THE NUMERICAL SOLUTION OF LINEAR ORDINARY DIFFERENTIAL EQUATIONS BY FEEDFORWARD NEURAL NETWORKS

Andrew J. Meade, Jr.* and Alvaro A. Fernandez
Rice University
Department of Mechanical Engineering
and Materials Science
Houston, Texas, 77251-1892, USA
Phone: (713) 527-8101 ext. 3590
email: meade@rice.edu

---

* Recipient of correspondence, email: meade@rice.edu
This work is supported under NASA grant NAG 1-1433.

THE NUMERICAL SOLUTION OF LINEAR ORDINARY DIFFERENTIAL EQUATIONS BY FEEDFORWARD NEURAL NETWORKS

Andrew J. Meade, Jr. and Alvaro A. Fernandez
Rice University
Department of Mechanical Engineering
and Materials Science

Key Words: artificial neural networks, neural computation, differential equations, basis functions

ABSTRACT

It is demonstrated, through theory and examples, how it is possible to construct directly and noniteratively a feedforward neural network to approximate arbitrary linear ordinary differential equations. The method, using the hard limit transfer function, is linear in storage and processing time, and the L_2 norm of the network approximation error decreases quadratically with the increasing number of hidden layer neurons. The construction requires imposing certain constraints on the values of the input, bias, and output weights, and the attribution of certain roles to each of these parameters.

All results presented used the hard limit transfer function. However, the noniterative approach should also be applicable to the use of hyperbolic tangents, sigmoids, and radial basis functions.

THE NUMERICAL SOLUTION OF LINEAR ORDINARY DIFFERENTIAL EQUATIONS BY FEEDFORWARD NEURAL NETWORKS

Andrew J. Meade, Jr.* and Alvaro A. Fernandez†
Department of Mechanical Engineering
and Materials Science,
Rice University
Houston, TX 77251-1892, USA

1 INTRODUCTION

The rapidly growing field of connectionism is concerned with parallel, distributed, and adaptive information processing systems. This includes such tools as genetic learning systems, simulated annealing systems, associative memories, and fuzzy learning systems. However, the primary tool of interest is the artificial neural network.

The term Artificial Neural Network (ANN) refers to any of a number of information processing systems that are more-or-less biologically inspired. Generally speaking they take the form of directed-graph type structures [1] whose nodes perform simple mathematical operations on the data to be processed. Information is represented within them by means of the numerical weights associated with the links between nodes. The mathematical operations performed by these nodes, the manner of their connectivity to other nodes, and the flow of information through the structure is patterned after the general structure of biological nervous systems. Much of the terminology associated with these systems is also biologically inspired; thus networks are commonly said to be "trained," not programmed, and to "learn" rather than model.

ANNs have proven to be versatile tools for accomplishing what could be termed higher order tasks such as sensor fusion, pattern recognition, classification, and visual processing. All these applications are of great interest to the engineering field, however, present ANN applications have created the opinion that networks are unsuited for use in tasks requiring accuracy and precision, such as mathematical modelling and the physical analysis of engineering systems. Certainly the biological underpinnings of the neural network concept suggest networks would perform best on tasks at which biological systems excel, and worse or not at all at other tasks.

---

* Assistant Professor. Recipient of correspondence, email: meade@rice.edu
† Research Assistant
This work is supported under NASA grant NAG 1-1433.

Contrary to this opinion the authors believe that continued research into the approximation capabilities of networks will show that ANNs can be as numerically accurate and predictable as conventional computational methods. It is also believed that in viewing ANNs as numerical tools, advances in training and analyses of existing architectures can be made. In a more immediate sense, benefits of this approach will enable the neural network paradigm, with all of its richness of behavior and adaptivity, to be mated to the more purely computational paradigms of mathematically oriented programming used by engineers in modelling and analysis.

Presently, the most popular application of ANNs in science and engineering is the emulation of physical processes by a feedforward artificial neural network (FFANN) architecture using a training algorithm. Because this paradigm requires exposure to numerous input-output sets, it can become memory intensive and time consuming. Considerable effort may be saved if the mathematical model of a physical process could be directly and accurately incorporated into the FFANN architecture without the need of examples, thereby shortening or eliminating the learning phase.

