ARTIFICIAL NEURAL NETWORKS FOR CONSTRAINED AND UNCONSTRAINED OPTIMIZATION

By

Jiahan Chen

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1992

ABSTRACT

ARTIFICIAL NEURAL NETWORKS FOR CONSTRAINED AND UNCONSTRAINED OPTIMIZATION

By

Jiahan Chen

Finding an accurate solution to linear and nonlinear programming problem formulations is a challenge. Fast, accurate and cost-effective methods for solving simultaneous linear and nonlinear equations, for example those found in large-scale power system control problems, have been sought for decades. In this dissertation, new approaches for solving these problems using artificial neural networks (ANNs) are presented. Foremost, the reason for degenerating accuracy of previously developed ANNs for constrained optimization problems is discovered, and a new combination penalty function for a neural network is proposed which can ensure that an equilibrium point is sufficiently close to the optimal point. Then, an ANN for unconstrained optimization is proposeed which leads to development of linear and nonlinear equation solvers. Correspondingly, network formulations for the solvers are defined, and conditions for network stability and convergence are identified and proven. An architecture design for the solvers, in which a network is constructed from a neuron array and a resistance connection matrix, is described and the hardware complexity is given. Furthermore, the linear and nonlinear equation solvers are applied to solving power load flow and contingency analysis problems. The related issues in these applications, such as formulation modification and a relationship between the transmission line

parameters in a power system and the stability property of the neural network, are analyzed. In addition, other applications of the linear equation solver are discussed. These include matrix inversion, determining the stability of a linear control system, and determining matrix singularity. Simulation experiments show encouraging results for test examples with linear and nonlinear programming, linear system problems, and power system control problems. Moreover, it is shown that the time complexity of the linear and nonlinear equation solvers is problem size independent so that real-time computing for large-scale power systems will become possible when the approaches are implemented by VLSI technology.

ACKNOWLEDGMENTS

It is my great pleasure to thank my major advisor, Dr. Michael A. Shanblatt, for his guidance and encouragement throughout the years of my graduate studies. I especially appreciate his constructive critiques and intellectually challenging questions that initiated significant research directions.

I also want to thank my other Ph.D. Committee memberers, Dr. P. David Fisher, Dr. Robert A. Schlueter, Dr. Gerald L. Park, and Dr. Joseph Gardiner, for their valuable comments and suggestions on this work.

Furthermore, I wish to appreciate the financial support from the Center for Remote Sensing and the Case Center, Michigan State University, which has lasted for more than four years.

Finally, I wish to dedicate this dissertation to my parents for getting me started right and encouraging me to pursue higher education, my wife, Zhenying Shen, for her love, patience, understanding, and support, and my lovely daughters, Hallie and Quanda.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Since the late 1980s, Artificial Neural Networks (ANNs) have captivated the interest of researchers. The use of ANNs introduces fundamentally new concepts into machine intelligence computing problems. ANNs have been successfully applied in areas such as pattern recognition, associative memory, adaptive control, intelligent robotics, and optimization. This dissertation presents a new approach for improving the performance of artificial neural networks for linear and nonlinear programming, and develops a neural network-based linear and nonlinear equation solver. This includes the theoretical analysis, network formulation, architecture configuration, and approach verification. This chapter begins with an overview of neural computing, followed by the problems to be solved and research tasks of this work.

1.1 Overview

The birth of the electronic digital computer, the Von Neumann machine, marked a new era of information processing in human civilization. Since then, computers have developed from the first generation to the fifth generation, as devices, architectures and computing mechanisms have advanced. From the electronic tube, to the transistor, to the integrated circuit, to VLSI chips, the devices have become smaller, faster, and

cheaper. A variety of architectures such as SISD, SIMD, MISD, and MIMD have been used in the Von Neumann-based paradigm. Due to the control-driven property, it is difficulty to obtain a higher throughput in Von Neumann computers. Consequently, the dataflow and reduction computers have been proposed [Dec89]. The former is data-driven, that is, program instructions are executing as soon as their operands are ready, and the later is demand-driven, that is, program instructions are carrying out when results are needed for other calculations. All of these computers are called conventional computers because their computing is based on the algorithm-oriented processes. Of course, these computers have played very important roles in scientific, industrial, and commercial sectors, because they are much superior to the human brain in speed and accuracy for conventional computation tasks. They, however, are far inferior to the human brain in *intelligence, perception, fault tolerance, and intolerance to fuzzy input.*

A nascent research area — the sixth generation, brain-like computers — has emerged, which requires cooperation among the following scientific disciplines: biology, psychology, mathematics, and computer science and engineering. The research area is divided into three aspects: *Science, Technology and Applications* [SoSo88]. Scientists are exploring intrinsic principles such as perception, cognition, and intelligence. Technologists are developing new techniques for adaptive learning, pattern recognition and decision making. At the same time, they are envisioning and implementing new hardware architectures, new processing algorithms, and new silicon devices. Application engineers will enhance information processing capabilities dramatically in some areas, such as pattern recognition, computer vision, robotics, and optimization. In addition, they will open new ways to developing much more intelligent machines. Consequently, tremendous changes will take place in the future. However, it is impossible for the neural computers to replace the the conventional computers. Because it has been proven that the conventional computers are more suitable for tasks such as logical reason, sequential mathematical operations, planning, and language

understanding. In other words, both the neural computers and the conventional computers will be developed harmoniously. Furthermore, a hybrid computer may be developed in the future which associates the real-time computing property of the neural computer with the features of the conventional computer, such as general purpose and program flexibility, so that an excellent overall performance can be achieved. Figure 1.1 shows a classification of computer architectures.

With the distinct features of asynchronous parallel processing, continuous-time dynamics, and global interaction of network elements, artificial neural networks create an opportunity for enhancing real-time control and decision-making processes. A major candidate in these processes is large-scale power system control problems because they are complex and time consuming. In this dissertation, a set of fundamental research tasks aimed toward developing artificial neural network techniques for power system control will be performed. The results of this research will lay the fundamental mathematical and analytical groundwork to take advantage of neural computational concepts in this application area. This will be accomplished by a series of research tasks which will lead to implementing the algorithms and architectures from this research in VLSI. This may ultimately be the key to providing a robust computing environment for enhanced on-line control and decision-making, a development that would greatly improve the security of electric power systems.

1.2 Problem Statement

Artificial neural networks, especially feedback networks, have been used to solve optimization problems. However, the computing accuracy from the present models cannot meet the demand for some applications. For example, the relative error is *1-3%* for linear and nonlinear programming [ChLi84, KeCh88, Roe*t*90]. The problem here is to find the reason for degenerating computing accuracy and a method for improving

accuracy.

Solving linear equations is a fundamental computation task. The time complexity of all the present methods is size-dependent: $O(n^3)$ for the software approach, $O(n)$ for the hardware approach [HwCh82, Pr*et*86, CoRo87, JoJe88, JeJo90]. As the scale of the system increases, the computing efficiency becomes a critical issue. In order to overcome the problem of the size-independence, a new approach should be developed.

On-line control of large-scale power systems has always been weakened by an inability to solve large-scale complex power load flow and contingency analysis problems in time frames required for real-time response to rapidly changing operating conditions [BuHW82, CoDK86, MoPG87, BaEM89]. The problem has been aggravated by the lack of robust algorithms and implementation technology which would allow for the design of economically feasible dedicated high-speed computational hardware architecture.

Artificial neural networks have shown great potential for solving complex computation problems in real-time. A lot of progress has been made in ANN theory, architectures, and applications [TuHW86, AtSu89, Ag*et*89, WiLe90, Fi*et*89, SoPa89, MaSh90, TaTr91, TrWa91, WeEl91]. Therefore, a significant opportunity exists for applying the ANN technology to the above problems. By utilizing these computational advances and linking them to power system analysis, the research work will focus on:

(1) analyzing artificial neural networks for linear and nonlinear programming from the viewpoint of optimization theory;

(2) developing an ANN solver for general linear equations;

(3) developing ANN techniques for the power contingency analysis;

(4) exploring a method for solving nonlinear equations by artificial neural networks and applying it to solve the load flow problem.

1.3 Research Tasks

The following research plan is used to achieve the goals of this research in a stepwise and overlapping fashion and to set the stage for subsequent developmental research further exploiting the expected results.

Task 1: *ANN Performance Improvement*

Objective: Analyze various network formulations of feedback models from the viewpoint of optimization theory in order to discover the reason for degenerating computing accuracy.

Significance: This analysis will generate insights which will be useful in developing a new method for improving performance of ANN for optimization problems.

Approach: ANN feedback models for optimization will be investigated first. The role of the feasible region boundary for linear and nonlinear programming with constraints will be identified. In reviewing general optimization techniques, the penalty function method will be analyzed, particularly its behavior in a neighborhood range of the boundary conditions. Based on these analyses, a new combination penalty function will be proposed which exhibits good behavior in both the boundary neighborhood and far away regions. This function must also be easily implemented with physical circuits. Then, one ANN model, the Kennedy-Chua network, for example, will be chosen to test the new penalty function with a modified circuit scheme for the constraint amplifier unit, and to check the computing accuracy for linear and nonlinear programming.

Task 2: *ANN Solver for Linear Equations*

Objective: Establish a relation between the quadratic optimization and the linear equations solution, and map the latter into an artificial neural network structure.

Significance: This step in this research will lay a theoretical foundation in linking artificial neural networks and linear equations solution. In consequence, a new application area of ANN, a completely parallel equation solver, will be explored.

Approach: Quadratic programming without constraints and its application will be investigated. In fact, there exists a relation between the optimal point of quadratic function and the solution of linear equations. The idea here is to map the general linear equations ($A\mathbf{x} = \mathbf{b}$) into an ANN model. The primary theories used in deriving the formulation will include matrix theory, differential equation theory, and optimization theory [BaSt70, HiSm74, BaSh79, Cro80, HiLi86, Ort87b]. It is essential to guarantee stability and convergence of an ANN model. The stability of the quadratic optimization network depends on the coefficient matrix A, particularly on the eigenvalues of A. Based on this initial understanding, the problem can be divided into three cases:

(1) **A** is a symmetric positive definite matrix. The optimization network is equivalent to linear equations solution in a sense of equilibrium.

(2) **A** is an arbitrary nonsingular matrix. It is necessary to find an equivalent transformation L such that $L(A)$ is a symmetric positive definite matrix. Case 2, therefore, will be turned into case 1.

(3) **A** is a symmetric or asymmetric matrix with positive real part of eigenvalues. There exists an unique stationary point of the optimization network which is satisfied with the linear equations.

Correspondingly, a hardware configuration of ANN solver will be sought in a modular fashion.

Task 3: *Steady-State Contingency and Power Load Flow Formulations*

Objective: Use the models and mappings to express contingency analysis and power load flow as corresponding artificial neural network formulations.

Significance: This task will provide a case study verification of the models and mappings. Furthermore, the results will indicate the capability for getting a feasible solution to power system control problems by ANN.

Approach: The results from Task 1 and Task 2 will be applied in the contingency analysis and power load flow problems. Some approximation techniques will be used for simplifying formulations. A mapping relationship between the optimization networks and the nonlinear equations solution will be sought. An emphasis will be put on how to get a feasible solution for power load flow from the viewpoint of the security. In order to work more efficiently, the formulations will be divided into three steps: (1) the DC contingency, (2) the AC contingency and decoupled load flow, and (3) the full load flow.

Task 4: *Simulations*

Objective: Produce simulations of artificial neural network approaches to the new penalty function, the ANN solver, the contingency analysis and power load flow.

Significance: The simulations will demonstrate the degree of practicality of implementing these approaches. The comparison will show the advantages and disadvantages of the approaches proposed with respect to other approaches.

Approach: Simulation programs for all the above approaches will be developed in C language. They will be used to provide performance metrics for comparison with other approaches. The simulations will also be used to study convergence properties, computing accuracy, and network sensitivity regarding certain parameters, for example, the input coefficients for the new penalty function, the time increment for numerical integration, and the initial value for power load flow. According to the simulation results, possible modification and reformulation of the networks will be done by going back to the previous tasks for the purpose of performance improvement.

1.4 Organization of the Dissertation

The remainder of the dissertation is organized as follows. Chapter 2 contains a background of artificial neural networks. It begins with the basic concepts of neurons and neural networks, and then it briefly describes the distinct features of neural computing, followed by the discussion of learning methods and artificial neural network architectures. It ends with an outline of ANN applications and a general procedure for solving problems using the neural network technique.

Chapters 3, 4 and 5 form the backbone of the dissertation, which deal with *Research Tasks 1, 2 and 3*, respectively.

Chapter 3 presents a method for improving the computing accuracy of artificial neural networks for linear and nonlinear programming. Based on investigation of boundary situation of optimization problems with constraints, a new combination penalty function is proposed which can ensure that the equilibrium point is sufficiently close to the optimal point. A modified ANN architecture with the new penalty function circuit scheme is given.

Chapter 4 presents a new method for solving linear system problems using an artificial neural network. Based on a mapping between the solution of linear equations and quadratic minimization, solving linear equations can be viewed as finding the minimum of a quadratic function. The conditions for stability and convergence of the neural network-based dynamic system are identified and proven. A feedback ANN model with symmetric or asymmetric connections for optimization is proposed to form an ANN linear equation solver which can be implemented with VLSI technology. In addition, other applications of the approach are discussed. These include matrix inversion, determining the stability of a linear control system, and determining matrix singularity.

Chapter 5 presents a new approach for solving nonlinear equations using an artificial neural network technique. From an engineering point of view, a formulation of an ANN nonlinear equation solver is proposed which can significantly simplify hardware architecture. The conditions for stability and convergence of neural network-based on the formulation are proven. Furthermore, ANN formulations for the full power load flow, the decoupled load flow, and the DC load flow are given, and corresponding architectures for them are proposed, which can be implemented with VLSI technology.

Finally, Chapter 6 summarizes this work, identifies the major contributions, and points out the future research.

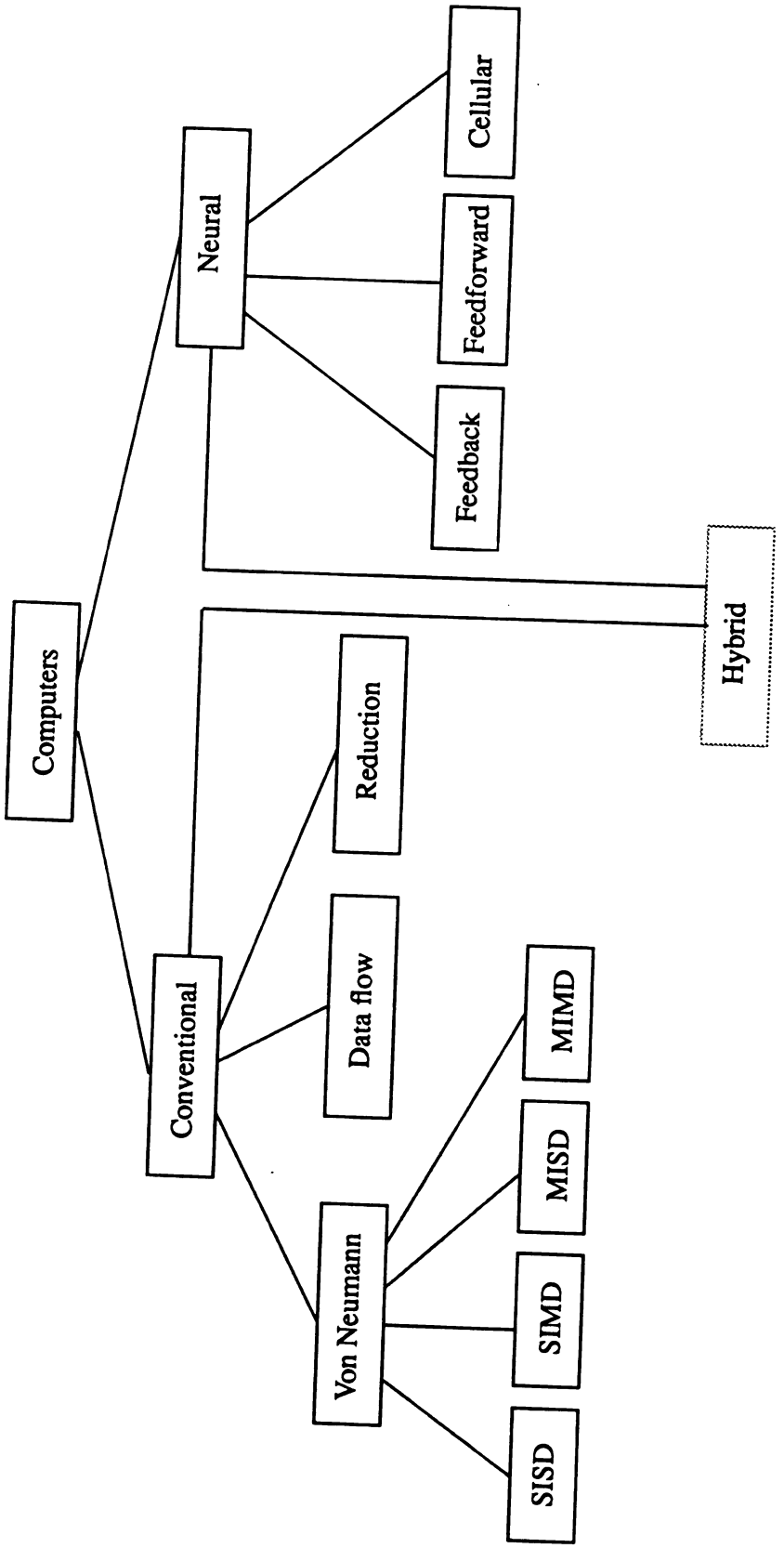


Figure 1.1. A classification of computer architectures.

Chapter 2

Artificial Neural Networks

In this Chapter, a brief review of the development of artificial neural networks is given. Then, the basic concepts, features, learning algorithms, and architectures of neural networks are described. Finally, some issues of ANN applications are discussed.

2.1 Background

The notation and concept of neural networks began in the 1940s with a linking between biology and mathematics. By the late 1950s, the first neural network simulations emerged [WiHo60]. After that, some learning rules, system control theories, and algorithms for updating connection weights were released. Neural network research, however, suddenly declined in the late 1960s due to a book titled *Perceptrons* written by Marvin Minsky and Seymour Papert [MiPa69]. These leading artificial intelligence scientists favored the Von Neumann machine and over-criticized the immature neural networks' shortcomings. The research was not revived until 1982 when John Hopfield presented a paper to the National Academy of Science [Hop82]. Hopfield's theory and experiment re-convinced contemporary scientists of the benefits and feasibility of ANNs. A great return to neural network research followed. Since then, many

researchers have been involved in biological neuron modeling, neural dynamics, learning algorithms, neural architectures, applications and their implementations [RuHW86, Mea88, MeIs89, GrKu89]. The following sections summarize the basic issues of neural network technology.

2.1.1 Basic concepts

A neuron model is an abstract description of a neural cell. There are two major categories of models [DaDe91]. The first category attempts to represent the behavior of nerve cells in the deepest mathematical details, often involving differential equations. This model is sufficient for use as a tool to validate and predicate the behavior of the human nerve system. Based on a number of simplifications, the second category of models places focus on the cause-effect and functions of nerve cells. Most neural network models, including the ones discussed below, use the simplified model rather than the detailed models.

Figure 2.1 illustrates a simplified neuron model, called a processing element. Neuron i receives a set of input signals, $(x_1, x_2, \cdots, x_n)$, each of which is multiplied by a weighting factor, w_{ki}. The combination function adds up all weighted input signals to produce the summation u_i. The transfer function maps the summation signal to an output signal v_i according to a certain relationship. The normalized range of output signals is between 0 and 1. The output signals are then transmitted to other neurons.

The weighting factors have a significant meaning for a real or artificial system. A positive weighting factor represents an excitation connection because the level of activation will increase from the positive weighted input. Conversely, a negative weighting factor represents an inhibitory connection because the level of activation will be reduced. The weighted connections play an important role in a neural network, as these connections form a weight matrix which characterizes the dynamic behavior of

the neural network to a great extent.

A variety of functions, such as linear, piece-linear, threshold, and sigmoidal, can be used to represent the transfer functions. Among them, the sigmoidal function

$$v_i = f(u_i) = \frac{1}{1 + e^{-u_i}} \tag{2.1}$$

is most commonly used for the transfer function. The reason is that the sigmoidal function has the following useful properties:

(1) The function is similar to the response function in the biological neuron.

(2) The output is bounded from both above and below;

(3) The function is continuous and continuously differentiable;

(4) The function is monotonically increasing, because

$$f'(u_i) = \frac{e^{-u_i}}{(1 + e^{-u_i})^2} > 0. \tag{2.2}$$

At the kernel of neural computing is the artificial neural network (ANN) which has been defined as: "... *massively parallel, interconnected networks of simple (usually adaptive) elements and their hierarchical organizations which are intended to interact with the objects of the real world in the same way as the biological nervous system does*" [Koh87]. It is important to point out that the ANN is not a model of the human brain which is much more complex, but a model inspired by the human brain's parallel processing aspect. Table 2.1 indicates the similarities between the human nervous system and an ANN system [KaBr89].

A threshold function is a simple transfer function. For the binary output case, as shown in Figure 2.1, if $u_i \geq T$, then $v_i = 1$, otherwise, $v_i = 0$, where T is a threshold. ANNs are often built in a multi-layer architecture (see Sections 2.2.2.2 and 2.2.2.3) so that they can perform more complex information processing. As an example, Figure 2.2 illustrates that four basic logic problems (AND, OR, XOR, XNOR) can be solved

by an identical neural network with different weights and thresholds. Table 2.2 lists their corresponding values.

2.1.2 Features

Artificial neural network technology introduces completely different concepts into computing, that is, *non-algorithmic computing, learning, distributed memory, fault tolerance, and parallel processing* [KaBr89].

2.1.2.1 Non-algorithm Computing

Conventional computing is an algorithm-oriented process in which each task is described in an exact program and is fulfilled by serial or partially parallel machine instructions. Neural computing is a dynamics-oriented process in which each task is mapped into a proper network architecture, and is fulfilled by the network's inherent convergent behavior.

2.1.2.2 Learning

Many desired applications of conventional computer technology can not be economically developed because the software needed is often too expensive to design, write, and debug. Ideal neural networks overcome this bottleneck because they do not have to be programmed for an application but theoretically learn an application by training through examples. In other words, neural networks possess artificial intelligence based on their accumulated "experience".

2.1.2.3 Distributed Memory

Conventional computer memory stores data and instructions in an address-accessed manner. Neural computing technology stores information in a distributed interconnected weight matrix.

2.1.2.4 Fault Tolerance

Conventional computer is very sensitive to its component failures, because the computer is composed of a set of different functional units without redundance or with little redundance. Neural networks can keep in operation although a few components fail to work, because they have redundant connections and form a homogeneous structure [NeSY92].

2.1.2.5 Parallel Processing

Parallel processing in conventional computers is often based on time overlapping (e.g., pipeline) and spatial duplication (e.g., vector register and systolic array) such that an instruction is divided into a series of sub-instructions which can be executed simultaneously. Neural computing technology has an inherent parallel behavior because it uses numbers of processing elements with adaptive connections such that signals can be transmitted and processed in parallel throughout the network.

2.1.3 Learning

In order to imitate human intelligence, a neural network must learn how to organize itself. This means the network will determine its connection weights by training for meeting the demands of a particular problem. Leaning has been defined as " *a change in behavior which is to a significant degree permanent in nature and which results from activity, training or observation*" [Jac88]. There are three basic learning methods: supervised learning, reinforcement learning, and unsupervised leaning. In supervised

learning, the input and the corresponding target for an example are known. The learning process adjusts connection weights within a network such that the difference between the output and the target approaches to a minimum. In reinforcement learning, the target is not given in an exact quantity, but in a "*good*" or "*bad*" fashion. In unsupervised learning, without the target description for an example, a network will develop its own classification by extracting features and using the dynamic relationship between the inputs and the connection weights.

Learning rules are exact algorithms which give an explicit description for updating connection weights in a network dynamically. Four well-known learning rules are briefly described below.

2.1.3.1 The Delta Rule

The Delta Rule is expressed in

$$\Delta W_{ij} = e * (t_j - a_j) * a_i \tag{2.3}$$

where W_{ij} is the connection weight from element i to element j, e is a learning constant, t_j is the target activation of unit j, a_j is the actual activation of unit j, and a_i is the input node's activation.

As supervised learning, the Delta rule is very efficient for two-layer networks. A well-known learning rule, *Back Propagation*, is used for three-layer and multi-layer networks. An interesting link between back propagation learning and multivariate non-linear least square estimation is established in [Ang89]. The purpose of learning is updating weight matrix according to training data such that the squared error between the desired output and the computed output is minimized. Using a normalization form, the squared error can be written as

$$D^2(S,Y) = \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{l}(S_{ji} - Y_{ji})^2 \tag{2.4}$$

$$= \frac{1}{n}\sum_{j=1}^{n}\sum_{t=1}^{o}\left[f\left[\sum_{k=1}^{h}W_{tk}^{OH}f\left[\sum_{i=1}^{I}W_{ki}^{HI}X_{ji}\right]\right] - Y_{jt}\right]^2$$

where

X_{ij} is the input value,

Y_{ij} is the desired output value,

S_{ij} is the computed output value,

I, o, h are the number of input, output, and hidden neurons, respectively,

n is the number of training exemplars,

W^{HI} is the weight matrix between the input layer and the hidden layer,

W^{OH} is the weight matrix between the hidden layer and the output layer,

f is the transfer function.

From a mathematical point of view, the problem can be viewed as solving the set of equations

$$\frac{\partial[D^2(S, Y)]}{\partial W_{ki}^{HI}} = \frac{\partial[D^2(S, Y)]}{\partial W_{tk}^{OH}} = 0. \tag{2.5}$$

Here the number of equations is p_1 $(= n \cdot o)$ and the number of unknown variables is p_2 $(= I \cdot h + h \cdot o)$. There are three cases for the general equations. If $p_1 = p_2$, the problem is determined since there may exist a unique solution. If $p_1 > p_2$, the problem is overdetermined since there may exist no solution. If $p_1 < p_2$, the problem is underdetermined since there may exist infinite solutions. For coping with all of the cases, a solution can be found in the sense of the least square estimation. Using the Newton-Raphson iteration [Sca66], the updated weight scheme can be obtained as

$$W_{ki}^{HI}(m+1) = W_{ki}^{HI}(m) - \alpha\frac{\partial[D^2(S, Y)]}{\partial W_{ki}^{HI}(m)} \tag{2.6a}$$

$$W_{tk}^{OH}(m+1) = W_{tk}^{OH}(m) - \alpha\frac{\partial[D^2(S, Y)]}{\partial W_{tk}^{OH}(m)} \tag{2.6b}$$

where m is an iteration index, and α is a learning factor ($0 < \alpha < 1$). α is based on the following assumption for the Newton-Raphson method

$$\frac{1}{f'(\mathbf{x}^{(m)})} \equiv \alpha. \tag{2.6c}$$

2.1.3.2 Adaptive Resonance Theory (ART)

ART is as an unsupervised learning rule. By using long term and short term memories, ART can classify the input and enhance its memory capability. This means if the input pattern stored in short term memory is similar to the expected pattern stored in long term memory, the input is classified to that category. If there is no similarity between the input pattern and expected patterns, a new class is formed.

2.1.3.3 Kohonen Learning

The Kohoren Learning Rule is written as

$$W_i^{new} = W_i^{old} + a\,(X_i - W_i^{old}) \tag{2.7}$$

where $X_i = \{x_{1i}, x_{2i}, \cdots x_{ni}\}$ is the input vector for processing element i, W_i is the weight vector associated with inputs for processing element i, and a is a learning constant.

The Kohonen learning rule is useful for unsupervised learning.

2.1.3.4 Boltzmann Learning

This is a reinforcement learning rule [HiSe86]. The significant features of Boltzmann learning are that a probability function, the Boltzmann distribution in this case, is used to adjust connection weights, and simulated annealing is used to control dynamical settling of the network into a minimum energy state.

The above learning rules are widely used in feedforward neural networks. For feedback neural networks, the learning rule is often defined as the state updating of a network [Hec90]. For example,

$$x_i^{new} = sgn(\sum_{j=1}^{n} W_{ij} x_j^{old} - T_i) \tag{2.8}$$

where x_i is the state, sgn is a sign function, T_i is a threshold, and w_{ij} is the connection weight. In general, w_{ij} takes a fixed value, and $w_{ij} = w_{ji}$.

Another aspect of learning within a feedback neural network is that an energy function E is defined and E is always decreasing as the state is updating. Eventually, the network arrives at a stable state when the learning process converges [KaBr89, Hec90].

2.2 ANN Architectures

Many ANN architectures have been proposed for solving a variety of problems such as optimization, pattern recognition, computer vision, and associative memory [Mea88, Hop84, TaHo86, AnRo88, FoTa88, TaTr91i, MoAG91]. According to their topologies, the architectures can be divided into three categories: feedback neural networks, feedforward neural networks and cellular neural networks.

2.2.1 Feedback Neural Networks

In this type of neural network there exist feedback paths from outputs to inputs of neurons. A key dynamic issue of these networks is that the energy derivative must be strictly decreasing so that the system equilibrium state can be reached.

2.2.1.1 The Hopfield Model

The significance of the Hopfield model is that it can be built with analog circuit components and is suitable for analog VLSI implementation [Hop84, TaHo86]. In this model, each processing element is an amplifier with a capacitor C_i and a resistor ρ_i at the input node, and a sigmoidal transfer function from input u_i to output v_i. Processing element i is connected to processing element j via a finite conductance T_{ij} all of which form a symmetric connection weight matrix T. The architecture of the general Hopfield network is shown in Figure 2.3. By using KCL, the time derivative of input potential u_i can be written as

$$C_i \frac{du_i}{dt} = \sum_{j=1}^{n} T_{ij} v_j - \frac{u_i}{R_i} + I_i \tag{2.9}$$

where

$$\frac{1}{R_i} = \frac{1}{\rho_i} + \sum_{j=1}^{n} \frac{1}{R_{ij}} = \frac{1}{\rho_i} + \sum_{j=1}^{n} T_{ij}, \tag{2.10}$$

I_i is an input bias current for neuron i.

An energy function is defined by integral of equation (2.9) as

$$E = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} T_{ij} v_i v_j - \sum_{i=1}^{n} I_i v_i + \sum_{i=1}^{n} \frac{1}{R_i} \int_{o}^{v_i} f^{-1}(\xi_i) d\xi_i . \tag{2.11}$$

The time derivative of the energy function can be found as

$$\frac{dE}{dt} = -\sum_{i=1}^{n} \frac{1}{C_i} \frac{df(u_i)}{du_i} \left(\frac{\partial E}{\partial v_i}\right)^2. \tag{2.12}$$

Because $v_i = f(u_i)$ is monotonically increasing, and C_i is positive, $\frac{dE}{dt} \leq 0$ for all t, the value of the energy function is strictly decreasing and becomes zero only at equilibrium point at which $\frac{\partial E}{\partial v_i} = -C_i \frac{du_i}{dt} = 0$ for all i. As described at the end of section 2.1.3.4, this represents the learning in the Hopfield model [KaBr89].

The Hopfield model has been used in combinatorial optimization, linear programming, dynamic programming, signal and image processing applications [Hop84, ChMS91, NaCh92]

2.2.1.2 The Kennedy-Chua Model

Kennedy and Chua proposed a cononical circuit model with feedback [ChLi84, KeCh88]. Their model can be applied to both linear and nonlinear programming. In addition, the networks' structural parameters are in correspondence with the coefficients of the objective function and constraints descriptions. Figure 2.4 illustrates an architecture of the model, where p-cells are constraint amplifier units, and $\int$-cells are integrators (neurons).

The basic relation holds for the circuits as given in [KeCh88]

$$C_i \frac{dv_i}{dt} = - \frac{\partial \phi}{\partial v_i} - \sum_{j=1}^{m} i_j \frac{\partial g_j}{\partial v_i} \tag{2.13}$$

where $\phi(\mathbf{v})$ is the objective function, $\mathbf{g}(\mathbf{v})$ are constraints, $i_j = p_j(g_j(\mathbf{v}))$, C_i is capacitance, and $\mathbf{v}$ is the node voltages $v_1, v_2, ..., v_n$.

The energy function can be defined as

$$E(\mathbf{v}) = \phi(\mathbf{v}) + \sum_{j=1}^{m} \int_0^{g_i(\mathbf{v})} p_j(\xi) d\xi. \tag{2.14}$$

The derivative of energy function is

$$\frac{dE}{dt} = = -\sum_{i=1}^{n} C_i \left(\frac{dv_i}{dt} \right)^2. \tag{2.15}$$

Obviously, $C_i > 0$, $\left(\frac{dv_i}{dt}\right)^2 \geq 0$, $\Rightarrow \frac{dE}{dt} \leq 0$. Therefore, $E(\mathbf{v})$ is a Lyapunov function which ensures that the system is completely stable.

Because the linear programming network of Hopfield obeys the same unifying stationary cocontent theorem as the cononical nonlinear programming circuit of Kennedy and Chua, the Hopfield model can be regarded as a special case of the Kennedy-Chua model [KeCh87].

2.2.1.3 The Rordriguez Model

Although it is similar to the Kennedy-Chua model in principle and purpose, the Rordriguez model has a distinct feature. The RC-active technique is replaced by SC-reactive (Switched-Capacitor) technique which is more suitable for VLSI implementation [Ro*et*90].

Figure 2.5 is a block diagram of this model. The time derivative of the objective function $\Psi(\cdot)$ can be expressed as

$$\frac{d\Psi}{dt} = \sum_{i=1}^{n} \frac{\partial \Psi}{\partial x_i} \frac{dx_i}{dt} = \sum_{i=1}^{n} (-\tau \frac{dx_i}{dt}) \frac{dx_i}{dt} = -\tau \sum_{i=1}^{n} (\frac{dx_i}{dt})^2 \qquad (2.16)$$

where τ is a positive parameter. Therefore, $\frac{d\Psi}{dt} \leq 0$.

In order to implement a discontinuous pseudo-cost function, a basic switched-capacitor integrating "neuron" is used in this model. Figure 2.6 shows this scheme. By using *even* and *odd* clock phases, the operation of Figure 2.6 becomes threshold-controlled [Ro*et*90].

2.2.2 Feedforward Neural Networks

A feedforward network is composed of multiple layers of neurons. Each layer forms an individual group to perform a specified information task. The whole network is connected from layer to layer, but there exists no connection in the same layer. The signals propagate from the first layer (input layer) to the last layer (output layer). The transfer functions used may be different from layer to layer, but they must be the same

within a given layer.

2.2.2.1 The Two-layer Model

This model consists of only an input-layer and an output-layer. It can be used in simple information processing. One kind of two-layer model has been used in an associative memory [Kos87].

2.2.2.2 The Three-layer Model

When the representation provided by the outside world is such that the similarity structure of the input and output patterns are very different, the two-layer network without internal stages will be unable to perform the necessary mapping. The three-layer model is composed of an input-layer, an output-layer, and a hidden-layer. This model is often used in pattern recognition based on some learning algorithms [Hec87, WiLe90].

2.2.2.3 The Multi-layer Model

This model is similar to that of the three-layer model except with more than one hidden layers. The multi-layer is capable to handle more complex information processing tasks. One thing here worth pointing out is that the hidden layers must use non-linear transfer functions. Because multiple hidden layers with linear transfer function are equivalent to a single hidden layer [DaDe91]. In other words, it makes no sense to use multiple hidden layered network when linear transfer functions are used.

2.2.3 The Cellular Neural Networks

In fact, many live organs are built as a cellular structure so that connections and interactions among cells within an organ are more efficient. Chua and Yang developed

a model of cellular neural networks [ChYa88a, ChYa88b]. An example of a two dimensional cellular neural network with 4×4 cells is shown in Figure 2.7, where each cell consists of linear capacitors, linear resistors, linear and nonlinear controlled sources and independent sources.

Due to their binary value outputs, the cellular neural networks have been used in image processing for feature extraction and character recognition [RoKa82, ChYa88b]. Moreover, the nearest neighbor interactive property and regularity of cellular neural networks makes them suitable for VLSI implementation.

2.3 ANN Applications

By far the largest area of research in neurocomputing is that of applications. Meanwhile, it has been shown that some significant achievements in neurocomputing were associated with applications [WiHo60, StPi63, Hop82, CaGr87]. In this Section, application categories, methodology and related techniques are described.

2.3.1 Categories of Applications

Neurocomputing applications can be roughly divided into four classes: *Pattern Recognition, Control, Data Analysis, and Optimization.*

Pattern recognition is the major application area of neurocomputing [Gro76, Hec90]. The input of the neurocomputing system is spatial image data or time serial signals. The neural network fulfills a mapping from an input vector to an output classification. Associative memory can be regarded as a special example of this category.

Adaptive control has a great significance for real world problems. By supervised traing or reinforced training, neural network controllers attempt to adjust the system

state by minimizing the difference between actual output and desired output. The Vision-Based Broomstick Balancer [ToWi88] and Automobile Autopilot [She88] are good examples of this kind of application.

Data analysis is the highest potential application area for emulating human brain behavior to some extent. The ultimate purpose of data analysis is usually to extract concentrated information from fuzzy raw data. Based on the knowledge from expert systems, the trained neural network can make decisions for a variety of input information in a very short time. For example, the One-Minute Manager has shown surprisingly good performance for a neural network based medical diagnosis system [Bla86].

Optimization is a straight forward application of neural networks, because neurocomputing is a dynamic process such that an equilibrium point of the system is often associated with an optimal solution for a given problem. Recently, several achievements have been made for different problems in this area such as combinatorial optimization [Hop84], linear and nonlinear programming [TaHo86, KeCh88, MaSh92 ChMS92], and dynamic programming [ChMS91]. Since the dissertation focuses on this area, the issues concerned in great detail will be provided in the following Chapters.

2.3.2 Methodology of Neurocomputing Applications

In order to solve real problems with neurocomputing efficiently, it is necessary to combine neurocomputing with other technological methods. In other words, methodology plays an important role in neurocomputing application projects. In the following three key issues will be briefly discussed.

2.3.2.1 Problem Solving Methodology

It is essential for application researchers to have a comprehensive understanding of the work domain for which the application is intended. Therefore, the primary methodology for neurocomputing applications is to "work in reverse" [Hec90]. That is,

first, a list of neural networks is constructed by the domain specialists. The list describes each network's problem solving capabilities and training requirements. Then, a set of operational functions is developed. Furthermore, applications in each of these areas are considered, and candidate applications are evaluated as well.

2.3.2.2 Functional Specification

The purpose of functional specification is to define an application project precisely so that the developers who are not domain specialists can work on the project more efficiently. Generally, the functional specification can be outlined as follows [Hec90]:

(1) Description: Describe the capabilities of application project to be developed.

(2) System Interface: Identify requirements for interfaces between the neurocomputing system and other systems.

(3) Human Interface: Identify requirements or constraints regarding the interface between the developed system and humans.

(4) Performance: Provide requirements on the overall performance of the system, such as accuracy, speed, reliability, size and mass.

(5) Economic factors: Estimate development cost and potential benefit of the project.

2.3.2.3 Technological Development

Once the functional specifications have been made, technical development will begin. This includes mathematical mapping, gathering training data, developing learning algorithms, and modifying network architecture. Of course, technical details depend on the particular application. A procedure for optimization applications is presented next.

2.3.3 A Procedure for Optimization Applications

Since ANNs for optimization problems will be developed and applied in the following Chapters, a general procedure for solving this kind of problems is described below:

(1) Translate a real problem into an optimization problem;

(2) Develop a dynamical system for the optimization problem;

(3) Choose proper variables, and encode them;

(4) Map the the dynamical system into a neural network;

(5) Define the energy function for the network;

(6) Develop a learning algorithm for updating connection weight, or identify and prove the conditions for stability and convergence of the network;

(7) Choose a proper initial point, and find a solution to the problem by the network (simulation or physical implementation);

(8) Improve the performances of the network.

In this procedure, heuristics are frequently used. For example, in the Traveling-Salesman Problem (TSP), the best path is always chosen from a city to one of its four nearest neighbors; if two cities A and B are as far apart as possible, they will tend to occur near opposite sites of the closed route [HoTa85]. In the pattern recognition, the expert's knowledge often gives a good hint for feature extraction. In other words, human intelligence still plays a dominant role in the design of ANN application systems.

