MODELING THE DYNAMIC RESPONSE OF THE HUMAN SPINE TO MECHANICAL SHOCK AND VIBRATION USING AN ARTIFICIAL NEURAL NETWORK

by

Jordan James Nicol

B.A.Sc., Simon Fraser University, 1994

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF APPLIED SCIENCE

in the School of Engineering Science

ISBN 0-612-17031-4

Title of Thesis/Project/Extended Essay

"Modeling The Dynamic Response Of The Human Spine To Mechanical Shock And Vibration Using An Artificial Neural Network"

Author: ______________
(signature)

____
(name)

July 25, 1996
(date)

APPROVAL

Name:	Jordan James Nicol
Degree:	Master of Applied Science
Title of Thesis:	Modeling the dynamic response of the human spine to mechanical shock and vibration using an artificial neural network

Examining Committee:

CHAIR: Dr. Shahram Payandeh

____

Dr. Andrew Rawicz
Senior Supervisor
School of Engineering Science

[illegible]
____

Dr. James B. Morrison
Senior Supervisor
School of Kinesiology

[illegible]
____

Dr. Mehrdad Saif
School of Engineering Science

____

Dr. John Jones
Internal Examiner
School of Engineering Science

Date Approved: Aug 8, 1996

ABSTRACT

The ability to model the spinal response to shock and vibration is an important step in assessing the health hazard effects of repeated impacts to vehicle passengers. Current methods used for this purpose, such as the Dynamic Response Index and the British Standard 6841 filter, were found to perform poorly when the input consists of large-magnitude shocks typical of those experienced by occupants of tanks, trucks, and other off-road vehicles. In this thesis I present a novel approach to the problem of modeling the spinal response of the seated passenger to vertical accelerations applied at the seat. The modeling approach taken utilizes an artificial neural network (ANN) to predict the z-axis (vertical) acceleration at the fourth lumbar vertebra based on measured z-axis seat acceleration. An ANN is a universal approximator, capable of modeling any continuous function if trained with a sufficiently representative set of measured input-output data. The seat-spine system was modeled as a network with five inputs and one output. The Levenberg-Marquardt algorithm was used to train the network by adjusting the network parameters so as to minimize the square of the prediction error. The inputs to the network are delayed samples of the measured inputs and predicted outputs of the nonlinear simulation. It is shown that the trained network significantly outperforms three different linear models examined for predicting the z-axis acceleration at the L-4 vertebra.

Dedicated to my parents, Ken and Marguerite,

for their love, support, and encouragement.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. James Morrison for his supervision and guidance throughout this project. Recognition is also extended to Dr. Andrew Rawicz for his consultation and assistance. Finally, special thanks to BC Research Inc. and the BC Advanced Systems Institute for their financial support of this project.

TABLE OF CONTENTS

LIST OF FIGURES

List Of Tables

CHAPTER 1 INTRODUCTION

Modeling physiological systems is an important tool in understanding how the human body operates and is affected by its environment. The complexity and nonlinear nature of living systems often makes the development of an accurate model difficult when traditional linear modeling techniques are employed. In these circumstances, a nonlinear modeling approach can yield a more accurate model, leading to a better understanding of the real system.

A traditional modeling strategy is to develop the model structure from first principles and then to estimate the values of model coefficients from measured input and output data. This analytical approach suffers if the system or process is either not well understood or overly complex. In addition, the simplifying assumptions upon which the model is based may be incorrect under certain conditions. In many cases, linearity is assumed over a certain operating range. The simplified system can then be represented using a variety of well-developed linear modeling techniques. However, if the nonlinearity is strong or a general model is required, a nonlinear modeling approach is preferable.

One example of interaction between the human body and the environment is the case of a seated passenger in a moving vehicle. If the vehicle travels over rough terrain the passenger's body will be subjected to a variety of motion-related stresses. For simplicity, we can categorize these motion-related stresses as two distinct: phenomena: vibration and mechanical shocks. For most of us, vehicle vibration and shocks are low magnitude and infrequent, except for the occasional pothole. However, for passengers in rough-terrain vehicles (tanks, mining and logging trucks, for example), these stresses are severe and frequent enough to lead to discomfort and adverse health effects (Backman, 1983; Beevis and Forshaw, 1985; Konda *et al.* 1985). Epidemiological studies suggest that exposure to shock and vibration can lead to fatigue, gastro-intestinal/cardiovascular problems, and back disorders, such as vertebral disk degeneration (Guignard, 1972; Sturges, 1974; Hanson and Holm, 1991).

These problems may be dealt with through changes in the design of the vehicle to provide a smoother ride, or by limiting the exposure of the occupant. In the latter case, an international standard exists for exposure to constant vibration (ISO 2631, 1985). However, no appropriate standard exists for the type of high amplitude shocks experienced in off-road vehicles.

The lack of applicable standards is largely due to limitations of the models upon which the standards are based. For example, two second order linear models, the Dynamic Response Index model utilized in the Air Standardization Coordinating Committee (1982) and the British Standard 6841 filter (1987), do not perform well for large amplitude shocks (Cameron *et al.*, 1996; Payne, 1991). Therefore, there exists a need for a model which can adequately characterize the spinal response to such shocks.

In this thesis, I develop such a model for the spinal response based on experimental data obtained from a series of experiments designed to simulate the shocks experienced by occupants in military tactical ground vehicles. Since the spine is a complex musculo-skeletal structure, whose biomechanical properties are not fully known, a system identification approach is taken to the modeling problem. In system identification, the model is developed based only on measured input and output data. Moreover, since numerous pieces of evidence suggest that the spinal response is nonlinear, a nonlinear system identification strategy is taken utilizing an artificial neural network. An ANN is a universal approximator that can model any continuous function provided it is trained with a representative set of input-output data (Cybenko, 1989; Funahashi,1989).

The thesis is organized as follows. Chapter 2 provides a literature review of the human response to shock and vibration, existing models, and exposure standards. Chapter 3 and 4 describe artificial neural networks and system identification, respectively. In Chapter 5, the objective and methodology of the model development is explained. The modeling results are provided and discussed in Chapters 6 and 7, respectively. Where possible, technical details such as mathematical derivations are provided for completeness in the appendices.

CHAPTER 2 HUMAN RESPONSE TO MECHANICAL SHOCK AND VIBRATION

When traveling in a vehicle, the human body is subjected to vibration and intermittent mechanical shocks. The magnitude and frequency characteristics of these signals depend on both the vehicle design and the traveling surface. The passenger experiences motion (displacement, velocity, and acceleration) in six degrees of freedom: fore-aft, vertical, lateral, pitch, roll and yaw. Accelerations in the vertical direction typically have the greatest magnitude since shocks and vibration of the vehicle are transmitted upwards through the seat.

The purpose of this chapter is threefold. Firstly, the levels of shock and vibration reported for various types of vehicles is described. Secondly, the reported health effects of short and long term exposure to shock and vibration are discussed, thereby indicating the need for reducing exposure. Finally, some models and international standards for the response to shock and vibration are described.

2. 1 Mechanical Shock and Vibration in Vehicles

Vibration may be defined as oscillations which result in zero mean displacement, or rotation. A mechanical shock, on the other hand, may be defined as an input to the body (force, displacement, velocity, or acceleration) that results in a forced disturbance of the relative position of body parts (Village *et al.*, 1995) These two types of motions represent opposite ends of a spectrum of vehicle motions. That is, as the time between successive shocks decreases, the motion signal increasingly resembles vibration. In the relevant literature and international standards, both shock and vibration are usually measured and discussed in terms of acceleration.

Two quantitative measures of vehicle acceleration signatures are the root mean square (RMS) value and the crest factor. The RMS value is a measure of the energy contained in the signal and is defined as

$$a_{rms} = [\frac{1}{T}\int_0^T a^2(t)dt]^{1/2} \qquad (2.1)$$

where a(t) is acceleration and T is the duration of the signal.

The crest factor is a measure of the peakedness of the signal or, in other words, the degree of amplitude variation:

$$\text{crest factor} = \frac{|a_{\max} - a_{\min}|}{2a_{rms}} \qquad (2.2)$$

where $a_{\max}$ and $a_{\min}$ are the maximum and minimum values, respectively, observed over duration T.

Numerous investigators have sought to determine the vibration level for on-road vehicles (cars, trucks, buses), off-road vehicles (tractors, skidders, military tanks, construction equipment) air transport (helicopters, fixed-wing) and water transport (ships). A comprehensive review of these studies is provided in Village *et al.* (1995).

The reported ranges of vibration levels for cars are 0.2-1.0 m/s^2 in the vertical direction, and 0.02-0.45m/s^2 in the horizontal directions. A spectral analysis of these signals indicated that their energy was concentrated in the 6-12 Hz range (vertical) and 1-3Hz range (horizontal). For trucks, the reported accelerations were somewhat higher: 0.4-1.5m/s^2 (vertical) and 0.15-0.65m/s^2 (horizontal). The corresponding dominant frequencies were 6-12 Hz and 1-4 Hz, respectively. Crest factors were reported by Griffin (1984) to be 3.9 and 4.8 for cars and trucks, respectively.

In contrast, reported values of acceleration and crest factors for off-road vehicles were significantly higher. The highest values were reported for dozers, graders, underground mining trucks, tractors, and military tanks. Dupuis (1980) reported vertical acceleration values of 1.6-2.5m/s^2. and 4-10m/s^2 for tanks (1974). Griffin (1984) reports tractors having vibration levels of 0.67-2.12 m/s^2 (vertical) and 0.59-1.86m/s^2 (horizontal). Crest factors in the vertical direction ranged from 5 to 22 for tractors (Monsees et al, 1989), 5 to8.79 for skidders (Golsse and Hope, 1987), and up to 21 for tanks (Griffin, 1986). (To put these numbers into perspective, a crest factor of 21 on a vibration signal with an rms value of 2.5m/s^2 would indicate the presence of shocks of over 50m/s^2 in magnitude). The dominant frequencies for off-road vehicle vibration ranged from 1-6 Hz (vertical) and 1-4 Hz (horizontal) for tanks, and 1.6-10Hz (vertical) and 1.6-3Hz (horizontal) for mining trucks (Village *et al.*, 1995).

2. 2 Health Effects

Numerous studies have indicated that shock and vibration can result in both short (acute) and long term (chronic) health effects for the vehicle occupant. The majority of literature focuses on the effects of whole body vibration rather than repeated impacts. However, a significant amount of data has been gathered from both epidemiological and cadaver studies of pilot ejection, horizontal seated impacts due to vehicle collision, life boat free falls, and blast in ships. Whole body vibration has two types of physiological consequences:

i) Those due to the movement of organs or tissues;

ii) A general stress response.

Animal studies have indicated that prolonged exposure to high levels of vibration can result in hemorrhaggic and degenerative changes in organs and various other systems in the body. Such injuries include: injury of the viscera, lungs, myocardium (Guignard, 1972); gastro-intestinal bleeding (Sturges, 1974); and hemorrhage of the kidney and brain (Guignard, 1972).

In addition, whole body vibration result in a generalized stress response due to an over stimulation of the sympathetic nervous system. This response manifests itself as increases in heart rate, cardiac output, peripheral vasoconstriction, respiratory rate and oxygen uptake. Such stress-induced stimulation of the cardiovascular system may result in fatigue but there are no indications of more serious health effects (Village *et al.*, 1995).

2.2.1 Vertebral Effects

The spine is a complex structure consisting of a number of rigid elements (vertebrae) connected by flexible, visco-elastic elements (disks). The lateral and posterior views of the spine are shown in Figure 2.1. One such disk and the superior and inferior vertebrae constitute a spinal unit, as pictured in Figure 2.2.

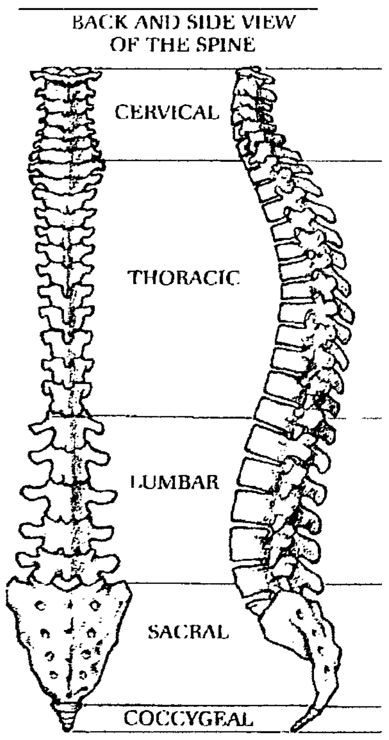

Figure 2.1 The posterior (left) and lateral(right) views of the human spine.

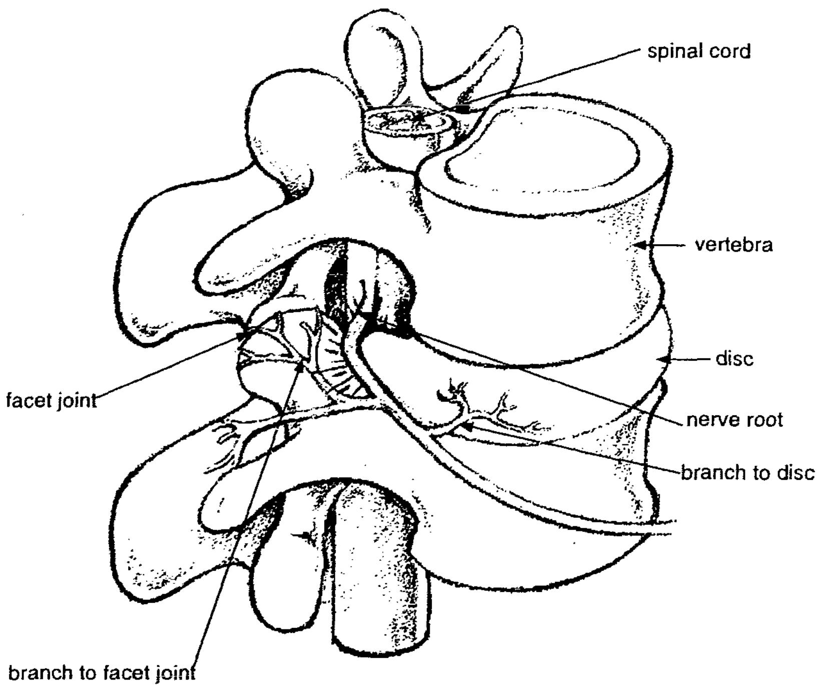

Figure 2.2 Spinal Unit

Exposure to whole body vibration and shock can result in back pain and back disorders. Back pain is a term for a general class of back ailments which are diagnosed on a subjective basis, whereas (clinical) back disorders are diagnosed through more objective measures (radiological methods, for example). There is a high reported incidence of back pain among heavy equipment operators, tractor drivers, truck, bus and car, heavy equipment operators (e.g. excavators), and pilots. Paulson (1949) reported that of 23 tractor drivers, 43.5% complained of back pain. Moreover, the study indicated that there was correlation between the reported severity of pain and the roughness of the ground. Among large population of container tractor drivers (540) and truck drivers (633), approximately 40% experienced back pain (Konda *et al.*, 1985; Backman, 1983). Beevis and Forshaw (1985) reported an 89% incidence of back pain among trainees for M113 Armoured Personnel Carriers. Gruber (1976) reported that truck drivers have higher rate of premature deformation of the spinal column, back pain and sprains than air traffic controllers and bus drivers.

For incidence of back disorders, Kristen (1981) reported an 81% rate among truck drivers. Similar rates for truck drivers were reported by Schmidt(1969): 79.5% versus 61.1% for the control group. Rehm and Wieth (1984) reported rates of 65% for retired truck drivers, 77% for truck/car drivers, and 80.3% for heavy equipment operators, compared with 62% in the control group.

Damage to the spine in response to high amplitude impacts typically consists of end plate fractures, whereas long-term exposure to vibration and/or repeated impacts is usually associated with degenerative damage. This pattern of failure is analogous to engineering materials which can fracture due to a single loading beyond their elastic limit and suffer material fatigue due to repetitive loading. It, therefore, follows that understanding the bio-mechanical properties of the spine will lead to an understanding of its failure mechanisms.

A number of researchers have investigated the mechanical properties of the spinal unit *in vitro* (Crocher and Higgins,1967; Henzel, Mohr, and van Gierke, 1968; Markolf, 1970; and *in vivo* (Nachemson and Morris, 1964; Christ and Dupuis, 1966; Pope *et al.*, 1991). Two mechanisms for chronic degeneration of tissues due to long-term exposure of vibration have been proposed: mechanical fatigue and impairment of nutrition.

Several studies (Henzel *et al.* 1968, Rolander and Blair, 1975; Brinckmann, 1988) have indicated that when the spinal unit is compressed, the disk, being virtually incompressible, bulges only slightly along its radial axes. Thus, any forced displacement of the two vertebrae is due not to compression of the disk but rather an inward deformation of the vertebral end plates. Such findings lead Brinkman (1988) to suggest that disk herniation is a result of repetitive loading of the disk , resulting in a fatigue failure of the disk rather than a single mechanical overload. This argument is supported by clinical symptoms of disk herniation which include detached pieces of annular material and sometimes fragments of cartilagenous end plate.

Hansom and Holm (1991) speculate that tissue damage is partially a result of impaired nutrition to the disk and end plate structures. The authors state that vibration may lead to a disruption of blood flow in vessels surrounding the annulus fibrosis and under the endplate, thereby reducing the diffusion of nutrients to tissues.

Sandover's (1983) hypothesis links the nutrition and mechanical failure mechanisms. He suggests that compressive loading leads to fatigue-induced micro fractures of the end plate or the subchondral trabecular bone. The repair process leaves deposits of callous which lead to reduced nutrient diffusion to the end plate and the disk, thereby resulting in eventual tissue degeneration.

2.3 Existing Models

A number of models have been developed which simulate the response of the human body to vibration and shocks. These models can generally be classified as being either biodynamic models or physiological models. Biodynamic models attempt to reproduce the dynamical characteristics of the body, but do not represent real anatomy or neuro-muscular effects. Instead, these models usually contain interconnected springs, masses, and damping elements. Biodynamic models range from lumped parameter models having a single degree of freedom and linear characteristics (Payne, 1991) to discrete parameter models having multiple degrees of freedom[1] (Belytschko and Privitzer, 1978; Amirouche, 1987) and models having nonlinear characteristics (Payne and Band, 1971; Hopkins, 1972). The model parameters were adjusted using measured input and output data to provide the best possible prediction. This undermines the validity of the model (unless it is verified using an independent data set) as it is no longer independent of the experimental data with which it was compared (Amirouche, 1987). It has been shown

[1] A few words about terminology for clarification: The usage of "lumped parameter", "discrete-parameter", and "degrees of freedom" in the literature differs somewhat from conventional engineering usage. In this context, "lumped" means the body is treated as several large masses, representing the head, thorax, abdomen, etc., whereas in "discrete" models, the body is represented by individual elements, such as masses for each vertebra. In the systems theory, both of these models would be considered to be of the lumped parameter type. In reality, most real systems are of the distributed parameter type but to simplify analysis, we lump the system parameters into discrete elements.

Furthermore, "the degree of freedom" refers to the system rather than the number of variable model parameters. For example, a single degree of freedom model of a spinal unit would predict motion (usually acceleration) in one of the six independent directions of movement available in three dimensions.

that the biodynamic response can be predicted reliably within certain ranges of motion using a linear lumped parameter model (Fairley and Griffin ,1989). However, Muksian and Nash (1974) demonstrated that a nonlinear model was required to accurately simulate biodynamic response over a wide range of frequencies and amplitudes.

A separate modelling approach incorporates anatomical structures and their properties as determined experimentally (for example the stiffness and damping of intervertebral discs and ligaments, muscle recruitment patterns and force-velocity characteristics). Passive models of this type representing the geometry and material properties of the vertebral column have been shown to be informative in predicting internal stresses and compression failures (Orne and Lui, 1971; Prasad and King, 1974). Active models incorporating muscle characteristics have been developed for predicting vertebral compression forces in activities such as lifting and carrying (Marras and Sommerich, 1991; McGill, 1992). These models require a knowledge of body segment kinematics as input data, and for this reason are sometimes referred to as inverse dynamic models. However, these models have not been validated for whole body vibrations and shock environments and, unlike lumped or discrete parameter models, cannot predict body segment accelerations or vertebral compressive forces from a knowledge of vehicle motion (e.g. seat acceleration data) alone.

2.4 Standards of Exposure

A variety of guidelines exist for the evaluation of human exposure to whole body vibration and shock: International Standards Organization (ISO) 2631 (1985), British Standard 6841 (1987), Air Standardization Coordinating Committee (ASCC) (1982). Of these methods, the most widely used is the ISO 2631, which provides a method for calculating the exposure limit for health effects of vibration. The method of calculation is based on the vector sum of the frequency weighted accelerations at the seat in all three biodynamic axes. The use of root mean square (RMS) acceleration values in the standard tends to smooth the effect of single, high amplitude events such as impacts or mechanical shocks. For this reason the standard excludes acceleration signals containing crest factors greater than 6. Although crest factors up to 12 are proposed in a draft revision of the standard, it is widely recognized that the rms method is inadequate for evaluation of the health effects of non-stationary signals.

A more sensitive method of evaluating non-stationary signals containing vibration and shocks is described by Griffin (1986) and is included in Appendix A of the British Standard 6841. This method calculates a vibration dose value (VDV) based on the fourth power of the weighted acceleration signal at the seat,

$$VDV = \left(\int_0^t a_w^4(t)dt \right)^{\frac{1}{4}} \quad \mathrm{m{\cdot}s^{-1.75}} \qquad (2.3)$$

where a_w(t) represents the BS 6841 filter output due to acceleration at the seat.

Although the BS 6841 does not define limits for health effects, it is estimated that a VDV of 15 causes severe discomfort, and it is also assumed that increased exposure will be accompanied by increased risk of injury. A revision of this standard utilizes a biodynamic model developed by Fairley and Griffin (1989) to weight the accelaration prior to application of the VDV equation. This model, based on the data of 60 subjects exposed to low amplitude vibrations (1.0 $\mathrm{m{\cdot}s^{-2}}$), has a natural frequency at 5 Hz and a critical damping ratio of 0.48.

The DRI developed by Payne (1965) and utilized in the ASCC advisory publication is designed specifically for the analysis of the spinal injury risk of large amplitude accelerations (10 to 200 $\mathrm{m/s^2}$) in the vertical direction (+z axis). Unlike the frequency weighting approach of the ISO 2631 and BS 6841, the DRI is based on a biodynamic model having a single degree of freedom. It is assumed that the output of the model, represented by the peak force in the spring component, is proportional to the stress developed in the human body. The model, shown in Figure 2.3, consists of a second order linear system which can be defined by its natural frequency (f_n= 8.4 Hz) and critical damping ratio ($\bar{c}$ = 0.224). In a draft revision of the DRI, Payne (1991) recommended f_n that should be increased to 11.9 Hz with a $\bar{c}$ of 0.35.

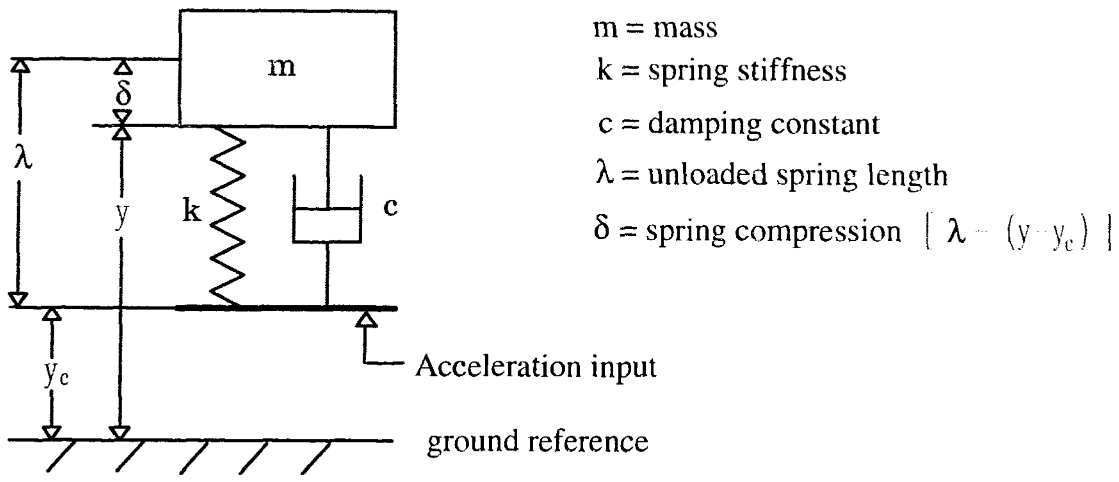

Figure 2.3 Biodynamic model used in Dynamic Response Index.

The DRI model acts as a low pass filter, attenuating the higher frequency acceleration components at the seat, and magnifying acceleration waveforms close to the resonant frequency of the model. By focusing on the peak output of the system it avoids the RMS "averaging" process contained in the ISO 2631 and more accurately reflects the effect of impacts. Although originally designed to evaluate single impacts, the DRI has been extended to evaluate the injury risk from repeated shocks (Allen, 1976) using a peak stress summation method based on the fatigue failure characteristics of biological materials (Sandover, 1985). In this model the critical DRI for j occurrences of a given acceleration magnitude is expressed as

$$\Delta DRI_j = \Delta DRI_0 \cdot j^{-\frac{1}{8}} \qquad (2.4)$$

where DRI_j is the DRI value of the acceleration waveform input at the seat,

DRI_0 is the DRI value required to cause injury from a single impact, and

$\Delta DRI = DRI - 1$ is an offset factor for gravitational loading (Payne, 1991).

The ability of the DRI and VDV models to evaluate health effects depends on the degree of accuracy with which their filter outputs simulate the human response to vibration and repeated shocks. The response characteristics of the two models upon which these standards are based demonstrate distinct differences as shown in Figure 2.4. In addition, both models describe linear systems, whereas the human response is generally accepted to be nonlinear in nature.

The different natural frequencies of the Fairley-Griffin and DRI models representing low amplitude and high amplitude acceleration, respectively, would appear to confirm the nonlinear characteristic of the human body. Furthermore, *in vitro* compression testing of lumbar1-lumbar2 spinal units have indicated a non-linear force-displacement curve(Crocher and Higgins,1967). Thus, it is unlikely that a simple linear model is capable of representing a wide range of vibration and shock amplitudes.

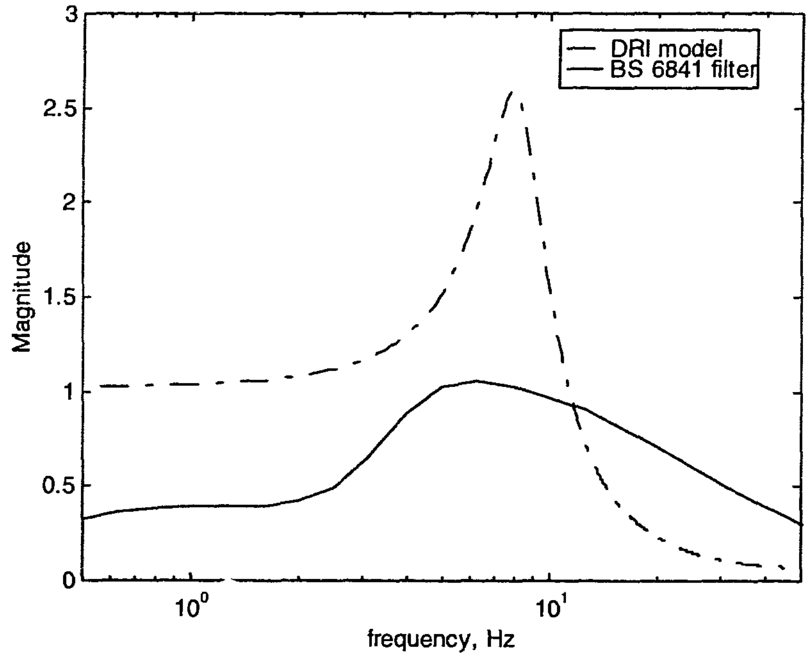

Figure 2.4 Magnitude frequency response of the DRI model (dashed line) and the BS 6841 filter (solid line).

CHAPTER 3 ARTIFICIAL NEURAL NETWORKS

Artificial neural networks are a class of computational structures which, in some respects, emulate biological neural networks. Some shared characteristics include high connectivity, massive parallelism, and adaptation to stimulus. Artificial neural networks have applications in engineering (modeling, control systems, signal processing, speech recognition), as well as cognitive science and neurophysiology (as models of high and medium level brain function, respectively). They have been used by financial analysts to forecast commodity prices, meteorologists to predict weather, and biologists to interpret nucleotide sequences.

A simplified model of a typical biological neuron is shown in Figure 3.1. The neuron consists of three anatomical regions: dendrites, soma, and axon. The dendrites are the receiving terminals for the majority of incoming neural signals. The soma, or cell body, controls the metabolism/reproduction of the neuron, and the axon transmits outgoing signals to other neurons or effector cells such as muscle.

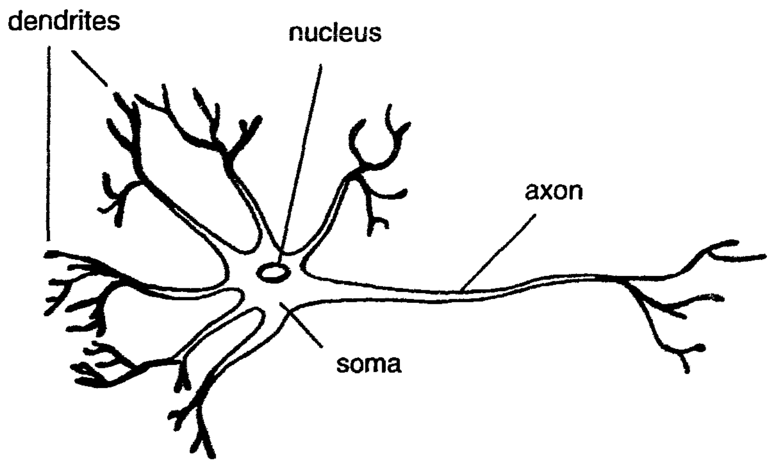

Figure 3.1 Biological Neuron.

Neural signals propagate between neurons at connections called synapses. When a neural signal reaches an end terminal of an axon, chemical messengers called neurotransmitters are released from the axon and bind to the membrane of a post-synaptic neuron. Each bound neurotransmitter results in a small change in the receiving neuron's membrane potential. If, and only if, the net sum of all synaptic inputs causes the neuron's membrane potential to exceed a threshold value, an action potential is produced, and the neural signal propagates to other neurons.

A number of factors govern the probability of action potential generation, including the magnitude of excitatory and inhibitory post-synaptic potentials and the effectiveness (or strength) of the synapse. It is believed that long-term modification of synaptic strength is the basic mechanism through which a biological neural network learns.

An artificial neural network (ANN) consists of elementary processing units analogous to biological neurons in function but far less complex. A network of interconnected processing elements (PEs) may be implemented as a computer program or as an electronic circuit. Connections between PEs are unidirectional communication channels with a scalar gain factor called the connection weight. Inputs to each PE are amplified or attenuated by the weight of each connection and then summed. The sum of these weighted inputs is then passed through an activation function, resulting in an output which is distributed to other PEs. Whereas biological neurons fire an action potential only if the the activation threshold is reached, artificial neurons produce an output for all ranges of input signals. The input-output characteristics of the PE depend on the particular activation function chosen.

Artificial neural networks adapt through modifications of the connection weights according to a pre-defined adaptation algorithm, or training rule. The training rule and arrangement of PEs (network architecture) is what distinguishes different types of artificial neural networks.

ANN architecture may be classified as either static or dynamic. A static network performs a nonlinear transformation of the form $y = G(x)$ where$x \in \Re^n$ and $y \in \Re^m$. Static networks are, therefore, memoryless systems since the current output is a function of only the current input. These networks are usually referred to as feedforward networks (FFNN) because information flows uni-directionally, from input to output PEs without any cycles.

Dynamic networks contain feedback connections and, thus, their output is a function of both the current input and the current network state. Because the output must be calculated recursively, these networks are usually referred to as recurrent neural networks (RNN). In signal processing terminology, recurrent neural networks are equivalent to nonlinear infinite impulse response (IIR) filters.

The operation of an ANN can normally be divided into a training phase and a recall phase. During training, input stimulus are presented to the network and synaptic weights are modified according to the training rule. In the recall phase, the adaptation mechanism is normally deactivated and the network simply responds to further stimulii as it has been trained to do. In some cases, as when employed as adaptive equalizers, these two phases can occur at the same time.

Training rules may be categorized as being supervised or unsupervised. In supervised learning, the network is presented with example input/output data and trained to implement a mapping that matches example data as close as possible. A "supervisor" detects incorrect network responses and adjusts the weights accordingly. The supervisor typically takes the form of a performance criterion, such as the sum of squared errors between the desired response and the network output. Some examples of supervised learning algorithms include Backpropagation (Werbos, 1974; Rummelhart and McLelland, 1986), Cascade Correlation (Hecht-Nielsen, 1987), Learning Vector Quantization (Kohonen *et al.*, 1988), Real-Time Recurrent Learning (Zipser and Williams, 1989), and Extended Kalman Filtering (Singhal and Wu, 1989)

From the point of view of pattern recognition, supervised learning utilizes pattern class membership information (Kosko, 1992). If the network performs a nonlinear mapping of the form $y = G(x)$ where$x \in \Re^n$ and $y \in \Re^m$, then the training process partitions the input space $\Re^n$ into k pattern classes of dimension M. After training, the network output indicates the degree to which the input pattern belongs to a particular pattern class. Since the membership classes are known *a priori*, the supervisor can detect incorrect pattern classifications during training and correct them through weight adjustments.

In unsupervised learning, the pattern membership classes are not known beforehand. Rather the network is presented with unlabelled patterns and evolves its own membership classes, or pattern clusters. Patterns are clustered on the basis of similarity defined by some metric, such as the Nearest Neighbor Rule (Simpson, 1993). Due to this behavior, such neural networks are described as self-organizing. Some examples include Kohonen Self-Organizing Map (Kohonen, 1984), Adaptive Resonance Theory (Carpenter and Grossberg, 1987), Discrete Hopfield Networks (Hopfield, 1982), and Temporal Associative Memory (Kosko , 1988)

3.1 The Multi-Layer Perceptron

Probably the most commonly used ANN is the multilayer perceptron (MLP), also referred to as a multilayer feedforward network . These networks are modification of the single layer, linear Perceptron that was first developed by Rosenblatt in 1958. The structure of a MLP consists of a number of PEs, arranged in layers as depicted in Figure 3.2. Information flows from the input layer to the output layer without any feedback.

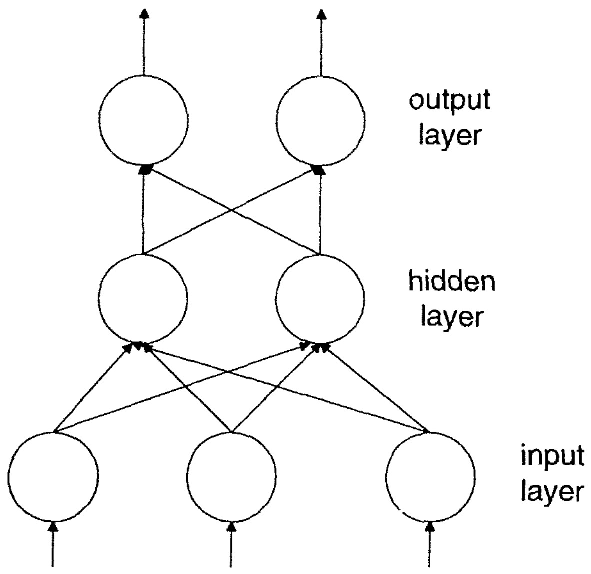

Figure 3.2 Multilayer Perceptron

PEs in the input layer are typically data buffers which distribute the input to the rest of the network. PEs in the hidden layer typically consist of a summer and an activation function. Each input to the summer is multiplied by the connection weight. The activation function introduces a non linearity which makes these networks powerful modeling tools. Without it, the network can at most perform a linear mapping. Output PEs may have nonlinear or linear activation functions, depending on the application.

Consider a MLP consisting of M layers. The i'th PE in the l'th layer (Figure 3.3) produces an output that is a nonlinear function of weighted inputs from PEs in the previous layer. The inputs are first summed and then passed through an activation function F(•). The input to the activation function is given by

$$v_i^l(t) = \sum_{j=J}^{n_{l-l}} w_{ij}^l x_j^{l-l}(t) + b_i^l \qquad (3.1)$$

where n_{l-1} is the number of PEs in the (l-1)'th (or previous) layer, w_{ij}^l is the weight connecting the j'th PE in layer l-1 to the i'th PE in layer l, $x_j^{l-1}(t)$ is the output of the j'th PE in the (l-1)'th layer, and b_i^l is the threshold parameter, or bias, associated with the l'th layer.

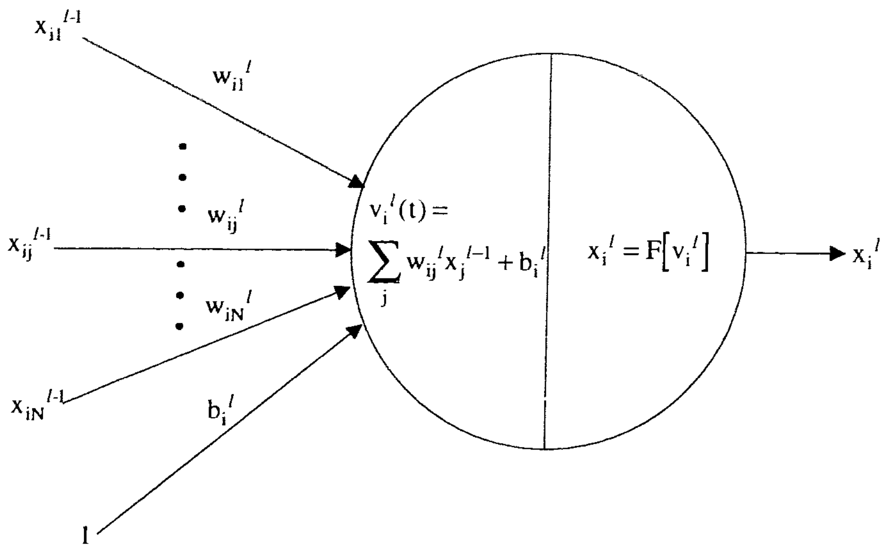

Figure 3.3 Basic Processing Element.

