig. 2.9, the matrices involved are:

$$[M(\Theta)] = \begin{bmatrix} l_2^2 m_2 + 2l_1 l_2 m_2 c_2 + l_1^2(m_1 + m_2) & l_2^2 m_2 + l_1 l_2 m_2 c_2 \\ l_2^2 m_2 + l_1 l_2 m_2 c_2 & l_2^2 m_2 \end{bmatrix}, \tag{4.2}$$

$$\{V(\Theta,\dot{\Theta})\} = \begin{Bmatrix} -m_2 l_1 l_2 s_2 \dot{\Theta}_2^2 - 2m_2 l_1 l_2 s_2 \dot{\Theta}_1 \dot{\Theta}_2 \\ m_2 l_1 l_2 s_2 \dot{\Theta}_1^2 \end{Bmatrix}, \text{ and} \tag{4.3}$$

$$\{G(\Theta)\} = \begin{Bmatrix} m_2 l_2 g c_{12} + (m_1 + m_2) l_1 g c_1 \\ m_2 l_2 g c_{12} \end{Bmatrix} \tag{4.4}$$

These equations are derived using the Newton-Euler algorithm (shown in Table 2.2) based on the following assumptions:

1. All mass exists as a point mass at the distal end of the link.
2. Inertial tensor written at the center of mass for each link is the zero matrix.
3. There are no forces acting on the end-effector.

The idea of inverse dynamics is to seek a non-linear feedback control law

$$\tau = f(\Theta,\dot{\Theta}), \tag{4.5}$$

which when substituted in Eq.(4.1), results in a linear closed loop system. It is quite difficult or impossible to find control parameters for general non-linear systems. Since

$[M(\Theta)]$ is invertible, we may solve for joint acceleration $\{\ddot{\Theta}\}$ of the manipulator as

$$\{\ddot{\Theta}\} = [M]^{-1} \left[\{\tau\} - \{V(\Theta,\dot{\Theta})\} - \{G(\Theta)\} \right] \tag{4.6}$$

To achieve control, a finite difference scheme where $\{\dot{\Theta}\}$ and $\{\Theta\}$ are expressed in terms of $\{\ddot{\Theta}\}$ mentioned above, is used. This is discussed later in Section 4.3.1.

4.3 Evaluation of Gain Parameters for Trajectory Control

In a PD control scheme, the torque equation is given by

$$\{\tau_c\} = [K_p]\{e\} + [K_v]\{\dot{e}\} \tag{4.7}$$

$$\text{where } \{e\} = \{\Theta_d(t+1)\} - \{\Theta_a(t)\} \text{ and}$$

$$\{\dot{e}\} = \{\dot{\Theta}_d(t+1)\} - \{\dot{\Theta}_a(t)\}$$

From Fig. 4.1, the following observations can be drawn:

1. The kinematic parameters with the subscript *d* represent the desired values on the trajectory. These values are computed based on Table 2.2. The subscript *a* refers to the actual values of the kinematic parameters obtained by solving the control equations which involve the finite difference scheme and the relevant equations are Eq.(4.10),(4.11) and (4.12) mentioned in Section 4.3.1.

2. It should be noted that the desired values using Table 2.2 can be computed off-line based on the trajectory planning. On the other hand, the actual values and the errors etc., have to be computed on-line. One should try to minimize the on-line computations

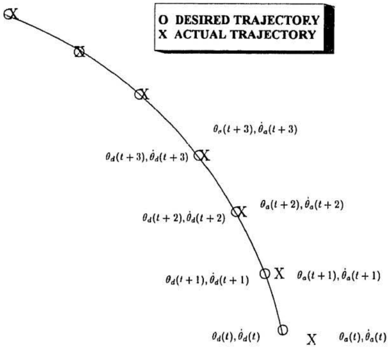


Figure 4.1 Specifications of the Desired and the Actual Trajectory

to increase the speed with which the task is performed.

3. $[K_p]$ and $[K_v]$ are diagonal matrices with diagonal elements consisting of position gain k_p and velocity gain k_v values respectively. Mathematically, they are written as

$$[K_p] = \begin{bmatrix} k_{p1} & 0 \\ 0 & k_{p2} \end{bmatrix} \quad and \quad [K_v] = \begin{bmatrix} k_{v1} & 0 \\ 0 & k_{v2} \end{bmatrix} \tag{4.8}$$

The use of a single value respectively for the entire trajectory for k_p and k_v may not be able to produce torques to follow the desired trajectory. The trajectory control can be achieved by evaluating the set of gain parameters for the entire trajectory using non-linear optimization method (the optimal control method) as described in the Section 4.3.1 on a point by point basis. This requires the gain values to be different for each point along the trajectory of the manipulator. The objective of the optimal control is to minimize the errors in joint positions and joint velocities between the actual values and the desired values, based on the gain variables $[K_p]$ and $[K_v]$ as discussed below.

4.3.1. Evaluation of Gain Values Using Non-Linear Optimization Method

A simple planar two-link manipulator having two revolute joints shown in Fig.2.9 was considered. The trajectory involved with associated velocity profile are shown in Figs. 4.2 and 4.3 respectively. In Fig.4.3, the end effector accelerates from point A to

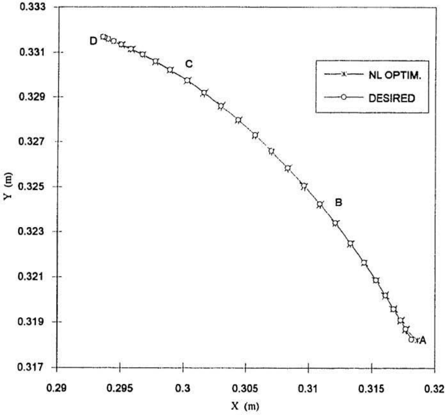


Figure 4.2 Desired Trajectory and the Trajectory Obtained Using Non-Linear Optimization Method

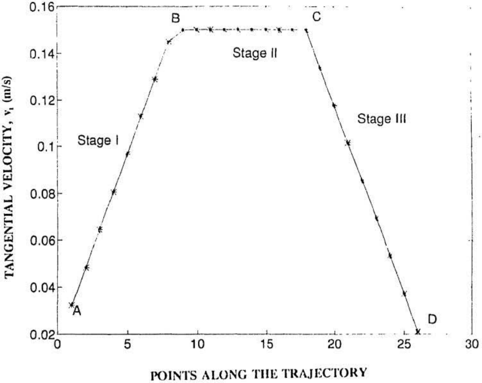


Figure 4.3 Desired Tangential Velocity Profile

Table 4.1 Various Parameters used for the Trajectory Control

	Link 1	Link 2
Link Lengths (m)	0.3	0.2
Mass (kg)	4.0	3.0
Radius of the Circle (m)	0.05	
Maximum Tangential Velocity v_t, (m/s)	0.15	
Initial Position Gain Values	$k_{p1} = 100.5$; $k_{p2} = 200.10$;	
Initial Velocity Gain Values	$k_{v1} = 50.6$; $k_{v2} = 80.8$;	
Step value in time, Δt (s)	0.01	
Initial Position (rad)	$\Theta_1 = 1.1469$; $\Theta_2 = -0.9228$;	

point B (Stage I) and then traverses along the trajectory at constant tangential velocity v_t with 0.15 m/s (Stage II) to C and then decelerates to zero speed at D (Stage III). The various parameters used in the trajectory control are shown in Table 4.1. The variation of $\Theta_1, \Theta_2, \dot{\Theta}_1$ and $\dot{\Theta}_2$ of the desired trajectory are shown in Figs. 4.4 and 4.5 respectively.

The various steps involved (shown in Fig.4.6) are:

1. Note the link parameters like lengths, mass etc., (refer to Table 4.1).

2. Calculate the coordinates of the desired circular trajectory using (shown in Fig.2.22 with the specifications $\Theta_1 = 45^\circ, \Theta_2 = 0^\circ$; r = 0.05 m) a single variable α. The transformation matrices used were:

$$\begin{Bmatrix} x \\ y \\ 0 \\ 1 \end{Bmatrix} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & l \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & l_1 \\ \sin(\theta_2) & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & (l_2\ r) \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{Bmatrix} \tag{4.9}$$

3. Calculate the joint parameters such as $\{\Theta_d(t)\}$ and the joint velocity vector $\{\dot{\Theta}_d(t)\}$ and joint acceleration $\{\ddot{\Theta}(t)\}$ for the desired trajectory using Eqs.(2.23) and (2.50) (corresponding to each point on the velocity profile). It can be done of[illegible]lso. Note the initial joint position $\{\Theta_a(t=0)\}$ and joint velocities $\{\dot{\Theta}_a(t=0)\}$ of the manipulator, and compute $\{e\}$ and $\{\dot{e}\}$ (shown in Figs. 4.7 and 4.8) mentioned in Eq.(4.7).

4. Compute the torque $\{\tau_c\}$ based on the error in joint position $\{e\}$ and joint velocity $\{\dot{e}\}$ using the Eq.(4.7). Substitute the torque in the dynamic equation, Eq.(4.6), and evaluate the joint acceleration $\{\ddot{\Theta}(t)\}$ using the expression:

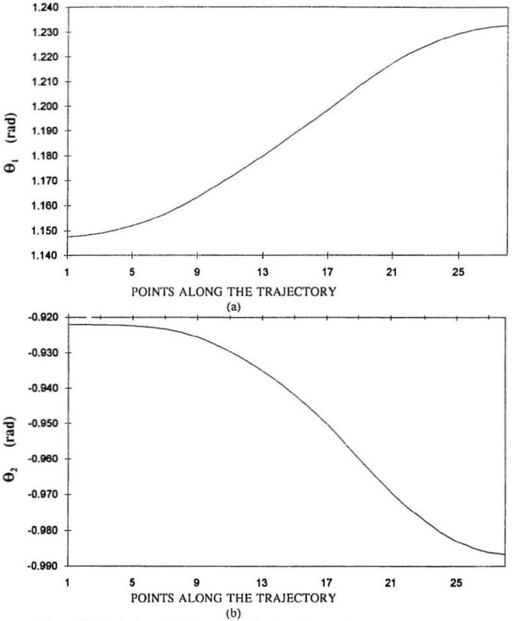
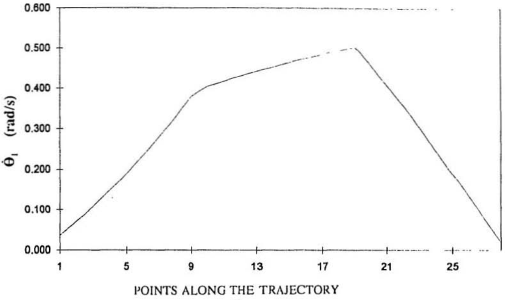
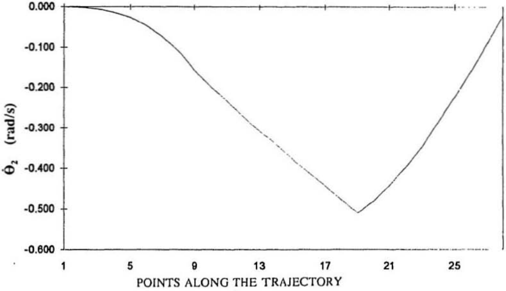


Figure 4.4 Variation of (a) Θ_1 and (b) Θ_2 Along the Desired Trajectory



(a)



(b)

Figure 4.5 Variation of (a) $\dot{\theta}_1$ and (b) $\dot{\theta}_2$ Along the Desired Trajectory

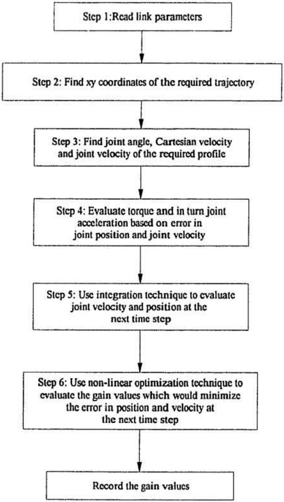
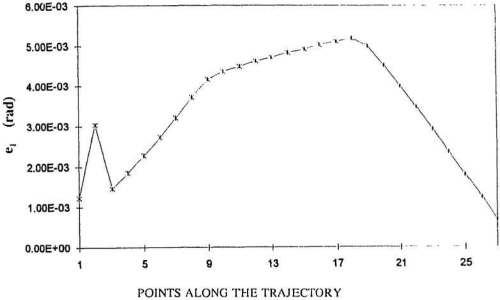
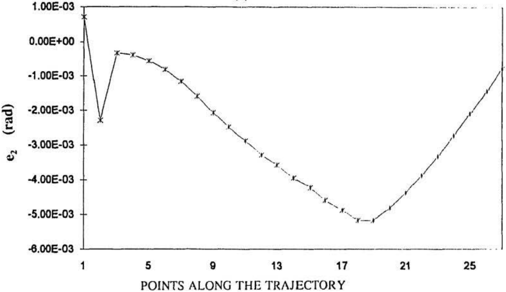


Figure 4.6 Flow Chart - Trajectory Control Using Non-Linear Optimization Method



(a)



(b)

Figure 4.7 Variation of (a) e_1 and (b) e_2 Along the Trajectory

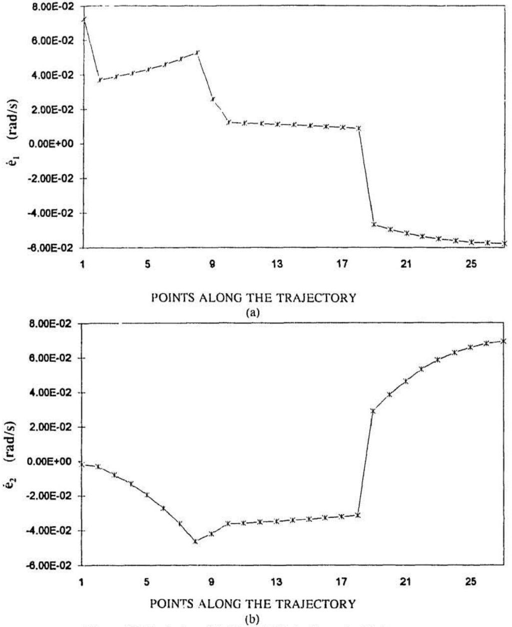


Figure 4.8 Variation of (a) $\dot{e}_1$ and (b) $\dot{e}_2$ Along the Trajectory

$$\{\ddot{\Theta}_a(t)\} = [M(\Theta_a(t))]^{-1} [\{\tau_c\} - \{V(\Theta_a(t),\dot{\Theta}_a(t))\} - \{G(\Theta_a(t))\}] \quad (4.10)$$

5. Having calculated the joint accelerations, numerically integrate forward in steps of time Δt and obtain $\{\Theta_a(t+1)\}$ and $\{\dot{\Theta}_a(t+1)\}$ using Newmark-β scheme. The equations involved are:

$$\{\dot{\Theta}_a(t+1)\} = \{\dot{\Theta}_a(t)\} + [(1 - \beta)\ \{\ddot{\Theta}_a(t)\} + \beta\ \{\ddot{\Theta}_a(t+1)\}]\ \Delta t \quad (4.11)$$

$$\{\Theta_a(t+1)\} = \{\Theta_a(t)\} + \Delta t\ \{\dot{\Theta}_a(t)\} + [(0.5 - \alpha)\ \{\ddot{\Theta}_a(t)\} + \alpha\ \{\ddot{\Theta}^*(t+1)\}]\ \Delta t^2 \quad (4.12)$$

where α and β are known constants ($\alpha = 0.5$ and $\beta = 0.001$ were used in the present problem). The calculation of $\ddot{\Theta}^*(t+1)$ involves the following steps:

(a) Write an equation similar to Eq.(4.10) where the subscript a is replaced by d at time $t+1$. This can be mathematically expressed as:

$$\{\ddot{\Theta}^*(t+1)\} = [M(\Theta_d(t+1))]^{-1} [\{\tau^*\} - \{V(\Theta_d(t+1),\dot{\Theta}_d(t+1))\} - \{G(\Theta_d(t+1))\}] \quad (4.13)$$

where

$$\{\tau^*\} = [K_p]\ \{\ \Theta_d(t+2) - \Theta_d(t+1)\ \} + [K_v]\ \{\ \dot{\Theta}_d(t+2) - \dot{\Theta}_d(t+1)\ \} \quad (4.14)$$

The computation of $\{\ddot{\Theta}^*(t+1)\}$ involves the quantities known at this step. There is no iteration required here.

6. The gain values are evaluated using non-linear optimization routine that would

minimize $\| \Theta_a(t+1) - \Theta_d(t+1) \|$ and $\| \dot{\Theta}_a(t+1) - \dot{\Theta}_d(t+1) \|$. Obtain $[K_p]$ and $[K_v]$ using the Hookes and Jeeves method (Rao, 1978) explained below:

Define

$$\{X\} = \begin{Bmatrix} k_{p1} \\ k_{p2} \\ k_{p3} \\ k_{p4} \end{Bmatrix}, \tag{4.15}$$

and

$$F(\{X\}) = \{\Theta_a(t+1) - \Theta_d(t+1)\}^2 + \{\dot{\Theta}_a(t+1) - \dot{\Theta}_d(t+1)\}^2 \tag{4.16}$$

The objective function can be mathematically written as

$$U(\{X\}) = F(\{X\}) + \sum_{k=1}^{m} P_k g_k^2 H(g_k) \tag{4.17}$$

where $\{X\}$ is the design vector and

k constraints are represented as g_k and

$H(g_k)$ is the Heavyside unit step function defined so that

$$H(g_k) = 1 \text{ for } g_k \geq 0 \text{ or,}$$

$$H(g_k) = 0 \text{ for } g_k < 0 \tag{4.18}$$

In Eq.(4.17), P_k are large penalty constants which are positive because the present problem is a minimization problem. Next, one needs to solve for the minimum of $U(\{X\})$ using the Hookes and Jeeves method and the step-by-step procedure for a design vector $\{X\}$ having n components is mentioned below:

(i) Start with an initial estimate of the design vector

$$\{X\} = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} \tag{4.19}$$

and choose Δx_i, $i = 1,2,\ldots,n$ as step lengths in each of the coordinate directions u_i, $i = 1,2,\ldots,n$.

(ii) Set temporary base point $\{Y_{k,0}\} = \{X_k\}$

(iii) Start the exploratory move by perturbing one design variable at a time in order to find the improved value of the objective function. Set:

$$\begin{aligned}
\{Y_{k,i}\} &= \{Y_{k,i-1}\} + \Delta x_i \{u_i\} \quad \text{if } U^{+} = U(\{Y_{k,i-1}\} + \Delta x_i \{u_i\}) \\
&\qquad\qquad < U = U(\{Y_{k,i-1}\}) \\
\{Y_{k,i}\} &= \{Y_{k,i-1}\} - \Delta x_i \{u_i\} \quad \text{if } U^{-} = U(\{Y_{k,i-1}\} - \Delta x_i \{u_i\}) \\
&\qquad\qquad < U = U(\{Y_{k,i-1}\}) \\
&\qquad\qquad < U^{+} = U(\{Y_{k,i-1}\} + \Delta x_i \{u_i\}) \\
\{Y_{k,i}\} &= \{Y_{k,i-1}\} \quad \text{if } U = U(\{Y_{k,i-1}\}) < \min(U^{+}, U^{-})
\end{aligned} \tag{4.20}$$

In this way, all the design variables x_n are perturbed and the improved position $\{Y_{k,n}\}$ found.

(iv) If the point $\{Y_{k,n}\}$ is not different from $\{X_k\}$, reduce the step lengths Δx_i; set $i = 1$ and go to step (iii). If $\{Y_{k,n}\}$ is different from $\{X_k\}$ obtain the new base point as

$$\{X_{k+1}\} = \{Y_{k,n}\} \tag{4.21}$$

(v) Find the pattern direction {S} using

$$\{S\} = \{X_{k+1}\} - \{X_k\} \tag{4.22}$$

Find the point $\{Y_{k+1,0}\}$ as

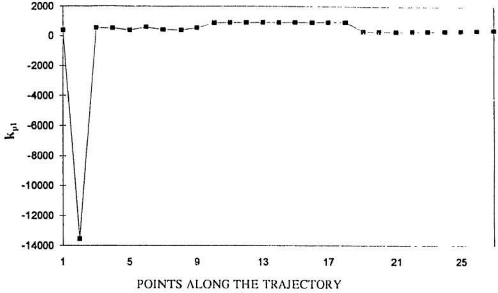
$$\{Y_{k+1,0}\} = \{X_{k+1}\} + \lambda\ \{S\} \tag{4.23}$$

Find λ^*, the optimum step length in the direction {S} and use λ^* in Eq.(4.23).

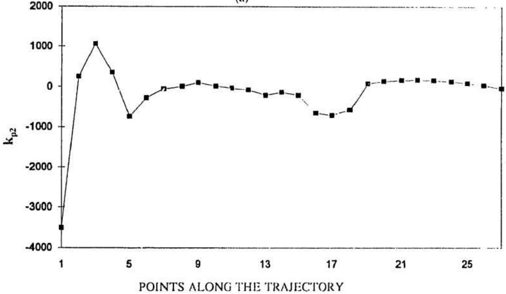
(vi) Set $k = k+1$, $U_k = U(\{Y_{k0}\})$ and $i = 1$; repeat step (iii). If at the end of step (iii), $U(\{Y_{k,n}\}) < U(X_k)$ use the new base point as $\{X_{k+1}\} = \{Y_{k,n}\}$ and go to step (v). If $U(\{Y_{k,n}\}) >= U(\{X_k\})$, set $\{X_{k+1}\} = \{X_k\}$ and reduce step lengths; set $k = k+1$ and go to step (ii)

(vii) The process is terminated if the step lengths become less than ϵ, a very small quantity.

These gain values when substituted in Eq.(4.7) would result in the desired torque. The steps 3 to 6 were repeated for the entire trajectory and a set of gain values was obtained. The objective here was to have the position control primarily. The results in Fig.4.2 show that this objective was realized very well. The gain values for this trajectory are shown in Figs. 4.9 and 4.10 and the gain values change quite significantly along the trajectory. This is because the matrices $[M(\Theta)]$, $\{V(\Theta,\dot{\Theta})\}$ etc., undergo continuous change from position to position which also includes the changes in the velocity vector.