Loosely speaking, ANN applications can be divided into two categories: stochastic-oriented and deterministic-oriented. Pattern recognition, classification, robot vision and event forecasting are the stochastic applications, whereas the well-defined optimization such as the TSP, linear and nonlinear programming problems are the deterministic applications. In the stochastic applications, the computing accuracy

depends on learning algorithm and variable choice. The relative error is measured by the comparison between the computed result and the actual value, with a range from 1% to 8% [EOMD91, PaEM91, SrLC91]. In the deterministic applications, there exists an exact solution to the problem so that the relative error can be defined quantitatively. The computing accuracy depends on the mathematic formulation and architecture. In addition, the environmental situation exerts an influence on the accuracy. The influential factors include parameters such as the time increment for the Runge-Kutta numerical integration in the software simulation, and the quality index of circuit components in the hardware implementation. From an engineering point of view, it is essential to improve the computing accuracy for ANN applications. This issue will be discussed In more detail in the next Chapter.

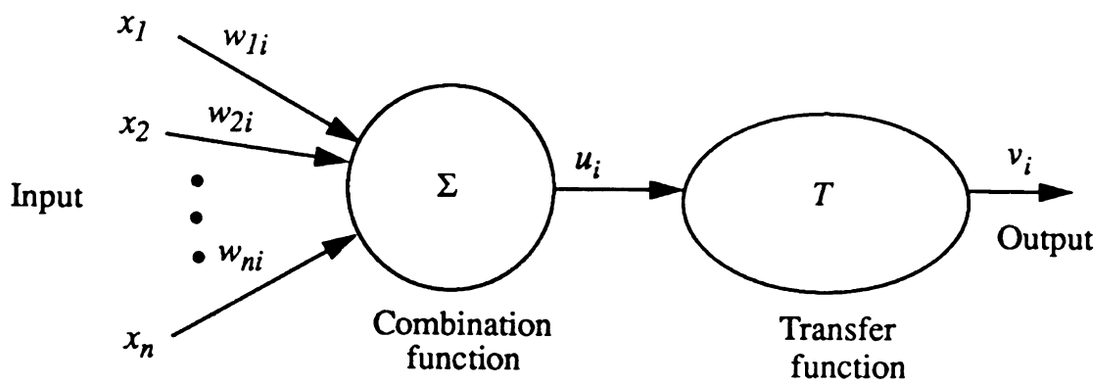


Figure 2.1. A processing element.

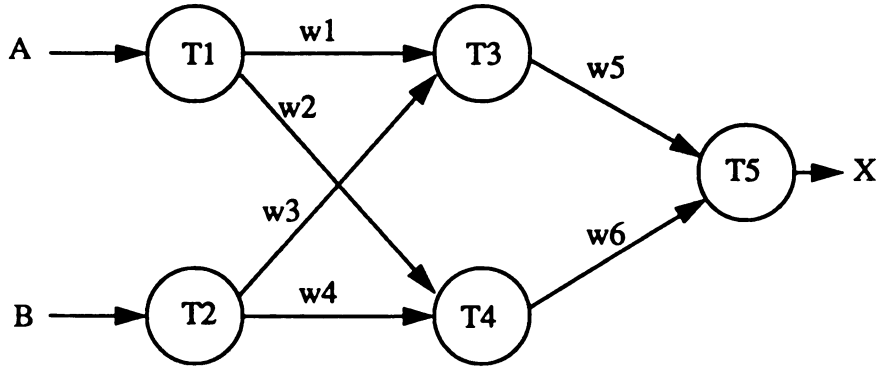


Figure 2.2. A logic unit.

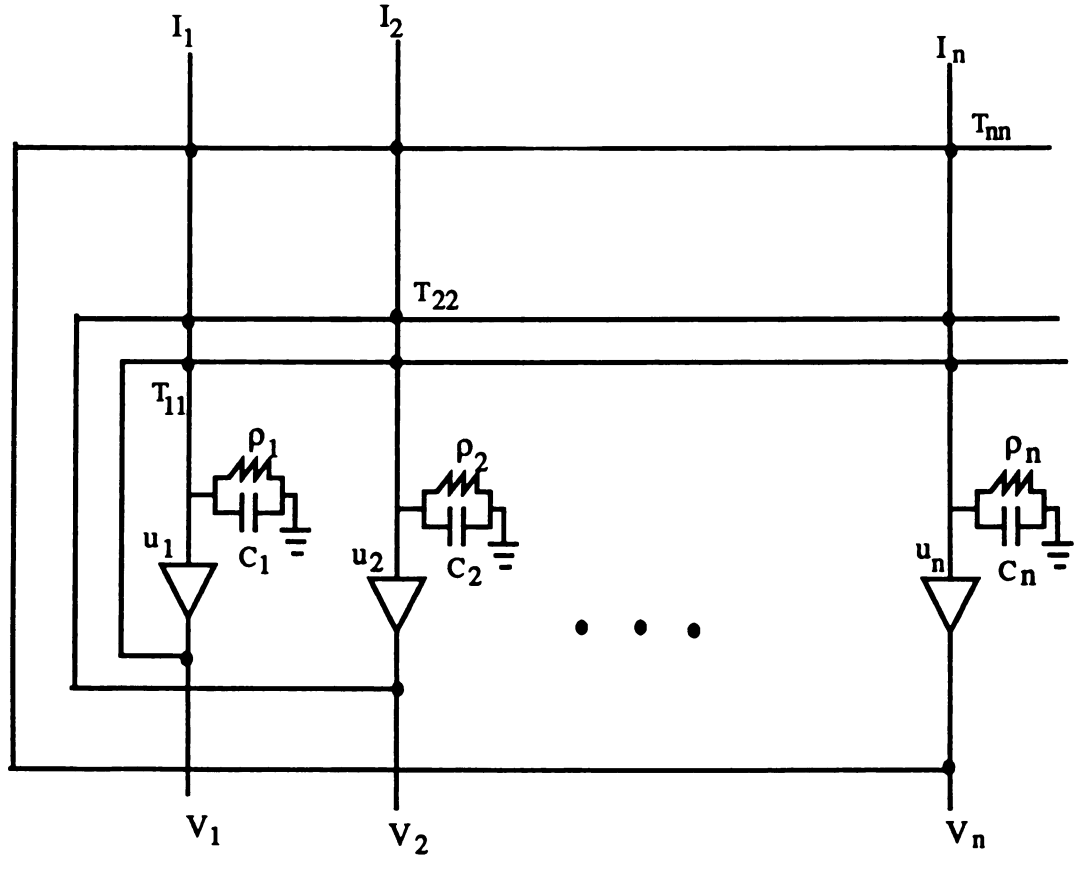


Figure 2.3. The Hopfield network.

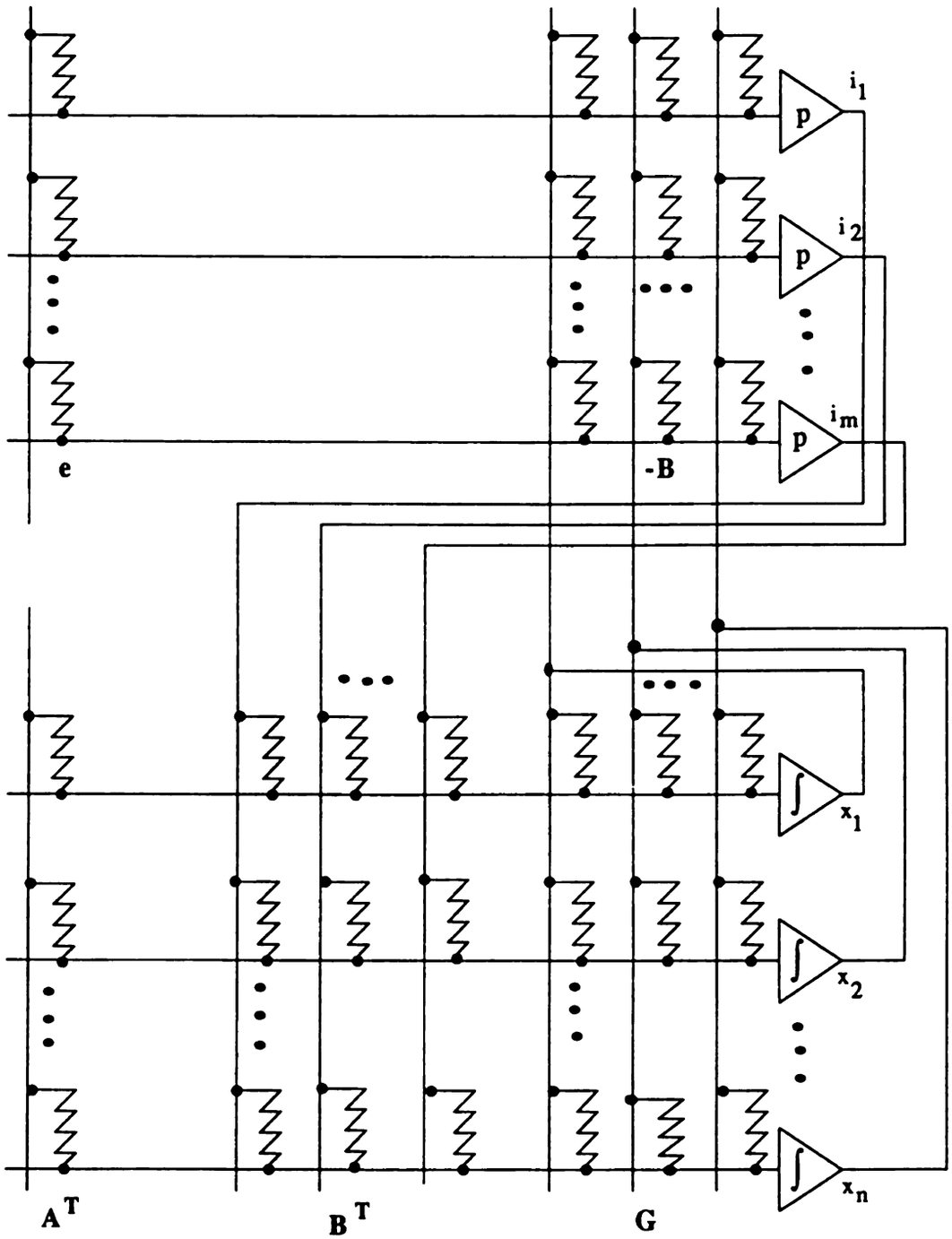


Figure 2.4. The Kennedy-Chua network.

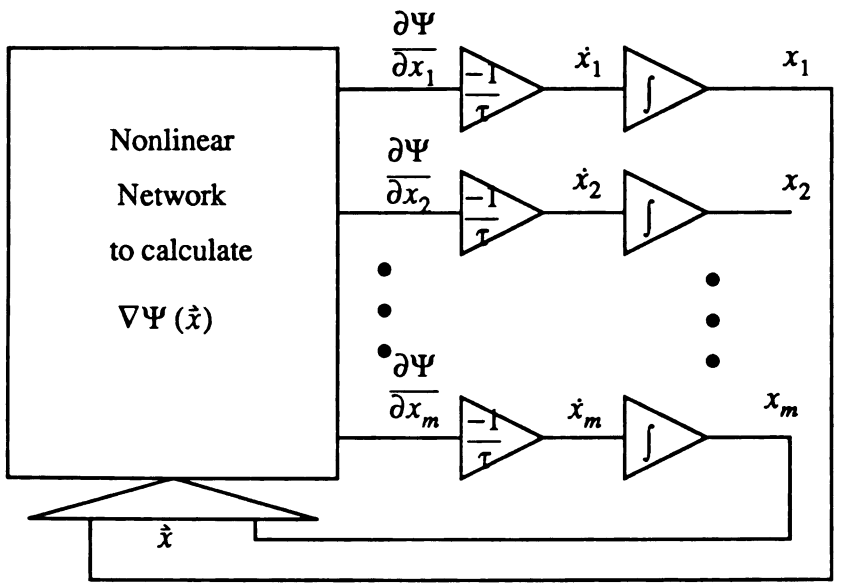


Figure 2.5. The block diagram of Rordriguez network .

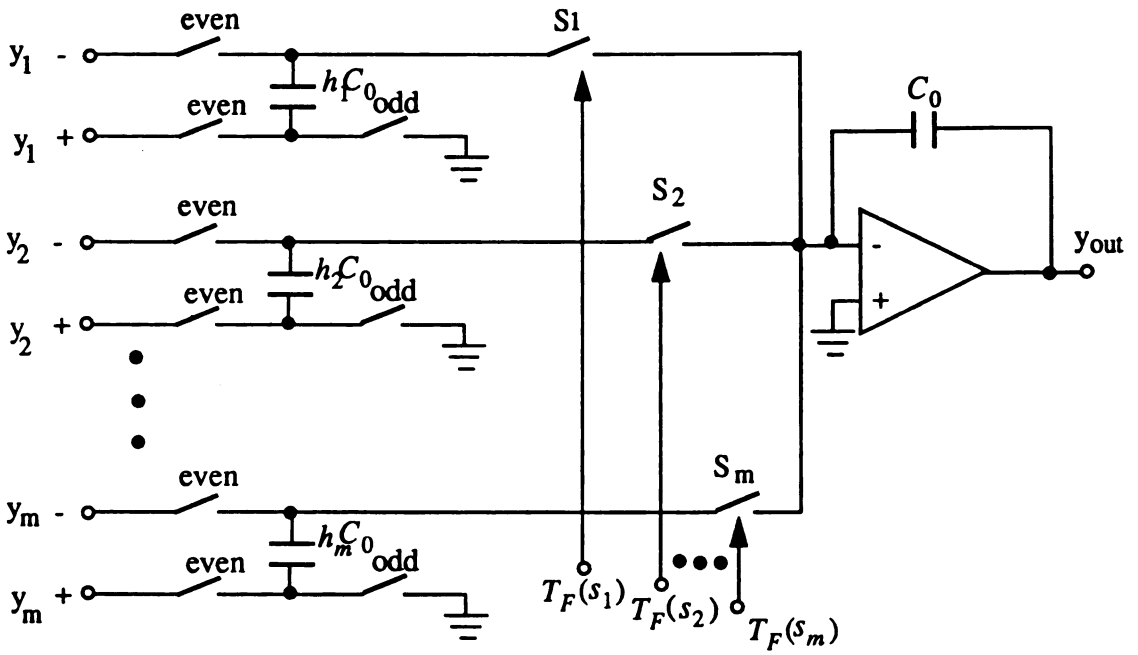


Figure 2.6. The basic switched-capacitor integrating neuron.

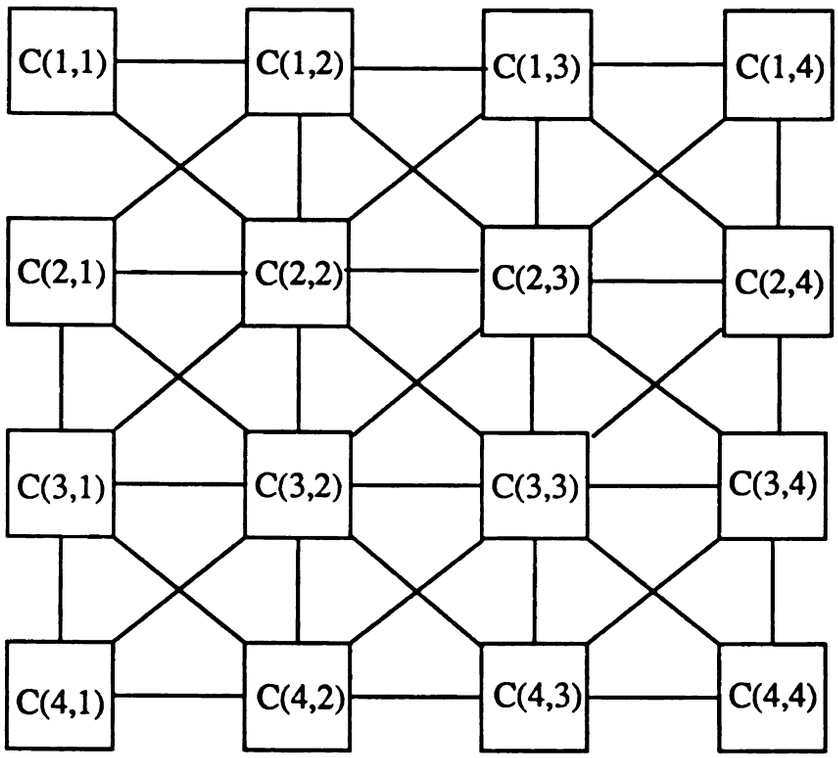


Figure 2.7. A cellular network .

Table 2.1. Comparison of the nervous system and an ANN system.

Nervous system	ANN system
Neuron	Processing element
Dendrites	Combining function
Cell body	Transfer function
Axons	Element output
Synapses	Weights

Table 2.2. Weights and thresholds of a neural logic unit.

Operation	w_1	w_2	w_3	w_4	w_5	w_6	T_1	T_2	T_3	T_4	T_5	X
AND	1	0	0	1	0.5	0.5	1	1	1	1	1	$A \cdot B$
OR	1	0	0	1	1	1	1	1	1	1	1	$A + B$
XOR	1	-1	-1	1	1	1	1	1	1	1	1	$A \oplus B$
XNOR	0.5	-1	0.5	-1	1	1	1	1	1	0	1	$A \odot B$

Chapter 3

Neural Networks for Linear and Nonlinear Programming

In Chapter 3, a method for improving the performance of artificial neural networks for linear and nonlinear programming is presented. By analyzing the behavior of the conventional penalty function, the reason for the inherent degenerating accuracy is discovered. Based on this, a new combination penalty function is proposed which can ensure that the equilibrium point is sufficiently close to the optimal point. A known neural network model has been modified using the new penalty function and a corresponding circuit scheme is given. Simulation results show that the relative error for linear and nonlinear programming is substantially reduced by the new method.

3.1 Introduction

Artificial Neural Networks (ANNs) have been applied as models of computation for solving a wide variety of problems in such diverse fields as combinatorial optimization [Hop84, TaHo86, FoTa88], computer vision [RuHW86], and pattern recognition

[GaGr87]. In particular, some networks have been used for solving linear and non-linear programming problems. There are three well-known models among these networks.

Tank and Hopfield used sigmoidal input-output cells as a neuron model and mapped the cost function and constraints into close-looped networks [Hop84, TaHo86]. When a constraint violation occurs, the magnitude and direction of the violation is fed back to adjust the states of the neurons in the network. Their model can be directly implemented with analog circuits, however, the global optimal solution can be obtained only under the assumption of very high gain and infinite conductance.

Kennedy and Chua used integrator cells to model neurons and mapped the cost function and constraints into a canonical nonlinear circuit [KeCh88]. Their model can be applied to both linear and nonlinear programming. In addition, the networks' structural parameters are in correspondence with the coefficients of the objective function and constraint descriptions. The shortcoming of their network, however, is that the optimal solution can only be guaranteed with infinite conductance.

Rodriguez-Vazquez, *et al.*, also used integrator cells for the neurons and mapped the cost function and constraints into a switched-capacitor network [Roet90]. Their pseudo-cost function is not continuous from the feasible region to the infeasible region. The distinct feature of their model is in a substitution of the RC-active technique by an SC-reactive technique which is more suitable for VLSI implementation. However, a new problem arises — there is no equilibrium point in the system because of discontinuous behavior.

Due to the limitations mentioned above, the relative error of results from these models is 1% to 3% [KeCh88, ChLi84], although, it is recognized that these are case-dependent results [MaSh89]. The motivation for this work is to develop a new method for improving the computing accuracy of linear and nonlinear programming.

In this Chapter, the role of the boundary for linear and nonlinear programming is investigated. By analyzing the behavior of the conventional penalty function, the reason for degenerating its accuracy is discovered. Based on this, a new combination penalty function is proposed which can ensure that the equilibrium point is sufficiently close to the optimal point. By using the new penalty function, a modified ANN model is described and an alternative circuit scheme for the constraint amplifier unit is given. The stability and convergence of the modified model is addressed. Finally, a few examples of linear and nonlinear programming are solved by simulation of the new model.

3.2 Methodology Review

A large class of problems in science and engineering can be formulated as optimization problems in which a minimum or maximum solution is sought to an objective function subject to certain constraints. From a mathematical point of view, however, the case of maximizing a concave function is equivalent to that of minimizing a convex function [BaSh79]. Therefore, only the latter will be discussed here. Consider the following optimization formulation:

Minimize

$$\phi(\mathbf{x}) \tag{3.1}$$

subject to

$$g_i(\mathbf{x}) \geq 0 \qquad i = 1, \cdots, m \tag{3.2}$$

where $\phi(\mathbf{x})$ is a nonlinear strictly convex function, $\mathbf{x} \in R^n$, and $g_i(\mathbf{x})$ is convex for all i. The nonempty convex set $S=\{\mathbf{x}:g_i(\mathbf{x}) \geq 0, i=1, 2, ..., m\}$ is called the feasible region.

The penalty function and the Lagrange multiplier are two well-known methods for solving the problem [BaSh79, HiLi86]. The basic idea is to translate the constrained minimization problem into a new unconstrained minimization problem by introducing a modified objective function.

3.2.1 The Penalty Function Method

Define a new objective function as

$$\phi^*(\mathbf{x}) = \phi(\mathbf{x}) + \mu\alpha(\mathbf{x}) \tag{3.3}$$

where

$$\alpha(\mathbf{x}) = \sum_{i=1}^{m} p_i(\mathbf{x}) \tag{3.4}$$

and

$$p_i(\mathbf{x}) = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \geq 0, \\ |g_i(\mathbf{x})|^p & \text{if } g_i(\mathbf{x}) < 0. \end{cases} \tag{3.5}$$

For the conventional penalty function, $\mu > 0$, and p is a positive integer, in general, $p \geq 2$.

A theoretical analysis has shown that only when μ approaches infinity can the solution to (3.3) be the same as the solution to (3.1) and (3.2) [BaSh79]. Two problems arise: it is difficult to realize infinite μ physically, and too large a μ may result in an ill-conditioned problem [BaSh79].

3.2.2 The Lagrange Multiplier Method

This method can only handle the exclusive equality constraints. By introducing a relaxed function S_i, the inequality constraints can be transformed into equalities:

$$q_i = g_i(\mathbf{x}) - S_i = 0 \tag{3.6}$$

where S_i is a non-negative function depending on a relaxed variable x_{n+i}, generally taking the form as

$$S_i = x_{n+i}{}^2. \tag{3.7}$$

Thus, the problem is changed to minimizing a modified objective function without constraints

$$\phi^*(\mathbf{x}) = \phi(\mathbf{x}) + \sum_{i=0}^{m} \lambda_i q_i \tag{3.8}$$

where λ_i is a unknown constant. A minimum point must satisfy

$$\begin{cases} \dfrac{\partial \phi^*}{\partial x_i} = 0 & \text{for } i = 1, 2, \cdots, n \\ q_i = 0 & \text{for } i = 1, 2, \cdots, m. \end{cases} \tag{3.9}$$

Consequently, solving the minimization problem with equality constraints is equivalent to finding a root of the simultaneous equations with real coefficients as formulated in (3.9).

3.2.3 ANN Techniques

Tank and Hopfield first developed a simple optimization neural network and applied it to the linear programming problem [TaHo86]. Their network consists of two parts: an amplifier unit and a constraint unit. They are associated by the following equations:

$$C_i \frac{du_i}{dt} = -\left(a_i + \frac{u_i}{R} + \sum_{j=1}^{m} i_j \frac{\partial f_j}{\partial v_i}\right) \tag{3.10}$$

$$= -\left(a_i + \frac{u_i}{R} + \sum_{j=1}^{m} i_j d_{ji}\right)$$

and

$$v_i = g(u_i) \tag{3.11}$$

where $g(\cdot)$ is a sigmoid function.

$$\phi(\mathbf{v}) = \mathbf{v}^T \mathbf{a} \tag{3.12}$$

is the objective function subject to the constraints

$$f_j(\mathbf{v}) = \mathbf{v}^T \mathbf{d}_j - b_j \geq 0 \tag{3.13}$$

for $j = 1$ to m, and $i_j = q(f_j(\mathbf{v}))$ is the penalty function.

The energy function for the network is defined as

$$E(\mathbf{v}) = \mathbf{v}^T \mathbf{a} + \sum_{i=1}^{n} \frac{1}{R} \int_0^{v_i} g^{-1}(\zeta) d\zeta + \sum_{j=1}^{m} \int_0^{f_j(\mathbf{v})} q(\varsigma) d\varsigma \tag{3.14}$$

and the derivative of the energy function can be found as

$$\frac{dE}{dt} = -\sum_{i=1}^{n} C_i \frac{dv_i}{dt} \frac{du_i}{dt} \tag{3.15}$$

$$= -\sum_{i=1}^{n} C_i \frac{dg(u_i)}{du_i} \left(\frac{du_i}{dt}\right)^2.$$

Since $g(u_i)$ is a monotone increasing function, $\frac{dg(u_i)}{du_i}$ is positive. Thus, $\frac{dE}{dt} \leq 0$ indicating that the system will tend to move toward a local minimum point.

In a modification to this model, Kennedy and Chua proposed a neural network for nonlinear programming described by a set of first-order differential equations in (2.13). The energy function $E(\mathbf{x})$ is defined in (2.14). It has been shown that this network can cope with linear and nonlinear programming [KeCh88].

Networks of this genre were further developed by Maa and Shanblatt in which a variety of ANN optimization models for linear, quadratic and general nonlinear programming were analyzed. They showed that the optimization network formed by

$$\dot{\mathbf{x}} = -\nabla\phi(\mathbf{x}) - s\left[\sum_{j=1}^{m} g_j^{+}(\mathbf{x}) \nabla g_j(\mathbf{x})\right] \tag{3.16}$$

fulfills both the Kuhn-Tucker optimality conditions and the criteria of the penalty function method [MaSh91]. Under an assumption of convexity, the network described by the dynamics of equation (3.16) traverses along the surface of the energy function and eventually settles on a local minimum. Furthermore, a two-phase optimization network model was proposed such that different dynamics are used as the phase is changed by a predetermined timing switch. As a consequence, the exact solutions of the network can be guaranteed by adjusting the penalty parameter.

3.3 A New Combination Penalty Function

Since the penalty function method is widely used in ANN optimization models, it is essential to investigate the behavior of the conventional penalty function technique in order to propose an improved network for optimization problems.

3.3.1 The Boundary Situation

Let $\mathbf{x}^{**}$ be the solution to (3.1) without constraints, and let $\mathbf{x}^{*}$ be the solution to (3.1) and (3.2) with constraints. The following Theorem can be stated.

Theorem 3.1: If $\mathbf{x}^{**} \in S$, then $\mathbf{x}^{*} = \mathbf{x}^{**}$; otherwise, $\mathbf{x}^{*} \in B(S)$, where S is bounded and B(S) is the boundary of S.

Proof can be found in [BaSh79].

From Theorem 3.1, the constrained problem can be divided into two cases: case 1 in which the solution lies in the interior of the feasible region, and, case 2 where the solution lies on the boundary of the feasible region. For the former, the solution is the same as that without constraints and can be expressed as a set of algebraic equations by setting the partial differentials of $\phi(\mathbf{x})$ equal to zero. For the latter, (nonlinear programming with constraints — NLPW), obtaining the solution is more difficult due to

possible complex situations on the boundary. In practice, most optimization problems belong to case 2 and therefore it will be examined in detail.

For linear programming (LP) without constraints there is no minimum point. The minimum point for LP with constraints (LPW) can only be on the boundary of the feasible region [HiLi86] provided the feasible region is bounded. Thus, a strategy for improving computing accuracy will be useful to both NLPW and LPW.

Consider a gradient system which is often used for optimization by ANNs:

$$\dot{\mathbf{x}} = -\nabla\phi^*(\mathbf{x}) = -[\nabla\phi(\mathbf{x}) + \nabla\mu\alpha(\mathbf{x})]. \tag{3.17}$$

Due to the convex assumption, the unique equilibrium point (i.e., the minimum point for (3.1) and (3.2)) must satisfy

$$\dot{\mathbf{x}} = 0 \tag{3.18}$$

which means

$$\nabla\phi(\mathbf{x}) = -\nabla\mu\alpha(\mathbf{x}). \tag{3.19}$$

Figure 3.1 illustrates the situation for case 2 where dashed curves indicate equipotential surfaces of $\phi(\mathbf{x})$. Let $\mathbf{x}_b$ be the solution of the problem, described by equations (3.1) and (3.2), lying on the boundary of the feasible region, *i.e.*, $\mathbf{x}_b = \mathbf{x}^*$. Let $\mathbf{x}_{bo}$ be a point outside the feasible region S but in a neighborhood of $\mathbf{x}_b$ so that $\phi(\mathbf{x}_{bo}) < \phi(\mathbf{x}_b)$. According to the assumption of convexity, and equation (3.19), it can be concluded that vectors $\nabla\phi(\mathbf{x}_b)$ and $\nabla\mu\alpha(\mathbf{x}_b)$ are in the opposite direction as shown in Figure 3.1. Let $l(\mathbf{x}_b) = (\mathbf{x}_{b-\delta}, \mathbf{x}_b)$, called a small "left" neighborhood of $\mathbf{x}_b$. If the penalty function is too weak to increase $\mu\alpha(\mathbf{x})$ enough to offset the decreasing $\phi(\mathbf{x})$ in the range $l(\mathbf{x}_b)$, then $\phi^*(\mathbf{x}_{bo})$ may be slightly lower than $\phi^*(\mathbf{x}_b)$, or almost equal to $\phi^*(\mathbf{x}_b)$, where $\mathbf{x}_{bo} \in l(\mathbf{x}_b)$. For the conventional penalty function, it is clear that $|g_i(\mathbf{x}_{bo})|^{p-1}$ is so small that its offset can be almost ignored. This is the reason why the computing accuracy of the previous models, which use the conventional penalty

function, is not satisfactory.

3.3.2 A New Combination Penalty Function

According to the above analysis, it is desirable to find a new penalty function which can ensure that a solution to (3.3) is sufficiently close to that of the original problem (3.1) and (3.2) [ChMS92].

The objective is to make $\frac{\partial p_i}{\partial g_i(\mathbf{x}_{s-})}$ increase rapidly. A fraction-exponent function can be chosen as

$$p_i(\mathbf{x}) = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \geq 0, \\ |g_i(\mathbf{x})|^{\frac{p+1}{p}} & \text{if } g_i(\mathbf{x}) < 0 \end{cases} \tag{3.20}$$

where p is a positive integer. Correspondingly, the derivative of $p(\cdot)$ is

$$\frac{\partial p_i(\mathbf{x})}{\partial g_i(\mathbf{x})} = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \geq 0, \\ \frac{p+1}{p}|g_i(\mathbf{x})|^{\frac{1}{p}} & \text{if } g_i(\mathbf{x}) < 0 \end{cases} \tag{3.21}$$

Suppose $g_i(\mathbf{x}) = x$, Figure 3.2 illustrates three example penalty functions' derivatives $|x|$, $|x|^2$, and $|x|^{\frac{1}{8}}$. Obviously, the last one has a good behavior in $l(0)$.

One new problem arises: for $|g_i(\mathbf{x})| \geq 1$, the growth of $|g_i(\mathbf{x})|^{\frac{1}{p}}$ is much smaller than that of a linear function, therefore, the penalty function has a reduced influence on $\phi^*(\mathbf{x})$ as $|g_i(\mathbf{x})|$ increases. At this time, however, $|g_i(\mathbf{x})|$ becomes large enough to be a penalty. Therefore, a sectioned function can be used as

$$p_i(\mathbf{x}) = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \geq 0, \\ |g_i(\mathbf{x})|^{\frac{p+1}{p}} & \text{if } g_i(\mathbf{x}) < 0 \text{ and } |g_i(\mathbf{x})|^{\frac{1}{p}} \geq |g_i(\mathbf{x})|, \\ |g_i(\mathbf{x})|^2 & \text{if } g_i(\mathbf{x}) < 0 \text{ and } |g_i(\mathbf{x})|^{\frac{1}{p}} < |g_i(\mathbf{x})|. \end{cases} \tag{3.22}$$

An illustration of the effect of the derivatives of this sectioned function is shown in Figure 3.3.

Furthermore, simplifying equation (3.22) by comprehensively utilizing $|g_i(\mathbf{x})|^{\frac{p+1}{p}}$ and $|g_i(\mathbf{x})|^2$, provides a combination penalty function of the form

$$p_i(\mathbf{x}) = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \geq 0, \\ k_1 |g_i(\mathbf{x})|^{\frac{p+1}{p}} + k_2 |g_i(\mathbf{x})|^2 & \text{if } g_i(\mathbf{x}) < 0 \end{cases} \tag{3.23}$$

where $k_1 > 0$, $k_2 > 0$. The purpose of putting these two coefficients together is to simplify implementing circuits which will be discussed in Section 3.4.

Equation 3.23 is the new desired penalty function, whose derivative is shown in Figure 3.4. Comparing Figure 3.3 and Figure 3.4, it is apparent that the latter is smoother and can yield a more accurate result. Since the derivative of the latter is continuous, and the value of the combination function at any given x ($x < 0$) is greater than that of the corresponding selectioned function. In addition, the hardware implementation for this combination penalty function will be more efficient than that of a sectioned function which requires a comparator (see Section 3.4).

Note that the combination penalty function is still a continuous function since it is the sum of two continuous functions.

3.4 Hardware Configuration

As described in Section 2.2, the feedback neural networks are built in a "closed" form so that they can be more easily implemented. Among them, Kennedy and Chua's network is a continuous model, and can handle linear and nonlinear situations without a structural revision [KeCh88]. By modifying of the energy function and constraint amplifier circuit of Kennedy and Chua's canonical nonlinear programming model, an

implementation of the combination penalty function proposed in Section 3.3 is discussed here.

Consider a quadratic programming problem:

Minimize

$$\phi(\mathbf{x}) = \mathbf{A}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{G}\mathbf{x} \tag{3.24}$$

subject to

$$\mathbf{g}(\mathbf{x}) = \mathbf{B}\mathbf{x} - \mathbf{e} \geq 0 \tag{3.25}$$

where **A** and **x** are n-vectors, **g** and **e** are m-vectors, **B** is an m×n matrix, and **G** is an n×n symmetric, positive definite matrix. If **G** = 0, then this becomes a linear programming problem.

Define the energy function as

$$E(\mathbf{x}) = \phi(\mathbf{x}) + \sum_{j=1}^{m} p_j(\mathbf{x}) \tag{3.26}$$

where $p_j(\mathbf{x})$ is the new penalty function from equation (3.23), and the dynamical equation at node i can be written as

$$\frac{dx_i}{dt} = -\frac{\partial \phi}{\partial x_i} - \sum_{j=1}^{m} \frac{\partial p_j}{\partial g_j}\frac{\partial g_j}{\partial x_i}. \tag{3.27}$$

Due to the continuity of $p_j(\cdot)$, the derivative of the energy function can be obtained as follows:

$$\begin{aligned}
\frac{dE}{dt} &= \sum_{i=1}^{n} \frac{\partial \phi}{\partial x_i}\frac{dx_i}{dt} + \sum_{j=1}^{m} \frac{\partial p_j}{\partial g_j}\frac{\partial g_j(\mathbf{x})}{\partial t} \qquad (3.28)\\
&= \sum_{i=1}^{n} \frac{\partial \phi}{\partial x_i}\frac{dx_i}{dt} + \sum_{j=1}^{m} \frac{\partial p_j}{\partial g_j}\sum_{i=1}^{n} \frac{\partial g_j(\mathbf{x})}{\partial x_i}\frac{dx_i}{dt}\\
&= \sum_{i=1}^{n} \left[\frac{\partial \phi}{\partial x_i} + \sum_{j=1}^{m} \frac{\partial p_j}{\partial g_j}\frac{\partial g_j(\mathbf{x})}{\partial x_i}\right] \frac{dx_i}{dt}
\end{aligned}$$

$$= \sum_{i=1}^{n} (-\frac{dx_i}{dt}) \frac{dx_i}{dt} = -\sum_{i=1}^{n} (\frac{dx_i}{dt})^2 .$$

Obviously, $(\frac{dx_i}{dt})^2 \geq 0, \Rightarrow \frac{dE}{dt} \leq 0$. Therefore, $E(\mathbf{x})$ is a Lyapunov function which ensures that the system is completely stable.

Based on the above, the circuit implementation will be straightforward. According to (3.24) and (3.25), variable $\mathbf{x}$ can be represented as voltage, and integrators are used to solve dynamic equation (3.27), where $\frac{\partial g_j}{\partial x_i}$ is constant so that it can be implemented as a resistance. The derivative of the penalty function, $\frac{\partial p_j}{\partial g_j}$, can be represented as current. Thus, a circuit scheme is shown in Figure 3.5, where p-cells are constraint amplifier units, ∫-cells are integrators and the interconnection network is resistive. This network is similar to Kennedy and Chua's model with the exception of the constraint amplifier units. The penalty function in Kennedy and Chua's model can be written as

$$p_j = \begin{cases} 0 & \text{if } g_j(\mathbf{x}) \geq 0 \\ \frac{R_i}{R_0} |g_j(\mathbf{x})|^2 & \text{if } g_j(\mathbf{x}) < 0 \end{cases} \tag{3.29}$$

where R_0, and R_i are resistances. The model proposed here uses the new penalty function of equation (3.23) for the constraint amplifier unit. Comparing (3.23) and (3.29) (see Figure 3.4), it is observed that the former will shrink the equilibrium region efficiently and puts the trajectory back to the boundary quickly. Thus, the network will converge to a stable result more quickly.

As mentioned above, current is used to represent the derivative of the penalty function, i.e.,

$$i_j = \frac{\partial p_j}{\partial g_j} = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \geq 0, \\ k_1 \frac{p+1}{p} |g_i(\mathbf{x})|^{\frac{1}{p}} + 2k_2 |g_i(\mathbf{x})| & \text{if } g_i(\mathbf{x}) < 0. \end{cases} \tag{3.30}$$

Let

$$k_1 = \frac{p}{p+1} \quad \text{and} \quad k_2 = \frac{1}{2}, \tag{3.31}$$

then (3.23) can be rewritten as

$$i_j = \frac{\partial p_j}{\partial g_j} = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) \geq 0, \\ |g_i(\mathbf{x})|^{\frac{1}{p}} + |g_i(\mathbf{x})| & \text{if } g_i(\mathbf{x}) < 0. \end{cases} \tag{3.32}$$

This describes the new constraint amplifier circuit. A problem remains in the implementation of $x^{\frac{1}{p}}$ which requires a complex circuit. By using the mathematical manipulation

$$x^{\frac{1}{p}} = e^{\ln x^{\frac{1}{p}}} = e^{\frac{1}{p}\ln x}, \tag{3.33}$$

the radication is replaced by a simple logarithm, proportion, and exponentiation. The circuit scheme for the modified constraint amplifier unit is shown in Figure 3.6, where T_1 is the I/V inverter, T_2 is the comparator, T_3 is the log unit, T_4 is the proportion unit, T_5 is the antilog unit, and T_6 is the sum unit; in T_4, p of pR is the parameter for the new penalty function. In fact, the circuit consists of operational amplifiers, transistors and resistors with a regularity in configuration so that it can be implemented by VLSI techniques without difficulty.

3.5 Experiments and Simulation Results

According to the approach in Section 3.3 and the architecture proposed in Section 3.4, a simulation was developed in C language. The fourth-order Runge-Kutta numerical integration [WeRe66] was used for simulating the integrator cell in Figure 3.5. The parameters for the penalty function in formulas (3.3) and (3.32) take values: μ = 1 ~ 1.5, p = 10. The typical cases (linear, quadratic and 3rd-order programming) are

considered. The three experimental examples are chosen from the recent literature [Hop84, ChLi84, KeCh88, Ro*et*90] to provide comparative data. In all of the examples, the initial points are located in different directions from the optimal point, including the outside, the inside, and the boundary, of the feasible region to test the capability of the new penalty function. The examples and their results are described below.

3.5.1 Examples

Example 1 [Hop84, ChLi84, KeCh88, Ro*et*90]:

Minimize

$$\phi(x_1, x_2) = c_1 x_1 + c_2 x_2$$

subject to

$$\frac{5}{12}x_1 - x_2 \le \frac{35}{12},$$
$$\frac{5}{2}x_1 + x_2 \le \frac{35}{2},$$
$$-x_1 \le 5,$$

and

$$x_2 \le 5,$$

where

(a) $c_1 = -1, c_2 = -1$;

(b) $c_1 = -1, c_2 = 1$;

(c) $c_1 = 1, c_2 = 1$;

(d) $c_1 = 1, c_2 = -1$.