The PE output is then given by

$$x_i^l(t) = F[v_i^l(t)]. \tag{3.2}$$

Using the above equations the entire network (Figure 3.4) can be described by

$$x_i^l(t) = F[\sum_{j=1}^{n_{l-1}} w_{ij}^l x_j^{l-1}(t) + b_i^l] \quad \text{for } l = 1,\dots,M \tag{3.3}$$

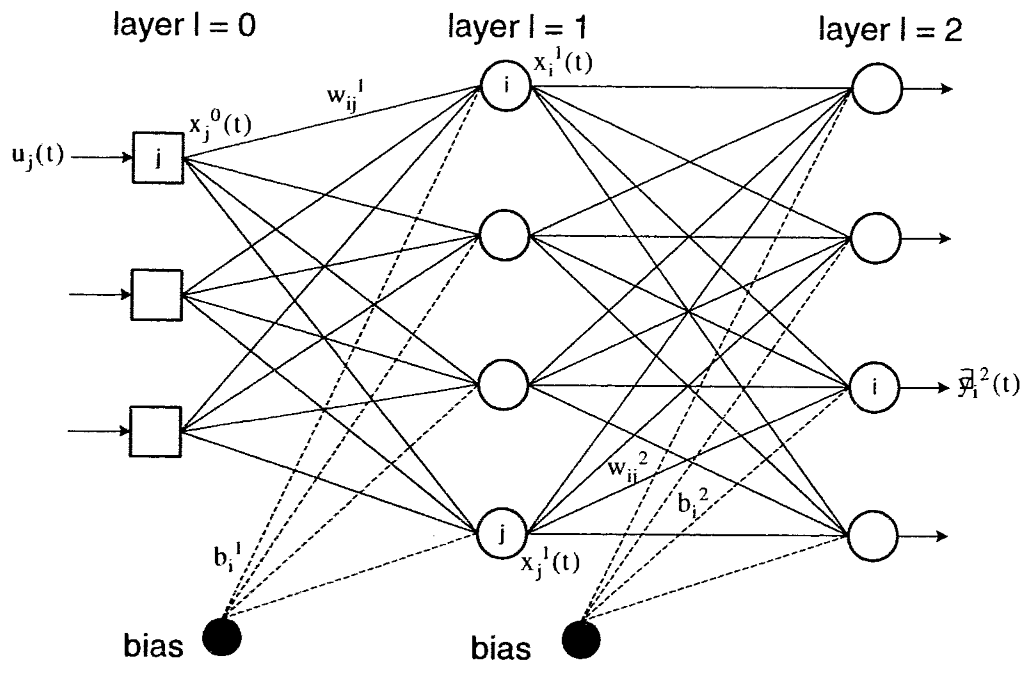

Figure 3.4 M layer Perceptron

The number of inputs and outputs then becomes, respectively, n_0 and n_M. To differentiate the network input from the input to any PE, we can redefine the input to the j'th PE of the input layer (l=0) as

$$u_j(t) = x_j^0(t) \tag{3.4}$$

Similarly, to differentiate the network output from the output of any PE, we can redefine the ouput of the i'th PE of the output layer (l=M) as

$$\hat{y}_i(t) = x_i^M(t) \tag{3.5}$$

where $\hat{y}_i(t)$ is the network approximation of the target variable $y_i(t)$.
The most common choice of activation function are either the sigmoid or the hyperbolic tangent but, in theory, any differentiable function can be used. The sigmoidal activation function is described by

$$F(v) = \frac{1}{1+e^{-v}} \tag{3.6}$$

whereas *v* is the sum of weighted inputs to the PE. Similarly, the hyperbolic tangent function is defined by

$$F(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}. \tag{3.7}$$

An advantage of these two functions is that their respective first derivatives can be expressed as a simple function of itself. For example, the first derivative of the sigmoidal function is given by

$$\frac{dF}{dv} = F(v)[1 - F(v)]. \tag{3.8}$$

3.2 Training

During training, the network is presented with example inputs and outputs. The weights and bias values are then adjusted to minimize some cost function, typically the sum of squared errors between the network output and the actual output. To elaborate on this concept first let us define a three layer network (M=2) with a single PE in the output layer. The total number of network parameters (weights and biases) is then $N_\Theta = n_1(n_0 + 2) + 1$. Now define a network parameter vector, Θ, that contains both the weight and bias values, such that

$$\Theta = \begin{bmatrix} \theta_1 & \dots & \theta_{N_\Theta} \end{bmatrix}^T. \qquad (3.9)$$

Furthermore, let

$$\varepsilon(t) = y(t) - \hat{y}(t). \qquad (3.10)$$

The cost function used in training the network is then

$$J(\Theta) = \frac{1}{2N} \sum_{t=1}^{N} [\varepsilon(t,\Theta)]^2 \qquad (3.11)$$

The objective of training is to find the value of Θ for which $J(\Theta)$ is minimized. The optimal vector is found iteratively using the generalized update equation

$$\Theta^{(k)} = \Theta^{(k-1)} + \alpha s(\Theta^{(k-1)}) \qquad (3.12)$$

where s is the search direction on the error surface at iteration step k, and α is a small positive constant called the learning rate which determines the adaptation step size.

3.3 Steepest Descent Algorithms

The simplest search direction is called steepest descent, in which the parameter change is along the negative gradient of the error surface J. That is, we choose

$$s(\Theta) = -\nabla J(\Theta) \tag{3.13}$$

where

$$\nabla J(\Theta) = \frac{\partial J}{\partial \Theta}. \tag{3.14}$$

Define the gradient of the network output with respect to Θ as

$$\Psi(t,\Theta) = [\frac{d\hat{y}(t,\Theta)}{d\Theta}]^T. \tag{3.15}$$

Then it can be shown that Equation 3.13 can be expressed as

$$s(\Theta) = -\nabla J(\Theta) = \frac{1}{N}\sum_{t=1}^{N}\Psi(t,\Theta)\varepsilon(t,\Theta) \tag{3.16}$$

The update equation then becomes

$$\Theta^{(k)} = \Theta^{(k-1)} - \alpha[\nabla J(\Theta)]^{(k-1)} \tag{3.17}$$

The most popular form of steepest descent for neural networks is the backpropagation algorithm first proposed by Werbos (1974) and modified by Rummelhart and McLelland (1986). Backprogation takes its name from how the weight changes are ordered. Updates are made from the output layer to the input layer, using calculation of the error dependent on the earlier steps, so that the error tends to propagate backwards through the network. The main disadvantage of backpropagation and steepest descent in general is that the algorithm may converge to a local minimum of the error function. In addition, convergence tends to be slow due to the typical shape of the error surface. Sigmoidal activation functions tend to result in error surfaces which alternate between very flat regions, in which learning is painstakingly slow, and steep regions which can causes the algorithm to become unstable (Hush and Horne, 1993).

3.4 Gauss Newton Methods

A more efficient search direction is used in Gauss-Newton-based methods such as Levenberg-Marquardt (Marquardt, 1963), Extended Kalman Filter, and Recursive Prediction Error (Chen, Billings, and Grant, 1989). These algorithms use second order derivative information of the error surface which results in an increased rate of convergence. Unfortunately they also require significantly more computation, and unlike backpropagation do not take advantage of the parallel structure of the network.

Gauss-Newton methods modify the steepest descent search direction through multiplication of the negative gradient by a special matrix which contains information about the error surface shape. This matrix is the inverse of the approximate Hessian of the cost function. The approximation of the Hessian is

$$H(\Theta) = \frac{1}{N}\sum_{t=1}^{N} \Psi(t,\Theta)\Psi^T(t,\Theta) + \lambda I \quad \text{for } \lambda > 0 \tag{3.18}$$

(The addition of the second term ensures that $\mathbf{H}(\Theta)$ is nonsingular.) The modifued search direction then becomes

$$\mathrm{s}(\Theta) = -[\mathbf{H}(\Theta)^{-1}\nabla J(\Theta)] \tag{3.19}$$

3.5 Generalization

Generalization is the ability of the trained network to perform well on unseen data sets. Cross validation is the most common test for generalization. Prior to training, a subset of the available data called the testing set, is set aside for validation purposes. This set is assumed to be contained within the input space bounded by the training data. In other words, it should represent data that is similar, though not identical, to that used for training, where the definition of similar is application specific. The trained network's performance on the testing set indicates the degree to which the model represents the underlying system as opposed to merely being a good fit to the training data.

Network generalization is affected by the number of data samples in the training set (and how well they represent the problem), the complexity of the underlying system, and the complexity of the neural network model (Hush and Horne, 1993). In general, as the amount of training data increases, so too will the ability of the network to generalize.

The complexity of the neural network is measured by the number of hidden layer PEs and the number of hidden layers. These parameters determine the complexity of the nonlinear function that can be approximated. If too few PEs are present, the representational capacity of the network will be limited. On the other hand, if there are too many, the network will be prone to overfitting the training data. Optimal performance will be obtained when the complexity of the neural network matches that of the system or function to be modeled.

The number of hidden layer PEs can be optimized using cross validation. A plot of the performance function over the testing set will often show a minimum value when the structure of the neural network corresponds to the true system structure (Billings, Jamaluddin and Chen, 1991). A similar strategy is to compare the performance of the network on the training set and the test set. The optimal number is obtained when the prediction error is approximately the same on both sets. Some strategies, such as the Cascade Correlation method, add PEs during training as needed.

The number of hidden layers is similarly related to network's representational ability. Increasing the number of hidden layers results in an increase in the dimensionality of the problem space, making it easier for the network to approximate the nonlinear function. It has been shown that a single hidden layer is sufficient to approximate any continuous nonlinear function (Funahashi, 1989; Cybenko, 1989). Therefore, this same increase can be achieved by adding PEs to a single hidden layer but the number required may be astronomical. An analogous situation occurs in digital circuit design in which any binary function may be implemented as a sum of products (or product of sums) but may require a large number of logic gates. The total number of logic gates may be drastically reduced by instead using several layers of logic.

Various other strategies exist to increase network generalization: early stopping, pruning and complexity regularization. Early stopping (Cohn, 1993) involves limiting the number of training iterations. During the learning process, the performance of the network on the training set will continue to improve. However, at some point the performance on the testing set will decrease. Overtraining causes the network to learn the particular noise realization of the training set, as opposed to learning the input-output behavior of the underyling system.

Network pruning is based on the view of training as a process of encoding system information in the network weights. It assumes that a trained network contains redundant information which may be removed, thereby decreasing the network complexity. Pruning strategies usually start with a large network and systematically delete weights and PEs. One approach is to simply delete connections with very small weight values. It has been shown that a more effective strategy called Optimal Brain Damage (Cun, Denker, and Solla, 1989) is to delete the weights with the smallest *saliency*, i.e. whose removal disturb the solution the least.

Complexity regularization (Larsen and Hanson, 1994) increase generalization by penalizing network solutions that are overly complex. The penalty is imposed by adding a complexity term, such as the sum of squared weights to the cost function. This penalty term causes the weights to converge to smaller absolute values than they otherwise would. A common method of regularization is through weight decay in which the cost function takes the form

$$J(\Theta) = \sum_{p=1}^{N} J(p,\Theta) + \frac{\lambda}{2}\sum_{i}\theta_i^2 \qquad (3.20)$$

In this equation, N represents the number of training patterns, θ_i is the i'th weight contained in the vector Θ, and λ is the weight decay parameter which controls the influence of the second term relative to the squared error term. Weight decay results in a smaller average weight size which tends to discourage overfitting.

CHAPTER 4 RECURRENT NEURAL NETWORKS FOR SYSTEM IDENTIFICATION

Two fundamental approaches exist for developing a mathematical model of a system. In the analytical approach, physical laws such as the conservation of energy or mass are used to develop differential or difference equations. Analytical modeling may be used only when adequate knowledge about the system is available and when the system is not overly complex. When these conditions do not hold, an alternative strategy is identify the system from input and output observations.

System identification is one of the main concerns in the discipline of mathematical systems theory. Over the past five decades, considerable mathematical tools have been developed for analyzing and designing systems. Most of these tools are based on linear algebra, complex variable theory, and the theory of ordinary, linear differential equations (Narendra and Parthasarathy, 1990). As a result, there exist well-developed techniques for the analysis of linear systems. A similar set of tools does not exist for nonlinear systems and, consequently, modeling such systems is considerably more difficult.

The purpose of this chapter is to introduce some fundamental concepts of system identification, and to discuss recurrent neural networks in the context of nonlinear systems identification.

4.1 System Identification: Preliminary Concepts

What is a system? Sinha and Kuszta (1983) define a system as "a collection of objects arranged in ordered form, which, in some sense, are purpose or goal directed." More specifically, a system transforms a set of inputs, or causes, into a set of outputs, or effects. The form of the inputs, outputs, and the type of transformation distinguishes one system from another.

Systems may be categorized as being either static or dynamical. A static system is memoryless in that its output is a function of the current input only. In contrast, a dynamical system generates an output that is a function of the input as well as the current state of the system. Since the current state is determined by past states and previous inputs, a dynamical system has memory. In addition, systems may be characterized on the basis of stability, causality, linearity, lumpedness, and time-invariance. For additional discussion of these topics, the reader is referred to any introductory textbook on systems theory such as Chen (1989) or Oppenheim and Schaffer (1989).

The model of a system may be mathematically defined by an operator, P, which maps the input space U into the output space Y. P may be realized in a variety of mathematical forms, such as differential equations, difference equations, or as a transfer function. For static systems, U and Y are subsets of $\Re^n$ and $\Re^m$, respectively. For dynamical systems, U and Y are subsets of bounded integrable functions defined on the intervals [0, T] or [0,∞) (Narendra and Parathasarathy, 1990). The goal of system identification, then, is to determine an operator $\hat{P}$ which approximates P according to some criterion.

Figure 4.1 depicts a single-input, single-output (SISO) system to be modeled, where u(t) is the input signal, w(t) is additive system noise, r(t) is the unobservable system output, n(t) is measurement noise, and y(t) is the observable system output. Given a set of measured input data, u(t), and output data, y(t), the objective is to determine $\hat{P}$ such that

$$\|\hat{y}-y\| = \|\hat{P}(u)-P(u)\| \le \varepsilon \qquad u \in U \tag{4.1}$$

for some $\varepsilon > 0$ and some defined norm, denoted by $\|\bullet\|$, such as the root mean squared (RMS) error.

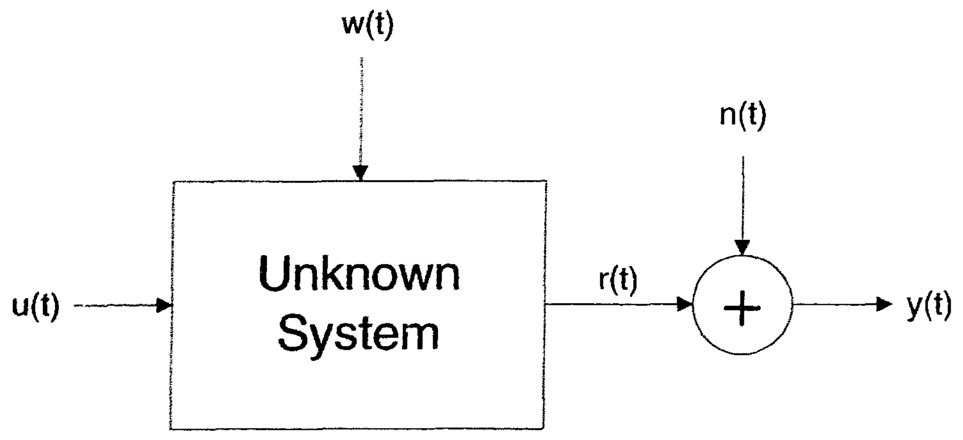

Figure 4.1 System identification approach.

In other words, the system is identified by assuming a parametric model of some suitable structure and then adjusting the parameters such that the discrepancy between the model output and the system output is minimized. The model accuracy will depend on a number of factors such as the choice of model structure, the parameter estimation method, the type of input signal, the complexity of the system, and the nature of the disturbances, w(t) and n(t).

System identification approaches may be "on-line" or "off-line". When the model is constructed off-line, input and output data are recorded and stored in computer memory. The model is then constructed in batch mode, meaning all at once. The advantages of off-line methods are a greater choice of estimation algorithm and a greater freedom in selection of input signals. For these two reasons, the estimation accuracy of these methods are typically superior to on-line approaches.

With on-line system identification, model parameters are updated recursively at each sampling instant. Such methods are suitable for modeling a system whose characteristics change with time, or when it is not practical to wait for all the data to be collected. An example application is an adaptive equalizer which corrects distortion in communication channels. Advantages of the on-line approach are that all the data need not be stored and special input is not required. However, since computation of parameter adjustments must be a fraction of the sampling period, the choice of estimation algorithms is limited.

4.1.1 Input-Output Models

When identifying a system based on input-output data, a common approach is to express the system as function of delayed inputs and outputs. This formulation of the modeling problem may take one of two forms: the series-parallel model or the parallel model. In the series-parallel model, the predicted output, $\hat{y}(t)$ is expressed as a function of previous inputs,*u*, and previous measured outputs,*y*, as expressed by

$$\hat{y}(t) = \hat{P}[u(t-1), u(t-2), \ldots, u(t-l), y(t-1), y(t-2), \ldots, y(t-m)] \qquad (4.2)$$

where *m* and *l* are the orders of the input-output model, respectively; and $\hat{P}$ is the model operator (a linear or nonlinear function). This model is appropriate for applications in which a one step-prediction is required. In other words, the model is intended for use on-line (when the system's output is available in real-time).

In some applications the model is required to be used off-line, i.e., when the system's output is not easily obtained in real-time. In this case, the parallel model must be used in which the actual lagged outputs in Equation 4.2 are replaced by lagged values of the model's predicted output. Thus, the parallel model is expressed as

$$\hat{y}(t) = \hat{P}[u(t-l), u(t-2), \ldots, u(t-l), \hat{y}(t-l), \hat{y}(t-2), \ldots, \hat{y}(t-m)] \qquad (4.3)$$

The disadvantage of the parallel model is that convergence of the model parameters to the desired values is not guaranteed. Even for the linear case, stable adaptive laws for the parallel model have yet to be found (Narendra and Parathasarathy, 1990). On the other hand, if the system is bounded-input/bounded-output stable, the series-parallel model is guaranteed to converge (Narendra and Parathasarathy, 1990). Moreover, if the output error drops to a small value such that $\hat{y}(t)$ approximates $y(t)$, then the series/parallel model may be replaced by the parallel model. That is, once the parameters have been established, the series/parallel model may be operated off-line by feeding back its own predictions *in lieu* of the system outputs.

4.1.2 Model Order

The performance of an input-output model not only depends on the method of function approximation, but also on the determination of the correct model orders (He and Asada, 1993). In general, the order of a model is the number of state variables required to adequately characterize the dynamics of the system. For continuous-time models, this quantity is equivalent to the degree of the system's characteristic equation or the order of the describing differential equation. For discrete-time models, the model order refers to the number of lagged inputs and outputs in the series-parallel or parallel models.

Failure to include adequate dynamics in the input-output model will likely lead to a poor model fit to the data (Billings *et al.*, 1991). Therefore, it follows that a trial-and-error approach may be taken to determine the required number of lagged inputs and outputs. Specifically, the prediction error will show a minimum for those lag values which are optimal. However, this approach results in excessive computation especially when other model parameters must be optimized. In other words, if the model orders can be determined *a priori*, then optimization of other parameters is more efficient in terms of time and computations.

Numerous such methods have been developed to identify the correct model orders for linear systems (Woodside, 1971; Wellstead,. 1976; 1978; Young *et al.*, 1980). However, few results have been reported for nonlinear systems. One recent method (He and Asada, 1993) exploits the continuity property of nonlinear functions which represent input-output models of the system. This technique requires only measured input-output data and has been shown to be effective for determining the correct lags for chaotic dynamic systems and non linear plant models.

4.1.3 The Effects of Noise

Measurement data is inevitably contaminated with noise, from external and internal sources as well as from the measuring instruments themselves. When identifying a system based on input and output observations, noise can result in a biased estimation of the true model parameters. Bias is a statistical term which describes how close the average estimate is to the actual value of the parameter. For example, the parameter estimate, $\hat{\Theta}$, is said to be unbiased if and only if

$$E\left\{\hat{\Theta}\right\} = \Theta \tag{4.4}$$

where $E\{\bullet\}$ is the expectation operator. In other words, the average value of a set of estimates will, in fact, be the true value.

A biased model will likely predict well for the set of data on which the model was trained, but relatively poorly on an unseen data set. That is, the model is biased towards the training set data. This discrepancy indicates that the model provides a curve fit to the data but does not represent the underlying system. Thus, bias is closely related to the concept of generalization.

A common strategy to remove bias is to model the noise process. For example, if the system is perturbed by additive, coloured noise, a linear filter can be identified to remove this effect. When the system is linear, a noise source applied anywhere in the system may be transposed to the output (using the principle of superposition) and dealt with in this manner. For nonlinear systems, superposition does not hold and other methods must be applied.

One strategy is to choose a nonlinear model structure based on assumptions of the noise source location. Depending on where the disturbance is applied three variations of input/output models may be applied.

In the output error model (Equation 4.5), it is assumed that the disturbance is applied at the output, but does not feed back into the system and, therefore, does not effect future outputs. This assumption is generally true when the major source of noise is on the output measuring device.

$$\begin{aligned} r(t) &= P[u(t-1),\ldots,u(t-l),r(t-1),\ldots,r(t-m)] \\ y(t) &= r(t)+n(t) \end{aligned} \tag{4.5}$$

where u(t), r(t), n(t) ,and y(t) are defined as in Figure 4.1 above. This assumption leads to the predictor

$$\hat{y}(t) = \hat{P}[u(t-1),\ldots,u(t-l),\hat{y}(t-1),\ldots,\hat{y}(t-m)] \tag{4.6}$$

which is equivalent to the parallel model of Equation 4.3.

In the NARX (Nonlinear Auto-Regressive with Exogenous inputs) or equation error model, the disturbance is also applied to the output of the system but in such a way that it feeds back and affects future outputs. This system is described by the following equations:

$$\begin{aligned} r(t) &= P[u(t-1),\ldots,u(t-l),r(t-1),\ldots,r(t-m)]+n(t) \\ y(t) &= r(t) \end{aligned} \tag{4.7}$$

This system results in the predictor

$$\hat{y}(t) = \hat{P}[u(t-1),\ldots,u(t-l),y(t-1),\ldots,y(t-m)] \tag{4.8}$$

which is equivalent to the series-parallel model of Equation 4.2.

Finally, with the NARMAX (Nonlinear Auto-Regressive Moving Average with Exogenous inputs) model, it is assumed that the noise is applied to the input of the system, such that the output is a function of the current and previous noise samples as described by

$$\begin{aligned} r(t) &= P[u(t-1),\dots,u(t-l), r(t-1),\dots,r(t-m), w(t-1),\dots,w(t-p)] + n(t) \\ y(t) &= r(t) \end{aligned} \tag{4.9}$$

where w(t) is the system disturbance. In this case, the predictor is described by

$$\hat{y}(t) = \hat{P}[u(t-1),\dots,u(t-l), y(t-1),\dots,y(t-m), e(t-1),\dots,e(t-p)] \tag{4.10}$$

where $e(t) = y(t) - \hat{y}(t)$.

The notion of feeding back the residuals (the e(t)'s) is based on a principle of estimation theory that states the optimal estimates occur when the residuals are uncorrelated with all combinations of past inputs and outputs. That is, as the prediction improves, e(t) increasingly resembles white noise.

4.1.4 Choice of Input Signals

Correct identification of a system also requires the use of an appropriate input signal. The input should be such that it excites all modes of the system (Sinha and Kuszta, 1983). For linear systems it is sufficient that the input has a flat power spectrum over the frequency range of interest, so that the system is excited by a wide spectrum of frequencies. Ideally, the best choice is an impulse function, but such signals are impossible to generate in practice. A suitable alternative is a white noise signal such as the pseudo random binary sequence (PRBS).

For nonlinear systems, the PRBS is not necessarily adequate to excite all the modes because, although its spectrum is flat, it has a constant amplitude in the time domain. In linear systems a constant amplitude can be used since the response at any other amplitude will only differ in amplitude (i.e. by a scaling factor) but not in frequency. This scalability of response does not hold for nonlinear systems, which may exhibit completely different responses for two signals of similar shape but unequal amplitudes. Intuitively, then the PRBS may be modified for nonlinear systems by allowing it to vary in amplitude.

4.2 Modeling Dynamical Systems with Recurrent Neural Networks

Linear system identification techniques are well-known and numerous: frequency response method, step response method, deconvolution, correlation method, least squares estimation, Kalman filtering, maximum likelihood, and instrumental variables. These methods benefit from the well-developed theory of linear systems and the simplifications which come with the assumption of linearity. For example, it is well-known that a linear system is stable if the roots of the characteristic equation are inside the unit circle. However, for nonlinear systems, no equivalent condition exists and stability must be checked on a system-by-system basis (Narendra and Parathasarathy, 1990). Furthermore, if the order of a linear system is known then the model structure is uniquely defined and system identification reduces to the problem of parameter estimation. In contrast, nonlinear systems have an infinite variety of possible representations for a given set of input output observations (He and Asada, 1993). Thus, the choice of model structures must be chosen from a specified set of model classes which are assumed to have the ability to realize the system's input-output behavior (Levin and Narendra, 1996). In other words, the type of model must be justified for the intended application.

Some early methods for modeling nonlinear systems include the works of Volterra (1959), the Hammerstein model (Chang and Luus, 1971) and nonlinear least squares estimation (Hsia, 1968). More recently, numerous nonlinear function approximation and modeling approaches have been developed: radial basis functions (Powell, 1987; Moody and Darken, 1989), Nonlinear Auto-Regressive Moving Average with eXogenous inputs (NARMAX) models (Leontardis and Billings, 1985), fuzzy logic (Takagi and Sugeno, 1985), local technique (Farmer and Sidorowich, 1988). Artificial neural networks, in particular, have received considerable attention for modeling nonlinear dynamical systems (Narendra and Parathasarathy, 1990; Chen *et al.*, 1990; MacMurray and Himmelblau, 1993; Minderman and McAvoy, 1993; Fernandez *et al.*, 1993; Kechriotis *et al.*, 1994; Puskorius and Feldkamp, 1994).

Numerous classes of artificial neural networks can be used to model a dynamical system, including radial basis function networks, recurrent neural networks, and multi-layer perceptrons. To date most applications have focused on the latter due to the availability and simplicity of effective training algorithms such as backpropagation. A common approach is the time-delayed neural network which can be implemented using a multi-layer perceptron in which the input layer consists of a tapped delay line of inputs (Figure 4.2). However, since the model memory is determined solely by the number of input PEs, this approach is limited to systems of relatively low order.

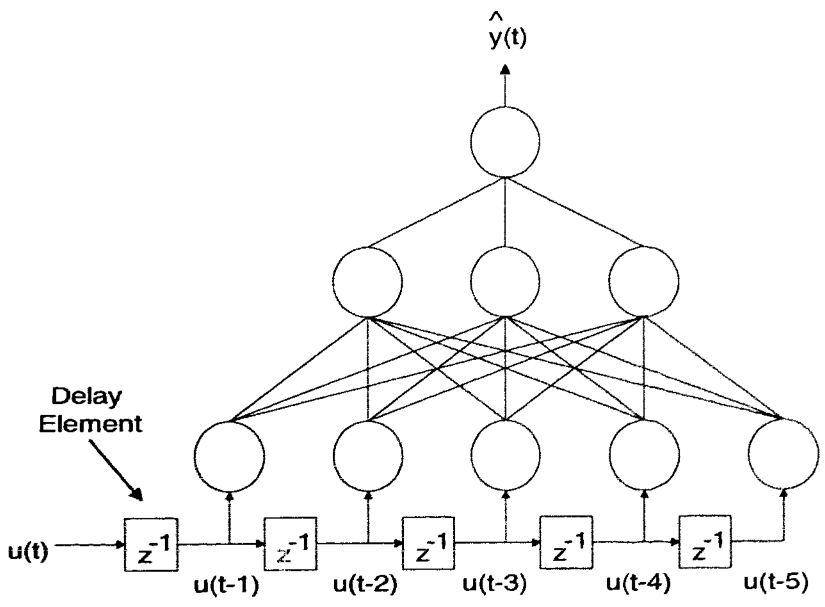

Figure 4.2 Time-delayed neural network.

A variation of the time-delayed neural networks is to also include delayed system outputs in the input layer, resulting in the NARX model of Equation 4.7. While there is an appearance of recurrence in the NARX case, the network is actually static because the lagged outputs consist of actual system measurements. We now turn our attention to the case of truly recurrent neural networks.

Recurrent neural networks, introduced in the works of Hopfield (1982), have long been recognized for their powerful mapping and representational capabilities. Because a RNN has feedback, its memory capacity exceeds that of a time-delayed neural network even when a fraction of the number of PEs are used. (Using signal processing terminology, a time-delayed neural network is a nonlinear finite impulse response filter whereas a RNN is a nonlinear infinite impulse response filter) Despite this advantage, only in recent years have their use become more popular due to the advent of effective 2nd order training algorithms (.e.g. Real-time recurrent learning , Extended Kalman Filtering, Recursive Prediction Error.)

A number of different architectures have been developed including Hopfield networks (1982), recurrent multi-layer perceptrons (Fernandez *et al.,* 1990) , and Elman networks. These variations fall into one of three categories: externally-recurrent, internally recurrent, and fully-recurrent. In an externally-recurrent network (Figure 4.3), information from the output layer feeds back to PEs in the input layer. A network may also be internally-recurrent (Figure 4.4), in which information from one layer may feed back to itself or other layers. The most general case is the fully-recurrent network (Figure 4.5) in which all PEs are interconnected.

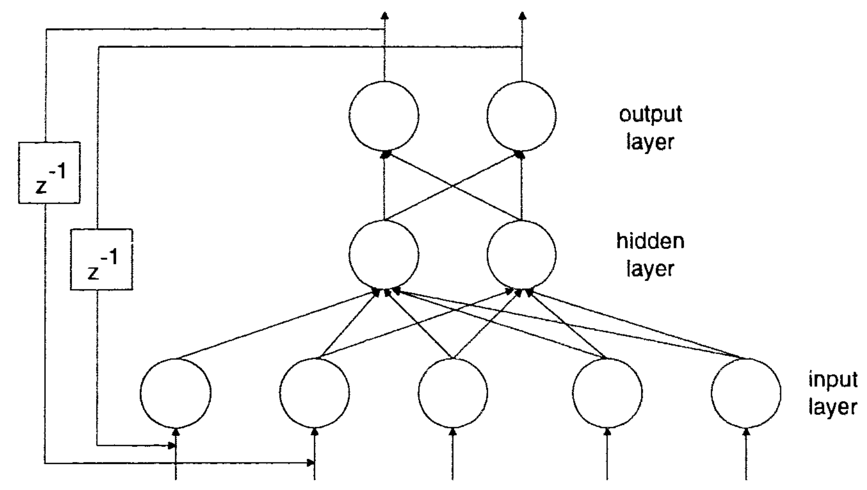

Figure 4.3 Externally-recurrent neural network.

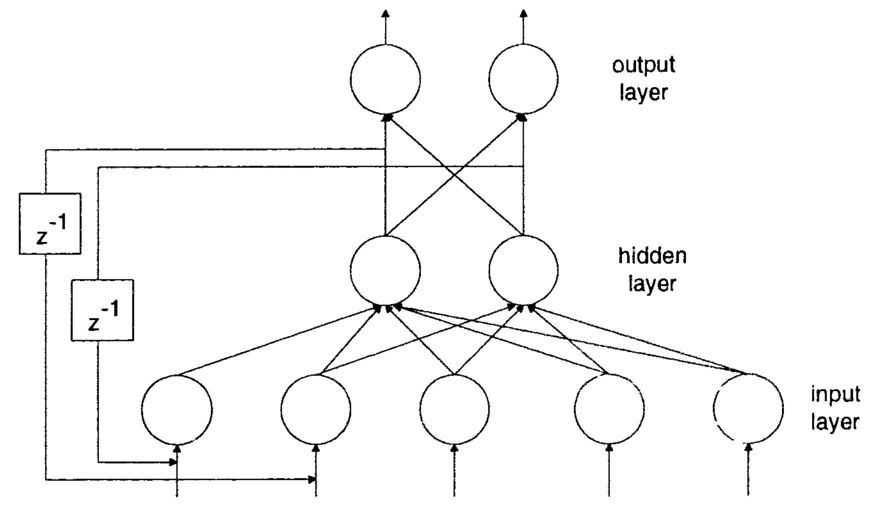

Figure 4.4 Internally-recurrent neural network.

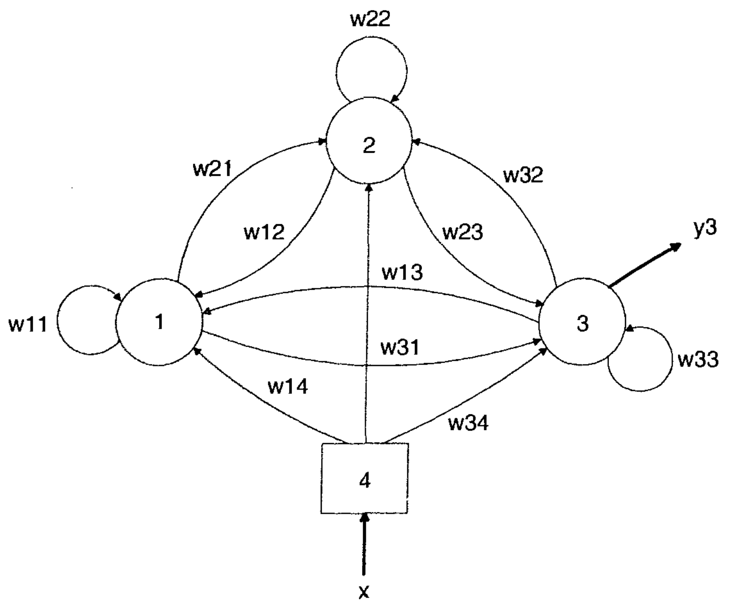

Figure 4.5 Fully-recurrent neural network.

The internally- and fully-recurrent versions have the advantage that the exact number of lagged inputs and outputs does not have to be determined *a priori* (MacMurray and Himmelblau, 1993). Rather, these networks have their own internal memory which stores information about the past. On the other hand, these networks require more computationally-expensive training algorithms due to their inherently recursive structures. (Real-time recurrent learning has a complexity of $O(w^2)$ compared with $O(w)$ for standard backpropagation, where *w* is the number of adjustable .veights). Moreover, an externally-recurrent network can be trained using actual system outputs in the input layer as opposed to the network's own predictions. In other words, the network is trained as a series-parallel model, which facilitates parameter convergence, and then operated as a parallel model, whereby the network's predicted outputs are fed back to the input layer.

CHAPTER 5 METHODOLOGY

5.1 Statement of Problem

Simply stated, the objective of this thesis is to model the mechanical response of the spine to seat-imposed vibration and shocks. A system identification approach has been taken to meet this objective, so that the model will be developed using experimentally measured input and output data. Furthermore, since evidence exists to suggest that the mechanical response is nonlinear in nature, a nonlinear systems identification approach will be taken which will utilize an artificial neural network.

The problem is described by the system in Figure 5.1 (a). The input to the system, **s(t)**, is acceleration at the vehicle seat, and the output, **v(t)** is acceleration at the vertebrae. The seat motion consists of linear and angular acceleration in six degrees of freedom as depicted in Figure 5.1 (b). (the x,y, z axes and rotation about these axes). Seat acceleration transmits upwards through the spine resulting in acceleration of each vertebra, also in six degrees of freedom. Moreover, the response of the spine will differ at each vertebra.

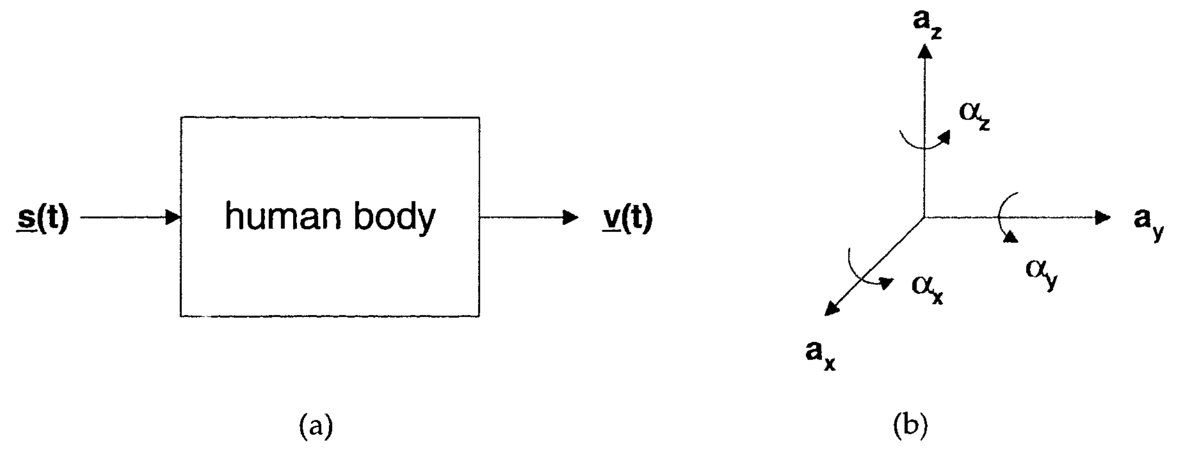

Figure 5.1 (a) Block diagram of the seat-spine system; (b) Degrees of freedom in terms of acceleration of the human body.

The system in Figure 5.1 (a) can be described mathematically by an operator, **P**, which maps the seat acceleration to the vertebrae acceleration such that

$$\mathbf{v}(t) = \mathbf{P}[\mathbf{s}(t)] \tag{5.1}$$

where

$$\mathbf{s}(t) = \begin{bmatrix} a_x(t) & a_y(t) & a_z(t) & \alpha_x(t) & \alpha_y(t) & \alpha_z(t) \end{bmatrix}$$

and

$$\mathbf{v}(t) = \begin{bmatrix} \mathbf{a}_x & \mathbf{a}_y & \mathbf{a}_z & \alpha_x & \alpha_y & \alpha_z \end{bmatrix}.$$

Note that **v**(t) is matrix of dimension N × 6 matrix where

$$\mathbf{a}_\bullet = \begin{bmatrix} a_{\bullet,1}(t) & \dots & a_{\bullet,N-1}(t) & a_{\bullet,N}(t) \end{bmatrix}^T,$$
$$\alpha_\bullet = \begin{bmatrix} \alpha_{\bullet,1}(t) & \dots & \alpha_{\bullet,N-1}(t) & \alpha_{\bullet,N}(t) \end{bmatrix}^T,$$

and N is the number of vertebrae in the spine. Thus, **P** maps the six acceleration components at the seat to the six acceleration components at each of the N vertebra.

To simplify the modeling task, a number of constraints are imposed. Firstly, the problem is limited to seat and spine acceleration along the z-axis. One disadvantage of this approach is that it ignores the cross-axis transmission; an impact at the seat in the x or y axis will result in acceleration in the z-axis at the vertebral level. However, this constraint may be justified on several grounds. Shocks in the z-axis typically have the largest amplitude since mechanical impacts travel upwards through the vehicle and seat into the body, and therefore presumably have the greatest effect on tissues. As some studies suggest compression of the spine, due largely to z-axis shocks, results in fatigue failure of vertebral end plates.

Secondly, the model will only predict acceleration at a single vertebra. This constraint greatly reduces the complexity of the model and the number of computations required in the identification process. It should be fairly straightforward to extend the model to other spinal levels. The lumbar level was chosen because the main health effect of exposure to vibration and mechanical shock reported in the literature is lower-back pain. Moreover, while only the response at a single lumbar vertebra has been modeled, it has been shown (Cameron *et al.*, 1996) that the shape and magnitude of the response at adjacent vertebrae is similar.

The third constraint is optimizing the model based on data collected for only one subject. Ideally, the model would be universal such that it predicts the response for the entire population of vehicle occupants. Realistically, we can only hope to characterize the average response. System identification techniques are intended for modeling one particular system, whether it is a biological process or an industrial plant. In this case, however, we wish also to characterize an entire class of systems (i.e. more than one subject) so that the resulting model is universal in its applicability. A universal mode will not only depend on how well we identify this one system but also on the degree of variance between individuals and the degree to which the chosen subject represents the average response

Given the above constraints, the identification problem is now formulated as follows:

We wish to predict the z-axis acceleration at the fourth lumbar vertebra (L4) for one particular subject of a military population, for the types of z-axis seat impacts and vibration experienced by occupants in tactical ground vehicles. The developed model is intended to predict the spinal response based on vehicle seat only, as it is impractical to measure the vertebral acceleration of vehicle passengers in the field. The simplified system to be identified is shown in Figure 5.2

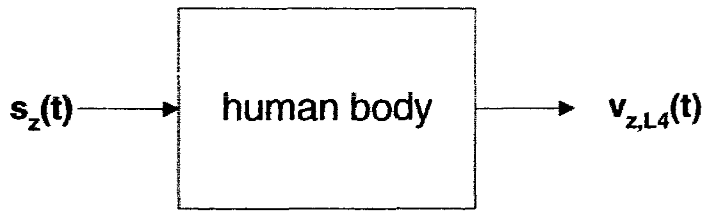

Figure 5.2 Simplified seat-spine system, showing the seat z-axis acceleration, $s_z(t)$, and the z-axis acceleration at the fourth lumbar vertebra, $v_{z,L4}(t)$.

If the model can predict the lumbar acceleration then methods exist to relate this to effective stress on the vertebral tissues. Using material fatigue theory and epidemiological data of back injuries it is possible to estimate the probability of injury due to exposure to mechanical shocks (BC Research Incorporated, 1996). The exposure time for vehicle occupants may then be limited to maintain the risk of injury below an acceptable level.