(a)



(b)

Figure 4.9 Variation of (a) k_{p1} and (b) k_{p2} Along the Trajectory

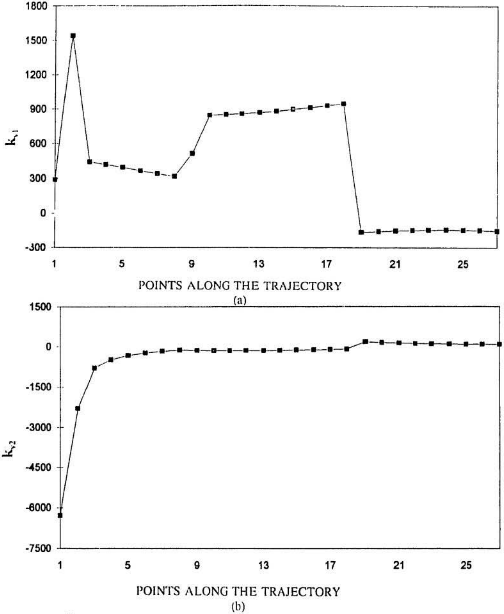


Figure 4.10 Variation of (a) k_{v1} and (b) k_{v2} Along the Trajectory

4.4 Neural Networks in Trajectory Control of Two-Link Manipulators

Neural network method has been widely used in many control applications. LP neuro method was found to be very effective in the mechanical design problems which was discussed in Chapter 3. It would be quite beneficial to have a weight matrix which relates the input vector

$$\{I\} = \{\Theta_1, \Theta_2, \dot{\Theta}_1, \dot{\Theta}_2, e_1, e_2, \dot{e}_1, \dot{e}_2\}^T \tag{4.24}$$

and the output vector

$$\{O\} = \{k_{p1}, k_{p2}, k_{v1}, k_{v2}\}^T \tag{4.25}$$

Several sets of {I} and {O} can be computed by starting with different initial conditions, but the same trajectory as shown in Figs. 4.2 and 4.3. Then the weight matrix [W] can be obtained from these sets of {I} and {O} in accordance with the descriptions in Chapter 2. The problem of trajectory control computationally becomes a lot simpler now with the known weights [W]. The steps involved in obtaining the weight matrix as shown in Fig.4.11 are as follows:

<u>Step 1:</u> Let the end-effector be at some initial position having coordinates (x,y).

<u>Step 2:</u> Use non-linear optimization method to evaluate gain values off-line for different trajectories.

<u>Step 3:</u> Note the input and output parameters for various trajectories.

<u>Step 4:</u> Compute [W] using the LP-neuro method.

<u>Step 5:</u> Use [W] on-line to evaluate the gain values (k_{p1}, k_{p2} etc ,) as shown in Figs. 4.12 and 4.13. Here, the weight matrix relates the input and output vectors

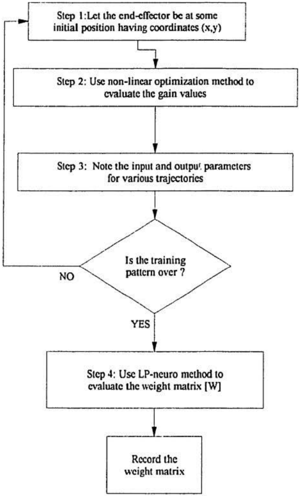
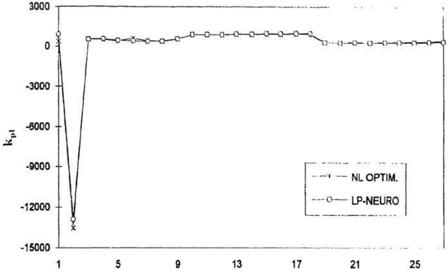
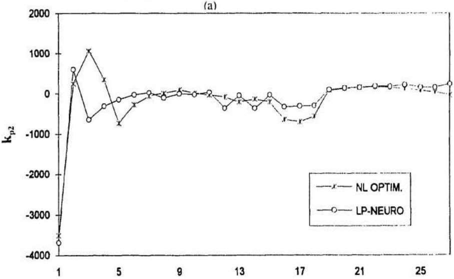


Figure 4.11 Flow Chart - Evaluation of Weight Matrix [W] for Trajectory Control Using LP-Neuro Method



POINTS ALONG THE TRAJECTORY
(a)



POINTS ALONG THE TRAJECTORY
(b)

Figure 4.12 Comparison of Gain Values (a) k_{p1} and (b) k_{p2} Obtained Using Non-Linear Optimization Method and LP-Neuro Method

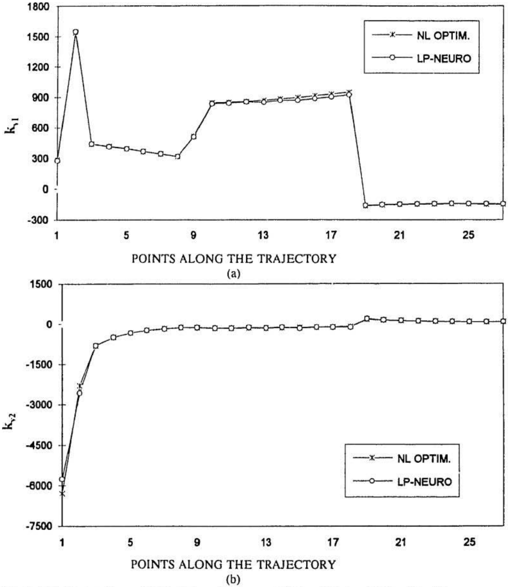


Figure 4.13 Comparison of Gain Values (a) k_{v1} and (b) k_{v2} Obtained Using Non-Linear Optimization Method and LP-Neuro Method

as

$$
\begin{Bmatrix} k_{p1} \\ k_{p2} \\ k_{v1} \\ k_{v2} \end{Bmatrix} = [W] \begin{Bmatrix} \Theta_1 \\ \Theta_2 \\ \dot{\Theta}_1 \\ \dot{\Theta}_2 \\ e_1 \\ e_2 \\ \dot{e}_1 \\ \dot{e}_2 \end{Bmatrix} \tag{4.26}
$$

The results corresponding to the steps mentioned above, carried out for the trajectory of a two-link manipulator, are shown in Fig. 4.14. These results clearly show the applicability of neural network method for on-line control of robotic manipulators.

4.5 Conclusions

Based on this work, the following conclusions can be drawn:

1. The non-linear control method yields accurate results.
2. LP-neuro method yields sufficiently accurate weight matrices for on-line control of robotic manipulators.

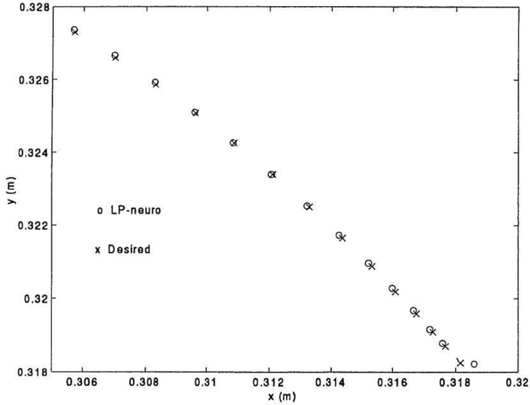


Figure 4.14 Desired Trajectory and the Trajectory Obtained Using LP-Neuro Method

Chapter 5
Conclusions

In this work, at first a new neural network learning algorithm called the LP-Neuro method for obtaining the weights was developed. The problems in the design of mechanisms were solved using this method and the backpropagation method. After this, the gain values were obtained by optimal control technique in the case of robotic manipulators. These gain values were then used to compute the weights for the on-line control problems.

5.1 Conclusions

Based on this work, the following conclusions can be drawn:

1. Neural network method can be used in the velocity analysis of robotic manipulators near singular configurations, by establishing a mathematical relationship between the angular velocity and Cartesian velocity vectors.

2. In the inverse velocity analysis near singular configurations, the neural network method yielded more accurate results than the DLS method.

3. The neural network method established relationship over a segment of a trajectory rather than a point as in the case of the DLS method.

4. A new algorithm called LP-neuro method was developed in which elements of the weight matrix were formulated as the unknown variables of the LP problem.

5. The use of LP-neuro method resulted in faster convergence as compared to BP method in all cases.

6. One can successfully use the neural network technique to solve the mechanism design problems.

7. The non-linear optimization method was found successful in evaluating the set of gain values for the manipulator to follow the desired trajectory.

8. LP-neuro method can be used to evaluate the weight matrix [W] that can be used in the on-line control of robotic manipulators.

5.2 Future Recommendations of the Work

Future research work can be pursued on the following topics:

1. LP-neuro method can be extended to many applications in the on-line control of robotic manipulators.

2. Efficient techniques like Karmarkar's algorithm can be applied in Linear Programming for faster convergence for problems involving large number of design variables.

3. It may be possible that the Optimal Control Method can be used in the on-line control of manipulators having more degrees of freedom if computations are done on parallel processors.

4. The neural network method can also be used to solve problems involving friction and other uncertainities in the trajectories of robotic manipulators.

REFERENCES

Akhras, R., and Angeles, J., 1990, "Unconstrained Nonlinear Least-Square Optimization of Planar Linkages for Rigid-Body Guidance", *Mechanisms and Machine Theory*, Vol.25, No.1, pp.97-118.

Akio, I., Takeshi, F., and Shigeru, O., 1992, "A Neural Network Compensator for Uncertainities of Robotic Manipulators", *IEEE Transactions on Industrial Electronics*, Vol.39, No.6, pp.565-569.

Angeles, J., Alizivatos, A., and Akhras, R., 1988, "An Unconstrained Non-Linear Method for Optimization of RRRR Planar Path Generation", *Mechanisms and Machine Theory*, Vol. 23, No.5, pp.343-353.

Asada, H., 1990, "Teaching and Learning of Compliance Using Neural Nets: Representation and Generation of Nonlinear Compliance", *Proc. IEEE Int. Conf. on Robotics and Automation"*, Vol.2, pp.1237-1244.

Baba, N., 1989, " A New Approach for Finding the Global Minimum of Error Function of Neural Networks", *Neural Networks*, Vol.2, pp.367-373.

Balasubramanian, R., and Sharan, A.M., 1993, "LP-Neuro Method - A New Approach for Solving Neural Network Problems", *Int. Simulation/WNN/ANN Conference*, San Francisco.

Barmann, F., and Biegler-Konig, F., 1992, "On a Class of Efficient Learning Algorithms for Neural Networks", *Neural Networks*, Vol.5, pp.139-142.

Chiaverni, S., 1992, "Inverse Differential Kinematics of Robotic Manipulators at Singular and Near-Singular Configurations", *IEEE International Conference on Robotics and Automation*, Tutorial M1, Nice, France, pp.2.1-2.9.

Craig, J.C., 1986, *Introduction to Robotics - Mechanics & Control*, Addison-Wesley Publishing Company, Massachusetts.

Deo, A.S., and Walker, I.D., 1992, "Robot Subtask Performance with Singularity Robustness using Optimal Damped Least-Squares", *Proc. of the 1992 IEEE International Conf. on Robotics and Automation*, Nice, France, Vol.1, pp.434-441.

Erdman, A.G., and Sandor, G.N., 1984, *Mechanism Design: Analysis and Synthesis*, Vols.1&2, Prentice-Hall, Englewood Cliffs, New Jersey.

Francisco, J.A., 1990, "Multilayer Back Propagation Network for Learning the Forward and Inverse Kinematics Equations", *Proc. Int. Joint Conf. on Neural Networks*, Washington,D.C., pp.II.319-II.321.

Freudenstein, F., 1955, "Approximate Synthesis of Four-Bar Linkages", *Transactions of the ASME*, Vol.77, pp.853-861.

Freudenstein, F., and Sandor, G.N., 1959, "Synthesis of Path-Generating Mechanism by Means of a Programmed Digital Computer", *ASME Journal of Engineering for Industry*, Series B, Vol.81, pp.159-168.

Fukuda, T., Shibata, T., Kosuge, K., 1991, "Neuromorphic Sensing and Control - Applications to Position, Force and Impact Control for Robotic Manipulators", *Proc. of the 30th Conf. on Decision and Control*, Brighton, England, pp.162-167.

Fukuda, T., and Shibata, T., 1992, "Theory and Applications of Neural Networks for Industrial Control Systems", *IEEE Transactions on Industrial Electronics*, Vol.39, No.6, pp.472-487.

Grossberg, G., 1982, *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition and Motor Control*, Boston, MA.

Gu, Y. and Chan, J.W.M., 1989, "On design of Non-linear Robotic Control System with Neural Networks", *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*, Cambridge, Massachusetts, pp.200-205.

Gulati, S., Barhen, J., and Iyengar, S.S., 1990, "Computational Neural Learning Formalisms for Manipulator Inverse Kinematics", *NASA Conference on Space Telerobotics*, Washington, Vol.1, pp.333-342.

Guo, J., and Cherkassky, V., 1989, "A Solution to the Inverse Kinematic Problem in Robotics using Neural Network Processing", *Proc. Int. Joint Conf. on Neural Networks*, Washington, D.C., pp.II.299-II.304.

Hecht-Nielsen, R., 1990, *Neurocomputing*, Addison-Wesley Publishing Co., Massachusetts.

Helferty, J.J., and Biswas, S., 1990, "Neuromorphic Control as a Self-Tuning Regulator", *Proc. 5th IEEE Int. Symp. on Intelligent Control*, Philadelphia, Pennsylvania, pp.506-511.

Hopfield, J.J., 1982, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proc. Nat. Acad. Sci.*, U.S.A, Vol.79, pp.2554-2558.

Jain, J., 1988, *Computer-Aided Kinematic and Dynamic Analyses of Flexible Spatial Manipulators of Arbitrary Architecture*, M.Eng. Thesis, Memorial University of Newfoundland, St.John's, Newfoundland, Canada.

Jamshidi, M., Horne, B., and Vadiee, N., 1990, "A Neural Network-Based Controller for a Two-Link Robot", *Proc. 29th Conference on Decision and Control*, Honolulu, Hawaii, pp.3256-3257.

Karakasoglu, A. and Sundareshan, M.K., 1990, "Decentralized Variable Structure Control of Robotic Manipulators: Neural Computational Algorithms", *Proc. 29th Conference on Decision and control*, Honolulu, Hawaii, pp.3258-3259.

Karayiannis, N.B., and Venetsanapoulos, A.N., 1992, "Fast Learning Algorithms for Neural Networks", *IEEE Transactions on circuits and Systems*, Vol.39, No.7, pp.453-473.

Kawato, M., 1990, "Computational Schemes and Neural Network Models for Formation and Control of Multi-joint Arm Trajectory", *Neural Networks for Control*, The MIT Press, London, pp.197-228.

Kohonen, T., 1988, "An Introduction to Neural Computing", *Neural Networks*, Vol.1, No.1, pp.3-16.

Kulkarni, A.D., 1991, "Solving Ill-Posed Problems with Artificial Neural Networks", *Neural Networks*, Vol.4, pp.477-484.

Lawson, C.L., and Hanson, R.J., 1974, *Solving Least Squares Problems*, Prentice Hall Inc., Englewood Cliffs, New Jersey.

Liu, Z., and Angeles, J., 1992, "Least-Square Optimization of Planar and Spherical Four-Bar Function Generator Under Mobility Constraints", *ASME J. of Mechanical Design*, Vol.114, pp.569-573.

Maciejewski, A.A., and Klein, C.A., 1989, " The Singular Value Decomposition: Computation and Applications to Robotics", *The International Journal of Robotics Research*, Vol.8, No.6, pp.63-79.

Mayorga, R.V., Wong, A.K.C., and Milano, N., 1992, "A Fast Least Squares Solution to Manipulator Inverse Kinematics and Singularities Prevention", *Proc. IEEE International Conference on Intelligent Robots and Systems*, Raleigh, NC.

McCormick, S.T., 1990, "Making Sparse Matrices Sparser: Computational Results", *Mathematical Programming*, Vol.49, pp.91-111.

Mohan Rao, A.V., Sandor, G.N., Kohli, D., and Soni, A.H., 1973, "Closed Form Synthesis of Spatial Function Generating Mechanisms for the Maximum Number of Precision Points", *ASME J. of Engineering for Industry*, Vol.95, pp.725-736.

Morgan, A.P., and Wampler, C.W., 1989, "Solving a Four-Bar Design Problem Using Continuation", *Advances in Design Automation - 1989; Mechanical Systems Analysis, Design and Simulation*, B.Ravani, ed.; ASME DE- Vol.19-3, pp.409-416.

Nakamura, Y., and Hanafusa, H., 1986, "Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control", *Journal of Dynamic Systems, Measurement and Control*, Vol.108, pp.163-171.

Nguyen, L., Patel, R.V., and Khorasani, K., 1990, "Neural Network Architectures for the Forward Kinematics Problem in Robotics", *Proc. Joint IEEE International Neural Network Conf.*, Sandiego, California , pp.393-399.

Rao, S.S., 1970, *Optimization Theory and Applications*, McGraw-Hill, New York.

Rumelhart, D.E., and McClelland, J.L., 1986, *Parallel Distributed Processing*, MIT Press, Massachusetts, Vols.1 and 2.

Sharan, A.M., and Balasubramanian, R., 1993, "Neural Network Method for Inverse Kinematics Near Singular Configurations", *OMAE '93 Conference*, Scotland.

Shigley, J.E., and Uicker, J.J., 1980, *Theory of Machines and Mechanisms*, McGraw-Hill, New York.

Siddal, J.N., 1982, *Optimal Engineering Design - Principles and Applications*, Marcel Dekker Inc., New York.

Simpson, P.I., 1990, *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementation*, Pergamon Press, New York.

Subbian, T., and Flugrad, D.R., 1989, "Four-Bar Path Generation Synthesis by a Continuation method", *Advances in Design Automation - 1989; Mechanical Systems Analysis, Design and Simulation*, B.Ravani, ed.; ASME DE- Vol.19-3, pp.425-432.

Suh, C.H., and Radcliffe, C.W., 1978, *Kinematics and Mechanisms Design*, Robert E. Krieger Publishing Co., Malabar, Florida.

Tabary, G., and Salaun, I., 1992, "Control of a Redundant Articulated System by Neural

Networks", *Neural Networks*, Vol.5, pp.305-311.

Tinubu, S.O., and Gupta, K.C., 1984, "Optimal Synthesis of Function Generators without the Branch Defect", *ASME J. of Mechanisms Transmissions, and Automation in Design*, Vol.106, pp.348-354.

Tsai, L.W., and Lu, J.J., 1989, "Coupler-Point-Curve Synthesis Using Homotopy Methods", *Advances in Design Automation - 1989; Mechanical Systems Analysis, Design and Simulation*, B.Ravani, ed.; ASME DE- Vol.19-3, pp.417-424.

Wampler, C.W., and Leifer, L.J., 1988, "Applications of Damped Least-Squares Methods to Resolved-Rate and Resolved-Acceleration Control of Manipulators", *Journal of Dynamic Systems, Measurement and Control*, Vol.110, pp.31-38.

Wampler, C.W., 1986, "Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Method", *IEEE Transactions on Systems, Man and Cybernetics*,
Vol.16, No.1, pp.93-101.

Wampler, C.W., Morgan, A.P., and Sommese, A.J., 1992, "Complete Solution of the Nine-Point Path Synthesis Problem for Four-Bar Linkages", *J. of Mechanical Design, Trans. ASME*, Vol.114, pp.153-159.

Wasserman, P.D., 1989, *Neural Computing Theory and Practice*, Van Nostrand Reinhold, Nework.

Wilde, D.J., 1982, "Error Linearization in the Least-Squares Design of Function Generating Mechanisms", *ASME J. of Mechanical Design*, Vol.104, pp.881-884.

Yamamura, A.A., Sideris, A., Ji, C., and Psaltis, D., 1990, "Neural Network Control of a Two-Link Manipulator", *Proc. 29th Conf. on Decision and Control*, Honolulu, Hawaii, pp.3303-3307.

Yuh, J., 1992, "On Neural Net Controllers for Robotic Manipulators", *Proc. 4th Int. Symp. on Robotics and Manufacturing*, pp.757-762.

Zurada, J.M., 1992, *Introduction to Artificial Neural Systems*, West Publishing Company, New York.

Appendix A: Program Listings

(All programs are written by the author)

A.1 Velocity Analysis near Singular Points

The Damped Least squares method, Psuedo-inverse method are implemented in the velocity analysis in this section. The input and output training vectors are obtained and the backpropagation algorithm is used to train the neural network. The analysis are carried out for a planar two-link manipulator (A.1.1) as well as PUMA-560 manipulator (A.1.2).

A.2 Acceleration Analysis of a Two-Link Manipulator

Acceleration analysis of a two-link manipulator are carried out and the neural network using backpropagation and LP-neuro method are used to train the network. The program **LP-Neuro** automatically forms coefficient matrix [A] as well as the objective function according to Eq.(2.42).

A.3 Torque Analysis of a Two-Link Manipulator

Torque analysis is carried out using LP-neuro method as well as backpropagation method.

A.4 Nine-point Path Problem

LP-neuro method is used to train the network to design the four-bar mechanism based on several input-output training pattern.

A.5 Design of Four-Bar Function Generator

LP-neuro algorithm is again used to train the network to design a four-bar function generator.

A.6 Trajectory control of a Two-Link Manipulator

Variable gain values are obtained using a non-linear optimization routine and LP-neuro method is used to predict gain values based on error history in position and joint velocity. The weight matrix is used on-line to identify gain values for the required trajectory.

In general **OPTIVAR** - optimization library routine in (Siddall, 1982) is widely used for linear programming and non-linear optimization methods.

A.1 VELOCITY ANALYSIS NEAR SINGULAR POINTNS

NOTE: FOLLOWING ARE THE LIST OF FORTRAN CODES USED: THESE CODES DO NOT BELONG TO A SINGLE FORTRAN PROGRAM.

A.1.1 Case 1: A Planar Two-link Manipulator

Step 1: Find the necessary cartesian coordinates, joint displacements, joint velocities of the desired trajectory.

Step 2: Prepare a database of input and output vector and train the neural network restraining the maximum joint velocity.