The simulation results are listed in Table 3.1 and the trajectories for (a) are shown in Figure 3.7. In this Figure 3.7 the solid lines represent the trajectories and the dashed lines represent the boundary of the feasible region. (The same convention is also used in the following examples).

Example 2 [ChLi84, KeCh88, Roet90]:

Minimize

$$\phi(x_1, x_2, x_3) = 0.4x_1 + \frac{1}{2}(5x_1^2 + 8x_2^2 + 4x_3^2) - 3x_1x_2 - 3x_2x_3$$

subject to

$$x_1 + x_2 + x_3 \geq 1$$

and

$$x_1, x_2, x_3 \geq 0.$$

The results are listed in Table 3.1, and the 2-D trajectories on the $x_3 = 0.415$ projection plane are shown in Figure 3.8 for clarity.

Example 3 [ChLi84, KeCh88, Roet90]:

Minimize

$$\phi(x_1, x_2) = 0.4x_1 + x_1^2 + x_2^2 - x_1x_2 + \frac{1}{30}x_1^3$$

subject to

$$x_1 + 0.5x_2 \geq 0.4,$$

$$0.5x_1 + x_2 \geq 0.5,$$

and

$$x_1, x_2 \geq 0.$$

The results are again listed in Table 3.1 and the corresponding trajectories are shown in Figure 3.9.

3.5.2 Discussion

Starting at an initial point, each trajectory in the above examples is moving towards the boundary. It then continuously converges to the equilibrium point on the boundary. The relative errors are 0.03 - 0.08%, 0.02%, and 0.73% for example 1, 2, and 3 (linear, quadratic, and 3th-order nonlinear programming), respectively. For comparison, the results of hardware implementation measurements by Chua and Lin

[ChLi84], and Kennedy and Chua [KeCh88] are also listed in Table 3.1.

3.6 Summary

NLPW and NPW problems have been studied from the aspect of computing accuracy. The conventional penalty function's weakness has been found. With a very big coefficient μ, the term $\mu\alpha(x)$ still may not provide a large enough "penalty" to $\phi^*(x)$ in the "left" neighborhood of the boundary due to the polynomial behavior of $\alpha(x)$. Based on this, a new combination penalty function has been proposed which exhibits good behavior in both the boundary neighborhood and far away regions. By manipulating Equation (3.33), a modified circuit scheme for the new penalty function is given. The circuit can be implemented in analog VLSI. The simulation for the modified Kennedy and Chua model has shown satisfactory results for the experimental examples as evidenced by a significant decrease in the resultant errors. This method can be used in other ANN models and other optimization algorithms as well.

This Chapter has presented an improved ANN model for the optimization with constraints. The other category, the optimization without constraints will be further investigated in Chapters 4 and 5.

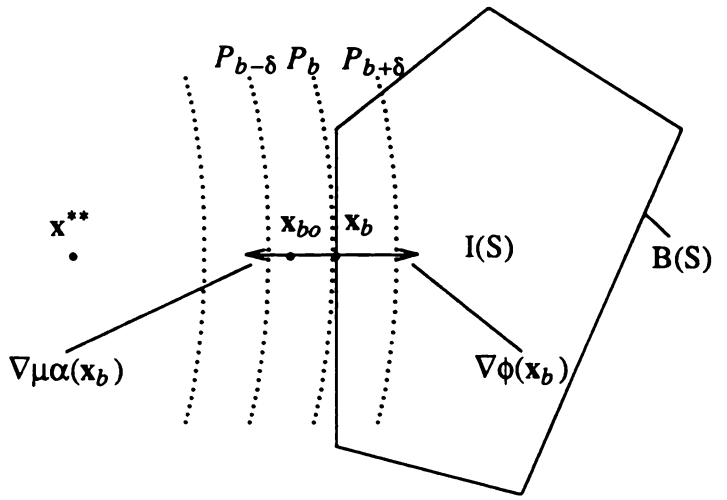


Figure 3.1. Equipotential surfaces in space and their relationship to the feasible region boundary.

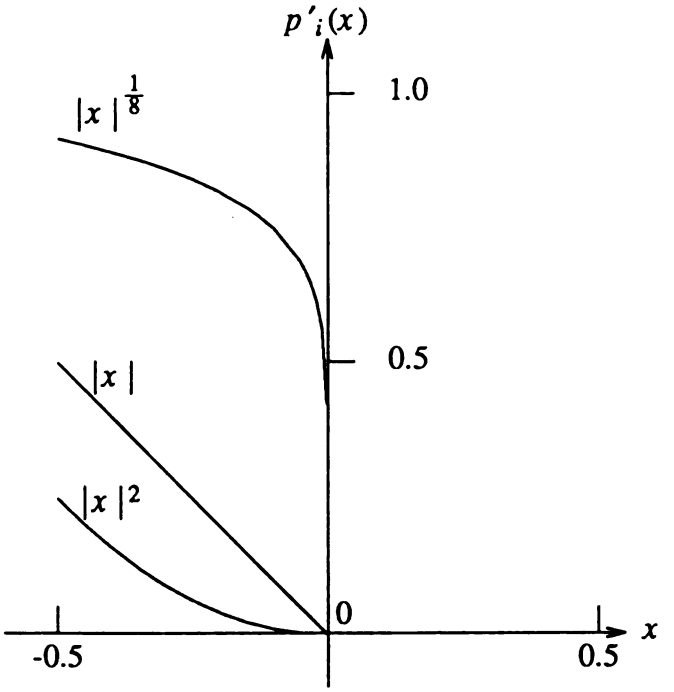


Figure 3.2. Different penalty functions.

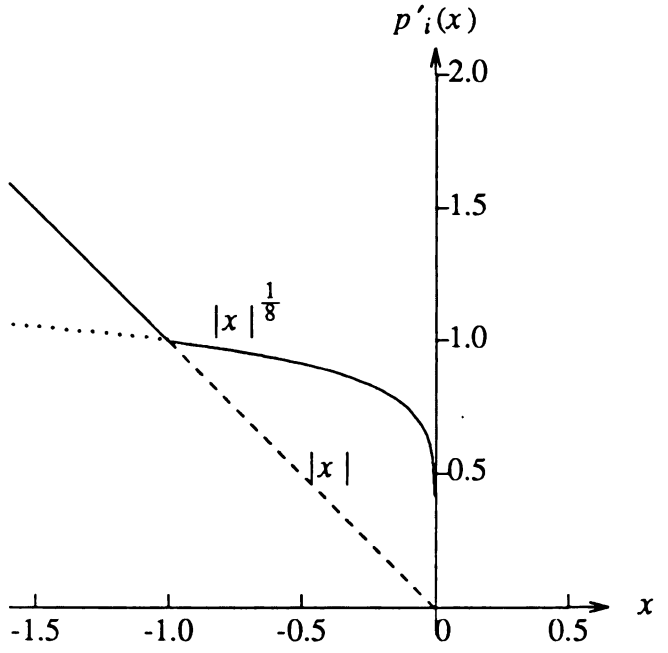


Figure 3.3. A sectioned penalty function.

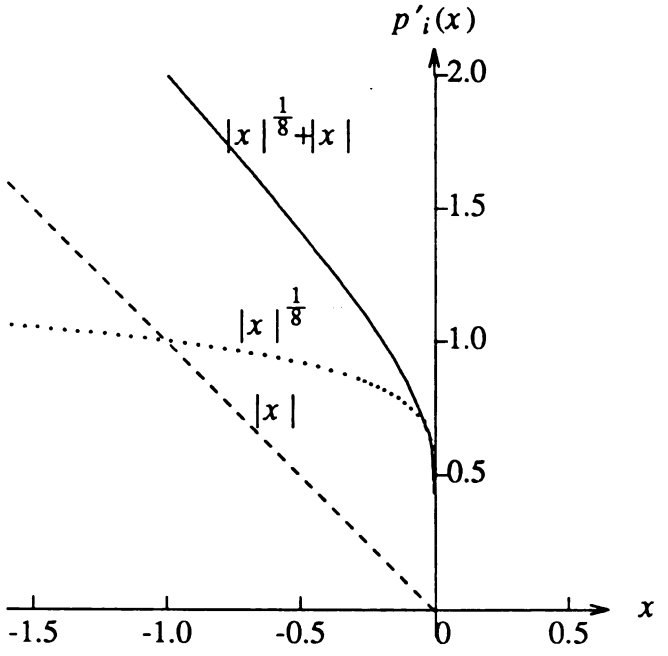


Figure 3.4. A combination penalty function.

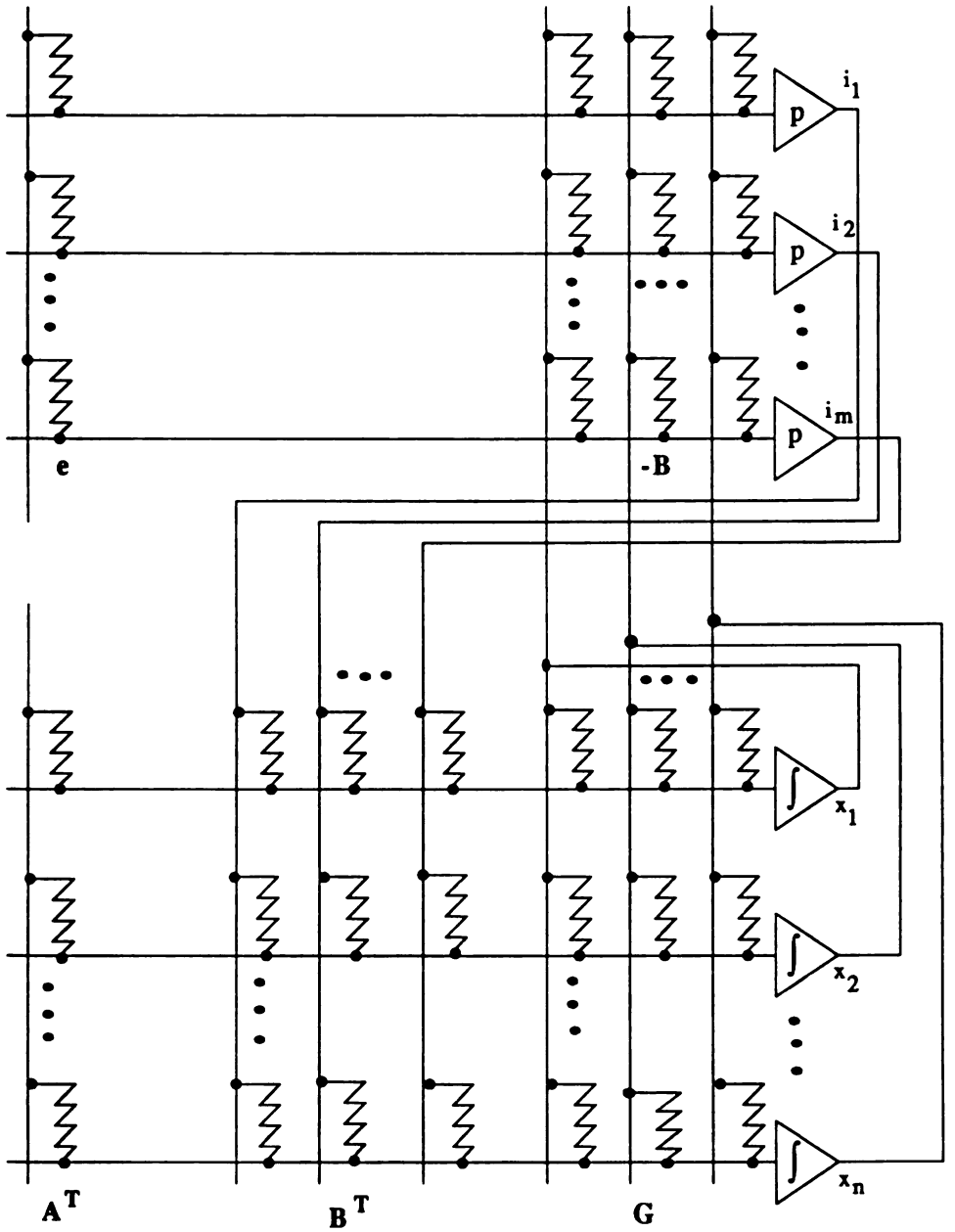


Figure 3.5. Quadratic programming circuit.

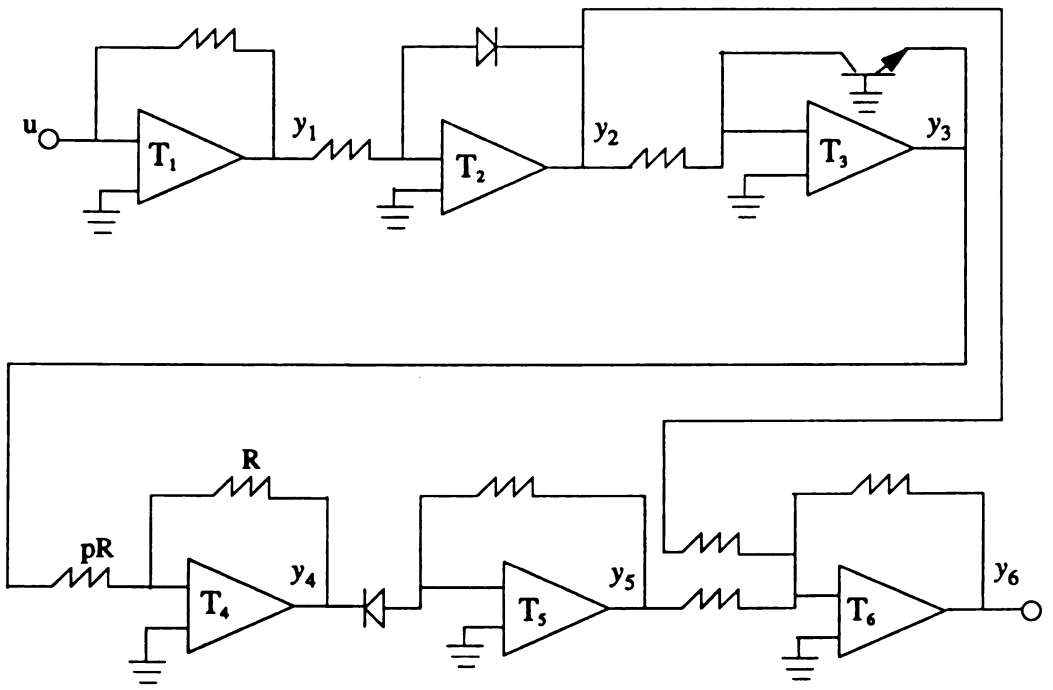


Figure 3.6. A modified scheme for the constraint amplifier unit.

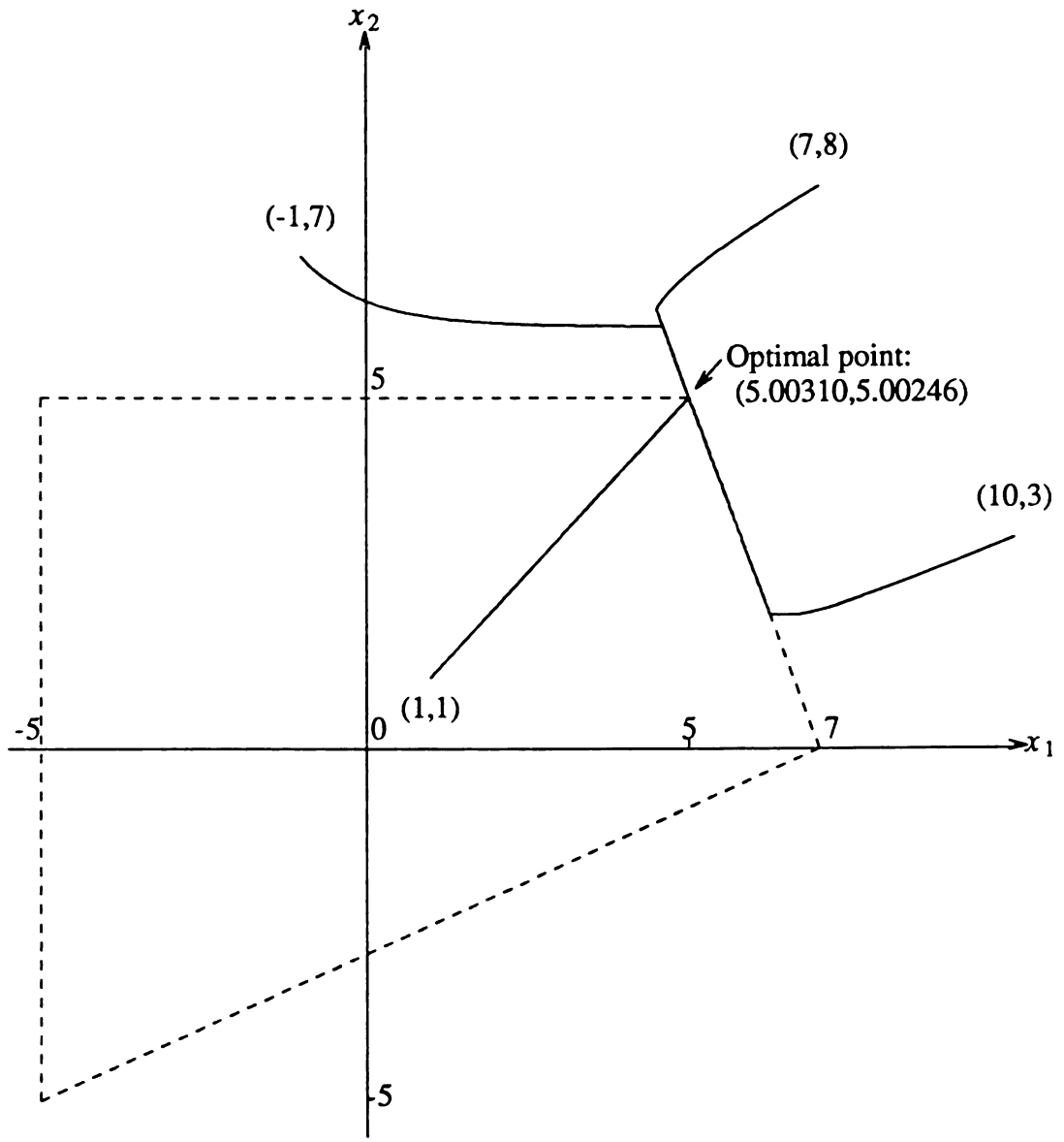


Figure 3.7. Trajectories for example 3.1 (a).

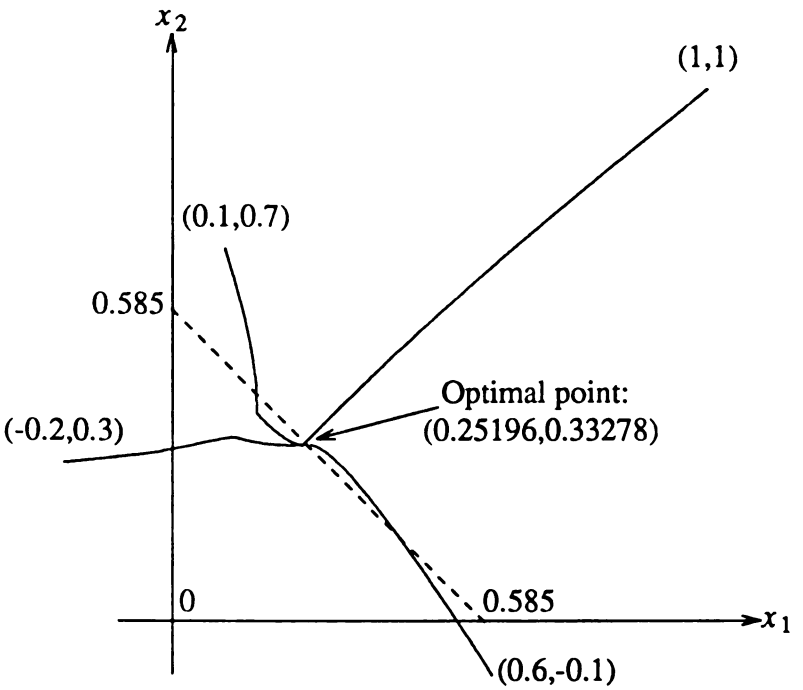


Figure 3.8. Trajectories for example 3.2 (for $x_3 = 0.415$).

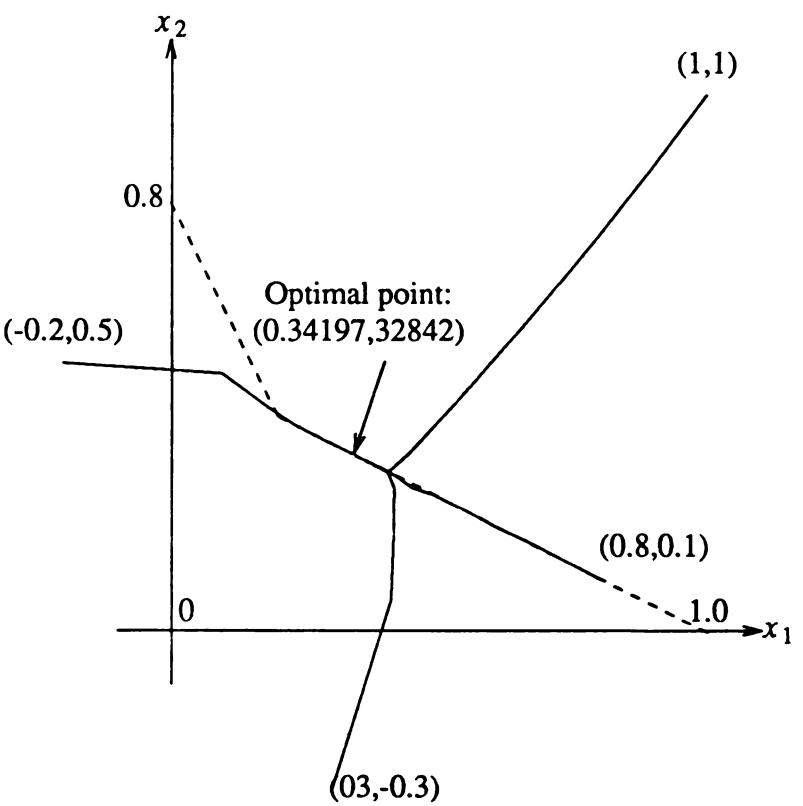


Figure. 3.9. Trajectories for example 3.3.

Table 3.1. Comparison results for linear and nonlinear programming.

Example	Theoretical	Chua & Lin	Kennedy & Chua	Proposed Network
E1a	x1=5.000		x1=5.075	x1=5.00310
	x2=5.000		x2=4.895	x2=5.00246
			error=2.1%	error=0.06%
E1b	x1=7.000		x1=7.088	x1=7.00524
	x2=0.000		x2=-0.017	x2=0.00032
			error=1.26%	error=0.08%
E1c	x1=-5.000		x1=-4.976	x1=-5.00064
	x2=-5.000		x2=4.978	x2=-5.00150
			error=0.48%	error=0.03%
E1d	x1=-5.000		x1=-4.966	x1=-4.99920
	x2=5.000		x2=4.906	x2=5.00091
			error=1.88%	error=0.02%
E2	x1=0.2520	x1=0.258	x1=0.257	x1=0.25196
	x2=0.3328	x2=0.329	x2=0.332	x2=0.33278
	x3=0.4150	x3=0.407	x3=0.412	x3=0.41495
		error=2.38%	error=1.98%	error=0.03%
E3	x1=0.3395	x1=0.336	x1=0.3406	x1=0.34197
	x2=0.3302	x2=0.320	x2=0.3385	x2=0.32842
		error=3.09%	error=2.51	error=0.73%

Chapter 4

Solving Linear System Problems Using An Artificial Neural Network

---

In this Chapter, a new method for solving linear system problems using an artificial neural network is presented. Foremost, a mapping between the solution of linear equations and quadratic minimization is established so that solving linear equations can be viewed as finding the minimum of a quadratic function. The conditions for stability and convergence of the neural network-based dynamic system are proven, providing a mathematical basis for the approach. A feedback ANN model with symmetric or asymmetric connections for optimization is proposed to form an ANN linear equation solver which can be implemented with VLSI technology. Moreover, it is shown that the time complexity of this technique is problem size independent. In addition, other applications of the approach are discussed. These include matrix inversion, determining the stability of a linear control system, and determining matrix singularity.

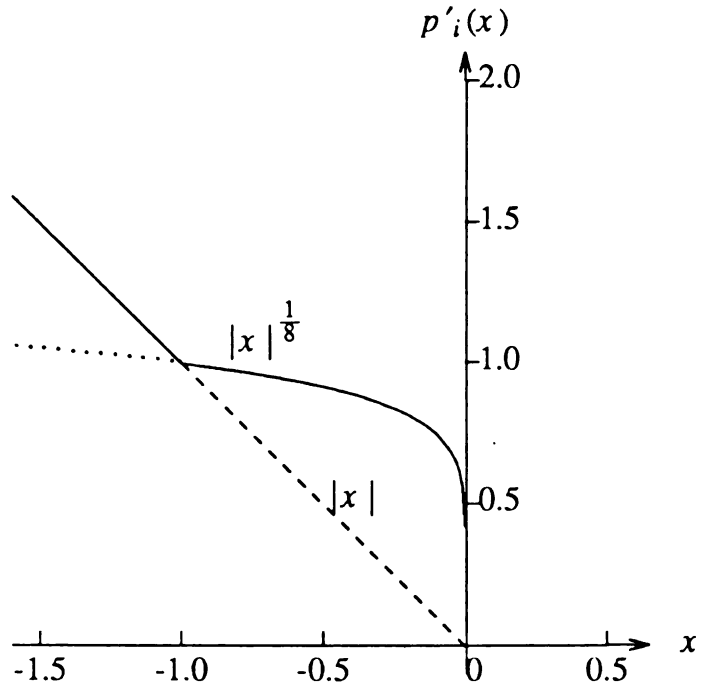


Figure 3.3. A sectioned penalty function.

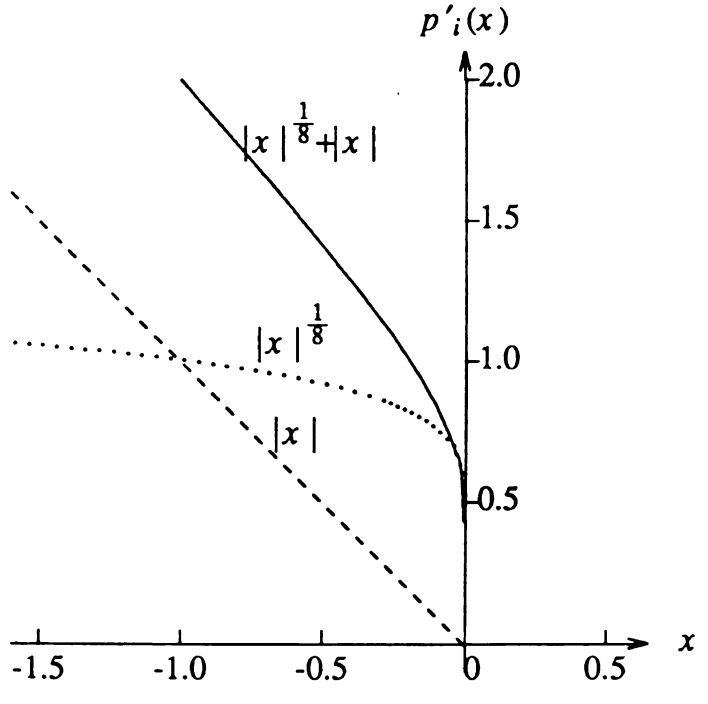


Figure 3.4. A combination penalty function.

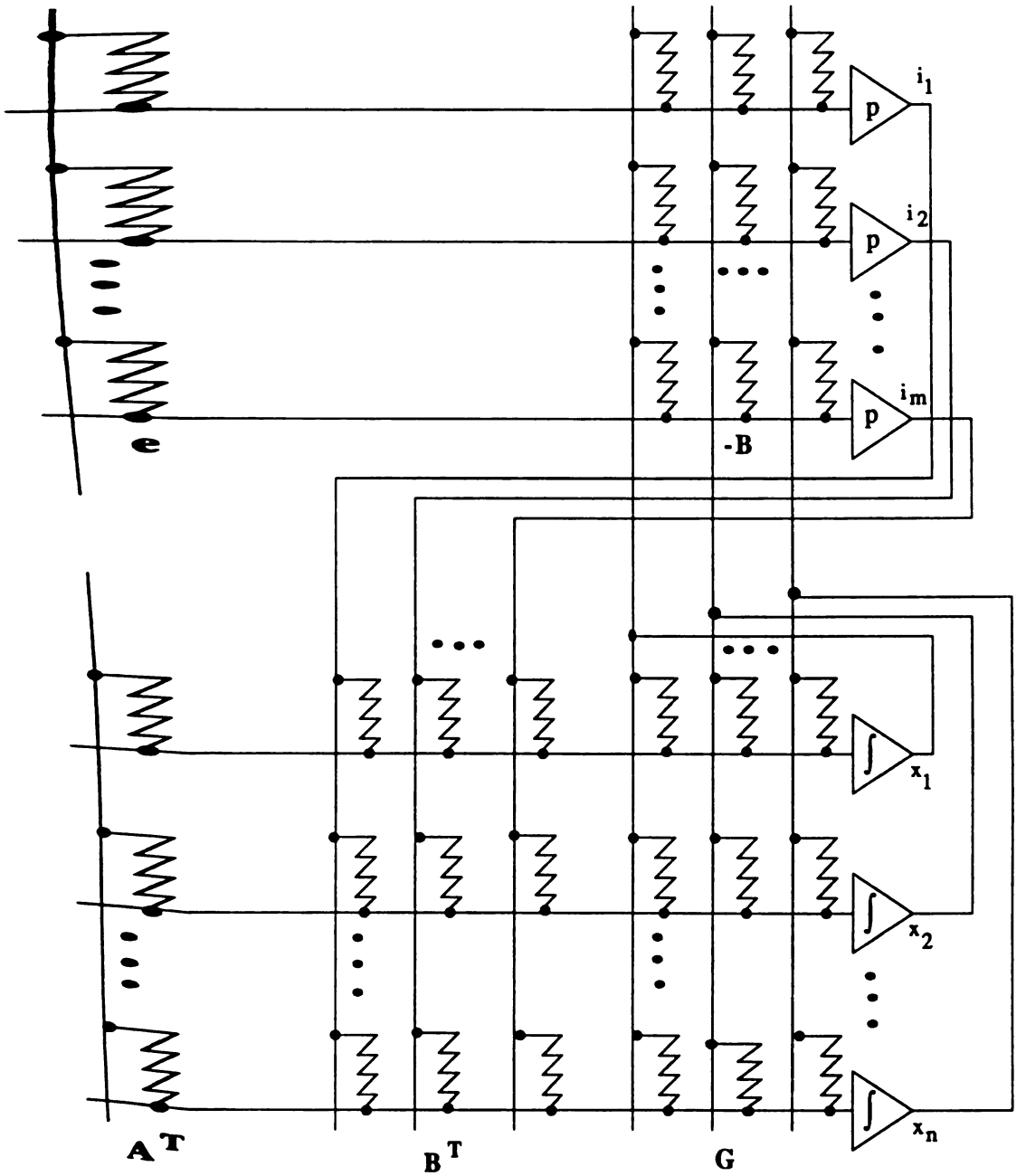


Figure 3.5. Quadratic programming circuit.

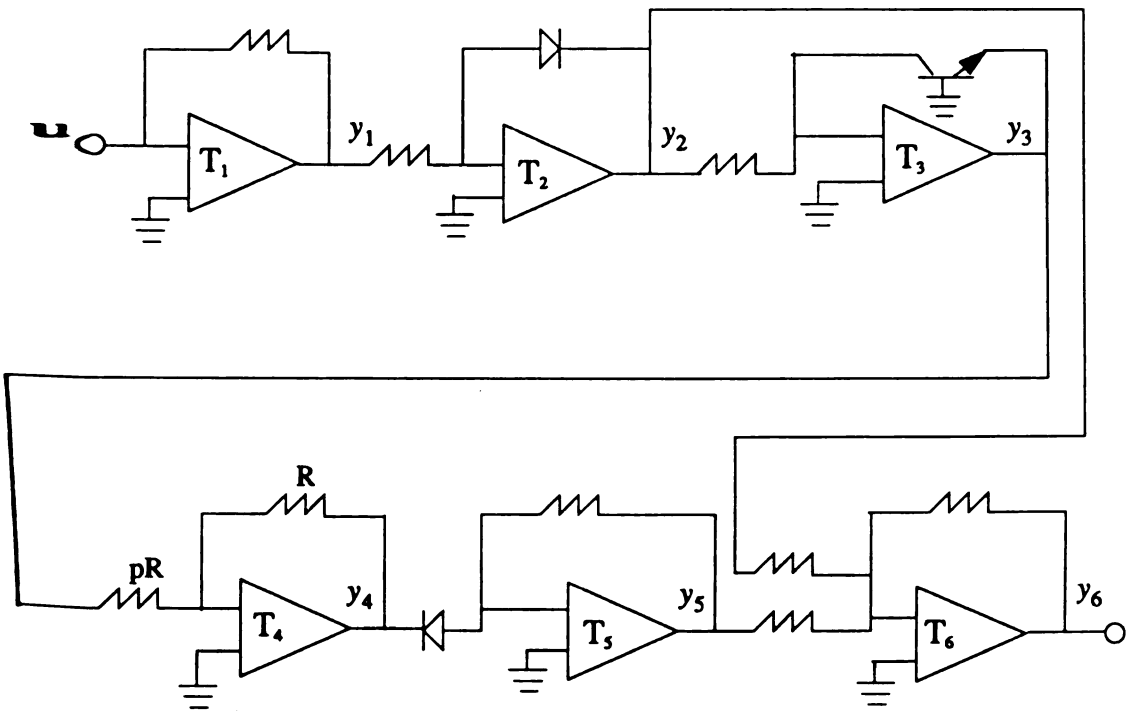


Figure 3.6. A modified scheme for the constraint amplifier unit.

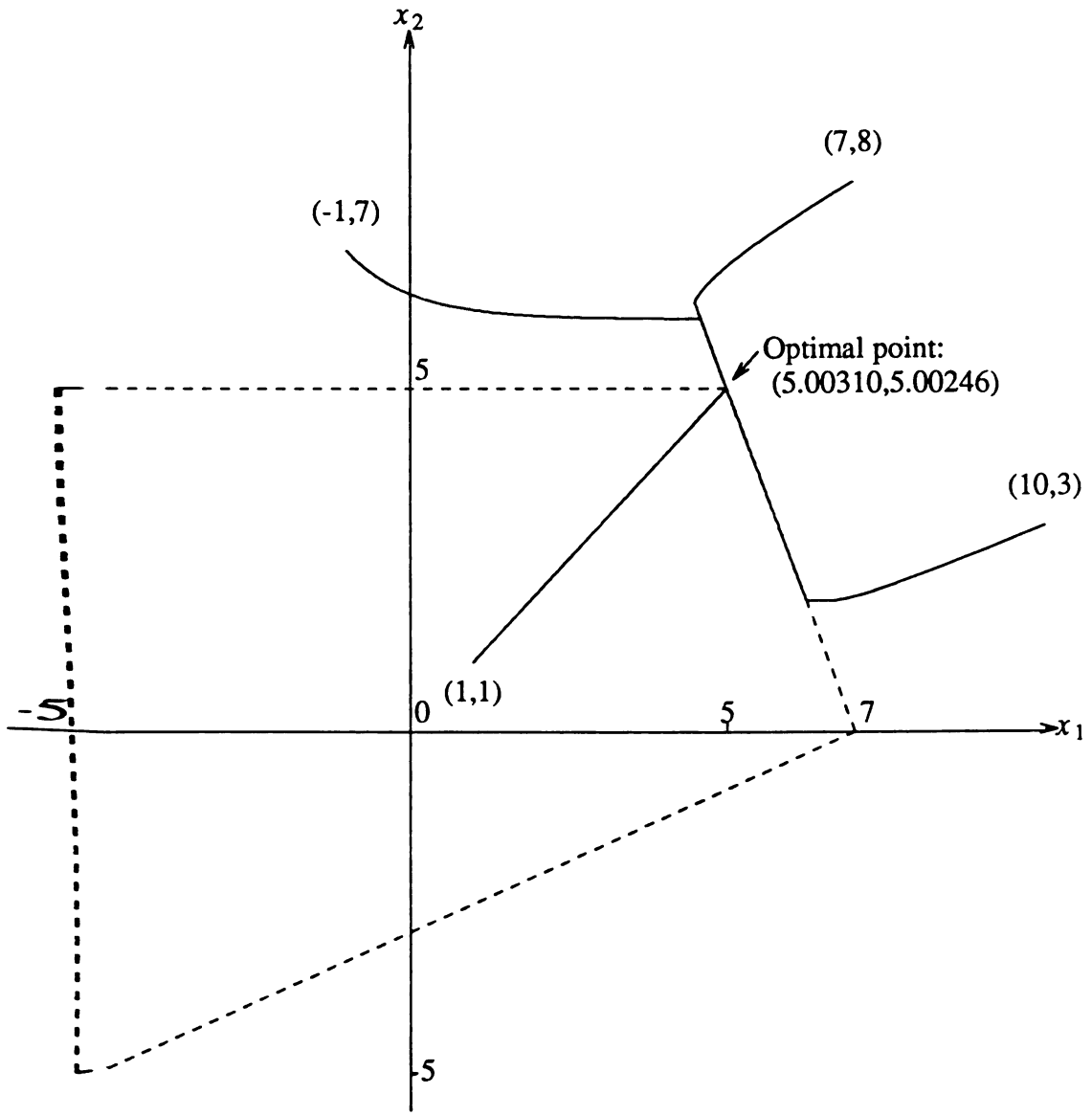


Figure 3.7. Trajectories for example 3.1 (a).

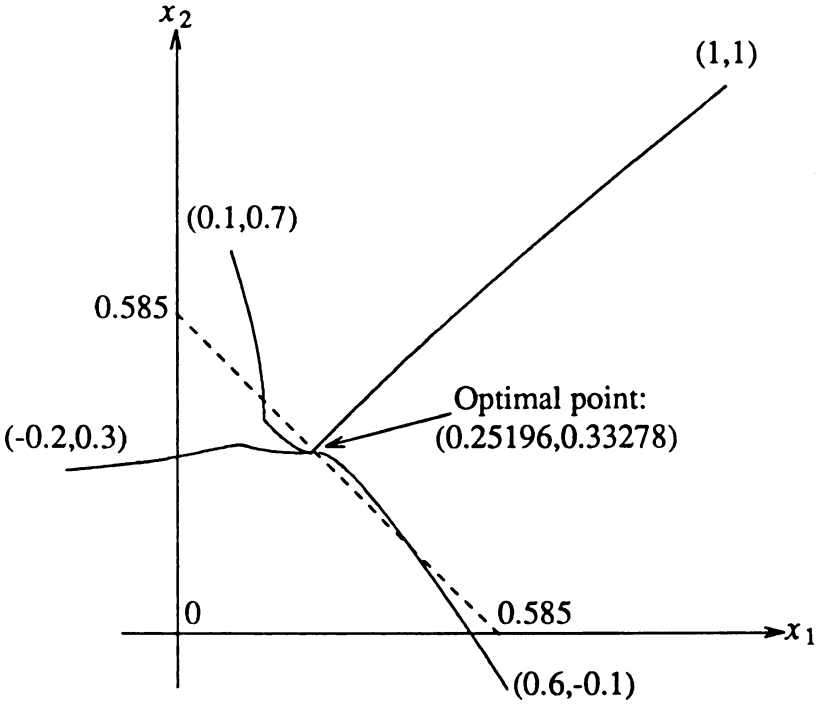


Figure 3.8. Trajectories for example 3.2 (for $x_3 = 0.415$).