The strategy to be employed in the development of the model is as follows:

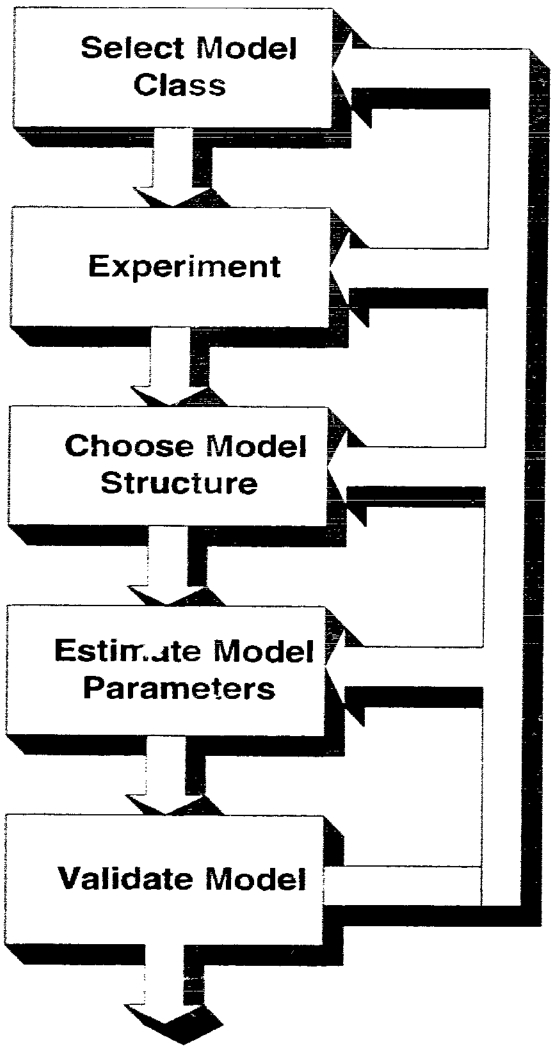

Figure 5.3 System Identification Flowchart.

The flowchart illustrates the iterative nature of system identification. In many cases it may be necessary to back up a step and re-examine the assumptions made about the system. Most development time is spent in steps 3 to 5. We try to adjust the structure and estimate parameters to obtain a good model. If we are unsuccessful we may have to take more drastic measure such as re-designing the experiment (perhaps the test data does not excite all the system modes adequately) or choosing another class of model. These steps are discussed in the following sections.

Except where noted, the data processing and model development was performed using the MatLab software package (Mathworks), running on an IBM 486 - 50 MHz Personal Computer. The neural network training and simulation routines were provided by a publicly available neural network MatLab toolbox (Norgaard, 1995). In many cases, these programs were modified to suit this application. All MatLab m-file scripts are provided in Appendix A.

5.2 Selection of model class

5.2.1 Justification for a nonlinear model

The choice of a nonlinear model was based on several pieces of evidence which indicate that the response of the spine to seat shocks is nonlinear. Cameron *et al.* (1996) found that a nonlinear relationship exists between the peak z-axis acceleration measured at L4 and the peak z-axis seat accleration in response to positive and negative seat shocks. This relationship is illustrated in Figures 5.4 and 5.5. If the response was linear then one would expect the plotted ratios to be constant over different ranges of seat shock amplitudes, based on the homogeneity property of linear systems (Chen, 1989).

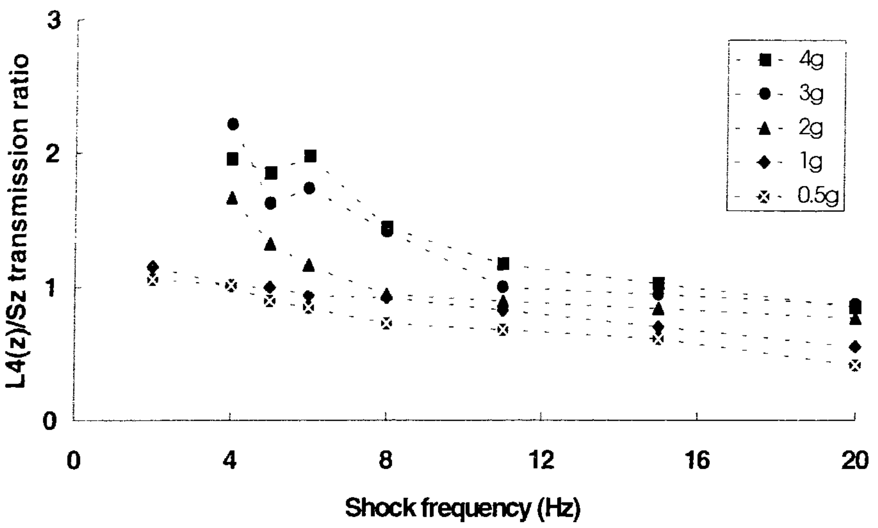

Figure 5.4 Peak transmission ratio versus frequency for positive amplitude seat shocks. (From Cameron *et al.*, 1996).

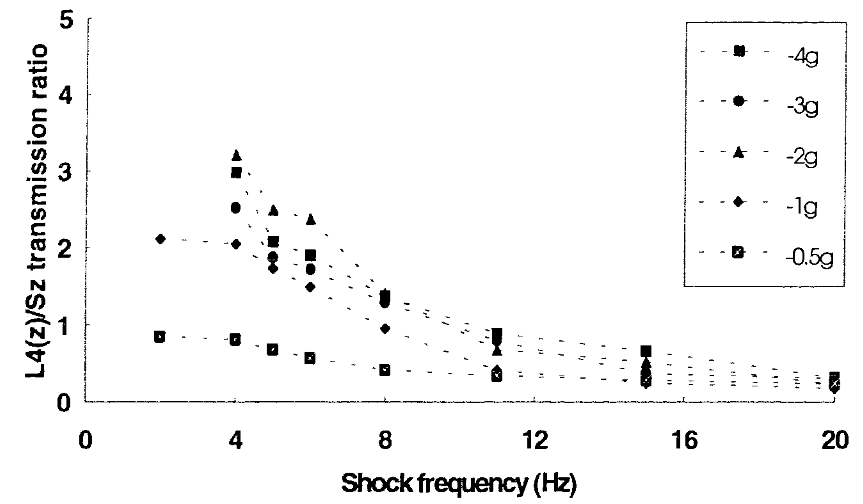

Figure 5.5 Peak transmission ratio versus frequency for negative amplitude seat shocks. (From Cameron *et al.*, 1996).

In addition, the shape of the response at the spine changes with the amplitude of the seat shock. This is demonstrated in Figures 5.6 and 5.7 which show the L4 acceleration in response to 4Hz shocks of amplitudes 3g and -3g, respectively. If the response was linear then the two responses would have the same shape and differ only by a scale fac⁺or of -1.

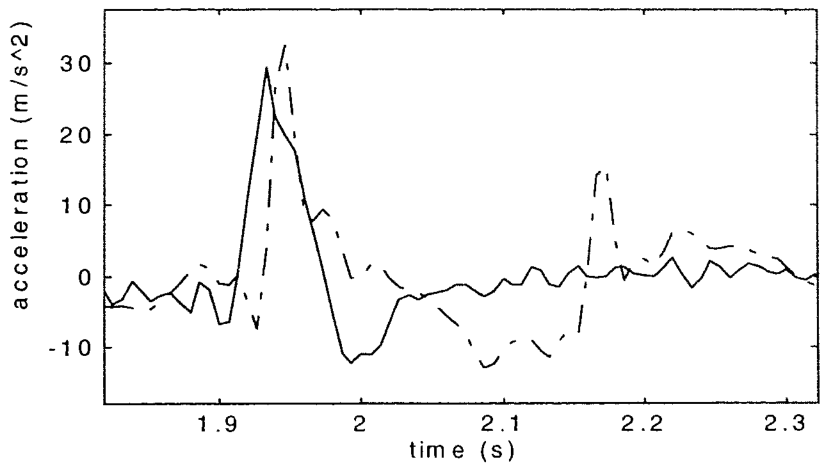

Figure 5.6 L4 response (dashed line) to 3g, 4Hz seat shock (solid line).

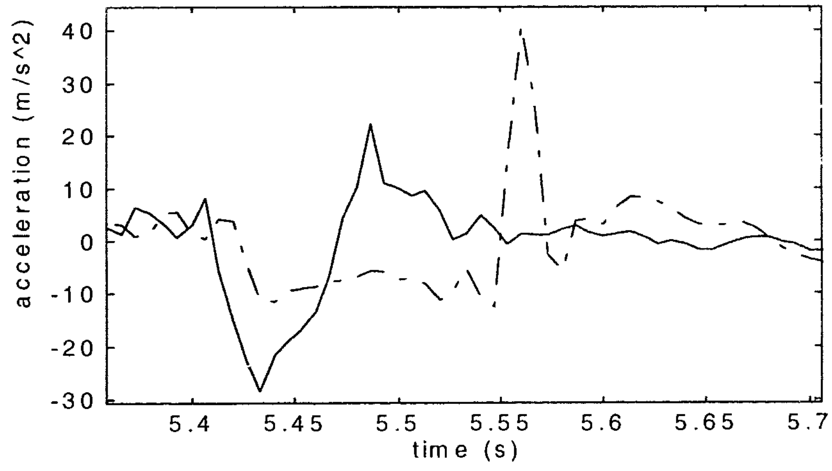

Figure 5.7 L4 response (dashed line) to -3g, 4Hz seat shock (solid line).

The nonlinear response of the spine has also been reported by several researchers (Crocher and Higgins, 1967; Griffin, 1986). Finally, the different natural frequencies of the Fairley-Griffin and DRI models representing low amplitude and high amplitude vibration, respectively, imply the nonlinear characteristic of the human body. Hence, the use of a nonlinear model appears to be warranted.

5.2.2 Justification of the neural network model

The choice of an artificial neural network to model the system is justified on the basis of the Stone-Weierstrass theorem (Cotter, 1991) which states that any continuous function can be approximated arbitrarily closely by a polynomial of sufficient degree.. Stone-Since it has been demonstrated that a neural network with an arbitrary number of PEs in a single hidden layer is capable of approximating any polynomial (Cybenko, 1989; Funahashi, 1989), it follows that an ANN is a universal approximator. A corollary to this axiom is that a recurrent network of correct order can model any continuous system.

To model the system, an externally-recurrent neural network was chosen which represent the parallel identification model

$$\hat{y}(t) = P[u(t-1),\ldots,u(t-l),\hat{y}(t-1),\ldots,\hat{y}(t-m)] \tag{5.2}$$

This above model is the nonlinear output error(NOE) formulation given in Section 4.2 and is recurrent of order m. The choice of this model class, as opposed to the NARMAX, or NARX formulation, was based on the need for the model to be used off-line. (Recall that both the NARMAX and NARX models require the actual system output to generate their prediction.)

It is possible to first train the model using the NARX structure and then to use this model off-line. Training involves the finding the set of network parameters for which Equation 3.11 is minimized.

When Equation 4.2 (the NARX model) is substituted into the above equation for $\hat{y}(t)$, the minimization process is identical to the maximum likelihood estimation approach (Werbos, McAvoy, and Su, 1992). We will, therefore, refer to the network parameters optimized in this manner as the maximum likelihood values.

There are two problems with the NARX (or maximum likelihood) approach. Firstly, the resulting model is optimized for single-step ahead prediction and likely will not perform well as a multi-step predictor. During multi-step forecasting, prediction errors feed back into the input of the model resulting in continually worse predictions. Secondly, noise can result in a biased parameter estimates if the NARX model is employed.

However, the output error model is difficult to train because, in general, neither convergence nor stability is guaranteed for parallel identification models. This difficulty is overcome using a combination training approach, called the Compromise Method (Werbos *et al.*, 1992). Essentially, an NARX model is trained until convergence is attained. The NARX model parameters are then used as initial values for the training of a NOE model. The details of this training method are discussed in Section 5.4. The problem of model bias due to noisy training data is overcome by validating the model on a test data set.

5.3 Experiment

Acceleration data was collected at the multi-axis ride simulator (MARS) of the U.S. Army Aeromedical Research Laboratory, Fort Rucker, Alabama, during a series of experiments designed to assess the health effects of repeated impacts (Cameron *et al.*, 1996). These experiments were performed by researchers of USAARL and BC Research Inc. and are not considered part of this thesis.

The MARS consists of a hydraulically-driven platform, two hydraulic pumps in parallel, three orthogonal hydraulic actuators, three fail-safe valves, and a Schenck/Pegasus multi-channel servo controller, all of which are controlled by a DEC-PDP11 computer. The MARS has an acceleration range of ±4g, a displacement range of ±8.9 cm (3.5 in.) and a frequency response of 2-40 Hz. The vibration frequency and acceleration amplitude are controlled by a computer synthesized displacement command signal.

The subjects for these experiments were 10 healthy, male army volunteers between the ages of 19 and 40 years. All subjects had previous experience with motion from military or civilian exposure (eg. TGV's, air transport, participation in Operation Desert Shield/Storm). Prior to the experiment, subjects were required to pass a medical examination, attended a briefing session, signed informed consent forms, and participated in a trial exposure in order to become familiar with the motion and instrumentation.

The subject sat on the MARS platform (Figure 5.8), which simulated accelerations typical of those experienced during cross-country vehicle rides. The subject sat face forward in a comfortable upright position on a seat with no back rest (most occupants of TGVs don't have backrests)

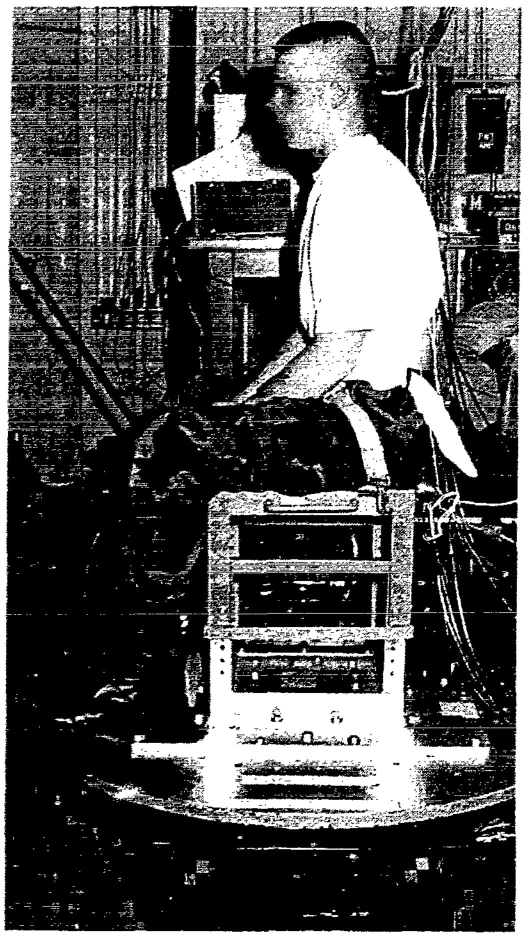

Figure 5.8 MARS table and subject.

The MARS was programmed to apply a series of acceleration impulses in the vertical direction (biodynamic z-axis), having various waveform frequencies and acceleration magnitude profiles. Six sets of input signals were used, each of which was applied for a duration of 5.5 minutes. Each shock was presented twice. All signal sets contained gaussian background vibration (RMS value of 0.05g and a bandwidth of 2-40Hz) and periodic shocks that occurred at a rate of 6 shocks/minute. These signals were designed on the basis of a number of vehicle characterization experiments (Roddan *et al.*, 1995); US Army TVGs were driven in operational manner over typical terrain and seat accelerations were measured in three axes.

Each input shock consisted of a single cycle damped sinusoid as depicted in Figure 5.9. The amplitude and frequency range (f = 1/T, where T is the period of the shock waveform) of the applied shocks were +/-0.5g to +/- 4g, and 2 Hz to 20 Hz, respectively.

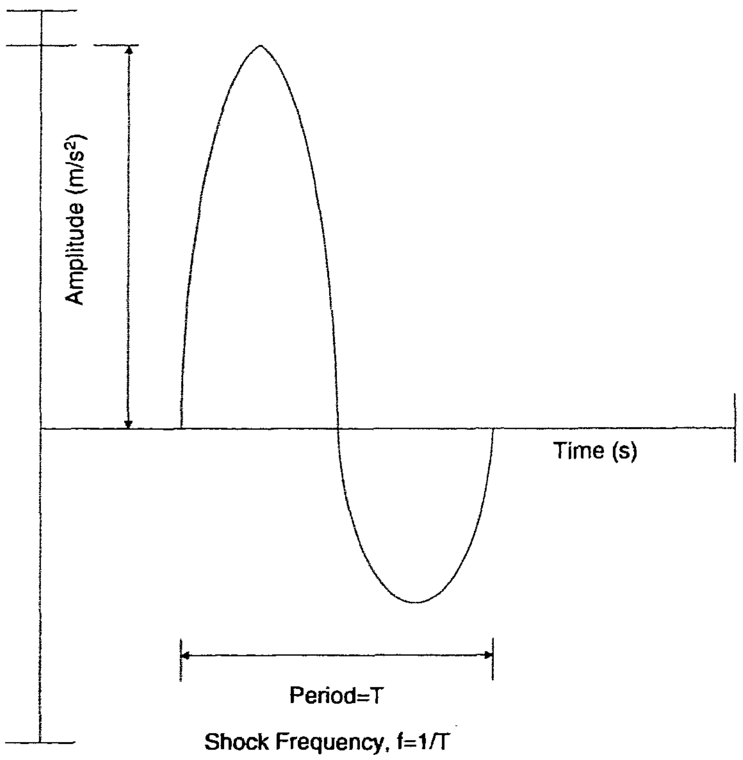

Figure 5.9 Input shock waveform diagram.

The MARS seat acceleration was measured with a tri-axial Entran accelerometer housed in a flexible epoxy seatpad, built according to specs SAE(1974). The seatpad was positioned on the seat between the subject and a thin bean-bag cushion. Both cushion and seatpad were secured firmly to the metal frame of seat by tape. Acceleration over the spinous process of the L4 vertebra was measured with a 0.3 gram miniature accelerometer (Entran EGAX ±25 g) attached to the skin by a small square of two-sided carpet tape with a thin strip of surgical tape placed over the top.

The seat and spine acceleration data were sampled at 500 Hz and high-pass filtered at 0.5 Hz. The measured seat acceleration contained high frequency components that were imposed on the original shock signal computed for input to the MARS controller. The main source of these high frequency components was the result of the subject leaving the seat and then impacting the seat in response to both positive and negative 2, 3, and 4g shocks. These components were removed by low-pass filtering the seat data at 110Hz. The lumbar acceleration was processed to compensate for the movement of skin over the vertebrae. Details of this procedure from Cameron *et al.* (1996) are provided in Appendix B.

5.3.1 Data Processing for Model Development

For the purpose of developing a neural network model, the seat and lumbar acceleration files were combined into two data sets--a training data set and a testing set. Each set consisted of a seat shock (input) file and a lumbar response (output) file. Input shocks and their corresponding spinal responses were copied from the raw data files and compiled in either the training or testing data set. Approximately 60% of the data were used for the training set and the remaining 40% were saved for validation purposes. The order of shocks in both sets were randomized to obtain an equal representation of shock frequencies and magnitudes in each set. The number and type of shocks represented in the training and testing sets are listed in Tables 5.1 and 5.2, respectively.

Table 5.1 Training Data Profile

Amplitude	Frequencies
+4g	5,6,6,8,15,15,20,20
+3g	4,5,8,20,20
+2g	6,8,8,20
+1g	4,4,5,5,6,6,8,8,15,20,20
-1g	4,4,5,5,6,6,8,15,15,20,20
-2g	5,5,11,12,20
-3g	4,6,8,11,12,15,20,12
-4g	4,5,5,6,8,11,11,15,20

Table 5.2 Testing Data Profile

Amplitude	Frequencies
+4g	4,4,5,5,8,11,11
+3g	8,15,15
+2g	5,5,6,11
+1g	2,2,8,11,11,15
-1g	2,2,8,11,11,15
-2g	20,6,8,8
-3g	5,8
-4g	4,5,5,8,15,15

During this process, significant sections of the signals were removed that did not contain any shocks, only background vibration. This editing was necessary so that the neural network training would not be overly biased toward learning the response of the system to the vibration as opposed to high-amplitude shocks.

Power spectral analysis of the condensed data indicated that most significant signal energy occurred at frequencies less than 75 Hz. To reduce the amount of data, the signals were down-sampled to 150 Hz, using the **resample** function provided by MatLab Signal Processing Tool Box. Down-sampling had the effect of attenuating some of the peaks of the spinal acceleration by roughly 5-10%. This attenuation was the direct result of the anti-aliasing filter built into the resample function which removed frequencies greater than 75 Hz. However, it was decided that the removal of this portion of the signal's spectrum was warranted by the 50% reduction in the amount of data to be processed. This reduction resulted in savings in processing time and computer memory which in turn allowed testing larger neural network architectures.

5.3.2 Determination of Training Subject and Training/Testing Data

In order that the resulting model can be generalized to other subjects, the neural network was trained with data from the subject that exhibited the most typical response. This data set was found by averaging the responses for all of the subjects, and then determining the individual response which was closest to the group average according to some distance measure.

The averaging process was carried out in the frequency domain. The power spectral density (PSD) was found for the spinal acceleration response of each subject due to negative input shocks of various frequencies with amplitudes of -1,-2, -3, and -4g. (Positive input shocks are not included in this calculation based on the assumption of symmetry. In other words, if two subjects demonstrated similar responses due to negative input shocks, then they are likely to show similar response due to positive input shocks.) The mean PSD was then calculated by summing the frequency components of each PSD and dividing the result by the number of subjects.

The similarity between the PSD of each individual and the mean PSD was quantified by calculating the normalized root mean square (RMS) error as described by the following equation:

$$E_j = \frac{1}{M}\sqrt{\sum_{i=1}^{M}\frac{\left[\overline{P}(i) - P_j(i)\right]^2}{\left[\overline{P}(i)\right]^2}} \tag{5.3}$$

where $\overline{P}(i)$ is the i'th frequency component of the mean PSD, $P_j(i)$ is the i'th component of the j'th subject's PSD, and M (=64) is the size of the Discrete Fourier Transform used to compute the PSD.

5.4 Selection of Model Structure

5.4.1 Determination of model orders

The optimal model order (input and output lags) was determined using the Lipschitz quotient method developed by He and Asada (1993). This method is described below.

First, we assume that the system is modeled using a generalized form of the input-output model of equation 4.2:

$$y = f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n) \tag{5.4}$$

In this formulation, the x_i' s represent lagged inputs, outputs or both.

Furthermore, we assume that the partial derivatives of y with respect to the input variables exist and are bounded (i.e. $f(\mathbf{x})$ is continuous and smooth) such that

$$|f_i| = \left|\frac{\partial f}{\partial x_i}\right| \leq M \qquad \text{for i} = 1,\ldots,n \tag{5.5}$$

The objective of system identification is to approximate $f(\mathbf{x})$ based on the input-output data pairs $(\mathbf{x}_i, y_i)$. Define the Lipschitz quotient q_{ij} as

$$q_{ij} = \frac{|y_i - y_j|}{|\mathbf{x}_i - \mathbf{x}_j|} \qquad i \neq j\,. \tag{5.6}$$

The denominator $|\mathbf{x}_i - \mathbf{x}_j|$ is the Euclidean distance between points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the input space and numerator $|y_i - y_j|$ is the difference between $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$. If $f(\mathbf{x})$ is continuous, the Lipschitz condition states that the Lipschitz quotient is bounded for any input-output data pairs, such that

$$0 \le q_{ij} \le L \tag{5.7}$$

By applying a sensitivity analysis to the relationship between q_{ij} and the number of input variables, it can be shown that

$$q_{ij}^{(n)} = \frac{|\partial y|}{\sqrt{(\partial x_1)^2 + \ldots + (\partial x_n)^2}} \le M \frac{|\partial x_1 + \ldots + \partial x_n|}{\sqrt{(\partial x_1)^2 + \ldots + (\partial x_n)^2}} \le \sqrt{n} M \tag{5.8}$$

where the superscript (n) represents the number of input variables included in the input-output model. This expression states that if $f(\mathbf{x})$ satisfies $|f_i| \le M$ and all the input variables are included in the reconstruction of $f(\mathbf{x})$, then the Lipschitz quotient must be less than $\sqrt{n}M$ for all input-output pairs. When one or more input variables are missing, the Lipschitz quotient may be very large or unbounded. On the other hand, if a redundant variable is included, the value of the quotient will only be slightly smaller. Hence, information from these quotients may be used to determine the optimal number of input variables required. Specifically, this is done by calculating the following index

$$q^{(n)} = \left[\prod_{k=1}^{P} \sqrt{n} q^{(n)}(k) \right]^{1/P} \tag{5.9}$$

where $q^{(n)}(k)$ is the k'th largest Lipschitz quotient among all $q^{(n)}{}_{ij}$ $(i \ne j; i,j = 1,2,\ldots,N)$; n is the number of input variables $(x_1, x_2, \ldots, x_n)$; and P is a positive integer chosen in the range $P = 0.1N \ldots 0.2N$. By plotting this index (the

Lipschitz number) against n, the optimal model orders can be visually determined; they correspond to the value of n at which the curve begins to flatten out.

A listing of the Turbo-C code for the Lipschitz method is provided in Appendix C.

5.4.2 Determination of Hidden Layers

The model was developed using only one hidden layer due to limitations of the neural network software. However, only one hidden layer is theoretically required to model any nonlinear system. Furthermore, this limitation actually simplifies the modeling task in that the number of network structures to be optimized is reduced. The remaining task is to determine the number of PEs in the one hidden layer.

The number of hidden layer PEs was determined using a trial and error approach described in Werbos *et al.* (1992). The objective is to find the number of PEs for which the error on the test set is minimized. In doing so, caution must be taken to avoid confounding the results by the effect of different starting points. In other words, has the error improved due to the number of hidden PEs or because of where the algorithm started on the error surface? To answer this question we can repeatedly train each network (with a constant number of PEs), always initializing the weights randomly. We can then choose the network that yielded either the lowest average error or the lowest overall error . The disadvantage of this technique is the large number of computations involved.

Instead of the above approach, the network was trained only once, but always with the parameters initialized to zero. This constant initialization avoids the effect of different starting points. Unfortunately, initializing all the weights to zero results in final weight values which are all identical. To avoid this problem, different learning rates were assigned to each weight. As in Werbos *et al.* (1992), the learning rate was multiplied by a linearly decreasing factor which varies with the number of hidden layer PEs. Thus, the learning rate is actually a vector given by $\mathbf{v} = \alpha[1 \quad 1 \quad \ldots \quad 2/n \quad 1/n]$.

where α is a small positive constant, and n is the number of hidden PE's. This learning rate had the effect of multiplying the gradient by the matrix $\text{diag}\begin{bmatrix}1 & 1 & \dots & 2/n & 1/n\end{bmatrix}$. It can be proved that an optimization algorithm using this transformed gradient will lead to a minimum in the error (Werbos *et al.*, 1992). Therefore, this technique allowed the weights to be initialized to zero.

The following cross validation procedure was then used to determine the optimal number of hidden PE's:

1. Start with $n = 1$.

2. Initialize the network parameters to zero as discussed above.

3. Train the network model to a minimum, using the Levenberg-Marquardt algorithm (described in Section 5.4).

4. Validate the model on the test data set.

5. Let $n = n + 1$ and repeat steps 2 through 5, until the prediction error on the test set begins to increase.

Once the correct number of hidden PEs was determined, a similar method was used to optimize the number of training iterations, Ni. Both optimizations were performed on the equation error model formulation since training is significantly faster than the output error model. The optimal number of n and Ni were then used in the compromise method to train the model as described below.

5.5 Parameter Estimation

Estimation of the correct network parameters consisted of two steps. The Compromise Method was used to determine the cost function to be optimized. Essentially, the Compromise Method configures the cost function for optimization in terms of either the NARX model, the NOE model or an average of the two. In the second step, the

Levenberg-Marquardt algorithm was then used to optimize the parameters of the specified cost function. These two steps are discussed further in the following sections.

5.5.1 The Compromise Method

The Compromise Method determines the way in which the cost function, J (Equation 5.2), is calculated. Rather than minimizing J in terms of the NARX model or the NOE model, the Compromise Method combines these approaches by defining an alternative input-output model

$$\hat{y}(t) = P[\bar{y}(t-1),\ldots,\bar{y}(t-m),u(t-1),\ldots,u(t-l)]. \tag{5.10}$$

The signal $\bar{y}(t)$ which is fed back to the network input layer is a weighted average defined as

$$\bar{y}(t) = (1-w)\hat{y}(t) + wy(t), \tag{5.11}$$

where $\hat{y}(t)$ is the network's prediction, y(t) is the actual output, and w is a constant which varies between 0 and 1. The cost function to be minimized is still

$$J(\Theta) = \frac{1}{2N}\sum_{t=1}^{N}[\hat{y}(t,\Theta) - y(t)]^2 \tag{5.12}$$

but the network output signal is calculated differently, depending on the value of w. Note that when the w = 1, the above expression is equivalent to the maximum likelihood (NARX) case, and when w=0, it is equivalent to the NOE case.

The Compromise Method then consists of the following steps.

1. Initialize the network parameters randomly.

2. Setting w = 1, train the network, using the Levenberg-Marquardt method to determine the maximum likelihood values.

3. Initialize the parameters to their maximum likelihood values and re-train the network using successively stiffer forms of compromise (w = 0.8, 0.5, 0.1, 0.01, 0.001).

4. Repeat steps 1-3 several times and choose the network which exhibits the lowest error on the test set. This last step helps the algorithm avoid local minima.

5.5.2 The Levenberg-Marquardt Method

The Levenberg-Marquardt method is a variation of the Gauss-Newton method for finding the minimum of a function. The concept behind all Newton-like optimization methods is to approximate the function f(x) about some point $f(\mathbf{x}+\delta)$ by a truncated Taylor series expansion:

$$f(\mathbf{x}^{(k)}+\delta) \approx q^{(k)}(\delta) = f^{(k)} + \mathbf{g}^{(k)T}\delta + \frac{1}{2}\delta^T\mathbf{H}^{(k)}\delta \qquad (5.\ 13)$$

In this equation $\mathbf{g}^{(k)}$ and $\mathbf{H}^{(k)}$ are the gradient vector and Hessian matrix of f(x), respectively, calculated at $\mathbf{x} = \mathbf{x}^{(k)}$. The minimum of f(x) can then be found iteratively by finding the minimum of the quadratic approximating function, $q^{(k)}(\delta)$. The minimizing value of δ is calculated by setting the derivative of Equation 5.13 to zero and solving. That is, we solve

$$\mathbf{H}^{(k)}\delta = -\mathbf{g}^{(k)}. \qquad (5.14)$$

The k'th iteration of Newton's method can be written as:

1. Solve Equation 5.14 for δ.

2. Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta^{(k)}$.

The Gauss-Newton method differs from Newton's method in that it utilizes a first order approximation of the Hessian.

A desirable property of an optimization algorithm is global convergence, which means that the algorithm will converge to a local minima regardless of the starting point. In other words, the minima will be found in a finite number of iterations even if $\mathbf{x}^{(0)}$ is far from the solution. However, global convergence does not imply that the procedure finds the global minimum of f(x). The latter can usually be found by repeating the algorithm from different initial points, and choosing the minimum of all local minima found.

In all Newton-like methods, a necessary condition for global convergence is that the Hessian or the approximate Hessian is a positive definite matrix. However, this condition may not be true, especially if $\mathbf{x}^{(k)}$ is far from the solution. To solve this problem, the Levenberg-Marquardt approach biases the search direction towards the steepest descent direction through the addition of a small positive constant, λ, to the diagonal elements of **H**. The system to solve thus becomes

$$\left[\mathbf{H}^{(k)} + \lambda\mathbf{I}\right]\delta^{(k)} = -\mathbf{g}^{(k)} \qquad \lambda > 0 \tag{5. 15}$$

and we iterate steps 1 and 2 as before. If λ is small then the search direction δ approaches the Gauss-Newton direction, $-\mathbf{H}^{-1}\mathbf{g}$, whereas if λ is large then δ approaches the steepest descent direction, -g.

The version of Levenberg-Marquardt described by Fletcher (1987) adapts the size of λ based on the closeness of fit between f(x) and its quadratic approximation. $q^{(k)}(\delta)$. This closeness is measured as a ratio between the actual reduction in f(x) given by

$$\Delta f^{(k)} = f^{(k)} - f(\mathbf{x}^{(k)} + \delta^{(k)}) \tag{5.16}$$

and the predicted reduction in f(x) (based on $q^{(k)}(\delta)$) given by

$$\begin{aligned}\Delta q^{(k)} &= q^{(k)}(0) - q^{(k)}(\delta^{(k)}) \\ &= f^{(k)} - q^{(k)}(\delta^{(k)})\end{aligned} \quad . \tag{5.17}$$

We, therefore, define the performance metric upon which we predicate changes in the size of λ as

$$r^{(k)} = \frac{\Delta f^{(k)}}{\Delta q^{(k)}} \ . \tag{5.18}$$

This version of Levenberg-Marquardt algorithm is as follows:

1. Given $\mathbf{x}^{(k)}$ and $\lambda^{(k)}$, calculate $\mathbf{g}^{(k)}$ and $\mathbf{H}^{(k)}$.

2. Solve Equation 5.15 to give $\delta^{(k)}$.

3. Evaluate $f(\mathbf{x}^{(k)} + \delta^{(k)})$ and hence $r^{(k)}$.

4. If $r^{(k)} < 0.25$ set $\lambda^{(k+1)} = 2\lambda^{(k)}$.

   If $r^{(k)} > 0.75$ set $\lambda^{(k+1)} = \frac{\lambda^{(k)}}{2}$

   Otherwise set $\lambda^{(k+1)} = \lambda^{(k)}$

5. If $r^{(k)} \le 0$ set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$, otherwise set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta^{(k)}$.

We can now formulate this algorithm in terms of the neural network model that we wish to train. Let the network parameters (weights and bias values) be contained in the vector $\boldsymbol{\Theta}$ as in Chapter 2. Furthermore, define the error function

$$J(\Theta)=\frac{1}{2N}\sum_{t=1}^{N}[e(t,\Theta)]^2 \tag{5.19}$$

where $e(t,\Theta)$is the discrepancy between the actual output y(t) and the network prediction $\hat{y}(t,\Theta)$, and N is the length of the training data. The objective then is to train the network by finding the Θ which minimizes J. This objective is achieved using the Levenberg-Marquardt algorithm to iteratively search the error surface, J, in the direction specified by the vector $\mathbf{s}(t,\Theta^{(k)})$.

Specifically, the procedure for determining the optimal network parameters is as follows:

1. Initialize the vector Θ and initialize λ to a small positive constant. Set k = 0.

2. Solve $\left[\mathbf{H}^{(k)}+\lambda\mathbf{I}\right]\mathbf{s}^{(k)}=-\mathbf{g}^{(k)}$ for $\mathbf{s}^{(k)}$.

3. Evaluate $J(\Theta^{(k)}+\mathbf{s}^{(k)})$ and hence $r^{(k)}$.

4. If $r^{(k)}<0.25$ set $\lambda^{(k+1)}=2\lambda^{(k)}$ (If the predicted decrease in J is close to the actual decrease, let the search direction approach the Gauss-Newton search direction.)

If $r^{(k)}>0.75$ set $\lambda^{(k+1)}=\frac{\lambda^{(k)}}{2}$ (If the predicted decrease in J is far from the actual decrease, let the search direction approach the gradient (steepest descent) direction.)

Otherwise set $\lambda^{(k+1)}=\lambda^{(k)}$.

5. If $J(\Theta^{(k)}+\mathbf{s}^{(k)})<J(\Theta^{(k)})$, then accept updated parameters for Θ and set k = k +1.

6. If the stop criterion is met (k > maximum number of iterations or $J(\Theta^{(k)}) <$ error bound) is not satisfied then go to step 2.

The gradient of J is calculated using Equation 3.16 and the Hessian is approximated using

$$\mathbf{H}(\Theta^{(k)}) = \frac{1}{N}\sum_{t=1}^{N} \Psi(t,\Theta^{(k)})\Psi(t,\Theta^{(k)})^T \qquad (5.20)$$

where $\Psi(t,\Theta)^T = \frac{d\hat{y}(t,\Theta)}{d\Theta}$

5.6 Model Validation

Having determined the best model parameters, for the chosen structure, we need now to determine whether the model is "good enough". This is the problem of model validation. The question has several aspects:

i) Does the model agree sufficiently well with the observed data?

ii) Is the model good enough for my purposes?

iii) Does the model describe the true system? (Ljung, 1987)

To answer these questions, the proposed model was tested in several ways. First, a pure simulation of the model (i.e., using only the input and not the measured output) was performed to generate the model predicted output. The model predicted output was then plotted against the actual output for visual comparison. This metric is more effective at revealing the inadequacies of the model than simply using the single-step prediction (Ljung, 1987; Billings *et al.*, 1991).

In addition, the model was compared to three linear models using both a visual inspection of the model predicted outputs and an objective error measure. The error measure chosen is the RMS error given by

$$E = \sqrt{\frac{1}{N}\sum_{t=1}^{N}[y(t) - \hat{y}(t)]^2}\,. \qquad (5.21)$$

The linear models chosen were the DRI model, the BS 6841 filter, and a 20'th order ARX model identified from input-output data using Least Squares Estimation (See Appendix D). This comparison will indicate whether the neural network model is an improvement on some of the existing models. It will also help to demonstrate that a linear approach is generally not well-suited to this particular modeling problem.

The above validation process was then repeated for unseen data obtained from a different subject. This test helps to indicate whether the model can be applied to subjects outside the training set.

Finally, using the data from the same unseen data the neural network was validated on input consisting of only low-level vibration. The RMS error and time-series plot of the neural network's output was then compared to those of the DRI, BS 6841, and ARX models. Finally, a power spectral analysis of the outputs was performed in order to compare the performance of the four models in predicting the correct response to vibration.

CHAPTER 6 RESULTS

6.1 Determination of Subject Data

Figure 6.1 shows the average, normalized power spectral density of the L4 response to a single -2g, 4 Hz seat shock. The graph indicates that the majority of the signal energy is between 1 Hz and 25 Hz.

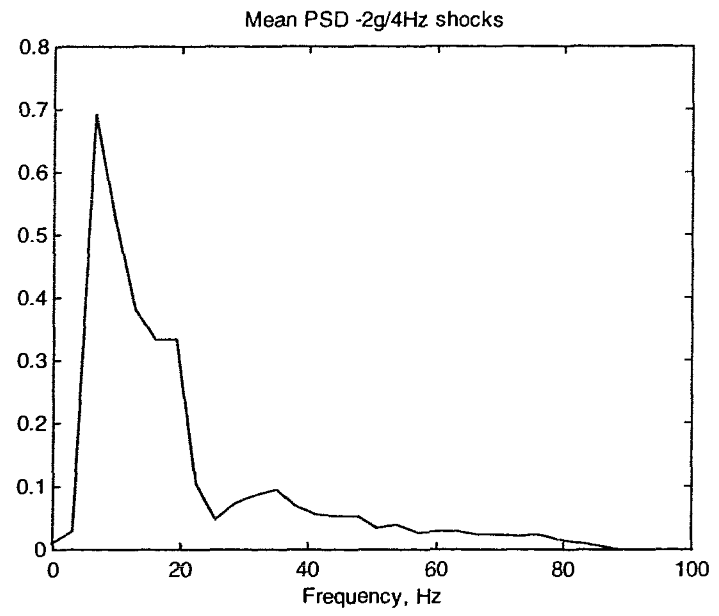

Figure 6.1 The mean normalized power spectral density of the lumbar-4 response.

Table 6.1 lists the RMS error between the mean and individual PSD plots (as defined by Equation 5.3) for all subjects in response to three types of input shocks: -2g/ 4Hz, -3g/15 Hz, and -4g/ 4Hz. The data from subject #1 and subject #11 were incomplete as these subjects did not complete the test protocol. In addition, some of the data obtained from Subjects #3 and #5, contained large amplitude, high frequency noise, possibly due to a loosely attached accelerometer. Therefore, this data was not included in the averaging procedure.

The table indicates that Subject #9 has the lowest aggregate score and, therefore, exhibits a response to all three types of input shocks that is the closest to the average response. Thus, the data from Subject 9 was used for training the model.

Table 6.1 Similarity between subject and mean PSD for various input shock responses, expressed as the normalized RMS error.

/

Subject #	-2g/4Hz	-3g/15Hz	-4g/4Hz	Total
1	not avail.	not avail.	not avail.	
2	0.727	0.825	0.556	2.11
4	0.644	1.320	0.860	2.82
6	0.597	0.750	0.508	1.86
7	1.038	0.547	0.564	2.15
8	0.645	0.477	0.681	1.80
9	0.735	0.501	0.529	1.77
10	2.267	1.303	0.687	4.57
11	0.560	not avail.	not avail.-	

6.2 Determination of Model Orders

The Lipschitz numbers for various input and output lags (Refer to Section 5.3.1.) are listed in Table 6.2. Using the two-dimensional search of the lag space described by He and Asada (1993), three possibilities for the lag structure were identified: 2 previous inputs/2 previous outputs, 2 previous inputs/3 previous outputs, and 3 previous inputs/2 previous outputs. For example, by setting m = 2, we can plot a slice of the lag space (Figure 6.2) and observe how the Lipschitz index changes with the number of delayed inputs. The knee of the curve seems to occur at l=2 or l=3. At this point, the additional input terms have diminishing effect on the Lipschitz index, and, therefore, are not crucial to the reconstruction of f(x).