Step 3: Compare the joint velocity values obtained by the neural network method with the joint velocities obtained by psuedo-inverse method and SVD calculations.

```
C     ------------------------PROT2.FOR---------------------
C     TO FIND THE CARTESIAN COORDINATES OF THE
C     DESIRED TRAJECTORY
C     STRAIGHT LINE PART OF THE TRAJECTORY
C     TWO D.O.F - PLANAR TWO LINK MANIPULATOR
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DATAN2D,DCOSD,DSIND
      DIMENSION R(3,3),XP(3),X(3),DT(3),VDT(3)
      OPEN(1, FILE = 'PROT2.DAT' , STATUS = 'OLD')
      OPEN(2, FILE = 'XY.MAT', STATUS = 'NEW' )
      OPEN(3, FILE = 'VEL.MAT', STATUS = 'NEW' )
      OPEN(4, FILE = 'PHI.MAT', STATUS = 'NEW' )
      OPEN(5, FILE = 'SLOPE.MAT', STATUS = 'NEW')
      OPEN(6, FILE = 'PFDIST.MAT', STATUS = 'NEW')
      OPEN(7, FILE = 'DIST.MAT', STATUS ='NEW')
C     ************************************************************
      READ(1,*)VEL
      READ(1,*)RADIUS
      READ(1,*)ORANG
      READ(1,*)ITER
      READ(1,*)STAR2
      READ(1,*)STEP2
      XO = 0.530D0 * DCOSD(ORANG)
      YO = 0.530D0 * DSIND(ORANG)
C     ************************************************************
      DO 110 NO = 1,ITER
```

```
      R(1,1) = DCOSD(ORANG)
      R(1,2) = -DSIND(ORANG)
      R(1,3) = XO
      R(2,1) = DSIND(ORANG)
      R(2,2) = DCOSD(ORANG)
      R(2,3) = YO
      R(3,1) = 0.0D0
      R(3,2) = 0.0D0
      R(3,3) = 1.0D0
      XP(1) = (RADIUS - STAR2)
      XP(2) = 0.0D0
      XP(3) = 1.0D0
      DIST = DIST + STEP2
      WRITE(7,*)DIST
      WRITE(4,*)STAR2
      CALL RMATVEC(R,XP,X)
      WRITE(2,*)X(1),X(2)
C     ***********************************************************
      DT(1) = VEL
      DT(2) = 0.0D0
      DT(3) = 0.0D0
      CALL RMATVEC(R,DT,VDT)
      WRITE(3,*)VDT(1),VDT(2)
      DO 120 I = 1,3
      XP(I) = 0.0D0
      X(I) = 0.0D0
      DO 120 J = 1,3
      R(I,J) = 0.0D0
120   CONTINUE
      STAR2 = STAR2 + STEP2
110   CONTINUE
      WRITE(6,*)DIST
      STOP
      END

      SUBROUTINE RMATVEC(A,B,C)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(3,3),B(3),C(3)
      DO 30 I = 1,3
      C(I) = 0.0D0
      DO 40 J = 1,3
      C(I) = C(I) + A(I,J) * B(J)
40    CONTINUE
30    CONTINUE
```

```
      RETURN
      END

C     --------------------------PROT1.FOR-----------------------
C     TO FIND THE CARTESIAN COORDINATES OF THE
C     DESIRED TRAJECTORY
C     ARC PART OF THE PROJECTORY
C     TWO D.O.F - PLANAR TWO LINK MANIPULATOR
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DATAN2D,DCOSD,DSIND
      DIMENSION R(3,3),XP(3),X(3),DT(3),VDT(3)
      OPEN(1, FILE = 'PROT1.DAT' , STATUS = 'OLD')
      OPEN(2, FILE = 'XY.MAT', STATUS = 'NEW' )
      OPEN(3, FILE = 'VEL.MAT', STATUS = 'NEW' )
      OPEN(4, FILE = 'PHI.MAT', STATUS = 'NEW' )
      OPEN(5, FILE = 'SLOPE.MAT', STATUS = 'NEW')
      OPEN(6, FILE = 'PFDIST.MAT', STATUS = 'OLD')
      OPEN(7, FILE = 'DIST.MAT', STATUS = 'NEW')
C     ************************************************************
      PI = 3.141592654D0
      READ(1,*)VEL
      READ(1,*)RADIUS
      READ(1,*)ORANG
      READ(1,*)ITER
      READ(1,*)STAR1
      READ(1,*)STEP1
      READ(6,*)PFD
      XO = 0.530D0 * DCOSD(ORANG)
      YO = 0.530D0 * DSIND(ORANG)
C     ************************************************************
      DO 10 NO = 1,ITER
      R(1,1) = DCOSD(ORANG)
      R(1,2) = -DSIND(ORANG)
      R(1,3) = XO
      R(2,1) = DSIND(ORANG)
      R(2,2) = DCOSD(ORANG)
      R(2,3) = YO
      R(3,1) = 0.0D0
      R(3,2) = 0.0D0
      R(3,3) = 1.0D0
      XP(1) = RADIUS * DCOSD(STAR1)
      XP(2) = RADIUS * DSIND(STAR1)
      XP(3) = 1.0D0
```

```
      DIST = (PI/180.0D0)*(STEP1)*RADIUS*REAL(NO) + PFD
      WRITE(4,*)STAR1
      WRITE(7,*)DIST
      CALL RMATVEC(R,XP,X)
      WRITE(2,*)X(1),X(2)
      PR1 = XP(2)
      PR2 = -XP(1)
      ANG = DATAN2D(PR2,PR1)
C     ***********************************************************
      WRITE(5,*)ANG
      DT(1) = VEL * DCOSD(ANG)
      DT(2) = VEL * DSIND(ANG)
      DT(3) = 0.0D0
      CALL RMATVEC(R,DT,VDT)
      WRITE(3,*)VDT(1),VDT(2)
      DO 20 I = 1,3
      XP(I) = 0.0D0
      X(I) = 0.0D0
      DO 20 J = 1,3
      P(I,J) = 0.0D0
20    CONTINUE
      STAR1 = STAR1 + STEP1
10    CONTINUE
C     ****************************************
      STOP
      END

C     ------------------ANGFIND.FOR---------------------
C     TO FIND THE JOINT DISPLACEMENTS
C     FOR A TWO DOF PLANAR MANIPULATOR
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DATAN2D
      OPEN(1, FILE = 'ANGFIND.DAT', STATUS = 'OLD')
      OPEN(2, FILE = 'XY.MAT' , STATUS = 'OLD')
      OPEN(3, FILE = 'THETA.MAT', STATUS = 'NEW')
      OPEN(4, FILE = 'ANOTHETA.MAT',STATUS = 'NEW')
C     ****************************************************
      READ(1,*)RL1,RL2
      READ(1,*)ITER
      DO 10 NO = 1,ITER
      READ(2,*)XO,YO
      T = 0.0D0
      RJ = 2. * YO * RL1
      RI = 2. * XO * RL1
```

```
      RK = YO*YO + XO*XO + RL1*RL1 - RL2*RL2
      T = (RJ*RJ) + (RI * RI) - (RK*RK)
      ALPHA1 = DATAN2D(RJ,RI)+DATAN2D(DSQRT(T),RK)
      ALPHA2 = DATAN2D(RJ,RI)+DATAN2D(-DSQRT(T),RK)
      BETA1 = DATAN2D((YO-RL1*DSIND(ALPHA1)),
     % (XO-RL1*DCOSD(ALPHA1)))-ALPHA1
      BETA2 = DATAN2D((YO-RL1*DSIND(ALPHA2)),
     % (XO-RL1*DCOSD(ALPHA2)))-ALPHA2
      WRITE(3,*) ALPHA1,BETA1
      WRITE(4,*)ALPHA1,BETA1
      WRITE(4,*)ALPHA2,BETA2
      WRITE(4,*)
10    CONTINUE
      STOP
      END

C     ----------------------TDTFIND.FOR------------------------
C     TO FIND JOINT VELOCITY
C     INVERSE KINEMATICS FOR TWO D.O.F PLANAR MANIPULATOR
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DATAN2D,DCOSD,DSIND
      DIMENSION VEL(2),THETA(2),RJ(2,2),RJINV(2,2),TDOT(2)
      OPEN(1, FILE = 'TDTFIND.DAT', STATUS = 'OLD')
      OPEN(2, FILE = 'THETA.MAT', STATUS = 'OLD')
      OPEN(3, FILE = 'VEL.MAT', STATUS = 'OLD')
      OPEN(4, FILE = 'DETER.MAT', STATUS = 'NEW')
      OPEN(5, FILE = 'THETADT.MAT', STATUS = 'NEW')
      OPEN(6, FILE = 'JACOBIAN.MAT', STATUS = 'NEW')
C     ****************************************************
      READ(1,*)RL1,RL2
      READ(1,*)ITER
      DO 10 NO = 1,ITER
      READ(2,*)THETA(1),THETA(2)
      READ(3,*)VEL(1),VEL(2)
      RJ(1,1) = -RL1*DSIND(THETA(1)) - RL2*DSIND(THETA(1) +
     * THETA(2))
      RJ(1,2) = -RL2*DSIND(THETA(1) + THETA(2))
      RJ(2,1) = RL1*DCOSD(THETA(1)) + RL2*DCOSD(THETA(1) +
     * THETA(2))
      RJ(2,2) = RL2*DCOSD(THETA(1) + THETA(2))
      DO 15 II = 1,2
      WRITE(6,*)(RJ(II,JJ),JJ=1,2)
15    CONTINUE
      DETER = RL1*RL2*DSIND(THETA(2))
```

```
      WRITE(4,*)DETER
      RJINV(1,1) = RJ(2,2)/DETER
      RJINV(1,2) = -RJ(1,2)/DETER
      RJINV(2,1) = -RJ(2,1)/DETER
      RJINV(2,2) = RJ(1,1)/DETER
      CALL RMATVEC(RJINV,VEL,TDOT)
      WRITE(5,*)TDOT(1),TDOT(2)
      DO 20 I = 1,2
      VEL(I) = 0.0D0
      THETA(I) = 0.0D0
      TDOT(I) = 0.0D0
      DO 30 J = 1,2
      RJ(I,J) = 0.0D0
      RJINV(I,J) = 0.0D0
30    CONTINUE
20    CONTINUE
10    CONTINUE
      STOP
      END

C     ---------------------MAX.FOR-----------------------
C     TO FIND THE MAXIMUM AND MINIMUM OF JOINT VELOCITIES
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION TDOT1(100),TDOT2(100)
      OPEN(1, FILE= 'THETADT.MAT', STATUS='OLD')
      OPEN(2, FILE= 'SDOT.MAT', STATUS='NEW')
      READ(*,*)NO
      DO 10 I = 1, NO
      READ(1,*)TDOT1(I),TDOT2(I)
10    CONTINUE
      CALL MINMAX(TDOT1,SMALL1,PLARGE1,NO)
      CALL MINMAX(TDOT2,SMALL2,PLARGE2,NO)
      DO 20 I = 1,NO
      SDOT1 = (TDOT1(I) - SMALL1)/(PLARGE1 - SMALL1)
      SDOT2 = (TDOT2(I) - SMALL2)/(PLARGE2 - SMALL2)
      WRITE(2,*)SDOT1,SDOT2
20    CONTINUE
      STOP
      END


      SUBROUTINE MINMAX(A,B,C,NO)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(100)
```

```
C     INITIALIZE THE SMALLEST AND LARGEST AS THE FIRST ENTRY
      SMALL = A(1)
      PLARGE = A(1)
C     SEARCH THE REST OF THE ARRAY FOR BETTER VALUES
      DO 100 I = 2,NO
        IF(A(I).LT.SMALL) THEN
            SMALL = A(I)
        ELSE
            IF (A(I).GT.PLARGE) THEN
              PLARGE = A(I)
            ENDIF
         ENDIF
100   CONTINUE
      B = SMALL
      C = PLARGE
      RETURN
      END

C     -------------------NURAL.FOR----------------------
C     A NEURAL NET  PROGRAM (BACK PROPOGATION ALGORITHM)
C     ***************************************************
C     VELOCITY ANALYSIS OF TWO-LINK MANIPULATOR
C     ***************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DEXP
      DIMENSION V(20,20),W(20,20)
      DIMENSION XDOT(100),YDOT(100),ZDOT(100)
      DIMENSION TDOT1(100),TDOT2(100),TDOT3(100)
      DIMENSION DTDOT1(100),DTDOT2(100),DTDOT3(100)
      DIMENSION RI(100),RM(100),RO(100)
      DIMENSION
    % FRM(100),FRO(100),FTDOT1(100),FTDOT2(100),FTDOT3(100)
      DIMENSION FRMD(100),FROD(100)
      DIMENSION EOS(100),EMS(100)
      OPEN(1, FILE='NURAL.DAT', STATUS='OLD')
      OPEN(3, FILE='VEL.MAT', STATUS='OLD')
      OPEN(4, FILE='ALPHA.MAT', STATUS='NEW')
      OPEN(5, FILE='BETA.MAT', STATUS='NEW')
C     ***************************************************
C     ENTER NO. OF INPUT,MIDDLE AND OUTPUT LAYER NEURONS
      READ(1,*)INNO
      READ(1,*)MIDNO
      READ(1,*)NOUTNO
C     ENTER NO. OF INPUT TRAINING DATAS
```

```
      READ(1,*)NODATA
C     ENTER NO. OF ITERATIONS
      READ(1,*)ITER
C     ENTER THE LEARNING RATE
      READ(1,*)ETA
C     ****************************************************
C     READ THE TRAINING INPUT VALUES & THEIR CORRESPONDING
C     OUTPUT VALUES
      DO 10 I = 1,NODATA
      READ(2,*)TDOT1(I),TDOT2(I)
      READ(3,*)XDOT(I),YDOT(I)
      FTDOT1(I) = (1.0D0/(1.0D0 + DEXP(-TDOT1(I))))
      FTDOT2(I) = (1.0D0/(1.0D0 + DEXP(-TDOT2(I))))
10    CONTINUE
C     ****************************************************
C     RANDOM NUMBER GENERATION FOR INITIAL INPUT AND
C     OUTPUT WEIGHT MATRICES
C     ****************************************************
C     SEED FOR RANDOM NUMBER GENERATOR
      ISEED = 23148
      JSEED = 32124
      DO 20 I = 1,MIDNO
      DO 20 J = 1,INNO
      RV = RAN(ISEED)*2.0D0 - 1.0D0
      V(I,J) = RV
20    CONTINUE
      DO 30 I = 1,NOUTNO
      DO 30 J = 1,MIDNO
      RW = RAN(JSEED)*2.0D0 - 1.0D0
30    CONTINUE
C     *****************************************************
C     TRAINING STARTS HERE
C     ****************************************************
      DO 40 NO = 1,ITER
C     **************************************************
      DO 50 KD = 1,NODATA
      RI(1) = XDOT(KD)
      RI(2) = YDOT(KD)
      DO 60 KR = 1,MIDNO
      RM(KR) = 0.0D0
      DO 70 KP = 1,INNO
      RM(KR) = RM(KR) + V(KR,KP)*RI(KP)
70    CONTINUE
      FRM(KR) = (1.0D0/(1.0D0 + DEXP(-RM(KR))))
```

```
      FRMD(KR) = FRM(KR)*(1.0D0 - FRM(KR))
60    CONTINUE
      DO 80 KR = 1,NOUTNO
      RO(KR) = 0.0D0
      DO 90 KP = 1,MIDNO
      RO(KR) = RO(KR) + W(KR,KP)*FRM(KP)
90    CONTINUE
      FRO(KR) = (1.0D0/(1.0D0 + DEXP(-RO(KR))))
      FROD(KR) = FRO(KR)*(1.0D0 - FRO(KR))
80    CONTINUE
C     ****************************************************
C     ERROR UPDATE
C     ****************************************************
      DO 100 J = 1,MIDNO
      EMS(J) = 0.0D0
100   CONTINUE
      EOS(1) = FTDOT1(KD) - FRO(1)
      EOS(2) = FTDOT2(KD) - FRO(2)
      DO 110 IE = 1,NOUTNO
      DO 120 JE = 1,MIDNO
      EMS(JE) = EMS(JE) + EOS(IE)*FROD(IE)*W(IE,JE)
120   CONTINUE
110   CONTINUE
C     ********************************************************
C     WEIGHT UPDATE
C     ********************************************************
      DO 200 IRT = 1,NOUTNO
      DO 210 JRT = 1,MIDNO
      W(IRT,JRT) = W(IRT,JRT) + ETA*EOS(IRT)*FROD(IRT)*FRM(JRT)
210   CONTINUE
200   CONTINUE
      DO 220 IRT = 1,MIDNO
      DO 230 JRT = 1,INNO
      V(IRT,JRT) = V(IRT,JRT) + ETA*EMS(IRT)*FRMD(IRT)*RI(JRT)
230   CONTINUE
220   CONTINUE
C     ****************************************************
      DO 300 I = 1,MIDNO
      RM(I) = 0.0D0
      DO 310 J = 1,INNO
      RM(I) = RM(I) + V(I,J)*RI(J)
310   CONTINUE
      FRM(I) = (1.0D0 /(1.0D0 + DEXP(-RM(I))))
300   CONTINUE
```

```
      DO 320 I = 1,NOUTNO
      RO(I) = 0.0D0
      DO 330 J = 1,MIDNO
      RO(I) = RO(I) + W(I,J)*FRM(J)
330   CONTINUE
320   CONTINUE
C     *********************************************************
C     FINAL OUTPUT AT THE END OF EACH ITERATION
C     ******************************************************
      DTDOT1(KD) = RO(1)
      DTDOT2(KD) = RO(2)
50    CONTINUE
40    CONTINUE
C     ****************************************************
C     COMPARISON OF NEURAL AND ACTUAL OUTPUT
C     ***************************************************
      DO 500 IT = 1,NODATA
      WRITE(4,*)DTDOT1(IT),TDOT1(IT)
      WRITE(5,*)DTDOT2(IT),TDOT2(IT)
500   CONTINUE
      STOP
      END

c     -------------------REBAK.FOR----------------------
C     TO FIND THE MAGNITUDE OF THE ERROR
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION TDOT1(100),TDOT2(100)
      DIMENSION DTDOT1(100),DTDOT2(100)
      OPEN(1, FILE= 'THETADT.MAT', STATUS='OLD')
      OPEN(3, FILE= 'ALPHA.MAT', STATUS='OLD')
      OPEN(4, FILE= 'BETA.MAT', STATUS='OLD')
      OPEN(6, FILE= 'REBAK.MAT', STATUS='NEW')
      OPEN(7, FILE= 'RERR.MAT', STATUS='NEW')
      READ(*,*)NO
      DO 10 I = 1, NO
      READ(1,*)TDOT1(I),TDOT2(I)
      READ(3,*)DTDOT1(I)
      READ(4,*)DTDOT2(I)
10    CONTINUE
      CALL MINMAX(TDOT1,SMALL1,PLARGE1,NO)
      CALL MINMAX(TDOT2,SMALL2,PLARGE2,NO)
      DO 20 I = 1,NO
      SDOT1 = DTDOT1(I)*(PLARGE1 - SMALL1) + SMALL1
      SDOT2 = DTDOT2(I)*(PLARGE2 - SMALL2) + SMALL2
```

```
      WRITE(6,*)SDOT1,SDOT2
C     TO FIND OUT THE MAGNITUDE OF ERROR
      UPP = TDOT1(I) - SDOT1
      VPP = TDOT2(I) - SDOT2
      ERR = SQRT(UPP*UPP + VPP*VPP )
      WRITE(7,*)ERR
20    CONTINUE
      STOP
      END

C     ------------------PSUD.FOR------------------------
C     PSEUDO-INVERSE SOLUTION OF A TWO-LINK MANIPULATOR
C     NEAR SINGULARITIES
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER (NRA=2,NCA=2,LDA=NRA,LDGINV=NCA)
      DIMENSION A(LDA,NCA),AINV(LDGINV,NRA)
      DIMENSION V(2),PST(2)
      OPEN(1, FILE = 'JACOBIAN.MAT', STATUS = 'OLD')
      OPEN(2, FILE = 'VEL.MAT', STATUS = 'OLD')
      OPEN(3, FILE = 'PSTETA.MAT', STATUS = 'NEW')
      WRITE(*,*)'NO OF DATAS ?'
      READ(*,*)ITER
      DO 10 NO = 1,ITER
      DO 20 I = 1,NRA
      READ(1,*)(A(I,J),J=1,NCA)
20    CONTINUE
      READ(2,*)V(1),V(2)
      TOL = 10.0 * AMACH(4)
C     ****************************************************
C     CALLING IMSL ROUTINE FOR PSUEDO-INVERSE
C     ****************************************************
      CALL DLSGRR(NRA,NCA,A,LDA,TOL,IRANK,AINV,LDGINV)
      CALL RMATVEC(AINV,V,PST)
      WRITE(3,*)PST(1),PST(2)
10    CONTINUE
      STOP
      END

C     --------------------IYO.FOR---------------------
C     COMPARISON OF THE VALUES AND PRINTING THE RESULTS
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DSQRT
      DIMENSION ALPHA(2),BETA(2),GAMMA(2)
      OPEN(1, FILE = 'THETADT.MAT', STATUS = 'OLD')
```

```
      OPEN(2, FILE = 'REBAK.MAT', STATUS = 'OLD')
      OPEN(5, FILE = 'PSTETA.MAT',STATUS = 'OLD')
      OPEN(3, FILE = 'SNORM.MAT', STATUS = 'NEW')
      OPEN(4, FILE = 'PHI.MAT', STATUS = 'OLD')
      WRITE(*,*)'ITERATIONS ? '
      READ(*,*)ITER
      DO 10 NO = 1,ITER
      READ(1,*)ALPHA(1),BETA(1)
      READ(2,*)ALPHA(2),BETA(2)
      READ(5,*)GAMMA(1),GAMMA(2)
      READ(4,*)PHI
      TSQP = DSQRT(ALPHA(1)*ALPHA(1) + BETA(1)*BETA(1))
      SQP = DSQRT(ALPHA(2)*ALPHA(2)+BETA(2)*BETA(2))
      PERT = DSQRT(GAMMA(1)*GAMMA(1)+GAMMA(2)*GAMMA(2))
      WRITE(3,*)TSQP,SQP,PERT
10    CONTINUE
      STOP
      END
```

A.1.2 Case 2: Puma-560 Manipulator

Step 1: Find the necessary cartesian coordinates, joint displacements, joint velocities of the desired trajectory.

Step 2: Prepare a database of input and output vector and train the neural network restraining the maximum joint velocity.