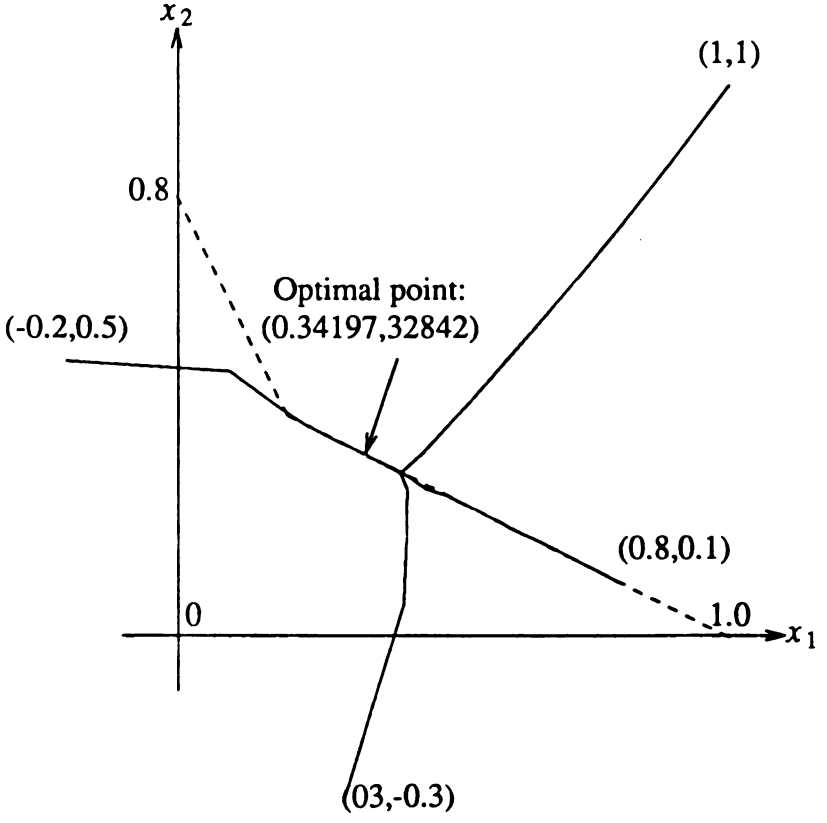


Figure. 3.9. Trajectories for example 3.3.

Table 3.1. Comparison results for linear and nonlinear programming.

Example	Theoretical	Chua & Lin	Kennedy & Chua	Proposed Network
E1a	x1=5.000		x1=5.075	x1=5.00310
	x2=5.000		x2=4.895	x2=5.00246
			error=2.1%	error=0.06%
E1b	x1=7.000		x1=7.088	x1=7.00524
	x2=0.000		x2=-0.017	x2=0.00032
			error=1.26%	error=0.08%
E1c	x1=-5.000		x1=-4.976	x1=-5.00064
	x2=-5.000		x2=4.978	x2=-5.00150
			error=0.48%	error=0.03%
E1d	x1=-5.000		x1=-4.966	x1=-4.99920
	x2=5.000		x2=4.906	x2=5.00091
			error=1.88%	error=0.02%
E2	x1=0.2520	x1=0.258	x1=0.257	x1=0.25196
	x2=0.3328	x2=0.329	x2=0.332	x2=0.33278
	x3=0.4150	x3=0.407	x3=0.412	x3=0.41495
		error=2.38%	error=1.98%	error=0.03%
E3	x1=0.3395	x1=0.336	x1=0.3406	x1=0.34197
	x2=0.3302	x2=0.320	x2=0.3385	x2=0.32842
		error=3.09%	error=2.51	error=0.73%

Chapter 4

Solving Linear System Problems Using An Artificial Neural Network

---

In this Chapter, a new method for solving linear system problems using an artificial neural network is presented. Foremost, a mapping between the solution of linear equations and quadratic minimization is established so that solving linear equations can be viewed as finding the minimum of a quadratic function. The conditions for stability and convergence of the neural network-based dynamic system are proven, providing a mathematical basis for the approach. A feedback ANN model with symmetric or asymmetric connections for optimization is proposed to form an ANN linear equation solver which can be implemented with VLSI technology. Moreover, it is shown that the time complexity of this technique is problem size independent. In addition, other applications of the approach are discussed. These include matrix inversion, determining the stability of a linear control system, and determining matrix singularity.

4.1 Introduction

The linear system is an important mathematical model as it forms a kernel for solving complex problems such as finite element analysis, and the numerical solution of differential equations. In addition, under some conditions, a nonlinear system can be replaced by a linear system for approximation analysis. Therefore, linear system computing techniques play a key role in a wide variety of disciplines. The basic functions in linear system manipulation are finding a solution to simultaneous linear equations [Cra56, Sti63, FoMo67], inverting a matrix, finding the determinant of a matrix [WiRe71, Par80, JoRi82], finding eigenvalues and eigenvectors of a matrix [GoLo83, Joh87, LiZS91]. The recent availability of advanced computers has had a significant impact on all spheres of linear system computation. In the past three decodes, much effort has been devoted to the development of new linear equation solvers for large-scale systems [Sto73, HwCh80, HwCh82, Mol83, CoRo87, BiVo91, Wri91]. The major concern of these effects is to speed up the solution procedure in order to find a feasible result as soon as possible. These techniques typically take advantage of one or more of the following methodologies:

- decomposing the system into small subsystems;
- distributing computation tasks to a parallel architecture;
- increasing data independency;
- reducing communication cost.

All of these methods, however, are fundamentally based on the traditional technique of elimination and separation processes. Their time complexity is size-dependent: $O(n^3)$ for the software approach and $O(n)$ for the hardware approach [HwCh80, Pre*t*86]. In this Chapter, an artificial neural network approach and architecture for linear system manipulation is developed whose time complexity is size-independent.

The Chapter is organized as follows. First, general linear system computing techniques are reviewed. Then, a mapping between the linear equation solution and quadratic minimization is established so that solving linear equations becomes equivalent to finding the minimum of the quadratic function. The conditions for stability and convergence of such a quadratic gradient system are established. Based on these mathematical results, a configuration of an artificial neural network linear equation solver is given. Applications of this approach, such as matrix inversion, determining the stability of a linear control system and determining the singularity of a matrix, are described.

4.2 Methodology Review

Typical linear system computing techniques such as Gaussian elimination, systolic arrays, and MIMD algorithms are summarized as follows.

4.2.1 The Traditional Methods

A set of simultaneous linear equations can be written in matrix form as

$$A\,\mathbf{x} = \mathbf{b} \tag{4.1}$$

where A is n × n matrix, $\mathbf{b}$ and $\mathbf{x}$ are known and unknown column vectors, respectively. When A is nonsingular, there exists an unique solution to the equations. Only this case is considered because most applications deal with this situation.

4.2.1.1 Gaussian Elimination

Gaussian elimination [FoMo67] is the most important method for solving linear system problems and serves as the basis for a variety of other approaches. The technique can be considered as a process of eliminating variables by the elementary

transformation

$$a_{ij}' = a_{ij} - \frac{a_{i1}a_{ij}}{a_{11}} \tag{4.2a}$$

and

$$b_i' = b_i - \frac{a_{i1}b_1}{a_{11}}. \tag{4.2b}$$

This process is repeated until all entries below the main diagonal are zero so that the coefficient matrix is reduced to an upper triangular matrix in the form:

$$\begin{bmatrix} a_{11}' & a_{12}' & \cdot & a_{1n}' \\ 0 & a_{22}' & \cdot & a_{2n}' \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & a_{nn}' \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} b_1' \\ b_2' \\ \cdot \\ \cdot \\ \cdot \\ b_n' \end{bmatrix}. \tag{4.3}$$

Then, back-substitution, the solution vector can be obtained.

4.2.1.2 Gauss-Jordan Elimination

This is a numerical method based on the elementary row and column transformations of column-augmented matrices

$$[A]\cdot[\mathbf{x}\ Y] = [\mathbf{b}\ I] \tag{4.4}$$

where A is a known matrix, Y is the unknown inverse matrix of A, I is the identity matrix, b and **x** are known and unknown column vectors, respectively [WiRe71].

When A is reduced to the identity matrix by a sequence of the elementary row or column transformations, the right-hand of (4.4) becomes the inverse of A.

4.2.1.3 LU Decomposition

Suppose A is an $n \times n$ nonsingular matrix, then A can be uniquely reduced into a upper triangular matrix U by a sequence of equivalent transformations.

$$U = A^{(n-1)} = M_{n-1}M_{n-2}...M_2M_1A. \tag{4.5}$$

Set

$$L = M_1^{-1}M_2^{-1} \cdots M_{n-1}^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ m_{21} & 1 & \cdots & 0 & 0 \\ m_{31} & m_{32} & \cdots & 0 & 0 \\ . & . & \cdots & . & . \\ . & . & \cdots & 1 & 0 \\ m_{n1} & m_{n2} & \cdots & m_{n(n-1)} & 1 \end{bmatrix}, \tag{4.6}$$

then, the LU decomposition of matrix A can be expressed as

$$A = M_1^{-1}M_2^{-1} \cdots M_{n-1}^{-1}U = LU. \tag{4.7}$$

In fact, the LU decomposition can be obtained during a process of Gaussian elimination [Ort87b], that is, the final upper triangular matrix is U, and the multipliers at each step are the entries of the lower triangular matrix L :

$$m_{i1} = \frac{a_{i1}}{a_{11}} \quad \text{for } i = 2, \cdots, n; \tag{4.8a}$$

$$m_{i2} = \frac{a_{i1}'}{a_{22}'} \quad \text{for } i = 3, \cdots, n; \tag{4.8b}$$

$$m_{i3} = \frac{a_{i1}'}{a_{33}'} \quad \text{for } i = 4, \cdots, n; \tag{4.8c}$$

and so on. It should be noted that a_{ij}' is updated at each step of elimination.

The advantages of the LU decomposition of a matrix are:

(1) The solution of a triangular set of equations is quick. The triangularized system can be solved by forward and backward substitution.

(2) The determinant of a matrix is just the product of the diagonal entries of matrix U verified by

$$det(A) = det(LU) = det(L) \cdot det(U) = 1 \cdot det(U) = \prod_{i=1}^{n} u_{ii}. \tag{4.9}$$

(3) The inverse of A can be found by

$$A^{-1} = U^{-1}L^{-1}. \tag{4.10}$$

(4) The solution accuracy of linear equations can be improved by using LU factorization iteratively.

4.2.2 Systolic Arrays

A systolic array is a computing network which consists of a set of interconnected processor elements (PE) each capable of performing some arithmetic operations [Dec89]. The array maximizes the computational concurrence by multiprocessing and pipeline processing techniques very effectively reducing computation time. The major design issues here include mapping a computational algorithm into a systolic array, and specifying the array in terms of communication topology and the arithmetic operation of individual PEs. Driven by many practical applications, the LU decomposition method has been implemented in a systolic array architecture [HwCh82, HwBr84]. Figure 4.1. shows a systolic array for the LU decomposition without pivoting where A is 4×4. In general, $(n-1)^2$ M cells, and $(n-1)$ D cells are required for a $n \times n$ matrix. The array has $(4n-2)$ I/O ports, and its time complexity is $(3n+1)$ [HwCh80].

Another interesting example is a systolic array system for linear state equations

$$\dot{\mathbf{x}}(t) = A'\mathbf{x}(t) + C\,\mathbf{u}(t) \tag{4.11}$$

where $\mathbf{x}(t)$ is the state vector of size n, $\mathbf{u}(t)$ is the input vector of size m, and A', and C are $n \times n$ and $n \times m$ matrices, respectively [JoJe88]. Using the backward Euler method in discrete form [JoRi82], the problem can be decomposed into two simple stages:

(1) compute $B(t_n) = B' + C\,\mathbf{u}(t_n)$, where $B' = \mathbf{x} \cdot \frac{1}{h}$, h is the time increment

$(h = t_n - t_{n-1})$;

(2) solve the linear equations $A\,\mathbf{x}(t_n) = B(t_n)$, where $A = (A' + I \cdot \frac{1}{h})$, I is the identity matrix.

These two stages are mapped to two systolic arrays, one for the matrix-vector multiplication-accumulation, the other one for the Gauss-Jordan elimination. Figure 4.2 illustrates their topologies and PE operations. The latency of this technique is $(n+m-1)$ and $(4n-3)$ for stage 1, and stage 2, respectively. When $n > m$, this structure can overlap the two computations so that the system latency is $(4n-3)$ which has been shown to be the optimal for this problem [JoJe88].

4.2.3 The MIMD Method

The Multi-Instruction and Multi-Data (MIMD) methods can be divided into two categories based on their data distribution scheme: the *row/column distribution* and the *square mesh distribution* [BiVo91]. In general, these methods can only cope with special cases, such as triangular systems, and banded systems, efficiently.

4.2.3.1 Triangular Systems

The grid-oriented structure is used for solving triangular system, because the structure has good load balance property and a low communication overhead [BiVo91]. This method is effective for solving the lower triangular system. Suppose there are p processors which are organized in a $Q \times Q$ mesh $(p = Q^2)$ each with local memory. Each process is executed on one processor, denoted by (s, t), $0 \le s, t < Q$. The communication occurs only among the direct neighbors, *i. e.*, two processors (s, t) and (s', t') are able to communicate if and only if

$$|s - s'| + |t - t'| = 1. \tag{4.12}$$

A theoretical analysis [BiVo91] indicates that the total time required by the parallel algorithm is

$$T_p \leq \frac{n^2}{p} + (4\alpha + 5)n, \tag{4.13}$$

where α is a ratio between communication to computation. Considering the time for the best sequential algorithm

$$T_s = n^2 - n, \tag{4.14}$$

the speedup is

$$\frac{n - 1}{\frac{n}{p} + (4\alpha + 5)}, \tag{4.15}$$

and an efficiency of 50 percent or more can be achieved if $p \leq \frac{n}{4\alpha + 5}$.

4.2.3.2 Banded Linear Systems

A banded linear system is described as

$$A\,\mathbf{x} = \mathbf{b} \tag{4.16}$$

where A is an $n \times n$ nonsingular matrix and $A_{ij} = 0$ if $|i - j| > k$, and k is an integer (usually $1 < k \ll n$). Obviously, the bandwidth of the matrix is $2k + 1$. Such systems arise in a great deal of applications, such as the numerical solution of differential equations and the optimal control.

This method can be thought of as the Gaussian elimination with a certain restricted pivoting strategy [Wri91]. First, the matrix is broken up along the diagonal into a number (say p) blocks (called sub-blocks), and each sub-block is separated by a small dense $k \times k$ block (called separator). Then, A reduced system is formed from the rows of matrix A corresponding to the separators. After the reduced system is solved, and the remaining variables are recovered using data stored during the initial

factorization stage. The experiments for problems with $n = 2000$ to 5000, $k = 2$ to 5 using 2, 4, 8, 16 processors showed that the maximum speedup over a single processor was 4.2 to 5.7 ($p = 16$). When $2 \leq p \leq 4$, the speedup is very low due to the boundary condition operation cost.

The time complexity of the above methods is size-dependent as indicated in Table 4.1.

4.3 A Neural Network Solution to Linear Equations

An neural network linear equation solver based on the optimization network discussed in Chapter 3 first requires the establishment of a mapping between the solution of linear equations and quadratic minimization.

Consider the linear system of equations

$$A\,\mathbf{x} = \mathbf{b} \tag{4.17}$$

where A is an $n \times n$ real coefficient matrix, $\mathbf{b}$ is a real column vector, and $\mathbf{x}$ is a vector of unknowns. A must be nonsingular for a unique solution to exist.

Let $\phi(\mathbf{x})$ be a quadratic function of the form

$$\phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\,\mathbf{x} - \mathbf{b}^T\mathbf{x} \tag{4.18}$$

where A is a real symmetric $n \times n$ matrix, $\mathbf{x}$ and $\mathbf{b}$ are real column vectors.

A minimum point of $\phi(\mathbf{x})$ must satisfy the necessary condition

$$\nabla\phi = \begin{bmatrix} \dfrac{\partial\phi}{\partial x_1} \\ \dfrac{\partial\phi}{\partial x_2} \\ \vdots \\ \dfrac{\partial\phi}{\partial x_n} \end{bmatrix} = 0. \tag{4.19}$$

Correspondingly, the partial derivative of the right hand side of equation (4.31) should also be zero. It can be written in simple matrix form as

$$A\,\mathbf{x} - \mathbf{b} = 0 \tag{4.20}$$

due to the symmetry of A. Equations (4.19) and (4.20) indicate the important fact that the minimum point of a quadratic function must satisfy the corresponding linear equations. In other words, by defining an artificial neural network for the quadratic minimization problem as

$$\begin{aligned} \dot{\mathbf{x}} &= -\nabla\phi(\mathbf{x}) \\ &= -\nabla\left(\frac{1}{2}\mathbf{x}^T A\,\mathbf{x} - \mathbf{b}^T\mathbf{x}\right) \\ &= -(A\,\mathbf{x} - \mathbf{b}), \end{aligned} \tag{4.21}$$

the solution to associated the linear equations is found when the network settles on its minimum point. This network is called an ANN linear equation solver. An important issue for the ANN linear equation solver is ensuring its stability as will be discussed in next section.

4.4 The Relationship to Linear Systems

The following theorems establish conditions for guaranteeing a unique bounded minimum point of a quadratic function for the ANN linear equation solver. Moreover, a general formulation of the ANN linear equation solver is proposed which can cope with both symmetric and asymmetric coefficient matrices.

Theorem 4.1: If A is a positive definite matrix, then the unique minimum point of the quadratic function is the solution to the linear equations.

A formal proof can be found in [Ort87b]. A geometric explanation is given here for clarity. For the two dimensional case, suppose

$$A = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

where $a_{11} > 0$, $a_{22} > 0$.

Substituting the above A and $\mathbf{b}$ into equation (4.18) and manipulating algebraically, $\phi(x_1, x_2)$ can be rewritten as

$$\phi(x_1, x_2) = \frac{1}{2}[x_1\ x_2]\begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - [b_1 b_2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4.22}$$

$$= \frac{1}{2}a_{11}x_1^2 + \frac{1}{2}a_{22}x_2^2 - b_1x_1 - b_2x_2$$

$$= \frac{1}{2}a_{11}[(x_1^2 - 2\frac{b_1}{a_{11}}x_1 + (\frac{b_1}{a_{11}})^2] + \frac{1}{2}a_{22}[(x_2^2 - 2\frac{b_2}{a_{22}}x_2 + (\frac{b_2}{a_{22}})^2] - \frac{1}{2}\frac{b_1^2}{a_{11}} - \frac{1}{2}\frac{b_2^2}{a_{22}}$$

$$= \frac{(x_1 - \frac{b_1}{a_{11}})^2}{(\sqrt{\frac{2}{a_{11}}})^2} + \frac{(x_2 - \frac{b_2}{a_{22}})^2}{(\sqrt{\frac{2}{a_{22}}})^2} - \frac{1}{2}(\frac{b_1^2}{a_{11}} + \frac{b_2^2}{a_{22}}).$$

The image of $\phi(x_1, x_2)$, as shown in Figure 4.3, is an ellipse-parabolic surface whose apex is the lowest point at $(\frac{b_1}{a_{11}}, \frac{b_2}{a_{22}})$, which is exactly equal to $A^{-1}\mathbf{b}$. When $a_{12} = a_{21} \neq 0$ and $a_{11}a_{22} > a_{12}a_{21}$, the statement is still true but the main-axis of ellipse is rotated. For higher dimension cases, a similar image of $\phi(\mathbf{x})$ will be a super ellipse-parabolic surface whose apex coordinates are still the solution to the linear equations.

The positive definiteness of A, however, is a special case. In order to develop a general technique, Theorem 4.1 must be extended as follows for finding the solution to arbitrary linear equations.

Theorem 4.2: If A is an arbitrary nonsingular matrix, then there exists a transformation L such that $L(A)$ is positive definite, and the new equations are equivalent to the original ones.

Proof: Define

$$L(A) = A^T A, \tag{4.23}$$

then

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

in other words, $A^T A$ is symmetric. At the same time,

$$\begin{aligned} \mathbf{x}^T (A^T A)\mathbf{x} &= (\mathbf{x}^T A^T) A\,\mathbf{x} \\ &= (A\,\mathbf{x})^T (A\,\mathbf{x}) \\ &= ||A\,\mathbf{x}||^2 > 0, \quad \text{for } \mathbf{x} \neq 0 \end{aligned}$$

where $||\cdot||$ represents the Euclidean norm. The inequality holds because of the nonsingularity of A. Therefore, $A^T A$ is positive definite.

Multiplying both sides of equation (4.17) by A^T gives:

$$A^T A\,\mathbf{x} = A^T \mathbf{b} \tag{4.24a}$$

$$A^T A\,\mathbf{x} = \mathbf{b}' \tag{4.24b}$$

where

$$\mathbf{b}' = A^T \mathbf{b}. \tag{4.24c}$$

Next, it must be shown that the solution to equation (4.24b) is satisfied with equation (4.17) as well. The solution to (4.24b) can be expressed as

$$\mathbf{x} = (A^T A)^{-1} \mathbf{b}'.$$

Using the relation between $\mathbf{b}'$ and $\mathbf{b}$ in (4.24c), the solution is given by

$$\mathbf{x} = (A^T A)^{-1} A^T \mathbf{b}$$

$$= A^{-1}(A^T)^{-1}A^T\mathbf{b}$$

$$= A^{-1}I\,\mathbf{b}$$

$$= A^{-1}\mathbf{b}. \square$$

Furthermore, when A satisfies a certain condition, the solution can be directly obtained without the transformation. The following theorem explains this situation.

Theorem 4.3: If A is an arbitrary (symmetric or otherwise) nonsingular matrix with eigenvalues whose real part is positive, then there exists a unique stationary point of a quadratic minimization problem which is satisfied by the solution to the corresponding linear equations.

Proof: In artificial neural network theory, a gradient system is often used for finding an equilibrium point

$$\frac{dx_i}{dt} = -\frac{\partial \phi}{\partial x_i} \quad \text{for all } i. \tag{4.25}$$

Associating equations (4.19) and (4.20), the dynamic behavior of a quadratic system can be described by a set of first-order ordinary linear differential equations in matrix form as

$$\dot{\mathbf{x}} = -A\,\mathbf{x} + \mathbf{b}u \tag{4.26a}$$

where

$$u = \begin{cases} 0 & \text{for } t < 0 \\ 1 & \text{for } t \geq 0. \end{cases} \tag{4.26b}$$

When A is asymmetric, the energy function, $\phi(\mathbf{x})$, takes on a modified form of equation (4.18) (see Section 4.5). In this case, however, the mathematical model of the neural network can be directly defined by equation (4.26a).

If the system is relaxed, then using the state transition matrix $\psi(t, \tau)$ [Che84], the solution for $t \geq 0$ can be found as

$$\mathbf{x}(t) = \psi(t, 0)\mathbf{x}_0 + \int_0^t \psi(t, \tau)\mathbf{b}\,d\tau \tag{4.27}$$

where

$$\psi(t, \tau) = e^{-A(t-\tau)}. \tag{4.28}$$

The first term of equation (4.27) comes from the initial condition $\mathbf{x}(0)$, and the second term comes from a constant input $\mathbf{b}u$ (= $\mathbf{b}$). A stable solution exists if the real part of the eigenvalues of matrix $(-A)$ are negative [BaSt70]. In this case, the first term of (4.27) will approach zero as $t \to \infty$. An equilibrium solution to the system will be determined only by the second term of (4.27). Taking the limit of (4.27) as $t \to \infty$ yields

$$\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}(0) \lim_{t\to\infty} e^{-At} + \lim_{t\to\infty} \int_0^t e^{-A(t-\tau)}\mathbf{b}\,d\tau \tag{4.29}$$

$$= 0 + A^{-1}\mathbf{b} \lim_{t\to\infty} e^{-A(t-\tau)} \Big|_0^t$$

$$= A^{-1}\mathbf{b}(e^0 - e^{-\infty})$$

$$= A^{-1}\mathbf{b}.$$

In other words, the equilibrium point of the differential equations must satisfy the corresponding linear equations.

Moreover, it is easy to show that the signs of the eigenvalues of A are opposite to those of $-A$, because

$$A\mathbf{x} = \lambda\mathbf{x} \;\Rightarrow\; -A\mathbf{x} = -\lambda\mathbf{x} \;\Rightarrow\; (-A)\mathbf{x} = (-\lambda)\mathbf{x} \tag{4.30}$$

where λ is an eigenvalue of A.

Therefore, A with eigenvalues whose real part is positive will guarantee that a unique solution to the linear equation system can be found. □

4.5 Architecture

A theoretical basis for an ANN linear equation solver has been presented in the previous sections. In what follows, an architecture for the ANN linear equation solver is proposed.

Artificial neural network computing is a dynamical process which is mapped into a network architecture with an associated dynamical feature. The computing is performed by the network's inherent convergent behavior. There are three types of ANN architectures: feedback, feedforward, and cellular models. The feedback neural networks are organized in a simple and regular form and they can easily express the equality relation of a dynamical system. Therefore, they are the logical choice for building the linear equation solver. As mentioned in Section 4.3, the function of the linear equation solver is equivalent to that of an unconstrained optimization network. The ANN solver can be constructed by a feedback minimization network. The present models, however, work well only with the symmetric feedback connections [TaHo86, KeCh88]. The stability property of a system, determined by matrix A, is the key issue for design of the ANN architecture. As discussed in Section 4.4, matrix A in the case of Theorems 4.1 and 4.2 (after transformation) is a positive definite matrix whose eigenvalues are positive real numbers [Che84], whereas eigenvalues of matrix A in Theorem 4.3 can be written as $\lambda_i = \alpha_i + j\beta_i$, where $\alpha_i > 0$, $j = \sqrt{-1}$. Theorems 4.1 and 4.2 can be regarded as a special case of Theorem 4.3 where $\beta_i \equiv 0$. As in Theorem 4.3, the architecture proposed here is not restricted to matrices with symmetric connections. This significantly broadens its range of applications.

Figure 4.4 shows a configuration of the linear equation solver composed of integration units, a resistive network, and feedback links. The resistive network is organized according to equation (4.26a). Thus, the following relation is held for each integrator (∫-cell):

$$\frac{dx_i}{dt} = -\left(\sum_{j=1}^{n} a_{ij}x_j - b_i \cdot 1\right) \tag{4.31}$$

where x_i is the voltage at node i, and the 1 in the term $b_i \cdot 1$ is a DC source voltage (1V). The terms $a_{ij}x_j$ and $b_i \cdot 1$ are thus interpreted as current terms. The negative summation is implemented by connecting all current inputs to the negative (minus) terminal of the integrator. Let the normalization values of current be 0.1 *ma*, then the value of the resistors can be calculated as

$$R_{ij} = \frac{10}{|a_{ij}|} k\Omega \tag{4.32a}$$

or

$$R_{bi} = \frac{10}{|b_i|} k\Omega \tag{4.32b}$$

where R_{ij} and R_{bi} are resistors representing values in matrix A and vector **b**, respectively. The lower ends of R_{ij} and R_{bi} are connected to the input of integrator i, whereas the connections of their upper ends depend on the values of a_{ij} and b_i as indicated in Table 4.2.

In order to verify the stability and convergence of this network (without the symmetry restriction), a shift transformation is carried out for the dynamical system as described by equation (4.26a).

$$\begin{aligned} \dot{\mathbf{x}} &= -(A\,\mathbf{x} - \mathbf{b}) \qquad (4.33a) \\ &= -(A\,\mathbf{x} - A\,\mathbf{x}^*) \\ &= -A\,(\mathbf{x} - \mathbf{x}^*) \end{aligned}$$

$$= -A\,\mathbf{y}$$

where

$$\mathbf{y} = \mathbf{x} - \mathbf{x}^* \tag{4.33b}$$

and

$$A\,\mathbf{x}^* = \mathbf{b}.$$

Because of the constant (unknown) vector $\mathbf{x}^*$, the time derivative of $\mathbf{y}$ must satisfy

$$\dot{\mathbf{y}} = \dot{\mathbf{x}} = -A\,\mathbf{y}. \tag{4.34}$$

When the condition of Theorem 4.3 is satisfied for A, there must exist a unique solution P for the Lyapunov equation

$$PA + A^T P = Q \tag{4.35}$$

where Q is an arbitrary positive definite matrix, and P is a positive definite matrix (the solution) [BaSt70].

Define the energy function $E(\mathbf{y})$ as

$$E(\mathbf{y}) = \mathbf{y}^T P\,\mathbf{y}. \tag{4.36}$$

It is apparent that $E(\mathbf{y}) > 0$ for $\mathbf{y} \neq 0$ because of the positive definiteness of P.

The derivative of the energy function for $\mathbf{y} \neq 0$ can be found as

$$\begin{aligned} \dot{E}(\mathbf{y}) &= \mathbf{y}^T P\,\dot{\mathbf{y}} + \dot{\mathbf{y}}^T P\,\mathbf{y} \\ &= -\mathbf{y}^T PA\,\mathbf{y} - \mathbf{y}^T A^T P\,\mathbf{y} \\ &= -\mathbf{y}^T (PA + A^T P)\mathbf{y} \\ &= -\mathbf{y}^T Q\,\mathbf{y} < 0. \end{aligned} \tag{4.37}$$

The above equation indicates that the energy function is always decreasing with time. Therefore, the energy function is a global Lyapunov function which guarantees that the network is completely stable and there exists a unique equilibrium point at

$$\mathbf{y}_e = 0. \tag{4.38}$$

Using equation (4.33b), the corresponding equilibrium point for $\mathbf{x}$ is

$$\begin{aligned} \mathbf{x}_e &= \mathbf{y}_e + \mathbf{x}^* \\ &= 0 + \mathbf{x}^* \\ &= A^{-1}\mathbf{b}. \end{aligned} \tag{4.39}$$

This means that when the network arrives at the equilibrium state, the output $\mathbf{x}$ is the exact solution of the linear equations.

4.6 Other Applications

The technique for solving linear equations is of fundamental importance to many computing tasks. The method described in the previous sections can be further extended to other applications.

4.6.1 Matrix Inversion

If matrix A satisfies the conditions of Theorems 4.1 or 4.3, then the inverse of A can be found by the following procedure.

Let

$$A^{-1} = (\mathbf{y}_1\ \mathbf{y}_2 \cdots \mathbf{y}_n),$$

then

$$AA^{-1} = A(\mathbf{y}_1\ \mathbf{y}_2 \cdots \mathbf{y}_n) = I = \begin{bmatrix} 1 & 0 & 0 & .. & 0 \\ 0 & 1 & 0 & .. & 0 \\ 0 & 0 & 1 & .. & 0 \\ . & . & . & .. & . \\ . & . & . & .. & . \\ 0 & 0 & 0 & .. & 1 \end{bmatrix} \tag{4.40}$$

where $\mathbf{y}_i$ is the *ith* column vector of A^{-1} and I is the identity matrix. The above

equations can be rewritten in separate form as

$$A\,\mathbf{y}_i = \mathbf{e}_i \tag{4.41}$$

for $i = 1, 2, ..., n$, where $\mathbf{e}_i$ is a base vector (1 at the *ith* entry and 0 at all others). Therefore, finding the inverse of A is equivalent to solving equation (4.41) n times with a base vector as the right hand side.

As an extension of the approach, if matrix A is negative definite or $Re(\lambda_i) < 0$ for all i, where λ_i is an eigenvalue of A, then $-A$ must satisfy the conditions of Theorems 4.1 or 4.3. As a consequence, the inverse of A can be directly found by

$$A^{-1} = -(-A)^{-1} \tag{4.42}$$

where $(-A)^{-1}$ is obtained from the ANN linear equation solver.

4.6.2 Determining the Stability of a Linear Control System

Criteria such as Routh-Hurwitz and Nyquist are most often used to determine the stability of a linear control system [Che84]. For large-scale systems, the process involved in using either of the above criteria becomes very time-consuming due to the computational complexity. A completely parallel approach based on Theorem 4.3 is proposed here. From control theory, a linear system is stable in the sense of Lyapunov if and only if all eigenvalues of A have no positive real parts. (For zero eigenvalues, the order of the Jordan blocks should be one [Che84].) Without loss of generality, let the input be zero. Then the control system can be written as

$$\begin{cases} \dot{\mathbf{x}} = -A\,\mathbf{x} \\ \mathbf{y} = \mathbf{c}\mathbf{x} + \mathbf{e} \end{cases} \tag{4.43}$$

where $\mathbf{x}$ is a vector of state variables, $\mathbf{y}$ is a vector of outputs, $\mathbf{c}$ is a constant matrix, and $\mathbf{e}$ is a constant vector. Thus, the problem can be converted to solving the related linear equations. Using equation (4.27) and starting at any $\mathbf{x}(0) \neq 0$, the final state, $\mathbf{x}(\infty)$, can be expressed as

$$\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}(0)\lim_{t\to\infty} e^{-At} + \lim_{t\to\infty}\int_0^t e^{-A(t-\tau)}0d\tau \tag{4.44}$$

$$= \mathbf{x}(0)e^{-A(t-\tau)} + 0$$

$$= \begin{cases} 0 \text{ or finite value} & (\textit{stable}) \\ \infty & (\textit{unstable}). \end{cases}$$

Correspondingly, the linear equations

$$A\,\mathbf{x} = 0 \tag{4.45}$$

can be solved with a starting point $\mathbf{x}(0) \neq 0$. When a bounded solution is found the system is stable and matrix A is called a stable matrix; otherwise, the system is unstable.

4.6.3 Determining the Singularity of a Matrix

Since singular matrices have different properties than nonsingular matrices, it is desirable to determine whether or not a matrix is singular. For example, when A represents a linear transformation, a nonsingular A always indicates 1-to-1 mapping whereas a singular A indicates m-to-1 mapping. The traditional methods for determining the singularity of a matrix include the LU decomposition and the similarity transformation. The former calculates the determinant of matrix A by

$$det(A) = det(LU) \tag{4.46}$$

$$= det(L)\cdot det(U)$$

$$= 1\cdot det(U)$$

$$= det(U)$$

where L is a lower triangular matrix with a unit diagonal, and U is an upper triangular matrix. The time complexity for this method is $O(\frac{1}{3}n^3)$ for the software approach,

and $O(3n - 5)$ for the hardware approach [HwBr84, Pr*et*86].

The purpose of the similarity transformation is to find the eigenvalues of A. If there exist any zero eigenvalues, then the matrix is singular. The computational cost for this is as high as $O(n^5)$ [Pr*et*86].

The singularity of a matrix can be distinguished by the following two operations:

(i) Determine whether matrix A (or $-A$) is stable. If yes, go to step (ii); otherwise, stop. (In this case the method fails because eigenvalues of A lie in both right and left halves of the complex plane so that the final state is not bounded).

(ii) Arbitrarily choose some $\mathbf{b} \neq 0$. Solve the set of equations $A\mathbf{x} = \mathbf{b}$ (or $-A\mathbf{x} = \mathbf{b}$) for two different starting points. If a unique solution can be found, then matrix A is nonsingular. If the solution is dependent on the initial point, or no solution can be found (*i.e.*, no equilibrium point so that the state variables keep changing: $|x_i^{(k+1)} - x_i^{(k)}| > \varepsilon$ for all i, where ε is a convergence limit, (k) is an iteration index), then the matrix A is singular. For singular matrix A, the solution situation depends on vector $\mathbf{b}$; if augmented matrix $[A\ \mathbf{b}]$ has the same rank as matrix A, then there exist multiple solutions. Otherwise ($rank[A\ \mathbf{b}] > rank[A]$) there is no solution. Thus, this technique not only gives a judgement as to the singularity, but also indicates the rank relationship between $[A\ \mathbf{b}]$ and $[A]$, that is, for the multi-solution case, $rank(A) = rank(A\ b)$, and for the no-solution case, $rank[A\ \mathbf{b}] > rank[A]$.

For clarity, the above procedures for the different applications and their relationship are summarized in Figure 4.5. In fact, this approach is valid for the single-side eigenvalues "distribution". If all eigenvalues of A lie in the right half of the complex plane, then A is used; if all eigenvalues of A lie in the left half of the complex plane, then $-A$ is used.

4.7 Simulation Results

Experiments include both software and hardware simulations. The former attempts to verify the correctness of the approach, and the latter provides evidence of the size-independent property of the ANN solver.

4.7.1 Verification

A simulation is designed in the C language and based on the architecture of Figure 4.4. Integrators are simulated by a fourth-order Runge-Kutta numerical integration (explicit formulation), and the resistive networks are simulated by 1-D and 2-D arrays.

4.7.1.1 Linear Equation Solution

Experimental examples are arbitrarily chosen for the three cases in Section 4.4 from $n = 2$ to $n = 20$. The ANN linear equation solver does indeed find a unique solution for every example no matter what initial point is chosen. Table 4.3 illustrates three examples associated with three situations in Section 4.4 for matrix A. The convergence is determined by

$$| x_i^{(k+1)} - x_i^{(k)} | < \varepsilon \quad \text{for all } i \tag{4.47}$$

where $x_i^{(k)}$ is the value of x_i at the kth iteration. $\varepsilon = 0.1E{-}7$ is used for all the examples. The time increment dt used for the Runge-Kutta integration is 0.05 for example 1 and 2, and 0.001 for example 3. The simulation time is defined as

$$T = dt \cdot N \tag{4.48}$$

where N is the number of iteration steps required for convergence. The experiments show that T has an obvious relationship to the initial point; the further the initial point is from the equilibrium point, the more iterations are required. For examples 1 to 3, T is 18.05, 7.0 and 14.67, respectively.

In order to measure the accuracy, an error e is defined as

$$e = \sum_{i=1}^{n}(\sum_{j=1}^{n}a_{ij}x_j - b_j)^2 \tag{4.49}$$

which represents the sum of the square of the difference between the left hand side and the right hand side for each equation. In the ideal case, e is zero. The error curves for the three examples are presented in Figure 4.6. Given enough iterations, the error approaches zero for all practical purposes. Four trajectories for example 1 are shown in Figure 4.7, indicating that the solution is initial-point independent.

4.7.1.2 Inverse Matrix Computation

Table 4.4 lists the matrix inversions (examples 4 and 5) where matrix E is a measure of the error determined by

$$E = A^{-1}A - I \tag{4.50}$$

where I is the identity matrix. Ideally, E should be a zero matrix. The simulation shows that all entries of E are very close to zero (0.000000001 to 0.000001).

4.7.1.3 Stability

Two examples of the stability judgement are presented in Table 4.5. Example 6 always results in a bounded solution using the technique described in Section 4.6. This indicates that the system is stable. Example 7 results in infinite solutions, which indicates the system is unstable. The corresponding eigenvalues are calculated and listed in Table 4.5 for checking consistency.

4.7.1.4 Singularity

Table 4.6 gives the results of two example singularity calculations (No. 8 and 9). Figure 4.8 shows the trajectories of a multi-solution for the following equation with a

singular matrix

$$\begin{bmatrix} 1.0 & 0.5 \\ 4.0 & 2.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 4.0 \end{bmatrix}.$$

It is obvious that all the solutions are on a co-linear pattern indicated by a dashed line in Figure 4.8. Moreover, it is easy to shown that eigenvectors associated with eigenvalues $\lambda = 0$ and $\lambda = 3$ of this matrix are

$$x_2 = -2x_1$$

and

$$x_2 = 4x_1,$$

respectively, indicated by dotted lines in Figure 4.8. Some interesting observations are gleaned from this experiment. The line for all the solutions is parallel to eigenvector $x_2 = -2x_1$. Moreover, all the trajectories are straight lines and they are parallel to eigenvector $x_2 = 4x_1$.

4.7.2 Hardware Simulation

The mathematical model has verified the correctness of the proposed approach. The objective of the hardware simulation is to check the time-property of the ANN linear equation solver architecture proposed in Section 4.5. SPICE, a computer-aided design tool for circuit analysis is used to verify the ANN linear equation solver at the component level. A simulated solver is built with 1, 2, 3, 5, and 10 neurons, respectively. To provide comparison, all data are normalized so that starting points are located at the origin and solutions are within (-1, 1). The results indicate that the resolution time T_s is in the same range for all the examples. T_s depends only on the time constant of the integrator, τ ($= C_0 R_0$). For example, when $\tau = 2\mu s$, $T_s = 2.25ms$ to $2.45ms$; when $\tau = 0.2\mu s$, $T_s = 0.223ms$ to $0.247ms$. The slight deviation in T_s is that the distance from the staring point to an equilibrium point is not identical. Figures 4.9,

4.10, and 4.11 show the curves of transient analysis for the simulations with $\tau = 2\mu s$. The preliminary conclusion here is that the solution time is size-independent.