Table 6.2 Lipschitz number q(n) for various input lags (l) and output lags (m).

	m= 1	*m*= 2	*m*= 3	*m*=4	*m*=5	*m*= 6	*m*= 7
l = 0	144479.5	517.2	53.2	33.0	21.5	18.1	17.1
l = 1	502.9	50.2	21.0	15.2	13.3	13.6	13.2
l = 2	118.0	29.0	17.0	14.0	13.0	13.3	13.4
l = 3	71.9	23.2	15.6	14.2	13.1	13.4	13.6
l = 4	59.1	19.7	14.3	14.2	13.2	13.6	13.8
l = 5	54.4	18.6	14.6	14.7	13.4	13.9	14.1
l = 6	48.3	17.9	14.8	15.1	13.7	14.1	14.4

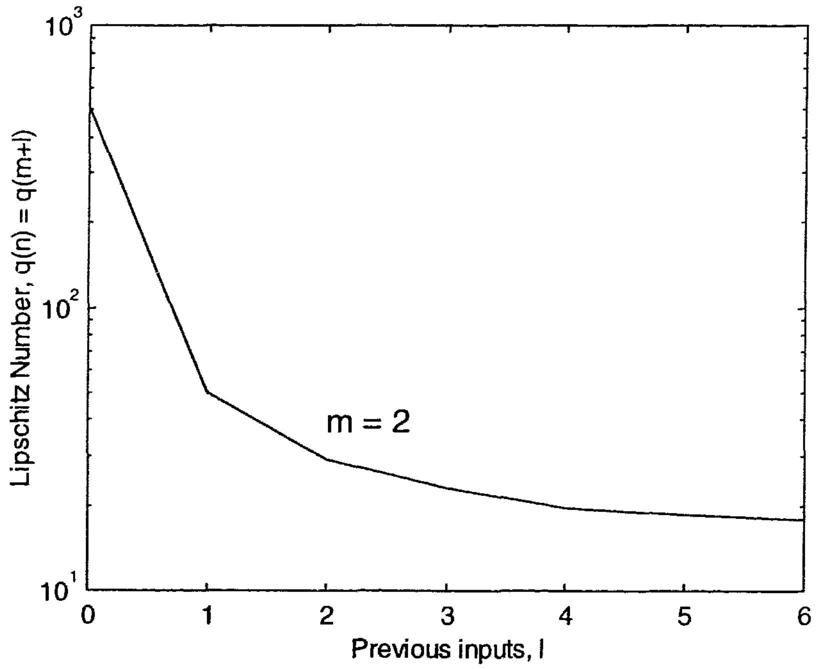

Figure 6.2 Plot of the lag space for input-output data (m = 2)..

The m=2/l=3 lag structure was found to yield the best results of the three combinations identified.

6.3 Determination of the Number of Hidden PEs

The number of hidden PEs and size of the training file was constrained to less than 16 PEs and 2500 samples, respectively, due to computer memory limitations. In addition, it was discovered that a smaller training file resulted in a model that could better predict peak values. Thus, a training file of 500 samples was used for optimizing the network for both hidden PEs and number of training iterations. The ramifications of such a small training set will be discussed further in Chapter 7.0.

Figure 6.3 shows the RMS error on the test set for different numbers of hidden PEs. The error was calculated using the model-predicted output. The graph indicates that three PEs should be included in the hidden layer.

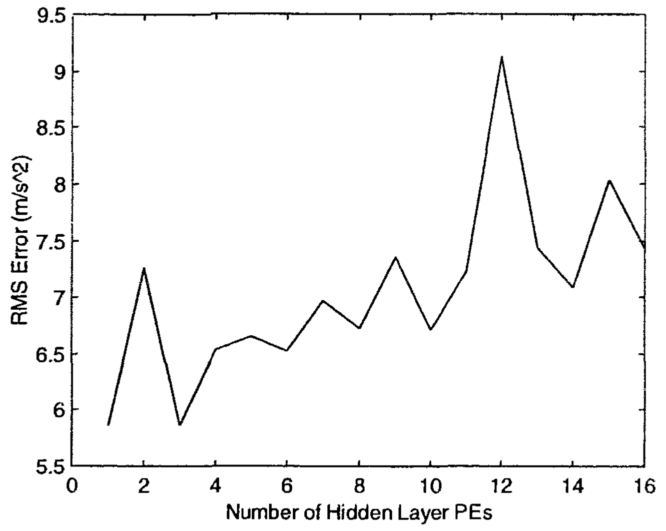

Figure 6.3 RMS error vs hidden PEs.

Figure 6.4 was used to indicate the optimal number of training iterations for the network. We see that the RMS error tends to decrease for the first 30 iterations, after which network learning slows considerably. According to the notion of Early Stopping (Cohn, 1993), any additional decrease in the RMS error is likely due to the network learning the stochastic variations in the training set data, as opposed to learning the underlying system. Thus, in order to avoid overtraining, the number of iterations was limited to 30.

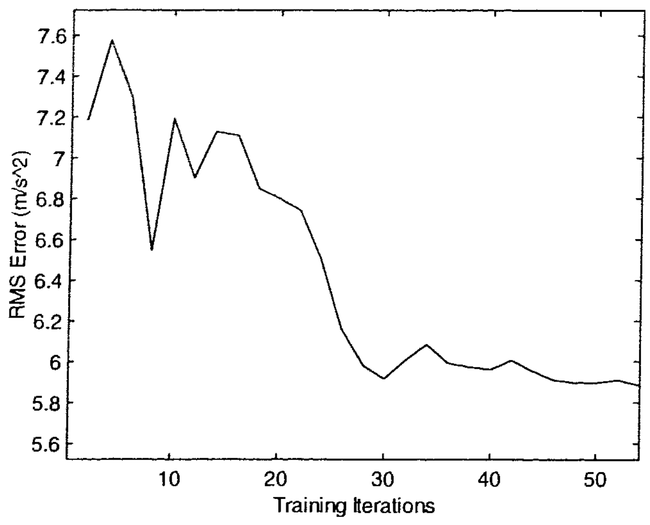

Figure 6.4 RMS Error vs. training iterations.

6.4 Final Model Structure

The optimized model structure is shown below in Figure 6.5. This final model can be represented as the nonlinear difference equation

$$y_p(t) = P[y_p(t-1), y_p(t-2), u(t-1), u(t-2), u(t-3)]. \qquad (6.1)$$

The input layer consists of 5 unity-gain buffers which serve only to distribute the inputs to the hidden layer PEs. The hidden layer PEs utilize a hyperbolic tangent activation function which enable the network to approximate nonlinear functions. However, the output PE utilizes a linear activation function, such that the output is a linear summation of its weighted inputs. A linear PE provides a greater dynamic range for the output than one which utilizes a hyperbolic tangent activation function. The latter is limited to an output range of [-1,1].

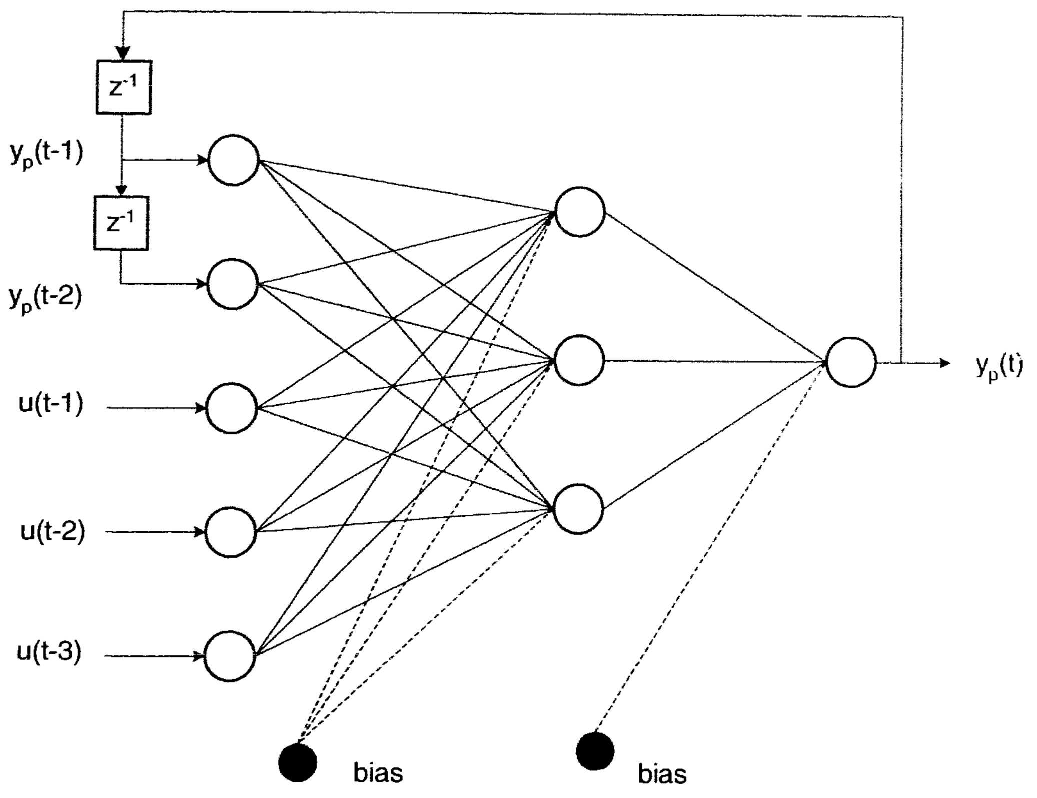

Figure 6.5 Finalized neural network structure.

The hidden and output layers also include a bias unit which adds a constant value to each PE. The inclusion of the bias removes the constraint that the network's approximation of the system pass through the origin. The bias input to each PE is determined through the training process along with the network weights.

6.5 Model Validation

The following plots show the model predicted output (dashed line)compared with the actual lumbar-4 acceleration (solid line) for the test data set.

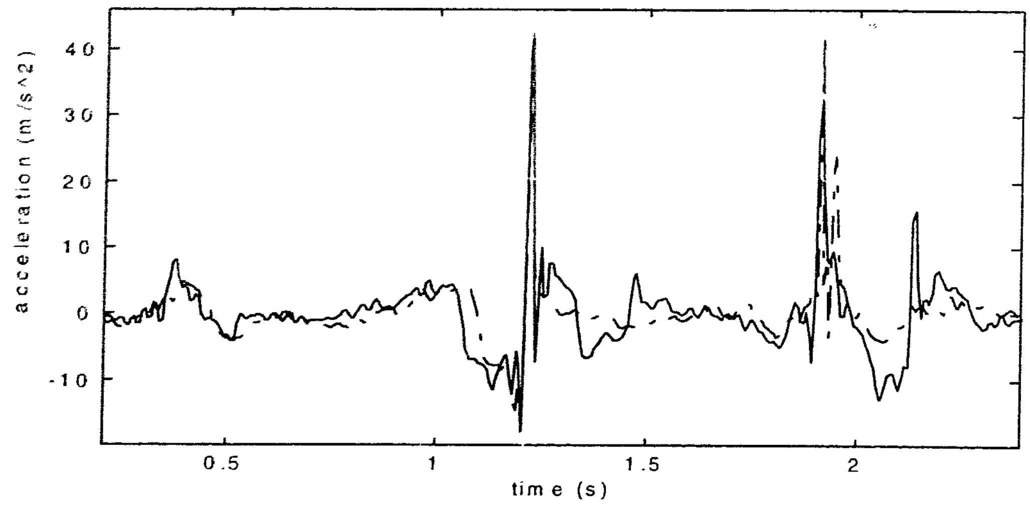

Figure 6.6 Response to 1g/5Hz, -2g/5Hz, 3g/5Hz input shocks.

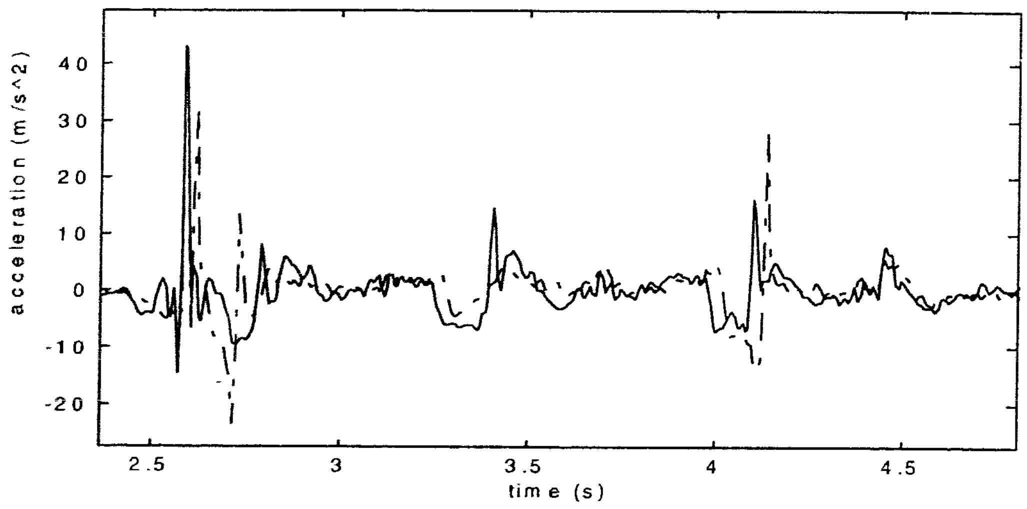

Figure 6.7 Response to 4g/11Hz, -1g/3Hz, -2g/6Hz, 1g/11Hz input shocks.

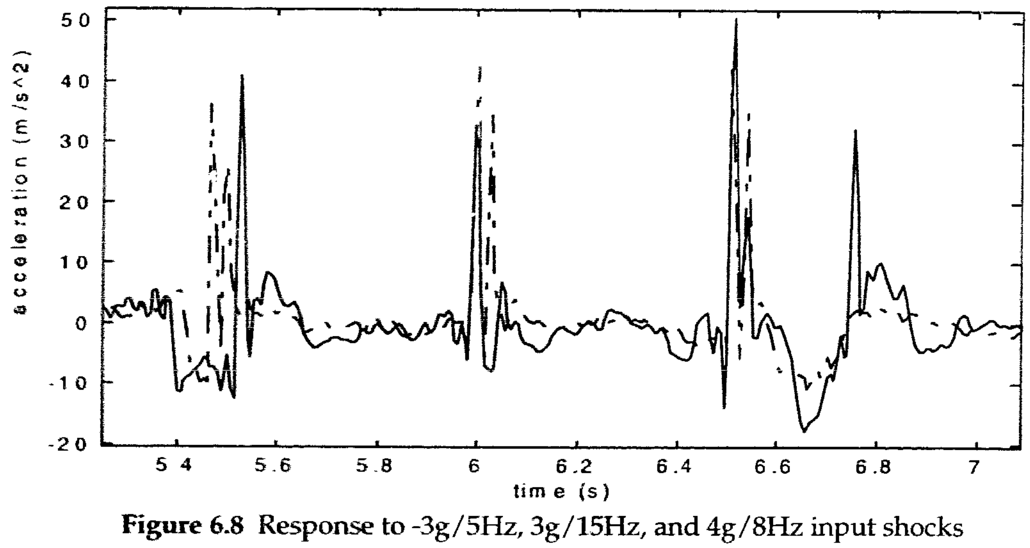

Figure 6.8 Response to -3g/5Hz, 3g/15Hz, and 4g/8Hz input shocks

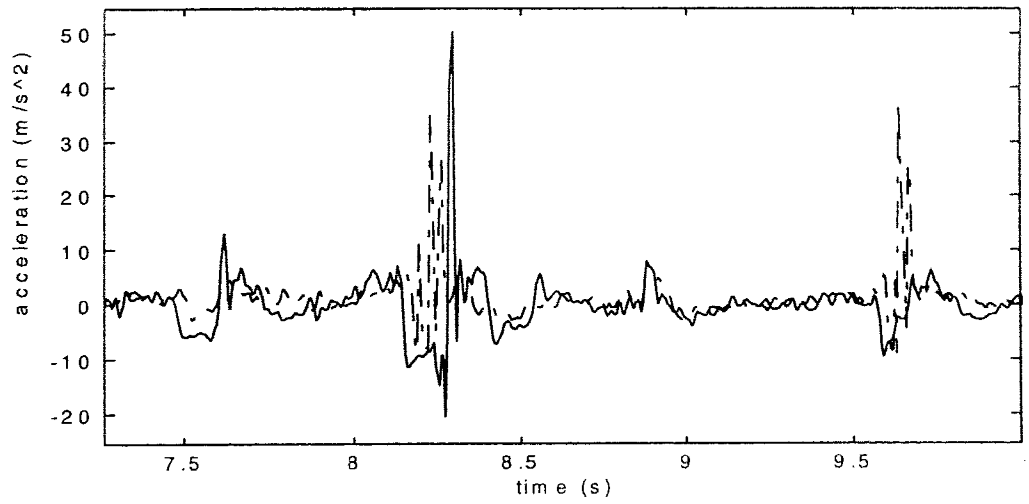

Figure 6.9 Response to -1g/4Hz, -4g/8Hz, 1g/6Hz, and -3g/15Hz input shocks.

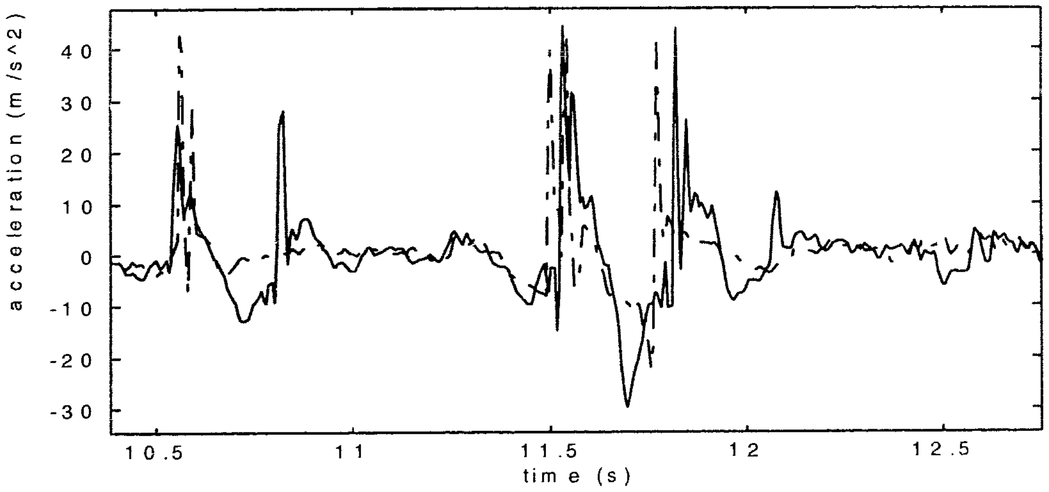

Figure 6.10 Response to 2g/5Hz, 4g/4Hz, -1g/6Hz input shocks.

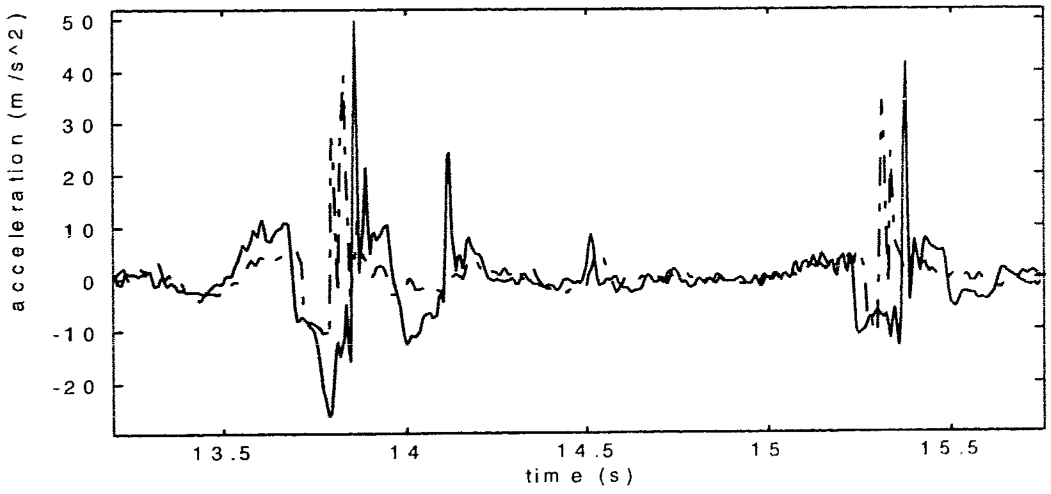

Figure 6.11 Response to.-3g/6Hz, 1g/11Hz, and -2g/8Hz input shocks.

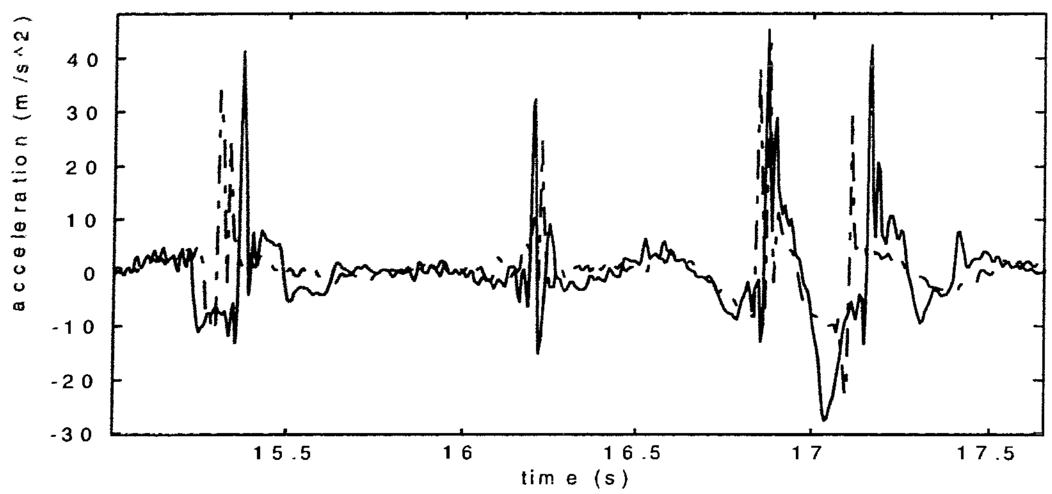

Figure 6.12 Response to -2g/8Hz, 3g/20Hz, and 3g/4Hz input shocks.

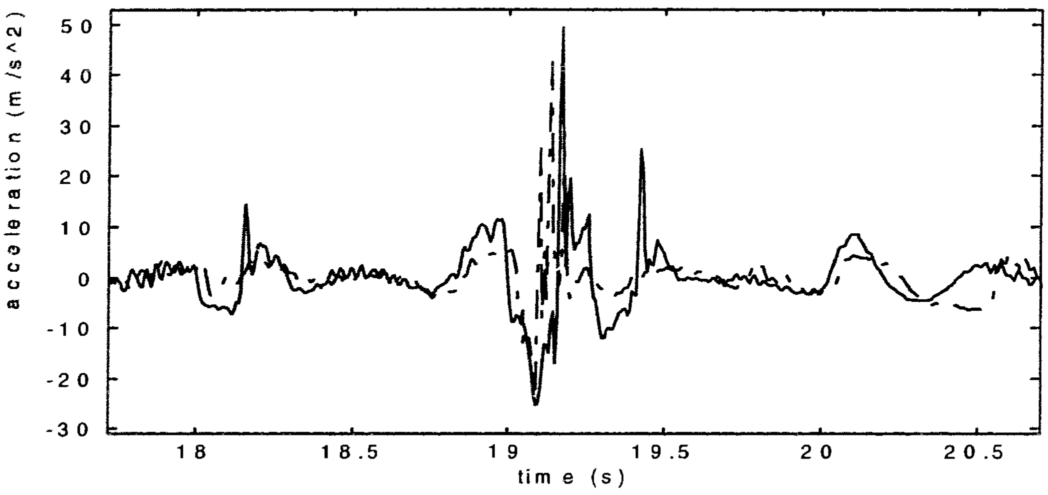

Figure 6.13 Response to -1g/4Hz, -4g/5Hz, and 1g/2Hz input shocks.

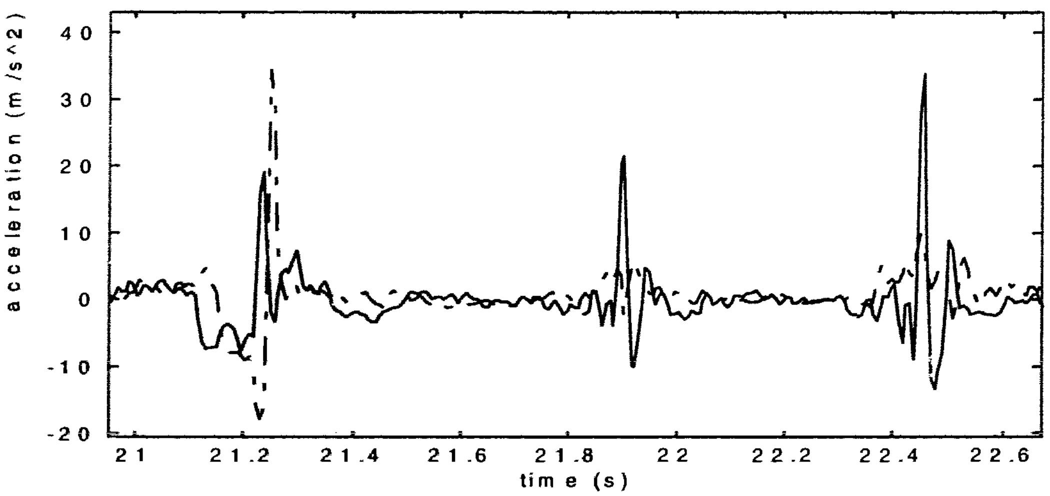

Figure 6.14 Response to -2g/11Hz, 2g/20Hz, and 3g/15Hz input shocks.

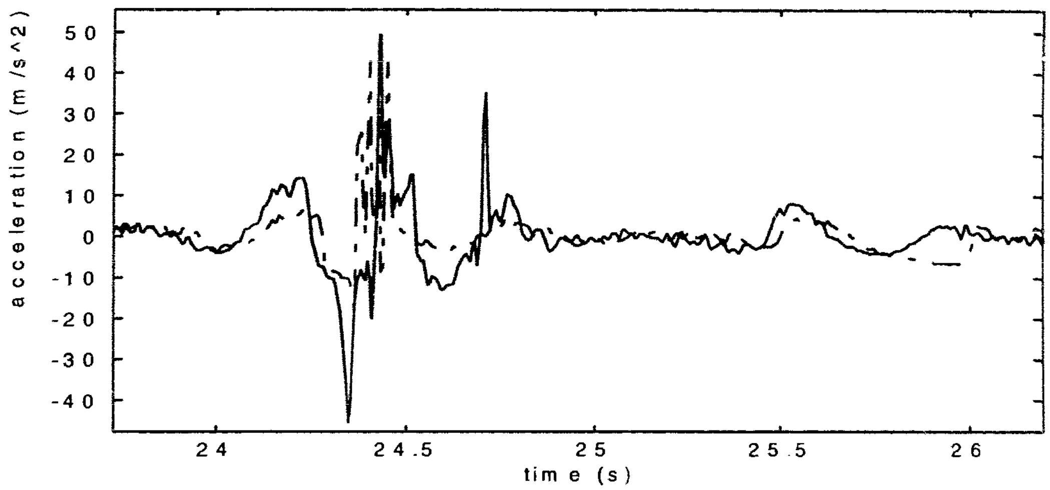

Figure 6.15 Response to.-3g/4Hz, and 1g/2Hz input shocks.

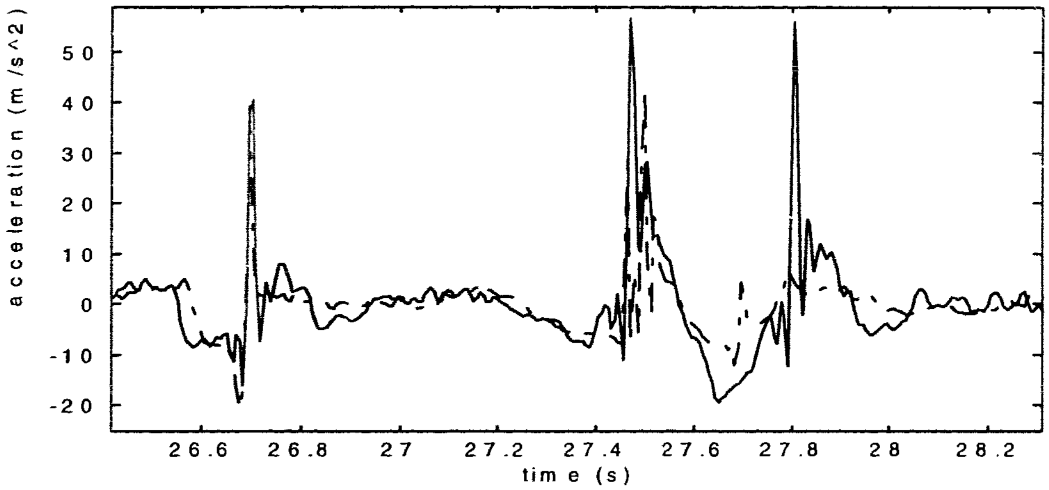

Figure 6.16 Response to -2g/6Hz, and 3g/4Hz input shocks.

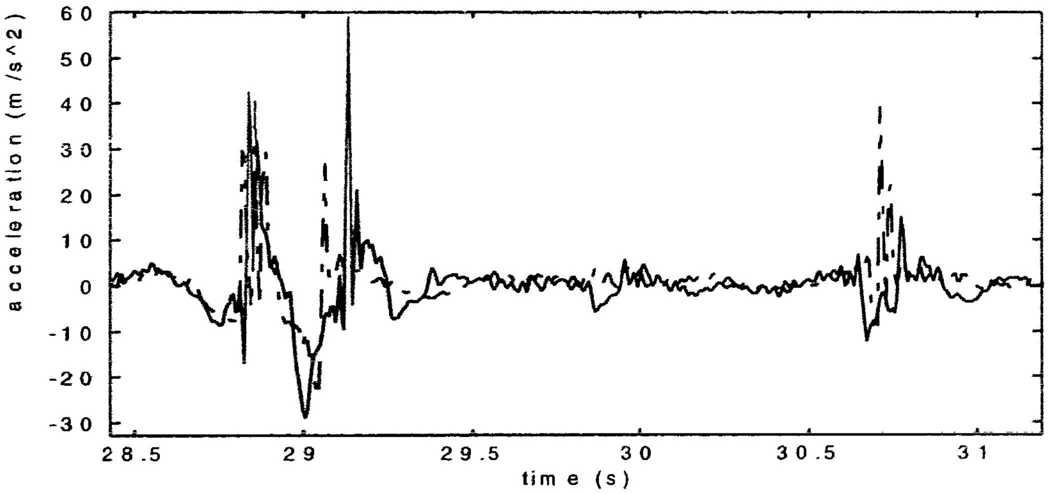

Figure 6.17 Response to 4g/4Hz, -1g/8Hz and -4g/12Hz. input shocks.

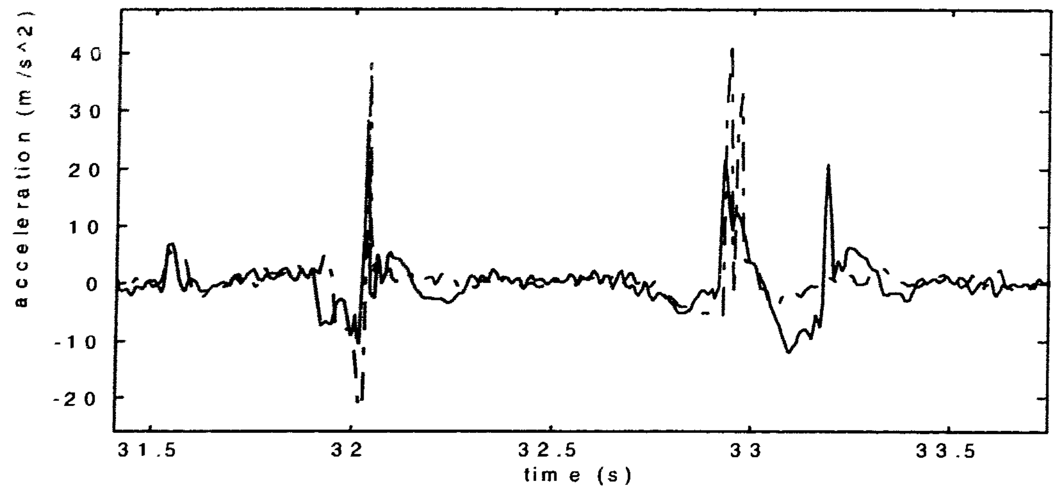

Figure 6.18. Response to 1g/11Hz, -2g/8Hz, and 2g/4Hz input shocks.

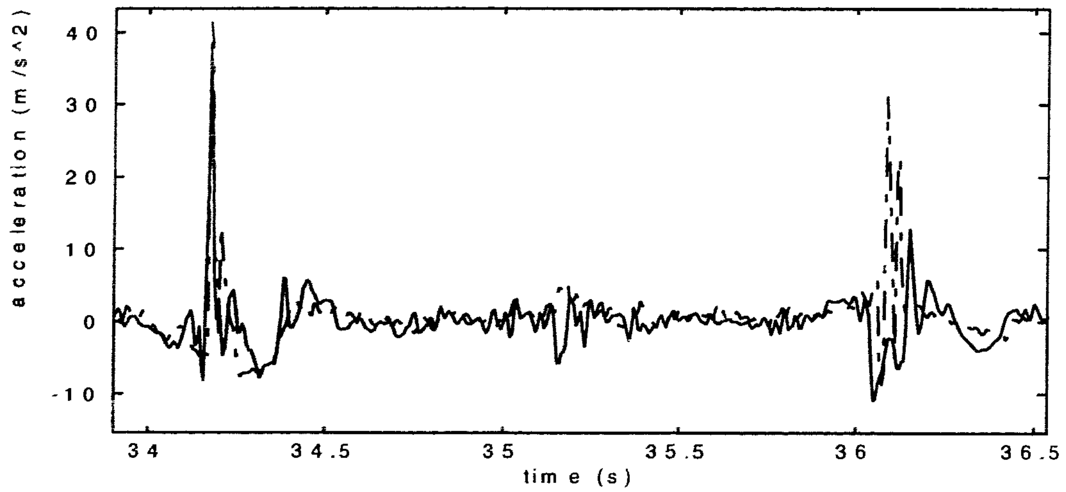

Figure 6.19 Response to 4g/11Hz, -1g/15Hz, and -4g/8Hz input shocks.

The plots in Figures 6.6 to 6.19 seem to indicate a fairly good fit between the model's output and the measured lumbar-4 acceleration. In the majority of cases, the model's response has a similar shape although the peak amplitudes and timing are not exactly matched. The model exhibits the worst performance on the following shocks (listed in order of appearance in Figures 6.6-6.19): -1g/3Hz, -1g/4Hz, -4g/8Hz, -3g/15Hz, -1g/15Hz, 2g/20Hz, 3g/15Hz, -1g/8Hz, -4g/11Hz, and -1g/20Hz shocks. Thus, the model seems to perform poorly on low amplitude negative shocks, high frequency negative shocks, and high frequency, high amplitude positive shocks. On some of these shocks (eg. -4g/8Hz from Figure 6.19) the model overestimates the response. In others (e.g. -2g/20Hz from Figure 6.14) the model underestimates or misses the peak response.

6.6 Model Comparisons

This section shows a comparison between the recurrent neural network model, the Dynamic Response Index model, the British Standard 6841 filter, and a 20'th order ARX model. The implementation details of the DRI and BS 6841 models are provided in Appendix E. The input for all four models consisted of the seat acceleration of the test data set, sampled at 150Hz. The models are compared on the basis of the RMS error (Table 6.3) and visual inspection of the model predicted output (dashed line) plotted against the actual output (solid line) in Figures 6.20-6.38.

Table 6.3 Prediction errors for the four model types.

Model Type	RMS Error
20'th order ARX	4.90
5-3-1 neural network	6.62
BS 6841 filter	7.66
DRI	9.33

6.6.1 RNN Model

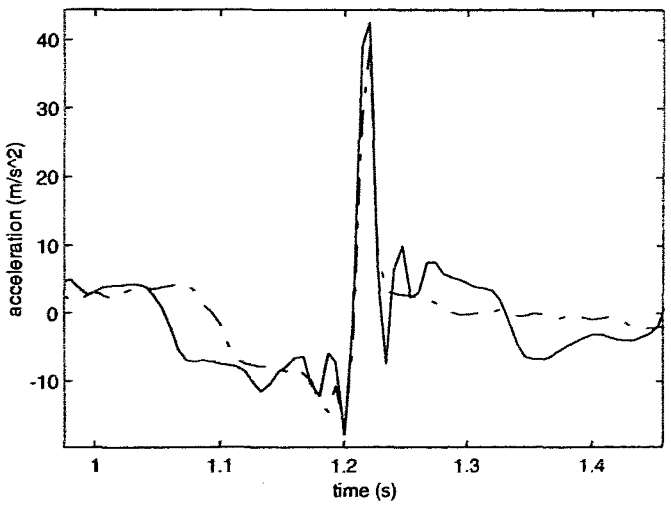

Figure 6.20 RNN (dashed) and measured (solid) responses to a -2g/5Hz input shock.

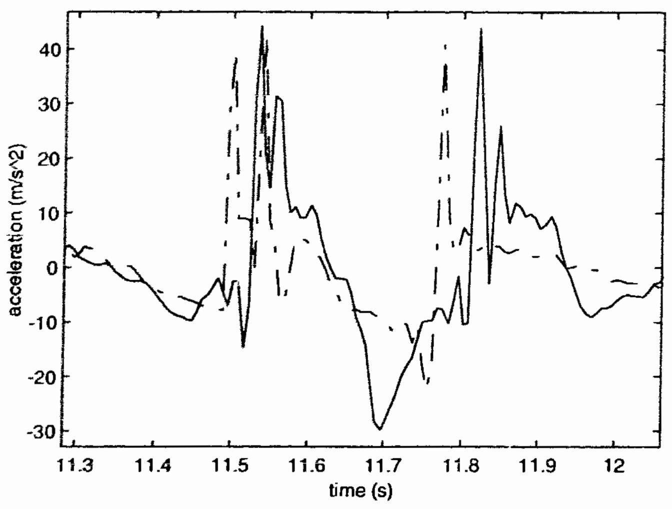

Figure 6.21 RNN (dashed) and measured (solid) responses to a 4g/4Hz input shock.

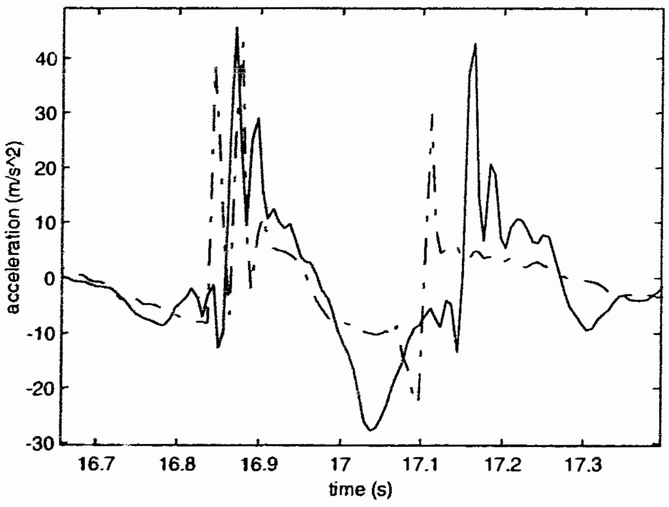

Figure 6.22 RNN (dashed) and measured (solid) responses to a 3g/4Hz input shock.

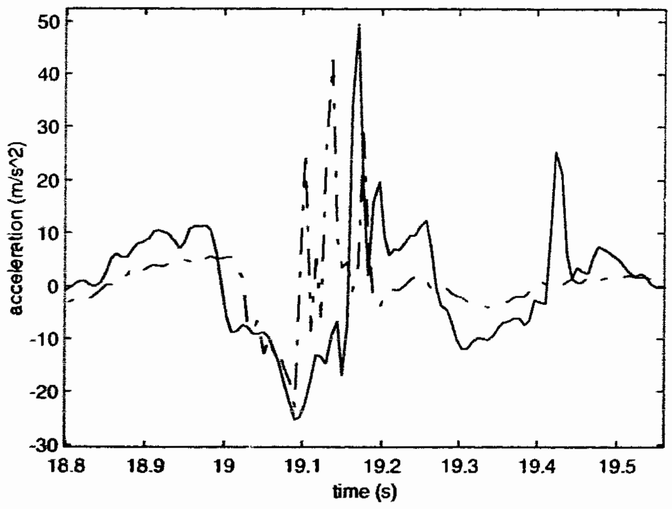

Figure 6.23 RNN (dashed) and measured (solid) responses to a -4g/5Hz input shock.