Step 3: Compare the joint velocity values obtained by the neural network method with the joint velocities obtained by psuedo-inverse method and SVD calculations.

```
C     **************************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      OPEN(UNIT=1,FILE='RANG.MAT',STATUS='NEW')
      WRITE(*,*)'NO. OF DATAS'
      READ(*,*)NO
      WRITE(*,*)'INITIAL THETA3 ?'
      READ(*,*)T3
      WRITE(*,*)'INCREMENTAL THETA3 ?'
      READ(*,*)DELT
      T1 = 0.0D0
      T2 = -1.90843D0
```

```
      DO 10 I = 1,NO
      T3 = T3 + DELT
      WRITE(1,*)T1,T2,T3
10    CONTINUE
      STOP
      END

C     ******************************************************
C     TO GENERATE CARTESIAN COORDINATES
C     FOR THE CIRCULAR TRAJECTORY
C     ******************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DSIND,DCOSD
      EXTERNAL DMURRV
      DIMENSION TRA(4,4),P3(4),XYZ(4)
      OPEN(UNIT=1,FILE='RANG.MAT',STATUS='OLD')
      OPEN(UNIT=2,FILE='PUMA.DAT',STATUS='OLD')
      OPEN(UNIT=3,FILE='COORD.MAT',STATUS='NEW')
      READ(2,*)A2,A3,D3,D4
      WRITE(*,*)'NO OF DATAS ? '
      READ(*,*)ITER
      DO 10 I = 1,ITER
      READ(1,*)T1,T2,T3
      C1 = DCOSD(T1)
      S1 = DSIND(T1)
      C2 = DCOSD(T2)
      S2 = DSIND(T2)
      C23 = DCOSD(T2+T3)
      S23 = DSIND(T2+T3)
      TRA(1,1) = C1*C23
      TRA(1,2) = -C1*S23
      TRA(1,3) = -S1
      TRA(1,4) = C1*A2*C2 - S1*D3
      TRA(2,1) = S1*C23
      TRA(2,2) = -S1*S23
      TRA(2,3) = C1
      TRA(2,4) = S1*A2*C2 + C1*D3
      TRA(3,1) = -S23
      TRA(3,2) = -C23
      TRA(3,3) = 0.0D0
      TRA(3,4) = -A2*S2
      TRA(4,1) = 0.0D0
      TRA(4,2) = 0.0D0
      TRA(4,3) = 0.0D0
```

```
      TRA(4,4) = 1.0D0
      P3(1) = A3
      P3(2) = D4
      P3(3) = 0.0D0
      P3(4) = 1.0D0
      CALL DMURRV(4,4,TRA,4,4,P3,1,4,XYZ)
      WRITE(3,*)XYZ(1),XYZ(2),XYZ(3)
10    CONTINUE
      STOP
      END

C     ********************************************************
C     A PROGRAM FOR GENERATING THE CARTESIAN VELOCITIES
C     FOR A PUMA-560 MANIPULATOR
C     ********************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DATAN2D,DSIND,DCOSD
      OPEN(UNIT=1,FILE='PUMA.DAT',STATUS='OLD')
      OPEN(UNIT=3,FILE='PUMAVEL.MAT',STATUS='NEW')
      OPEN(UNIT=4,FILE='COORD.MAT',STATUS='OLD')
      READ(1,*)A2,A3,D3,D4
      WRITE(*,*)'NO OF DATAS ?'
      READ(*,*)NO
      VELOCITY = 0.03D0
      DO 10 I = 1,NO
      READ(4,*)X,Y,Z
      SL1 = A2 - X
      SL2 = Z
      SLOPE = DATAN2D(SL1,SL2)
      VX = VELOCITY * DCOSD(SLOPE)
      VY = 0.0D0
      VZ = VELOCITY * DCOSD(270.0D0+SLOPE)
      WRITE(3,*)VX,VY,VZ
10    CONTINUE
      STOP
      END


C     ********************************************************
C     A PROGRAM FOR GENERATING THE CARTESIAN VELOCITIES FOR
C     A PUMA-560 MANIPULATOR
C     ********************************************************
      IMPLICIT REAL*8(A-H,O-Z)
```

```
      INTRINSIC DATAN2D,DSIND,DCOSD
      DIMENSION RJ(3,3),VEL(3),THDT(3),RJINV(3,3)
      DIMENSION FTHDT(3)
      OPEN(UNIT=1,FILE='PUMA.DAT',STATUS='OLD')
      OPEN(UNIT=2,FILE='RANG.MAT',STATUS='OLD')
      OPEN(UNIT=3,FILE='PUMAVEL.MAT',STATUS='OLD')
      OPEN(UNIT=5,FILE='TDOT.MAT',STATUS='NEW')
      OPEN(UNIT=6,FILE='DETER.MAT',STATUS='NEW')
      READ(1,*)A2,A3,D3,D4
      WRITE(*,*)'NO OF DATAS ?'
      READ(*,*)NO
      DO 43 I = 1,NO
      READ(2,*)T1,T2,T3
      S1 = DSIND(T1)
      S2 = DSIND(T2)
      C1 = DCOSD(T1)
      C2 = DCOSD(T2)
      C23 = DCOSD(T2+T3)
      S23 = DSIND(T2+T3)
      RJ(1,1) = -A3*S1*C23 + D4*S1*S23- A2*C2*S1 - D3*C1
      RJ(1,2) = -A3*C1*S23 - D4*C1*C23 - A2*S2*C1
      RJ(1,3) = -A3*C1*S23 - D4*C1*C23
      RJ(2,1) = A3*C1*C23 - D4*C1*S23 + A2*C2*C1 - D3*S1
      RJ(2,2) = -A3*S1*S23 - D4*S1*C23 - A2*S2*S1
      RJ(2,3) = -A3*S1*S23 - D4*S1*C23
      RJ(3,1) = 0.0D0
      RJ(3,2) = -A3*C23 + D4*S23 - A2*C2
      RJ(3,3) = -A3*C23 + D4*S23
C     ******************************************************
C     COMPUTE THE DETERMINANT OF JACOBIAN
C     ******************************************************
      XT1 = RJ(2,2)*RJ(3,3) - RJ(3,2)*RJ(2,3)
      XT2 = RJ(2,1)*RJ(3,3) - RJ(3,1)*RJ(2,3)
      XT3 = RJ(2,1)*RJ(3,2) - RJ(3,1)*RJ(2,2)
      DETER = RJ(1,1)*XT1 - RJ(1,2)*XT2 + RJ(1,3)*XT3
      WRITE(6,*)DETER
      CALL DLINRG(3,RJ,3,RJINV,3)
      READ(3,*)VEL(1),VEL(2),VEL(3)
      CALL DMURRV(3,3,RJINV,3,3,VEL,1,3,FTHDT)
      WRITE(5,*)FTHDT(1),FTHDT(2),FTHDT(3)
      DO 50 INK = 1,3
      DO 50 JNK = 1,3
      RJ(INK,JNK) = 0.0D0
      RJINV(INK,JNK) = 0.0D0
```

```
50    CONTINUE
43    CONTINUE
      STOP
      END


      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION TDOT1(100),TDOT2(100),TDOT3(100)
      OPEN(UNIT=1,FILE='TDOT.MAT',STATUS='OLD')
      OPEN(UNIT=2,FILE='SDOT.MAT',STATUS='NEW')
      READ(*,*)NO
      DO 10 I = 1, NO
      READ(1,*)TDOT1(I),TDOT2(I),TDOT3(I)
10    CONTINUE
      CALL MINMAX(TDOT2,SMALL2,PLARGE2,NO)
      CALL MINMAX(TDOT3,SMALL3,PLARGE3,NO)
      DO 20 I = 1,NO
c     SDOT1 = (TDOT1(I) - SMALL1)/(PLARGE1 - SMALL1)
      SDOT2 = (TDOT2(I) - SMALL2)/(PLARGE2 - SMALL2)
      SDOT3 = (TDOT3(I) - SMALL3)/(PLARGE3 - SMALL3)
      WRITE(2,*)SDOT1,SDOT2,SDOT3
20    CONTINUE
      STOP
      END

C     A NEURAL NET  PROGRAM (BACK-PROPAGATION ALGORITHM)
C     **************************************************
C     VELOCITY ANALYSIS OF PUMA-560 MANIPULATOR
C     ************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DEXP
      DIMENSION V(20,20),W(20,20)
      DIMENSION XDOT(100),YDOT(100),ZDOT(100)
      DIMENSION TDOT1(100),TDOT2(100),TDOT3(100)
      DIMENSION DTDOT1(100),DTDOT2(100),DTDOT3(100)
      DIMENSION RI(100),RM(100),RO(100)
      DIMENSION
     % FRM(100),FRO(100),FTDOT1(100),FTDOT2(100),FTDOT3(100)
      DIMENSION FRMD(100),FROD(100)
      DIMENSION EOS(100),EMS(100)
      OPEN(UNIT=1,FILE='RAKO.DAT',STATUS='OLD')
      OPEN(UNIT=2,FILE='SDOT.MAT',STATUS='OLD')
      OPEN(UNIT=3,FILE='PUMAVEL.MAT',STATUS='OLD')
      OPEN(UNIT=4,FILE='ALPHA.MAT',STATUS='NEW')
```

```
      OPEN(UNIT=5,FILE='BETA.MAT',STATUS='NEW')
      OPEN(UNIT=6,FILE='GAMMA.MAT',STATUS='NEW')
C     **************************************************
C     ENTER NO. OF INPUT,MIDDLE AND OUTPUT LAYER NEURONS
      READ(1,*)INNO
      READ(1,*)MIDNO
      READ(1,*)NOUTNO
C     ENTER NO. OF INPUT TRAINING DATAS
      READ(1,*)NODATA
C     ENTER NO. OF ITERATIONS
      READ(1,*)ITER
C     ENTER THE LEARNING RATE
      READ(1,*)ETA
C     **************************************************
C     READ THE TRAINING INPUT VALUES & THEIR CORRESPONDING
C     OUTPUT VALUES
C     **************************************************
      DO 10 I = 1,NODATA
      READ(2,*)TDOT1(I),TDOT2(I),TDOT3(I)
      READ(3,*)XDOT(I),YDOT(I),ZDOT(I)
      FTDOT1(I) = (1.0D0/(1.0D0 + DEXP(-TDOT1(I))))
      FTDOT2(I) = (1.0D0/(1.0D0 + DEXP(-TDOT2(I))))
      FTDOT3(I) = (1.0D0/(1.0D0 + DEXP(-TDOT3(I))))
10    CONTINUE
C     **************************************************
C     RANDOM NUMBER GENERATION FOR INITIAL INPUT AND
C     OUTPUT WEIGHT MATRICES
C     **************************************************
C     SEED FOR RANDOM NUMBER GENERATOR
      ISEED = 23148
      JSEED = 32124
      DO 20 I = 1,MIDNO
      DO 20 J = 1,INNO
      RV = RAN(ISEED)*2.0D0 - 1.0D0
      V(I,J) = RV
20    CONTINUE
      DO 30 I = 1,NOUTNO
      DO 30 J = 1,MIDNO
      RW = RAN(JSEED)*2.0D0 - 1.0D0
30    CONTINUE
C     **************************************************
C     TRAINING STARTS HERE
C     **************************************************
      DO 40 NO = 1,ITER
```

```
C     ************************************************
      DO 50 KD = 1,NODATA
      RI(1) = XDOT(KD)
      RI(2) = YDOT(KD)
      RI(3) = ZDOT(KD)
      DO 60 KR = 1,MIDNO
      RM(KR) = 0.0D0
      DO 70 KP = 1,INNO
      RM(KR) = RM(KR) + V(KR,KP)*RI(KP)
70    CONTINUE
      FRM(KR) = (1.0D0/(1.0D0 + DEXP(-RM(KR))))
      FRMD(KR) = FRM(KR)*(1.0D0 - FRM(KR))
60    CONTINUE
      DO 80 KR = 1,NOUTNO
      RO(KR) = 0.0D0
      DO 90 KP = 1,MIDNO
      RO(KR) = RO(KR) + W(KR,KP)*FRM(KP)
90    CONTINUE
      FRO(KR) = (1.0D0/(1.0D0 + DEXP(-RO(KR))))
      FROD(KR) = FRO(KR)*(1.0D0 - FRO(KR))
80    CONTINUE
C     ****************************************************
C     ERROR UPDATE
C     ****************************************************
      DO 100 J = 1,MIDNO
      EMS(J) = 0.0D0
100   CONTINUE
      EOS(1) = FTDOT1(KD) - FRO(1)
      EOS(2) = FTDOT2(KD) - FRO(2)
      EOS(3) = FTDOT3(KD) - FRO(3)
      DO 110 IE = 1,NOUTNO
      DO 120 JE = 1,MIDNO
      EMS(JE) = EMS(JE) + EOS(IE)*FROD(IE)*W(IE,JE)
120   CONTINUE
110   CONTINUE
C     ********************************************************
C     WEIGHT UPDATE
C     ********************************************************
      DO 200 IRT = 1,NOUTNO
      DO 210 JRT = 1,MIDNO
      W(IRT,JRT) = W(IRT,JRT) + ETA*EOS(IRT)*FROD(IRT)*FRM(JRT)
210   CONTINUE
200   CONTINUE
      DO 220 IRT = 1,MIDNO
```

```
      DO 230 JRT = 1,INNO
      V(IRT,JRT) = V(IRT,JRT) + ETA*EMS(IRT)*FRMD(IRT)*RI(JRT)
230   CONTINUE
220   CONTINUE
C     ******************************************************
      DO 300 I = 1,MIDNO
      RM(I) = 0.0D0
      DO 310 J = 1,INNO
      RM(I) = RM(I) + V(I,J)*RI(J)
310   CONTINUE
      FRM(I) = (1.0D0 /(1.0D0 + DEXP(-RM(I))))
300   CONTINUE
      DO 320 I = 1,NOUTNO
      RO(I) = 0.0D0
      DO 330 J = 1,MIDNO
      RO(I) = RO(I) + W(I,J)*FRM(J)
330   CONTINUE
320   CONTINUE
C     ********************************************************
C     FINAL OUTPUT AT THE END OF EACH ITERATION
C     ******************************************************
      DTDOT1(KD) = RO(1)
      DTDOT2(KD) = RO(2)
      DTDOT3(KD) = RO(3)
50    CONTINUE
40    CONTINUE
C     ****************************************************
C     COMPARISON OF NEURAL AND ACTUAL OUTPUT
C     ****************************************************
      DO 500 IT = 1,NODATA
      WRITE(4,*)DTDOT1(IT),TDOT1(IT)
      WRITE(5,*)DTDOT2(IT),TDOT2(IT)
      WRITE(6,*)DTDOT3(IT),TDOT3(IT)
500   CONTINUE
      STOP
      END


      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION TDOT1(100),TDOT2(100),TDOT3(100)
      DIMENSION DTDOT1(100),DTDOT2(100),DTDOT3(100)
      OPEN(UNIT=1,FILE='TDOT.MAT',STATUS='OLD')
      OPEN(UNIT=3,FILE='ALPHA.MAT',STATUS='OLD')
      OPEN(UNIT=4,FILE='BETA.MAT',STATUS='OLD')
```

```
      OPEN(UNIT=5,FILE='GAMMA.MAT',STATUS='OLD')
      OPEN(UNIT=6,FILE='REBAK.MAT',STATUS='NEW')
      OPEN(UNIT=7,FILE='RERR.MAT',STATUS='NEW')
      READ(*,*)NO
      DO 10 I = 1, NO
      READ(1,*)TDOT1(I),TDOT2(I),TDOT3(i)
      READ(3,*)DTDOT1(I)
      READ(4,*)DTDOT2(I)
      READ(5,*)DTDOT3(I)
10    CONTINUE
      CALL MINMAX(TDOT2,SMALL2,PLARGE2,NO)
      CALL MINMAX(TDOT3,SMALL3,PLARGE3,NO)
      DO 20 I = 1,NO
      SDOT2 = DTDOT2(I)*(PLARGE2 - SMALL2) + SMALL2
      SDOT3 = DTDOT3(I)*(PLARGE3 - SMALL3) + SMALL3
      WRITE(6,*)SDOT1,SDOT2,SDOT3
C     TO FIND OUT THE MAGNITUDE OF ERROR
      UPP = TDOT1(I) - SDOT1
      VPP = TDOT2(I) - SDOT2
      WPP = TDOT3(I) - SDOT3
      ERR = SQRT(UPP*UPP + VPP*VPP + WPP*WPP)
      WRITE(7,*)ERR
20    CONTINUE
      STOP
      END

C     ********************************************************
C     A SIMPLE PROGRAM FOR CHECKING THE CARTESIAN VELOCITIES
C     FOR A PUMA-560 MANIPULATOR
C     ********************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DATAN2D,DSIND,DCOSD
      DIMENSION RJ(3,3),VEL(3),THDT(3),RJINV(3,3)
      DIMENSION FTHDT(3),TDOT(3),OVEL(3),OTDOT(3)
      OPEN(UNIT=1,FILE='PUMA.DAT',STATUS='OLD')
      OPEN(UNIT=2,FILE='RANG.MAT',STATUS='OLD')
      OPEN(UNIT=7,FILE='REBAK.MAT',STATUS='OLD')
      OPEN(UNIT=8,FILE='VDOT.MAT',STATUS='NEW')
      OPEN(UNIT=9,FILE='TDOT.MAT',STATUS='OLD')
      OPEN(UNIT=10,FILE='OVDOT.MAT',STATUS='NEW')
      READ(1,*)A2,A3,D3,D4
      WRITE(*,*)'NO OF DATAS ?'
      READ(*,*)NO
      DO 43 I = 1,NO
```

```
      READ(2,*)T1,T2,T3
      S1 = DSIND(T1)
      S2 = DSIND(T2)
      C1 = DCOSD(T1)
      C2 = DCOSD(T2)
      C23 = DCOSD(T2+T3)
      S23 = DSIND(T2+T3)
      RJ(1,1) = -A3*S1*C23 + D4*S1*S23- A2*C2*S1 - D3*C1
      RJ(1,2) = -A3*C1*S23 - D4*C1*C23 - A2*S2*C1
      RJ(1,3) = -A3*C1*S23 - D4*C1*C23
      RJ(2,1) = A3*C1*C23 - D4*C1*S23 + A2*C2*C1 - D3*S1
      RJ(2,2) = -A3*S1*S23 - D4*S1*C23 - A2*S2*S1
      RJ(2,3) = -A3*S1*S23 - D4*S1*C23
      RJ(3,1) = 0.0D0
      RJ(3,2) = -A3*C23 + D4*S23 - A2*C2
      RJ(3,3) = -A3*C23 + D4*S23
      READ(7,*)TDOT(1),TDOT(2),TDOT(3)
      READ(9,*)OTDOT(1),OTDOT(2),OTDOT(3)
      CALL DMURRV(3,3,RJ,3,3,TDOT,1,3,VEL)
      CALL DMURRV(3,3,RJ,3,3,OTDOT,1,3,OVEL)
      WRITE(8,*)VEL(1),VEL(2),VEL(3)
      WRITE(10,*)OVEL(1),OVEL(2),OVEL(3)
      SS = SQRT(VEL(1)*VEL(1)+VEL(2)*VEL(2)+VEL(3)*VEL(3))
      PS = SQRT(OVEL(1)*OVEL(1)+OVEL(2)*OVEL(2)+OVEL(3)*OVEL(3))
      PRINT *,SS,PS
      DO 50 INK = 1,3
      DO 50 JNK = 1,3
      RJ(INK,JNK) = 0.0D0
50    CONTINUE
43    CONTINUE
      STOP
      END
```