4.7.3 Hardware Complexity

In the development of computing technology, the computing time and the device space are always associated with each other. An estimate of the hardware complexity of the ANN linear equation solver is derived as follows.

Let the size of a system be n, then the total numbers of resistors and integrators in the architecture shown in Figure 4.4 are $n^2 + n$, and n, respectively. Assume the average resistance of resistors is $50K\,\Omega$, and the routing area is 40% of the circuit area [AlHo87], then the chip area of the solver for a CMOS implementation is

$$S = 1.4[(n^2 + n)a + nb] \tag{4.51}$$

where a is the unit area for a resistor ($a = 6.25\times10^{-3}mm^2$), and b is the unit area for an integrator ($b = 1.5\times10^{-2}mm^2$) [AlHo87, Rei87]. For clarity, a relationship between the chip area and the size n is illustrated in Figure 4.12.

Connectivity is another important issue in hardware implementation. The topology of an architecture, the connection order of a component, and the number of crossovers in a circuit are three key factors of the connectivity. As shown in Figure 4.4, the connection pattern of the solver takes a matrix form which can be implemented in a "bus" structure so that connections among components are significantly simplified. It is easy to show that the number of crossovers of this network is

$$N_{cross} = n^2 - 1. \tag{4.52}$$

This is another advantage of a feedback network paradigm as it can be shown that N_{cross} for a three-layer feedforward network is on the order $O(n^4)$. Moreover, routing network is straightforward. As shown in Figure 4.4, there are three bus lines: horizontal (H), vertical (V), and feedback (F). H lines are on layer 1, V lines and F lines are

on layer 2. On each crossover, an isolated "stamp" is made between H and V. For circuit components, each resistor is on layer 1 with connections to H and V, and each integrator is on layer 1 with connections to F and H.

The network requires n I/O pins for operands and additional pins for voltage sources, control signals, and ground.

4.8 Summary

A new approach to solving linear systems using an artificial neural network has been proposed. A mapping between the linear equation solution and a quadratic minimization problem without constraints has been established. Three theorems in Section 4.4 guarantee the stability and convergence of the quadratic system so that a unique accurate solution can be obtained. Correspondingly, an ANN architecture for the solver is given which can be implemented in VLSI technology. Because of the inherent features of dynamic convergence and parallel processing in ANNs, the time complexity of the approach is size-independent. The approach can also be used for finding the inverse of a matrix, determining the stability of a linear control system, and determining the singularity of a matrix. Simulation experiments have verified the theoretical correctness of the technique.

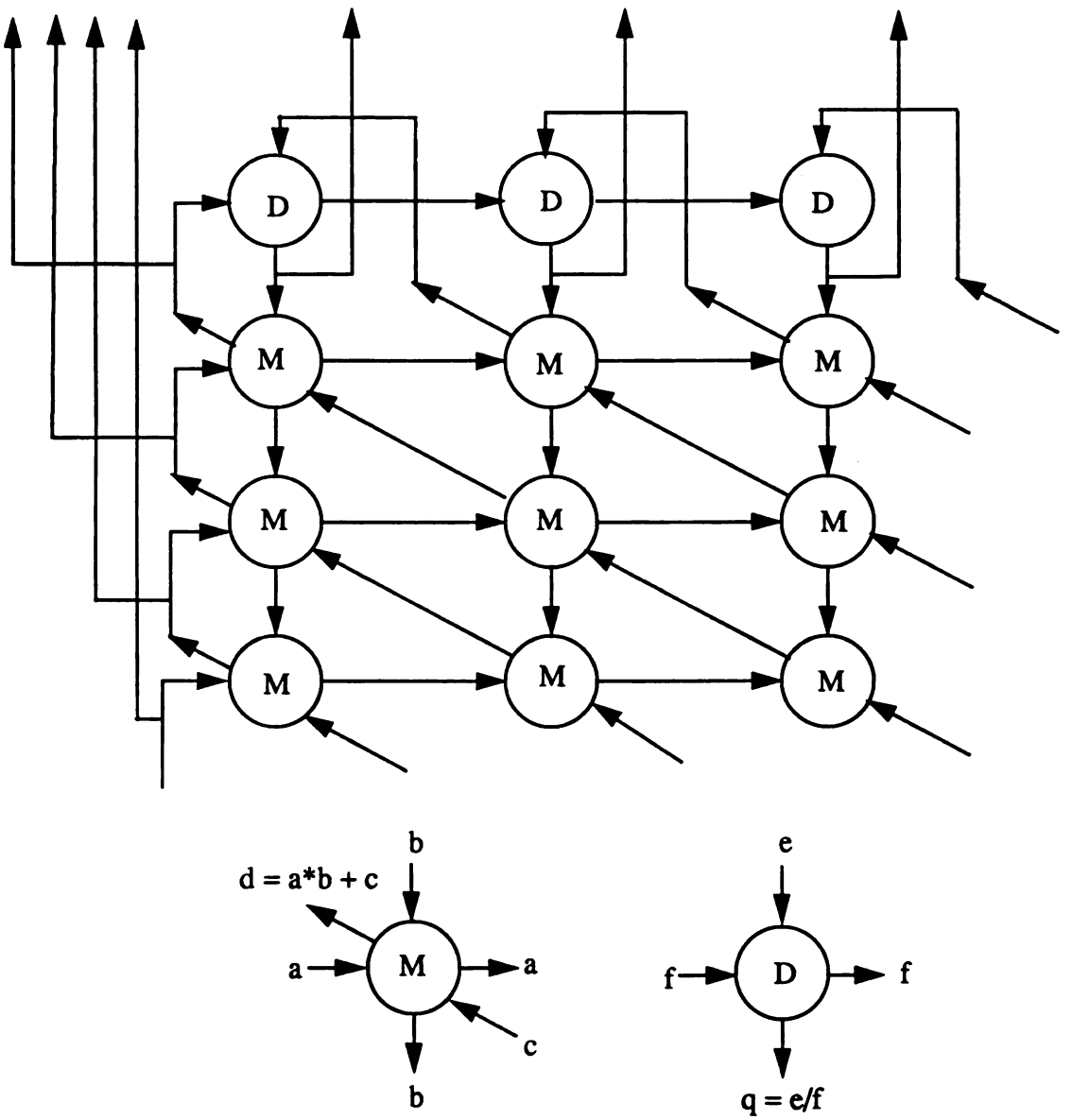


Figure 4.1. A systolic array for LU decomposition (4x4).

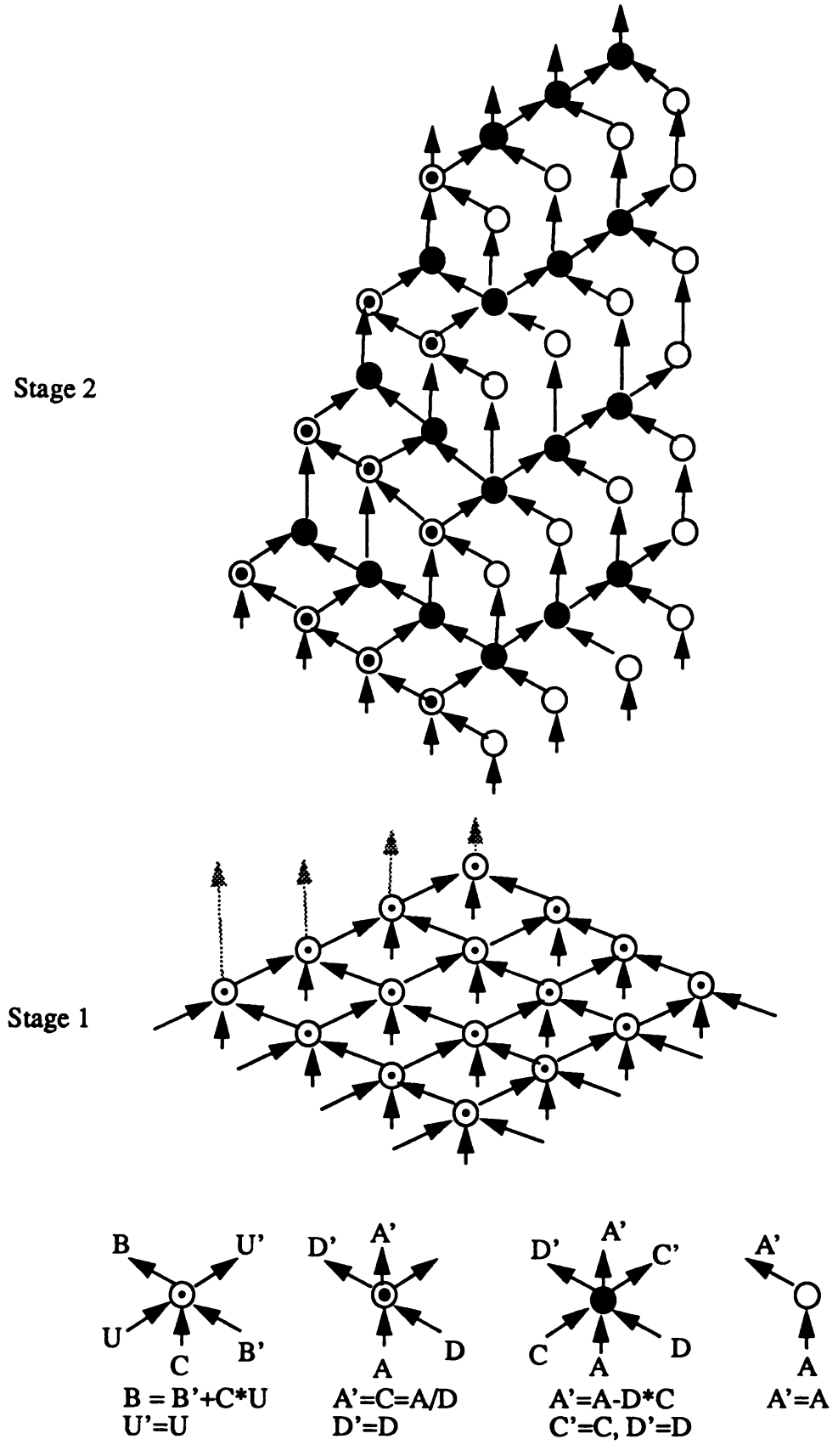


Figure 4.2. A systolic array for linear state equations .

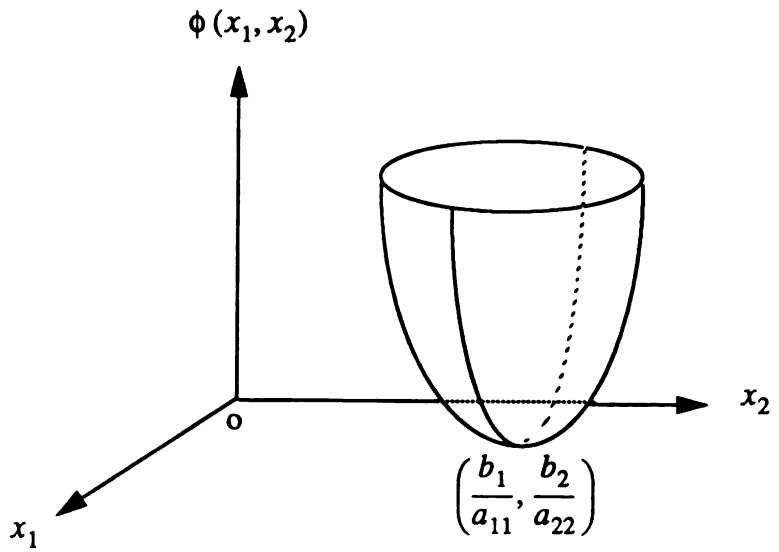


Figure 4.3. The image of $\phi(x_1, x_2)$ from eqaution (4.22), an ellipse-parabolic surface.

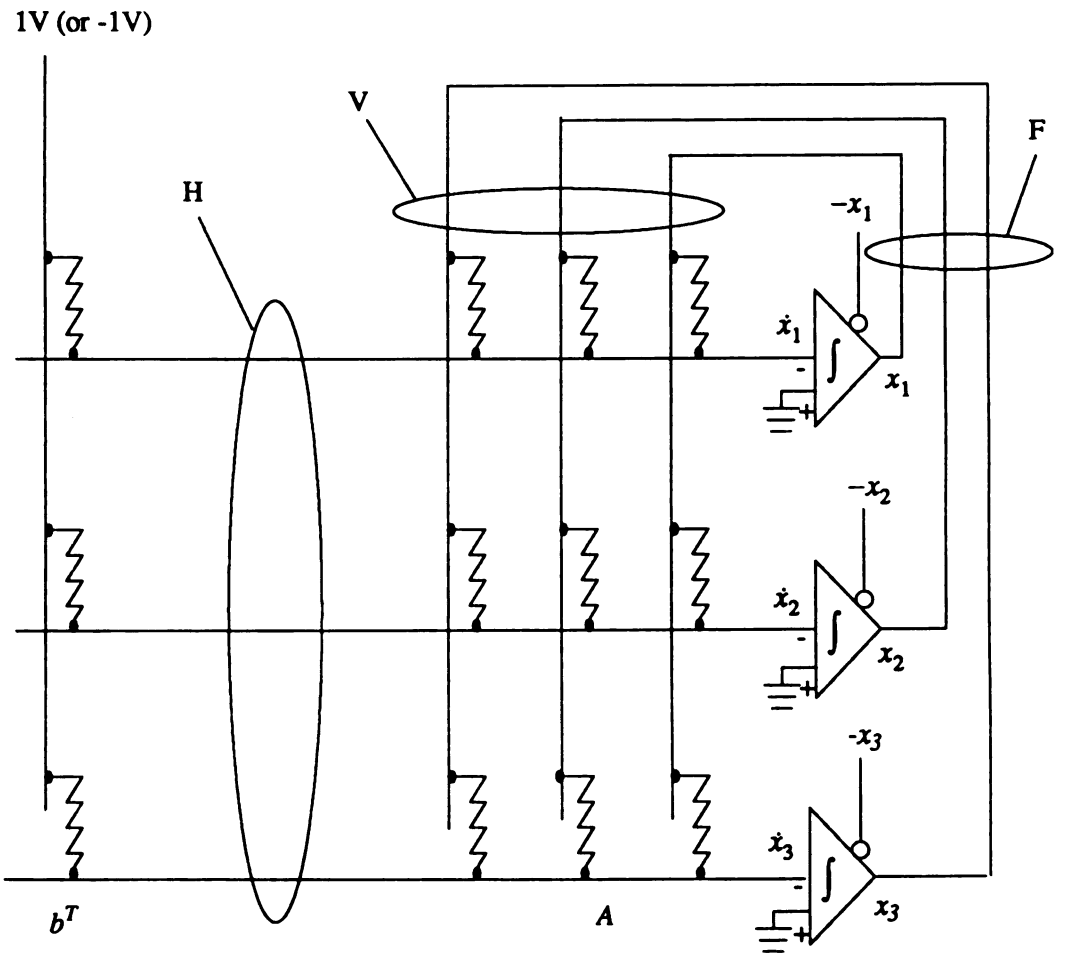


Figure 4.4. A 3x3 network architecture for the linear equation solver.

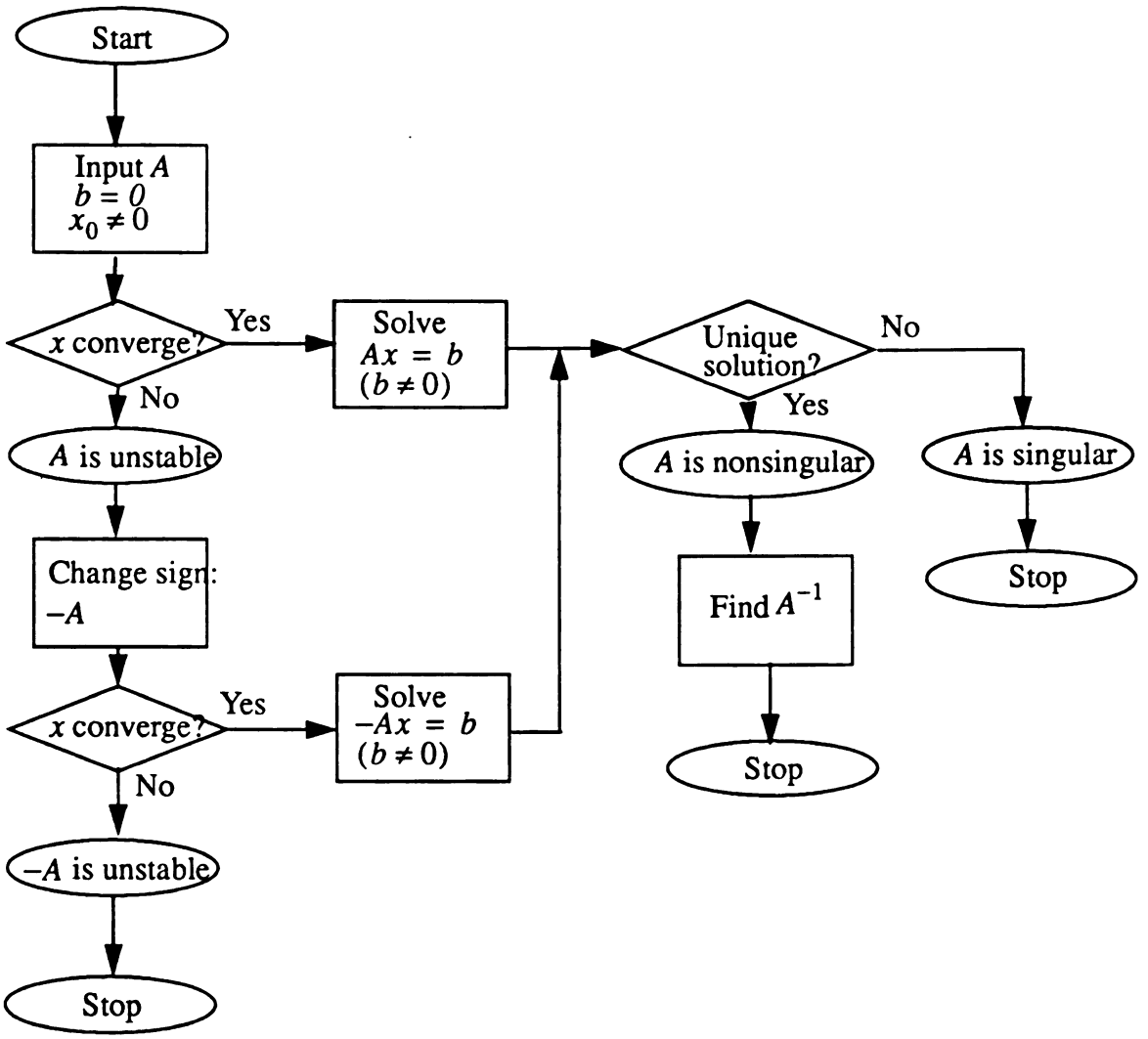


Figure 4.5. Applications of the ANN solver.

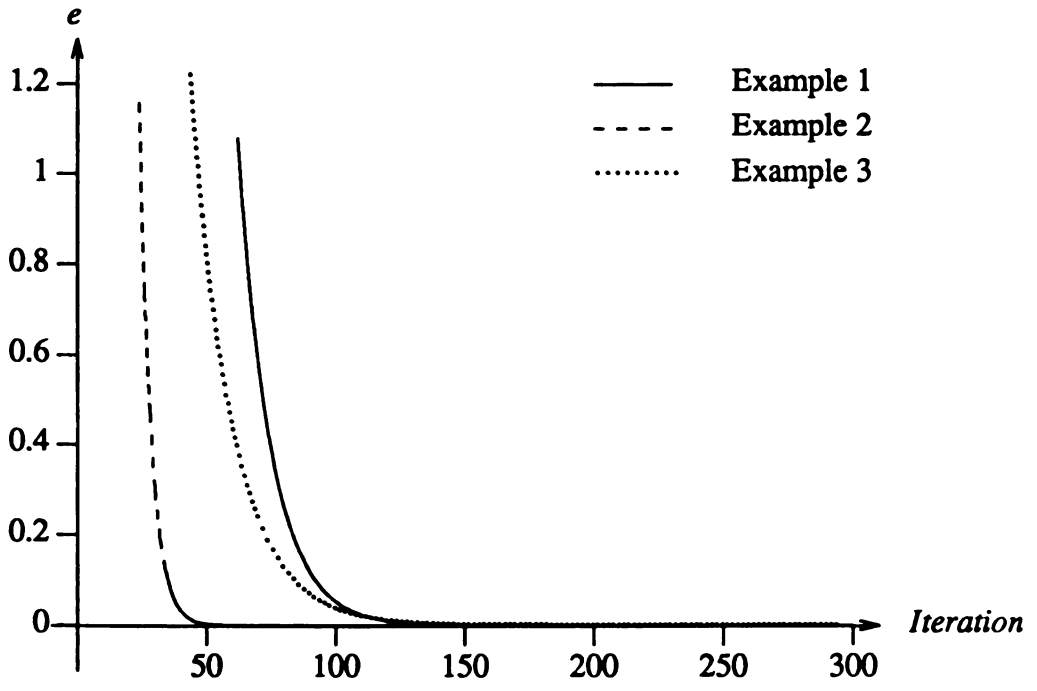


Figure 4.6. Error curves for examples.

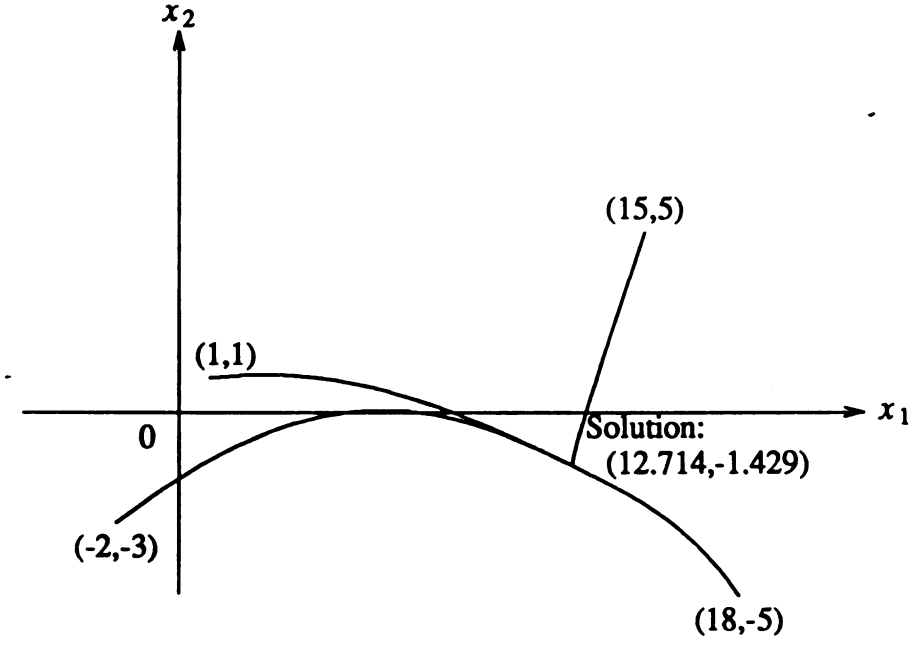


Figure 4.7. Trajectories for an example using different initial points.

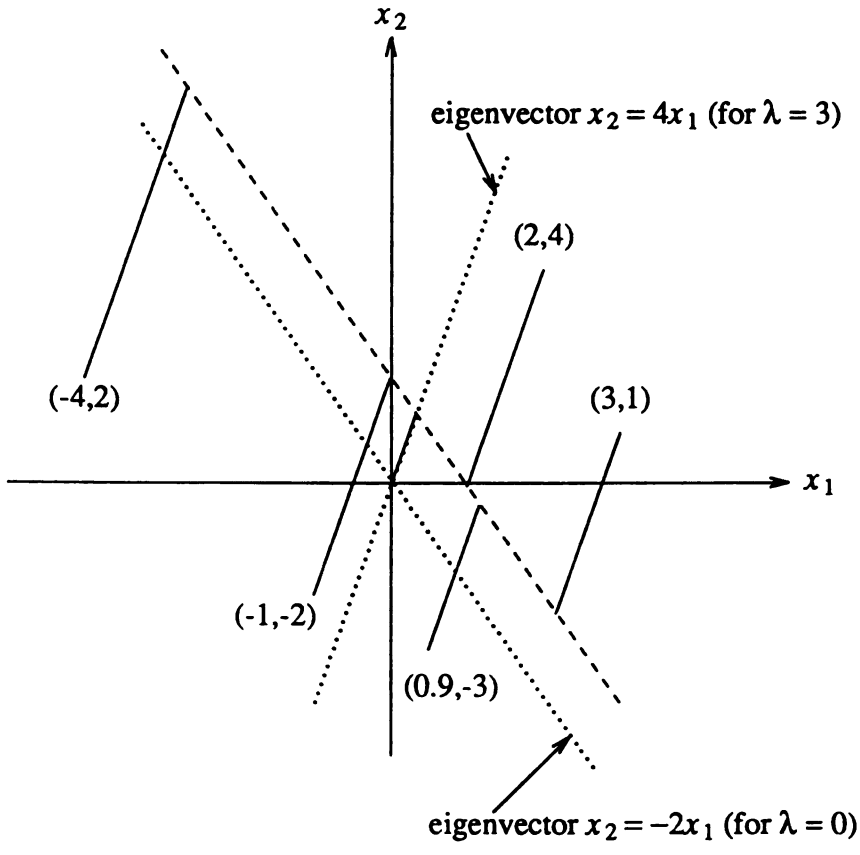


Figure 4.8. Trajectories of multi-solution (a singularity matrix).

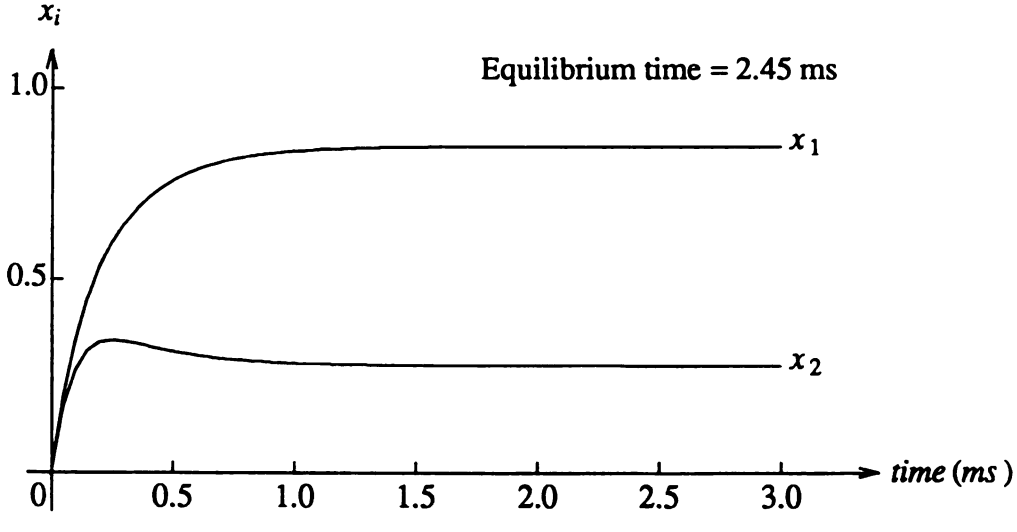


Figure 4.9. The result of SPICE simulation (two neurons).

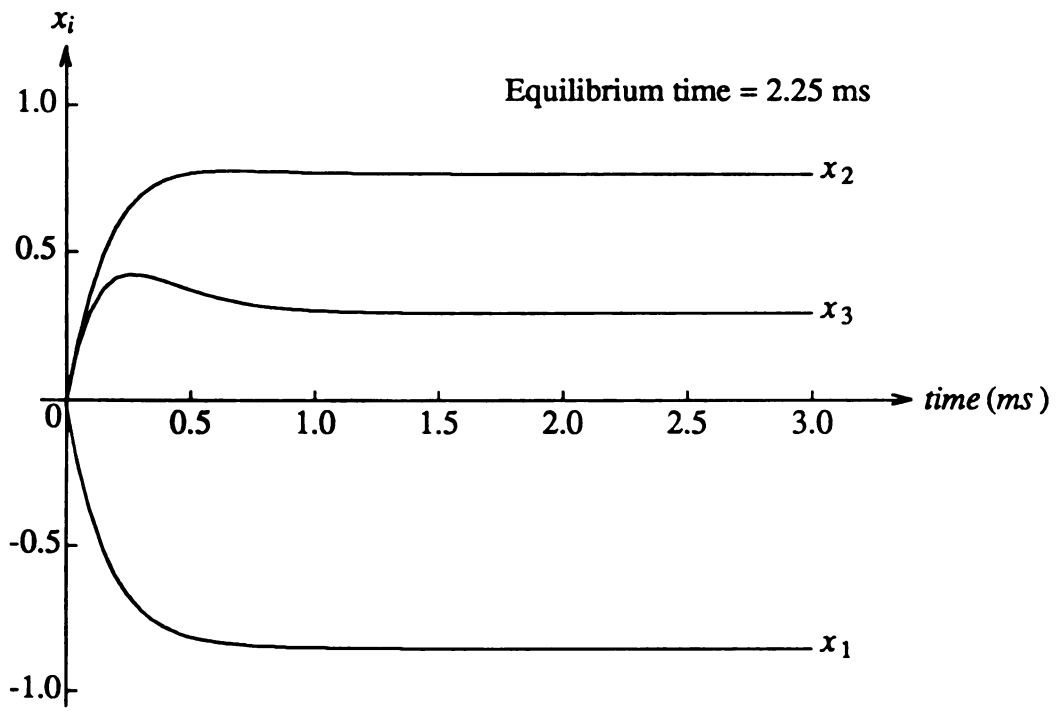


Figure 4.10. The result of SPICE simulation (three neurons).

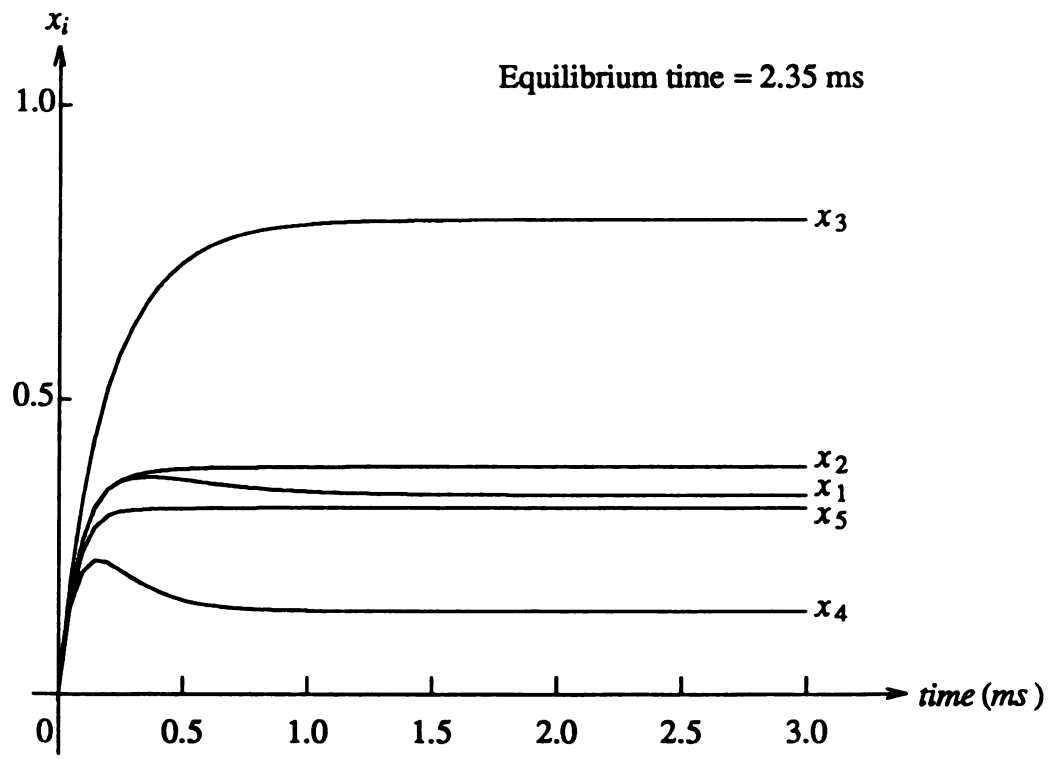


Figure 4.11. The result of SPICE simulation (five neurons).

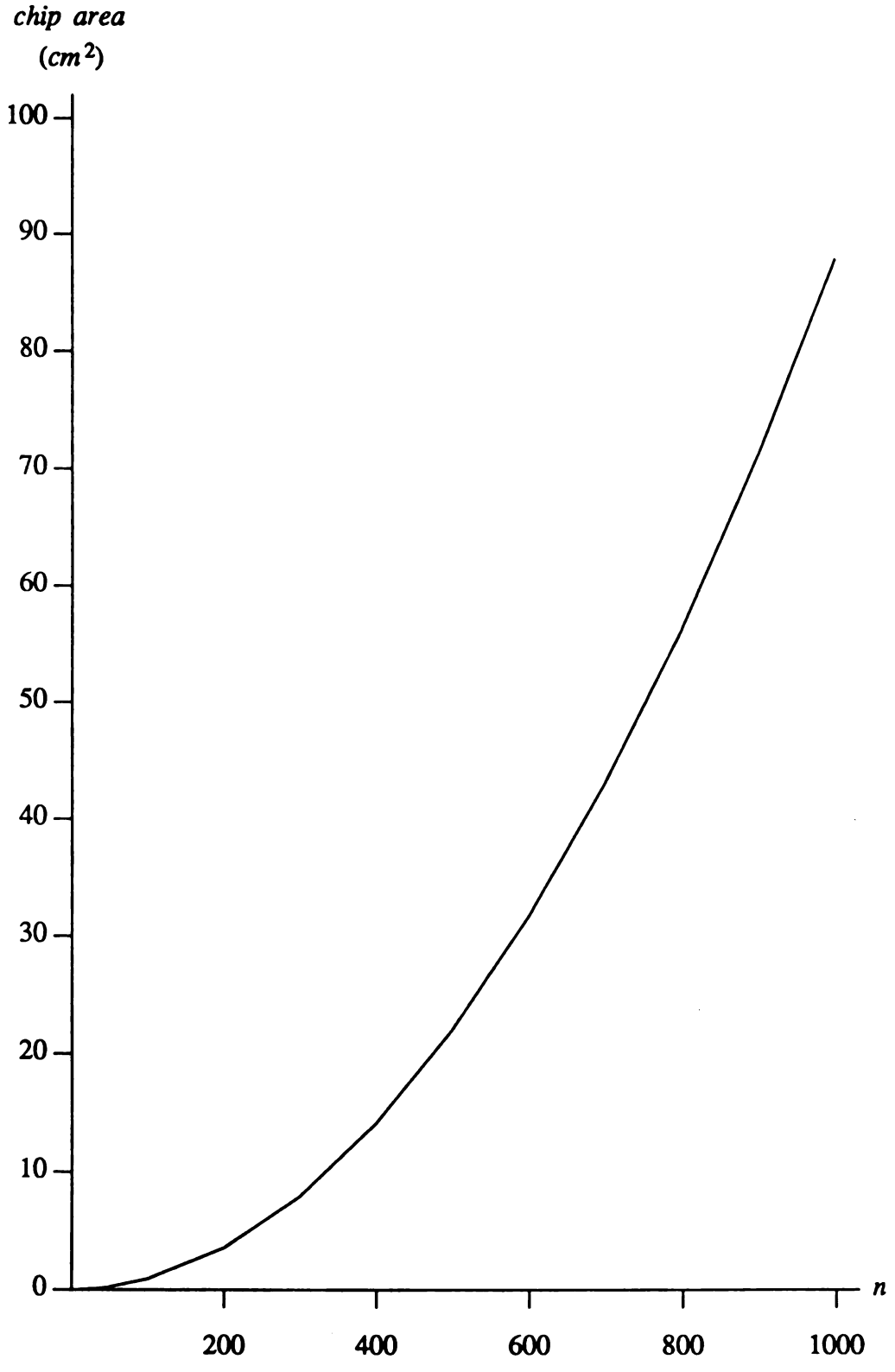


Figure 4.12. A relationship beteewn the chip area and the size.

Table 4.1. The time complexity for linear system computing.

Method	Time Steps
Gaussian Elimination	$\frac{1}{3}n^3+O(n^2)$
Gauss-Jordan Elimination	n^3
LU Decomposition by Software	$\frac{1}{3}n^3$
LU Decomposition by Systolic Array	$3n+1$
Jacobi Transformation	$O(n^3)\cdot O(n^2)$
Householder Reduction	$(n-2)\cdot O(n^3)$
QL Algorithm	$O(n^3)\cdot O(n)$

Table 4.2. The connection of resistors.

Value	Positive	Negative	Zero
a_{ij}	x_j	$-x_j$	unconnected
b_i	$-1\ V$	$+1\ V$	unconnected

Table 4.3. Linear equations solutions.

No.	Case	A	b	Solution	Error
1	Positive definite	A_1	12.0 3.5	12.7142744064 -1.4285681248	0.11E-9
2	Positive real part of eigenvalues	A_2	3.0 4.0 12.0 -6.0 8.0	0.6346439123 -0.3502970338 2.9066379070 -2.8384561539 1.4461678267	2.3E-10
3	Arbitatry	A_3	2.3 4.1 1.4 2.6 -4.2 -2.8 12.0 -21.0 10.0 7.7	0.3921020627 -0.6200531721 -5.0949549675 -3.2620067596 0.3290435970 1.7367919683 0.5648825169 -1.3450118303 -3.6507625580 -2.1458382607	4.7E-6

where

$$A_1 = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 2.0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 11.0 & 1.5 & -1.2 & 0.6 & 1.2 \\ 2.0 & 4.9 & 0.5 & -0.8 & 0.5 \\ 1.2 & 0.6 & 5.0 & 1.8 & 1.4 \\ 0.5 & -1.5 & 1.2 & 4.2 & 1.1 \\ 0.5 & -0.5 & 1.7 & 2.0 & 5.7 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 2.0 & -11.0 & 1.5 & -1.2 & 0.6 & 1.2 & -2.0 & -1.1 & 0.8 & 0.6 \\ 1.2 & -0.6 & -2.1 & 2.5 & -0.9 & 1.0 & 2.0 & 1.3 & 0.5 & -0.8 \\ 1.4 & 1.2 & -1.3 & -2.1 & -2.1 & 1.2 & 0.6 & 3.0 & 1.8 & 1.4 \\ 2.3 & 2.1 & -1.6 & -2.5 & 1.8 & 1.5 & -1.5 & 1.2 & 3.2 & 1.1 \\ 3.1 & 2.0 & -1.2 & -1.2 & 1.8 & 0.5 & -0.5 & 1.7 & 2.0 & 2.7 \\ 0.9 & 0.5 & -0.6 & 1.5 & 2.5 & 3.1 & -2.1 & -1.6 & 1.3 & 1.6 \\ -1.3 & -1.6 & -2.1 & 2.1 & 1.6 & 2.5 & 2.1 & 3.4 & -2.4 & 1.2 \\ 0.5 & 0.2 & 0.7 & 2.4 & 1.3 & -3.1 & -2.5 & 0.8 & 0.2 & 0.7 \\ 1.4 & 2.1 & -1.5 & -1.2 & 1.7 & 1.4 & -5.6 & -7.8 & 2.4 & 1.1 \\ 8.2 & 7.1 & -5.2 & 0.2 & -0.8 & 1.6 & 3.2 & 6.1 & 1.7 & 3.2 \end{bmatrix}$$

Table 4.4. Matrix inversion.