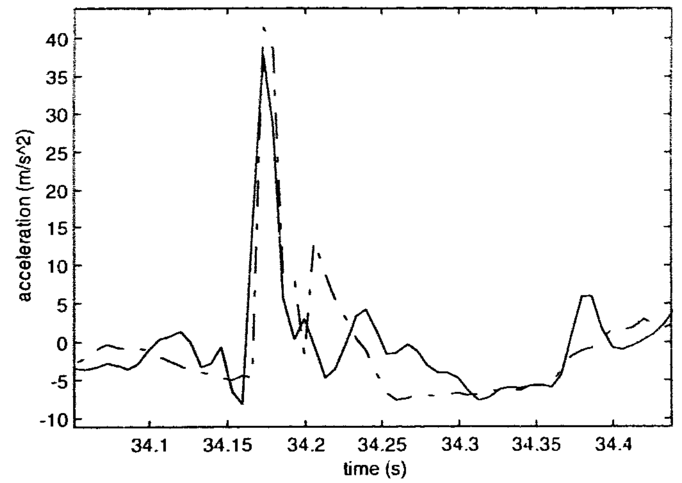

Figure 6.24 RNN (dashed) and measured (solid) responses to a 4g/11Hz input shock.

6.6.2 The Dynamic Response Index model

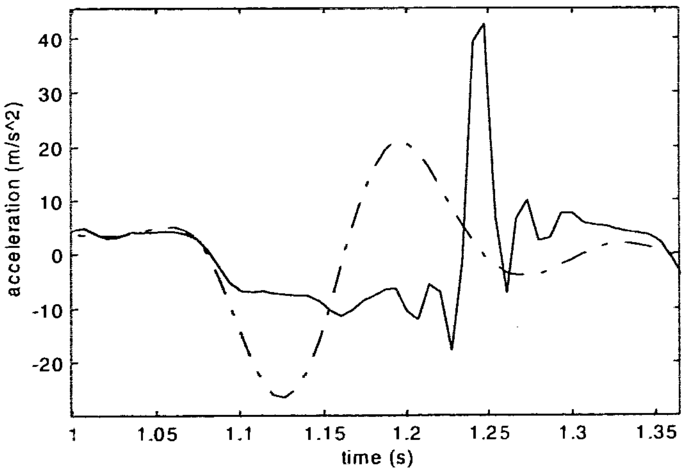

Figure 6.25 DRI (dashed) and measured (solid) responses to a -2g/5Hz input shock.

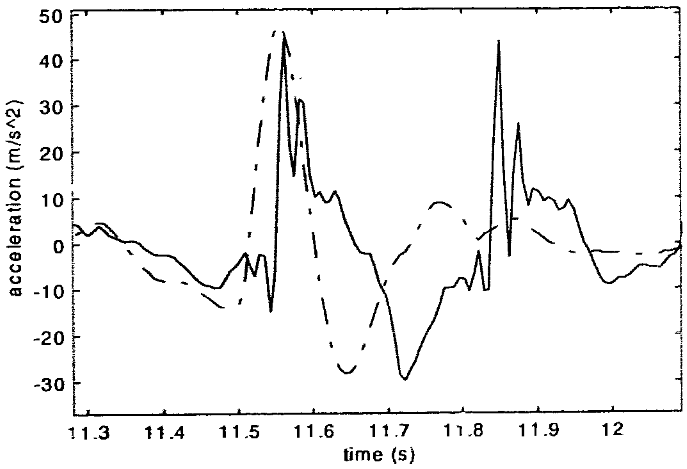

Figure 6.26 DRI (dashed) and measured (solid) responses to a 4g/4Hz input shock.

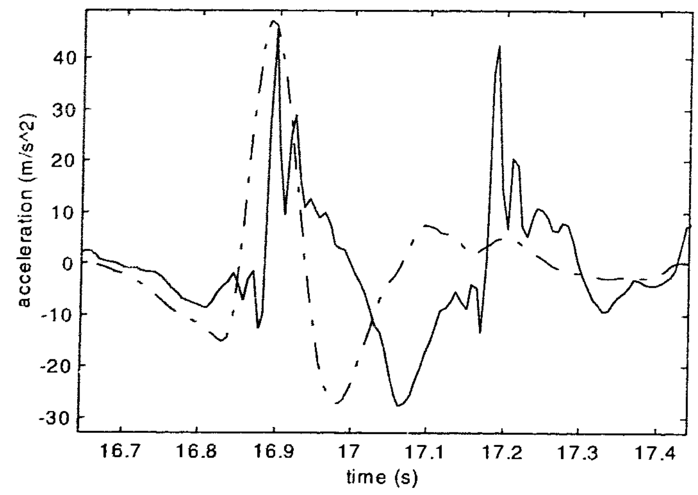

Figure 6.27 DRI (dashed) and measured (solid) responses to a 3g/4Hz input shock.

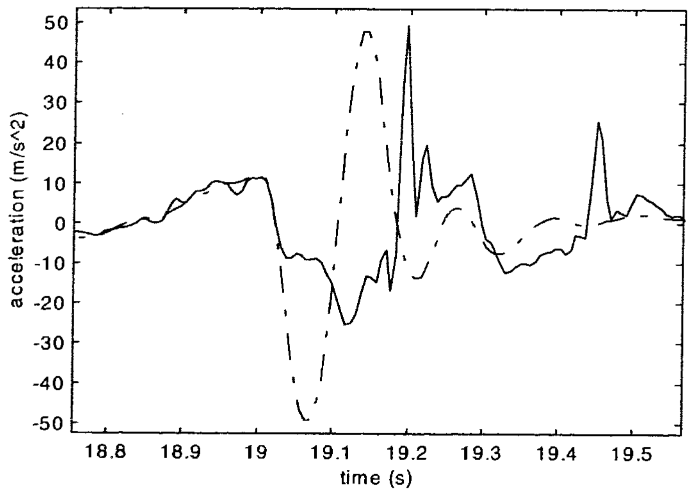

Figure 6.28 DRI (dashed) and measured (solid) responses to a -4g/5Hz input shock.

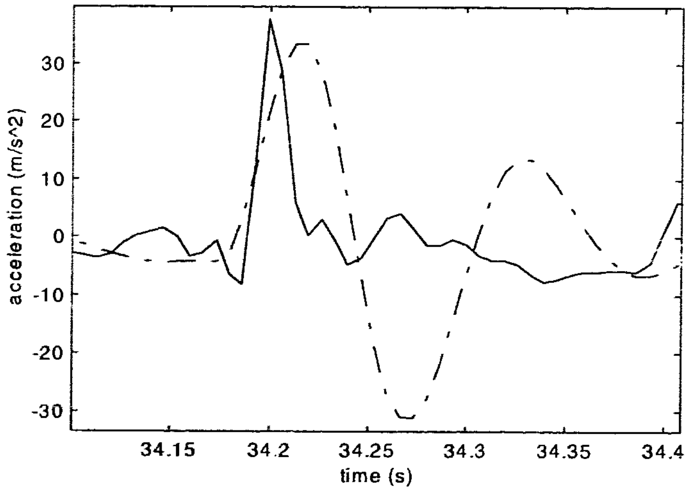

Figure 6.29 DRI (dashed) and measured (solid) responses to a 4g/11Hz input shock.

6.6.3 The British Standard 6841 Filter

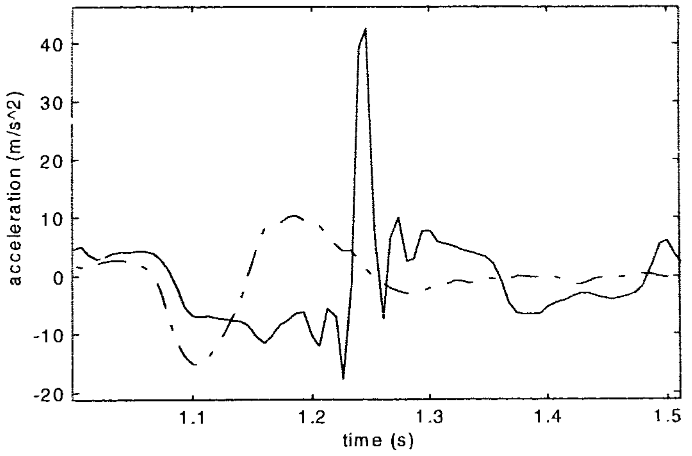

Figure 6.30 BS 6781 filter (dashed) and measured (solid) responses to a -2g/5Hz input shock.

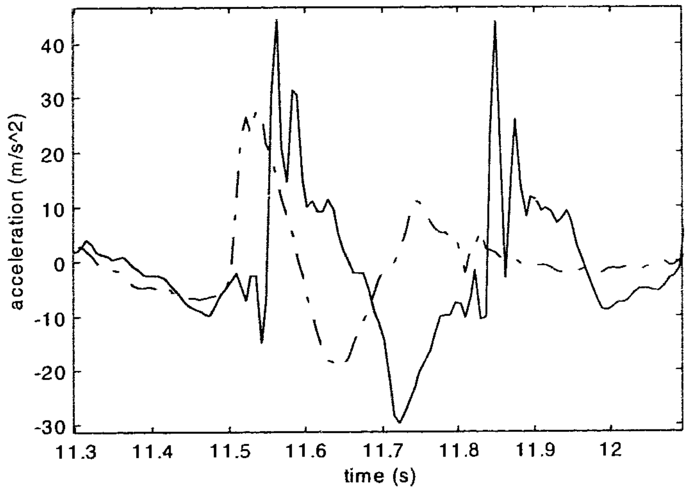

Figure 6.31 BS 6781 filter (dashed) and measured (solid) responses to a 4g/4Hz input shock.

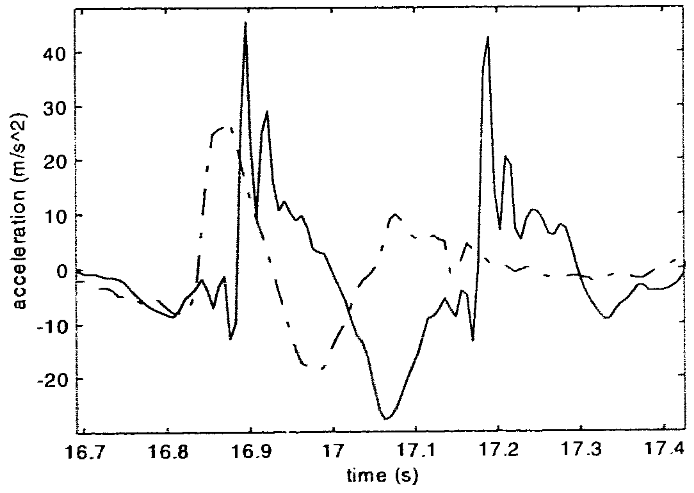

Figure 6.32 BS 6781 filter (dashed) and measured (solid) responses to a 3g/4Hz input shock.

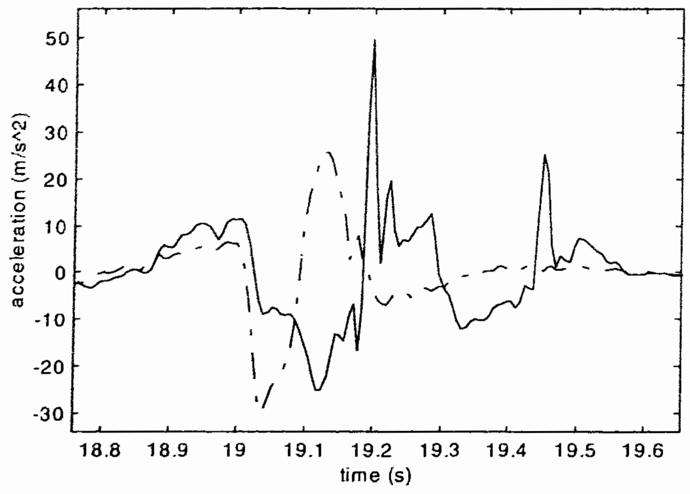

Figure 6.33 BS 6781 filter (dashed) and measured (solid) responses to a -4g/5Hz input shock.

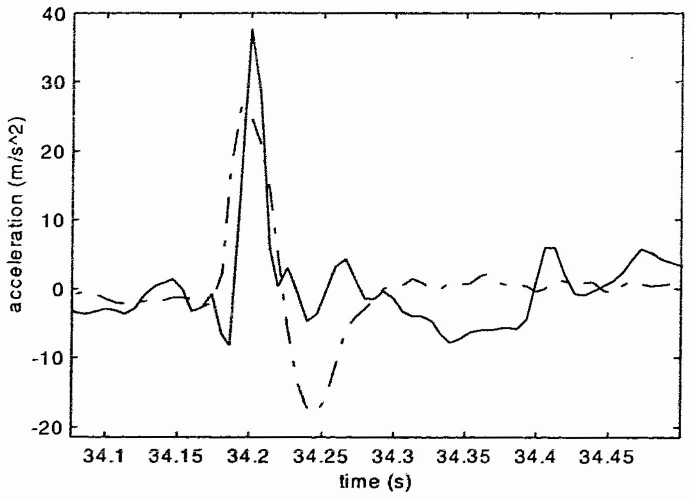

Figure 6.34 BS 6781 filter (dashed) and measured (solid) responses to a 4g/11Hz input shock.

6.6.4 ARX Model

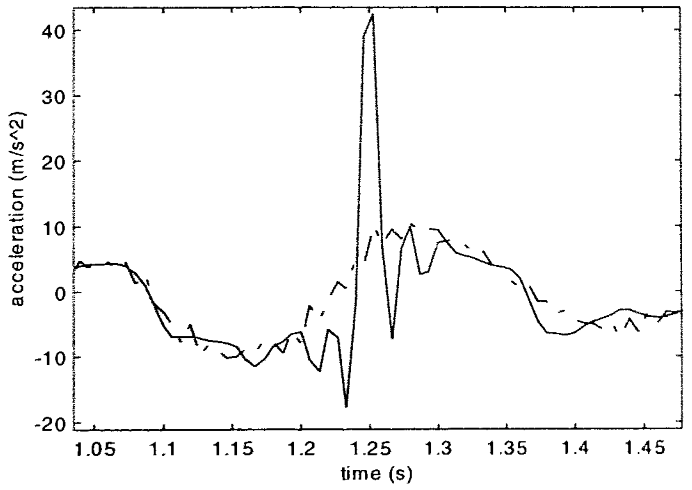

Figure 6.35 ARX (dashed) and measured (solid) responses to a -2g/5Hz input shock.

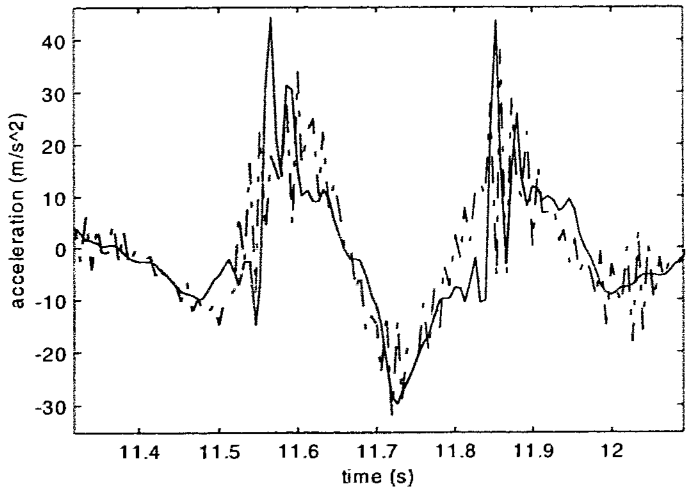

Figure 6.36 ARX (dashed) and measured (solid) responses to a 4g/4Hz input shock.

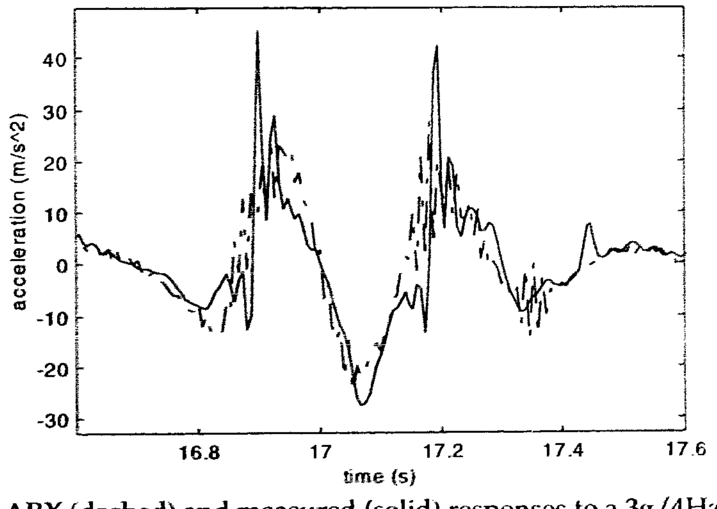

Figure 6.37 ARX (dashed) and measured (solid) responses to a 3g/4Hz input shock.

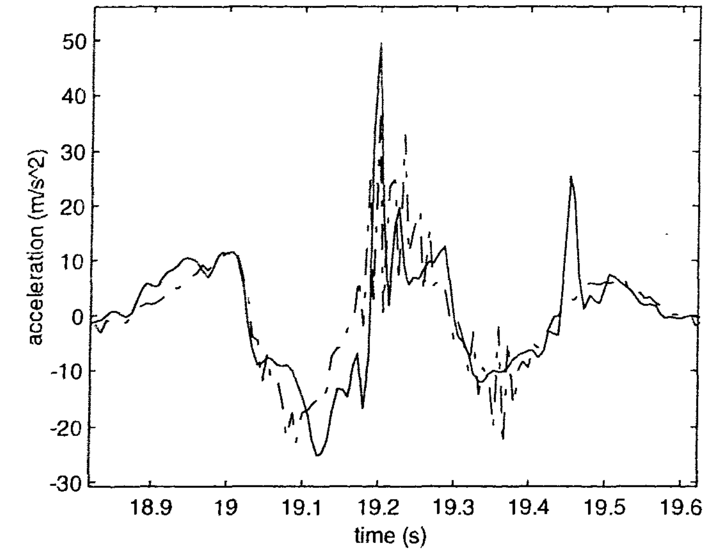

Figure 6.38 ARX (dashed) and measured (solid) responses to a -4g/5Hz input shock.

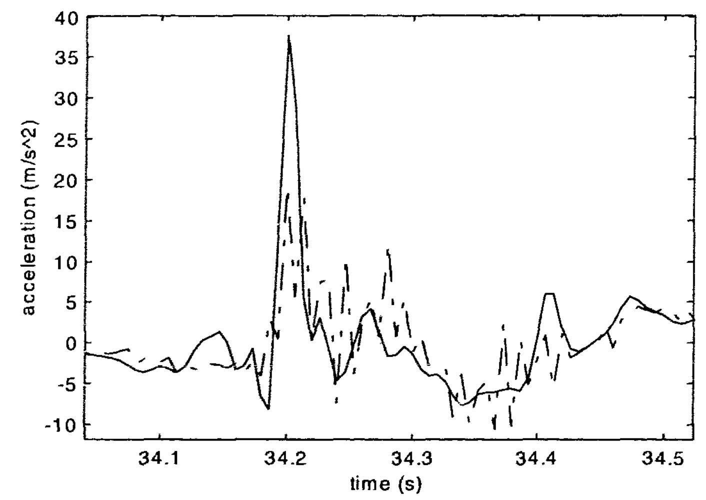

Figure 6.39 ARX (dashed) and measured (solid) responses to a 4g/11Hz input shock.

Based on visual inspection of the plots in Figures 6.20 to 6.39, the neural network clearly outperforms the other three models for predicting the response to the seat shocks examined. The DRI model, which is optimized for positive input shocks predicts the peak value of the initial L4 response fairly well. The shape of the DRI response is typical of a second order, underdamped linear model and, hence, does not reflect the more complex shape of the measured data. Moreover, this model fails to predict the correct peak and shape for negative input shocks as indicated by Figure 6.25. Finally, unlike the RNN model, the DRI is unable to predict the secondary peak of the response as seen in Figure 6.26 and 6.27.

The BS 6841 filter performs even worse than the DRI in response to all the shocks examined. It fails to predict the correct shape and peak values. This result is not surprising in that this model was optimized to predict the response to low amplitude vibration.

The ARX model predicts the overall shape of the response fairly accurately in Figures 6.36, 6.37, and 6.38. Like the RNN model it is able to predict the secondary peak in Figures 6.36 and 6.37, but not as accurately. In terms of the RMS error, this model exhibits the best performance. However, it performs dismally on predicting the response in Figures 6.35 and 6.39. In addition, the model seems to add high frequency noise to the output.

6.7 Cross-Subject Validation

The prediction errors for the four models applied to a different subject (No. 7) are shown in Table 6.4. The error values here cannot be compared directly with those presented in Table 6.3 since the number and type of shocks in the test set for the two subjects are slightly different. Furthermore, the amount of vibration compared with shocks is not exactly the same either, and thus, the performance on predicting the shocks is weighted differently for the two subjects. However, the relative order of the model predictions is the same in both tables. For both subjects the neural network model outperforms the BS 6841 filter and the DRI model.

Table 6.4 Prediction errors for the four model types for Subject 7.

Model Type	RMS Error
20'th order ARX	2.91
5-3-1 neural network	3.25
BS 6841 filter	4.67
DRI	5.49

Figures 6.40 to 6.45 shows the neural network prediction (dashed line) on individual shocks compared with the measured data (solid line). These plots correspond with Figures 6.20 to 6.24, so that the model's response to the same type of shocks can be compared visually for the two subjects. As seen in the following plots, the neural network model appears to be capable of predicting the overall shape of the unseen subject's response. This result seems to indicate that the neural network is able to generalize and is not subject specific.

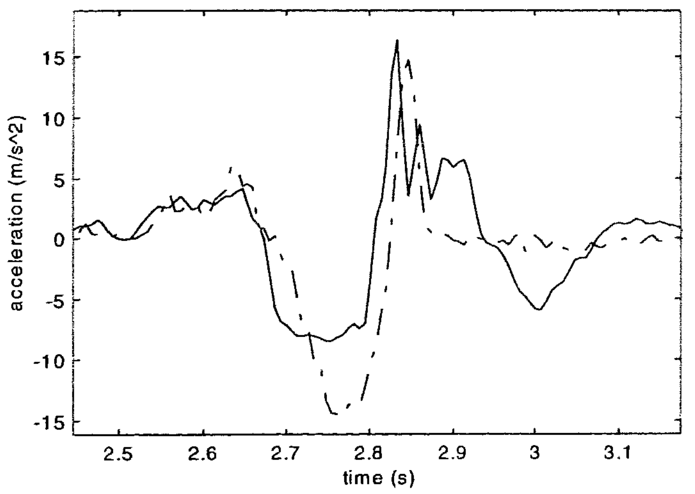

Figure 6.40 Neural network (dashed) and measured (solid) response to a -2g/5Hz shock.

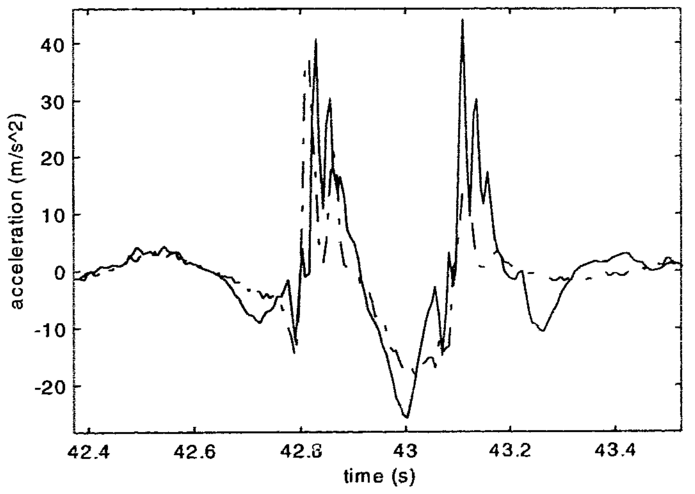

Figure 6.41 Neural network (dashed) and measured (solid) response to a 4g/4Hz shock.

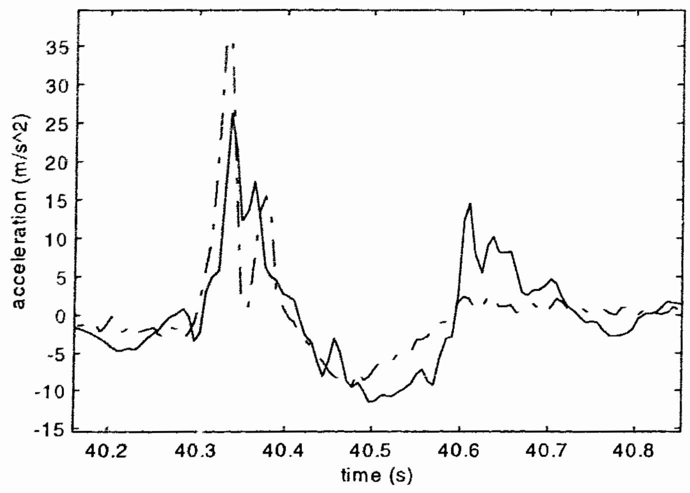

Figure 6.42 Neural network (dashed) and measured (solid) response to a 3g/4 Hz shock.

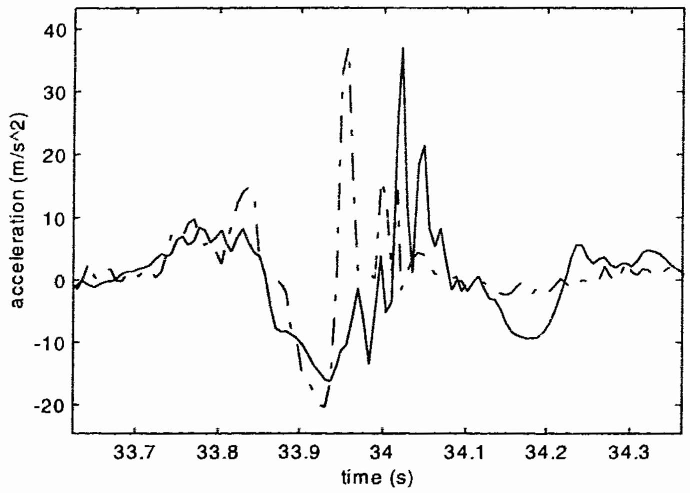

Figure 6.43 Neural network (dashed) and measured (solid) response to a -4g/5Hz shock.

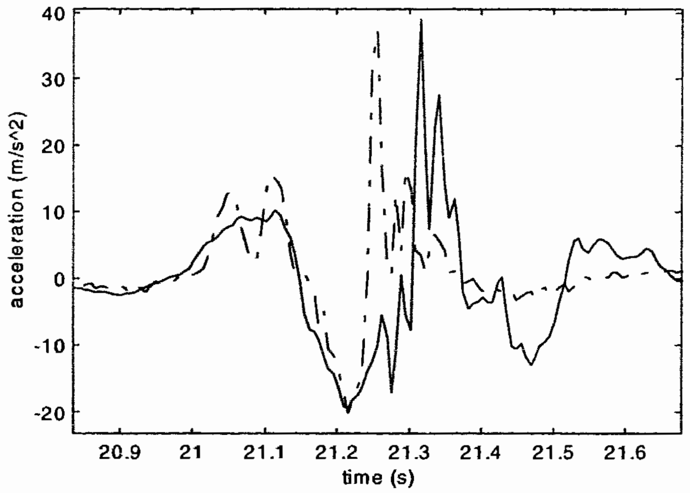

Figure 6.44 Neural network (dashed) and measured (solid) response to a -3g/4Hz shock.

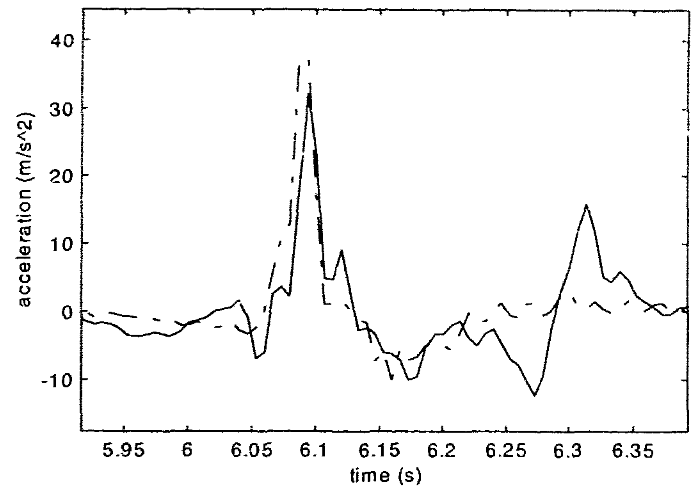

Figure 6.45 Neural network (dashed) and measured (solid) response to a 4g/11Hz shock.

6.7.1 Cross-Subject Validation of the Model's Response to Vibration

In order to test the neural network's ability to model the spinal response to low-level seat vibration, a number of comparisons were made with the three other models examined. The data used for these comparisons consisted of 5.8 seconds of seat vibration and the corresponding lumbar-4 response of the unseen subject (No. 7).

The RMS prediction error for all four models are listed in Table 6.5. For the purposes of visual comparison a 1 second window of the models' outputs are plotted against the measured data in Figures 6.46-6.49. The difference in the RMS prediction errors between the four models is small and perhaps does not accurately reflect the performance of the models. For example, the neural network model output appears to match the measured response more closely than the DRI which contains large amplitude oscillations. Yet on the basis of the RMS error, these two models perform equivalently.

Table 6.5 Time-domain prediction errors for the four model types for Subject 7 in response to vibration.

Model Type	RMS Error
BS 6841 filter	0.742
DRI model	0.763
5-3-1 Neural network	0.764
ARX model	0.789

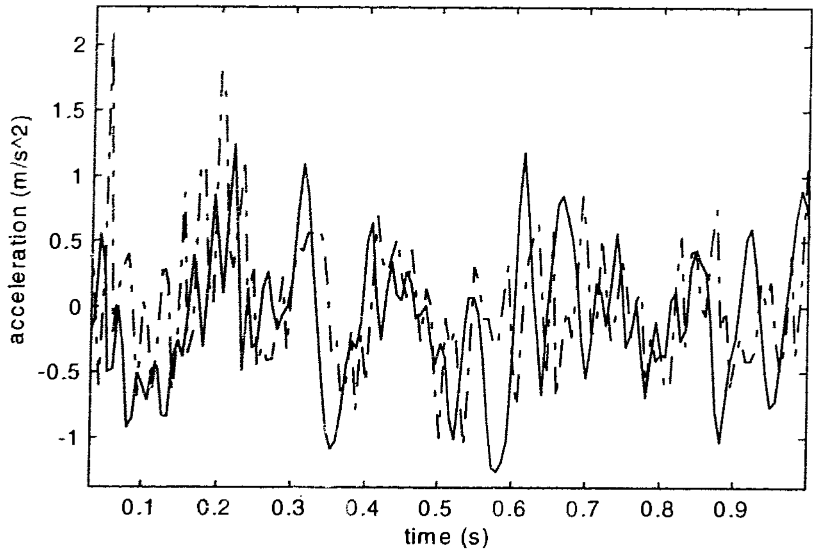

Figure 6.46 Neural network (dashed) and measured (solid) response to background vibration with an RMS value of 0.05g.

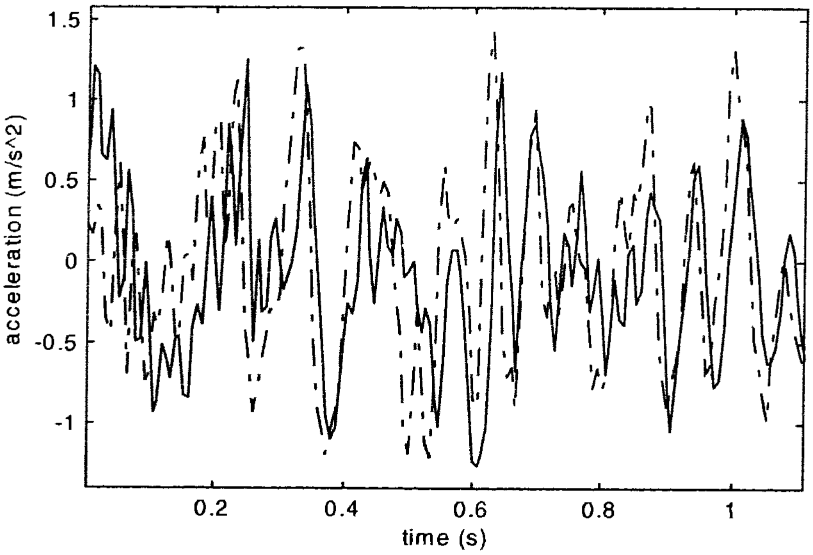

Figure 6.47 BS 6841 filter (dashed) and measured (solid) response to background vibration with an RMS value of 0.05g.

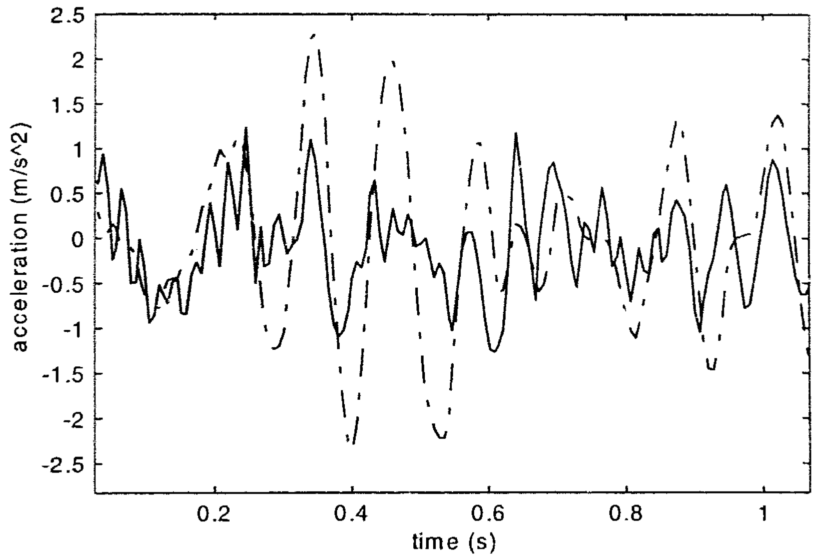

Figure 6.48 DRI model (dashed) and measured (solid) response to background vibration with an RMS value of 0.05g.

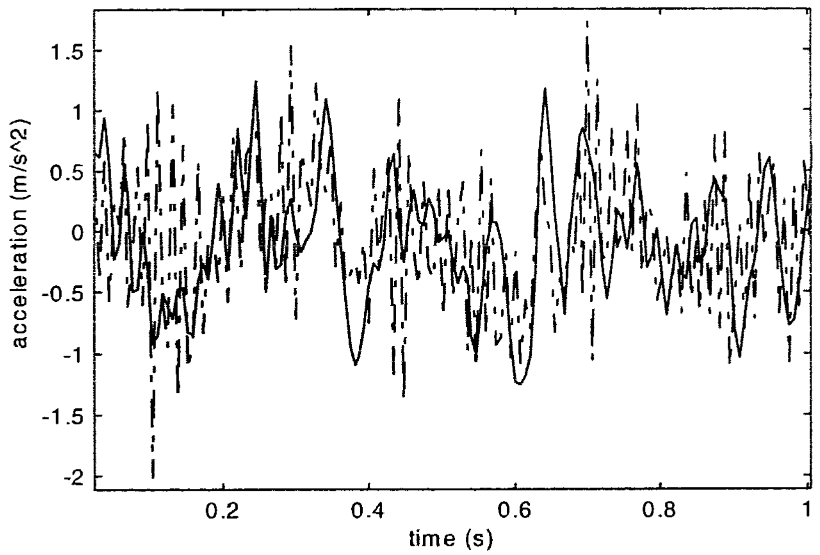

Figure 6.49 20'th order ARX model (dashed) and measured (solid) response to background vibration with an RMS value of 0.05g.

A frequency-domain analysis was performed by estimating the power spectral density of the models' outputs. The Welch's periodogram method was used to estimate the power spectral density (PSD)of each signal. This method used a Fast Fourier Transform size of 256, and a Hanning window with 50% overlap between successive windows. The PSD plots are shown in Figures 6.50-6.53. The RMS error between the PSD of the measured and that of each model's output is listed in Table 6.6.

The PSD of the neural network response compares favorably with the other models examined. The neural network seems to underestimate the low-frequency (0-20Hz) response while overestimating the higher frequency (30-40Hz) response. However, the overall shape of the PSD is fairly accurate. In contrast, the PSD of the BS 6841 model appears to match the shape of the measured PSD accurately, but tends to overestimate the amplitude consistently throughout the spectrum. The PSD RMS errors of the neural network and BS 6841 filter are both considerably smaller than those of the DRI and the ARX model.

The poor RMS error performance of the DRI and ARX models is reflected in their respective PSD plots. The DRI model greatly overestimates the amount of energy contained in the lower frequency range (0-15Hz), while underestimating the energy of signals above 15Hz. The ARX model, on the other hand, matches the overall pattern of the measured PSD fairly well except that it exhibits a large response above 55Hz.

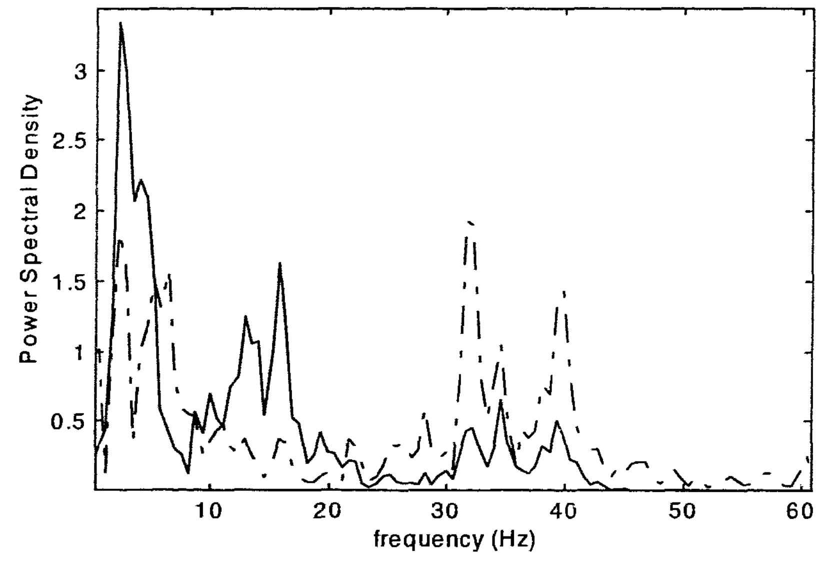

Figure 6.50 Power spectral density estimate of neural network (dashed) and measured (solid) response to background vibration with an RMS value of 0.5g.

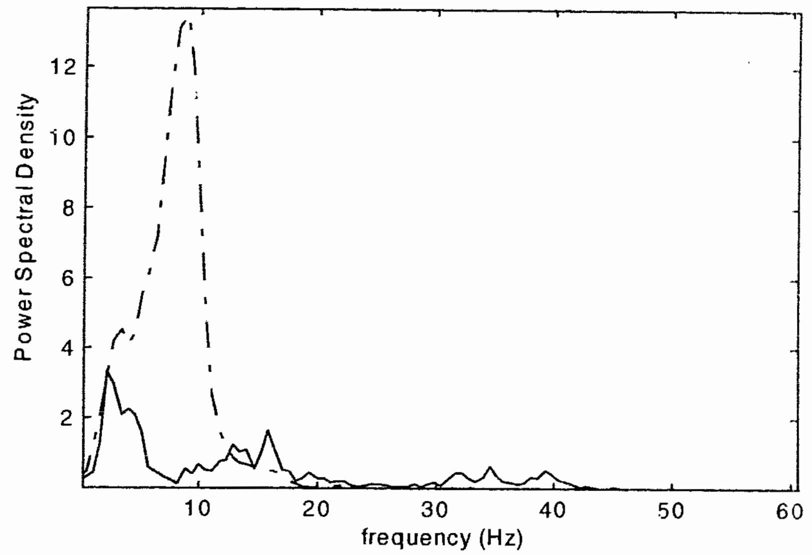

Figure 6.51 Power spectral density estimate of DRI model (dashed) and measured (solid) response to background vibration with an RMS value of 0.5g.