A.2 ACCELERATION ANALYSIS OF A TWO-LINK MANIPULATOR

```
C     ************************************************
C      ACCELERATION ANALYSIS OF A TWO-LINK
C      PLANAR MANIPULATOR
C
C     STEPS FOLLOWED :
C     ****************
C
C     1. FIND OUT THE CO-ORDINATES OF THE CIRCULAR
```

```
C        PATH
C      2. FOR THE GIVEN COORDINATES FIND OUT THE JOINT
C        VARIABLES TH_K1 AND TH_K2
C      3. RESOLVE TANGENTIAL VELOCITY V_T IN THE GLOBAL
C        COORDINATE SYSTEM
C      4. EVALUATE THE ROTATION RATES THDT_K1 AND THDT_K2
C        USING JACOBIAN MATRIX
C      5. RESOLVE ACCELERATION A_P IN THE GLOBAL C.S
C      6. FIND OUT THE JOINT ACCELERATIONS THDDT_K1 AND
C        THDDT_K2
C      *************************************************
c      STEP 1: FIND THE COORDINATES OF THE CIRCULAR
C            TRAJECTORY
c      *************************************************
       IMPLICIT REAL*8(A-H,O-Z)
       INTRINSIC DSIND,DCOSD,DATAN2D
       REAL*8 TR1(4,4),TR2(4,4),TR3(4,4),XY(4),PV(4),TR4(4,4)
       REAL*8 TR5(4,4)
       OPEN(1, FILE = 'LINK.DET', STATUS = 'OLD')
       OPEN(2, FILE = 'XY.MAT', STATUS = 'NEW')
       READ(1,*)RL1,RL2
       READ(1,*)ST
       READ(1,*)NPOINT
C      STEP ANGLE FOR COORDINATE GENERATION
       READ(1,*)STANG
C      RADIUS
       READ(1,*)RD
       CALL TRANS(45.0,0.0,0.0,0.0,TR1)
       CALL TRANS(0.0,RL1,0.0,0.0,TR2)
       CALL MATMAT(TR1,TR2,TR3,4,4,4)
       PHI = ST
       DO 100 I = 1,NPOINT
       CALL TRANS(PHI,0.1,0.0,0.0,TR4)
       CALL MATMAT(TR3,TR4,TR5,4,4,4)
       PV(1) = RD
       PV(2) = 0.0D0
       PV(3) = 0.0D0
       PV(4) = 1.0D0
       CALL MATVEC(TR5,PV,XY,4,4)
       WRITE(2,*)XY(1),XY(2)
       PHI = PHI + STANG
100    CONTINUE
       STOP
       END
```

```
      SUBROUTINE MATVEC(A,B,C,M,N)
      IMPLICIT REAL*8(A-H,O-Z)
      REAL*8 A(4,4),B(4),C(4)
      DO 1200 I = 1,M
      C(I) = 0.0D0
      DO 1200 J = 1,N
      C(I) = C(I) + A(I,J)*B(J)
 1200 CONTINUE
      RETURN
      END

      SUBROUTINE MATMAT(A,B,C,M,N,L)
      IMPLICIT REAL*8(A-H,O-Z)
      REAL*8 A(4,4),B(4,4),C(4,4)
      DO 1300 I = 1,M
      DO 1300 J = 1,N
      C(I,J) = 0.0D0
      DO 1300 K = 1,L
      C(I,J) = C(I,J) + A(I,K)*B(K,J)
 1300 CONTINUE
      RETURN
      END


      SUBROUTINE ROTZ(ANG,RZ)
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DSIND,DCOSD,DATAN2D
      REAL*8 RZ(3,3)
      RZ(1,1) = DCOSD(ANG)
      RZ(1,2) = -DSIND(ANG)
      RZ(1,3) = 0.0D0
      RZ(2,1) = DSIND(ANG)
      RZ(2,2) = DCOSD(ANG)
      RZ(2,3) = 0.0D0
      RZ(3,1) = 0.0D0
      RZ(3,2) = 0.0D0
      RZ(3,3) = 1.0D0
      RETURN
      END

      SUBROUTINE TRANS(ANG,PX,PY,PZ,TR)
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DSIND,DCOSD,DATAN2D
      REAL*8 TR(4,4)
```

```
      TR(1,1) = DCOSD(ANG)
      TR(1,2) = -DSIND(ANG)
      TR(1,3) = 0.0D0
      TR(1,4) = PX
      TR(2,1) = DSIND(ANG)
      TR(2,2) = DCOSD(ANG)
      TR(2,3) = 0.0D0
      TR(2,4) = PY
      TR(3,1) = 0.0D0
      TR(3,2) = 0.0D0
      TR(3,3) = 1.0D0
      TR(3,4) = 0.0D0
      TR(4,1) = 0.0D0
      TR(4,2) = 0.0D0
      TR(4,3) = 0.0D0
      TR(4,4) = 1.0D0
      RETURN
      END

C     ***************************************************
C     STEP 2: FIND THE JOINT COORDINATES FOR THE
C          CIRCULAR TRAJECTORY
C     ***************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DSIND,DCOSD,DATAN2D,DSQRT
      REAL*8 THET(4)
      OPEN(1, FILE = 'LINK.DET', STATUS = 'OLD')
      OPEN(2, FILE = 'XY.MAT', STATUS = 'OLD')
      OPEN(3, FILE = 'THET.MAT', STATUS = 'NEW')
      READ(1,*)RL1,RL2
      READ(1,*)ST
      READ(1,*)NPOINT
C     STEP ANGLE FOR COORDINATE GENERATION
      READ(1,*)STANG
      DO 200 I = 1,NPOINT
      READ(2,*)XP,YP
      RJJ = 2.0D0 * YP * RL1
      RII = 2.0D0 * XP * RL1
      RKK = XP*XP + YP*YP + RL1*RL1 - RL2*RL2
      RTT = RJJ*RJJ + RII*RII - RKK*RKK
      THET(1) = DATAN2D(RJJ,RII) + DATAN2D(DSQRT(RTT),RKK)
      PAR1 = YP - RL1*DSIND(THET(1))
      PAR2 = XP - RL1*DCOSD(THET(1))
      THET(2) = DATAN2D(PAR1,PAR2) - THET(1)
```

```
      WRITE(3,*)THET(1),THET(2)
200   CONTINUE
      STOP
      END


C     ****************************************************
C     STEP 3: RESOLVE TANGENTIAL VELOCITY V_T IN THE
c           GLOBAL COORDINATE SYSTEM
C     ****************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DSIND,DCOSD,DATAN2D
      REAL*8 XY(2),VP(3),RZZA(3,3),VPG(3)
      OPEN(1, FILE = 'LINK.DET', STATUS = 'OLD')
      OPEN(2, FILE = 'VTRES.DAT', STATUS = 'OLD')
      OPEN(3, FILE = 'XY.MAT', STATUS = 'OLD')
      OPEN(4, FILE = 'VPG.MAT', STATUS = 'NEW')
      READ(1,*)RL1,RL2
      READ(1,*)ST
      READ(1,*)NPOINT
C     STEP ANGLE FOR COORDINATE GENERATION
      READ(1,*)STANG
      READ(2,*)V_T,XC,YC,RAD
      DO 300 I = 1,NPOINT
      READ(3,*)XY(1),XY(2)
      PKL = XY(2)-YC
      PKK = XY(1)-XC
      ALPH = DATAN2D(PKL,PKK)
      VP(1) = V_T*DCOSD(90.0D0+ALPH)
      VP(2) = V_T*DCOSD(ALPH)
      VP(3) = 1.0D0
      WRITE(4,*)VP(1),VP(2)
300   CONTINUE
      STOP
      END


C     ********************************************************
C     STEP 4: EVALUATE THE ROTATION RATES THDT_K1 AND THDT_K2
C           USING JACOBIAN MATRIX
C     ********************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DSIND,DCOSD,DATAN2D
      REAL*8 THET(2),VPG(2),THDT(2),RINV(2,2)
```

```
      OPEN(1, FILE = 'LINK.DET', STATUS = 'OLD')
      OPEN(2, FILE = 'THET.MAT', STATUS = 'OLD')
      OPEN(3, FILE = 'VPG.MAT', STATUS = 'OLD')
      OPEN(4, FILE = 'THDT.MAT', STATUS = 'NEW')
      READ(1,*)RL1,RL2
      READ(1,*)ST
      READ(1,*)NPOINT
C     STEP ANGLE FOR COORDINATE GENERATION
      READ(1,*)STANG
      DO 400 I = 1,NPOINT
      READ(2,*)THET(1),THET(2)
      READ(3,*)VPG(1),VPG(2)
      UT11 = -RL1*DSIND(THET(1)) - RL2*DSIND(THET(1)+THET(2))
      UT12 = -RL2*DSIND(THET(1)+THET(2))
      UT21 = RL1*DCOSD(THET(1)) + RL2*DCOSD(THET(1)+THET(2))
      UT22 = RL2*DCOSD(THET(1)+THET(2))
      DET = UT11*UT22 - UT21*UT12
      RINV(1,1) = UT22/DET
      RINV(1,2) = -UT12/DET
      RINV(2,1) = -UT21/DET
      RINV(2,2) = UT11/DET
      CALL MATVEC(RINV,VPG,THDT,2,2)
      WRITE(4,*)THDT(1),THDT(2)
400   CONTINUE
      STOP
      END

C     ****************************************************
C     STEP 5: RESOLVE ACCELERATION A_P IN THE GLOBAL
C           COORDINATE SYSTEM
C     ****************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DSIND,DCOSD,DATAN2D
      REAL*8 XY(2),AP(3),RZAP(3,3),APG(3),bp(2)
      OPEN(1, FILE = 'LINK.DET', STATUS = 'OLD')
      OPEN(2, FILE = 'VTRES.DAT', STATUS = 'OLD')
      OPEN(3, FILE = 'XY.MAT', STATUS = 'OLD')
      OPEN(4, FILE = 'APG.MAT', STATUS = 'NEW')
      READ(1,*)RL1,RL2
      READ(1,*)ST
      READ(1,*)NPOINT
C     STEP ANGLE FOR COORDINATE GENERATION
      READ(1,*)STANG
      READ(2,*)V_T,XC,YC,RAD
```

```
      DO 300 I = 1,NPOINT
      READ(3,*)XY(1),XY(2)
      PKK = XY(2)-YC
      PKL = XY(1)-XC
      ALPH = DATAN2D(PKK,PKL)
      AP(1) = (-(V_T*V_T)/RAD)*DCOSD(ALPH)
      AP(2) = (-(V_T*V_T)/RAD)*DSIND(ALPH)
      AP(3) = 1.0D0
      WRITE(4,*) AP(1),AP(2)
300   CONTINUE
      STOP
      END

C     ******************************************************
C     STEP 6: FIND OUT THE JOINT ACCELERATIONS THDDT_K1 AND
C           THDDT_K2
C     ******************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      INTRINSIC DSIND,DCOSD,DATAN2D
      REAL*8 THET(2),VPG(2),THDT(2),RINV(2,2),THDDT(2)
      REAL*8 APG(2),AXT(2),PXT(2),PT1(2,2)
      OPEN(1, FILE = 'LINK.DET', STATUS = 'OLD')
      OPEN(2, FILE = 'THET.MAT', STATUS = 'OLD')
      OPEN(3, FILE = 'VPG.MAT', STATUS = 'OLD')
      OPEN(4, FILE = 'THDT.MAT', STATUS = 'OLD')
      OPEN(7, FILE = 'APG.MAT', STATUS = 'OLD')
      OPEN(8, FILE = 'THDDT.MAT', STATUS = 'NEW')
      READ(1,*)RL1,RL2
      READ(1,*)ST
      READ(1,*)NPOINT
C     STEP ANGLE FOR COORDINATE GENERATION
      READ(1,*)STANG
      DO 400 I = 1,NPOINT
      READ(2,*)THET(1),THET(2)
      READ(3,*)VPG(1),VPG(2)
      READ(4,*)THDT(1),THDT(2)
      READ(7,*)APG(1),APG(2)
      PT1(1,1) = -RL1*DCOSD(THET(1))*THDT(1)
     %  - RL2*DCOSD(THET(1)+THET(2))*(THDT(1) + THDT(2))
      PT1(1,2) = -RL2*DCOSD(THET(1)+THET(2))*(THDT(1)+THDT(2))
      PT1(2,1) = -RL1*DSIND(THET(1))*THDT(1)
     %  - RL2*DSIND(THET(1)+THET(2))*(THDT(1) + THDT(2))
      PT1(2,2) = -RL2*DSIND(THET(1)+THET(2))*(THDT(1)+THDT(2))
```

```
      CALL MATVEC(PT1,THDT,AXT,2,2)
      PXT(1) = APG(1) - AXT(1)
      PXT(2) = APG(2) - AXT(2)
      UT11 = -RL1*DSIND(THET(1)) - RL2*DSIND(THET(1)+THET(2))
      UT12 = -RL2*DSIND(THET(1)+THET(2))
      UT21 = RL1*DCOSD(THET(1)) + RL2*DCOSD(THET(1)+THET(2))
      UT22 = RL2*DCOSD(THET(1)+THET(2))
      DET = UT11*UT22 - UT21*UT12
      RINV(1,1) = UT22/DET
      RINV(1,2) = -UT12/DET
      RINV(2,1) = -UT21/DET
      RINV(2,2) = UT11/DET
      CALL MATVEC(RINV,PXT,THDDT,2,2)
      WRITE(8,*)THDDT(1),THDDT(2)
400   CONTINUE
      STOP
      END

C     ****************************************************
C     GENERATION OF [C] MATRIX FOR LINEAR PROGRAMMING
C     ************************************************
      DIMENSION X1(23),X2(23),X3(23),X4(23),X5(23)
      DIMENSION X6(23),X7(23),X8(23),X9(23),X10(23)
      DIMENSION A(1,1100),Y(23)
      OPEN(1, FILE = 'ALPGa.IN', STATUS = 'OLD')
      OPEN(2, FILE = 'CLPGa.OUT', STATUS = 'NEW')
      DO 14 I = 1,23
      READ(1,*)X1(I)
14    CONTINUE
      DO 15 I = 1,23
      READ(1,*)X2(I)
15    CONTINUE
      DO 16 I = 1,23
      READ(1,*)X3(I)
16    CONTINUE
      DO 17 I = 1,23
      READ(1,*)X4(I)
17    CONTINUE
      DO 18 I = 1,23
      READ(1,*)X5(I)
18    CONTINUE
      DO 19 I = 1,23
      READ(1,*)X6(I)
19    CONTINUE
```

```
      DO 20 I = 1,23
      READ(1,*)X7(I)
20    CONTINUE
      DO 21 I = 1,23
      READ(1,*)X8(I)
21    CONTINUE
      DO 22 I = 1,23
      READ(1,*)X9(I)
22    CONTINUE
      DO 23 I = 1,23
      READ(1,*)X10(I)
23    CONTINUE
      DO 44 K = 1,23
      Y(K) = X1(K) + X2(K) + X3(K) + X4(K) + X5(K) + X6(K) + X7(K)
     # + X8(K) + X9(K) + X10(K)
44    CONTINUE
      NS = 1
      NT = 46
80    CONTINUE
      IF(NT.GT.1100) GOTO 100
      MT = 1
      DO 310 I = NS,NT,2
      A(1,I) = Y(MT)
      MT = MT + 1
310   CONTINUE
      MT = 1
      DO 320 I = NS+1,NT,2
      A(1,I) = -Y(MT)
      MT = MT + 1
320   CONTINUE
      DO 330 I = NT+1,NT+4
      A(1,I) = 10.0
330   CONTINUE
      DO 340 I = NT+2,NT+3
      A(1,I) = -10.0
340   CONTINUE
      NS = NS + 50
      NT = NT + 50
      GO TO 80
100   CONTINUE
1050  CONTINUE
      WRITE(2,*)(A(1,J),J = 1,1100)
      STOP
      END
```

```
C       *******************LPNEURO.FOR************************
C       LP-NEURO METHOD SUBROUTINE
C       *************************************************
C       A LP APPROACH FOR NEURAL NETWORKS
C       *************************************************
        DIMENSION YX(1),YXSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1)
        DIMENSION YW(150),ZX(1100),ZA(220,1100),ZB(220),ZC(1100)
        DIMENSION ZAP(220,1100),ZCP(1100),ZBP(220),ZW(50000)
        DIMENSION BBW(22,23),BBC(22),BBD(22),ZTOB(23)
        COMMON
     % /SEEK/IDATA,IPRINT,NSHOT,NTEST,MAXM,F,G,TOL,ZERO,
     %  R,REDUCE
        COMMON /BL1/ZC,ZX,ZW,ZAP,ZCP,ZBP
        COMMON /BL3/ZB
        COMMON /BL2/MZ,NZ,NUTS,NT,NI,NOUT,NOI
        COMMON /FAL/ZTOB
        COMMON /ANT/ZA
        OPEN(2, FILE = 'BLPGA.OUT', STATUS = 'OLD')
        OPEN(3, FILE = 'CLPGA.OUT', STATUS = 'OLD')
        OPEN(14, FILE = 'LIN.DAT', STATUS = 'OLD')
        OPEN(15, FILE = 'KARI.DAT', STATUS = 'OLD')
        OPEN(8, FILE = 'OBJ.MAT', STATUS = 'NEW')
        OPEN(10, FILE ='FWT.MAT', STATUS = 'NEW')
        READ(14,*)MZ
        READ(14,*)NZ
        READ(14,*)NUTS
        READ(14,*)NT,NI
        READ(14,*)NOUT,NOI
        DO 884 I = 1,23
        READ(15,*)ZTOB(I)
884     CONTINUE
        MAXM = 10
        CALL ALPG
        DO 196 I = 1,MZ
        READ(2,*)ZB(I)
196     CONTINUE
        READ(3,*)(ZC(I),I = 1,NZ)
        NY = 1
        NCONS = 0
        NEQUS = 0
        NPENAL = 3
        IDATA = 0
        DATA RMAX/10.0/
```

```
      DATA RMIN/-10.0/
      DATA YXSTRT/0.611/
      NOISE = 1
      CALL
      SEEK(NY,NCONS,NEQUS,NPENAL,RMAX,RMIN,YXSTRT,YX,YU,PHI,
%      PSI,NVIOL,YW)
      CALL ANSWER(YU,YX,PHI,PSI,NY,NCONS,NEQUS)
      WRITE(*,*)'DO YOU WISH TO CONTINUE ?'
      WRITE(*,*)'1:  YES '
      WRITE(*,*)'2: NO '
      READ(*,*)NDEC
      IF(NDEC.EQ.1) THEN
      GO TO 1
      ELSE
      END IF
      STOP
      END


      SUBROUTINE UREAL(YX,YU)
      DIMENSION YX(1),YXSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1)
      DIMENSION YW(150),ZX(1100),ZA(220,1100),ZB(220),ZC(1100)
      DIMENSION ZAP(220,1100),ZBP(220),ZCP(1100)
      DIMENSION ZW(50000),ZTOB(22)
      DIMENSION BBW(22,23),BBC(22),BBD(22),D(2)
      DIMENSION SAK(4),RT3(22)
      COMMON /BL1/ZC,ZX,ZW,ZAP,ZCP,ZBP
      COMMON /BL3/ZB
      COMMON /BL2/MZ,NZ,NUTS,NT,NI,NOUT,NOI
      COMMON /SIMPLE/NSTOP,IDATA,NNDEX
      COMMON /PAKS/BBW,BBC,BBD
      COMMON /FAL/ZTOB
      COMMON /ANT/ZA
      COMMON /DEPUT/D,RMX,RMN
      IDATA = 0
      NNDEX = 3
      DO 20 I = 1,MZ
      KR = 1
      DO 30 J = 1,NZ
      IF(J.GT.(50*KR)) THEN
      KR = KR + 1
      END IF
      IF((J.GE.(1+50*(KR-1))).AND.(J.LT.(47+50*(KR-1)))) THEN
      ZAP(I,J) = YX(1)*ZA(I,J)
```

```
      ELSE
      ZAP(I,J) = ZA(I,J)
      END IF
30    CONTINUE
20    CONTINUE
      SQM = 0.0
      DO 40 I = 1,MZ
      ZBP(I) = YX(1)*ZB(I)
      SQM = SQM + ZB(I)
40    CONTINUE
      KR = 1
      DO 50 J = 1,NZ
      IF(J.GT.(50*KR)) THEN
      KR = KR + 1
      END IF
      IF((J.GE.(1 + 50*(KR-1))).AND.(J.LT.(47+50*(KR-1)))) THEN
      ZCP(J) = YX(1)*ZC(J)
      ELSE
      ZCP(J) = ZC(J)
      END IF
50    CONTINUE
      CALL SIMPLE(NZ,MZ,ZAP,ZBP,ZCP,ZX,ZU,ZW)
      CALL WASS(ZX,BBW,BBC,BBD)
      YT1 = 0.0
      YT2 = 0.0
      YT3 = 0.0
      DO 55 NM = 1,46,2
      DO 60 MM = NM,1100,50
      PRINT *,YX(1),ZC(NM),ZX(MM)
      YT1 = YX(1)*(ZC(NM)*ZX(MM)) + YT1
60    CONTINUE
55    CONTINUE
      DO 65 NM = 2,46,2
      DO 70 MM = NM,1100,50
      YT2 = YX(1)*(ZC(NM)*ZX(MM)) + YT2
70    CONTINUE
65    CONTINUE
      DO 75 NM = 47,50
      DO 80 MM = NM,1100,50
      YT3 = (ZC(NM)*ZX(MM)) + YT3
80    CONTINUE
75    CONTINUE
      YT4 = YX(1)*SQM
      CALL RMIXD(BBW,BBC,BBD,YX,ZTOB,RT3)
```

```
      YU = (YT1 + YT2 + YT3 - YT4)**2
      WRITE(*,*)'SLOPE IS',YX(1)
      WRITE(*,*)'SEEK OBJECTIVE FUNCTION IS',YU
      IF (NSW.EQ.1) THEN
      STOP
      END IF
      IF(YU.LT.(2.0E-4)) THEN
      CALL RMIXD(BBW,BBC,BBD,YX,ZTOB,RT3)
      WRITE(18,*)'LAST ITERATION'
      WRITE(18,*)(RT3(JK),JK=1,22)
      NSW = 1
      ELSE
      END IF
      RETURN
      END


      SUBROUTINE CONST(YX,NCONS,PHI)
      DIMENSION PHI(1),D(2),YX(1)
      COMMON /DEPUT/D,RMX,RMN
      PRINT *,'SEEK ITERATIONS',NOISE
      NOISE = NOISE + 1
      RETURN
      END

      SUBROUTINE EQUAL(YX,PSI,NEQUS)
      DIMENSION YX(1),PSI(1)
      RETURN
      END

      SUBROUTINE WASS(ZX,BBW,BBC,BBD)
      DIMENSION ZX(1100),BBW(22,23),BBC(22),BBD(22)
      ND = 1
      RAMYA = 0.0
      DO 432 I = 1,22
      DO 433 J = 1,23
      RAMYA = ZX(ND) - ZX(ND + 1)
      BBW(I,J) = RAMYA
      ND = ND + 2
433   CONTINUE
      BBC(I) = ZX(ND) - ZX(ND + 1)
      ND = ND + 2
      BBD(I) = ZX(ND) - ZX(ND + 1)
      ND = ND + 2
```

```
432   CONTINUE
      RETURN
      END

      SUBROUTINE RMIXD(BBW,BBC,BBD,YX,ZTOB,RT3)
      DIMENSION YX(1),CBC(22),CBD(22),RT1(22),RT2(22),RT3(22)
      DIMENSION ZTOB(23),BBW(22,23),BBC(22),BBD(22)
      DO 2339 I = 1,22
      CBC(I) = 0.0
      CBD(I) = 0.0
      RT1(I) = 0.0
      RT2(I) = 0.0
      RT3(I) = 0.0
2339  CONTINUE
      CALL RMATMUL(BBW,ZTOB,RT1,22,23)
      DO 2340 I = 1,22
      CBC(I) = (1.0/YX(1))*BBC(I)
      CBD(I) = (1.0/YX(1))*BBD(I)
2340  CONTINUE
      CALL RMATADD(RT1,CBC,RT2,22)
      CALL RMATSUB(RT2,CBD,RT3,22)
      WRITE(18,*)(RT3(JK),JK=1,22)
      RETURN
      END

      SUBROUTINE RMATMUL(A,B,C,M,N)
      DIMENSION A(M,N),B(N),C(M)
      DO 4430 I = 1,M
      C(I) = 0.0
      DO 4440 J = 1,N
      C(I) = C(I) + A(I,J)*B(J)
4440  CONTINUE
4430  CONTINUE
      RETURN
      END

      SUBROUTINE RMATADD(A,B,C,M)
      DIMENSION A(M),B(M),C(M)
      DO 4256 I = 1,M
      C(I) = A(I) + B(I)
4256  CONTINUE
      RETURN
      END
```

```
      SUBROUTINE RMATSUB(A,B,C,M)
      DIMENSION A(M),B(M),C(M)
      DO 4169 I = 1,M
      C(I) = A(I) - B(I)
4169  CONTINUE
      RETURN
      END

      SUBROUTINE ALPG
C     GENERATION OF [A] MATRIX FOR LINEAR PROGRAMMING
      DIMENSION X(23),ZA(220,1100)
      COMMON /ANT/ZA
      OPEN(17, FILE = 'ALPGA.IN', STATUS = 'OLD')
      DO 1050 NI = 1,220,22
      KNI = NI
      NS = 1
      NT = 46
      DO 714 I = 1,23
      READ(17,*)X(I)
714   CONTINUE
780   CONTINUE
      IF(NT.GT.1100) GOTO 7100
      MT = 1
      DO 710 I = NS,NT,2
      ZA(KNI,I) = X(MT)
      MT = MT + 1
710   CONTINUE
      MT = 1
      DO 720 I = NS+1,NT,2
      ZA(KNI,I) = -X(MT)
      MT = MT + 1
720   CONTINUE
      DO 730 I = NT+1,NT+4
      ZA(KNI,I) = 1.0
730   CONTINUE
      DO 740 I = NT+2,NT+3
      ZA(KNI,I) = -1.0
740   CONTINUE
      KNI = KNI + 1
      NS = NS + 50
      NT = NT + 50
      GO TO 780
7100  CONTINUE
1050  CONTINUE
```

```
RETURN
END
```