No.	A	Inversion	Error ($E=AA^{-1}-I$)
4	A_4	A_4^{-1}	E_4
5	A_2	A_2^{-1}	E_2

where

$$A_4 = \begin{bmatrix} 0.8 & 0.4 & 0.3 \\ 0.4 & 1.2 & 0.8 \\ 0.3 & 0.8 & 1.6 \end{bmatrix}$$

$$A_4^{-1} = \begin{bmatrix} 1.50234556 & -0.46948257 & -0.04694886 \\ -0.46948251 & 1.39671147 & -0.61032742 \\ -0.04694865 & -0.61032748 & 0.93896627 \end{bmatrix}$$

$$E_4 = \begin{bmatrix} 0.00000113 & 0.00000006 & -0.00000056 \\ 0.00000037 & 0.00000107 & 0.00000052 \\ -0.00000002 & 0.00000055 & 0.00000054 \end{bmatrix}$$

$$A_2^{-1} = \begin{bmatrix} 0.09709457 & -0.4654792 & 0.04247883 & -0.03103656 & -0.02080171 \\ -0.04244077 & 0.24640946 & -0.04885584 & 0.08177510 & -0.01646140 \\ -0.00934555 & -0.05908667 & 0.23789524 & -0.09630578 & -0.03269460 \\ -0.02375325 & 0.10910744 & -0.07685770 & 0.31758156 & -0.04698043 \\ -0.00111820 & 0.00503705 & -0.05199542 & -0.07281359 & 0.20205469 \end{bmatrix}$$

$$E_2 = \begin{bmatrix} 0.00000018 & 0.00000004 & -0.00000006 & -0.00000001 & -0.00000009 \\ 0.00000029 & 0.00000036 & 0.00000019 & 0.00000005 & 0.00000024 \\ -0.00000033 & 0.00000012 & 0.000000030 & -0.00000003 & -0.00000024 \\ 0.00000048 & -0.00000014 & 0.00000029 & 0.00000012 & 0.00000035 \\ -0.00000008 & 0.00000002 & -0.00000005 & 0.00000000 & 0.00000012 \end{bmatrix}$$

Table 4.5. Stability.

No.	-A	Solution	Stable	Eigenvalues
6	A_5	Exist	Yes	λ_1=6.710624 λ_2=2.144688+i 0.784808 λ_3=2.144688–i 0.784808
7	A_6	Infinite	No	λ_1=5.970251 λ_2=2.420508 λ_3=–3.390759

where

$$A_5 = \begin{bmatrix} 3.0 & 2.0 & 1.0 \\ 1.0 & 4.0 & 2.0 \\ 2.0 & 1.0 & 4.0 \end{bmatrix}$$

$$A_6 = \begin{bmatrix} -3.0 & 2.0 & 1.0 \\ 1.0 & 4.0 & 2.0 \\ 2.0 & 1.0 & 4.0 \end{bmatrix}$$

Table 4.6. Singularity.

No.	A	$\mathbf{b}^T$	Solution	Singularity
8	A_7	(1.0 4.0) (2.0 4.0)	Multiple No	Yes
9	A_8	(1.0 4.0)	Unique	No

where

$$A_7 = \begin{bmatrix} 1.0 & 0.5 \\ 4.0 & 2.0 \end{bmatrix}$$

$$A_8 = \begin{bmatrix} 1.0 & 0.5 \\ 1.0 & 2.0 \end{bmatrix}$$

Chapter 5

ANN Techniques for Power System Control Analysis

This Chapter presents a new approach for solving nonlinear equations using an artificial neural network technique which shows great potential for more efficiently solving a class of power system problems. Foremost, methodologies for solving nonlinear equations addressing power load flow and contingency analysis problems are reviewed. Then, a mapping between the solution of nonlinear equations and quadratic-nonlinear minimization is established so that solving nonlinear equations can be viewed as finding the minimum of a quadratic-nonlinear function. From an engineering point of view, a new formulation for an ANN nonlinear equation solver is proposed which can significantly simplify hardware architecture. The conditions for stability and convergence of neural network based on the new formulation are proven, providing a mathematical basis for the approach. Furthermore, ANN formulations for the full power load flow, the decoupled load flow, and the DC load flow are given, and corresponding architectures for them are proposed, which can be implemented with VLSI technology.

5.1 Introduction

On-line control of large-scale power systems is a difficult problem exacerbated by the computational complexity of solving large-scale load flow and contingency analysis formulations in time frames required for real-time response to rapidly changing operating conditions. The problem has been aggravated by the lack of robust algorithms and implementation technology which would allow for the design of economically feasible dedicated high-speed computational hardware.

A large class of problems in science and engineering, including a group of these power system analysis, formulations can be formulated as optimization problems. Conventional computers are sometimes adequate for solving this kind of problem providing that it is within reasonable bounds. Most real world problems, however, are too large to be solved in this manner. As described in the previous chapters, a new computational trend, artificial neural network computation, has recently emerged for solving these difficult problems. Hopfield and Tank proposed a neural network for solving linear programming [TaHo86]. Kennedy and Chua developed a canonical form of neural networks for finding the minimum for nonlinear programming [KeCh88]. For combinatorial optimization problems, such as the Traveling Salesman Problem (TSP), several algorithms and architectures have been reported [HoTa85, AiFS90, WiPa88]. One of the significant features of these approaches is that the time complexity is size insensitive so that solving large-scale systems in real-time is possible if these approaches are implemented in hardware. Linking ANN optimization networks with techniques for solving nonlinear equations, this Chapter proposes an artificial neural network approach and architecture for power load flow and contingency analysis with the following characteristics: A mapping is established between an optimization problem and nonlinear equations solution. A simpler formulation is proposed for an ANN nonlinear equation solver so that the method is more suitable for VLSI

implementation. Advantage of automatical convergence is taken so that the time complexity of the solver is size-independent. The formulation uniform is made for solving both power load flow and contingency analysis, and resulting in the identical procedure for single and multiple outages in contingency analysis. A straightforward process is provided for solving the problem without training and testing. An accurate result is guaranteed for power load flow and contingency analysis.

This Chapter is organized as follows. First, the problem descriptions for power load flow and contingency analysis are given, and a variety of approaches to these problems are reviewed. Then, an ANN approach for solving nonlinear equations is proposed in which a mapping between an optimization problem and nonlinear equation solution is established. A formulation of an ANN solver for the general nonlinear equations is defined, and the conditions for the stability and convergence of networks are proven. Based on the general nonlinear equation solver, the ANN formulations for the full power load flow, the decoupled load flow and the DC load flow are represented. Furthermore, ANN architectures for three cases of power load flow, and the simulation results are presented.

5.2 Methodology Review

A description for power load flow and contingency problems is given first, and then typical computational techniques for solving the problems are reviewed.

5.2.1 Problem Statement

5.2.1.1 Power Load Flow

The power load flow is an analysis problem which deals with finding the steady-state solution of bus voltage magnitudes and phase angles for given load demands at the various load busses and for assumed generation levels. The load flow

is of fundamental importance to power systems control because many practical applications, such as transmission planning, contingency analysis, VAR/voltage analysis, on-line control, and security enhancement, require a load flow solution [Deb88, Co*et*86, Mo*et*87].

Let $k(i)$ denote the set of buses connected to bus i. At bus i, P_i, Q_i, P_{Di}, Q_{Di} represent generated real, and reactive power, and load real and reactive power, respectively. Let V_i and δ_i be the magnitude and the phase angle of the complex voltage at bus i. The load flow problem can be written using these terms as

$$P_i - P_{Di} = V_i{}^2 G_{ii} - V_i \sum_{j \in k(i)} V_j [G_{ij} \cos\delta_{ij} + B_{ij} \sin\delta_{ij}], \tag{5.1a}$$

$$Q_i - Q_{Di} = -V_i{}^2 B_{ii} - V_i \sum_{j \in k(i)} V_j [G_{ij} \sin\delta_{ij} - B_{ij} \cos\delta_{ij}], \tag{5.1b}$$

where

$$G_{ii} \equiv \sum_{j \in k(i)} (G_{Sij} + G_{ij}), \tag{5.1c}$$

$$B_{ii} \equiv B_{Si} + \sum_{j \in k(i)} (B_{Sij} + B_{ij}), \tag{5.1d}$$

$$\delta_{ij} \equiv \delta_i - \delta_j. \tag{5.1e}$$

G_{ij} and B_{ij} are the conductance and admittance of the transmission line between bus i and bus j. For simplicity, these equations can be written vector form as

$$\begin{cases} \mathbf{P}(\delta, \mathbf{V}) = 0 \\ \mathbf{Q}(\delta, \mathbf{V}) = 0. \end{cases} \tag{5.2}$$

Because all parameters in a power system are under consideration, this formulation is called the full power load flow (FLF).

5.2.1.2 Steady-State Contingency Analysis

The steady-state contingency analysis focuses on predicting the power load flow following some event such as a transmission line outage or a transformer outage. For contingency evaluation, it is assumed that real and reactive loads, real power generation, and generator bus voltage magnitudes are unchanged before and after the outage. The objective is to judge the security of the power system against the outage by solving the load flow problem with changed parameters of the system topology. By comparing the solution with the constraint, a conclusion of security can be easily reached.

5.2.2 Methodology Review

5.2.2.1 Pattern Classification Method

If the operational conditions of a power system are regarded as the input variables, and the judgement of static security for the system is regarded as the output variable, then the contingency analysis can be treated as a typical pattern classification problem. Artificial neural networks have been applied to problems of this type [Ag*et*89, Fi*et*89, SoPa89, HuSh91]. The approach involves two phases: off-line training for determining parameters of the neural network, and on-line testing for contingency analysis. A feedforward neural network model is often used for building the classifier [[Ag*et*89]. Figure 5.1 shows a neural network architecture, where S, P, and Q, are apparent power, real power and reactive power, respectively, 1.0 is a constant bias for all thresholds. Each circle represens a processing element (neuron). There are three layers (input, hidden and output) and two connection matrices (W_{mn} and W_{no}). The back-propagation learning rule is used for training the network so that the connection weights are adaptively updated. When the values of the connection weights converge, the learning process is finished and the neural network is used to test different contingency situations. The attractive feature of this method is that the testing is a straight forward process which can be performed in real-time. However, the

convergence of the learning process is case dependent so that the general conditions for convergence are still subject to investigation [Deb88].

5.2.2.2 Newton-Raphson Method

One common way to solve the FLF problem is using the Newton-Raphson method [Ort87a]. By defining the Jacobian matrix of (5.2) as

$$J = \begin{bmatrix} \frac{\partial \mathbf{P}}{\partial \delta} & \frac{\partial \mathbf{P}}{\partial \mathbf{V}} \\ \frac{\partial \mathbf{Q}}{\partial \delta} & \frac{\partial \mathbf{Q}}{\partial \mathbf{V}} \end{bmatrix}, \tag{5.3}$$

an iteration solution can be found by

$$\begin{bmatrix} \delta^{(k+1)} \\ \mathbf{V}^{(k+1)} \end{bmatrix} = \begin{bmatrix} \delta^{(k)} \\ \mathbf{V}^{(k)} \end{bmatrix} + J^{-1}(\delta^{(k)}, \mathbf{V}^{(k)}) \begin{bmatrix} \Delta \mathbf{P} \\ \Delta \mathbf{Q} \end{bmatrix} \tag{5.4}$$

where k is the iteration index.

5.2.2.3 Estimation Method

Since transmission line resistances are much smaller than the corresponding reactances, angular differences across a transmission line are small, and the voltage magnitudes are close to the normal values ($V_i = 1.0\ p.u.$), some approximations can be used to simplifying the problem. These include the decoupled load flow (DLF) and the DC load flow (DCLF). The solution to DLF takes the form

$$\begin{bmatrix} \delta^{(k+1)} \\ \mathbf{V}^{(k+1)} \end{bmatrix} = \begin{bmatrix} \delta^{(k)} \\ \mathbf{V}^{(k)} \end{bmatrix} + \begin{bmatrix} B_{node}{}^{-1} & 0 \\ 0 & B_{load}{}^{-1} \end{bmatrix}^T \begin{bmatrix} \Delta \mathbf{P}^{(k)} / \mathbf{V}_{node}{}^{(k)} & 0 \\ 0 & \Delta \mathbf{Q}^{(k)} / \mathbf{V}_{load}{}^{(k)} \end{bmatrix} \tag{5.5}$$

where B_{node}, B_{load}, $\mathbf{V}_{node}$, and $\mathbf{V}_{load}$ are node-connection matrix, load-connection matrix, node-voltage matrix, and load-voltage matrix, respectively.

In DCLF, the variation of the voltage magnitude is ignored so that power load flow degenerates to a linear equation of the form

$$A\,\delta = \mathbf{P} \tag{5.6}$$

where A is a connection matrix, δ and $\mathbf{P}$ are phase angle vector and active power vector, respectively. Therefore, the technique for solving linear equations can be used.

5.2.2.4 Multi-level Screening Method

The objective of contingency analysis is to identify the critical branches which may suffer from violations. In general, the number of critical branches is very small so that approximate techniques can be used for filtering noncritical branches at the beginning, and more accurate analysis can be used at each higher screening level with a reduced set of candidates. In this way, a significant amount of computing time can be saved. Furthermore, a pre-screening algorithm has been developed for reducing the number of branches for the first process [Bret91]. An important assumption of this algorithm is the contingency localization since the effects of power system contingency are highly localized near the outaged branch. Consequently, the power system network can be divided into three parts: the outaged inside network, the stiff boundary and the rest of the network, where the first and the second parts constitute the local cut-off network. The third part can be ignored during the contingency analysis as it is considered far away from the outage. This approach can handle a single contingency situation. Tests of a 600-bus system have showed that the CPU time can be reduced by about 50% compared with the complete bounding method. The relative error was 1-2% compared with the full power load flow solutions [Bret91].

5.2.2.5 Homotopy Method

Homotopy is a global convergent method for solving nonlinear systems of polynomial equations [ChMY78]. A homotopy function is defined as

$$H(\mathbf{x}, t) = (1 - t)S(\mathbf{x}) + t\,T(\mathbf{x}) \tag{5.7}$$

where t is a real parameter varying from 0 to 1, $T(\mathbf{x})$ is the target system to be solved, and $S(\mathbf{x})$ is a simplier system for initial situations, called the starting system. For power load flow problem, $S(\mathbf{x})$ is often chosen as

$$S(\mathbf{x}) = \begin{bmatrix} a_1 x_1^2 - b_1 \\ a_2 x_2^2 - b_2 \\ \cdots\cdots \\ a_n x_n^2 - b_n \end{bmatrix}. \tag{5.8}$$

The procedure for homotopy method is defined as:

(i) Let $t_i = 0$. Randomly choose complex vectors $\mathbf{a} = (a_1, a_2, \cdots, a_n)$, $\mathbf{b} = (b_1, b_2, \cdots, b_n)$, and solve $S(\mathbf{x}) = 0$ to get a set of roots which are the starting points of homotopy curvers.

(ii) Update t_i by $t_i^{new} = t_i^{old} + \Delta t$, where Δt is stepping distance. If $t_i < 1.0$, go to (iii); otherwise, stop.

(iii) Use an iterative method, such as Newton Raphson method, to solve homotopy equation for each t_i

$$H(\mathbf{x}, t_i) = (1 - t_i)S(\mathbf{x}) + t_i T(\mathbf{x}) = 0. \tag{5.9}$$

(iv) go to step (ii).

This method can systematically find all possible solutions at the same time. Moreover, the speed of this process can be improved by means of dynamically adjusting Δt and using parallel computation techniques [Sa*et*89].

5.3 ANN Formulation for Power Load Flow

5.3.1 ANN Solution to Nonlinear Equations

Consider the set of nonlinear equations

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \cdots\cdots \\ f_n(x_1, x_2, \cdots, x_n) = 0 \end{cases} \tag{5.10a}$$

or

$$\mathbf{f}(\mathbf{x}) = 0 \tag{5.10b}$$

where $\mathbf{x}$ is a variable vector and $\mathbf{f}$ is a function vector. The Jacobian matrix of $\mathbf{f}(\mathbf{x})$ is defined as

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdot & \cdot & \cdots & \cdot \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}. \tag{5.11}$$

Note that J is a function matrix of $\mathbf{x}$. In most cases, it is desirable to evaluate J at a given point, say $\mathbf{x}_0$. As a consequence, J becomes a constant matrix, represented by $J(\mathbf{x}_0)$. Before deriving the ANN formulation, some definitions are given.

Definition 5.1: If the Jacobian matrix at $\mathbf{x}$, $J(\mathbf{x})$, is not a zero matrix for $\mathbf{x} \in D \subset R^n$, then it is called an unvanished Jacobian matrix on D.

Definition 5.2: If all eigenvalues of the Jacobian matrix at $\mathbf{x}$ satisfy $Re(\lambda_i) > 0$ for $\mathbf{x} \in D$, then it is called an eigen-positive Jacobian matrix.

Next, a mapping between an optimization problem without constraints and non-linear equation solution can be established.

Proposition 5.1: If the Jacobian matrix of nonlinear equations (5.10) is unvanished for any $\mathbf{x}$, then an optimum point is a solution to the nonlinear equations.

Proof: Define an objective function

$$\phi(\mathbf{x}) = \sum_{i=1}^{n} f_i^2(\mathbf{x}), \tag{5.12}$$

then an optimum point of $\phi(\mathbf{x})$ must satisfy the necessary condition

$$\nabla\phi = \begin{bmatrix} \frac{\partial\phi}{\partial x_1} \\ \frac{\partial\phi}{\partial x_2} \\ \vdots \\ \frac{\partial\phi}{\partial x_n} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{5.13a}$$

that is,

$$\begin{bmatrix} 2f_1\frac{\partial f_1}{\partial x_1} + 2f_2\frac{\partial f_2}{\partial x_1} + \cdots + 2f_n\frac{\partial f_n}{\partial x_1} \\ 2f_1\frac{\partial f_1}{\partial x_2} + 2f_2\frac{\partial f_2}{\partial x_2} + \cdots + 2f_n\frac{\partial f_n}{\partial x_2} \\ \cdots\cdots \\ 2f_1\frac{\partial f_1}{\partial x_n} + 2f_2\frac{\partial f_2}{\partial x_n} + \cdots + 2f_n\frac{\partial f_n}{\partial x_n} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.13b}$$

$$2\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}^T \begin{bmatrix} f_1 \\ f_2 \\ \cdots \\ f_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}. \tag{5.13c}$$

$$2J^T\begin{bmatrix} f_1 \\ f_2 \\ \cdots \\ f_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}. \tag{5.13d}$$

Since the Jacobian matrix is unvanished, the above equation implies that

$$\begin{bmatrix} f_1 \\ f_2 \\ \cdots \\ f_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}. \tag{5.13e}$$

In other words, the minimum point satisfies the corresponding nonlinear equations. □

Proposition 5.1 indicates that a minimum point of $\sum_{i=0}^{n} f_i^2(\mathbf{x})$ is a solution to the nonlinear equation. However, if a gradient system is directly derived from this objective function, then the corresponding artificial neural network for the problem becomes complex since it encounters both $\mathbf{f}(\mathbf{x})$ and $J(\mathbf{x})$. Alternatively, a simplier formulation is proposed below.

For the set of nonlinear equations described by (5.10), define a dynamic system as

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \cdots \\ \dot{x}_n \end{bmatrix} = - \begin{bmatrix} f_1(x_1, x_2, \cdots, x_n) \\ f_2(x_1, x_2, \cdots, x_n) \\ \cdots\cdots \\ f_n(x_1, x_2, \cdots, x_n) \end{bmatrix}. \tag{5.14}$$

When the system approaches an equilibrium, then

$$\dot{\mathbf{x}} = 0.$$

At that time, there must exist the equality

$$\mathbf{f}(\mathbf{x}) = 0.$$

We call this network, described in (5.14), an ANN nonlinear equation solver, because equation (5.14) can be regarded as a simplified formulation of an artificial neural network for optimization [KeCh88]. An important issue for the ANN nonlinear equation solver is ensuring its stability as will be discussed the following Theorem.

Theorem 5.1: If $\mathbf{f}(\mathbf{x})$ is continuously differentiable and its Jacobian matrix is eigen-positive, then the ANN nonlinear equation solver is stable in the neighborhood of a solution point and its equilibrium point must be satisfied with the nonlinear equations.

Proof is given in the Appendix A.

Theorem 5.1 establishes a formulation for solving general nonlinear equations using a newly developed artificial neural network. In what follows, this formulation

will be applied to power system control problems [ChSh92b].

5.3.2 The Full Power Load Flow

As prescribed by equations (5.2) and (5.14), the neural network for the full power load flow is formed as

$$\dot{\delta} = - \mathbf{P}(\delta, \mathbf{V}) \tag{5.15a}$$

$$\dot{\mathbf{V}} = - \mathbf{Q}(\delta, \mathbf{V}) \tag{5.15b}$$

where δ and $\mathbf{V}$ are the phase angle vector and the voltage vector, respectively. It is apparent that the Jacobian matrix described in (5.3) is continuously differential in the domain $\delta \in R^n$ and $\mathbf{V} \in R^n$. In order to investigate the eigen-positive property of the Jacobian matrix at the neighborhood of an equilibrium point, the following definition and theorem must be introduced.

Definition 5.3: If matrix A satisfies

$$a_{ii} > 0, \tag{5.16a}$$

$$a_{ii} \geq \sum_{j \neq i} | a_{ij} |, \tag{5.16b}$$

and

$$a_{ii} \geq \sum_{j \neq i} | a_{ji} |, \tag{5.16c}$$

then A is said to be diagonally dominant. Furthermore, if only inequality is held in the above, A is said to be strictly diagonal-dominant. For example, matrices A_1 and A_2 below

$$A_1 = \begin{bmatrix} 4.0 & 0.5 & -0.5 & -1.0 \\ 1.0 & 3.0 & 1.0 & 0.3 \\ 1.0 & -0.5 & 2.8 & 1.0 \\ 1.0 & -1.0 & 0.3 & 3.5 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 4.0 & 1.5 & -0.5 & -1.0 \\ 1.0 & 3.0 & 1.0 & 0.3 \\ 1.0 & -0.5 & 2.5 & 1.0 \\ 1.0 & -1.0 & 0.3 & 3.5 \end{bmatrix}$$

are strictly diagonal-dominant, and diagonal-dominant, respectively.

Theorem 5.2: A strictly diagonally dominant matrix is eigen-positive.

Proof can be found in [Jen77].

In general, there exist multiple solutions to power load flow due to its nonlinear behavior. From a practical point of view, however, the feasible solution is the most important because the power system operates under this situation. As derived in Theorem 5.1, a nonlinear system usually features a local stability. Thus, a problem arises in the choice of an appropriate initial point for an ANN solver in order to make sure that the equilibrium point is the desired feasible solution. Fortunately, this problem can be solved by setting the initial point at the ideal operation point for the power system. That is, all phase angles are 0, and the magnitude of all bus voltages is 1. It is certain that the objective of power system control is to make the difference between the system state $(\delta, \mathbf{V})$ and the ideal state $(\delta^0 = 0, \mathbf{V}^0 = 1)$ as small as possible by means of shunting capacitor banks, and adjusting transformer taps. Usually, an allowable range for the difference is defined as $|V - V^0| < 0.05$ and $|\delta - \delta^0| < 0.15$ (δ in radian). Next, the Jacobian matrix at the ideal point is discussed. Of course, in this case the Jacobian matrix only depends on topology parameters: conductance and susceptance of transmission lines, B, G. By plugging the values of δ_i and V_i (0,1) for all i into equation (5.3), the Jacobian matrix becomes

$$J(0,1) = \begin{bmatrix} a_{11} & \cdots & a_{1n} & b_{11} & \cdots & b_{1n} \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ a_{n1} & \cdots & a_{nn} & b_{n1} & \cdots & b_{nn} \\ c_{11} & \cdots & c_{1n} & d_{11} & \cdots & d_{1n} \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ c_{n1} & \cdots & c_{nn} & d_{n1} & \cdots & d_{nn} \end{bmatrix} \tag{5.17}$$

where

$$a_{ii} = -\sum_{j \in k(i)} B_{ij}; \tag{5.17a}$$

$$a_{ij} = B_{ij} \qquad (i \neq j); \tag{5.17b}$$

$$b_{ii} = 2G_{ii} - \sum_{j \in k(i)} G_{ij}; \tag{5.17c}$$

$$b_{ij} = -G_{ij} \qquad (i \neq j); \tag{5.17d}$$

$$c_{ii} = - \sum_{j \in k(i)} G_{ij}; \tag{5.17e}$$

$$c_{ij} = G_{ij} \qquad (i \neq j); \tag{5.17f}$$

$$d_{ii} = -2B_{ii} + \sum_{j \in k(i)} B_{ij}; \tag{5.17g}$$

$$d_{ij} = B_{ij} \qquad (i \neq j). \tag{5.17h}$$

It is apparent that every diagonal entry (a_{ii}, d_{ii}) of the Jacobian matrix is positive because

$$B_{ij} < 0 \Rightarrow a_{ii} = - \sum_{j \in k(i)} B_{ij} > 0 \tag{5.18a}$$

and

$$\begin{aligned} d_{ii} &= -2B_{ii} + \sum_{j \in k(i)} B_{ij} \qquad (5.18b) \\ &= -2(B_{si} + \sum_{j \in k(i)} B_{sij} + \sum_{j \in k(i)} B_{ij}) + \sum_{j \in k(i)} B_{ij} \\ &= -2B_{si} - 2 \sum_{j \in k(i)} B_{sij} - \sum_{j \in k(i)} B_{ij}. \end{aligned}$$

In power systems, $\sum_{j \in k(i)} |B_{ij}|$ represent the sum of susceptance of the all transmission lines to bus i, and $|B_{si}| + \sum_{j \in k(i)} |B_{sij}|$ represent the sum of susceptance of the capacitor bank at bus i, and susceptance of the all Π-equivalent sections at the side of bus i. In general, the former is much larger than the latter, that is,

$$\sum_{j \in k(i)} |B_{ij}| > 2(|B_{si}| + \sum_{j \in k(i)} |B_{sij}|) \Rightarrow d_{ii} > 0. \tag{5.18c}$$

Using the notation

$$\sum_{j \in k(i)} B_{ij} = B_{i0} + \sum_{(j \neq i)} B_{ij} \tag{5.19}$$

where B_{i0} represents admittance of the transmission line between bus i and the slack bus (bus 0), and

$$B_{ij} = \begin{cases} B_{ij} & \text{if there is a connection between bus } i \text{ and } j \\ 0 & \text{otherwise,} \end{cases} \tag{5.20}$$

then the conditions for making Jacobian matrix strictly dominant are found as

$$|B_{io}| > 2G_{ii} \tag{5.21a}$$

$$|b_{i0} + 2B_{si} + 2\sum_{j \in k(i)} B_{sij}| > 2\sum_{j!=i} |G_{ij}| + |G_{i0}|. \tag{5.21b}$$

In other words, when the parameters of a power system are satisfied with the conditions in (5.21), the neural network formulated in (5.15) is stable in the neighborhood of (0, 1) so that a feasible solution can be found by the network.

It is worth pointing out that the strict diagonal-dominance is a sufficient condition for an eigen-positive matrix. In some cases, a matrix is still eigen-positive, even though it isn't strictly diagonal-dominant. This situation will be discussed later because it often appears in the decoupled load flow and the DC load flow.

Definition 5.4: If a matrix can be written as a block form, and all its non-zero entries are in the diagonal blocks, then the matrix is called the diagonal-decoupled matrix; otherwise, it is called the diagonal-undecoupled matrix.

Theorem 5.3: If A is a symmetric, diagonal-undecoupled and dominant matrix with at least one strict dominant row (column) such that $a_{kk} > \sum_{j \neq k} a_{kj}$ and $a_{kk} > \sum_{j \neq k} a_{jk}$ for some k, then A is an eigen-positive matrix.

Proof: All eigenvalues of A must be real due to the symmetry [HiSm74], *i.e.*, $Im(\lambda_i) \equiv 0$.

According to Gerschgorin Theorem [HiSm74], every eigenvalue λ_i satisfies

$$|\lambda_i - a_{ii}| \leq \sum_{j \neq i} |a_{ij}| \tag{5.22a}$$

and

$$|\lambda_i - a_{ii}| \leq \sum_{j \neq i} |a_{ji}| \tag{5.22b}$$

where a_{ii} is a diagonal entry of the matrix. A graphic interpretation for the above relationship is that λ_i must lie within a union of row-discs (center at a_{ii} and radius = $\sum_{j \neq i} |a_{ij}|$) and column-discs (center at a_{ii} and radius = $\sum_{j \neq i} |a_{ji}|$) in the complex plane.

Using the dominant condition in (5.16a-5.16c), it is concluded that all eigenvalues are in the right half of the complex plane and $\lambda_i \geq 0$, since they are all real. Next, matrix A must be shown to be nonsingular. Without loss of generality, let

$$a_{ii} = \sum_{j \neq i} |a_{ij}| \quad \text{for } i = 1, 2, \cdots, n-1 \tag{5.23a}$$

$$a_{nn} > \sum_{j \neq n} |a_{ij}| \tag{5.23b}$$

using the row elementary transformation reduces A into an upper-triangle matrix. It can be shown that the diagonal dominance holds because of the diagonal un-decoupled assumption on A [HiSm74]. Therefore,

$$a_{ii}^{(k)} \geq \sum_{j \neq i} |a_{ij}^{(k)}| \tag{5.24a}$$

and

$$a_{ii}^{(k)} > 0 \tag{5.24b}$$

where (k) is the kth reduction such that $a_{ij}^{(k)} = 0$ for $j < i$.

The elementary transformation does not change the determinant of the matrix,

$$det(A) = a_{11}^{(0)} a_{22}^{(1)} \cdots a_{nn}^{(n-1)} > 0. \tag{5.25}$$

In other words, there is no zero eigenvalue in A so that

$$\lambda_i > 0 \quad \text{for all } i.$$

Therefore, A is an eigen-positive matrix. □

It is worthy mentioning that the diagonal un-decoupled feature is not a necessary condition for an eigen-positive matrix. For instance, two matrices

$$C_1 = \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & 4 & 5 \end{bmatrix} \quad C_2 = \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

are diagonal decoupled. However, C_1 is singular, and C_2 is nonsingular, because there exists at least one diagonal block with zero eigenvalue in C_1, whereas in C_2 each diagonal block has a strictly dominant row which makes the diagonal block nonsingular. In a power system, a diagonal-decoupled matrix indicates that the system consists of a group of buses in which there only exist connections between the slack bus and the individual groups, and there is no interconnection between the groups. Figure 5.2 shows a power system composed of three groups A_1, A_2, A_3. These groups are connected to the slack bus S. It is obvious that case C_1 never occurs because in each group, there is at least one bus (say bus k) connected to the slack bus so that a_{kk} is strictly dominant in the corresponding block matrix.

Based on Theorem 5.3, two Corollaries can be deduced for the decoupled load flow and the DC load flow.

5.3.3 Decoupled Power Load Flow

Corollary 5.1: Decoupled power (DCF) load flow can be solved by the ANN solver.

Proof: By ignoring the conductance of transmission lines, the ANN solver for DLF can be formed as

$$\dot{\delta} = -\,[-\,V_i \sum_{j \in k(i)} V_j B_{ij} \sin\delta_{ij} - (P_i - P_{Di})] \tag{5.26a}$$

$$\dot{\mathbf{V}} = -\,[-V_i{}^2 B_{ii} - V_i \sum_{j \in k(i)} V_j + B_{ij} \cos\delta_{ij} - (Q_i - Q_{Di})]. \tag{5.26b}$$

Under the assumption of $\sin\delta_{ij} = 0$ for all i, j, the corresponding Jacobian matrix at (0, 1) is

$$J(0,\,1) = \begin{bmatrix} a_{11} & \cdots & a_{1n} & 0 & \cdots & 0 \\ \cdot & \cdots & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdots & \cdot & \cdot & \cdots & \cdot \\ a_{n1} & \cdots & a_{nn} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & d_{11} & \cdots & d_{1n} \\ \cdot & \cdots & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdots & \cdot & \cdot & \cdots & \cdot \\ 0 & \cdots & 0 & d_{n1} & \cdots & d_{nn} \end{bmatrix} \tag{5.27}$$

which is a decoupled matrix. Obviously, every row is dominant, and there exists at least one strictly dominant row in each block. According to Theorems 5.2 and 5.3, the Jacobian matrix is eigen-positive which guarantees that the decoupled power load flow can be solved by the ANN solver. □

5.3.4 The DC Power Load Flow

Corollary 5.2: The DC power load flow (DCLF) can be solved by the ANN solver.

Proof: As mentioned in Section 5.2, the decoupled power flow degenerates to a linear equation. Consequencely, the ANN solver can be formulated as

$$\dot{\delta} = -\,(A\,\delta - \mathbf{P}) \tag{5.28}$$

where

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \cdot \\ \cdot \\ P_n \end{bmatrix}, \quad \delta = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \cdot \\ \cdot \\ \delta_n \end{bmatrix} \tag{5.29}$$

and

$$a_{ij} = \begin{cases} -\sum_{l \in k(i)} B_{il} & (i = j) \\ B_{ij} & (i \neq j). \end{cases} \tag{5.30}$$

Note that the Jacobian matrix is a constant matrix

$$J = A, \tag{5.31}$$

and A is still a diagonally dominant matrix with at least one strict dominant row so that A is eigen-positive. In this case, a unique solution can be found by the ANN solver due to the linear behavior of the system [Mea88]. □

5.3.5 Contingency Analysis

As described in Section 5.2, a kernel problem of the contingency analysis is to solve the power load flow with changed parameters of the power system. An assumption in the contingency analysis is that the whole network remains unisolated before and after outages so that the above topological conditions are unchanged. Therefore, the ANN formulations for the different cases of power load flow can be directly used for the contingency analysis. It is worthy to point out that the multi-outage (the parameter change of transmission lines can be regarded as a specific case of transmission line outage) share the identical ANN formulation with the single-outage. Consequently, the architecture proposed in next Section will handle all the situations of power load flow and contingency analysis.

5.4 Architecture

A theoretical basis for an ANN nonlinear equation solver and the ANN formulations for power load flow have been presented in Section 5.3. In what follows, an architecture of the ANN nonlinear equation solver for power load flow is proposed.

Figure 5.3 shows the architecture of the ANN nonlinear equation solver for the full power load flow. From a structural point of view, the architecture can be divided into two parts: the δ component (the upper part in Figure 5.3) and the V component (the lower part in Figure 5.3). These two components can be viewed as the mapping of equations (5.15a) and (5.15b), respectively. This network is composed of neurons, resistive matrices, analog multipliers, sinusoidal generators, feedback links and DC voltage sources. The neurons are implemented with integrators ($\int$ - units) which govern the dynamical process of the network for finding an equilibrium point. The dynamical relations in (5.15) are held by connecting all current outputs from P, B, G, G_{ll} and Q, G, B, B_{ll} to the inputs of integrators for $\boldsymbol{\delta}$ and $\mathbf{V}$, respectively, where the negative terminal of the integrator is used for implementing the negative sign in (5.15). Another distinct feature of this approach is that the feedback connections B and G are not restricted to symmetric matrices. This significantly broadens its range of applications.

In most power systems, the phase angle δ is usually near zero so that the following polynomials can be used to approximating the sinusoidal function

$$\sin\delta \simeq \delta - \frac{\delta^3}{6} \tag{5.32a}$$

$$\cos\delta \simeq 1 - \frac{\delta^2}{2}. \tag{5.32b}$$

In other words, the sinusoidal generator is replaced by a polynomial generator.

There are three nonlinear operational blocks in the right part of Figure 5.3. All of them are organized in $n \times n$ arrays. In the sinusoidal generator block (Sin/Cos), the outputs on the diagonal are $sin\,\delta_i$ and $cos\,\delta_i$, and the outputs off the diagonal are $sin\,(\delta_i - \delta_j)$ and $cos\,(\delta_i - \delta_j)$, where i is the row-index, j is the column-index. If there exists no connection between bus i and bus j, the output at (i, j) of the array is zero. Similarly, in the voltage multiplication block (M_1), the outputs on the diagonal

are V_i^2, and the outputs off the diagonal are $V_i V_j$ if there exists a connection between bus i and bus j. In the joint multiplication block (M_2), the outputs on the diagonal are $V_i sin\,\delta_i$ and $V_i cos\,\delta_i$, and the outputs off the diagonal are $V_i V_j sin(\delta_i - \delta_j)$, $V_i V_j cos(\delta_i - \delta_j)$ if there exists a connection between bus i and bus j.

As shown in Figure 5.3, V_i^2 from the voltage multiplication block is fedback to the B_{II}, G_{II} network, and all outputs from the joint multiplication block are fedback to the B, G networks. It is necessary to point out that the original power load flow formulation (5.1a, 5.1b) is obtained from decomposition of the complex power so that G, B, P, Q can be regarded as the unitless values. As a consequence, they can be implemented in a resistive matrix network. In general, the dimensions of these matrices are $n \times n$ for G, B, and $n \times 1$ for G_{II}, B_{II}, P, Q. As described in Section 4.5, let the normalization values of current be 0.1 ma, then the value of the resistors can be calculated as

$$R_{G_{ij}} = \frac{10}{|G_{ij}|} k\Omega \tag{5.33}$$

where $R_{G_{ij}}$ is a resistor representing values in matrix G. If $G_{ij} = 0$, $R_{G_{ij}}$ is infinite. In this case, the feedback to the corresponding position is suspended. R_B, $R_{G_{II}}$, $R_{B_{II}}$, R_P and R_Q are calculated and handled in the similar manner.

Let n be the number of buses (excluding the slack bus) in a power system, it is then straightforward to find the number of circuit components. Furthermore, using design data for CMOS technology [AlHo87, Rei87], a space estimation for building this nonlinear equation solver for the full power load flow is given in Table 5.1, where a represents the area for a resistor with an average resistance $50k\Omega$, and b represents the space area for an operational amplifier which is the basic circuit to build the integrator, the sinusoidal generator and the multiplier. Let the routing area be 40% of the circuit area [AlHo87], then the space area of a chip for a n-bus power system is

$$S = 1.4[(18n^2 + 2n)b + (4n^2 + 4n)a]. \tag{5.34}$$

For the decoupled power load flow, an architecture takes a simpler form shown in Figure 5.4. Without G blocks, this architecture makes the remaining same as that of the full load flow shown in Figure 5.3. A space estimation is given in Table 5.2. The space area of a chip for a $n-bus$ power system is

$$S = 1.4[(18n^2 + 2n)b + (2n^2 + 4n)a]. \tag{5.35}$$

For the DC load flow, all the blocks associated with V, G, the sinusoidal generator and multiplications are ignored so that the architecture becomes significantly simple as shown in Figure 5.5. A space estimation is given in Table 5.3. In fact, the space area of a chip for a $n-bus$ power system is the same as that of the linear equation solver described in (4.51). For convenience, it is written here again

$$S = 1.4[(nb + (n^2 + n)a]. \tag{5.36}$$

Using equations (5.34) to (5.36), the actual chip area for a 100-bus power system is $41.38cm^2$, $39.63cm^2$, and $0.905cm^2$ for FLF, DLF, and DCLF, respectively. The chip area for DCLF is much smaller than that of FLF and DLF due to without nonlinear structural components.

5.4.1 Verification

To verify the computational correctness of the proposed neural network and architecture, simulations are fulfilled, where integrators are simulated by a fourth-order Runge-Kutta numerical integration (explicit formulation), the resistive networks are simulated by 1-D and 2-D arrays, and the sinusoidal units are simulated by function-call.

5.4.1.1 The full power load flow

There are two cases in the FLF:

(1) Find both phase angle and voltage amplitude for each bus except the slack bus implying that there are $2n$ variables in the system. This case is called the *PQ−bus* situation because the real power P and the reactive power Q for all buses are specified. This is the general case as was discussed in the previous sections.

(2) Find the phase angle and voltage amplitude for load buses and find the phase angle only for generator buses implying that there are $(2n-g)$ variables in the system, where g is number of generators. (in this case the load bus is a PQ-bus, and the generator bus is a PV-bus [Ber86]) This case is called the *mix−bus* $(PQ-PV)$ situation, because the real power P and the reactive power Q for all loads, and the real power P and the magnitude V for all generator buses are specified. Obviously, this situation is a special case for the full power load flow which can still be covered by the formulation (5.15).

Here two examples are described to show these two situations. The first example is a 7-bus power system, shown in Figure 5.6 representing the *PQ−bus* situation. The original data for transmission lines provided in resistance and reactance (in the parentheses in Figure 5.6), a transformation from impedance to admittance is carried out first. The second example is the 39-bus in New England power system which represents the *mix−bus* situation. The data for the 39-bus system are given in Appendix B.