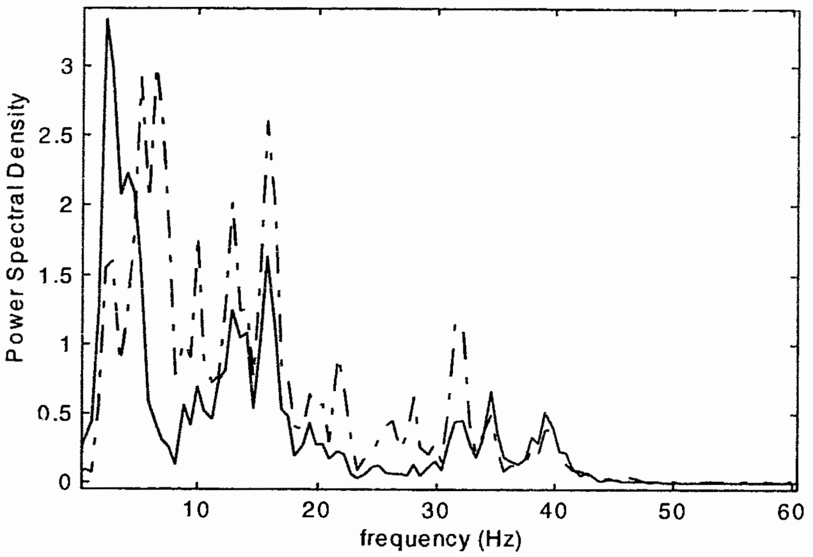

Figure 6.52 Power spectral density estimate of BS 6841 (dashed) and measured (solid) response to background vibration with an RMS value of 0.5g.

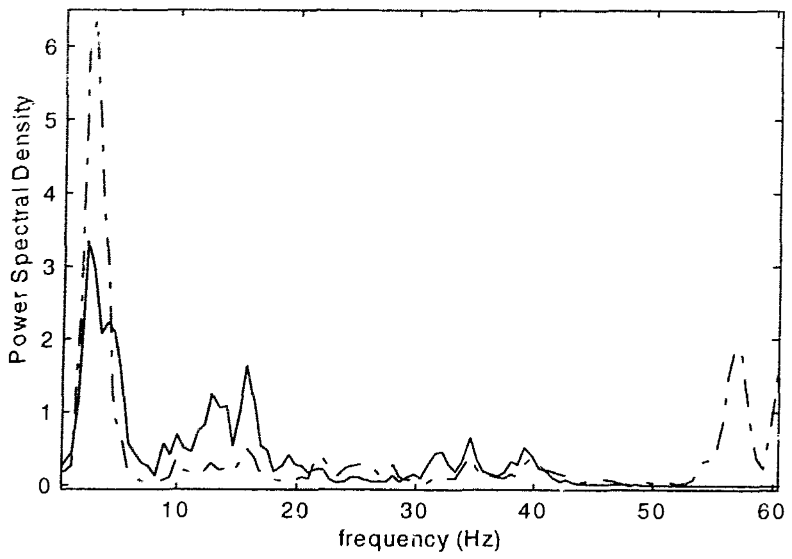

Figure 6.53 Power spectral density estimate of 20'th order ARX model (dashed) and measured (solid) response to background vibration with an RMS value of 0.5g.

Table 6.6 Frequency-domain prediction errors for the four model types for Subject 7 in response to vibration.

Model Type	RMS Error
5-3-1 neural network	0.473
BS 6841 filter	0.520
ARX model	1.08
DRI model	2.613

CHAPTER 7 DISCUSSION

The performance of the neural network relative to other models depends on the criteria used for evaluation. When the root mean squared error is used, the neural network prediction on the test set is second to that of the 20'th order ARX model, as shown in Table 6.3. However, the RMS error can be misleading for a number of reasons. For one, this error measure is heavily weighted by the performance of the model on the background vibration which constitutes the majority of the signal. Thus, a model that predicts the vibration response well will exhibit a low RMS error regardless of the predicted shock response. It is, therefore, not surprising that the ARX model performs well according to this criterion as it represents a linear least squares fit of the data. The neural network represents a nonlinear least squares fit to the data, but it was optimized for shock response prediction.

The second misleading aspect of the RMS error criterion is that it does not take into account time shifts between the predicted and measured response. For example, in Figure 6.10 the neural network predicts the overall shape of the response quite well but the prediction leads the actual response so that the peaks do not line up. This time shift results in a large RMS error which does not reflect the accuracy of the model in predicting the response pattern. Similarly, if we are interested in predicting only the peak acceleration values, then the DRI model actually performs quite well as can be seen from Figures 6.26 to 6.29.

For these two reasons, the RMS error, while a useful metric of model performance, is limited. Better insight into the suitability of the model can be obtained by visually inspecting the model predicted output.

Comparing these plots with those corresponding to the other models, a case can certainly be made that the neural network model is superior. The performance of the four models is examined more closely in the following sections.

7.1 Model Response to Positive Input Shocks

In response to a positive input shock, the seat accelerates upwards for 1/2 of the shock duration (T) and then accelerates downwards. Since the input is a damped sinusoid, the maximum positive acceleration is reached at approximately t= T/4, and the maximum positive velocity is reached at t = T/2.

If the input shock carries sufficient energy, the subject will leave the seat at some point in its trajectory resulting in a (later) secondary seat/subject collision. Such shocks have a bi-phasic lumbar response. In other words, the lumbar response has two peaks corresponding to the initial shock and the secondary collision. This type of response can be seen in Figure 6.8 between 6.5 and 6.8 seconds. A bi-phasic response results when the input consists of low frequency , high amplitude shocks (e.g., 2-5Hz and > 2g) since these shocks result in a greater separation between the seat and subject. Lower energy shocks (low amplitude and/or high frequency) exhibit little or no secondary response and are, therefore, referred to as mono-phasic. In these cases, the subject does not leave the seat and no secondary collision results. The lumbar response consists of only the initial acceleration shock which transmits upward through the spine. The shape of this response will be a function of the input shock and the biomechanical properties of the spine.

The neural-network's response to high and low energy shocks are shown in Figures 6.21, 6.22 and 6.24. Figures 6.21 and 6.22 show the predicted and measured response to 4g/4Hz and 3g/4Hz input shocks, respectively. The network seems to predict the general shape of the bi-phasic response, but with an error in the timing. Figure 6.24 shows the measured and predicted response to a 4g/11Hz input shock. The neural network output matches the measured response very closely in terms of shape, timing and amplitude. A comparison of these three plots with the corresponding ones for the other models indicates that the neural network model is better at predicting the lumbar-4 response to positive input shocks of the types examined.

7.2 Model Response to Negative Input Shocks

When a negative input shock is applied, the seat accelerates downwards, reaching a maximum acceleration at approximately $t = T/4$ and a maximum velocity at $t=T/2$. If the downward acceleration exceeds -1g, the subject becomes separated from the seat and a collision will result. The force of that collision will depend on the relative velocities of the subject and seat which, in turn, will depend on the frequency and amplitude of the input shock. In some cases, this collision will be sufficiently energetic that the subject will be bounced off the seat again causing yet another collision afterwards. Thus, a negative input shock may result in a mono-phasic or bi-phasic lumbar response.

As indicated by Figure 6.20, the neural network is capable of accurately predicting a mono-phasic response. Figure 6.23 shows the predicted and measured responses to a large amplitude, low frequency negative input shock. The model output follows the general shape of the actual response on the first peak but does not produce the smaller second peak. In addition, the first peak response seems to be time-shifted. Still, this model's prediction is considerably more accurate than the DRI, BS 6841 filter, and the ARX model's response to this type of shock.

7.3 Validation on an Unseen Subject

For the model to be useful it is necessary that it can be applied to subjects other than the one on which it was trained. Therefore, a randomly selected subject (Subject 7) was chosen to further validate the model.

The RMS error results for model validation on Subject 7 are provided in Table 6.4. These results rate the four models in the same order as on the training subject (Table 6.3), with the neural network model performing better than the BS 6841 filter and the DRI model but slightly worse than the ARX model

Figures 6.40 to 6.45 show the predicted response of the model compared with the actual responses. The model seems to predict the shape quite well, although not as well as on the training subject data. This discrepancy is to be expected due to slight differences in weight, height, muscle tone, and other biomechanical characteristics between subjects.

7.4 Model Performance on Low-Level Vibration

The performance of the model to low-level vibration is significant since long-term exposure to such motion can result in tissue damage similar to shorter term exposure to high amplitude shocks. It is, therefore, important that the model have the ability to predict both types of responses.

The model performance on low-level vibration was analyzed in the time and frequency domains, quantitatively and qualitatively. Based on the time-domain RMS prediction errors (Table 6.5), the neural network model performs slightly worse than the BS 6841 filter, roughly the same as the DRI model, and slightly better than the 20'th order ARX model. Visual inspection of the time-domain plots (Figures 6.46 - 6.49) lend support to superiority of the BS 6841 filter. However, visually, the neural network appears to predict the shape of the vibration better than the DRI model.

The results of the frequency-domain analysis seem to indicate that the neural network performance is comparable to that of the BS 6841 model. The PSD estimate of the latter (Figure 6.52) visually seems to match the PSD of the measured data better than does the PSD of the neural network output. However, the neural network exhibits a lower RMS error between the PSD of the measured data and that of the predicted output (Table 6.6). The DRI model is significantly worse than the neural network (and the BS 6841 filter) when compared on the basis of the power spectral density plots as well as the PSD RMS errors.

7.5 Model Limitations

From visual comparison of the model outputs, the neural network arguably performs better than the other three models examined when the input consists of shocks. Moreover, its performance on low-level vibration is better than the DRI and ARX models and comparable to that of the BS 6841 filter. However, there are a few limitations to the neural network's performance: it does not perform well on some types of shocks, it fails to predict the second peak of large amplitude negative input shocks, and the model output is time-shifted in some cases.

The most obvious explanation for the inadequacies of the model output is the small size of the training set. It was originally intended that the training set consist of approximately fifty shocks of varying amplitudes and frequencies. However, this training set resulted in a memory overflow condition during training and, therefore, had to be reduced in size. The maximum allowable training set was approximately 2500 samples, or 17 shocks. However, it was discovered that the network was better able to predict the peak values when a smaller training set was used. The final model was trained on a set of 500 samples, consisting of a -2g/5Hz input shock, a 2g/6Hz shock, a 3g/4Hz shock and approximately 300 samples (2 seconds) of low-level vibration.

In theory, the larger the training set the greater the ability of the network to generalize to unseen data. When the small training set was used, the ability to interpolate and extrapolate to shock types outside the training space is impaired. A close examination of plots in Figures 6.6 to 6.9 indicates that the model performs better on those shocks which are similar to the ones in the training set.

It is likely that better results would be achieved by increasing the variety of shocks in the training set. Numerous attempts to do so did not result in a model that could produce the peak acceleration values. This failure may be the result of insufficient training iterations, number of hidden PEs, number of hidden layers, or compromise method step size. Some of these parameters simply could not be fully optimized on the simulation platform for a variety of reasons. For example, PC memory limited the number of hidden PEs to approximately 10 when the maximum training set was used. Furthermore, the neural network software does not support multiple hidden layers. In some cases, these parameters were not exhaustibly searched due to the limited speed of the PC. Neural network training was often an overnight task but took as long as a week in some cases (e.g. maximum training set, maximum number of hidden PEs, and using the Compromise Method). A week is a long time to wait to find out that a particular method does not work! Hence, model development time was an important factor in the decision to limit the training set size.

The memory limitation problem could have been alleviated to some degree by using an iterative training algorithm, as this avoids the need to compute the gradient across the entire time trajectory. However, the iterative algorithm provided by the neural network software (Recursive Prediction Error, Chen *et al.*, 1990) has its own problems. It is significantly slower than the batch algorithm used. Moreover, this algorithm tends to become unstable, experiencing a problem called co-variance blowup. In fact, the author of the software advises against its use in favor of batch methods (Norgaard, 1995).

A second explanation of the model inadequacies is related to the problem of learning long-term time dependencies. A system displays long-term dependencies if the prediction of an event at time t depends on an input which occurs at an earlier time $\tau << t$. When training a recurrent network, parameters tend to settle into sub-optimal solutions that take into account short-term dependencies but not long-term dependencies. Bengio *et al.* (1994) demonstrated that gradient-based methods are ill-suited for learning long-term dependencies due to the fact that the derivative of the cost function at time t diminishes exponentially as t increases. This problem of the vanishing gradient likely hinders the network from learning the secondary response of large amplitude, low frequency shocks.

Another possible explanation for the model's shortcomings is that the training set lacks sufficient information to enable the network to properly map the input to the output. In training the network we have assumed that there is a one to one correspondence between the input acceleration and the output acceleration. However, this is not the case. The lumbar z-acceleration is actually a function of the seat acceleration in all three biodynamic axes. Moreover, a purely z-axis input shock will result in lumbar acceleration in x and y axes, not just the z-axis. Conceivably, better modeling results could be obtained by using information from all three axes at both the seat and the spine.

A third possible shortcoming of the developed model is the choice of the subject from whom the training data was collected. The power spectral density method used for selecting the typical subject has a few limitations. Only three types of negative shock responses were used in the calculation of the similarity score. It is possible that the chosen subject exhibits an atypical response to other types of seat shocks. A more thorough analysis which included a greater variety of response waveforms would likely lead to a more accurate choice of training subject. Also, the decision criterion was based only on the magnitude of the frequency response, and not the phase response. However, this limitation is possibly justified in that tissue-damaging effects are magnitude related.

Finally, it is debatable whether only one subject can be used as prototype for training a model that is intended for universal application. An alternative strategy may be to train the network on data collected from a number of different subjects. Ideally, the network would learn the similarities that exist in the response of each subject. However, it is also possible that as the network was trained on a new subject , it would "forget" the response of the previous subject. This type of memory resetting is a genuine problem in the training of neural networks. Training the network with multiple subjects was originally considered at the onset of the model development but was not explored further due to the difficulty experienced in modeling a single subject. Now that these problems have been (in most part) overcome, this alternative approach warrants further investigation as a means for developing a general model.

7.6 Stability

A useful model should be well-behaved in response to a variety of inputs. Often stability of the model is a measure of "good" model behaviour: we usually do not wish to see large oscillations or exponential growth in the model's output, regardless of the input applied. However, whereas the necessary conditions for stability of a linear system are well-known, no equivalent tests exist for nonlinear systems. Still, insight into the stability of the neural network model may be gained by examining the model's impulse response.

The discrete-time impulse (or unit sample sequence) is defined as $\delta(t) = 1$ for $t = 0$ and is equal to zero otherwise. A linear, time-invariant system is completely characterized by its response to this impulse. Furthermore, such a system is guaranteed to be stable if its impulse response is absolutely summable. That is, if

$$S = \sum_{t=-\infty}^{\infty} h(t) < \infty \tag{7.1}$$

where h(t) is the system's impulse response. Thus, if the impulse response decays to zero with time, the system is guaranteed to be stable.

However, a nonlinear system is not completely characterized by its impulse response as demonstrated by the Volterra Series expansion(Hsia, 1977)

$$\begin{aligned} y(t) = {} & \sum_{i=0}^{p} h(i)u(t-i) + \sum_{i=0}^{p}\sum_{j=0}^{p} h(i,j)u(t-i)u(t-j) + \ldots \\ & + \sum_{i=0}^{p}\sum_{j=0}^{p}\sum_{m=0}^{p} h(i,j,m)u(t-i)u(t-j)u(t-m) + \ldots \end{aligned} \tag{7.2}$$

for $t \geq p$.

The first right-hand term of the above equation is a convolution of the input signal with the system's impulse response. This term is the time-domain equivalent of multiplying the input by the system's transfer function, H(z). However, for nonlinear systems, the output, y(t), is also defined by the additional right-hand terms, which implies that neither the impulse function nor the transfer function adequately describes a nonlinear system's behavior.

To overcome the limitations of the impulse response in characterizing a nonlinear system, I have plotted the response of the neural network to impulses with various amplitudes other than unity. Figure 7.1 shows the model's response to impulses of weights 10g (98.1 m/s^2), 20g, 30g, and 40g. Figure 7.2 shows the model's response to impulses of weights -10g,-20g, -30g, and -40g. Figure 7.3 shows the model's response to impulse of weights +0.5g and -0.5g. All responses show a decay towards zero which is good evidence, albeit not conclusive, that the model is stable in the intended region of operation.

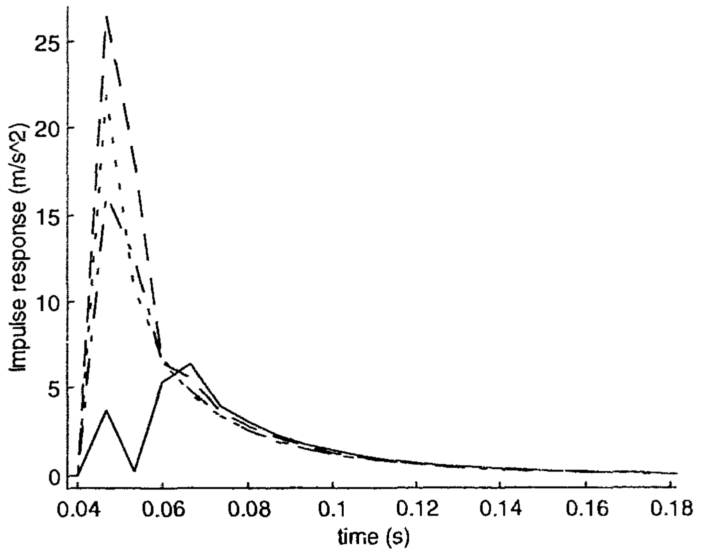

Figure 7.1 The neural network model response to impulses of 10g (solid line), 20g ('-.' line), 30g ('- -' line) and 40g ('. .' line).

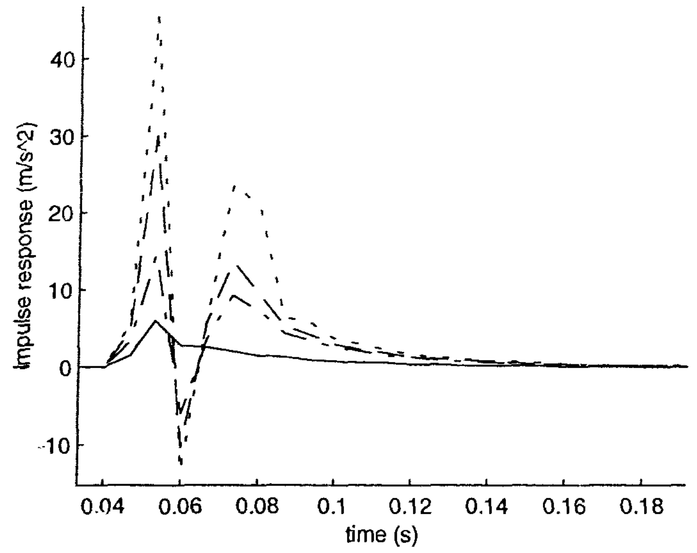

Figure 7.2 The neural network model response to impulses of -10g (solid line), -20g ('-.' line), -30g ('- -' line) and -40g ('. .' line).

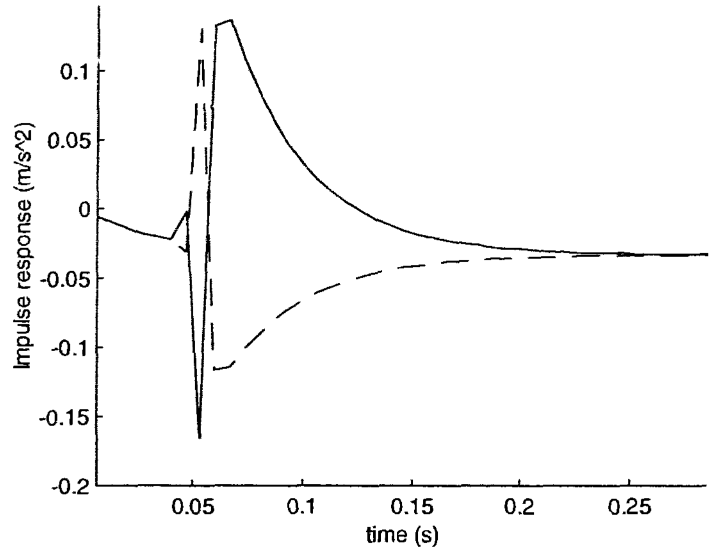

Figure 7.3 The neural network model response to impulses of 0.5g (solid line) and -0.5g ('-.' line).

There a few other interesting characteristic of these graphs worth mentioning. For example, the shape of the impulse response varies for different impulse amplitudes and polarities. If this model was linear, these responses would have the same shape and only differ by a scaling factor. Secondly, the impulse response to 0.5g and -0.5g (Figure 7.3) appears to be symmetrical about the horizontal axis as we would expect for a linear system. Such symmetry indicates that the model exhibits a fairly linear response to low-range of vibration and to low-level shocks. This result is expected in that many studies have demonstrated that the response of the spine to vibration is, in fact, linear.

7.7 Complexity/Implementation Issues

A useful characteristic of a model is parsimony, meaning frugal use of the model's parameters to represent the system. The developed neural network model meets this criteria. By using the Lipschitz algorithm, redundancy in the input/output lags was avoided. And certainly using only three hidden PEs results in a fairly simple network. The model's power lies in its recursive nature, essentially being a nonlinear infinite impulse filter. Still, lest the term "neural network" conjure up vision of some horribly complicated structure, the computational complexity of the final model is analyzed.

In many cases, the training algorithm for a neural network can be quite complicated, as we have seen, requiring an abundance of computer memory, speed, and special software. Once trained, however, the operation of the network can be simply implemented on nearly any processing platform that can perform floating point operations.

The neural network model can be expressed by a composition of the following analytical functions:

$$y_p(t) = \sum_{j=1}^{3} w_{1j}^{(2)} x_j^{(1)}(t) + b_1^{(2)} \tag{7.3}$$

$$x_j^{(1)}(t) = \tanh(w_{j1}^{(1)} u(t-1) + w_{j2}^{(1)} u(t-2) + w_{j3}^{(1)} u(t-3) + w_{j4}^{(1)} y_p(t-1) + w_{j5}^{(1)} y_p(t-2) + b_j^{(1)})$$

$$\text{for } j = 1, 2, 3. \tag{7.4}$$

The first equation requires 7 floating point operations (3 multiplications, 4 additions). Using a Taylor Series expansion, the hyperbolic tangent function may be approximated in the domain [-1.5, 1.5] by

$$\tanh(v) \approx v - \frac{v^3}{3} + \frac{v^5}{15}. \qquad (7.5)$$

Evaluation of this approximation requires 8 flops and evaluation of its argument, v, requires another 10 flops, for a total of 18. Since there are three hidden layer PEs the total number of flops for the hidden layer output is 54. Therefore, the network requires a total of 61 flops to compute each predicted output sample. For comparison, the computational complexity of the ARX, DRI and BS 6841 filter models are provided in Table 7.1

With a sampling rate of 150Hz, a processor would require a computational speed exceeding 9150 flops/second to compute the neural network output in real-time. This speed requirement is easily met by all modern PCs and DSP chips. For example, the AT&T DSP 32 series of processors can execute over 20 million flops/second.

Table 7.1 Computational complexity of the various models

Model	Computational Complexity (flops)
2nd order DRI	7
8'th order BS 6841 Filter	32
5-3-1 Neural Network	61
20'th order ARX	80

CHAPTER 8 CONCLUSIONS

In this thesis, I have demonstrated that an artificial neural network (ANN) can be used to model the dynamic response of the spine to seat-imposed acceleration. The neural network model developed utilizes a recurrent architecture which feeds back previous model predictions to its input. This recurrence results in a model capable of representing complex nonlinear dynamics. Thus, the neural network can model the nonlinear dynamics of the seat-spine system better than existing linear models which are used for assessing the health effects of impacts of repeated shocks and vibration. Specifically, I have shown that for predicting the response of the lumbar spine to large magnitude seat shocks, the neural network model outperforms the Dynamic Response Index model and the British Standard 6841 filter. In addition, the neural network's performance in response to low-level seat vibration is comparable to these established methods.

The recurrent structure of the model provides immense representational ability without requiring an overly complex network. The relatively small size of the model (5 input PEs, 3 hidden layer PEs, and 1 output PE) results in a computational complexity only slightly higher than current linear models. Moreover, once trained, the neural network can be easily implemented on a DSP chip or computer using any software which supports floating point or scaled arithmetic.

I have also demonstrated the use of a systematic method for determining the optimal model orders and have incorporated the results into the neural network structure. The results obtained seem to indicate that, for a limited data set representing -4g to +4g shocks and 0.05g RMS vibration in the -z axis direction, the seat-spine system can be modeled using three previous input samples and two previous output samples. A particular caution should be taken when using the model to extrapolate to regions far from the training range, as demonstrated by the model's performance on shocks which differed significantly from those in the training set. This restriction applies to all models based on interpolation techniques.

In principle, ANNs are an effective modeling tool for this type of problem. However, more work is needed to produce a comprehensive model. Although validated using unseen input data, the model is specific to a range of data obtained from a single subject. Tests on a different subject demonstrated a small degradation in model performance, although results were still generally superior to other methods. The development of a general model will likely require a much larger training set consisting of data obtained from multiple subjects. In addition, future work should strive to produce a neural network model that can predict the output acceleration in all three biodynamic axes. Finally, the model could be extended to predict the response at various vertebral levels, thus, reflecting the multiple degrees of freedom present in the human body.

Objective methods for assessing health risks associated with vehicle shock and vibration are important for limiting exposure time and improving vehicle design. In this thesis, I hope to have provided groundwork for the development of an improved ANN model to predict the spinal accelerations upon which these health risk assessments are based.

REFERENCES

Allen, G.R. (1976) "Progress on a specification for human tolerance of repeated shocks," *Proc. UK Group Meeting on human response to vibration*, Swindon, Royal Military College of Science.

Amirouche, F.M.L. (1987) "Modeling of human reactions to whole-body vibration," *Journal of Biomechanical Engineering*, vol. 109, pp. 210-217.

ASCC (1982) "Human tolerance to repeated shock" *ASCC Advisory Publication 61/25*, Air Standardization Coordinating Committee.

BS 6841 (1987) "Measurement and evaluation of human exposure to whole-body mechanical vibration and repeated shock," London, British Standards Institution.

Backman, A. L. (1983) "Health survey of professional drivers," *Scandinavain Journal of Work Environment & Health*, vol. 9, 30-35.

Beevis, D. and S.E. Forshaw (1985) "Back pain and discomfort resulting from exposure to vibration in tracked armoured vehicles," *Advisory Group for Aerospace Research & Development Conference Proceedings, Backache and Back Discomfort*, Pozzuli, Italy, North Atlantic traty Organization, No. 378, 1-10.

Belytschko, T. and Privitzer, E. (1978) "Refinement and validation of a three-dimensional head-spine model," Aerospace Medical Research Laboratory,Wright-Patterson AF Base, AMRL-TR-78-7, pp.1-159.

Bengio, Y., S. Simard, and P. Frasconi (1994) "Leaning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol.. 5, 157-166.

Billings, S.A., H.B. Jamaluddin and S. Chen (1992) "Properties of neural network with applications to modelling non-linear dynamical systems," *Int. Journal of Control*, Vol. 55, pp. 193-224.

Brinckmann, P. (1988) " Stress and strain of human lumbar discs," *Clinical Biomechanics*, vol. 3, 851-856.

Cameron, B. *et al.* (1996) "Development of a Standard for the Health Hazard Assessment of Mechanical Shock and Repeated Impact in Army Vehicles: Phase 4, Experimental Phase," *United States Army Aeromedical Research Laboratory Contract Report*, no. CR 96-1, Fort Rucker, Alabama.

Carpenter, G. and S. Grossberg (1987) "A massively parallelarchitecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Understanding*, vol. 37, 54-115.

Chang, F.H.I. and R. Luus (1971) "A noniterative method of identification using Hammerstein Model," *IEEE Transactions on Automatic Control*, Vol. AC-16, pp. 464-468.

Chen, S., S.A. Billings, and P.M. Grant (1989) "Non-linear sysyem identification using neural networks," *Int. Journal of Control*, vol. 51, 1191-1214.

Chen, C. (1989) *System and Signal Analysis*, Saunders College Publishing: New York.

Christ, W. and H. Dupis (1966) "Uber die beanspruchung der wirbelsaule unter dem einfluss sinusformiger und stochastischer schwingungen," *Internationale Zeitschrift fur Angewandte Physiologie einschlisslich Arbeitsphysiologie*, vol. 22, 258-278.

Cohn, D. (1993) Personal correspondence, Internet newsgroup comp.ai.neural-nets, Dept. of Brain & Cognitive Science, Massachussetts Institute of Technology, E10-243, Cambridge, MA, 02139.

Cotter, N. (1990) "Stone-Weierstrass theorem and its application to neural networks," IEEE Transactions on Neural Networks, vol. 1, no.4, pp. 290-295.

Crocker, E. N. and L. S. Higgins (1967) "Impact deformation of the vertebrae," *Annual Scientific Meeting of the Aerospace Medical Association*, 149-150.

Cun, Y.L., J.S. Denker, and S.A. Solla (1989) "Optimal Brain Damage," *Advances in Neural Information Processing Systems*, Denver , ed. D. Touretzsky, Morgan Kaufmann, 598-605.

Cybenko, G. (1989) "Approximations by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, 303-314.

Dupuis ,H. (1980) "Vibration exposure of sitting or lying persons in motor vehicless and ambulances" *Human Factors in Transport Research, Vol. 2. User Factors: Comfort, the Environment, and Behaviour*. D.J. Oborne and J.A.Levis, Editors. London: Academic Press, 123-130.

Fairley, T.E. and Griffin, M.J. (1989) "The apparent mass of the seated human body: vertical vibration," *Journal of Biomechanics*, vol. 22, pp. 81-94.

Farmer, J.D. and Sidorowich, J.J. (1988) "Exploiting chaos to predict the future and reduce noise," Preprint of Los Alamos National Laboratory.

Fernandez, B., A.G. Parlos, and W.K. Tsai (1990) "Nonlinear dynamic system identification using artificial neural networks (ANNs)," *International Joint Conference on Neural Networks*, San Diego, 1990, vol 2, pp. 133-141.

Funahashi, K.-I. (1989) "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol.2, 183-192.

Golsse, J.-M. and P.A. Hope (1987) " Analysis of whole-body vibration levels during skidding," *Forest Engineering Research Instituteof Canada*, FERIC Report No. TR-77, 1-22.

Griffin, M.J. (1984) "Vibration dose values for whole-body vibration and shock," *United Kingdom Informal Group Meeting on Human Response to Vibration*, Edinburgh: Heriot-Watt University.

Griffin, M.J. (1986) "Evaluation of vibration with respect to human response," *SAE Paper 860047*, Society of Automotive Engineers International Congress, pp. 11-34.

Gruber, G.J. (1976) "Relationship between wholebody vibration and morbidity patterns among interstate truck drivers," National Institute for Occupational Safety and Health.

Guignard, J. C. (1972) "Physiological effects of vibration" *Advisory Group for Aerospace Research and Development Proceedings, Aeromedical aspects of vibration and noise,"* J.C. Guignard and P.F. King, Editors, North Atlantic Treaty Organization, No. 151, 67-72.

Hansson, T., and S. Holm (1991) " Clinical implications of vibration-induced changes in the lumbar spine," *Orthopedic Clinics of North America*, vol. 22, 247-253.

He, X. and Asada, H. (1993) "A new method for identifying orders of input-output models for nonlinear dynamic systems," *Proceedings of the American Control Confcrence*, pp. 2520-2523.

Hecht-Nielsen, R. (1987) "Counterpropagation networks," *Applied Opt..*, vol. 26, 4979-4985.

Henzel, J. H. , G.C. Mohr, and H. E. von Gierke (1968) "Repraisal of biodynamic implications of human ejections," *Aerospace Medicine*, vol. 39, 231-240.

Hinz, B., Seidel, H., Brauer, D., Menzel, G., Bluthner, R., and Erdmann, H., (1988) "Examination of spinal column vibrations: a non-invasive approach," *Eur. Journal of Applied Physiology*, vol. 57, pp. 707-713.

Hopfield, J.J. (1982) "Neural Networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Science*, U.S., vol 79, pp. 2554-2558.

Hopkins, G.R. (1972) "Nonlinear lumped parameter mathematical model of dynamic response of the human body," Aerospace Medical Research Laboratory,Wright-Patterson AF Base, AMRL-TR-71-29, pp 649-669.

Hsia, T.C. (1968) "Least squares method for nonlinear discrete system identification," *Conference Record, 2nd Asimolar Conference on Circuits and Systems*, pp. 423-426.

Hsia, T.C. (1977) *System Identification*, DC Heath and Company: Lexington, Mass.

Hush, D.R. and B.G. Horne (1993) "Progress in Supervised Neural Networks," *IEEE Signal Processing Magazine,* January, 8- 37.

ISO 2631 (1985) "Evaluation of human exposure to whole body vibration - Part 1: General requirements," Geneva, International Organization for Standardization.

Kechriotis, G., E. Zervas and E.S. Manolakos (1994) "Using Recurrent neural networks for adaptive communication channel equalization," IEEE Transactions on Neural Networks, vol. 5, no.2, pp. 267-277.

Kitazaki, S. and Griffin, M.J. (1995) "A data correction method for surface measurement of vibration on the human body," *Journal of Biomechanics,* vol. 28, pp. 885-890.

Kohonen, T. (1984) *Self-Organization and Associative Memory,* Berlin: Springer-Verlaag.

Kohonen, T. (1988) "Statistical Pattern Recognition with Neural Network: Benchmark Studies" *Proceeding of the Second Annual Conference on Neural Networks,* Vol. 1.

Konda, Y., H. Mito, I. Kadowaki, N. Yamashita, and M. Hosokawa (1985) "Low back pain among container tractor drivers in harbor cargo transportation, *Proceedings of the Congress of the International Ergonomics Association,* Bournemouth: Taylor and Francis.

Kosko, B.(1988) "Bidirectional Associative Memories," *IEEE Transactions on Systems, Man, and Cybernetics,* vol SMC-18, 49-60.

Kosko, B. (1992) *Neural Networks and Fuzzy Systems,* Prentice Hall: NJ.

Kristen, H., HG. Lukeschitsch, and W. Ramach (1981) "Untersuchung der Lendenwirbelssaule bei Kleinlastransportarbeitern," *Arbeitsmedizin Sozialmedizin Praventivmedizin,* vol. 16, 226-229.

Larsen, J, and L.K. Hansen (1994) "Generalization performance of regularized neural network models," *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing,* vol IV, 42-51.

Leontardis, I.J. and Billings, S.A. (1985) "Input-output parametric model for nonlinear system part 1 and 2," *Int. Journal of Control,* vol. 41, pp. 303-344.

Levin, A.U. and K.S. Narendra (1996) "Control of nonlinear dynamical systems using neural networks--Part II: Observability, Identification, and Control," *IEEE Transactions on Neural Networks,* vol. 7, no. 1, pp. 30-47.

MacMurray, J. and Himmelblau, D. (1993) "Identification of a packed distillation column for control via artificial neural networks," *Proceedings of American Control Conference,* pp. 1455-1459.

Markolf, K. L. (1970) " Stiffness and damping characteristics of the thoracolumbar spine" *Proceedings of workshop on bioengineering approaches to problems of the spine.* Bethesda, Maryland: National Ministry of Health.

Marquardt, D. (1963) "An algorithm for least-squares estimation of nonlinear parameters," *SIAM Journal of Applied Mathematics*, vol. 11, 164-168.

Marras, W.S. and Sommerich, C. (1991) "A three-dimensional motion model of loads on the lumbar spine I: Model structure," *Human Factors*, vol. 33, pp. 123-137.

McGill, S.M. (1992) "A myoelectrically based dynamic 3-D model to predict loads on lumbar spine tissues during lateral bending," *Journal of Biomechanics*, vol. 25, pp. 395-414.

Minderman, P. A, and T.J. McAvoy (1993) "Neural Net Modeling and Control of a Municipal Waste Water Process," *Proceedings of the American Control Conference*, pp. 1480-1484.

Monsees, N., R.T. Whyte, J.A. Lines, and R.M. Stayner (1988) " Relationship between subjective assessment and objective measurement of tractor ride vibration (RMS and RMQ) *United Kingdom and French Joint Meeting of the Human Response to Vibration.*

Moody, J. and Darken, C. (1989) "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281-291.

Muksian, R. and Nash, C.D. (1974) "A model for the response of seated humans to sinusoidal displacements of the seat," *Journal of Biomechanics*, vol.7, pp. 209-215.

Nachemson, A. L. and Morris, J.M. (1964) "*In vivo measurements of intradiscal pressure," Journal of Bone and Joint Surgery,*" vol. 46, 1077-1092.

Narendra, K.S. and Parthasarathy, K. (1990) "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 3, pp. 4-27.

Norgaard, M. (1995) *Neural network based system identification toolbox*, Technical Report 95-E-773, Institute of Automation, Technical University of Denmark.

Oppenheim, A.V. and R.W. Schafer (1989) *Discrete-Time Signal Processing*, Prentice Hall: New Jersey.

Orne, D. and Liu, Y.K. (1971) "A mathematical model of spinal response to impact," *Journal of Biomechanics*, vol. 4, pp. 49-71.

Panjabi, M.M., Andersson, G.B.J., Jorneus, L., Hult, E. and Mattsson, L. (1986) "In vivo measurements of spinal column vibrations," *Journal of Bone and Joint Surgery*, vol. 68, pp. 695-702.

Paulson, E. C. (1949) "Tractor drivers' complaints," Minnesota Medicine, vol. 32, 386-387.

Payne, P. (1965) "Personnel restraint and support system dynamics" *Technical Report AMRL-TR-65-127*, Aerospace Medical Research Laboratories, Wright-Patterson Air Force Base, Ohio.

Payne, P. (1991) "A unification of the ASCC and ISO ride comfort methodologies," working paper 377-3, Payne Associates, Severna Park, Md, USA.

Payne, P.R. and Band, E.G.U. (1971) "A four-degree-of-freedom lumped parameter model of the seated human body," Aerospace Medical Research Laboratory, Wright-Patterson AF Base, AMRL-TR-70-35..

Pope, M.H., Svensson, M., Broman, H., and Andersson, G.B.J. (1986) "Mounting of the transducers in the measurement of segmental motion of the spine," *Journal of Biomechanics*, vol. 19, pp. 675-677.

Pope, M. H., A.M. Kaigle, M. Magnussonm, H. Broman, and T. Hansson (1991) "Intervertebral motion during vibration," *Proceedings Institution of Mechanical Engineers Part H, Journal of Engineering in Medicine*, vol. 205, 39-44.

Powell, M.J.D. (1987) "Radial basis function for multivariable interpolation: A review," *Algorithms for Approximation*, Oxford, Clarendon Press.

Prasad, P. and King, A.I. (1974) "An experimentally validated dynamic model of the spine," *Journal of Applied Mechanics*, vol. 41, pp. 546-550.

Puskorius, G.V. and L.A. Feldkamp (1994) "Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 279-290.

Rehm , S. and E. Wieth (1984), " Combined effects of noise and vibration on employees in the Rhenish brown coal opencast working," *Proceedings of the 1st International Conference on Combined Effects of Environment Factors*, O. Manninen, Editor, 281-315.

Roddan, G. *et al.* (1995) "Development of a Standard for the Health Hazard Assessment of Mechanical Shock and Repeated Impact in Army Vehicles: Phase 2" *United States Army Aeromedical Research Laboratory Contract Report*, no. CR 95-2, Fort Rucker, Alabama.

Rolander, S. D. and W.E. Blair (1975) "Deformation fracture of the vertebral end plate,: Orthopedic Clinics of North America, vol. 6, 75-81.

Rumelhart, D.E. and D.L. McLelland (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognitions*. MIT Press, Cambridge.

Sandover, J. (1983) "Dynamic loading as a possible source of low-back disorders," *Spine*, vol. 8, 652-658.

Sandover, J., (1985) "Vehicle vibratioı and back pain," *Advisory Group for Aerospace Research and Development Conference Proceedings*, North Atlantic Treaty Organization, no. 378, pp 1-8.

Saravan, N., Duyar, A., Guo, T.-H. and Merrill, W.C. (1993) "Modeling of the Space Shuttle Main Engine Using Feed-forward Neural Networks," *Proceedings of the American Control Conference*, pp. 2897-2899.

Schmidt, U. (1969) *Vergleichende Untersuchungen an Schwerlastwagenfahrern und Buroangestellten zur Frage der berufsbedingten VerschleiBschaden an der Wirlbelsaule und demn Gelenken der oberen Extremitaten*. Dissertation. Humboldt University, Berlin, GDR.