Backpropagation Method - Multilayer Neural Network

```
/**

    PROGRAM DESCRIPTION:

        THIS PROGRAM ALLOWS A USER TO BUILD A GENERALIZED
DELTA RULE NET FOR SUPERVISED LEARNING. USER CAN SPECIFY THE
NUMBER OF INPUT & OUTPUT UNITS, NUMBER OF HIDDEN LAYERS AND
NUMBER OF UNITS IN EACH HIDDEN LAYER. AFTER THE NET IS BUILT,
LEARNING TAKES PLACE IN THE NET WITH A GIVEN SET OF TRAINING
SAMPLES. USER SPECIFIES VALUES OF THE LEARNING RATE ETA, THE
MOMENTUM RATE ALPHA, MAXIMUM TOLERANCE ERRORS AND MAXIMUM
NUMBER OF ITERATIONS.

AFTER LEARNING, ALL THE INFORMATION RELEVANT TO THE
STRUCTURE OF THE NET, INCLUDING WEIGHTS AND THRESHOLDS ARE
STORED IN FILES.

OUTPUTS CAN BE GENERATED FOR NEW PATTERNS BY READING
FROM FILE AND BY RECONSTRUCTING THE NET.

TRAINING SET SAMPLES AND ADDITIONAL SAMPLES FOR PROCESSING ARE
STORED IN FILES.
**/

#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <curses.h>
#ifndef VAX /* for declaration of calloc() on PC or compatible    */
#include <malloc.h>
#endif


/* define constants used throughout functions              */

#define  NMXUNIT   50 /* max no. of units in a layer  (50)      */
#define  NMXHLR     5 /* max no. of hidden layers     ( 5)      */
#define  NMXOATTR  50 /* max no. of output features   (50)      */
#define  NMXINP   200 /* max no. of input samples     (200)     */
```

```
#define  NMXIATTR  50 /* max no. of input features     (50)      */
#define  SEXIT      3 /* exit successfully                  */
#define  RESTRT     2 /* restart                            */
#define  FEXIT      1 /* exit in failure                    */
#define  CONTNE     0 /* continue calculation                   */

/* Data base : declarations of variables */

float eta;          /** learning rate              **/
float alpha;        /** momentum rate              **/
float err_curr;     /** normalized system error    **/
float maxe;         /** max allowed system error   **/
float maxep;        /** max allowed patter error   **/
float *wtptr[NMXHLR+1];
float *outptr[NMXHLR+2];
float *errptr[NMXHLR+2];
float *delw[NMXHLR+1];
float target[NMXINP][NMXOATTR];
float input[NMXINP][NMXIATTR], ep[NMXINP];
float outpt[NMXINP][NMXOATTR];
int  nunit[NMXHLR+2], nhlayer, ninput, ninattr, noutattr;
int  result, cnt, cnt_num;
int  nsnew, nsold;
char  task_name[20];
FILE  *fp1, *fp2, *fp3, *fopen(),*foutt;
int  fplot10;

        /* random number generator
           (computer independent)  */

long randseed = 568731L;
int random()
{
   randseed = 15625L * randseed + 22221L;
   return((randseed >> 16) & 0x7FFF);
}

        /* allocate dynamic storage for the set */
void init()
{
   int len1, len2, i, k;
   float *p1, *p2, *p3, *p4;

   len1 = len2 = 0;
```

```
    nunit[nhlayer+2] = 0;

    for (i=0; i<(nhlayer + 2); i++) {
      len1 += (nunit[i] + 1) * nunit[i+1];
      len2 += nunit[i] + 1;
    }

                          /* weights */
    p1=(float *) calloc(len1+1,sizeof(float));
                          /* output  */
    p2=(float *) calloc(len2+1,sizeof(float));
                          /* errors  */
    p3=(float *) calloc(len2+1,sizeof(float));
                          /* delw    */
    p4=(float *) calloc(len1+1,sizeof(float));

           /* set up initial pointers */
    wtptr[0]  = p1;
    outptr[0] = p2;
    errptr[0] = p3;
    delw[0]   = p4;

           /* set up the rest of pointers */
    for (i=1; i < (nhlayer + 1); i++) {
      wtptr[i] = wtptr[i-1] + nunit[i] * (nunit[i-1] + 1);
      delw[i]  = delw[i-1] + nunit[i] * (nunit[i-1] + 1);
    }
    for (i=1; i < (nhlayer + 2); i++) {
      outptr[i] = outptr[i-1] + nunit[i-1] + 1;
      errptr[i] = errptr[i-1] + nunit[i-1] + 1;
    }

           /* set up threshold outputs */
    for (i=0; i < nhlayer + 1; i++) {
      *(outptr[i] + nunit[i]) = 1.0;
    }
}

           /* initialize weights with random
              numbers between -1.0 and +1.0  */
void initwt()
{
    int i, j;
```

```
    for (j=0; j < nhlayer + 1; j++)
        for (i=0; i < (nunit[j] + 1) * nunit[j + 1]; i++) {
            *(wtptr[j] + i) = (random() / pow(2.0,15.0))*2.0 - 1.0;
            *(delw[j] + i) = 0.0;
        }
}

            /* specify architecture of net and
               values of learning parameters  */
void set_up()
{
    int i;

    eta = 0.9;
    printf("\nMomentum rate eta (default = 0.9)?: ");
    scanf("%f", &eta);

    alpha = 0.7;
    printf("\nLearning rate alpha (default = 0.7)?: ");
    scanf("%f", &alpha);

    maxe = 0.01;  maxep = 0.001;
    printf("\nMax total error (default = 0.01)?: ");
    scanf("%f", &maxe);
    printf("\nMax individual error (default = 0.001)?: ");
    scanf("%f", &maxep);

    cnt_num = 1000;
    printf("\nMax number of iterations (default = 1000)?: ");
    scanf("%d", &cnt_num);

    printf("\nNumber of hidden layers?: ");
    scanf("%d", &nhlayer);

    for (i=0; i < nhlayer; i++) {
        printf("\n\tNumber of units for hidden layer %d?: ", i+1);
        scanf("%d", &nunit[i+1]);
    }

    printf("\nCreate error file? (Enter 1 for yes, 0 for no) : ");
    scanf("%d", &fplot10);

    printf("\nExecution starts ");
    printf(" -- if many iterations specified, go out for coffee...\n");
```

```
    nunit[nhlayer+1] = noutattr;
    nunit[0] = ninattr;
}

            /* read file for net architecture and learning
               parameters. File name has suffix _v.dat   */
void dread(char *taskname)
{
    int i,j,c;
    char var_file_name[20];

    strcpy(var_file_name, taskname);
    strcat(var_file_name, "_v.dat");
    if (( fp1 = fopen(var_file_name, "r")) == NULL)
    {
       perror("\n Cannot open data file ");
      exit(0);
    }

    fscanf(fp1, "%d%d%d%f%f%d%d", &ninput, &noutattr, &ninattr,
           &eta, &alpha, &nhlayer, &cnt_num);
    for (i=0; i < nhlayer + 2; i++)
      fscanf(fp1, "%d", &nunit[i]);

    if ((c=fclose(fp1)) != 0)
      printf("\nFile %s cannot be closed; error  %d ",
           var_file_name, c);
}

/* read file containing weights and thresholds
   and thresholds. File name has suffix _w.dat */

void wtread(char *taskname)
{
    int i,j,c;
    char wt_file_name[20];

    strcpy(wt_file_name, taskname);
    strcat(wt_file_name, "_w.dat");
    if (( fp2 = fopen(wt_file_name, "r")) == NULL)
    {
       perror("\n Cannot open data file ");
      exit(0);
    }
```

```
    for (i=0; i < nhlayer + 1; i++)  {
      for (j=0; j < (nunit[i] + 1) * nunit[i + 1]; j++)  {
        fscanf(fp2, "%f", (wtptr[i]+j));
      }
    }

    if ((c = fclose(fp2)) != 0)
      printf("\n File %sf cannot be closed; error %d ",
             wt_file_name, c);

}

/* create file for net architecture and learning
   parameters.  File name has suffix _v.dat */

void dwrite(char *taskname)
{
    int i,j,c;
    char var_file_name[20];

    strcpy(var_file_name, taskname);
    strcat(var_file_name, "_v.dat");
    if ((fp1 = fopen(var_file_name, "w+")) == NULL)
    {
       perror(" Cannot open data file ");
      exit(0);
    }
    fprintf( fp1, "%u %u %u %f %f %u %u\n", ninput, noutattr,
        ninattr, eta, alpha, nhlayer, cnt_num);

    for (i=0; i < nhlayer + 2; i++)  {
      fprintf(fp1, "%d ", nunit[i]);
    }

    fprintf(fp1, "\n%d %f\n", cnt, err_curr);

    for (i=0; i < ninput; i++)
    {
      for (j=0; j < noutattr; j++)
        fprintf(fp1, "%f      ", outpt[i][j]);
      fprintf(fp1, "\n");
    }
```

```
    if ((c=fclose(fp1)) != 0)
      printf("\nFile %s cannot be closed; error %d ",
           var_file_name, c);
}

/* create file for saving weights and thresholds
   learned from training.  File name has suffix
   _w.dat */

void wtwrite(char *taskname)
{
    int i,j,c,k;
    char wt_file_name[20];

    strcpy(wt_file_name, taskname);
    strcat(wt_file_name, "_w.dat");

    if ((fp2 = fopen(wt_file_name, "w+")) == NULL)
    {
       perror("\nCannot open data file ");
       exit(0);
    }

    k=0;
    for (i=0; i < nhlayer + 1; i++)
      for (j=0; j < (nunit[i] + 1) * nunit[i + 1]; j++) {
        if(k==8) {
          k=0;
          fprintf(fp2, "\n");
        }
        fprintf(fp2, "%f ", *(wtptr[i] + j));
        k++;
      }
      if ((c=fclose(fp2)) != 0)
        printf("\nFile %s cannot be closed; error %d ",
               wt_file_name, c);
}

/* bottom_up calculation of net for input
   pattern i                           */

void forward(int i)
{
    int m,n,p,offset;
```

```
    float net;

/* input level output calculation */

    for (m=0; m < ninattr; m++)
      *(outptr[0]+m) = input[i][m];

/* hidden & output layer output calculation */

    for (m=1; m < nhlayer + 2; m++) {
      for (n=0; n < nunit[m]; n++) {
        net = 0.0;
        for (p=0; p < nunit[m-1] + 1; p++) {
          offset = (nunit[m-1] + 1) * n + p;
          net += *(wtptr[m-1] + offset) *
                   (*(outptr[m-1] + p));
        }
        *(outptr[m]+n) = (2.0 / (1.0 + exp(-net)))-1.0;
      }
    }
    for (n=0; n < nunit[nhlayer + 1]; n++)
      outpt[i][n] = *(outptr[nhlayer + 1] + n);
}

/* several conditions are checked to see
   whether learning should terminate   */

int introspective(int nfrom, int nto)
{
    int i, flag;
    int kke;
   /* initscr();
    move(10,40);
    refresh();
    printf("%d",cnt); */
        /* reached max. iteration */
    if (cnt >= cnt_num) return(FEXIT);

    /* error for each pattern small enough? */
    nsnew = 0;
    flag = 1;
    for (i = nfrom; (i < nto) && (flag == 1); i++) {
      if (ep[i] <= maxep) nsnew++;
      else flag = 0;
```

```
    }
    if (flag == 1) return (SEXIT);

              /* system total error small enough? */
    if (err_curr <= maxe) return (SEXIT);
    return(CONTNE);
}

          /* threshold is treated as weight of link from
             a virtual node whose output value is unity */
int rumelhart(int from_snum, int to_snum)
{
    int i,j,k,m,n,p,offset,index;
    float out;
    char *err_file = "criter.dat";

    nsold = 0;
    cnt = 0;
    result = CONTNE;

    if (fplot10)
      if ((fp3 = fopen(err_file, "w")) == NULL)
      {
         perror( "\nCannot open error file ");
        exit(0);
      }
      do {
      err_curr = 0.0;
                     /* for each pattern */
      for (i=from_snum; i < to_snum; i++) {
         forward(i);  /* bottom_up calculation */

                     /* top_down error propagation */
                     /* output_level error */
        for (m=0; m < nunit[nhlayer + 1]; m++)  {
          out = *(outptr[nhlayer + 1] + m);
          *(errptr[nhlayer + 1] + m) = (target[i][m] - out) *
                       0.5 * (1 - out*out);
        }

                     /* hidden & input layer errors */
        for (m=nhlayer + 1; m >= 1; m--)  {
          for (n=0; n < nunit[m-1]+1; n++)  {
            *(errptr[m-1] + n) = 0.0;
```

```
          for (p=0; p < nunit[m]; p++) {
            offset = (nunit[m-1] + 1) * p + n;
            *(delw[m-1]+offset) = eta * (*(errptr[m]+p))
                    * (*(outptr[m-1] + n))
                    + alpha * (*(delw[m-1] + offset));
            *(errptr[m-1]+n) += *(errptr[m] + p)
                    * (*(wtptr[m-1] + offset));
          }
          *(errptr[m-1] + n) = *(errptr[m-1] + n) *
                  (1 - *(outptr[m-1] + n)
                    * (*(outptr[m-1] + n)));
        }
      }

                /* weight changes */
      for (m=1; m < nhlayer + 2; m++) {
        for (n=0; n < nunit[m]; n++) {
          for (p=0; p < nunit[m-1] + 1; p++) {
            offset = (nunit[m-1] + 1) * n + p;
            *(wtptr[m-1] + offset) += *(delw[m-1] + offset);
          }
        }
      }

      ep[i] = 0.0;
      for (m=0; m < nunit[nhlayer + 1]; m++) {
        ep[i] += fabs((target[i][m] -
              *(outptr[nhlayer+1] + m)));
      }
      err_curr += ep[i] * ep[i];
    }           /* normalized system error */
    err_curr = 0.5 * err_curr / ninput;

                /** save errors in file to draw the
                    system error with plot10      **/
    if (fplot10)
      fprintf(fp3, "%1d, %2.9f\n", cnt, err_curr);
        cnt++;

            /* check condition for terminating learning */
    result = introspective(from_snum, to_snum);
  } while (result == CONTNE);  /* end of long do-while */

            /* update output with changed weights */
```

```
    for (i=from_snum; i < to_snum; i++) forward(i);

/*    for (i=0; i < nhlayer + 1; i++) {
      index = 0;
      for (j=0; j < nunit[i+1]; j++)
      {
        printf("\n\nWeights between unit %d of layer %d",
                    j, j+1);
        printf(" and units of layer %d\n", i);
        for (k=0; k < nunit[i]; k++)
          printf(" %f", *(wtptr[i] + index++));
        printf("\n  Threshold of unit %d of layer %d is %f",
                      j, i+1, *(wtptr[i] + index++));
      }
    } */

/*    for (i=0; i < ninput; i++)
      for (j=0; j < noutattr; j++)
        printf("\n\n sample %d output %d = %f target %d = %f",
                   i, j, outpt[i][j],j,target[i][j]); */
    printf("\n\nTotal number of iterations is %d", cnt);
    printf("\nNormalized system error is %f\n\n\n", err_curr);
    return(result);
}

                      /* read in the input data file specified
                         by user during interactive session */
void user_session()
{
    int i,j,showdata;
    char fnam[20], dtype[20];
    FILE *fp;

    printf("\n Start of learning session");

                      /* for task with name task_name, input
                         data file of the task is automatically
                         set to be task_name.dat by program */
    printf("\n Enter the task name : ");
    scanf("%s", task_name);

    printf("\n How many features in input pattern?: ");
    scanf("%d", &ninattr);
```

```
    printf("\n How many output units?: ");
    scanf("%d", &noutattr);

    printf("\n Total number of input samples?: ");
    scanf("%d", &ninput);

    strcpy(fnam, task_name);
    strcat(fnam, ".dat");

    printf("\n Input file name is %s \n", fnam);
    if ((fp = fopen(fnam, "r")) == NULL)
    {
      printf("\nFile %s does not exist", fnam);
      exit(0);
    }

    printf("\n Do you want to look at data just read? (Y/N): " );
    scanf("%s", dtype);
    showdata = ((dtype[0] == 'y') || (dtype[0] == 'Y'));
    for (i=0; i < ninput; i++) {
      for (j=0; j < ninattr; j++) {
        fscanf(fp, "%f", &input[i][j]);
        if (showdata) printf("%f  ", input[i][j]);
      }
      for (j=0; j < noutattr; j++) {
        fscanf(fp, "%f", &target[i][j]);
        if (showdata) printf("%f\n", target[i][j]);
      }
    }
    if ((i = fclose(fp)) != 0)
    {
      printf("\nFile %s cannot be closed; error %d ", fnam, i);
      exit(0);
    }
}

                        /* main body of learning */
void learning()
{
    int result;

    user_session();
    set_up();
    init();
```

```
    do {
      initwt();
       result = rumelhart(0,ninput);
    } while (result == RESTRT);

    if (result == FEXIT)
    {
       printf("\n Max number of iterations reached, but failed");
       printf("\n to decrease system error sufficiently...\n");
    }
    dwrite(task_name);
    wtwrite(task_name);
}

                  /* main body of output generation */
void output_generation()
{
    int i,m,nsample;
    char ans[10];
    char dfile[20];
    foutt = fopen("tqnur.mat","w");
            /* If task is already in the memory, data files
               for task do not need to be read in.  But, if it
               is a new task, data files should be read in to
               reconstruct the net.   */
    printf("\nGeneration of outputs for a new pattern");
    printf("\n\t Present task name is %s", task_name);
    printf("\n\t Work on a different task? (Y or N): ");
    scanf("%s", ans);
    if ((ans[0]=='y') || (ans[0]=='Y'))
    {
      printf("\n\t Please enter the task name:  ");
      scanf("%s", task_name);
      dread(task_name);
      init();
      wtread(task_name);
    }

               /* input data for output generation
                  are created                    */
    printf("\nEnter file name for patterns to be processed: ");
    scanf("%s", dfile);
    if ((fp1=fopen(dfile, "r")) == NULL )
    {
```

```
      perror(" Cannot open dfile ");
      exit(0);
    }

    printf("\nEnter number of patterns for processing:  ");
    scanf("%d", &nsample);

    for (i=0; i < nsample; i++)
      for (m=0; m < ninattr; m++)
        fscanf(fp1, "%f", &input[i][m]);

              /* output generation calculation starts */
    for (i=0; i < nsample; i++)
    {
      forward(i);
      for (m=0; m < noutattr; m++)
        fprintf(foutt,"\n %f",
                *(outptr[nhlayer + 1] + m));
      fprintf(foutt,"");
    }
    printf("\nOutputs have been generated ");

    if ((i=fclose(fp1)) != 0)
      printf("\nFile %s cannot be closed; error %d", dfile, i);
}

/************************ MAIN *********************************/
void main()
{
    char select[20], cont[10];

    strcpy(task_name, "*********");
    do {
      printf("\n**Select L(earning) or O(utput generation)**\n");
      do {
        scanf ("%s", select);
        switch (select[0]) {
          case 'o':
          case 'O': output_generation();
                    break;
          case 'l':
          case 'L': learning();
                  break;
          default : printf("\n Please answer");
```

```
                printf(" learning or output generation ");
                break;
        }
    } while ((select[0] != 'o') && (select[0] != 'O')
        && (select[0] != 'l') && (select[0] != 'L'));
    printf("\nDo you want to continue?  ");
    scanf( "%s", cont);
  } while ((cont[0] == 'y') || (cont[0] == 'Y'));

  printf("\n\nIt is all finished. ");
  printf("\nGood bye...\n\n\n  ");
}
```

A.3 NINE-POINT PATH PROBLEM

```
C     A PROGRAM FOR SOLVING FOUR BAR MECHANISM
      DIMENSION TL1(10),TL2(10),TL3(10),TL4(10)
      DIMENSION TTH(10),TAL(10)
      OPEN(1, FILE = 'FBOUT.MAT', STATUS = 'OLD')
      OPEN(2, FILE = 'FBINP.MAT', STATUS = 'NEW')
      OPEN(3, FILE = 'GTT.DAT', STATUS = 'NEW')
      TL0 = 106.0
      DO 10 I = 1,10
      READ(1,*)TL1(I),TL2(I),TL3(I),TL4(I),TTH(I),TAL(I)
10    CONTINUE
      DO 20 I = 1,10
      DO 30 J = 1,9
      PRT = TTH(I) + 40.0*(FLOAT(J)-1.0)
      DB1 = TL0*COSD(180.0) + TL1(I)*COSD(PRT)
      DB2 = TL0*SIND(180.0) + TL1(I)*SIND(PRT)
      QL4 = SQRT(DB1**2 + DB2**2)
      OL4 = ATAN2D(DB2,DB1)
      T21 = OL4 + ACOSD((QL4**2 + TL2(I)**2 - TL3(I)**2)/
    %    (2.0*QL4*TL2(I)))
      T22 = OL4 - ACOSD((QL4**2 + TL2(I)**2 - TL3(I)**2)/
    %    (2.0*QL4*TL2(I)))
      T31 = OL4 + ACOSD((QL4**2 - TL2(I)**2 + TL3(I)**2)/
    %    (2.0*QL4*TL2(I)))
      T32 = OL4 - ACOSD((QL4**2 - TL2(I)**2 + TL3(I)**2)/
    %    (2.0*QL4*TL2(I)))
      T2 = T21 - 180.0
      PX = TL1(I)*COSD(PRT) + TL4(I)*COSD(T2 + TAL(I))
      PY = TL1(I)*SIND(PRT) + TL4(I)*SIND(T2 + TAL(I))
```

```
      WRITE(2,*)PX/100.
      WRITE(3,*)PX/100.
      WRITE(3,*)PY/100.
      WRITE(2,*)PY/100.
30    CONTINUE
      WRITE(3,*)TL1(I)/100.0
      WRITE(3,*)TL2(I)/1000.0
      WRITE(3,*)TL3(I)/1000.0
      WRITE(3,*)TL4(I)/100.0
      WRITE(3,*)TTH(I)/100.0
      WRITE(3,*)TAL(I)/100.0
20    CONTINUE
      STOP
      END
```

A.4 DESIGN OF FOUR-BAR FUNCTION GENERATOR

```
C     GENERATION OF y VALUES
C     ********************************************
C     GIVEN LINK LENGTHS, AND CHOOSING x ARBITRARILY
C     CALCULATE THETA AND THEN SOLVE A FOUR BAR MECHANISM
C     CALCULATE PHI AND THEN SOLVING FOR y
C     **************************************************
      DIMENSION X(8),TH(8),PHI(8),Y(8)
      READ(1,*)RL1,RL3,(TH(I),I=1,8)
C     **************************************************
C     SOLVE FOUR-BAR MECHANISM FOR PHI VALUES
C     **************************************************
      DO 20 I = 1,8
      BX = RL0*COSD(180.0) + RL1*COSD(180 - TH(I))
      BY = RL0*SIND(180.0) + RL1*SIND(180 - TH(I))
      BPS = SQRT(BX*BX + BY*BY)
      ABPS = ATAN2D(BY,BX) - 180.0
      YJD = ACOSD((BPS**2 + RL3**2 - RL2**2)/(2.0*BPS*RL3))
      PHI(I) = -(ABPS + YJD)
20    CONTINUE
      DO 25 I = 1,8
      Y(I) = (PHI(I) - PHI0)*(RANY/RANPHI) + Y0
25    CONTINUE
      WRITE(3,*)(Y(I),I=1,8)
1     CONTINUE
```

```
      STOP
      END

      PROGRAM ANSCHK
      DIMENSION RT3(11)
      DIMENSION X(8),TH(8),PTI(8),Y(8)
      OPEN(21 , FILE = 'LETAx.DAT', STATUS = 'OLD')
      OPEN(22, FILE = 'RT33.DAT', STATUS = 'OLD')
      OPEN(23, FILE = 'WANT.MAT', STATUS = 'NEW')
      READ(21,*)RANX,RANY
      READ(21,*)RANTH,RANPHI
      READ(21,*)TH0,PHI0,THF,PHF
      READ(21,*)X0,Y0
      READ(21,*)RL0
      DO 11 I = 1,11
      READ(22,*)RT3(I)
11    CONTINUE
      RL1 = RT3(1)
      RL2 = RT3(2)
      RL3 = RT3(3)
      DO 467 I = 1,8
      TH(I) = RT3(I+3)
467   CONTINUE
      DO 468 I = 1,8
      X(I) = (TH(I) - TH0)*(RANX/RANTH) + X0
      WRITE(23,*)X(I)
468   CONTINUE
C     ****************************************************
C     SOLVE FOUR-BAR MECHANISM FOR PHI VALUES
C     ****************************************************
      DO 469 I = 1,8
      BX = RL0*COSD(180.0) + RL1*COSD(180 - TH(I))
      BY = RL0*SIND(180.0) + RL1*SIND(180 - TH(I))
      BPS = SQRT(BX*BX + BY*BY)
      ABPS = ATAN2D(BY,BX) - 180.0
      YJD = ACOSD((BPS**2 + RL3**2 - RL2**2)/(2.0*BPS*RL3))
      PTI(I) = -(ABPS + YJD)
469   CONTINUE
      DO 474 I = 1,8
      Y(I) = (PTI(I) - PHI0)*(RANY/RANPHI) + Y0
      WRITE(23,*)Y(I)
474   CONTINUE
      STOP
```

```
      END
```