The convergence for the ANN simulation is determined by

$$|x_i^{(k+1)} - x_i^{(k)}| < \varepsilon \quad \text{for all } i \tag{5.37}$$

where $x_i^{(k)}$ is the value of x_i at the kth iteration. $\varepsilon = 0.1E-3$ and $\varepsilon = 0.1E-7$ are used for the both examples. The time increment used for the Runge-Kutta integration is 0.05 for the 7-bus case and 0.001 for the 39-bus system. The simulation time is defined as

$$T = dt \cdot N \tag{5.38}$$

where N is the number of iterations required for convergence. All starting points are set to 0 and 1 for δ and V, respectively. The equilibrium points are found for all the examples. The simulation time is dependent on ε: for $\varepsilon = 0.1E{-}3$, T is 2.15 and 2.35 for the 7-bus and the 39-bus cases, respectively; for $\varepsilon = 0.1E{-}7$, T is 13.15 and 14.55 for the 7-bus and the 39-bus cases, respectively. Furthermore, T has the same relationship to the initial point as that of linear equation solver described in Section 4.7.1.1. These results and the result from a hardware simulation for the ANN linear equation solver [ChSh92a], whose formulation is the same as that of the DCLF, have shown that the time complexity of this approach is size-independent due to its parallel architecture and circuit dynamic properties.

In order to measure the accuracy, the absolute error E_a, and the relative error E_r are defined as

$$E_a = \left[\sum_{i=1}^{k} (L_i - R_j)^2 \right]^{\frac{1}{2}} \tag{5.39a}$$

$$E_r = \frac{E_a}{\sum_{i=1}^{k} |L_i|} \tag{5.39b}$$

where

$$k = \begin{cases} 2n & \text{for all P-Q buses} \\ 2n - g & \text{for mixed buses} \end{cases} \tag{5.39c}$$

and L_i and R_i are the values for the left hand side and the right hand side of each equation in (5.1a) and (5.1b), respectively. Thus, the absolute error represents the sum of the difference between the left hand sides and the right hand sides, whereas the relative error represents a percentage of the absolute error over the absolute sum of the left hand sides (total real and reactive power).

For comparison, these two examples are solved by the Newton-Raphson method as well. Tables C.1 and C.2 in Appendix C present the simulation results and the

relative errors for the 7-bus system and the 39-bus system from the ANN approach, the Newton-Raphson version and the homotopy method as found in [Guo90]. It is apparent that the accuracy of this approach is best. The trajectories of δ and V, shown in Figures 5.7-5.10, indicate that the network approaches its equilibrium without oscillation.

5.4.1.2 Contingency analysis

A 5-bus power system is chosen to test contingency analysis using the ANN formulations for the decoupled load flow and the DC load flow. Figure 5.11 shows the topology of the system. As mentioned in Section 3, this approach can uniformly fulfill the contingency analysis for all the cases. In Figure 5.11, the dashed line represents the single transmission line outage, the dotted lines represent the multiple transmission line outage, and the numbers in parentheses represent parameter change.

The ANN simulation procedure for contingency analysis is the same as finding solution to the full power load flow. Here the simulation parameters take values: $\varepsilon = 0.1E{-}3$ and $dt = 0.05$. Table 5.4 presents the simulation results. In order to indicate the values of the approximation, the simulation result of the FLF for the same data is given as well. Apparently, the result of the DLF is very close to that of the FLF, whereas the result of DCLF has some difference from the FLF when the actual voltage magnitude has a large deviation from 1.0 for some buses. The security judgement is based on the following constraints:

$$| V_i - V^0 | < 0.05$$

$$| \delta_i - \delta^0 | < 9^\circ$$

Non-security cases are marked (*) in Table 5.4.

5.5 Summary

A new approach for solving problems of power load flow and contingency analysis using an artificial neural network has been proposed. A mapping between the nonlinear equation solution and quadratic-nonlinear minimization problem without constraints has been established. Furthermore, a simple formulation given by equation (5.14) for the artificial neural network has been defined and its stability condition has been identified. The ANN formulations for the full power load flow, the decoupled load flow and the DC load flow are obtained and their stability can be sufficiently guaranteed by using the diagonal dominance property of the power systems. Correspondingly, an ANN architecture for the three formulations is given which can be implemented in VLSI technology. The simulation experiments have shown excellent results for power load flow and contingency analysis. Due to dynamic convergence and distributed parallel processing in ANNs, the time complexity of the approach is size-independent. Therefore, this approach can provide the real-time solution for large-scale systems if it is implemented in hardware.

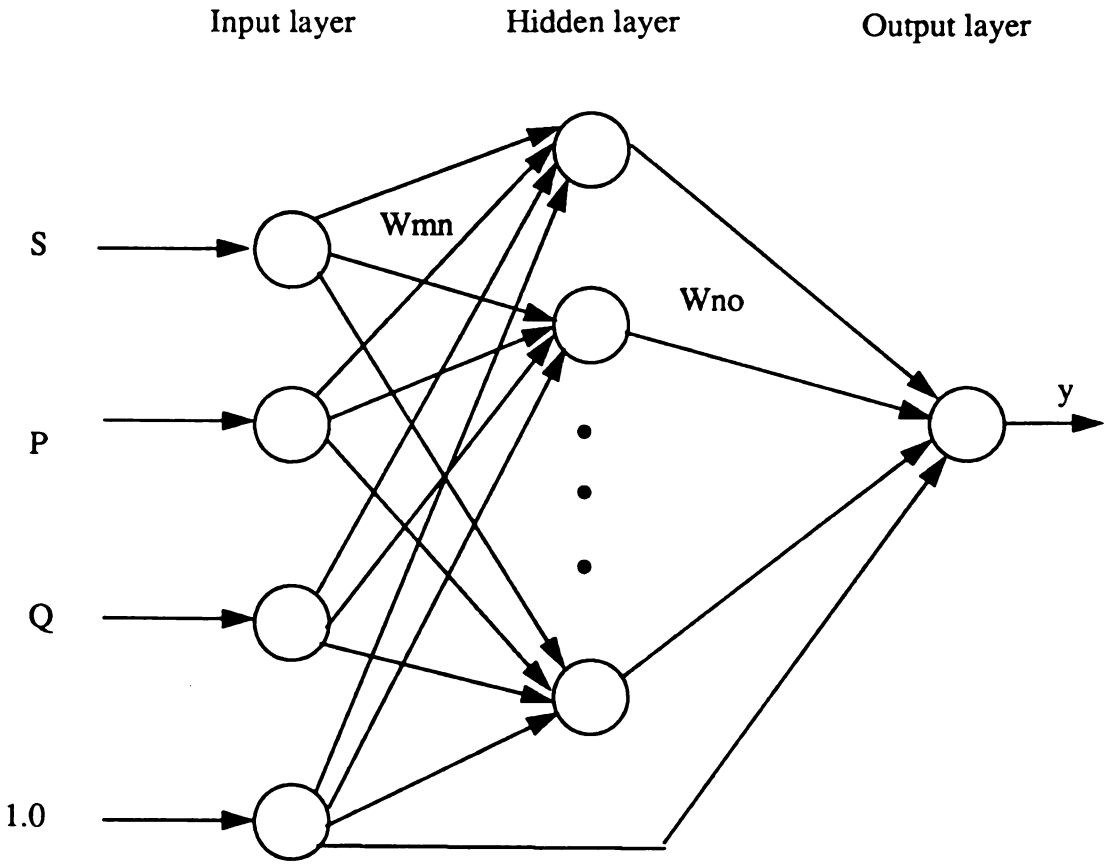


Figure 5.1. The neural network for classification.

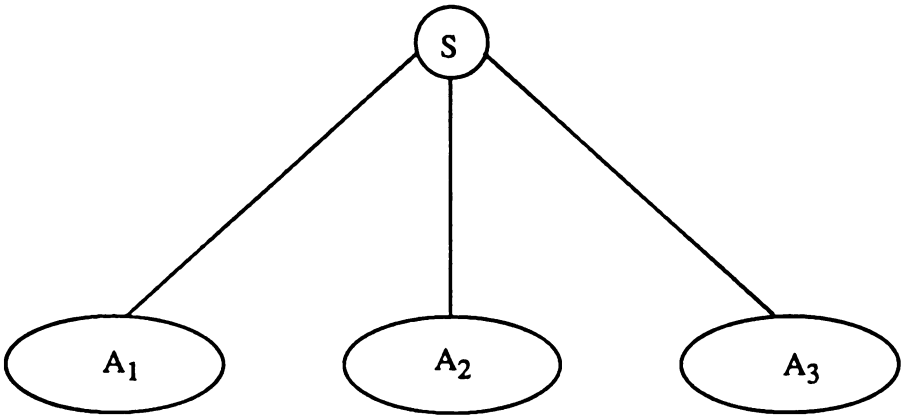


Figure 5.2. A power system consisting of 3 subsystems.

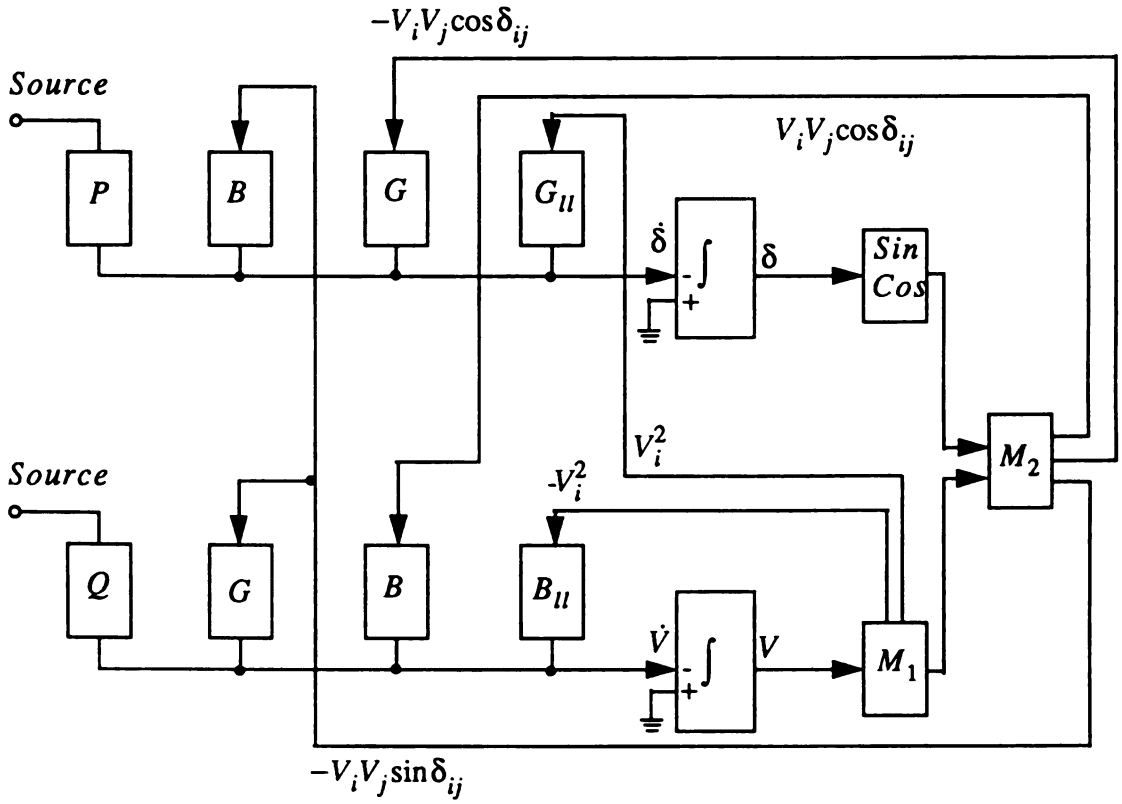


Figure 5.3. A network for the full power load flow.

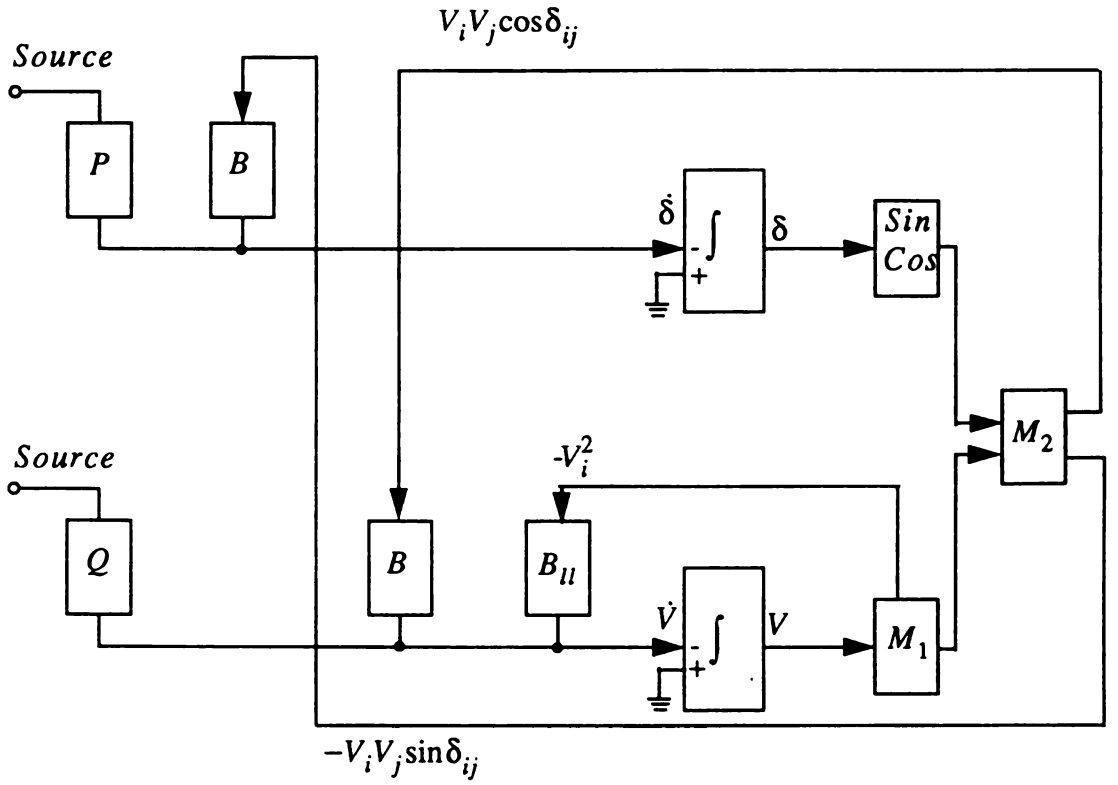


Figure 5.4 . A network for the decoupled load flow.

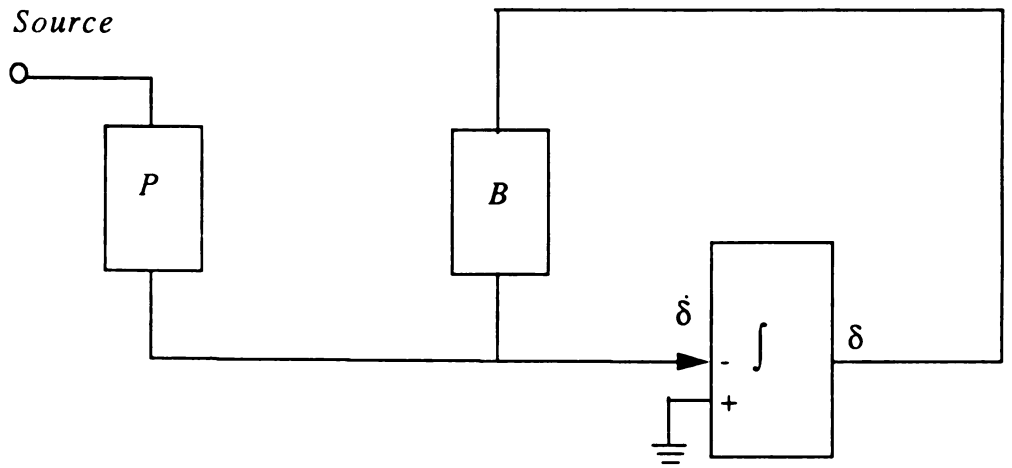


Figure 5.5. A network for the DC load flow.

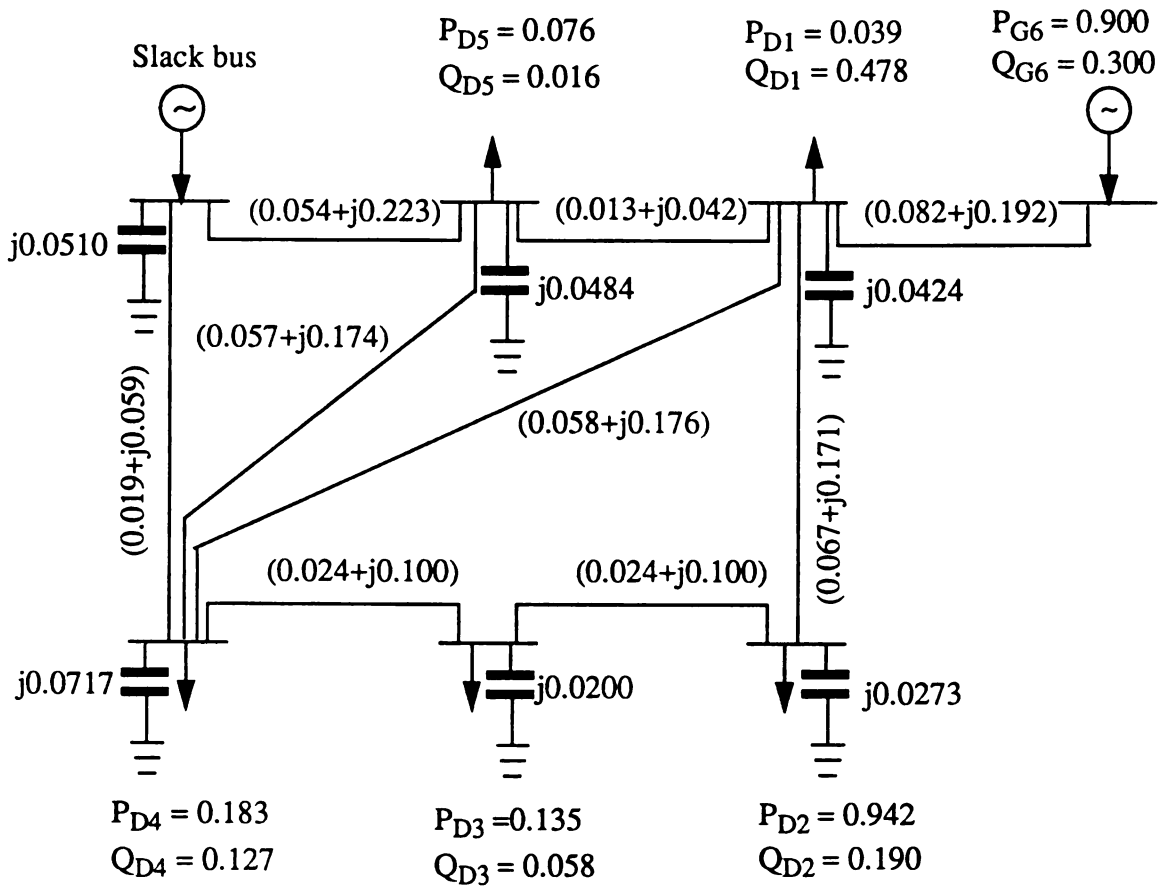


Figure 5.6. The 7-bus system.

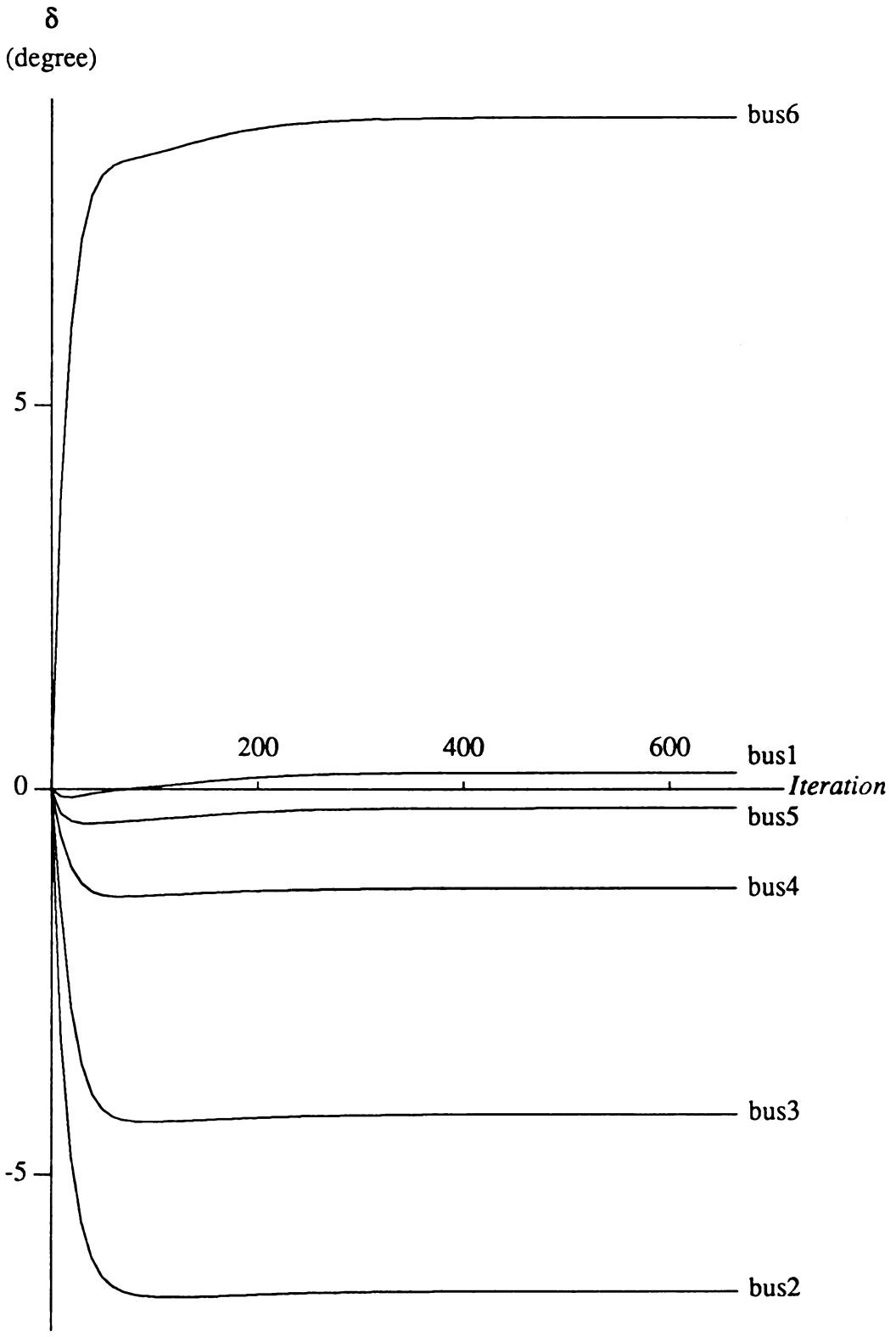


Figure 5.7. The trajectory of angle for the 7-bus system.

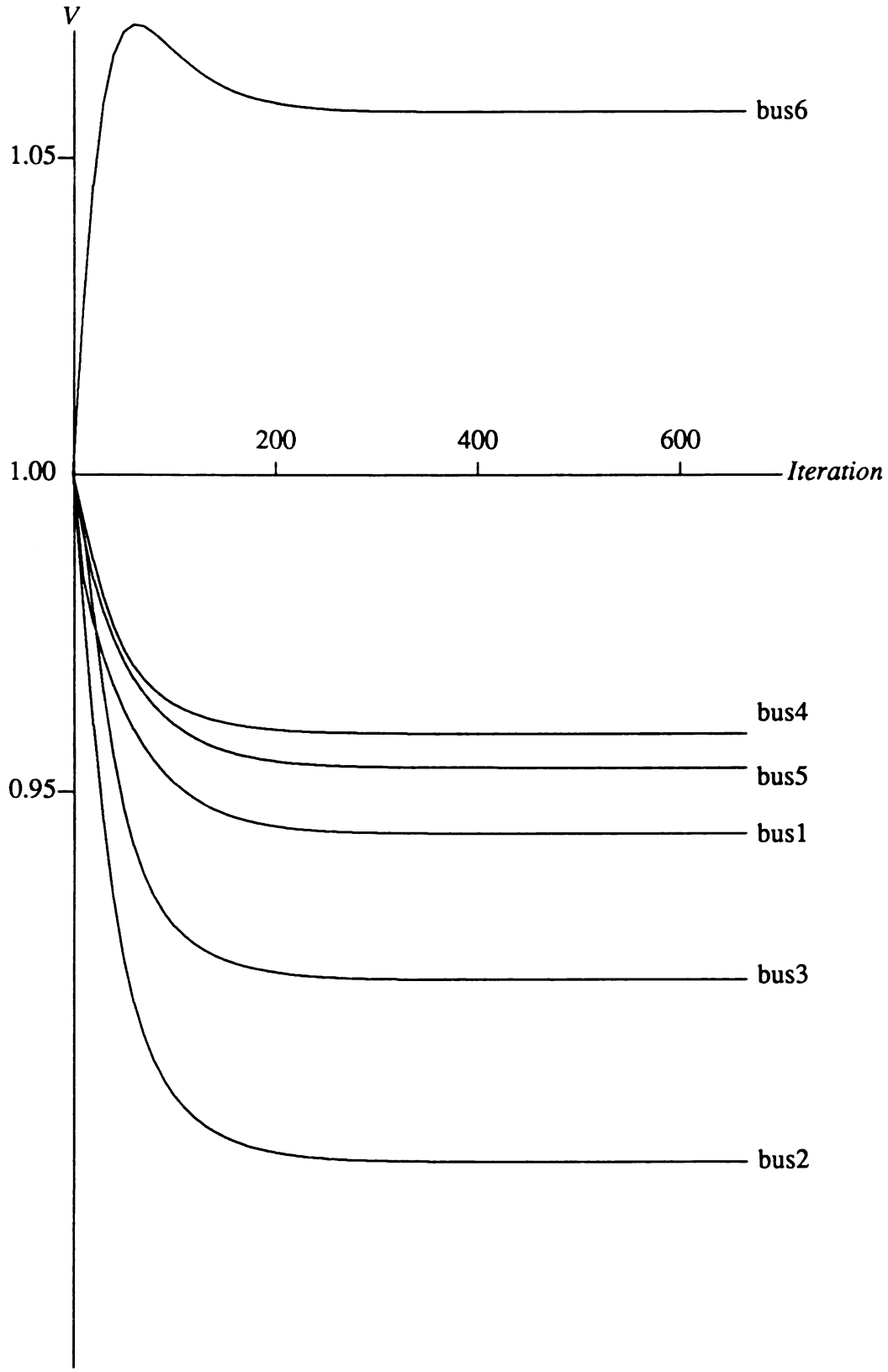
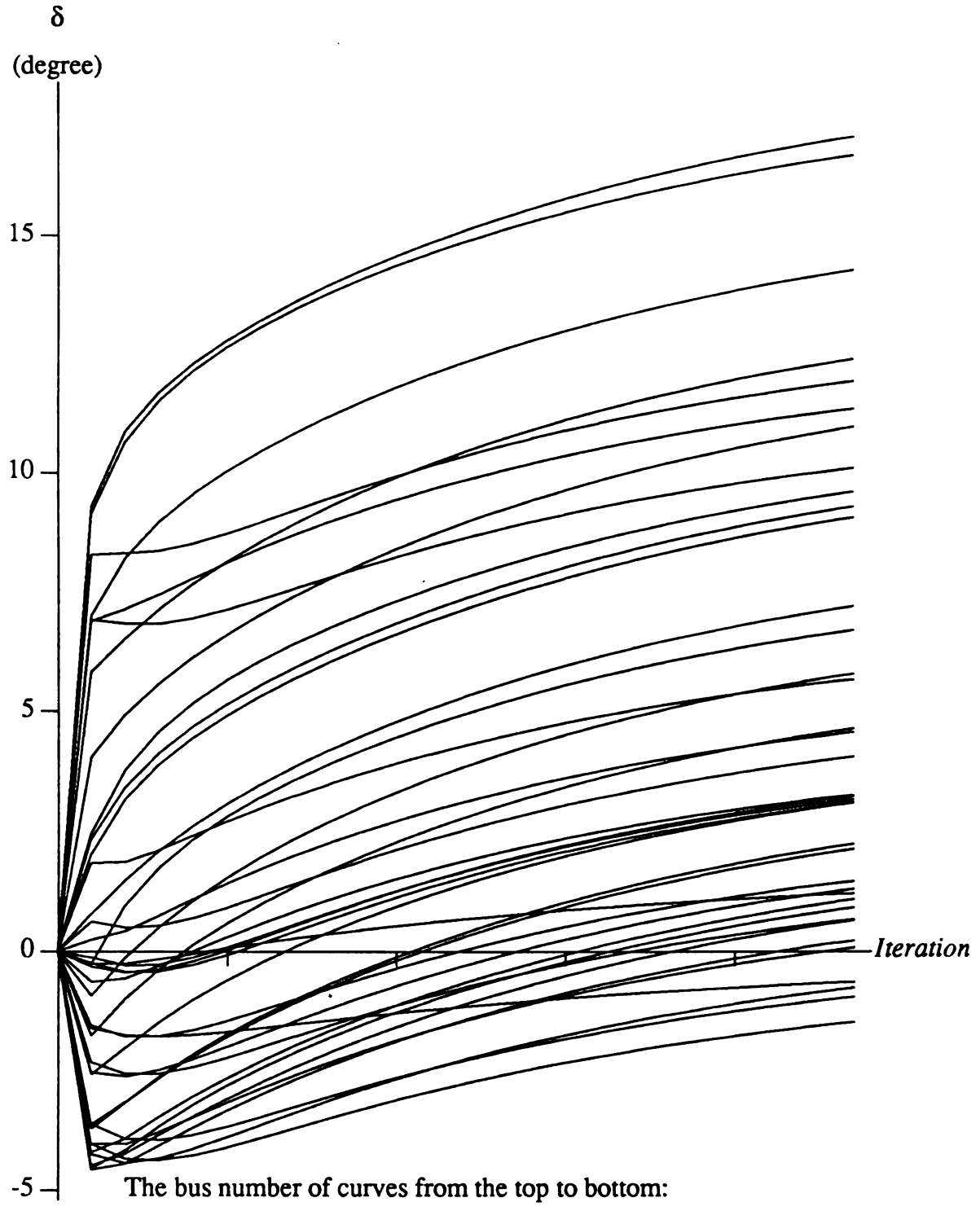
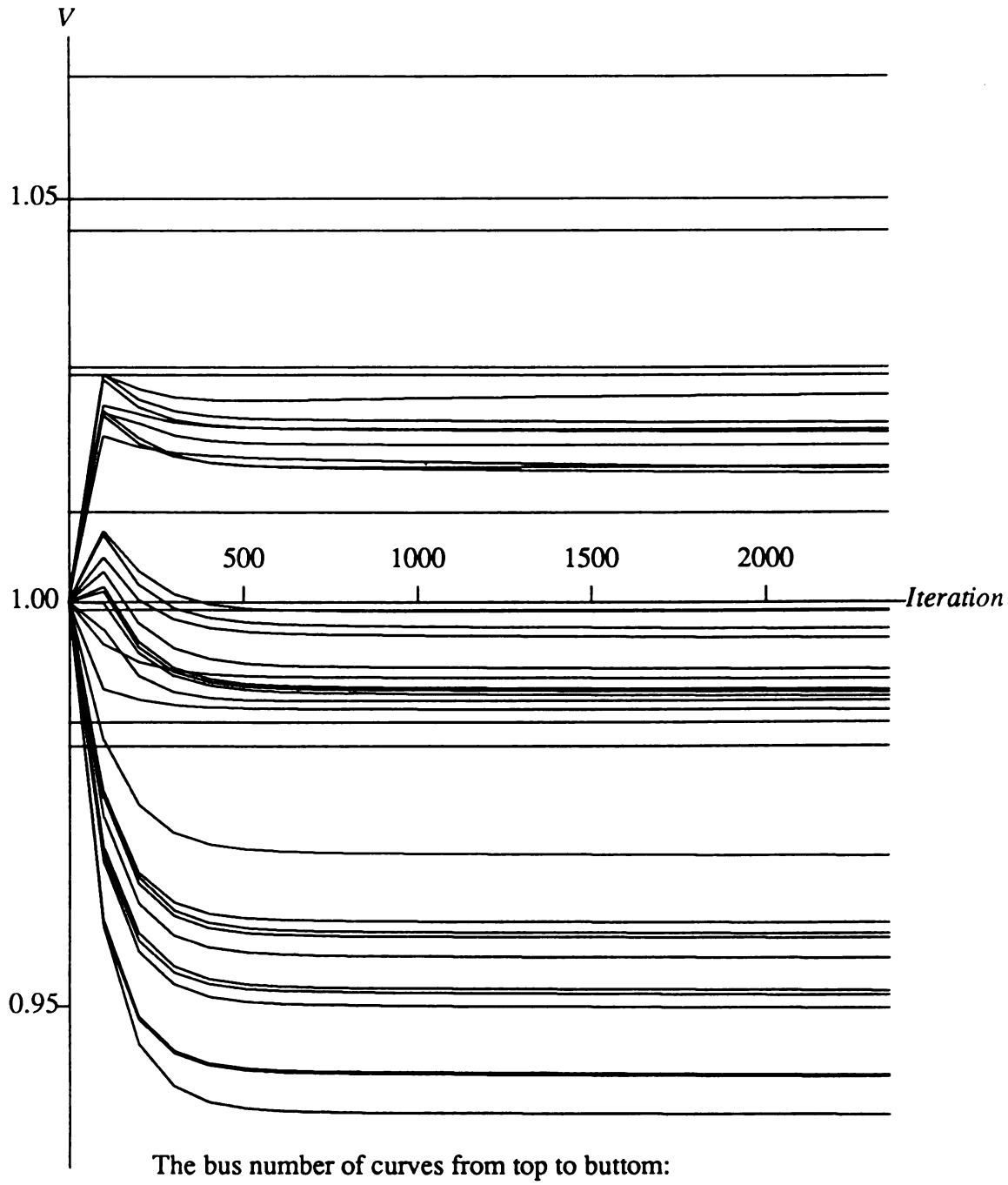


Figure 5.8. The trajectory of voltage for the 7-bus system.



The bus number of curves from the top to bottom:
36, 38, 35, 33, 32, 34, 37, 31, 29, 22, 23, 19, 28, 20, 30, 21, 25, 10, 26
13, 11, 2, 12, 24, 16, 6, 14, 17, 27, 15, 5, 1, 18, 3, 4, 7, 9, 8

Figure 5.9. The trajectory of angle for the 39-bus system.



The bus number of curves from top to buttom:

36, 35, 30, 37, 38, 25, 22, 29, 23, 28, 26, 1, 2, 34, 33, 27, 24, 21, 17

19, 18, 3, 16, 9, 20, 32, 31, 15, 10, 14, 13, 11, 4, 6, 5, 7, 8, 12

Figure 5.10. The trajectory of voltage for the 39-bus system.

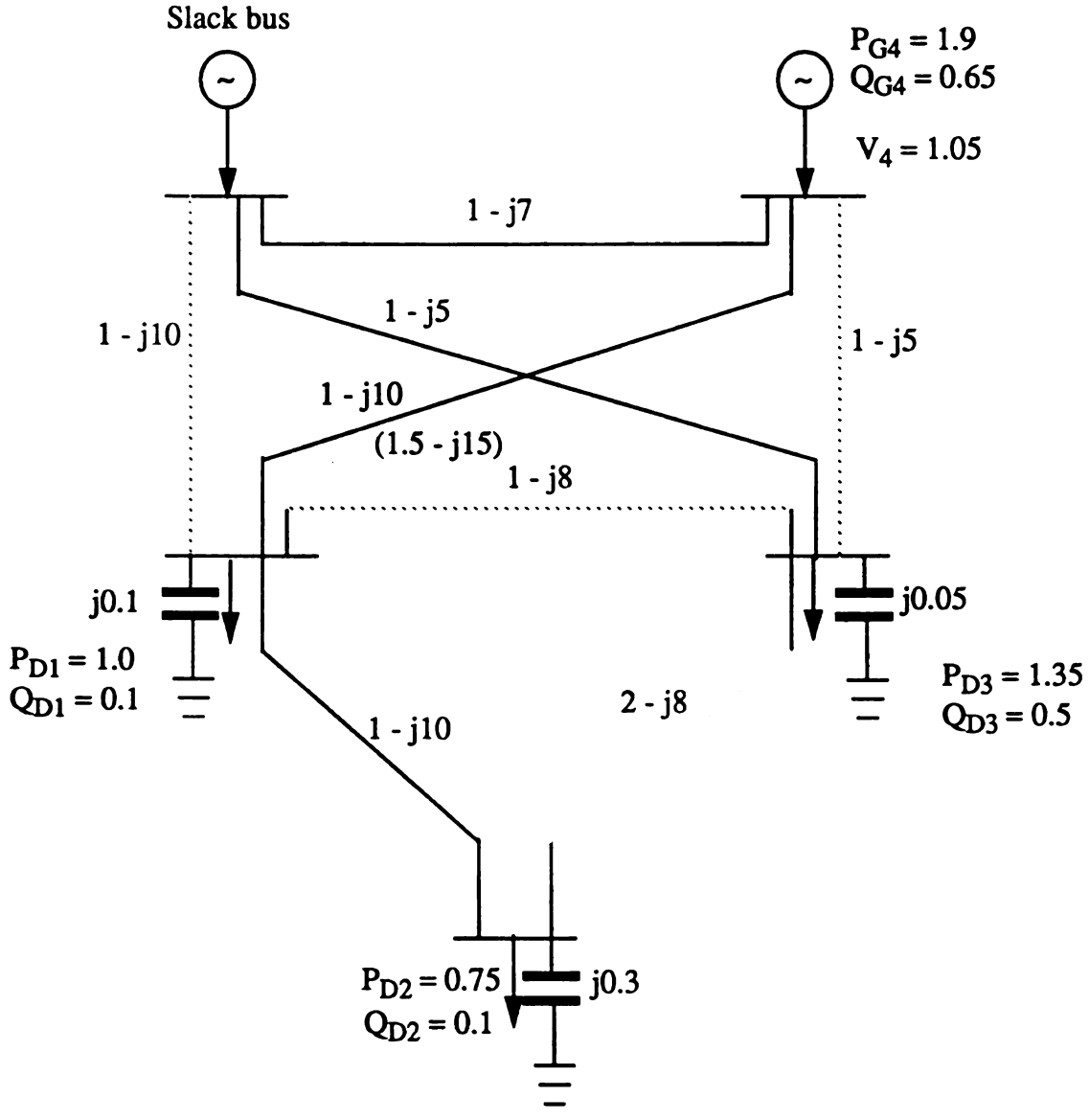


Figure 5.11. The contingency analysis for a 5-bus system.

Table 5.1. A space estaimation of the architecture for FLF.

	Resistors	Integrators	Sin & Cos generators	Multipliers
Count	$4n^2+4n$	$2n$	$2n^2$	$2n^2$
Unit area	a	b	5b	4b
Total area	$(4n^2+4n)a$	$2nb$	$10n^2b$	$8n^2b$

Note: $a=6.25\times10^{-3}mm^2$, $b=1.5\times10^{-2}mm^2$.

Table 5.2. A space estaimation of the architecture for DLF.

	Resistors	Integrators	Sin & Cos generators	Multipliers
Count	$2n^2+4n$	$2n$	$2n^2$	$2n^2$
Unit area	a	b	5b	4b
Total area	$(2n^2+4n)a$	$2nb$	$10n^2b$	$8n^2b$

Table 5.3. A space estaimation of the architecture for DCLF.

	Resistors	Integrators
Count	n^2+n	n
Unit area	a	b
Total area	$(n^2+n)a$	nb

Table 5.4. The contingency analysis.