Simpson, P. K. (1993) "Fuzzy Min-Max Neural Networks - Part 2: Clustering" *IEEE Transactions on Fuzzy Systems*, vol. 1 , 32-44.

Singhal, S. and L. Wu (1989) "Training Multilayer Perceptrons with the Extended Kalman Algorithm," *Advances in Neural Information Processing Systems I*, (Denver 1988) ed. D.S. Touretzky, 1330140, San Mateo, Ca.

Sinha, N.K. and B. Kuszta (1983) *Modeling and Identification of Dynamic Systems Van* Nostrand Reinhold Company, New York.

Smeathers, J.E. (1989) "Measurement of transmissibility for the human spine during walking and running," *Clinical Biomechanics*, vol. 4, pp. 34-40.

Sturges, D.V., D.W. Badger, R.N. Slarve, and D.E. Wasserman (1974) " Laboratory studies on chronic effects of vibration exposure," *Advisory Group for Aerospace Research and Development Conference Proceedings, Vibration and Combined Stress in Advanced Systems,"* H. E. von Gierke, Editor, Oslo, Norway:North Atlantic Treaty Organization, No. 145.

Takagi, T. and Sugeno, M. (1985) "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 1, pp. 116-132.

Village, J. *et al.* (1995) "Development of a Standard for the Health Hazard Assessment of Mechanical Shock and Repeated Impact in Army Vehicles: Phase 3, Pilot Tests," *United States Army Aeromedical Research Laboratory Contract Report*, no. CR 95, Fort Rucker, Alabama.

Volterra, V. (1959) *Theory of functionals and of integral and integro-differential equations*, Dover, New York.

Wellstead, P.E. (1978) "An instrumental product moment test for model order estimation." *Automatica*, vol. 14, pp. 71-89.

Werbos, P.J. (1974) *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. Dissertation in Statistics, Harvard University.

Werbos, P. J., T. McAvoy, and T. Su (1992) " Neural network, system identification, and control in the chemical process industries," *Handbook of Intelligent Control: Neural, Fuzzy, and Adpative Approaches*, Van Nostrand Reinhold: New York.

Williams, R.J. and D . Zipser (1989), "A learning algorithm for continually-running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. 270-280.

Woodside, C.M. (1971) "Estimation of the order of linear systems," *Automatica*, vol. 7, pp. 727-733.

Young, P., A. Jakeman, and R. McMurtes (1980) "An instrumental variable method for model order estimation," *Automatica*, vol. 16, pp. 281-294.

APPENDIX A MATLAB NEURAL NETWORK SCRIPT FILES

A.1 Training Routines

Main Program

```
%-----------------------------------------------------------------------------------------------------------%
%       sp_h.m                                                                                              %
%       Copyright 1996 Jordan Nicol                                                                         %
%                                                                                                           %
%       This program  trains a neural network to model the seat to spine transfer function. %
%       The input is the seat acceleration time series (z-axis) and the output is the z-axis  %
%       acceleration measured at the lumbar-4 vertebra.  Once trained the network          %
%       predicts the L-4 acceleration given only the measured seat input.                  %
%                                                                                           %
%       This program trains many different networks to optimize the number of hidden layers%
%       the number of training iterations. Each network is trained 10 times from different %
%       initial weight values to avoid local minima.                                        %
%-----------------------------------------------------------------------------------------------------------%
clear;

% Input and output lags
L_in = [1 2 3]
L_out = [1 2]

% number of times we re-train network with same parameters
M = 10;

% Specify the different number of iterations to train with and the number of hidden neurons.
iterate_sch = [1:2:100];
hid_sch = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16];

% size of training set
win_size = 500;
max_wins = 1;

% Training method (all use some form of the Levenberg-Marquardt algorithm)
%method = 'nnoe'
%method = 'nnarx';
method = 'nn_comp';

% Select the data set to train and tes the network with.
experiment = 'd';

% Initializations
best_vec = [];
vb_vec = [];
vb_err = 100;
best_mat = zeros(length(iterate_sch),length(hid_sch));
best_mat1 = best_mat;

% Iteration loops for training the neural network.  Outer loop optimizes the number of training
% iterations. Middle loop optimizes the number of hidden Pes. Inner loop tries M different initial
% values.
for jj = 1: length(iterate_sch)
   its = iterate_sch(jj);
   out_lags = L_out;
   prev_outs = length(out_lags);
   inp_lags = L_in;
   prev_ins = length(inp_lags);
   best_hid = 100;  % best error for hidden neurons

% Determine optimal number of hidden neurons.
for j = 1: length(hid_sch)

  hidden = hid_sch(j)
  err_sum = 0;
  best_err = 100;
  best_err1 = 100;

  %Given the number of training iterations and hidden neurons, train the network M times from
  %different initial values.
  for i=1:M

    [W1, W2, NN, NetDef, tra_err,tes_err1, tes_err2] = train(method,experiment, inp_lags,
out_lags,                                                          hidden,its,win_size,
max_wins);

    % Find best network from all the initial starting points
    if (tes_err2 < best_err)
       best_err = tes_err;
       best_W1 = W1;
       best_W2 = W2;
       best_index = i;
    end;


  end;   % End of M loop

  %Create history matrix of best hidden neuron results
  best_mat(jj,j) = best_err;

  % Find best network for varying hidden neurons
  if (best_err < best_hid)
       best_hid  = best_err;
       b_hid_W1 = best_W1;
       b_hid_W2 = best_W2;
       bh_index = j;
  end;

  clear u_test
  clear y_test
  clear Ysim;

end;  % hidden layer for loop
```

```
% Create error vector for best hidden at each iteration
vb_vec = [vb_vec; best_hid];

% Now of all hidden neuron results need to find best result for varying
% training iterations
% This is also the very best network tested.
if (best_hid < vb_err)
        vb_err  = best_hid;
        vb_W1 = b_hid_W1;
        vb_W2 = b_hid_W2;
        bi_index = jj;
        vbh_index = bh_index;
end;

end;  % varying iteration for loop

vb_err                          %Best of best networks test results
best_hid = hid_sch(vbh_index)   %Which number of hidden neurons
best_its = iterate_sch(bi_index)

% Show results of very best network
win_num = 1;
sc_flag = 1;
[u, y, uscale, yscale] = load_dat(experiment, 'training', win_size, sc_flag);
clear u; clear y;

sc_flag = 0;
[u, y, us, ys] = load_dat(experiment, 'training', win_size, sc_flag);

sc_flag = 1;
[u_test, y_test, ut_scale, yt_scale] = load_dat(experiment, 'testing', win_size,sc_flag);

if (strcmp(method,'arxoe1') | strcmp(method,'nn_comp') )
  method = 'nnarx';
end;

%Need to create NetDef to match number of hidden neurons of best network.
hid_str = 'H';
out_str = 'L';
for i = 1: best_hid-1
  hid_str = [hid_str 'H'];
  out_str = [out_str '-'];
end;
NetDef = [hid_str;out_str];

 [tra_err, tes_err, yp1, yp2] = validate(method,experiment,NetDef,NN,inp_lags,out_lags,vb_W1,
vb_W2, u,y, u_test, y_test );

% Save the best network to file
save best_net.mat vb_W1 vb_W2 NN NetDef vb_vec bi_index bh_index tra_err tes_err method
...best_hid best_mat vbh_index win_size hid_sch iterate_sch L_in L_out experiment;
```

```
function [W1, W2, NN, NetDef, tra_err, tes_err] = train(method, experiment, in_lags, out_lags,
                                                        hidden, its, win_size, max_wins);
%-----------------------------------------------------------------------------------------%
% train.m                                                                                 %
%                                                                                         %
% Copyright Jordan Nicol                                                                  %
% 1996                                                                                    %
%                                                                                         %
% Optimizes the neural network parameters on using three different                            %
% model paradigms: Output Error, Equation Error, or Compromise                            %
% Method. The Compromise Method is a weighted average of the two                          %
% approaches.                                                                             %
%---------------------------------------------------------------------------------------- %
% INPUTS:                                                                                 %
%        method          Training method to use                                           %
%        experiment      Data to set to train on                                          %
%        in_lags         The delay values of the input signal                             %
%        out_lags                The delay values of the ouput signal                     %
%        its             Number of training iterations                                    %
%        hidden          Number of hidden layer PEs                                       %
%        win_size                Size of the training file                                %
%        max_wins        No longer used                                                   %
%                                                                                         %
% OUTPUTS:                                                                                %
%    W1                  Re-scaled input-hidden layer weights                             %
%    W2                  Re-scaled hidden-output layer weights                            %
%    NetDef              Network structure string                                         %
%    tra_err             RMS error on training set                                        %
%    tes_err             RMS error on testing set                                         %
%-----------------------------------------------------------------------------------------%

% General Initializations
prev_ins = length(in_lags);
prev_outs = length(out_lags);
num_ins = prev_ins + prev_outs +1;
delay = ones(size(num_ins));
skip = [4];
W1 = rand(hidden,num_ins)-0.5;
W2 = rand(1,hidden + 1)-0.5;
stop_crit =  1e-4;
lambda_init = 0.1;
wt_decay =   0.00;

trparms = [its stop_crit decay_init wt_decay];

%Generate Network structure string
hid_str = 'H';
out_str = 'L';
for i = 1: hidden-1
  hid_str = [hid_str 'H'];
  out_str = [out_str '-'];
end;
NetDef = [hid_str;out_str];
```

```
% Train the network using the Output Error (NNOE), Equation Error (ARX)
% or a combination approach (Compromise Method)
if (strcmp(method,'nnoe'))
   % set up for oe training
   win_num = 1;
   sc_flag =1;
   [u, y, uscale, yscale] = load_dat(experiment, 'training', win_size,sc_flag);
   NN = [prev_outs prev_ins delay];
   [W1,W2,NSSEvec,iter,lambda] = nnoe(NetDef,NN,W1,W2,trparms,skip,y,u);

elseif (strcmp(method,'nnarx'))
   % Set up for nnarx training
   trparms = [its stop_crit decay_init wt_decay];
   NN = [prev_outs prev_ins delay];
   sc_flag = 1;
   [u, y, uscale,yscale] = load_dat(experiment, 'training', win_size, sc_flag);
   [W1,W2,NSSEvec,iter,lambda] = nnarx(NetDef,NN,in_lags,out_lags,W1,W2,trparms,y,u);


elseif (strcmp(method,'nn_comp'))
   % Use Compromise Method. Find intial weights usinf ARX (Max Likelihood).
   % Then move to NNOE (Pure Robust Method) in steps. If w = 1, then we
   %are doing ARX, if w = 0, then we are doing NNOE
   sc_flag = 1;
   [u, y, uscale,yscale] = load_dat(experiment, 'training', win_size, sc_flag);

   % Set up for nnarx training
   trparms = [40 stop_crit decay_init wt_decay];
   NN = [prev_outs prev_ins delay];
   [W1,W2,NSSEvec1,iter,lambda] = nnarx(NetDef,NN,in_lags,out_lags,W1,W2,trparms,y,u);
   lambda


   %Now train with Compromise Method
   decay_init = 0.1;
   trparms = [its stop_crit decay_init wt_decay];

   decay_init = 0.1;
   w = 0.8
   [W1,W2,NSSEvec3,iter,lambda] = nn_comp(NetDef,NN,W1,W2,trparms,skip,y,u,w);
   lambda

   decay_init = 0.1;
   w = 0.5
   [W1,W2,NSSEvec3,iter,lambda] = nn_comp(NetDef,NN,W1,W2,trparms,skip,y,u,w);
   lambda

   decay_init = 0.1;
   w = 0.3
   [W1,W2,NSSEvec3,iter,lambda] = nn_comp(NetDef,NN,W1,W2,trparms,skip,y,u,w);
   lambda

   decay_init = 0.1;
```

```
    w = 0.1
    [W1,W2,NSSEvec3,iter,lambda] = nn_comp(NetDef,NN,W1,W2,trparms,skip,y,u,w);
    lambda

    w = 0.001
    decay_init = 0.1;
    [W1,W2,NSSEvec5,iter,lambda] = nn_comp(NetDef,NN,W1,W2,trparms,skip,y,u,w);
    lambda
    method = 'nnarx';

 end;

%-----------------------------------------------------------------%
%              VALIDATION SECTION                                   %
%-----------------------------------------------------------------%
win_num = 1;
sc_flag = 0;
% Re-load training data to obtain the correct data scaling factors.
[u, y, us, ys] = load_dat(experiment, 'training', win_size, sc_flag);

sc_flag = 1;
% Load unscaled testing data.
[u_test, y_test, ut_scale, yt_scale]
            = load_dat(experiment, 'testing', win_size,sc_flag);

% Rescale the weights so unscaled data can be applied to data
[W1,W2]=wrescale(W1,W2,uscale,yscale,NN);

% Validate the network by performing a pure simulation on the training
% and test set input.
[tra_err, tes_err1, tes_err2] = validate(method,experiment,NetDef, NN,in_lags, out_lags,W1, W2,
u,y, u_test, y_test );
%------------------------------------------------------------------------------------------------%
```

A. 2 Validation Routines

```
function [tra_err, tes_err1, tes_err2,Ysim1, Ysim2] = validate(Method,experiment,NetDef, NN,
                                                lags_in,lags_out,W1, W2, U,Y,U_test,Y_test);%-
---------------------------------------------------------------------------------------------------%
% validate.m                                                                                       %
%Copyright  Jordan Nicol                                                                           %
% 1996                                                                                             %
%                                                                                                  %
% Validates the trained neiral network by calling Neural Net Toolbox functions nnsimul             %
%  for pure simulation (model predicted output) or nneval for one-step prediction                  %
%---------------------------------------------------------------------------------------------------

one_step = 0;

% Need to determine nmax (the no. of samples to skip cut off on the predicted outpt)
na = NN(1);
```

```
nb = NN(2);
if (strcmp(Method,'nnarmax')==1)
  nc = NN(3);
  nk = NN(4);
else
  nc = 0;
  nk = NN(3);
end;

li_max = max(lags_in);
lo_max = max(lags_out)
nmax = max([na,nb+nk-1,nc,lo_max, li_max])+1

if (one_step == 1)
   % Get one step ahead prediction.
   [Ysim1,NSSE1] = nnvalid(Method,NetDef,NN,W1,W2,Y,U);
   [Ysim2,NSSE] = nnvalid(Method,NetDef,NN,W1,W2,Y_test,U_test);
   NSSE1                    %Normalized sum sqaured error
   NSSE
   tra_err = NSSE1;
   tes_err = NSSE;
else
    % Get model predicted output
    Ysim1 = nnsimul(Method,NetDef,NN,lags_in,lags_out,W1,W2,Y,U);
    Ysim2 = nnsimul(Method,NetDef,NN,lags_in,lags_out,W1,W2,Y_test,U_test);
    [tra_err] = get_err(Ysim1,Y,nmax);
    [tes_err1] = get_err(Ysim2,Y_test,nmax);
 end;
%-------------------------------------------------------------------------------------------------------------------%
```

function [rms_err] = get_err(pred,actual,nmax);

```
% Calculate the root mean squared prediction erro
N = length(actual);
Np = length(pred);

actual(1:nmax) = [];
N = length(actual);
pred = reshape(pred,Np,1);
actual = reshape(actual,Np,1);

res = (pred-actual);
P = 2;
rms_err = (    (1/Np)*sum(res.^P)    ).^(1/P) ;
```

A. 3 Neural Network Toolbox Functions

function
[W1,W2,PI_vector,iteration,lambda]=nn_comp(NetDef,NN,W1,W2,trparms,skip,Y,U,w)

```
% Compromise Method (See Werbos) , April 25/96.
% ----
%       Determine a nonlinear output error (OE) model of a dynamic system
%       by training a two layer neural network with the Marquardt method.
%
%        yhat(t)=f(yhat(t-1),...,yhat(t-1),u(t-nk),...u(t-nk-nb+1))
%       The function can handle multi input systems (MISO).
%
% CALL:
% [W1,W2,NSSEvec,iteration,lambda]=nnoe(NetDef,NN,W1,W2,trparms,skip,Y,U)
%
% INPUTS:
% U     : Model input (= Control signal) (left out in the nnarma case)
%         matrix. Structure: [(inputs) | (# of data)]
% Y     : Output data. (1 | # of data)
% NN     : NN=[na nb nk]
%         na = # of past outputs used to determine prediction
%         nb = # of past inputs used to determine prediction
%         nk = time delay (usually 1)
%         For multi input systems, nb and nk contains as many columns as
%         there are inputs.
% W1,W2  : Input-to-hidden layer and hidden-to-output layer weights.
%         If they are passed as [], they are initialized automatically
% trparms : Containing parameters associated with the training (see marq)
%         if trparms=[], it is set to [1000 0 1 0]
% skip   : Don't use the first 'skip' samples for training in order to
%         reduce the influence from the transient occuring because of the
%         unknown initial prediction and gradient. If skip=[]
%         it is reset to skip=0.
%
%
% NB NB NB!
% ---------
% See the function "marq" for an explanation of the remaining inputs
% as well as of the returned variables.
%
```
% Programmed by : Magnus Norgaard, IAU/EI/IMM
% LastEditDate : Sep 8, 1995
% Modified by Jordan Nicol 1996 to train with the Compromise Method
% modifications indicated by bold type
```
%----------------------------------------------------------------------------------------
%--------------              NETWORK INITIALIZATIONS              -------------
%----------------------------------------------------------------------------------------
if skip==[],
 skip=0;
end
skip=skip+1;
Ndat    = length(Y);            % # of data
na      = NN(1);                % Order of polynomials
[nu,Ndat]= size(U);
nb      = NN(2:1+nu);
nk      = NN(1+nu+1:1+2*nu);
nmax    = max(na,nb+nk-1);                % Oldest signal used as input to the model
N       = Ndat - nmax;                    % Size of training set
N2      = N-skip+1;
nab     = na+sum(nb);                     % na+nb
```

```
hidden  = length(NetDef(1,:));                 % Number of hidden neurons
inputs  = nab;                                 % Number of inputs to the network
outputs = 1;                                   % Only one output
L_hidden = find(NetDef(1,:)=='L')';            % Location of linear hidden neurons
H_hidden = find(NetDef(1,:)=='H')';            % Location of tanh hidden neurons
L_output = find(NetDef(2,:)=='L')';            % Location of linear output neurons
H_output = find(NetDef(2,:)=='H')';            % Location of tanh output neurons
y1      = zeros(hidden,N);                     % Hidden layer outputs
y1      = [y1;ones(1,N)];
y2      = zeros(outputs,N);                    % Network output
E       = zeros(outputs,N);                    % Initialize prediction error vector
E_new   = zeros(outputs,N);                    % Initialize prediction error vector
index = outputs*(hidden+1) + 1 + [0:hidden-1]*(inputs+1); % A useful vector!
index2  = (0:N-1)*outputs;                     % Yet another useful vector
iteration= 1;                                  % Counter variable
dw      = 1;                                   % Flag telling that the weights are new
parameters1= hidden*(inputs+1);                % # of input-to-hidden weights
parameters2= outputs*(hidden+1);               % # of hidden-to-output weights
parameters=parameters1 + parameters2;          % Total # of weights
ones_h  = ones(hidden+1,1);                    % A vector of ones
ones_i  = ones(inputs+1,1);                    % Another vector of ones
if W1==[] | W2==[],                             Initialize weights if nescessary
  [W1,W2]=nnarx(NetDef,[na nb nk],[],[],[100 trparms(2:length(trparms))],Y,U);
end
                                               % Parameter vector containing all weights
theta = [reshape(W2',parameters2,1) ; reshape(W1',parameters1,1)];
theta_index = find(theta);                     % Index to weights<>0
theta_red = theta(theta_index);                % Reduced parameter vector
reduced  = length(theta_index);                % The # of parameters in theta_red
index3   = 1:(reduced+1):(reduced^2);          % A third useful vector
dy2dy    = zeros(na,N);                        % Der. of output wrt. the past outputs
dy1dy    = zeros(hidden,na);                   % Der. of hidden unit outp. wrt. past outputs
index4   = 1:na;                               % And a fourth
PSI_red  = zeros(reduced,N);                   % Deriv. of output w.r.t. each weight
RHO      = zeros(parameters,N);                % Partial -"-  -"-

if trparms==[],                                % Default training parameters
  max_iter  = 1000;
  stop_crit = 0;
  lambda    = 1;
  D         = 0;
else                                           % User specified values
  max_iter  = trparms(1);
  stop_crit = trparms(2);
  lambda    = trparms(3);
  if length(trparms)==4,                       % Scalar weight decay parameter
    D = trparms(4*ones(1,reduced))';
  elseif length(trparms)==5,                   % Two weight decay parameters
    D = trparms([4*ones(1,parameters2) 5*ones(1,parameters1)])';
    D = D(theta_index);
  elseif length(trparms)>5,                    % Individual weight decay
    D = trparms(4:length(trparms))';
  end
end
```

```
PI_vector = zeros(max_iter,1);                    % A vector containing the accumulated SSE


% >>>>> CONSTRUCT THE REGRESSION MATRIX PHI  <<<<<<<<<<<<<<<<<<<<<
PHI = zeros(nab,N);
jj  = nmax+1:Ndat;
for k = 1:na, PHI(k,:)    = Y(jj-k); end
index5 = na;
for kk = 1:nu,
 for k = 1:nb(kk), PHI(k+index5,:) = U(kk,jj-k-nk(kk)+1); end
 index5 = index5 + nb(kk);
end
PHI_aug = [PHI;ones(1,N)];                        % Augment PHI with a row containg ones
Y      = Y(nmax+1:Ndat);                          % Extract the 'target' part of Y


%-----------------------------------------------------------------------------
%--------------              TRAIN NETWORK                 -------------
%-----------------------------------------------------------------------------
%clc;
c=fix(clock);
fprintf('Network training started at %2i.%2i.%2i\n\n',c(4),c(5),c(6));

% >>>>>>>>>>>>> COMPUTE NETWORK OUTPUT  y2(theta)  <<<<<<<<<<<<<<<<<<<<<<
for t=1:N,
 h1 = W1*PHI_aug(:,t);
 y1(H_hidden,t) = pmntanh(h1(H_hidden));
 y1(L_hidden,t) = h1(L_hidden);

 h2 = W2*y1(:,t);
 y2(H_output,t) = pmntanh(h2(H_output,:));
 y2(L_output,t) = h2(L_output,:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HERE WE FILL UP PHI WITH AN AVERAGE OF THE PREDICTED OUTPUT AND%
% THE ACTUAL OUTPUT-- THE COMPROMISE METHOD                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 for d=1:min(na,N-t),
    PHI_aug(d,t+d) = (1-w)*y2(:,t) + w*Y(:,t);
 end
end
E = Y - y2;                                         % Prediction error
SSE     = E(skip:N)*E(skip:N)';                     % Sum of squared errors (SSE)
PI      = (SSE+theta_red'*(D.*theta_red))/(2*N2);   % Performance index

while iteration<=max_iter
if dw==1,
% >>>>>>>>>>>>>>  COMPUTE THE RHO MATRIX  <<<<<<<<<<<<<<<<<<<<<<<<<
% Partial derivative of output (y2) with respect to each weight and neglecting
% that the model inputs (the residuals) depends on the weights

 % ==========  Elements corresponding to the linear output units  ===========
 for i = L_output'
```

```
   index1 = (i-1) * (hidden + 1) + 1;

   % -- The part of RHO corresponding to hidden-to-output layer weights --
   RHO(index1:index1+hidden,index2+i) = y1;
   % ---------------------------------------------------------------

   % -- The part of RHO corresponding to input-to-hidden layer weights ---
   for j = L_hidden',
    RHO(index(j):index(j)+inputs,index2+i) = W2(i,j)*PHI_aug;
   end

   for j = H_hidden',
    tmp = W2(i,j)*(1-y1(j,:).*y1(j,:));
    RHO(index(j):index(j)+inputs,index2+i) = tmp(ones_i,:).*PHI_aug;
   end
   % ---------------------------------------------------------------
  end

  % ============ Elements corresponding to the tanh output units =============
  for i = H_output',
   index1 = (i-1) * (hidden + 1) + 1;

   % -- The part of RHO corresponding to hidden-to-output layer weights --
   tmp = 1 - y2(i,:).*y2(i,:);
   RHO(index1:index1+hidden,index2+i) = y1.*tmp(ones_h,:);
   % ---------------------------------------------------------------

   % -- The part of RHO corresponding to input-to-hidden layer weights ---
   for j = L_hidden',
    tmp = W2(i,j)*(1-y2(i,:).*y2(i,:));
    RHO(index(j):index(j)+inputs,index2+i) = tmp(ones_i,:).* PHI_aug;
   end

   for j = H_hidden',
    tmp  = W2(i,j)*(1-y1(j,:).*y1(j,:));
    tmp2 = (1-y2(i,:).*y2(i,:));
    RHO(index(j):index(j)+inputs,index2+i) = tmp(ones_i,:)...
                                   .*tmp2(ones_i,:).* PHI_aug;
    end
    % ---------------------------------------------------------------
   end
   RHO_red = RHO(theta_index(1:reduced),:);


 % >>>>>>>>>>>>>>>> COMPUTE THE PSI MATRIX <<<<<<<<<<<<<<<<<<<<<<<<<<<<
   % --------- Find derivative of output wrt. the past outputs ---------
   for t=1:N,
    dy2dy1 = W2(:,1:hidden);
    for j = H_output',
     dy2dy1(j,:) = W2(j,1:hidden)*(1-y2(j,t).*y2(j,t));
    end

    % Matrix of partial derivatives of the output from each hidden unit with
    % respect to each input:
```

```
    dy1dy(L_hidden,:) = W1(L_hidden,index4);
    for j = H_hidden',
      dy1dy(j,:) = W1(j,index4)*(1-y1(j,t).*y1(j,t));
    end

    % Matrix of partial derivatives of each output with respect to each input
    dy2dy(:,t)= (dy2dy1 * dy1dy)';
  end


  % ---------- Determine PSI by "filtering" ---------
  for t=1:N,
    PSI_red(:,t)=RHO_red(:,t);
    for t1=1:min(na,t-1),
      PSI_red(:,t)  = PSI_red(:,t)+dy2dy(t1,t)*PSI_red(:,t-t1);
    end
  end


% >>>>>>>>>>>>>>>>>>      COMPUTE h_k       <<<<<<<<<<<<<<<<<<<<<<<<<<<<
  % -- Gradient --
  G = PSI_red(:,skip:N)*E(skip:N)'-D.*theta_red;

  % -- Hessian --
  R = PSI_red(:,skip:N)*PSI_red(:,skip:N)';
  dw = 0;
 end

 H = R;
 H(index3) = H(index3)'+lambda+D;              % Add diagonal matrix

 % -- Search direction --
 h = H\G;                              % Solve for search direction

 % -- Compute 'apriori' iterate --
 theta_red_new = theta_red + h;              % Update parameter vector
 theta(theta_index) = theta_red_new;

 % -- Put the parameters back into the weight matrices --
 W1_new = reshape(theta(parameters2+1:parameters),inputs+1,hidden)';
 W2_new = reshape(theta(1:parameters2),hidden+1,outputs)';



% >>>>>>>>>>>  COMPUTE NETWORK OUTPUT  y2(theta+h)  <<<<<<<<<<<<<<<<<<<<<
for t=1:N,
 h1 = W1_new*PHI_aug(:,t);
 y1(H_hidden,t) = pmntanh(h1(H_hidden));
 y1(L_hidden,t) = h1(L_hidden);

 h2 = W2_new*y1(:,t);
 y2(H_output,t) = pmntanh(h2(H_output,:));
 y2(L_output,t) = h2(L_output,:);
```

```
%-----------------------------------------------------------------------------------------------------------------%
% Compromise Method modification                                                                                  %
%-----------------------------------------------------------------------------------------------------------------%
 for d=1:min(na,N-t),
  PHI_aug(d,t+d) = (1-w)*y2(:,t) + w*Y(:,t);
 end

end

E_new    = Y - y2;                % Prediction error
SSE_new  = E_new(skip:N)*E_new(skip:N)';% Sum of squared errors (SSE)
PI_new   = (SSE_new + theta_red_new'*(D.*theta_red_new))/(2*N2); % PI


% >>>>>>>>>>>>>>>>>     UPDATE lambda     <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 L = h'*G + h'*(h.*(D+lambda));

 % Decrease lambda if SSE has fallen 'sufficiently'
 if 2*N2*(PI - PI_new) > (0.75*L),
  lambda = lambda/2;

 % Increase lambda if SSE has grown 'sufficiently'
 elseif 2*N2*(PI-PI_new) <= (0.25*L),
  lambda = 2*lambda;
 end

% >>>>>>>>>>>>     UPDATES FOR NEXT ITERATION     <<<<<<<<<<<<<<<<<<<<
 % Update only if criterion has decreased
 if PI_new < PI,
  W1 = W1_new;
  W2 = W2_new;
  theta_red = theta_red_new;
  E = E_new;
  PI = PI_new;
  dw = 1;
  iteration = iteration + 1;
  PI_vector(iteration-1) = PI;                % Collect PI in vector
  fprintf('iteration # %i   PI = %4.3e\n',iteration-1,PI); % Print on-line inform
 end

%%%%%%%%%%%%%%%%%%% Stochastic annealing %%%%%%%%%%%%%%%%%%%%%%
% Modification by Jordan Nicol August 1995                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
epoch = 3;
jog =0;
if ((jog == 1)&( rem(iteration,epoch) == 0 )&(iteration > epoch));

 if (PI > 0.85*PI_vector(iteration-epoch))
   % Jog weights by a random amount to help get out of local minima
   W1 = W1 + 0.5*(rand(size(W1)) -0.5);
   W2 = W2 + 0.5*(rand(size(W2)) -0.5);
```

```
    fprintf('Weights jogged at iteration # %i\n\n,iteration');
   end;
 end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


  % Check if stop condition is satisfied
  if (PI < stop_crit) | (lambda>1e7), break, end
end
%-------------------------------------------------------------------------------------------
%--------------          END OF NETWORK TRAINING            ------------
%-------------------------------------------------------------------------------------------
PI_vector = PI_vector(1:iteration-1);
c=fix(clock);
fprintf('\n\nNetwork training ended at %2i.%2i.%2i\n',c(4),c(5),c(6));



function Yhat=nnsimul(method,NetDef,NN,lags_in,lags_out,W1,W2,Y,U,obsidx)
% NNSIMUL
% -------
%       Simulate a neural network model of a dynamic system from a sequence
%       of controls alone (not using observed outputs). The simulated output
%       is compared to the observed output.
%
% Call:
% Network generated by nnarx (or nnrarx):
%        Ysim = nnsimul('nnarx',NetDef,NN,W1,W2,Y,U)
% (Likewise for nnoe and nnarmax1+2)
%
% Network generated by nnssif:
%        Ysim = nnsimul('nnssif',NetDef,nx,W1,W2,Y,U,obsidx)
%
% Inputs:
%     See nnvalid/ifvalid
%
% Output:
%     Ysim: Simulated output.
%
% NB! Does not work for models generated by NNIOL.
%
% Programmed by : Magnus Norgaard, IAU/EI/IMM, Technical Univ. of Denmark
% LastEditDate  : Aug 14, 1995
% Modified by Jordan Nicol where bold type 1996

% >>>>>>>>>>>>>>>>>>>>     GET PARAMETERS     <<<<<<<<<<<<<<<<<<<<<<<<<<<<
Fs = 150;   % Hz

skip = 1;
if strcmp(method,'nnarx') | strcmp(method,'nnrarx'),
  mflag=1;

elseif strcmp(method,'nnarmax1') | strcmp(method,'nnrarmx1'),
 mflag=2;

elseif strcmp(method,'nnarmax2') | strcmp(method,'nnrarmx2'),
 mflag=3;

elseif strcmp(method,'nnoe'),
 mflag=4;

elseif strcmp(method,'nnssif')
 mflag=5;

else
 disp('Unknown method!!!!!!!!');
 break
end

% >>>>>>>>>>>>>>>>>>>>>   INITIALIZATIONS    <<<<<<<<<<<<<<<<<<<<<<<<<<<<<
[ny,Ndat] = size(Y);              % # of outputs and # of data
[nu,Ndat] = size(U);              % # of inputs
na      = NN(1);

% ---------- NNARX/NNOE model ----------
if mflag==1 | mflag==4,
 nb = NN(2:1+nu);
 nc = 0;
 nk = NN(2+nu:1+2*nu);

% --------- NNARMAX1 model --------
elseif mflag==2,
 nb = NN(2:1+nu);
 nc   = 0;
 nk   = NN(2+nu+1:2+2*nu);

% --------- NNARMAX2 model --------
elseif mflag==3,
 nb = NN(2:1+nu);
 nc   = NN(2+nu);
 nk   = NN(2+nu+1:2+2*nu);
end

%%%%%%%% MODIFICATION %%%%%
% To allow non-consecutive past inputs . Jan 13/95
%lags_in = [1 5 15];
%lags_out = [1 2];
nb = length(lags_in);
na = length(lags_out);
li_max = max(lags_in);
lo_max = max(lags_out);
nmax    = max([na,lo_max,nb+nk-1,nc,li_max]) + 1;
%nmax    = max([na,nb+nk-1]);        % 'Oldest' signal used as input to the model

N       = Ndat - nmax;                 %Size of training set
nab     = na+sum(nb);                  %na+nb
```

```
 nabc     = nab+nc;                       %na+nb+nc
 outputs    = 1;                          % Only MISO models considered

 % --------- NNSSIF model --------
 if mflag==5,
  nx = NN;
  na = nx;
  nab = nx+nu;
  nabc = nab+ny;
  nk     = 1;
  nmax   = 1;                             % 'Oldest' signal used as input to the model
  N      = Ndat - nmax;                   % Size of training set
  outputs = ny;
  obsidx=obsidx(:)';                      % Find row indices
  rowidx=obsidx;
  for k=2:ny,
   rowidx(k)=obsidx(k)+rowidx(k-1);
  end
  nrowidx = 1:nx;                         % Not row indices
  nrowidx(rowidx)=[];
  Cidx=[1 rowidx(1:ny-1)+1];
  C = zeros(ny,nx);
  C(1:ny,Cidx)=eye(ny);
end

% --------- Common initializations --------
L_hidden = find(NetDef(1,:)=='L')';      % Location of linear hidden neurons
H_hidden = find(NetDef(1,:)=='H')';      % Location of tanh hidden neurons
L_output = find(NetDef(2,:)=='L')';      % Location of linear output neurons
H_output = find(NetDef(2,:)=='H')';      % Location of tanh output neurons
[hidden,inputs] = size(W1);
inputs        = inputs-1;
y1      = [zeros(hidden,N);ones(1,N)];
Yhat    = zeros(outputs,N);

% >>>>>>>>>>>>>>>>  COMPUTE NETWORK OUTPUT  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<
% ---------- NNARX/NNOE/NNARMAX1/NNARMAX2 model ----------
if mflag==1 | mflag==2 | mflag==3 | mflag==4,

% ----- CONSTRUCT THE REGRESSION MATRIX PHI  -----
 PHI_aug = [zeros(nab,N);ones(1,N)];
 jj  = nmax+1:Ndat;
 index = na;

 %%%%% MODIFICATION %%%%%%%%%%
 % Added r = lags_in(k). Jan 13/95.
 for kk = 1:nu,
  for k = 1:nb(kk)
    r = lags_in(k);
    PHI_aug(k+index,:) = U(kk,jj-r-nk(kk)+1);
  end;
  index = index + nb(kk);
 end
 %%% ADDITION: This code never used to be here
 % Included for non-consec predicted outputs. Feb 16/96
 lag_ind(1) = lags_out(1);
 for k = 2:length(lags_out)
   lag_ind(k) = lags_out(k) - lags_out(k-1);
 end;


 % ----- DETERMINE SIMULATED OUTPUT -----
 for t=1:N,
  h1 = W1(:,[1:nab nabc+1])*PHI_aug(:,t);;
  y1(H_hidden,t) = pmntanh(h1(H_hidden));
  y1(L_hidden,t) = h1(L_hidden);

  h2 = W2*y1(:,t);
  Yhat(H_output,t) = pmntanh(h2(H_output,:));
  Yhat(L_output,t) = h2(L_output,:);

  %for d=1:min(na,N-t),
  % PHI_aug(d,t+d) = Yhat(:,t);
  %end
  %%%%% MOD for non-consec outputs Feb 16/96
  for d=1:min(na,N-t) %  min(na,N-t)
      r = lag_ind(d);
      PHI_aug(d,t+d) = Yhat(:,max(t-r+1,1));
  end;
 end

% ---------- State space model ----------
elseif mflag==5,
 % ----  CONSTRUCT THE REGRESSION MATRIX PHI  -----
 PHI = zeros(inputs,N);
 PHI(nx+1:nx+nu,:) = U(:,1:N);
 PHI_aug = [PHI;ones(1,N)];          % Augment PHI with a row containing ones

 % ----- DETERMINE SIMULATED OUTPUT -----
 for t=1:N,
  h1 = W1*PHI_aug(:,t);             % Hidden neuron outputs
  y1(H_hidden,t) = pmntanh(h1(H_hidden));
  y1(L_hidden,t) = h1(L_hidden);

  h2 = W2*y1(:,t);                  % Predicted states
  y2(H_output,t) = pmntanh(h2(H_output,:));
  y2(L_output,t) = h2(L_output,:);
  y2(nrowidx,t)  = y2(nrowidx,t) + PHI_aug(nrowidx+1,t);
  Yhat(:,t)      = C*y2(:,t);

  for d=1:min(1,N-t),
   PHI_aug(1:nx,t+1) = y2(:,t);
  end
 end
end
```

```
% >>>>>>>>>>>>>>>>>      PLOT THE RESULTS      <<<<<<<<<<<<<<<<<<<<<<<<<<<<
si = figure-1;
Y = Y(:,nmax+1:Ndat);
for k=1:outputs,
 if outputs>1,
  figure(si+k);
 end
 tvec = ( linspace(0,length(Y),length(Y)) )/Fs;
 plot(tvec,Y(k,:),'y-'); hold on
 plot(tvec,Yhat(k,:),'m--');hold off
 xlabel('time (seconds)')
 if outputs==1,
  title('Output (dashed) and simulated output (solid)')
 else
  title(['Output (dashed) and simulated output (solid)  #' int2str(k)])
 end

 grid
end
```

APPENDIX B PROCESSING OF LUMBAR ACCELERATION DATA TO ACCOUNT FOR THE MOVEMENT OF SUPRA-VERTEBRAL SKIN

The acceleration at the lumbar vertebra was measured using an acclerometer attached to the skin above the lumbar-4 spinous process. Therefore, the motion measured by the sensor will be similar but not equivalent to the motion of the vertebra. The latter is effectively filtered by the bio-mechanical characteristics of the skin and various other tissues that lie between the vertebra and the accelerometer. To compensate for this filtering we wish to identify a transfer function that maps the spinal acceleration to the acceleration measured at the skin. The inverse transfer function can then be applied to the measured data to obtain the actual spinal acceleration.

The identification and application of this skin transfer function was performed by researchers at BCRI. As such it is beyond the scope of this thesis but the rationale and methods are included here for completeness. The following excerpt from Cameron *et al* (1996) explains how the details of this procedure (reprinted with permission):

Skin Transfer Function

Prior to analysis of acceleration data, it was necessary to determine, and correct for, any movement of the skin surface relative to the underlying bone (spinous process). This correction required a knowledge of the "bone-skin transfer function" for the y and z spinal accelerometers for each subject. Measured acceleration signals were then multiplied by their respective inverse transfer functions. This correction eliminates any contribution of bone-skin movement and provides the true acceleration at the spinous process. In this document, this procedure is referred to as the skin transfer function (STF) method.

As x axis accelerometers measured motion perpendicular to the skin surface, they were not sensitive to shearing motion between the spinous process and the skin. Hence a skin transfer function was not computed for these accelerometers.

The influence of the STF on calculated transmission ratios, relative to low pass filtering at 150 Hz or at 40 Hz, is described below in the section "Comparison of Filtering Effects on the z axis Transmission Ratio".

Linear Modeling Approaches for Identifying the Transfer Function of the Skin.

Hinz et al. (1988) developed a method of calculating bone accelerations from miniature accelerometers attached to the skin. The soft tissues between the spinous process and the accelerometer were modeled as a simple Kelvin element [consisting of a mass with a spring and damping element in parallel], whose parameters described an approximate transfer function between the bone (input) and skin surface-accelerometer (output). System parameters were determined from free damped oscillations of the accelerometer- issue complex in response to an initial displacement, using the "logarithmic decrement" method (Korn & Korn, 1961). This approach was based on the assumption of Franke (1951) that the skin can be described in the first approximation as a Kelvin element for a single excitation and small amplitudes. A similar technique was reported by Smeathers (1989) for measuring accelerations of the spine during walking and running, and by Kitazaki and Griffin (1995) to measure accelerations of the spine and abdomen during low level (2.0 m.sec^{-2}) sinusoidal vibration. Both Hinz et al. (1988) and Smeathers (1989) applied this method to measurement of accelerations that were less than 20 Hz. Kitazaki and Griffin (1995) applied the method to accelerations below 35 Hz.