A.5 TRAJECTORY CONTROL OF A TWO-LINK MANIPULATOR

```
C     *******************************************
C     POSITION CONTROL OF A TWO-LINK MANIPULATOR
C     *******************************************
C     NOTE : ALL ANGLES IN RADIANS
C          ALL LENGTHS IN M
C     *****************************************************
      DIMENSION S(50),UVEL(50)
      DIMENSION TR1(4,4),TR2(4,4),TR3(4,4),PV(4),TR4(4,4)
      DIMENSION TR5(4,4),XY(4)
      COMMON /LINKD/ RL1,RL2,ST
      COMMON /XYC/ X(50),Y(50),I
      COMMON /XCC/ XC,YC,V_T
      COMMON /VELP/ VX(50),VY(50)
      COMMON /RMASS/ RM1,RM2
      COMMON /CONP1/ THCP1,THCP2,THDCP1,THDCP2
      COMMON /CONP2/ RKD(4),DELT
      COMMON /RMINP/
      TH_DE1(50),TH_DE2(50),THD_DE1(50),THD_DE2(50)
      COMMON /RMINQ/ TXX(2,1),TH_CP1(50),TH_CP2(50),THD_CP1(50)
      COMMON /RMINR/ THD_CP2(50)
      COMMON /VAND/ VVEL(50)
      COMMON /XCNN/ XCN(50),YCN(50)
      COMMON /VXNN/ VXN(50),VYN(50)
      COMMON /PKK/ PK1(50),PK2(50),PK3(50),PK4(50)
      OPEN(1, FILE = 'VELP.DAT', STATUS = 'OLD')
      OPEN(2, FILE = 'XY.MAT', STATUS = 'UNKNOWN')
      OPEN(9, FILE = 'CONT.PAR', STATUS = 'OLD')
      OPEN(20,FILE = 'VD.MAT', STATUS = 'UNKNOWN')
      OPEN(17,FILE = 'TH_D.MAT', STATUS = 'UNKNOWN')
      OPEN(18,FILE = 'THD_D.MAT', STATUS = 'UNKNOWN')
      OPEN(19,FILE = 'THDD_D.MAT', STATUS = 'UNKNOWN')
      OPEN(22,FILE = 'XCYC.MAT', STATUS = 'UNKNOWN')
      OPEN(45,FILE = 'VCWC.MAT', STATUS = 'UNKNOWN')
      OPEN(23,FILE = 'NORM.MAT', STATUS = 'UNKNOWN')
      OPEN(10,FILE = 'ERROR.MAT',STATUS = 'UNKNOWN')
      OPEN(16,FILE = 'GAIN.MAT', STATUS = 'UNKNOWN')
C     ***************************************************
```

```
      DO 1 I = 1,50
      WRITE(*,*)
1     CONTINUE
      WRITE(*,*)'POSITION CONTROL OF TWO LINK MANIPULATOR'
      WRITE(*,*)
C     LINK LENGTHS
      READ(1,*)RL1,RL2
C     MASS
      READ(1,*)RM1,RM2
      READ(1,*)ST
C     RADIUS
      READ(1,*)RAD
C     NO OF POINTS
      READ(1,*)TIME
      READ(1,*)NTIME
C     REQUIRED TANGENTIAL VELOCITY AND ACCELERATION FOR THE
C     PROFILE
      READ(1,*)V_T,ACC
C     COORDINATES OF THE CENTER OF THE CIRCLE
      READ(1,*)XC,YC
C     INITIAL POSITION
      READ(9,*)THCP1,THCP2
C     INITIAL JOINT VELOCITY
      READ(9,*)THDCP1,THDCP2
C     INITIAL GAIN VALUES
      READ(9,*)RKD(1),RKD(2),RKD(3),RKD(4)
C     TIME STEP
      READ(9,*)DELT
      CALL TRANS(0.785398,0.0,0.0,0.0,TR1)
     CALL TRANS(0.0,RL1,0.0,0.0,TR2)
     CALL RMATMAT(TR1,TR2,TR3,4,4,4)
      PHI = ST
      STIME = 0.01
C     GENERATING THE REQUIRED VELOCITY PROFILE AND
C     EVALUATING THE DISTANCE TRAVELLED
      DO 10 I = 1,28
      IF(I.LE.9) THEN
      TIME = STIME*I
C     ACCELERATION PROFILE
      S(I) = 0.50*ACC*(TIME)**2
      WRITE(51,*)S(I)
      VVEL(I) = SQRT(2.0*ACC*S(I))
      V_T = VVEL(I)
.     ELSE IF((I.GT.9).AND.(I.LT.20)) THEN
```

```
C     CONSTANT TANGENTIAL VELOCITY PROFILE
      RLP = S(9)
      TIME = STIME*(I-9)
      VVEL(I) = 0.15
      V_T = 0.15
      S(I) = RLP + V_T*TIME
      WRITE(51,*)S(I)
      ELSE
C     DECELERATION PROFILE
      RLP = S(19)
      TIME = STIME*(I-19)
      SSG = 0.150*TIME - 0.50*(ACC)*(TIME)**2
      S(I) = RLP + SSG
      WRITE(51,*)S(I)
      VVEL(I) = SQRT(0.150**2 - 2.0*ACC*SSG)
      V_T = VVEL(I)
      END IF
      STANG = S(I)/RAD
C     WRITE(20,*)STANG
      PHI = ST + STANG
C     EVALUATING COORDINATES OF THE TRAJECTORY IN GLOBAL
C     FRAME
      CALL TRANS(PHI,0.1,0.0,0.0,TR4)
      CALL RMATMAT(TR3,TR4,TR5,4,4,4)
      PV(1) = RAD
      PV(2) = 0.00
      PV(3) = 0.00
      PV(4) = 1.00
      CALL RMATVEC(TR5,PV,XY,4,4)
      X(I) = XY(1)
      Y(I) = XY(2)
      WRITE(2,*)X(I),Y(I)
      CALL THETA_FIND
      CALL CART_VEL
      CALL THDT_FIND
10    CONTINUE
      DO 20 I = 1,27
      CALL TRAJ_CONT
20    CONTINUE
      DO 24 I = 1,27
      WRITE(10,*)TH_CP1(I)
      WRITE(10,*)TH_CP2(I)
      WRITE(10,*)THD_CP1(I)
      WRITE(10,*)THD_CP2(I)
```

```
      WRITE(10,*)TH_DE1(I+1) - TH_CP1(I)
      WRITE(10,*)TH_DE2(I+1) - TH_CP2(I)
      WRITE(10,*)THD_DE1(I+1) - THD_CP1(I)
      WRITE(10,*)THD_DE2(I+1) - THD_CP2(I)
      WRITE(17,*)TH_CP1(I),TH_CP2(I)
      WRITE(18,*)THD_CP1(I),THD_CP2(I)
      WRITE(22,*)XCN(I+1),YCN(I+1)
      WRITE(23,*)SQRT((X(I+1) - XCN(I+1))**2+(Y(I+1) -
   %    YCN(I+1))**2)
      WRITE(45,*)VXN(I),VYN(I)
      WRITE(20,*)SQRT(VXN(I)**2+VYN(I)**2)
c     WRITE(19,*)THDD_CP1(I),THDD_CP2(I)
      WRITE(16,*)PK1(I)
      WRITE(16,*)PK2(I)
      WRITE(16,*)PK3(I)
      WRITE(16,*)PK4(I)
24    CONTINUE
      STOP
      END

      SUBROUTINE RMATMAT(A,B,C,M,N,L)
      DIMENSION A(M,L),B(L,N),C(M,N)
      DO 1300 I = 1,M
      DO 1300 J = 1,N
      C(I,J) = 0.00
      DO 1300 K = 1,L
      C(I,J) = C(I,J) + A(I,K)*B(K,J)
1300  CONTINUE
      RETURN
      END

      SUBROUTINE THETA_FIND
C     ***********************************************
C     STEP 2: FIND THE JOINT COORDINATES OF THE CIRCULAR
C           TRAJECTORY
C     ***********************************************
      DIMENSION THET(4)
      COMMON /LINKD/ RL1,RL2,ST
      COMMON /XYC/ X(50),Y(50),I
      COMMON /RMINP/
   % TH_DE1(50),TH_DE2(50),THD_DE1(50),THD_DE2(50)
      OPEN(3, FILE = 'THET.MAT', STATUS = 'UNKNOWN')
C     ***********************************************
      RJJ = 2.0 * Y(I) * RL1
```

```
      RII = 2.0 * X(I) * RL1
      RKK = X(I)*X(I) + Y(I)*Y(I) + RL1*RL1 - RL2*RL2
      RTT = RJJ*RJJ + RII*RII - RKK*RKK
      THET(1) = ATAN2(RJJ,RII) + ATAN2(SQRT(RTT),RKK)
      PAR1 = Y(I) - RL1*SIN(THET(1))
      PAR2 = X(I) - RL1*COS(THET(1))
      THET(2) = ATAN2(PAR1,PAR2) - THET(1)
      WRITE(3,*)THET(1),THET(2)
      TH_DE1(I) = THET(1)
      TH_DE2(I) = THET(2)
      RETURN
      END

      SUBROUTINE CART_VEL
C     **************************************************
C     STEP 3: RESOLVE TANGENTIAL VELOCITY V_T IN THE
c           GLOBAL COORDINATE SYSTEM
C     **************************************************
      DIMENSION VP(3)
      COMMON /LINKD/ RL1,RL2,ST
      COMMON /XYC/ X(50),Y(50),I
      COMMON /XCC/ XC,YC,V_T
      COMMON /VELP/ VX(50),VY(50)
      OPEN(4, FILE = 'VEL.MAT', STATUS = 'UNKNOWN')
      OPEN(7, FILE = 'PNV.MAT', STATUS = 'UNKNOWN')
      PKL = Y(I)-YC
      PKK = X(I)-XC
      ALPH = ATAN2(PKL,PKK)
      VP(1) = V_T*COS(1.570796327+ALPH)
      VP(2) = V_T*COS(ALPH)
      VP(3) = 1.0
      WRITE(4,*)VP(1),VP(2)
      VX(I) = VP(1)
      VY(I) = VP(2)
      PNORMV = SQRT(VX(I)**2+VY(I)**2)
      WRITE(7,*)PNORMV
      RETURN
      END


      SUBROUTINE THDT_FIND
C     *******************************************************
C     STEP 4: EVALUATE THE ROTATION RATES THDT_K1 AND THDT_K2
C           USING JACOBIAN MATRIX
```

```
C     ********************************************************
      DIMENSION VPG(2),THDT(2),RINV(2,2)
      COMMON /LINKD/ RL1,RL2,ST
      COMMON /XYC/ X(50),Y(50),I
      COMMON /RMINP/
   %  TH_DE1(50),TH_DE2(50),THD_DE1(50),THD_DE2(50)
      COMMON /XCC/ XC,YC,V_T
      COMMON /VELP/ VX(50),VY(50)
      OPEN(8, FILE = 'THDT.MAT', STATUS = 'UNKNOWN')
      VPG(1) = VX(I)
      VPG(2) = VY(I)
      UT11 = -RL1*SIN(TH_DE1(I)) - RL2*SIN(TH_DE1(I)+TH_DE2(I))
      UT12 = -RL2*SIN(TH_DE1(I)+TH_DE2(I))
      UT21 = RL1*COS(TH_DE1(I)) + RL2*COS(TH_DE1(I)+TH_DE2(I))
      UT22 = RL2*COS(TH_DE1(I)+TH_DE2(I))
      DET = UT11*UT22 - UT21*UT12
      RINV(1,1) = UT22/DET
      RINV(1,2) = -UT12/DET
      RINV(2,1) = -UT21/DET
      RINV(2,2) = UT11/DET
      CALL RMATVEC(RINV,VPG,THDT,2,2)
      WRITE(8,*)THDT(1),THDT(2)
      THD_DE1(I) = THDT(1)
      THD_DE2(I) = THDT(2)
      RETURN
      END

      SUBROUTINE TRAJ_CONT
C     **************************************************
C     STEP 5: TRAJECTORY CONTROL OF TWO-LINK MANIPULATOR
C     **************************************************
      DIMENSION RX(4),XSTRT(4),RMAX(4),RMIN(4),RPHI(4)
      DIMENSION RPSI(1),RW(44),XCP(50),YCP(50)
      COMMON /LINKD/ RL1,RL2,ST
      COMMON /XYC/ X(50),Y(50),I
      COMMON /XCC/ XC,YC,V_T
      COMMON /VELP/ VX(50),VY(50)
      COMMON /RMINP/
   %        TH_DE1(50),TH_DE2(50),THD_DE1(50),THD_DE2(50)
      COMMON /RMINQ/ TXX(2,1),TH_CP1(50),TH_CP2(50),THD_CP1(50)
      COMMON /RMINR/ THD_CP2(50)
      COMMON /SEEK/ IDATA,IPRINT,NSHOT,NTEST,MAXM,F,G,TOL,
   %  ZERO,R,REDUCE
      COMMON /CONP1/ THCP1,THCP2,THDCP1,THDCP2
```

```
      COMMON /CONP2/ RKD(4),DELT
      COMMON /PKK/ PK1(50),PK2(50),PK3(50),PK4(50)
      IDATA = 0
      IPRINT = 0
      TH_CP1(1) = THCP1
      TH_CP2(1) = THCP2
      THD_CP1(1) = THDCP1
      THD_CP2(1) = THDCP2
C     ************************************************
C     FIND THE OPTIMUM VALUE OF RKD THAT GIVES MINIMUM
C     ERROR IN POSITION
C     ************************************************
C     **************************************************
C     CALLING SIDDALL LIBRARY ROUTINE
C     **************************************************
      JN = 4
      JNCONS = 0
      JNEQUS = 0
      JNPENAL = 5
      F = 0.001
      G = 0.001
      MAXM = 25000
      DATA RMAX/1.0e3,1.0e3,1.0e3,1.0e3/
      DATA RMIN/-1.0e3,-1.0e3,-1.0e3,-1.0e3/
      DO 775 KK = 1,4
      XSTRT(KK) = RKD(KK)
775   CONTINUE
      PRINT *,'ITER = ',I
      CALL SEEK(JN,JNCONS,JNEQUS,JNPENAL,RMAX,RMIN,XSTRT,RX,
    % RU,RPHI,RPSI,JNVIOL,RW)
      PK1(I) = RX(1)
      PK2(I) = RX(2)
      PK3(I) = RX(3)
      PK4(I) = RX(4)
      RETURN
      END

      SUBROUTINE UREAL(RX,RU)
      DIMENSION RX(1)
      DIMENSION QM(2,2),QV(2,2),QG(2,2),QT1(2,1),QT2(2,1)
      DIMENSION THDDTI(2,1),QMI(2,2),THDD_CP1(50)
      DIMENSION THDD_CP2(50),TORI(2)
      DIMENSION XU(2,2),TDZ(2),VPN(2)
      COMMON /LINKD/ RL1,RL2,ST
```

```
      COMMON /RMINP/
     %       TH_DE1(50),TH_DE2(50),THD_DE1(50),THD_DE2(50)
      COMMON /RMINQ/ TXX(2,1),TH_CP!(50),TH_CP2(50),THD_CP1(50)
      COMMON /RMINR/ THD_CP2(50)
      COMMON /XYC/ X(50),Y(50),I
      COMMON /RMASS/ RM1,RM2
      COMMON /CONP2/ RKD(4),DELT
      COMMON /XCC/ XC,YC,V_T
      COMMON /VELP/ VX(50),VY(50)
      COMMON /XCNN/ XCN(50),YCN(50)
      COMMON /VXNN/ VXN(50),VYN(50)
C     ********************************************
      GH = 9.81
C     ******************************************************
C     CALCULATING TORQUE BASED ON ERROR IN POSITION
C     ******************************************************
      XCN(1) = RL1*COS(TH_CP1(1)) + RL2*COS(TH_CP1(1) +
     %    TH_CP2(1))
      YCN(1) = RL1*SIN(TH_CP1(1)) + RL2*SIN(TH_CP1(1) +
     %    TH_CP2(1))

      IF(I.EQ.2) THEN
      END IF
      TORI(1) =  RX(1)*(TH_DE1(I+1) - TH_CP1(I)) +
     %           RX(3)*(THD_DE1(I+1) - THD_CP1(I))
      TORI(2) =  RX(2)*(TH_DE2(I+1) - TH_CP2(I)) +
     %           RX(4)*(THD_DE2(I+1) - THD_CP2(I))
C     ******************************************************
C     CALCULATE THDDT AT TIME T
C     ******************************************************
      CALL STHEK(TH_CP1,TH_CP2,THD_CP!,THD_CP2,
     %   THDDTI,I,TORI)
      THDD_CP1(I) = THDDTI(1,1)
      THDD_CP2(I) = THDDTI(2,1)
C     ************************************************
C     HERE WE KNOW THET, THDT AND THDDT AT TIME T
C     ************************************************
C     CALCULATE THEDDT AT TIME T+DELT
C     ************************************************
C     CALCULATE THET,THDT AT TIME T+DELT
C     ************************************************
      TORI(1) =  RX(1)*(TH_DE1(I+1) - TH_DE1(I)) +
     %           RX(3)*(THD_DE1(I+1) - THD_DE1(I))
      TORI(2) =  RX(2)*(TH_DE2(I+1) - TH_DE2(I)) +
```

```
%          RX(4)*(THD_DE2(I+1) - THD_DE2(I))
      CALL STHEK(TH_CP1,TH_CP2,THD_CP1,THD_CP2,
%       THDDTI,I+1,TORI)
      THDD_CP1(I+1) = THDDTI(1,1)
      THDD_CP2(I+1) = THDDTI(2,1)
C     ******************************************************
      ALP = 0.5
      BET = 0.001
      THD_CP1(I+1) = THD_CP1(I) + (1.0-BET)*THDD_CP1(I)*DELT +
%       BET*THDD_CP1(I+1)*DELT
      THD_CP2(I+1) = THD_CP2(I) + (1.0-BET)*THDD_CP2(I)*DELT +
%       BET*THDD_CP2(I+1)*DELT
      TH_CP1(I+1) = TH_CP1(I) + THD_CP1(I)*DELT +
%       ((0.5-ALP)*THDD_CP1(I)+ALP*THDD_CP1(I+1))*DELT**2
      TH_CP2(I+1) = TH_CP2(I) + THD_CP2(I)*DELT +
%       ((0.5-ALP)*THDD_CP2(I)+ALP*THDD_CP2(I+1))*DELT**2
C     ******************************************************
      XCN(I+1) = RL1*COS(TH_CP1(I+1)) + RL2*COS(TH_CP1(I+1) +
%       TH_CP2(I+1))
      YCN(I+1) = RL1*SIN(TH_CP1(I+1)) + RL2*SIN(TH_CP1(I+1) +
%       TH_CP2(I+1))
      TDZ(1) = THD_CP1(I+1)
      TDZ(2) = THD_CP2(I+1)
      XU(1,1) = -RL1*SIN(TH_CP1(I+1)) - RL2*SIN(TH_CP1(I+1)
%        + TH_CP2(I+1))
      XU(1,2) = -RL2*SIN(TH_CP1(I+1)+TH_CP2(I+1))
      XU(2,1) = RL1*COS(TH_CP1(I+1)) + RL2*COS(TH_CP1(I+1)
%      + TH_CP2(I+1))
      XU(2,2) = RL2*COS(TH_CP1(I+1)+TH_CP2(I+1))
      CALL RMATVEC(XU,TDZ,VPN,2,2)
      VXN(I+1) = VPN(1)
      VYN(I+1) = VPN(2)
      QW1 = ((X(I+1) - XCN(I+1))*10.0)**2
      QW2 = ((Y(I+1) - YCN(I+1))*10.0)**2
      QW3 = ((VX(I+1) - VXN(I+1)))**2
      QW4 = ((VY(I+1) - VYN(I+1)))**2
      RU = ((QW1 + QW2)*1.0 +QW3 + QW4)*1.0e6
      RETURN
      END


      SUBROUTINE STHEK(TH_CP1,TH_CP2,THD_CP1,THD_CP2,
%       THDDTI,I,TORI)
      DIMENSION QM(2,2),QV(2,2),QG(2,2),QT1(2,1),QT2(2,1)
```

```
      DIMENSION THDDTI(2,1),QMI(2,2)
      DIMENSION TH_CP1(50),THD_CP1(50)
      DIMENSION TH_CP2(50),THD_CP2(50)
      DIMENSION XU(2,2),TDZ(2),VPN(2)
      DIMENSION RX(4),TORI(2)
      COMMON /LINKD/ RL1,RL2,ST
      COMMON /RMASS/ RM1,RM2
C     ******************************************
      GH = 9.81
      QM(1,1) = RL2**2*RM2 + 2.0*RL1*RL2*RM2*COS(TH_CP2(I))
     %   + RL1**2*(RM1 + RM2)
      QM(1,2) = RL2**2*RM2 + RL1*RL2*RM2*COS(TH_CP2(I))
      QM(2,1) = RL2**2*RM2 + RL1*RL2*RM2*COS(TH_CP2(I))
      QM(2,2) = RL2**2*RM2
      QV(1,1) = -RM2*RL1*RL2*SIN(TH_CP2(I))*THD_CP2(I)**2 -
     %  2.0*RM2*RL1*RL2*SIN(TH_CP2(I))*THD_CP1(I)*THD_CP2(I)
      QV(2,1) = RM2*RL1*RL2*SIN(TH_CP2(I))*THD_CP1(I)**2
      QG(1,1) = RM2*RL2*GH*COS(TH_CP1(I)+TH_CP2(I)) +
     %  (RM1 + RM2)*RL1*GH*COS(TH_CP1(I))
      QG(2,1) = RM2*RL2*GH*COS(TH_CP1(I) + TH_CP2(I))
      CALL QMINV(QM,QMI)
      CALL RMATADD(QV,QG,QT1,2,1)
      CALL RMATSUB(TORI,QT1,QT2,2,1)