Type	FLF	DLF	DCLF
Before	1.0323/-4.6884 0.9971/-7.9609 0.9805/-6.4424 1.0500/0.6858	1.0154/-4.4897 1.0189/-7.6689 0.9996/-6.4237 1.0500/1.1810	1.0/-4.5878 1.0/-7.8452 1.0/-6.5378 1.0/1.3753
Single outage	1.0067/-4.9227 1.0171/-9.1931 * 0.9741/-5.9313 1.0500/0.6871	1.0185/-4.7066 1.0373/-8.7828 * 0.9930/-5.9910 1.0500/1.1818	1.0/-2.2617 1.0/-3.6459 1.0/-3.0517 1.0/3.2250
Multi-outage	0.9936/-4.0240 0.9581/-10.3693 * 0.9050/-12.2557 * 1.0500/3.3805	1.0061/-3.8121 0.9857/-10.0404 * 0.9442/-12.5443 * 1.0500/3.8428	1.0/-1.8823 1.0/-5.5124 1.0/-4.6721 1.0/4.0918
Parameter change	1.0120/-4.1638 1.0051/-7.5489 0.9864/-6.2728 1.0500/-0.0180	1.0214/-3.9919 1.0236/-7.3278 1.0032/-6.3175 1.0500/0.3987	1.0/-5.6934 1.0/-8.9020 * 1.0/-7.5798 1.0/-0.6439

Chapter 6

Conclusions

New approaches for finding an accurate solution to linear and nonlinear programming and solving power system control problems using artificial neural network techniques have been described in this dissertation. This chapter starts with a summary of the research work, followed by an identification of the major contributions of this work. A discussion of future research issues concludes the chapter.

6.1 Summary

Improving the performance of ANNs for linear and nonlinear programming, and developing a fast approach for large-scale system computations are challenging problems in neurocomputing. Based on optimization theory and ANN technology, promising solutions to these problems have been obtained from this work.

The computing accuracy of constrained optimization from the previous ANN models does not meet the demand for some applications. By analyzing the behavior of the conventional penalty function in the neighborhood of the feasible region boundary, which is often used in these ANN models, the reason for its inherent degenerating accuracy has been discovered. Based on this, a new combination penalty function has been developed. This function can provide enough "penalty" to offset the decreasing of

$\phi(\mathbf{x})$ in both the range of $l(\mathbf{x}_b)$ and far away from the feasible region boundary so that an equilibrium point close to the optimal point can be located. A corresponding neural network formulation has been given whose stability is guaranteed by the assumption of a convex objective function and the continuously differentiable property of the combination penalty function. Based on unconstrained optimization, an ANN-based technique has been developed for solving simultaneous equations. For the linear system, a mapping between the solution of linear equations and quadratic minimization has been established so that solving linear equations can be viewed as finding the minimum of a quadratic function. A general formulation for the ANN linear equation solver has been defined, and the stability of the network has been established. A relationship between the ANN solver and the linear dynamic system is identified, broadening the scope of potential applications. An additional merit of the formulation is that symmetry of the coefficient matrix is not required, again broadening the range of applications. For the nonlinear system, a mapping between the solution of nonlinear equations and quadratic-nonlinear minimization has been established so that solving nonlinear equations can be viewed as finding the minimum of a quadratic-nonlinear function. A formulation for the ANN nonlinear equation solver has been defined and proofs of local stability and convergence of the solver have been given which provide a sound theoretical basis.

Based on these formulations, ANN architectures have been developed at the algorithm and structural level. The feedback neural network paradigm was used to express the dynamic behavior of the objective function. A constraint amplifier circuit for the derivative of the combination penalty function has been proposed for the constrained optimization network. The amplifier is simpler than previous versions due to the replacement of the radication operation. Another distinct feature of the architecture for the linear and nonlinear equation solvers is that the resistance connection matrix can be symmetric or asymmetric. The hardware complexity of the solvers has been estimated

as $O(n^2a + nb)$ for the linear equation solver, and $O(n^2a + n^2b)$ for the nonlinear equation solver, where $a(=6.25\times10^{-3}mm^2)$ is the unit area for a resistor, and $b(=1.5\times10^{-2}mm^2)$ is the estimated unit area per neuron.

The ANN linear and nonlinear equation solvers have be applied to solving the example power system control problems of power load flow and contingency analysis. The specific ANN formulations for the full power load flow, the decoupled load flow and the DC load flow have been established. The network stability for these formulations has been further analyzed. Moreover, the usefulness of the ANN linear equation solver has been extended. Other applications include matrix inversion, determining the stability of a linear control system, and determining matrix singularity.

The computational correctness of the approaches and architectures have been verified by simulations and it was discovered that the relative errors for linear and nonlinear programming examples are substantially reduced.

6.2 Contributions

The major contributions of this research are:

(1) An improved artificial neural network for linear and nonlinear programming has been developed. The reason for the degenerating computing accuracy of previous networks has been discovered and a new combination penalty function has been proposed. Both play key roles in improving the computational accuracy for ANN constrained optimization.

(2) This work has established a new link between ANN theory and unconstrained optimization. Correspondingly, an ANN linear equation solver has been developed at the algorithm and structural level, and a theoretical basis for the ANN nonlinear equation solver has been given.

(3) A direct ANN method (without training and testing) for power load flow and contingency analysis has been developed. The ANN formulations, their stability property and the relationship with the parameters of a power system have been provided.

6.3 Future Research

To implement the approaches proposed in this dissertation, a new method for a variable resistor connection matrix is needed. Recently, experimental programmable arrays have been reported for this purpose [MaHY90, FiFS91]. However, their adjustable range is too narrow to meet the demand of real problems. Studies in this area are continuing. Secondly, to take advantage of the program flexibility of the conventional computer, and the real-time computing property of the ANN equation solver, a hybrid interface should be developed. This interface should be able to provide data communication, A/D and D/A conversions, and computing task control between a digital computer and a neural network. Another issue in implementation is to explore a new method for handling the I/O problem for large systems; system decomposition being the most likely starting point.

Feedback network based learning is another challenging issue as the capability of the network will be greatly enhanced with built-in learning. There have been very few implementable algorithms presented in this area so far [BaAC91, SuCL91]. New research work includes some implementation-oriented efforts in architecture and device development.

Finally, the ANN linear equation solver described in this thesis may be further used for finding eigenvalues and associated eigenvectors. A relationship between state trajectories of a neural network and eigenvectors of the corresponding coefficient matrix can be obtained by expanding the matrix singularity analysis. Based on this

relationship and numerical techniques, eigenvalues can been found by solving a set of linear equations.

APPENDIX A

Proof of Theorem 5.1

Let $\mathbf{x}^*$ be an equilibrium point. After change of variable

$$\mathbf{y} = \mathbf{x} - \mathbf{x}^*, \tag{a.1}$$

equation (5.14) can be written as

$$\dot{\mathbf{y}} = \dot{\mathbf{x}} = -\mathbf{f}(\mathbf{y} + \mathbf{x}^*) = -\mathbf{q}(\mathbf{y}). \tag{a.2}$$

Note that $\mathbf{y} = 0$ is an equilibrium point of the system. It is easy to show that the following equality is held

$$J(\mathbf{x} = \mathbf{x}^*) = J(\mathbf{y} = 0). \tag{a.3}$$

Using the mean value theorem of derivative [Apo57], $\mathbf{q}(\mathbf{y})$ can be written as

$$\begin{aligned} \mathbf{q}(\mathbf{y}) &= \mathbf{q}(0) + \frac{\partial \mathbf{q}}{\partial \mathbf{y}}(\mathbf{c})\mathbf{y} \qquad (a.4) \\ &= 0 + \frac{\partial \mathbf{q}}{\partial \mathbf{y}}(\mathbf{c})\mathbf{y} \\ &= \frac{\partial \mathbf{q}}{\partial \mathbf{y}}(0)\mathbf{y} + \left[\frac{\partial \mathbf{q}}{\partial \mathbf{y}}(\mathbf{c}) - \frac{\partial \mathbf{q}}{\partial \mathbf{y}}(0)\right]\mathbf{y} \\ &= J\,\mathbf{y} + \mathbf{g}(\mathbf{y}) \end{aligned}$$

where

$$J = \frac{\partial \mathbf{q}}{\partial \mathbf{y}}(0), \quad \mathbf{g}(\mathbf{y}) = \left[\frac{\partial \mathbf{q}}{\partial \mathbf{y}}(\mathbf{c}) - \frac{\partial \mathbf{q}}{\partial \mathbf{y}}(0)\right]\mathbf{y} \tag{a.5}$$

and $\mathbf{c}$ is a point on a line segment from the origin to $\mathbf{x}$. According to the norm inequality relation and the assumption of continuity of the derivative, the following qualitative expressions can be obtained:

$$|| \mathbf{g}(\mathbf{y}) || \leq || \frac{\partial \mathbf{q}}{\partial \mathbf{y}}(\mathbf{c}) - \frac{\partial \mathbf{q}}{\partial \mathbf{y}}(0) || \cdot || \mathbf{y} ||, \tag{a.6}$$

$$\lim_{|| \mathbf{y} || \mapsto 0} \frac{|| \mathbf{g}(\mathbf{y}) ||}{|| \mathbf{y} ||} = 0, \tag{a.7}$$

and

$$|| \mathbf{q}(\mathbf{y}) || \leq \sigma || \mathbf{y} ||. \tag{a.8}$$

An eigen-positive J guarantees there must exist a solution P to the Lyapunov equation

$$PJ + J^T P = Q \tag{a.9}$$

where P, Q are positive definite matrices [BaSt70].

Define the energy function as

$$E(\mathbf{y}) = \mathbf{y}^T P \mathbf{y} \tag{a.10}$$

where $E(\mathbf{y}) > 0$ for $\mathbf{y} \neq 0$. The derivative of the energy function is

$$\begin{aligned} \dot{E}(\mathbf{y}) &= \dot{\mathbf{y}}^T P \mathbf{y} + \mathbf{y} P \dot{\mathbf{y}} \qquad (a.11) \\ &= (-J\mathbf{y} - \mathbf{g}(\mathbf{y})^T P y + \mathbf{y}^T P (-J\mathbf{y} - \mathbf{g}(\mathbf{y}) \\ &= -\mathbf{y}^T J^T P \mathbf{y} + \mathbf{y}^T P J \mathbf{y} + \mathbf{g}^T(\mathbf{y}) + \mathbf{y}^T P \mathbf{g}(\mathbf{y}) \\ &= -\mathbf{y}^T (J^T P + P J)\mathbf{y} + 2\mathbf{g}(\mathbf{y})^T P \mathbf{y} \\ &= -\mathbf{y}^T Q \mathbf{y} + 2\mathbf{g}^T(\mathbf{y}) P \mathbf{y} \\ &\leq -\mathbf{y}^T Q \mathbf{y} + 2\sigma || P || \cdot || \mathbf{y} ||_2^2 \\ &\leq -\lambda_{\min}(Q) \cdot || \mathbf{y} ||_2^2 + 2\sigma || P || \cdot || \mathbf{y} ||_2^2 \\ &= -(\lambda_{\min}(Q) - 2\sigma || P ||) \cdot || \mathbf{y} ||_2^2 \end{aligned}$$

where

$$|| P || = (\lambda_{\max}(P^T P))^{\frac{1}{2}} = \lambda_{\max}(P), \tag{a.12}$$

$\lambda_{min}(Q)$ is the minimum eigenvalue for Q, and $\lambda_{max}(P)$ is the maximum eigenvalue for P. When σ satisfies

$$\sigma < \frac{\lambda_{min}(Q)}{2\,||P||}, \tag{a.13}$$

then

$$\dot{E}(\mathbf{y}) < 0. \tag{a.14}$$

Without loss of generality, let Q be the identity matrix, then $\lambda_{min}(Q) = 1$. Inequality (3.18) can be further simplified as

$$\sigma < \frac{1}{2\,||P_I||}, \tag{a.15}$$

where P_I is the positive definite matrix determined by equation (3.14) for $Q = I$.

Inequality (a.15) guarantees that the energy function $E(\mathbf{y})$ is a Lyapunov function. Therefore, the network based on equation (5.14) is stable in the neighborhood of the origin in y-coordinate. When a starting point satisfies the condition

$$||\mathbf{q}(\mathbf{y})|| < \frac{||\mathbf{y}||}{2\,||P_I||}, \tag{a.16}$$

the dynamical system will approach an equilibrium point, the origin ($\mathbf{y}_e = 0$) due to change of variable. In x-coordinate, the equilibrium point is expressed

$$\mathbf{x}_e = \mathbf{y}_e + \mathbf{x}^*, \tag{a.17}$$

which is the exact solution to the nonlinear equation. □

APPENDIX B

Table B.1. The bus data for the 39-bus system.

Number	Name	P_D	Q_D	P_G	Q_G	$V_{specified}$
0	01THETAG					1.000
1	01ALPHA	0.0	0.0	0.0	0.0	
2	01KAPPA	0.0	0.0	0.0	0.0	
3	03ETA	3.22	0.024	0.0	0.0	
4	03THETA	5.00	1.84	0.0	0.0	
5	03IOTA	0.0	0.0	0.0	0.0	
6	07GAMMA	0.0	0.0	0.0	0.0	
7	06LAMMBDA	2.338	0.84	0.0	0.0	
8	07MU	5.22	1.766	0.0	0.0	
9	05NU	0.0	0.0	0.0	0.0	
10	05XI	0.0	0.0	0.0	0.0	
11	05MICRON	0.0	0.0	0.0	0.0	
12	06PI	0.085	0.88	0.0	0.0	
13	07RHO	0.0	0.0	0.0	0.0	
14	07SIGMA	0.0	0.0	0.0	0.0	
15	07TAU	3.20	1.53	0.0	0.0	
16	06EPSILON	3.294	0.323	0.0	0.0	
17	06PHI	0.0	0.0	0.0	0.0	
18	07CHI	1.58	0.30	0.0	0.0	
19	04PSI	0.0	0.0	0.0	0.0	
20	04OMEGA	6.80	1.03	0.0	0.0	
21	03ALPHA	2.74	1.15	0.0	0.0	
22	06BETA	0.0	0.0	0.0	0.0	
23	07BETA	2.47	0.846	0.0	0.0	
24	06DELTA	3.086	-0.922	0.0	0.0	
25	01GAMMA	2.24	0.472	0.0	0.0	
26	01DELTA	1.39	0.17	0.0	0.0	
27	04EPSILON	2.81	0.755	0.0	0.0	
28	01EPSILON	2.06	0.276	0.0	0.0	
29	01ZETA	2.835	0.269	0.0	0.0	
30	01KAPPAG	0.0	0.0	2.50	1.362	1.046
31	07GAMMAG	0.092	0.046	5.73	2.0458	0.982
32	04XIG	0.0	0.0	6.5	2.1036	0.985
33	04PSIG	0.0	0.0	6.32	1.1579	0.999
34	04OMEGAG	0.0	0.0	5.08	1.6	1.011
35	06BETAG	0.0	0.0	6.5	2.0967	1.050
36	07BETAG	0.0	0.0	5.6	1.0214	1.065
37	01GAMMAG	0.0	0.0	5.4	0.0409	1.029
38	01ZETAG	0.0	0.0	8.3	0.236	1.028

Table B.2. The branch data for the 39-bus system.

Number	Bus_a	Bus_b	R	X	B_t
1	1	2	0.0035	0.0411	0.6987
2	1	0	0.0010	0.0250	0.7500
3	2	3	0.0013	0.0151	0.2572
4	2	25	0.0070	0.0086	0.1460
5	2	30	0.0000	0.0181	0.0000
6	3	4	0.0013	0.0213	0.2214
7	3	18	0.0011	0.0133	0.2138
8	4	5	0.0008	0.0128	0.1342
9	4	14	0.0008	0.0129	0.1382
10	5	6	0.0002	0.0026	0.0434
11	5	8	0.0008	0.0112	0.1476
12	6	7	0.0006	0.0092	0.1130
13	6	11	0.0007	0.0082	0.1389
14	6	31	0.0000	0.0250	0.0000
15	7	8	0.0004	0.0046	0.0780
16	8	9	0.0023	0.0363	0.3804
17	9	0	0.0010	0.0250	1.2000
18	10	11	0.0004	0.0043	0.0729
19	10	13	0.0004	0.0043	0.0729
20	10	32	0.0000	0.0200	0.0000
21	12	11	0.0016	0.0435	0.0000
22	12	13	0.0016	0.0435	0.0000
23	13	14	0.0009	0.0101	0.1723
24	14	15	0.0018	0.0217	0.3660
25	15	16	0.0009	0.0094	0.1710
26	16	17	0.0007	0.0089	0.1342
27	16	19	0.0016	0.0195	0.3040
28	16	21	0.0008	0.0135	0.2548
29	16	24	0.0003	0.0059	0.0680
30	17	18	0.0007	0.0082	0.1319
31	17	27	0.0013	0.0173	0.3216
32	19	20	0.0007	0.0138	0.0000
33	19	33	0.0007	0.0142	0.0000
34	20	34	0.0009	0.0180	0.0000
35	21	22	0.0008	0.0140	0.2565
36	22	23	0.0006	0.0096	0.1846
37	22	35	0.0000	0.0143	0.0000
38	23	24	0.0022	0.0350	0.3610
39	23	36	0.0005	0.0272	0.0000
40	25	26	0.0032	0.0323	0.5130
41	25	37	0.0006	0.0232	0.0000
42	26	27	0.0014	0.0147	0.2396
43	26	28	0.0043	0.0474	0.7802
44	26	29	0.0057	0.0625	1.0290
45	28	29	0.0014	0.0151	0.2490
46	29	38	0.0008	0.0156	0.0000

APPENDIX C

Table C.1 The 7-bus system comparison results.

Method	ANN	Newton	Homotopy
Bus 0: v_0/δ_0	1.00/0.00	1.00/0.00	1.00/0.00
Bus 1: v_1/δ_1	0.9435/0.2153	0.9435/0.22	0.9775/-3.2639
Bus 2: v_2/δ_2	0.8922/-6.4871	0.8923/-6.48	0.9203/-8.2981
Bus 3: v_3/δ_3	0.9207/-4.2131	0.9208/-4.21	0.9416/-5.9225
Bus 4: v_4/δ_4	0.9591/-1.2851	0.9592/-1.28	0.9732/-2.7507
Bus 5: v_5/δ_5	0.9538/-0.2468	0.9539/-0.25	0.9802/-2.9283
Bus 6: v_6/δ_6	1.0573/8.7592	1.0573/8.76	1.0887/5.0118
Relative Error	ε_1: 0.1518% ε_2: 0.0007%	0.4788%	34.1466%

Note: (1) $\varepsilon_1 = 0.1E-3$, $\varepsilon_2 = 0.1E-7$

(2) δ unit: degree.

(3) Bus 0 is the slack bus.

Table C.2 The 39-bus system comparison results.

Method	ANN	Newton
Bus 0 (slack bus)	1.00/0.00	1.00/0.00
Bus 1	1.0158/1.6351	1.0158/1.64
Bus 2	1.0154/4.3654	1.0154/4.37
Bus 3	0.9885/1.3287	0.9885/1.33
Bus 4	0.9516/0.4860	0.9516/0.49
Bus 5	0.9494/1.8284	0.9494/1.83
Bus 6	0.9510/2.6193	0.9510/2.62
Bus 7	0.9411/0.1487	0.9411/0.15
Bus 8	0.9409/-0.4189	0.9409/-0.42
Bus 9	0.9875/-0.2177	0.9875/-0.22
Bus 10	0.9599/5.2923	0.9599/5/29
Bus 11	0.9556/4.3801	0.9556/4.38
Bus 12	0.9363/4.3576	0.9363/4.36
Bus 13	0.9581/4.4814	0.9581/4.48
Bus 14	0.9585/2.5919	0.9585/2.59
Bus 15	0.9681/2.0828	0.9681/2.08
Bus 16	0.9880/3.5988	0.9879/3.60
Bus 17	0.9914/2.5000	0.9914/2.50
Bus 18	0.9888/1.5884	0.9888/1.59
Bus 19	0.9903/8.8008	0.9903/8.80
Bus 20	0.9865/7.3955	0.9865/7.40
Bus 21	0.9952/6.1719	0.9952/7.17
Bus 22	1.0220/10.8770	1.0220/10.88
Bus 23	1.0208/10.6511	1.0208/10.65
Bus 24	0.9964/3.7238	0.9964/3.72
Bus 25	1.0258/5.7526	1.0258/5.75
Bus 26	1.0167/4.4514	1.0167/4.45
Bus 27	0.9988/2.3169	0.9988/2.32
Bus 28	1.0194/8.1715	1.0194/8.17
Bus 29	1.0213/11.0836	1.0213/11.08
Bus 30 (generator)	1.0460/6.8071	1.0460/6.81
Bus 31 (generator)	0.9820/11.3000	0.9820/11.30
Bus 32 (generator)	0.9850/13.1951	0.985/13.20
Bus 33 (generator)	0.9990/13.9817	0.9990/13.20
Bus 34 (generator)	1.0110/12.5860	1.0110/12.59
Bus 35 (generator)	1.0500/15.8461	1.0500/15.85
Bus 36 (generator)	1.0650/18.6514	1.0650/18.65
Bus 37 (generator)	1.0290/12.5581	1.0290/12.56
Bus 38 (generator)	1.0280/18.1446	1.0280/18.15
Relative Error	ε_1: 4.47% ε_2: 3.04%	6.49%

Note: (1) $\varepsilon_1 = 0.1E-3$, $\varepsilon_2 = 0.1E-7$.

(2) voltage/angle, δ unit: degree.

BIBLIOGRAPHY

[Ag*et*89] M. E. Aggoune, *et al.*, "Artificial Neural Networks for Power System Static Security Assessment," *ISCAS*, pp. 490-494, 1989.

[AiNF90] S. V. B. Aiyer, M. Niranjan, and F. Fallside, "A Theoretical Investigation into the Performance of the Hopfield Model," *IEEE Trans. on Neural Networks*, Vol. 1, No. 2, pp204-215, June, 1990.

[AlHo87] P. E. Allen and D. R. Holberg, CMOS Analog Circuit Design, Holt, Rinehart & Winston, New York, 1987.

[Ang89] J. E. Angus, "On the Connection between Neural Network Learning and Multivariate Nonlinear Least Squares Estimation," *Neural Networks,* Vol. 1, No. 1, January, 1989.

[AnRo88] J. A. Anderson and E. Rosefeld, Neurocomputing: Foundations of Research, The MIT Press, Cambridge, MA, 1988.

[Apo57] T. M. Apostol, Mathematical Analysis, Addison-Wesley, Reading, Massachusetts, 1957.

[AtSu89] L. E. Atlas and Y. Suzuki, "Digital Systems for Artificial Neural Networks," *IEEE Circuits and Devices Magazine*, November, 1989.

[BaAC91] V. C. Barbosa, L. Alfredo, and V. Carvalho, "Learning in Analog Hopfield Networks," *Proceedings of the International Joint Conference on Neural Networks*, II, pp.183-186, 1991.

[BaEM89] M. L. Baughman, N. A. Eisner and P. S. Merrill, "Optimizing Combined Cogeneration and Thermal Storage System: An Engineering Economic Approach," *IEEE Trans. on Power Systems,* Vol. 4, pp. 974-980, August 1989.

[BaSh79] M. S. Bazaraa and C. M. Shetty, Nonlinear Programming Theory and Application, John Wiley & Sons, New York, 1979.

[BaSt70] S. Barnett and C. Storey, Matrix Methods in Stability Theory, Barnes & Noble, Inc., New York, 1970.

[BiVo91] R. H. Bisseling and J. G. Vorst, "Parallel Triangular System Solving on a Mesh Network of Transputers," *SIAM Journal on Scientific and Statistical Computing*, Vol. 12, No. 4, pp. 787-799, July 1991.

[Bla86] K. Blanchard, The One-Minute Manager, Morrell Publishing, New York, 1986.

[Bra91] V. Brandwajn, *et al.*, "Pre-Screening of Single Contingencies Causing Network Topology Changes," *IEEE Trans. on Power Systems,* Vol. 6, No. 1, pp. 30-36, Feb. 1991.

[BuHW82] R. C. Burchett, H. H. Happ and K. A. Wirgau, "Large Scale Optimal Power Flow," *IEEE Trans. on Power Apparatus and Systems,*" Vol. 101, pp. 3722- 3732, October, 1982.

[CaGr87] G. Carpenter and S. Groosberg, "A massively Parallel Architecture for a Self-organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics and Image Processing,* Vol. 37, pp. 54-115, 1987.

[Che84] C.-T. Chen, Linear System Theory and Design, Holt, Rinehart & Winston, Inc., New York, 1984.

[ChLi84] L. O. Chua and G-N. Lin, "Nonlinear Programming without Computation," *IEEE Trans. on Circuits and Systems,* Vol. 31, pp. 182-188, February, 1984.

[ChMS91] C. Chiu, C.-Y. Maa, and M. A. Shanblatt, "Energy Function Analysis of Dynamic Programming Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 2, No. 4, pp. 418-426, July, 1991.

[ChMS92] J. Chen, C-Y. Maa, and M. A. Shanblatt, "Improved Artificial Neural Networks for Linear and Nonlinear Programming," to appear, *the International Journal of Neural Systems*, Vol. 2, No. 4, 1992.

[ChSh92a] J. Chen and M. A. Shanblatt, "Solving Linear Systems Using an Artificial Neural Network," (In review, *IEEE Trans. on Neural Networks*, 1992.)

[ChSh92b] J. Chen and M. A. Shanblatt, "Artificial Neural Network Techniques for Power Load Flow and Contingency Analysis," (In review, *IEEE Trans. on Power Systems*, 1992.)

[ChYa88a] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory," *IEEE Trans. on Circuits and Systems*, Vol. 35, pp. 1257-1271, October, 1988.

[ChYa88b] L. O. Chua and L. Yang, "Cellular Neural Networks: Applications," *IEEE Trans. on Circuits and Systems*, Vol. 35, pp. 1273-1290, October, 1988.

[CoDK86] G. C. Contaxis, C. Delkis and G. Korres, "Decoupled Optimal Load Flow Using Linear or Quadratic Programming," *IEEE Trans. on Power Systems*, Vol. 1, pp. 1-7, May, 1986.

[Co*et*86] G. C. Contaxis, *et al.*, Decoupled Optimal Load Flow Using Linear or Quadratic Programming," *IEEE Trans. on Power System*, Vol. 1, pp1-7, May, 1986.

[CoRo87] P. Comon and Y. Robert, "A Systolic Array for Computing BA^{-1}," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. 35, No. 6, pp. 717-723, June, 1987.

[Cra56] S. H. Crandall, Engineering Analysis: A Survey of Numerical Procedures, McGraw-Hill, Inc., New York, 1956.

[Cro80] J. Cronin, Differential Equations, Marcel Dekker, Inc., New York, 1980.

[DaDe91] J. Dalton and A. Deshmane, "Artificial Neural Networks," *IEEE Potential*, Vol. 10, No. 2, April 1991.

[Deb88] A. S. Debs, Modern Power Systems Control and Operation, Kluwer Academic Publishers, Boston, 1988.

[Dec89] A. L. DeCegama, Parallel Processing Architectures and VLSI Hardware, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[ELMA91] M. A. El-Sharkawi and R. J. Marks II, (Edited), *IEEE Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, WA, 1991.

[EOMD91] M. A. El-Sharkawi, S. Oh, R. J. Marks II and M. J. Damborg, "Short Term Electric Load Forecasting Using an Adaptively Trained Layered Perceptron," *IEEE Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seattle, WA, 1991.

[Fiet89] R. Fischl, *et al.*, "Screening Power System Contingencies Using A Back-Propagation Trained Multiperceptron," *ISCAS*, pp. 486-489, 1989.

[FiFS91] W. A. Fisher, R. J. Fujimoto, and R. C. Smithson, "A Programmable, Analog Neural Network Processor," *IEEE Trans. on Neural Networks*, Vol. 2, No. 2, pp. 222-229, March, 1991.

[FoMo67] G. E. Forsythe and C. B. Moler, Computer Solution of Linear Algebraic Systems, Prentice-Hall, Englewood, NJ, 1967.

[FoTa88] Y. S. Foo and Y. Takefuji, "Integer Linear Programming Neural Networks for Job-Shop Scheduling," *IEEE International Conference on Neural Networks*, Vol. II, pp. 341-348, 1988.

[GaGr87] G. Garpenter and S. Grossberg, "A Massively Parallel Architecture for a Self-organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics and Image Processing*, Vol. 37, pp. 54-115, 1987.

[GoLo83] G. H. Golub, and C. F. Van Loan, Matrix Computations, John Hopkins University Press, Baltimore, 1983.

[GrKu89] S. Grossberg and M. Kuperstein, Neural Dynamics of Adaptive Sensory Motor Control, Pergamon Press, Inc., Elmsford, New York, 1989.

[Gro76] S. Grossberg, "Adaptive Pattern Classification and Universal Recording: Parallel Development and Coding of Neural Feature Detectors," *Biological Cybernetics*, Vol. 23, pp. 121-134, 1976.

[Guo90] S. Guo, Determining the Steady Solutions of Nonlinear Models of Power Systems, Ph.D. Dissertation, EE Dept., Michigan State University, East Lansing, 1990.

[Hec87] R. Hecht-Nielsen, "Counterpropagation Networks," *IEEE First International Conference on Neural Networks*, Vol. 2, pp. 19-32, 1987.

[Hec90] R. Hecht-Nielsen, Neurocomputing, Addison-Wesley, Reading, MA 1990.

[HiLi86] F. S. Hillier and G. J. Lieberman, Introduction to Operation Research, Holden-Day, Inc., Oakland, CA, 1986.

[HiSe86] G. Hinton and T. Sejnowski, "Learning and Relearning in Boltzmann Machines," Parallel Distributed Processing Explorations and the Microstructures of Cognition, Edited by Rumelhart and McClelland, pp. 282-317, MIT Press, 1986.

[HiSm74] M. W. Hirsch and S. Smale, Differential Equations, Dynamical Systems, and Linear Algebra, Academic Press, New York, 1974.

[Hop82] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Science*, Vol. 79, pp. 2554-2558, 1982.

[Hop84] J. J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties Like Two-State Neurons," *Proc. Natl. Acad. Sci.*, Vol. 81, pp. 3088-3092, 1984.

[HoTa85] J. J. Hopfield and D. W. Tank, "'Neural' Computational of Decisions in Optimization problem," *Biol. Cybern.*, Vol. 52, pp. 141-152, 1985.

[HuSh91] K. C. Hui and M. J. Short, "A neural Networks Approach to Voltage Security Monitoring and Control," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seatle, WA, July, 1991.

[HwCh80] K. Hwang and Y.-H. Cheng, "VLSI Computing Structures for Solving Large-Scale Linear System of Equations," *Proceedings of International Conference on Parallel Processing*, pp. 217-227, 1980.

[HwCh82] K. Hwang and Y-H Cheng, "Partioned Matrix Algorithms for VLSI Arithmetic Systems," *IEEE Trans. on Computer*, Vol. 31, pp. 1215-1224, December 1982.

[HwBr84] K. Hwang and F. A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill Inc., New York, 1984.

[Jac88] A. Jackson, "Machine Learning," *Expert Systems*, Vol. 5, pp. 132, Oxford, May 1988.

[JeJo90] C-W. Jen and S-J. Jou, "Design of One-Dimensional Systolic-Array Systems for Linear State Equations," *IEE Proceedings*, Vol. 137, pp. 185-192, June 1990.

[Jen77] Alan Jennings, Matrix Computation for Engineers and Scientists, John Wiley & Sons, New York, 1977.

[JoJe88] S-J. Jou and C-W. Jen, "Design of a Systolic Array System for Linear State Equations," *IEE Proceedings,* Vol. 135, pp. 211-218, October 1988.

[Joh87] S. L. Johnson, "Solving Tridiagonal Systems on Ensemble Architectures," *SIAM Journal on Scientific and Statistical Computing,* No. 8, pp. 354-392, 1987.

[JoRi82] L. W. Johnson and R. D. Riess, Numerical Analysis, 2nd ed., Addison-Wesley, Reading, Massachusetts, 1982.

[KaBr89] K. N. Karna and D. M. Breen, "An Artificial Neural Networks Tutorial: Part 1 — Basics," *Neural Networks,* Vol. 1, No. 1, pp. 4-23, 1989.

[KeCh87] M. P. Kennedy and L. O. Chua, "Unifying the Tank and Hopfield Linear Programming Circuit and the Cononical Nonlinear Programming Circuit of Chua and Lin," *IEEE Trans. on Circuits and Systems,* Vol. 34, pp. 210-214, February 1987.

[KeCh88] M. P. Kennedy and L. O. Chua, "Neural Networks for Linear and Non-linear Programming," *IEEE Trans. on Circuits and Systems,* Vol. 35, pp. 554-562, May 1988.

[Koh87] T. Kohonen, "State of the Art in Neural Computing," *IEEE First International Conference on Neural Networks,* Vol. 1, pp. 81, 1987.

[Kos87] B. Kosko, "Competitive Adaptive Bi-directional Associative Memories," *IEEE First International Conference on Neural Networks,* Vol. 2, pp. 759-766, 1987.

[Lip87] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, April 1987.

[LiZS91] T.-Y. Li, H. Zhang, and X.-H. Sun, "Parallel Homotopy Algorithm for the Symmetric Tridiagonal Eigenvalue Problem," *SIAM Journal on Scientific and Statistical Computing*, Vol. 12, No. 3, pp. 469-487, May 1991.

[MaHY89] A. Masski, Y. Hirai, and M. Yamada, "Neural Networks in CMOS: A case study," *IEEE Circuits and Devices Magazine,* Vol. 6, pp. 12, July 1990.

[MaSh89] C-Y. Maa and M. A. Shanblatt, "Adjustment of Parameters for Signal Decision Networks," *Proceedings of International Joint Conference on Neural Networks*, Washington D. C., Vol. II, pp. 589, 1989.

[MaSh90] C-Y. Maa and M. A. Shanblatt, "A Neural Net Technique for Economic Power Dispatching with Consideration of Transmission Losses," (*In Publication*) 1990.

[MaSh91] C-Y Maa and M. A. Shanblatt, "Linear and Quadratic Programming Neural Network Analysis," to appear , *IEEE Trans. on Neural Networks,* 1992.

[Mea88] C. A. Mead, Analog VLSI and Neural Systems, Addison-Wesley, 1988.

[MeIs89] C. Mead and M. Ismail, Analog VLSI Implementation of Neural Systems, Kluwer Academic Publishers, 1989.

[MiPa69] M. Minsky and S. Papert, Perceptrons, MIT Press, 1969.

[MoAG91] A. Moore, J. Allman, and R. M. Goodman, "A Real-Time Neural System for Color Constancy," *IEEE Trans. on Neural Networks*, Vol. 2, No. 2, pp. 237-247, March, 1991.

[Moet87] A. Monticelli, *et al.*, Security-Constrained Optimal Power Flow with Post-Contingency Corrective Rescheduling," *IEEE Trans. on Power Systems,* Vol. 2, pp. 175-182, February, 1987.

[Mol83] D. I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays," *Proceedings of the IEEE*, Vol. 71, No. 1, January 1983.

[MoPG87] A. Monticelli, M. V. Pereira and S. Granville, "Security-Constrained Optimal Power Flow with Post-Contingency Corrective Rescheduling," *IEEE Trans. on Power Systems,* Vol. 2, pp. 175-182, February, 1987.

[NaCh92] N. M. Nasrabadi and C. Y. Choo, "Hopfield Network for Stereo Vision Correspondence," *IEEE Trans. on Neural Networks,* Vol. 3, No. 1, pp. 5-13, January 1992.

[NeSY92] C. Neti, M. H. Schneider, and E. D. Young, "Maximally Fault Tolerant Neural Networks," *IEEE Trans. on Neural Networks,* Vol. 3, No. 1, pp. 14-23, January 1992.

[Ort87a] J. M. Ortega, Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, New York, 1987.

[Ort87b] J. M. Ortega, Matrix Theory, Plenum Press, New York, 1987.

[PaEM91] D. C. Park, M. A. El-Sharkawi, and R. J. Marks II, "An Adaptively Trained Neural Network," *IEEE Trans. on Neural Networks*, Vol. 2, No. 3, May, 1991.

[Par80] B. N. Parlett, The Symmetric Eigenvalue Problem, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[Pret86] W. H. Press, *et al.*, Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, New York, 1986.

[Rei87] D. K. Reinhard, Instruction to Integrated Circuit Engineering, Houghton Mifflin Company, Boston, 1987.

[Roet90] A. Rodriguez-Vazquez *et al.*, "Nonlinear Switched-Capacitor 'Neural' Networks for Optimization Problems," *IEEE Trans. on Circuits and Systems*, Vol. 37, pp. 384-398, March 1990.

[RoKa82] A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1982.

[Rot91] M. W. Roth, "Analogical Versus Logical or Connectionist Versus Symbolic or Scruffy Versus Neat," *IEEE Trans. on Neural Networks*, Vol. 2, No. 6, pp. 545-546, November, 1991.

[RuHW86] D. Rumelhart, G. Hinton and R. Williams, "Learning Internal Representations by Error Propagation," Parallel Distributed Processing: Foundations, Edited by Rumelhart and McClelland, pp. 318-362, MIT Press, 1986.

[Saet89] F. M. Salam, *et al.* "Parallel Processing for the Steady State Solutions of Large-Scale Non-Linear Models of Power Systems," *IEEE International Symposium on Circuits and Systems*, Portland, May, 1989.

[Sar91] P. Saratchandran, "Dynamic Programming Approach to Optimal Weight Section in Multilayer Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 2, No. 4, pp. 465-466, July, 1991.

[Sca66] J. B. Scarborough, Numerical Mathematical Analysis, The Johns Hopkins Press, Baltimore, 1966.

[She88] J. F. Shepanski, "Fast Learning in Artificial Neural Systems, Multilayer Perceptron Training Using Optimal Estimation," *Proceedings of the International Conference on Neural Networks,* I., pp. 465-475, IEEE Press, New York, July 1988.

[SoPa89] D. J. Sobajic and Y-H. Pao, "Artificial Neural-Net Based Dynamic Security Assessment for Electric Power System," *IEEE Trans. on Power Systems,* Vol. 4, pp. 220-228, February, 1989.

[SoSo88] B. Soucek and M. Soucek, Neural and Massively Parallel Computers: The Six Generation, John Wiley & Sons, New York, 1988.

[SrLC91] D. Srinivasan, A. C. Liew, and J. S. P. Chen, "Short Term Forecasting Using Neural Network Approach," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seatle, WA, July, 1991.

[Sti63] E. L. Stiefel, An Introduction to Numerical Mathematics, Academic Press, New York, 1963.

[Sto73] H. S. Stone, "An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations," *Journal of ACM*, Vol. 20, No. 1, pp. 27-38, 1973.

[StPi63] K. Steinbuch and V. A. W. Piske, "Learning Matrices and Their Applications," *IEEE Trans. on Electron. Comput., vol. 12, pp. 846-862, 1963.*

[SuCL91] G.-Z. Sun, H.-H. Chen, and Y.-C. Lee, "A fast On-Line Learning Algorithm for Recurrent Neural Networks," *Proceedings of the International Joint Conference on Neural Networks,* II, pp.13-18, 1991.

[TaHo86] D. W. Tank and J. J. Hopfield, "Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Trans. on Circuits and Systems*, Vol. 33, pp. 533-541, May 1986.

[TaTr91] A. Tabatabai and T. P. Troudet, "A Neural Net Based Architecture for the Segmentation of Mixed Gray-Level and Binary Pictures," *IEEE Trans. on*

Circuits and Systems, Vol. 38, No. 1, pp. 66-77, 1991.

[ToWi88] V. V. Tolat and B. Widrow, "An Adaptive 'Broom Balancer' with Visual Inputs," *Proceedings of the International Conference on Neural Networks,* II, pp641-647, IEEE Press, New York, July 1988.

[TrWa91] T. P. Troudet and S. M. Walters, "Neural Network Architecture for Crossbar Switch Control," *IEEE Trans. on Circuits and Systems*, Vol. 38, No. 1, pp. 42-56, January 1991.

[WeEl91] S. Weerasooriya and M. A. El-Sharkawi, "Use of Karhunen-Loe've Expansion in Training Neural Networks for Static Security Assessment," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, Seatle, WA, July, 1991.

[WiHo60] B. Widrow and M. Hoff, "Adaptive Switching Circuits," *1960 IRE WESCON Connection Record,* Part 4, pp. 96-104, New York, 1960.

[WiLe90] B. Widrow and M. A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," *Proceedings of the IEEE*, Vol. 78, No. 9, pp. 1415-1442, September, 1990.

[WiPa88] V. Wilson and G. S. Pawley, "On the Stability of the TSP Problem Algorithm of Hopfield and Tank," *Biol. Cybern.*, Vol. 58, pp. 63-70, 1988.

[WiRe71] J. H. Wilkinson and C. Reinsch, Linear Algebra, Vol. 2 Handbook for Automatic Computation, Springer-Verlag, New York, 1971.

[Wri91] S. J. Wright, "Parallel Algorithms for Banded Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 12, No. 4, July 1991.