Skin perturbation data showed that the free response of the vertebra - skin subsystem contained both high and low frequency components. Hence the system could not be truly represented as a simple Kelvin element (i.e., a single degree of freedom, second order system). Frequency components of the y and z axis accelerations measured in response to shocks at the seat were inspected using power spectral analysis. Only low frequency accelerations (<20 Hz) were recorded at the y axis spinal accelerometers in response to shocks at the seat. Skin perturbation data collected in the y axis were therefore low pass filtered, and the tissue-accelerometer subsystem was then modeled as a simple Kelvin element.

y Axis Spinal Accelerations

Skin perturbation data were band pass filtered at 0.5 to 40 Hz. The free response of the tissue-accelerometer system to each perturbation was viewed on the computer monitor using MATLAB® software (The Mathworks Inc., Natick, MA). A typical free damped oscillation of the L3 accelerometer is shown in Figure F-11. The magnitude and timing of adjacent acceleration peaks were digitized on the display monitor. The damping ratio (ζ) and natural frequency (ϖ_n) of the tissue-accelerometer system were then calculated from the logarithmic decrement in amplitude (δ) and the period of the waveform (τ) using the relationships:

$$\zeta = \frac{\delta}{(4\pi^2 + \delta^2)^{1/2}};$$

and,

$$\varpi_n = \frac{2\pi}{\tau(1-\zeta^2)^{1/2}}$$

where,
δ = logarithmic decrement
τ = period (s)
ζ = fraction of critical damping
ϖ_n = undamped angular natural frequency (rads/s)

The transfer function, $H(\varpi)$, between the spinous process and accelerometer was characterized by the amplitude ratio

$$H(\varpi) = \sqrt{\frac{1+(2\zeta\varpi/\varpi_n)^2}{(1-(\varpi/\varpi_n)^2)^2+(2\zeta\varpi/\varpi_n)^2}}$$

and the phase angle, $\phi(\varpi)$

$$\phi(\varpi) = TAN^{-1}\frac{2\zeta(\varpi/\varpi_n)^3}{(1-(\varpi/\varpi_n)^2)^2+(2\zeta\varpi/\varpi_n)^2}$$

where

ϖ = angular frequency.

The bone-skin transfer function of each accelerometer for each subject was based on the average values of ζ and ϖ_n obtained from four separate perturbations.

To estimate the acceleration response of the vertebra underlying the accelerometer, spinal acceleration data of each experimental exposure were converted from the time domain to the frequency domain using a forward FFT. The frequency spectrum was multiplied by the inverse of the bone-skin transfer function, and the data then reconstructed in the time domain using an inverse FFT. This mathematical treatment of the data provided an estimate of the input acceleration signal at the spinous process necessary to produce the output acceleration signal measured at the skin surface.

z Axis Spinal Accelerations

Analysis of spinal accelerations in the z axis revealed substantial acceleration "spikes" in response to shocks input at the seat. These acceleration spikes occurred in response to 2, 3 and 4 g shocks and contained frequency components well above 20 Hz. The higher frequency responses (in the range 20 to 150 Hz) were most noticeable as a result of the 4 to 8 Hz shock inputs, and were present in response to both positive and negative shock directions. Acceleration spikes tended to coincide with the subject hitting the seat. Therefore, they could not be considered to be artifacts in the data which could be removed by low pass filtering. Hence, the assumption of a simple Kelvin element in determining the z axis skin transfer function was inadequate. When this model was applied (using the method described above), the inverse transfer function resulted in an artificial magnification of the high frequency components of the accelerometer signal. Theoretically, these high frequencies would not have been transmitted if the second order linear model was correct. To circumvent this problem, new approaches to modeling of the bone-skin transfer function were investigated. These method included parametric modeling using Least Squares Estimation, Prony's algorithm (Parks and Burrus, 1987) and Steiglitz-McBride (STMCB) iteration (Steiglitz and McBride, 1965); and a linear approximation of a two degrees of freedom model.

Parametric Modeling

A parametric model was developed based on measured input and output data. Assuming a linear system, there were various approaches which could be used for developing such a model. Several of these approaches were investigated and are discussed below.

The general parametric model for the skin transfer function was expressed as a linear, constant coefficient, differential equation which predicted the output acceleration given the input acceleration, and vice versa. Because data were sampled, the discrete-time version of this model was used, namely a recursive difference equation, in which the output of the system at a given time was a linear function of the previous inputs and previous outputs:

$$y(k) + a_{n-1}y(k-1) + \ldots + a_0y(k-n) = \\ b_mu(k) + b_{m-1}u(k-1)+\ldots+ b_0u(k-m),$$

where u(k) and y(k) were the input and output values, respectively, sampled at instant k, with coefficients a_i and b_i .

Three modeling methods were investigated, all of which involved optimizing the coefficients a_i and b_i given the measured input and output data from the system. The first method was the Least Squares Estimation (LSE) technique, which fits the data to a polynomial corresponding to the difference equation. As anticipated, this approach was not appropriate for the skin transfer function problem since there is, in fact, no input data, only an initial excitation (the skin pluck). It was hoped that the initial acceleration at time t=0 could be modeled as an input signal $u(k) = A_0$ for k=0 and u(k) = 0 for $k \neq 0$. This approach did not yield a functional result.

Two other approaches were tried: Prony's algorithm (Parks and Burrus, 1987) and the Steiglitz-McBride (STMCB) iteration (Steiglitz and McBride, 1965). Both approaches seemed to overcome the limitation of having no true input signal available. Both methods utilize the fact that a linear system can be completely characterized by its impulse response. (Intuitively, this can be seen from the fact that an impulse, by definition, consists of all frequencies and, therefore, excites all modes of a system). The Prony and STMCB algorithms determine the difference equation coefficients from the assumed impulse response of the system. It was assumed that plucking the skin resulted in an acceleration impulse and that the signal measured by the accelerometer was the approximate impulse response. Both algorithms produced a 4th order model with two resonant frequencies, the lower of which was similar to that obtained using the "logarithmic decrement" method. The main disadvantage was that a true impulse is assumed to have a weight of 1, which was not the case for the skin pluck impulse. As a result, the true magnitude of the transfer function's frequency response was subject to a scaling factor error.

Two Degrees of Freedom Linear Approximation

From the above analyses it appeared that the soft tissues between the vertebra and skin contained two resonant frequencies, or possibly represented a non-linear system. To overcome this problem and obtain a linear approximation of the system the following method was utilized.

A series of perturbations were applied to the skin immediately below the z axis accelerometer, and the resultant acceleration data recorded. The skin perturbation data were band pass filtered at 0.5 to 150 Hz. The free damped response of the tissue-accelerometer system to each perturbation was viewed on the display monitor using MATLAB®software. An example of free damped oscillation of the LT4 accelerometer is shown in Figure F-12. The frequency spectrum of the free damped oscillations of the vertebra-skin subsystem was computed and plotted, and the (two) dominant frequency components of the acceleration data were identified as shown in Figure F-13. The skin perturbation data were then low pass (0.5 to 50 Hz) and high pass filtered (50 to 150 Hz) to isolate the two main frequency components. The high and low pass time domain components of the same perturbation are shown in Figure F-14 and F-15). It was assumed that the data within each frequency band could be modeled independently as the outputs of separate Kelvin elements. The magnitudes and timing of adjacent acceleration peaks within each frequency band were digitized on the display monitor. The system parameters (ϖ_n and ζ) of the two frequency bands were then determined separately using the "logarithmic decrement" method. These parameters defined two independent models for the low and the high frequency components of the tissue-accelerometer subsystem. The bone-skin transfer function of each model was then determined from the system parameters as described above (ϖ_n and ζ).

A compensation filter was developed with frequency response characteristics derived from the low pass and high pass models. The transfer function of each 'model' was obtained using a MATLAB® subroutine. The magnitude of the frequency response curves were plotted and the frequency (f_i)at which the two curves intersected was determined. This frequency was used to delineate the low and high frequency ranges of the compensation filter. An example of this procedure is shown in Figure F-16.
In order to correct the spinal acceleration response, the following procedure was used. The measured spinal acceleration in the z axis was low pass, and then high pass filtered to create two separate data records. The common cut-off frequency (f_i) was determined as described above. The filter characteristics of the appropriate model were then applied to the low pass and high pass frequency components of the spinal acceleration data. The corrected acceleration data within the two frequency bands were then added to obtain the predicted acceleration at the spinous process. This procedure provided a piecewise inverse transfer function over the complete frequency range (0.5 to 150 Hz).

In summary, the spinal acceleration data were separated into low frequency and high frequency components; each component was treated separately with a linear correction; and then the two components were summed to obtain the corrected acceleration at the vertebra. The spinal acceleration data recorded by the L4 accelerometer in response to a negative 4 g, z axis shock at the seat is shown in Figure F-17. For comparison, the predicted acceleration at the spinous process after correction by the compensation filter is superimposed on the acceleration data.

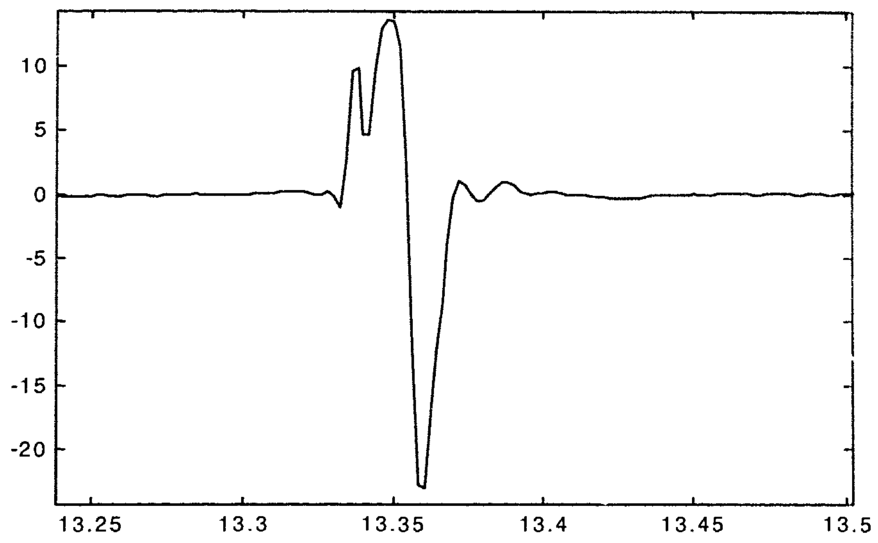

Figure F-12. Example of a free damped oscillation of the L4 acclererometer (z-axis) in response to perturbation of the skin.

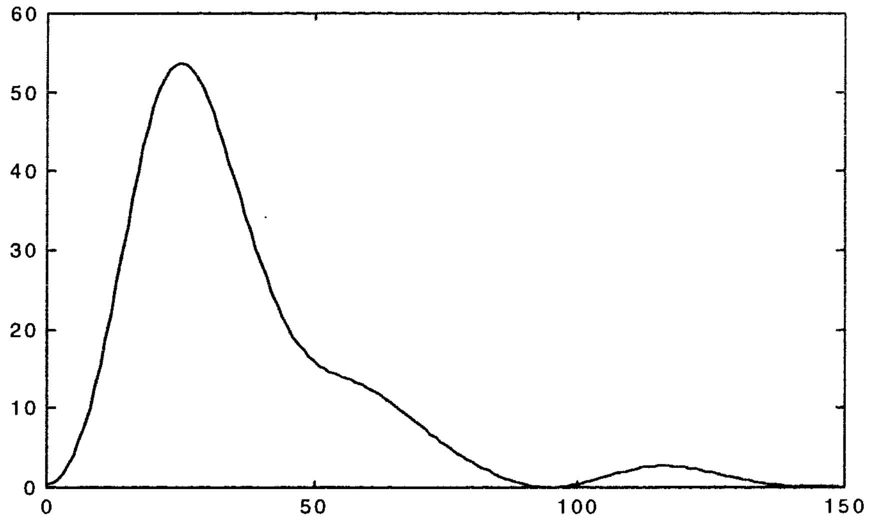

Figure F-13. Spectral density of a free damped oscillation of the skin-accelerometer system at L4 (z axis).

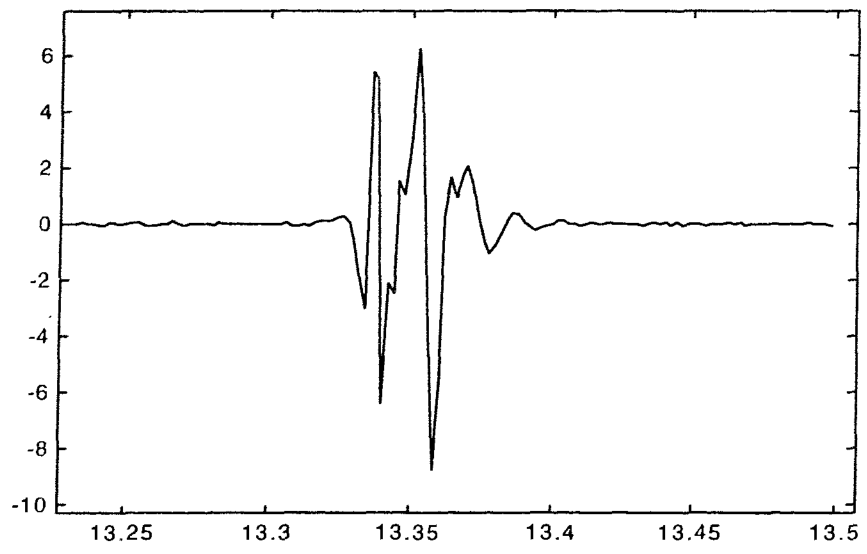

Figure F-14. The high pass acceleration component of a skin perturbation.

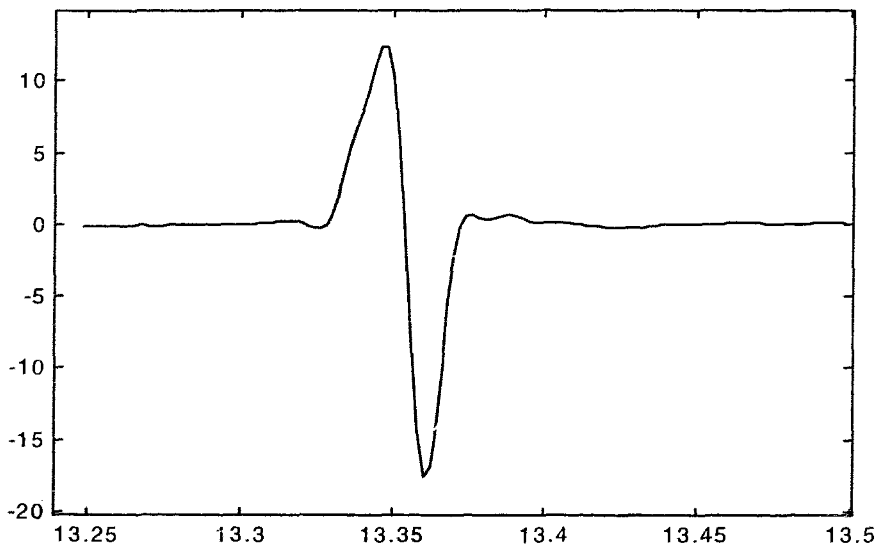

Figure F-15. The low pass acceleration component of a skin perturbation

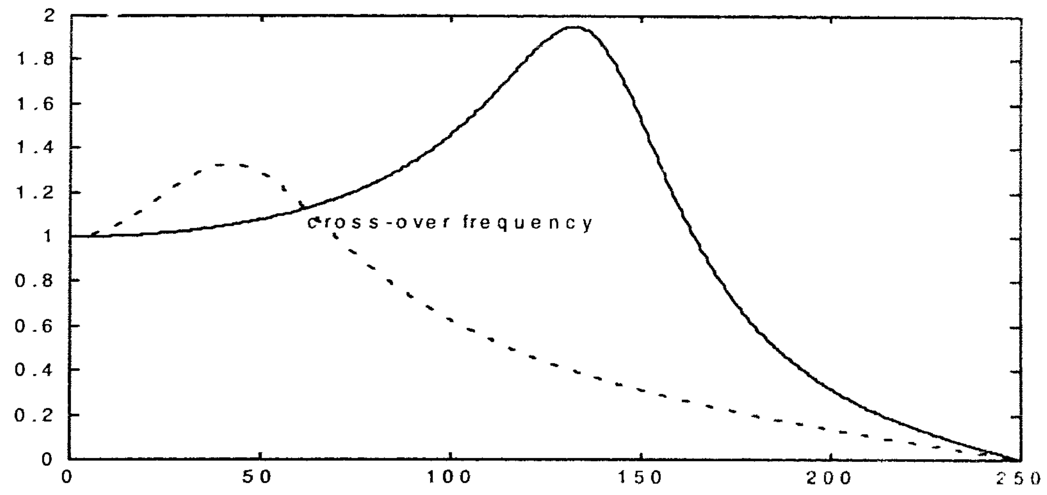

Figure F-16. The amplitude components of low frequency (dotted line) and high frequency (solid line) bone-skin transfer functions derived from a free damped oscillation. The cross-over frequency (fi) was used to establish the cut-off frequency for low pass and high pass filtering of the measured acceleration signal.

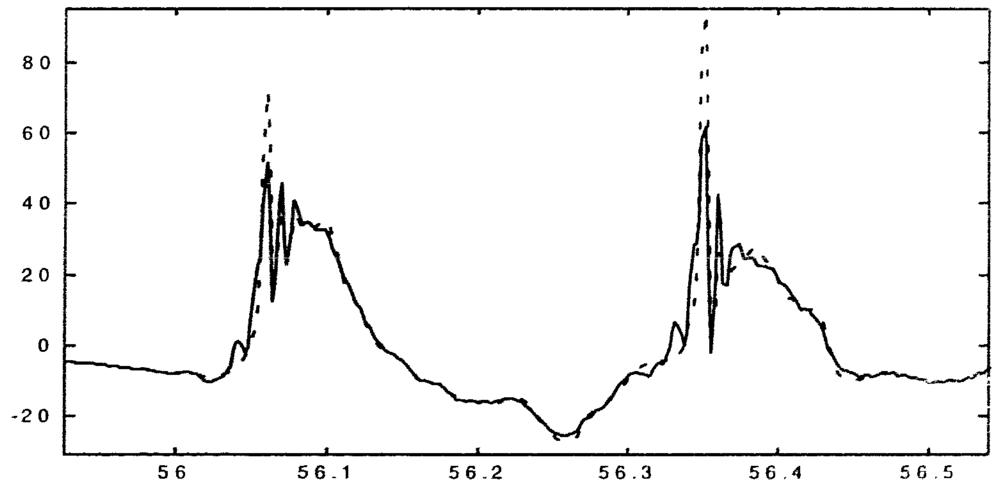

Figure F-17. Recorded L4 accelerometer response to a -4 g, z axis shock at the seat and the predicted acceleration at the spinous process after correction by the skin transfer function. Dotted line = recorded L4 response; Solid line = corrected response

References

Franke, E.K. (1951) "Mechanical Impedance of the surface of the body," *Journal of Applied Physiology,"* vol. 3, 582-590.

Hinz, B., S. Helmut, D. Brauer, G. Menzel, R. Bluthner, and T. Erdmann (1988) "Examination of spinal column vibrations: a non-invasive approach," *European Journal of Applied Physiology,* vol. 57, 707-713.

Kitazaki, S. and M.J. Griffin (1995) "A data correction method for the surface of the human body," *Journal of Biomechanics*, vol. 28, 885-890.

Korn, G.A. and T.M. Korn (1961) *Mathematical Handbook for Scientists and Engineers.* Toronto: McGraw-Hill Book Company Inc., 254.

Parks, T.W. and C.S. Burrus (1987) *Digital Filter Design.* New York: John Wiley and Sons Inc.

Smeathers, J.E. (1989) "Measurement of the transmissibilityfor the human spine during walking and running," *Clinical Biomechanics*, vol. 4, 34-40.

Steiglitz, K. and L.E. McBride (1965) "A technique for the identification of linear systems," *IEEE Transactions on Automatic Control*, vol. AC-10, 461-464.

APPENDIX C MODEL ORDER DETERMINATION SOURCE CODE

This appendix contains the source code which implements the Lipschitz Number algorithm used for determining the orders of nonlinear dynamical systems. The program is written in Turbo C, for implementation on a PC platform.

```
/*-----------------------------------------------------------------------------------*/
/*                          OrderFinder                                              */
/*                          Version 1.a                                              */
/*                          Copyright 1996  Jordan Nicol                             */
/*                                                                                   */
/*-----------------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------------*/
/*            About This Program                                                     */
/*                                                                                   */
/*      This program can be used to identify the model order numbers of a nonlinear */
/* dynamic system based on the input and output data only.  This is important for   */
/* predictive performance of the model. The technique is based on the algorithm devised */
/* by Xiangdong He and Haruhiko Asada of MIT (see reference below). Once the correct */
/* model numbers are obtained, an input-ouput model can be implemented with a        */
/* feedforward or a recurrent  neural network using the following format:            */
/*                                                                                   */
/*    y(t) = F( y(t-1),...,y(t-m), u(t-1),...,u(t-l) )                               */
/*                                     */
/* where, F is the nonlinear function to be approximated; y is the scalar output time */
/* series; u is the scalar input time series; m and l are the model order numbers    */
/*( sometimes called the the lags ).                                                 */
/*                                                                                   */
/*-----------------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------------*/
/*            How To Use This Program                                                */
/*                                                                                   */
/* This program calculates an index called the Lipschitz Number which is obtained    */
/* using couple of fairly complex equations, certainly too complex to describe with this */
/* editor.  The gist of this algorithm is a follows:                                 */
/*    1) Choose the model numbers (say m=1, l=1)                                     */
/*    2) Calculate the Lipschitz *QUOTIENTS* from the input-output                   */
/*       data, given m and l.                                                        */
/*    3) Choose the P largest L.Q.'s, where P = 0.01N to 0.02N                       */
/*       where N is the number of input samples                                      */
/*    4) Use these L.Q.s to calculate the Lipschitz Number q(n).                     */
/*                                                                                   */
/* By repeating this procedure for different values of m and l, the optimal model orders */
/* can be determined. Essentially, as you increase m and l, q(n) will decrease from a very */
/* large number. At some point the rate of decrease will level off--at the "knee" of the */
/* graph of  q(n) vs. n=m+l. This knee will occur at the optimal model order.        */
/*                                                                                   */
/*-----------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------------------------*/
/*                    References                                                                 */
/*                                                                                               */
/*  He, X. and Asada, H. "A new method for indentifying orders of input-output models           */
/*  for nonlinear dynamic systems," in Proceedings of the American Control Conference,          */
/*  June 1993, pp. 2520-2523.                                                                    */
/*                                                                                               */
/*  Xiangdong He. hxd@MIT.EDU  617-225-9789                                                      */
/*                      617-253-3772                                                             */
/*                                                                                               */
/*  Haruhiko Asada (Dr.)      617-253-6257                                                       */
/*                                                                                               */
/*-----------------------------------------------------------------------------------------------*/

/* Define Constants */
#define  N 3000                            /* Number fo inpu/output samples */
#define  P N/100
#define C 14                               /* Size of vector arrays: l+m <= C        */
#define n_max 7                            /* Max. combined orders : n = l=m         */
#define success 1
#define true 1
#define false 0
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* Define function prototypes */
char Get_Data(double *, double *);
double Calc_Qij(double *,double *, int, int, int, int);
double Calc_Index(double *, int);
void Print1(double *, double *, double *, double *);
char Get_Vector(double *, double *, int, int);
char Get_Denom(double *, double *,int,int,int,int,double *);
void Print2(double *);


/*--------------------------------------------------------------------------------------------  */
/*                   MAIN PROGRAM                                                               */
/*----------------------------------------------------------------------------------------------*/
int main(void)
{
    int  i,j,k,l,m,t, m_max, l_max,small_ind,n;
    double LipQ,small,Q[P],*Qptr,qn, X [N], *Xptr, Y [N], *Yptr;
    FILE *stream, *stream1, *stream2;
    char inputstr[20], *endptr;

    /* Intialize variables */
    Xptr = X;
    Yptr = Y;
    small = 0;
    m_max = 7;                             /* output lags */
    l_max = 6;                             /* input lags  */
    small_ind = 0;
    for (i = 0; i < P; i++)
            Q[i] = 0;

    l = 0;
    m = 0;


    stream1 = fopen("lags_err.out","w+");

    /* Read the input and output data from file */
    if (Get_Data(Xptr,Yptr)==false)
          return(0);

for (l = 0; l <=l_max; l++)
    {
      for (m = 1; m <=m_max; m++)
      {

      /* Intialize variables */
      Xptr = X;
      Yptr = Y;
      small = 0;
      small_ind = 0;
      for (i = 0; i < P; i++)
           Q[i] = 0;
     for (i = 0; i < N; i++)
     {
         /*  printf("Percent completed: %4.2f \n",(float) i/N); */
          for (j = 0; j < N; j++)
          {
            if (j != i)
            {
                    /* Calculate the Lipshitz coefficient and see if it is among the P largest  */
                    /* encountered so far.                                                      */
                    LipQ = Calc_Qij(Xptr,Yptr,i,j,l,m);

                    /* If LipQ comes back as zero just ignore it small is the smallest L.Q. in  */
                    /* the array Q. So, if the newly calculated L.Q. id larger than small       */
                    /* it should go in the array instead. Then we figure the new small one      */
                    /* for next time.                                                           */
                     if (LipQ > small)
                     {
                            small = LipQ;
                            Q[small_ind] = LipQ;

                            /* find new small */
                            for ( k = 0; k < P; k++)
                            {
                              if ( Q[k] < small )
                              {
                                     small = Q[k];
                                     small_ind = k;
                              }
```

```
                                }

                           }
                      }
                   }
            }

            /* Q contains the P largest Qij's */
            Qptr = Q;
            /* Use P largest Qij's to calculate the index value for the given */
            /* value of n, where n = m + l. Note m and l values set above.    */
            n = m+l;
            qn = Calc_Index( Qptr,n);


            printf("m = %d\n",m);
            printf("l = %d\n",l);
            printf("q(n) = %6.3f\n\n",qn);


            fprintf(stream1,"%6.3f       ",qn);


         }      /* m for loop */


         fprintf(stream1," \n");
      }         /* l for loop  */
         fclose(stream1);
         return(0);
      }
      /*------------------------------End of MAIN------------------------------------------------------------------------*/
```

```
/* Read data from file */
char Get_Data(double *xptr, double *yptr)
{
   FILE *stream;
   char inputstr[20], *endptr;
   int i;

   /* Open the input data file */
   stream = fopen("seat_zd.tra","r+");
   fseek(stream, 0, SEEK_SET);

    /* Read in the data to an array */
    for (i = 0; i < N; i++)
    {
          if (fgets(inputstr,14,stream))
          {

              *xptr = strtod(inputstr, &endptr);
              fgets(inputstr,1,stream); /* gobbles up the EOL */
```

```
                xptr++;
            }
            else
            {
                printf("Error reading string for file1.\n");
                return(false);
            }
        }
        fclose(stream);


        /* Open the output data file for reading.        */
         stream =fopen("lumb_zda.tra","r+");
        fseek(stream, 0, SEEK_SET);

        /* Read in the data to an array */
        for (i = 0; i < N; i++)
        {
            if (fgets(inputstr,14,stream))
            {

            *yptr = strtod(inputstr, &endptr);
                fgets(inputstr,1,stream);  /* gobbles up the EOL */
                yptr++;
            }
            else
            {
                printf("Error reading string for file2.\n");
                return(false);
            }
        }
        fclose(stream);
        return(success);
}
```

/* Calculates the Lipschitz quotient */
double Calc_Qij(double *xptr, double *yptr, int i, int j, int l, int m)
{

```
  double yi, result, denom, *denom_ptr, num;

  denom_ptr = &denom;

  /* calculate denominator  | Xi - Xj |  */
  if (Get_Denom(xptr,yptr,i,j,l,m,denom_ptr))
  {
        /*calculate numerator  | yi-yj |  */
        yptr += i;
        yi = *yptr;
        yptr = yptr - i + j;
        num = fabs(yi - *yptr);

        /* Calculate Lipschitz quotient */
```

```
            if (*denom_ptr !=0)
              result = num/(*denom_ptr);
    }
    else
            result = 0;

   /* restore ptr values */
   xptr -= j;
   yptr -= j;
   return(result);
}
```

/* Calculates the Lipschitz number--an average of Lipschitz quotients. */
double Calc_Index(double *qptr, int n)

```
{
    int k;
    double exp,result,nroot, prod;


    prod = 1;
    exp = P;
    exp = 1/exp;
    nroot = sqrt(n);
    for ( k = 1; k <= P; k++)
    {
            prod = nroot*prod*(*qptr);
            qptr++;
    }
    result = pow(prod,exp);

    return(result);
}
```

/* Calculates the demnominator portion of the Lipschitz quotient */
char Get_Denom(double *xptr, double *yptr,int i,int j,int l,int m, double *den_ptr)

```
{
    double Xi[C], *Xi_ptr, Xj[C], *Xj_ptr, sum, term;
    char S1, S2, S3, S4;
    int k;

   /*  for (k = 0; k < 10; k++)
    {
            Xi[k] = 0;
            Xj[k] = 0;
    }
   */
    Xi_ptr = Xi;
    Xj_ptr = Xj;

    S1 = Get_Vector(Xi_ptr,yptr,i,m);
    Xi_ptr += m;
    S2 = Get_Vector(Xi_ptr,xptr,i,l);
    S3 = Get_Vector(Xj_ptr,yptr,j,m);
    Xj_ptr += m;
    S4 = Get_Vector(Xj_ptr,xptr,j,l);
    Xi_ptr -= m;
    Xj_ptr -= m;

    if (S1&S2&S3&S4)
    {
          sum = 0;
          for (k=0; k < m +l; k++)
          {
             term = pow((*Xi_ptr - *Xj_ptr),2);
             sum += term;
             term = 0;
             Xi_ptr++;
             Xj_ptr++;
          }

          Xi_ptr -= m+l;
          Xj_ptr -= m+l;
          *den_ptr = sqrt(sum);
   /*     Print1(xptr,yptr,Xi_ptr,Xj_ptr);  */
          return(success);
     }
     else
          return(false);
}
```

```
/* Calculates one of the terms in the denominator of the L. quotient */
char Get_Vector(double *vectptr, double *datptr, int i, int r)
{
  int k;

  datptr += i;
  if (i >= r)
  {
    for ( k = 0; k < r; k++)
    {
          datptr--;
          *vectptr = *datptr;
          vectptr++;
    }
    return(success);
  }
  else
    return(false);
}
```

APPENDIX D THE ARX MODEL AND LEAST SQUARES ESTIMATION

The ARX (auto-regressive with exogenous variables) model expresses the current output of a linear, time-invariant system in terms of previous inputs and previous outputs. If we denote the input to the system at time t=k as u(k) and the output as y(k) then the ARX model can be expressed as

$$y(k) = \sum_{i=0}^{n} b_i u(k-i) - \sum_{i=1}^{m} a_i y(k-i). \tag{D.1}$$

Define the observation vector

$$\mathbf{x}_k = [-y_{k-1} \quad -y_{k-2} \quad \dots \quad -y_{k-n} \quad u_k \quad u_{k-1} \quad \dots \quad u_{k-m}]^T \tag{D.2}$$

and the parameter vector

$$\Theta = [a_1 \quad a_2 \quad \dots \quad a_m \quad b_0 \quad b_1 \quad \dots \quad b_n]^T. \tag{D.3}$$

Equation D.1 can then be expressed in vector notation as

$$y_k = \mathbf{x}_k^T \Theta. \tag{D.4}$$

We want to determine an estimate for Θ given our set of input output data. If we have N samples then we can define an observation matrix which consists of N rows of consecutive $\mathbf{x}_k$. That is,

$$\mathbf{X} = [\mathbf{x}_k^T \quad \mathbf{x}_{k+1}^T \quad \dots \quad \mathbf{x}_{k+N-1}^T \quad \mathbf{x}_{k+N}^T]^T. \tag{D.5}$$

A system of linear equations can then be set up to solve for Θ:

$$\mathbf{y} = \Theta \mathbf{X}. \tag{D.6}$$

If N=n+m+1, the matrix **X** is square and, thus, ;likely invertible. The solution to Equation D.6 is then given by

$$\Theta = \mathbf{X}^{-1} \cdot \mathbf{y}. \tag{D.7}$$

In most practical situations, the measurement data is corrupted by noise, and it is impossible to find the exact solution to Equation D.7. In this case, we want to find the least squared error fit of the parameters to the data. In general, the parameter estimate will be improved the greater the size of N (the length of data used to generate the estimate. Note, however, that when N > n+m + 1, the matrix X is no longer square and, hence, not invertible. Fortunately, this problem is taken care of by the formulation of the least squares solution.

Let the system of N equations be given by

$$\mathbf{y} = \mathbf{X}\Theta + \mathbf{e}, \tag{D.8}$$

where the **e** is the fitting error or residual given by

$$\mathbf{e} = [e(1) \quad e(2) \quad \dots \quad e(N)]^T \tag{D.9}$$

The squared error,J, is given by

$$\begin{aligned} \mathrm{J} &= \mathbf{e}^{\mathrm{T}}\mathbf{e}. \\ &= [\mathbf{y} - \mathbf{X}\Theta]^T[\mathbf{y} - \mathbf{X}\Theta] \end{aligned} \tag{D.10}$$

To find Θ that corresponds to the minimum J, we set

$$\frac{\partial \mathrm{J}}{\partial \Theta} = \mathbf{0}. \tag{D.11}$$

It can easily be shown that the minimum J occurs for

$$\hat{\Theta} = [\mathbf{X}^T\mathbf{X}]^{-1}\mathbf{X}^T\mathbf{y}. \tag{D12}$$

The following recursive algorithm from Sinha (1983) was used for solving the Equation D.12:

Let $\mathbf{x}_k$ and Θ be defined as above. Define **Q** and **P** as square matrices of dimension n+m+1, with $\mathbf{Q}_0 = \mathbf{I}$, and $\mathbf{P}_0 = \mathbf{0}$. The k'th step of the algorithm is then written as:

For k ≤ (m + n + 1),

$$\Theta_{k+1} = \Theta_k + \frac{\mathbf{Q}_k\mathbf{x}_k(y_k - \mathbf{x}_k^T\Theta_k)}{\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k} \tag{D.13}$$

$$\mathbf{Q}_{k+1} = \mathbf{Q}_k - \frac{\mathbf{Q}_k\mathbf{x}_k(\mathbf{Q}_k\mathbf{x}_k)^T}{\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k} \tag{D.14}$$

$$\begin{aligned} \mathbf{P}_{k+1} = \mathbf{P}_k &- \frac{\mathbf{P}_k\mathbf{x}_k(\mathbf{Q}_k\mathbf{x}_k)^T + \mathbf{Q}_k\mathbf{x}_k(\mathbf{P}_k\mathbf{x}_k)^T}{\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k} \\ &+ \frac{(\mathbf{Q}_k\mathbf{x}_k)(\mathbf{Q}_k\mathbf{x}_k)^T(1 + \mathbf{x}_k^T\mathbf{P}_k\mathbf{x}_k)}{(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_{k})^2} \end{aligned} \tag{D.15}$$

For k > (m + n + 1),

$$\Theta_{k+1} = \Theta_k + \frac{\mathbf{P}_k \mathbf{x}_k (y_k - \mathbf{x}_k^T \Theta_k)}{1 + \mathbf{x}_k^T \mathbf{P}_k \mathbf{x}_k} \qquad \text{(D.16)}$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \frac{\mathbf{P}_k \mathbf{x}_k (\mathbf{P}_k \mathbf{x}_k)^T}{1 + \mathbf{x}_k^T \mathbf{P}_k \mathbf{x}_k} \qquad \text{(D.17)}$$

Using the above algorithm and setting m=n, the ARXmodel was trained and tested for various model orders. The graph in Figure D.1 indicates that the optimal model was approximately 20.

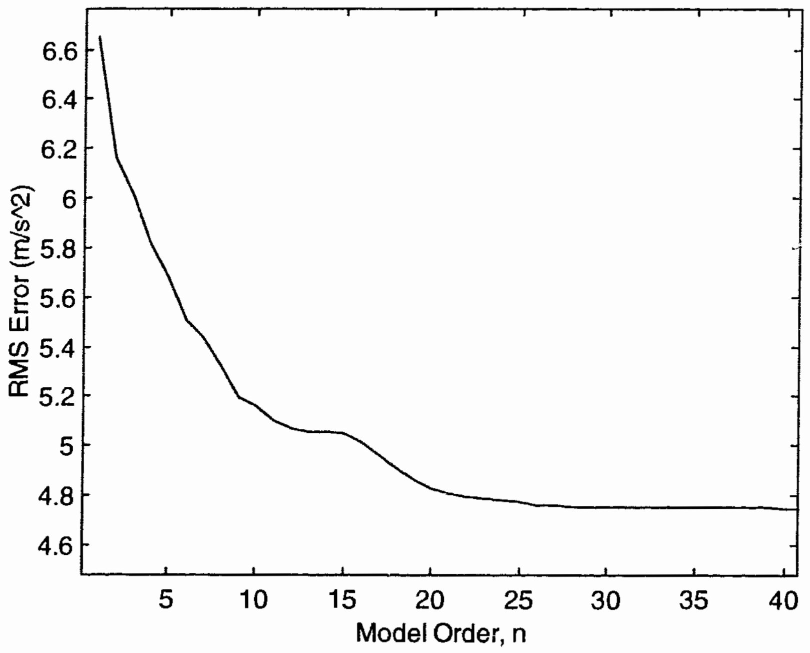

Figure D.1 Error vs Model order for ARX model indentified using least squares estimation.

APPENDIX E IMPLEMENTATION OF THE BRITISH STANDARD 6841 FILTER AND DYNAMIC RESPONSE INDEX MODEL

The British Standard 6841 model is a second order linear model of the form

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \qquad \text{(E.1)}$$

where ω_n is the undamped natural frequency, ζ is the damping ratio, and s is the complex variable used in the Laplace transform.

This model was implemented on MatLab as an 8'th order recursive infinite impulse response (IIR) filter. This implementation was chosen because only the model's magnitude frequency response, and not the model parameters, were available. Thus, instead of converting the continous-time model into a digital version, I decided to design a digital filter that exhibited the same frequency response characterstics.

This filter was designed using the **yulewalk** function provided by MatLab. **yulewalk** designs recursive IIR filters using a least squares fit to the specified frequency response. Specifically, it computes the filter coefficients using the modified Yule-Walker equations (Friedlander and Porat, 1984). Details of the filter design algorithm can be found in the MatLab Signal Processing Toolbox manual.

It was found that a filter order of 8 was required to adequately fit the desired frequency response. The designed filter was of the form

$$\frac{B(z)}{H(z)} = \frac{\sum_{i=0}^{8} b_i z^{-i}}{1 + \sum_{i=1}^{8} a_i z^{-i}}, \qquad \text{(E.2)}$$

with the following coefficients:

b = [0.4088 -0.3016 0.1048 -0.1465 -0.0640 0.0066 0.0921 0.1310 -0.0210 0.0026 0.0077];
a = [-1.4756 1.0700 -0.8198 0.3260 -0.0656 -0.1629 0.4302 -0.3084 0.1141 0.0191].

The stability of the filter was checked by verifying that the poles were contained within the unit circle of the z-plane.

The DRI model is also specified in the form of Equation E.1. The digital version of the model was implemented on MatLab as a second order, recursive IIR filter, whose coefficients were identified using the Steiglitz-McBride algorithm (Steiglitz and McBride, 1965). This algorithm is provided by the MatLab Signal Processing Toolbox under the name **stmcb**.

The resulting DRI filter is described by

$$H(z) = \frac{0.1252}{1 - 1.7372z^{-1} + 0.8552z^{-2}}. \tag{E.3}$$

The stability of the filter was ensured by verifying that the poles were inside the unit circle of the z-plane.

References

Steiglitz, K. and L.E. McBride (1965) "A technique for the identification of linear systems," *IEEE Transactions on Automatic Control,* vol. AC-10, 461-464.

Friedlander, B. and B. Porat (1984) "The modified Yule-Walker method of ARMA spectral estimation," IEEE Transaction on Aerospace Electronic Systems, No. 2, 158-173.