      CALL RMATVEC(QMI,QT2,THDDTI,2,2)
      RETURN
      END


      SUBROUTINE CONST(RX,JNCONS,RPHI)
      DIMENSION RX(1),RPHI(1)
      RETURN
      END

      SUBROUTINE EQUAL(RX,RPSI,JNEQUS)
      DIMENSION RX(1),RPSI(1)
      RETURN
      END


      SUBROUTINE QMINV(QM,QMI)
      DIMENSION QM(2,2),QMI(2,2)
      DO 410 I = 1,2
      DO 410 J = 1,2
      QMI(I,J) = 0.0
```

```
410   CONTINUE
      QMP = QM(1,1)*QM(2,2) - QM(1,2)*QM(2,1)
      QMI(1,1) = QM(2,2)/QMP
      QMI(1,2) = -QM(1,2)/QMP
      QMI(2,1) = -QM(2,1)/QMP
      QMI(2,2) = QM(1,1)/QMP
      RETURN
      END


C     *********************************************
C     TO NORMALIZE THE INPUT AND OUTPUT VECTOR
c     *********************************************
      DIMENSION PIN1(50),PIN2(50),PIN3(50),PIN4(50)
      DIMENSION PIN5(50),PIN6(50),PIN7(50),PIN8(50)
      DIMENSION PON1(50),PON2(50),PON3(50),PON4(50)
      OPEN(UNIT=1,FILE='ERROR.MAT',STATUS='OLD')
      OPEN(UNIT=2,FILE='GAIN.MAT',STATUS='OLD')
      OPEN(UNIT=3,FILE='INP.DAT',STATUS='UNKNOWN')
      OPEN(UNIT=4,FILE='OUT.DAT',STATUS='UNKNOWN')
      DO 10 I = 1,27
      READ(1,*)PIN1(I),PIN2(I),PIN3(I),PIN4(I)
      READ(1,*)PIN5(I),PIN6(I),PIN7(I),PIN8(I)
      READ(2,*)PON1(I),PON2(I),PON3(I),PON4(I)
10    CONTINUE
      CALL MINMAX(PIN1,SMALL1,PLARGE1,27)
      PRINT *,SMALL1,PLARGE1
      CALL MINMAX(PIN2,SMALL2,PLARGE2,27)
      PRINT *,SMALL2,PLARGE2
      CALL MINMAX(PIN3,SMALL3,PLARGE3,27)
      PRINT *,SMALL3,PLARGE3
      CALL MINMAX(PIN4,SMALL4,PLARGE4,27)
      PRINT *,SMALL4,PLARGE4
      CALL MINMAX(PIN5,SMALL5,PLARGE5,27)
      PRINT *,SMALL5,PLARGE5
      CALL MINMAX(PIN6,SMALL6,PLARGE6,27)
      PRINT *,SMALL6,PLARGE6
      CALL MINMAX(PIN7,SMALL7,PLARGE7,27)
      PRINT *,SMALL7,PLARGE7
      CALL MINMAX(PIN8,SMALL8,PLARGE8,27)
      PRINT *,SMALL8,PLARGE8
      CALL MINMAX(PON1,SMALL9,PLARGE9,27)
      PRINT *,SMALL9,PLARGE9
```

```
      CALL MINMAX(PON2,SMALL10,PLARGE10,27)
      PRINT *,SMALL10,PLARGE10
      CALL MINMAX(PON3,SMALL11,PLARGE11,27)
      PRINT *,SMALL11,PLARGE11
      CALL MINMAX(PON4,SMALL12,PLARGE12,27)
      PRINT *,SMALL12,PLARGE12
      DO 20 I = 1,27
      SDOT1 = (PIN1(I) - SMALL1)/(PLARGE1 - SMALL1)
      SDOT2 = (PIN2(I) - SMALL2)/(PLARGE2 - SMALL2)
      SDOT3 = (PIN3(I) - SMALL3)/(PLARGE3 - SMALL3)
      SDOT4 = (PIN4(I) - SMALL4)/(PLARGE4 - SMALL4)
      SDOT5 = (PIN5(I) - SMALL5)/(PLARGE5 - SMALL5)
      SDOT6 = (PIN6(I) - SMALL6)/(PLARGE6 - SMALL6)
      SDOT7 = (PIN7(I) - SMALL7)/(PLARGE7 - SMALL7)
      SDOT8 = (PIN8(I) - SMALL8)/(PLARGE8 - SMALL8)
      SDOT9 = (PON1(I) - SMALL9)/(PLARGE9 - SMALL9)
      SDOT10 = (PON2(I) - SMALL10)/(PLARGE10 - SMALL10)
      SDOT11 = (PON3(I) - SMALL11)/(PLARGE11 - SMALL11)
      SDOT12 = (PON4(I) - SMALL12)/(PLARGE12 - SMALL12)
      WRITE(3,*)SDOT1
      WRITE(3,*)SDOT2
      WRITE(3,*)SDOT3
      WRITE(3,*)SDOT4
      WRITE(3,*)SDOT5
      WRITE(3,*)SDOT6
      WRITE(3,*)SDOT7
      WRITE(3,*)SDOT8
      WRITE(4,*)SDOT9
      WRITE(4,*)SDOT10
      WRITE(4,*)SDOT11
      WRITE(4,*)SDOT12
20    CONTINUE
      STOP
      END

C     A LP APPROACH FOR NEURAL NETWORKS
C     ************************************************************
C     YW = N + 4*(N + NCONS + NEQUS)
C      WHERE N     = NUMBER OF DESIGN VARAIBLES = 1
C           NCONS = NUMBER OF INEQUALITY CONSTRAINTS = 1
C           NEQUS = NUMBER OF EQUALITY CONSTRAINTS = 1
C     YW = 1 + 4*(1+1+1) = 13
C                L.P.
C     ZW = M*(5+M)
```

```
C        - WHERE M IS NUMBER OF CONSTRAINING EQUATIONS = ROWS
C                   WHICH IS DEFINED BELOW
C     ******************************************************
C                 ZW = 36*(5+36) = 1476
C     ******************************************************
C           DIMENSIONS FOR [A] MATRIX
C           ROWS =  (NO.OF OUTPUT)*TOTAL.NO.OF TRIALS
C                =  4 * (9) = 36
C           COLUMNS = (NO.OF INPUT*2 + 4)*NO.OF OUTPUT
C                   = (8*2 + 4)*4 = 80
C     ******************************************************
      DIMENSION YX(1),YXSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1)
      DIMENSION YW(50),ZX(80),ZA(36,80),ZB(36),ZC(80)
      DIMENSION ZAP(36,80),ZCP(80),ZBP(36),ZW(1476)
      DIMENSION BBW(4,8),BBC(4),BBD(4),ZTOB(8)
      COMMON
     % /SEEK/IDATA,IPRINT,NSHOT,NTEST,MAXM,F,G,TOL,ZERO,
     %  R,REDUCE
      COMMON /BL1/ZC,ZX,ZW,ZAP,ZCP,ZBP
      COMMON /BL3/ZB
      COMMON /BL2/MZ,NZ,NUTS,NT,NI,NOUT,NOI
      COMMON /FAL/ZTOB
      COMMON /ANT/ZA
      OPEN(2, FILE = 'SCOUT-T.DAT', STATUS = 'OLD')
      OPEN(3, FILE = 'CLPTA.OUT', STATUS = 'OLD')
      OPEN(14, FILE = 'LIN.DAT', STATUS = 'OLD')
      READ(14,*)MZ
      READ(14,*)NZ
      READ(14,*)NUTS
      READ(14,*)NT,NI
      READ(14,*)NOUT,NOI
      MAXM = 10000
      CALL ALPG
1     WRITE(*,*)
      DO 196 I = 1,MZ
      READ(2,*)ZB(I)
196   CONTINUE
      READ(3,*)(ZC(I),I = 1,NZ)
      F = 0.001
      G = 0.01
      NY = 1
      NCONS = 0
      NEQUS = 0
      NPENAL = 3
```

```
      IDATA = 0
      DATA RMAX/0.10/
      DATA RMIN/-0.10/
      DATA YXSTRT/0.0013427/
      NOISE = 1
      CALL
    % SEEK(NY,NCONS,NEQUS,NPENAL,RMAX,RMIN,YXSTRT,YX,YU,PHI,
    %  PSI,NVIOL,YW)
      CALL ANSWER(YU,YX,PHI,PSI,NY,NCONS,NEQUS)
      WRITE(*,*)'DO YOU WISH TO CONTINUE ?'
      WRITE(*,*)'1:  YES '
      WRITE(*,*)'2: NO '
      READ(*,*)NDEC
      IF(NDEC.EQ.1) THEN
      GO TO 1
      ELSE
      END IF
      STOP
      END

      SUBROUTINE UREAL(YX,YU)
      DIMENSION YX(1),YXSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1)
      DIMENSION YW(50),ZX(80),ZA(36,80),ZB(36),ZC(80)
      DIMENSION ZAP(36,80),ZBP(36),ZCP(80)
      DIMENSION ZW(1476),ZTOB(8)
      DIMENSION BBW(4,8),BBC(4),BBD(4),D(2)
      DIMENSION SAK(4),RT3(4),ZZTOB(4)
      COMMON /BL1/ZC,ZX,ZW,ZAP,ZCP,ZBP
      COMMON /BL3/ZB
      COMMON /BL2/MZ,NZ,NUTS,NT,NI,NOUT,NOI
      COMMON /SIMPLE/NSTOP,IDATA,NNDEX
      COMMON /PAKS/BBW,BBC,BBD
      COMMON /FAL/ZTOB
      COMMON /ANT/ZA
      COMMON /DEPUT/D,RMX,RMN
      OPEN(15, FILE = 'SCINP-N.DAT', STATUS = 'OLD')
      OPEN(9, FILE =  'SCOUT-N.DAT', STATUS = 'OLD')
      OPEN(19, FILE = 'PERC.MAT', STATUS = 'NEW')
      OPEN(18, FILE = 'OBJ.MAT', STATUS = 'NEW')
      IDATA = 0
      NSTOP = 3000
      NNDEX = 1
      DO 20 I = 1,MZ
      KR = 1
```

```
      DO 30 J = 1,NZ
      IF(J.GT.(20*KR)) THEN
      KR = KR + 1
      END IF
      IF((J.GE.(1+20*(KR-1))).AND.(J.LT.(17+20*(KR-1)))) THEN
      ZAP(I,J) = YX(1)*ZA(I,J)
      ELSE
      ZAP(I,J) = ZA(I,J)
      END IF
30    CONTINUE
20    CONTINUE
      SQM = 0.0
      DO 40 I = 1,MZ
      ZBP(I) = YX(1)*ZB(I)
      SQM = SQM + ZB(I)
40    CONTINUE
      KR = 1
      DO 50 J = 1,NZ
      IF(J.GT.(20*KR)) THEN
      KR = KR + 1
      END IF
      IF((J.GE.(1 + 20*(KR-1))).AND.(J.LT.(17+20*(KR-1)))) THEN
      ZCP(J) = YX(1)*ZC(J)
      ELSE
      ZCP(J) = ZC(J)
      END IF
50    CONTINUE
      CALL SIMPLE(NZ,MZ,ZAP,ZBP,ZCP,ZX,ZU,ZW)
      XX = 0.0
      DO 501 J = 1,NZ
      XX = XX + ZCP(J)*ZX(J)
501   CONTINUE
      YYU = SQM - XX
      WRITE(*,*)'L.P. OBJECTIVE FUNCTION IS',YYU
c     CALL WASS(ZX,BBW,BBC,BBD)
      YT1 = 0.0
      YT2 = 0.0
      YT3 = 0.0
      DO 55 NM = 1,16,2
      DO 60 MM = NM,80,20
      YT1 = YX(1)*(ZC(NM)*ZX(MM)) + YT1
60    CONTINUE
55    CONTINUE
      DO 65 NM = 2,16,2
```

```
8       DO 70 MM = NM,80,20
        YT2 = YX(1)*(ZC(NM)*ZX(MM)) + YT2
70      CONTINUE
65      CONTINUE
        DO 75 NM = 17,20
        DO 80 MM = NM,80,20
        YT3 = (ZC(NM)*ZX(MM)) + YT3
80      CONTINUE
75      CONTINUE
        YT4 = YX(1)*SQM
C       CALL RMIXD(BBW,BBC,BBD,YX,ZTOB,RT3)
        YU = (YT1 + YT2 + YT3 - YT4)**4
        WRITE(*,*)'SLOPE IS',YX(1)
        WRITE(*,*)'SEEK OBJECTIVE FUNCTION IS',YU
        IF (NSW.EQ.1) THEN
        STOP
        END IF
        IF(YU.LT.(6.0E-6)) THEN
        DO 881 ISK = 1,9
        DO 884 I = 1,8
        READ(15,*)ZTOB(I)
884     CONTINUE
        DO 886 I = 1,4
        READ(9,*)ZZTOB(I)
886     CONTINUE
        CALL WASS(ZX,BBW,BBC,BBD)
        CALL RMIXD(BBW,BBC,BBD,YX,ZTOB,RT3)
        DO 923 JK = 1,4
        WRITE(19,*)ZZTOB(JK),RT3(JK)
        WRITE(18,*)RT3(JK)
923     CONTINUE
881     CONTINUE
        NSW = 1
        ELSE
        END IF
        RETURN
        END


        SUBROUTINE CONST(YX,NCONS,PHI)
        DIMENSION PHI(1),D(2),YX(1)
        COMMON /DEPUT/D,RMX,RMN
        NOISE = NOISE + 1
        RETURN
```

```
      END

      SUBROUTINE EQUAL(YX,PSI,NEQUS)
      DIMENSION YX(1),PSI(1)
      RETURN
      END

      SUBROUTINE RMATMUL(A,B,C,M,N)
      DIMENSION A(M,N),B(N),C(M)
      DO 4430 I = 1,M
      C(I) = 0.0
      DO 4440 J = 1,N
      C(I) = C(I) + A(I,J)*B(J)
4440  CONTINUE
4430  CONTINUE
      RETURN
      END


      SUBROUTINE ALPG
C     GENERATION OF [A] MATRIX FOR LINEAR PROGRAMMING
      DIMENSION X(8),ZA(36,80)
      COMMON /ANT/ZA
      OPEN(17, FILE = 'SCINP-T.DAT', STATUS = 'OLD')
      DO 1050 NI = 1,20,4
      KNI = NI
      NS = 1
      NT = 16
      DO 714 I = 1,8
      READ(17,*)X(I)
714   CONTINUE
780   CONTINUE
      IF(NT.GT.80) GOTO 7100
      MT = 1
      DO 710 I = NS,NT,2
      ZA(KNI,I) = X(MT)
      MT = MT + 1
710   CONTINUE
      MT = 1
      DO 720 I = NS+1,NT,2
      ZA(KNI,I) = -X(MT)
      MT = MT + 1
720   CONTINUE
      DO 730 I = NT+1,NT+4
```

```
      ZA(KNI,I) = 1.0
730   CONTINUE
      DO 740 I = NT+2,NT+3
      ZA(KNI,I) = -1.0
740   CONTINUE
      KNI = KNI + 1
      NS = NS + 20
      NT = NT + 20
      GO TO 780
7100  CONTINUE
1050  CONTINUE
      RETURN
      END

C     **************************************************
C     TO SCALE BACK THE GAIN AND ERROR VALUES
C     OBTAINED FROM THE NEURAL NETWORK
C     **************************************************
      DIMENSION GNR1(50),GNR2(50),GNR3(50),GNR4(50)
      DIMENSION PON1(50),PON2(50),PON3(50),PON4(50)
      OPEN(UNIT=1, FILE='GAIN.MAT', STATUS='OLD')
      OPEN(UNIT=2, FILE='SCKF-N.DAT', STATUS = 'OLD')
      OPEN(UNIT=3, FILE='KFRM-N.DAT', STATUS = 'UNKNOWN')
      DO 10 I = 1,27
      READ(1,*)PON1(I),PON2(I),PON3(I),PON4(I)
      READ(2,*)GNR1(I)
      READ(2,*)GNR2(I)
      READ(2,*)GNR3(I)
      READ(2,*)GNR4(I)
10    CONTINUE
      CALL MINMAX(PON1,SMALL9,PLARGE9,27)
      PRINT *,SMALL9,PLARGE9
      CALL MINMAX(PON2,SMALL10,PLARGE10,27)
      PRINT *,SMALL10,PLARGE10
      CALL MINMAX(PON3,SMALL11,PLARGE11,27)
      PRINT *,SMALL11,PLARGE11
      CALL MINMAX(PON4,SMALL12,PLARGE12,27)
      PRINT *,SMALL12,PLARGE12
      DO 20 I = 1,27
      GAIN1 = GNR1(I)*(PLARGE9-SMALL9) + SMALL9
      GAIN2 = GNR2(I)*(PLARGE10-SMALL10) + SMALL10
      GAIN3 = GNR3(I)*(PLARGE11-SMALL11) + SMALL11
      GAIN4 = GNR4(I)*(PLARGE12-SMALL12) + SMALL12
      WRITE(3,*)GAIN1,GAIN2,GAIN3.GAIN4
```

```
20    CONTINUE
      STOP
      END

C     ******************************************************
C     MATLAB FILES TO CHECK FOR THE TRAJECTORY USING
C     THE GAIN VALUES OBTAINED FROM LP-NEURO METHOD
c     AND AS WELL AS FROM OPTIMIZATION ROUTINE
C     ******************************************************
C                    final.m
C     ******************************************************
      clear
      load thet.dat
      load thdt.dat
      load gain.dat
      load xy.dat
      load xcyc.dat
      l1 = 0.3;
      l2 = 0.2;
      m1 = 4.0;
      m2 = 3.0;
      g = 9.81;
      for i = 1:27
      t1 = thet(i,1);
      t11 = thet(i+1,1);
      t2 = thet(i,2);
      t22 = thet(i+1,2);
      td1 = thdt(i,1);
      td11 = thdt(i+1,1);
      td2 = thdt(i,2);
      td22 = thdt(i+1,2);
      m(1,1) = l2^2*m2 + 2*l1*l2*m2*cos(t2) + l1^2*(m1+
      m(1,2) = l2^2*m2 + l1*l2*m2*cos(t2);
      m(2,1) = l2^2*m2 + l1*l2*m2*cos(t2);
      m(2,2) = l2^2*m2;
      v(1,1) = -m2*l1*l2*sin(t2)*td2^2 - 2*m2*l1*l2*sin(t2)*td1*td2;
      v(2,1) = m2*l1*l2*sin(t2)*td1^2;
      gg(1,1) = m2*l2*g*cos(t1+t2) + (m1+m2)*l1*g*cos(t1);
      gg(2,1) = m2*l2*g*cos(t1+t2) ;
      e1 = t11 - t1;
      e2 = td11 - td1;
      e3 = t22 - t2;
      e4 = td22 - td2;
      K1 = gain(i,1);
```

```
K2 = gain(i,2);
K3 = gain(i,3);
K4 = gain(i,4);
to(1,1) = K1*e1 + K3*e2;
to(2,1) = K2*e3 + K4*e4;
ja = inv(m)*(to - v - gg);
al = 0.5;
be = 0.001;
delt = 0.001;
fr
jv(1,1) = td1 + (1-be)*ja(1,1)*delt + be*jan(1,1)*delt ;
jv(2,1) = td2 + (1-be)*ja(2,1)*delt + be*jan(2,1)*delt;
jd(1,1) = t1 + td1*delt + ((0.5-al)*ja(1,1) + al*jan(1,1))*delt^2;
jd(2,1) = t2 + td2*delt + ((0.5-al)*ja(2,1) + al*jan(2,1))*delt^2;
thet(i+1,1) = jd(1,1);
thet(i+1,2) = jd(2,1);
%thdt(i+1,1) = jv(1,1);
%thdt(i+1,2) = jv(2,1);
x(i,1) = l1*cos(jd(1,1)) + l2*cos(jd(1,1) + jd(2,1));
y(i,1) = l1*sin(jd(1,1)) + l2*sin(jd(1,1) + jd(2,1));
jcb(1,1) = -l1*sin(jd(1,1)) - l2*sin(jd(1,1)+jd(2,1));
jcb(1,2) = -l2*sin(jd(1,1) + jd(2,1));
jcb(2,1) = l1*cos(jd(1,1)) + l2*cos(jd(1,1) +jd(2,1));
jcb(2,2) = l2*cos(jd(1,1) + jd(2,1));
vx(i,1) = jcb(1,1) * jv(1,1) + jcb(1,2)*jv(2,1);
vy(i,1) = jcb(2,1) * jv(1,1) + jcb(2,2)*jv(2,1);
vd(i,1) = sqrt(vx(i,1)^2 + vy(i,1)^2);
%ex(i,1) = xy(i+1,1) - xcyc(i,1);
%ey(i,1) = xy(i+1,2) - xcyc(i,2);
%nex(i,1) = xy(i+1,1) - x(i,1);
%ney(i,1) = xy(i+1,2) - y(i,1);
%er_op(i,1) = sqrt(ex(i,1)^2 + ey(i,1)^2);
%er_lp(i,1) = sqrt(nex(i,1)^2 + ney(i,1)^2);
end
```

```
c    *****************************************************
c                        fr.m
c    *****************************************************
load pthet.dat
load pthdt.dat
load gain.dat
pt1 = pthet(i,1);
pt11 = pthet(i+1,1);
```

```
pt2 = pthet(i,2);
pt22 = pthet(i+1,2);
ptd1 = pthdt(i,1);
ptd11 = pthdt(i+1,1);
ptd2 = pthdt(i,2);
ptd22 = pthdt(i+1,2);
pm(1,1) = l2^2*m2 + 2*l1*l2*m2*cos(pt2) + l1^2*(m1+m2);
pm(1,2) = l2^2*m2 + l1*l2*m2*cos(pt2);
pm(2,1) = l2^2*m2 + l1*l2*m2*cos(pt2);
pm(2,2) = l2^2*m2;
pv(1,1) = -m2*l1*l2*sin(pt2)*ptd2^2 - 2*m2*l1*l2*sin(pt2)*ptd1*ptd2;
pv(2,1) =  m2*l1*l2*sin(pt2)*ptd1^2;
pgg(1,1) = m2*l2*g*cos(pt1+pt2) + (m1+m2)*l1*g*cos(pt1);
pgg(2,1) = m2*l2*g*cos(pt1+pt2) ;
pe1 = pt11 - pt1;
pe2 = ptd11 - ptd1;
pe3 = pt22 - pt2;
pe4 = ptd22 - ptd2;
pK1 = gain(i,1);
pK2 = gain(i,2);
pK3 = gain(i,3);
pK4 = gain(i,4);
pto(1,1) = pK1*pe1 + pK3*pe2;
pto(2,1) = pK2*pe3 + pK4*pe4;
jan = inv(pm)*(pto - pv - pgg);
```