As a consequence, our efforts concentrate on developing a general noniterative method in which the FFANN architecture can be used to model accurately the solution to algebraic and differential equations, using only the equation of interest and the boundary and/or initial conditions. A FFANN constructed by this noniterative, and numerically efficient, method would be indistinguishable from those trained using conventional techniques.

A number of researchers have approached the problem of approximating the solution of equations, whether algebraic or differential, in a connectionist context. Most of these attempts have proceeded from an applications oriented viewpoint, where the solution of the equations is viewed as a new area in which to apply conventional connectionist paradigms. Specifically, the solution of the equations is transformed into an optimization problem by defining some objective function and its associated metric. The problem is then placed in a connectionist structure and solutions pursued by means of an optimization based training algorithm.

The majority of the work in connectionist equation solving is concerned with the solution of algebraic equations, such as the solution of an arbitrary linear system of equations. Takeda and Goodman [2] and Barnard and Casasent [3] use variations on the Hopfield network to solve linear systems of equations. The linear systems to be solved are expressed as the minimization of a quadratic function, and the weights of the Hopfield net are trained until the function is minimized. These algorithms have some interesting features, among them inherent high parallelism with very simple components and the promise of high speeds. However, they are generally computationally expensive to implement once the training time is taken into account [2], and they often rely on specialized hardware such as optical implementation to acquire the high speeds. In addition, representation of real numerical values can be somewhat involved if a discrete Hopfield model is used, as in reference [2].

In contrast, Wang and Mendel [4] use a feedforward architecture to solve linear algebra problems, with connection structure constrained by the nature of the problem being solved. For a particular matrix or algebraic system, the constrained network is trained to associate the input (for example, an input matrix A) with the output (the LU decomposition of A). The solution is then read from the connection weights of the trained network. The authors successfully train these networks in problems of LU decomposition, linear equation solving, and singular value decomposition. The algorithm can be implemented on more standard hardware than those based on the Hopfield model and is easily parallelizable according to the authors.

Lee and Kang [5] have solved first order differential equations using the optimization network paradigm. The differential equation is first discretized using finite difference techniques, and the resulting algebraic equations are transformed into an energy function to be minimized by the net. Unlike the energy functions arising from the formulation of the linear algebraic problem of [2] and [3], the energy functions to be minimized by the net in the solution of a general partial differential equation are not guaranteed to be quadratic in nature. Thus Lee and Kang use modified Hopfield nets able to minimize arbitrary energy functions. They then formulate the energy quantity to be minimized for a general order nonlinear partial differential equation on a unit hypercube, but only the solutions to first order linear differential equations are presented. The results are quite good, although the solutions produced show a tendency to oscillate about the exact solution in some fairly simple cases. Lee and Kang suggest an optical implementation for more complex problems.

The approach of this paper is markedly different from the previously cited works. Our efforts concentrate on developing a general, numerically efficient, noniterative method in which the FFANN architecture can be used to model accurately the solution to algebraic and differential equations, using only the equation of interest and the boundary and/or initial conditions. A FFANN constructed by this noniterative method would be indistinguishable from those trained using conventional techniques.

Progress in the noniterative approach should also be of interest to the connectionist community, since the approach may be used as a relatively straightforward method to study the influence of the various parameters governing the FFANN. When taken from the viewpoint of network learning, applying an FFANN to the solution of a differential equation can effectively uncouple the influences of the quality of data samples, network architecture, and transfer functions on the network approximation performance. Insights provided by this uncoupling may help in the development of faster and more accurate learning techniques [6]. We have borrowed a technique from applied mathematics, known as the method of weighted residuals (MWR), and shown how it can be made to operate directly on the net architecture. The method of weighted residuals is a generalized method for approximating functions, usually from given differential equations, and will be covered in detail in Section 4 of this paper.

The development of the noniterative approach to the determination of the FFANN weights can be likened to the development of a new numerical technique. When evaluating the capabilities of a new numerical method for science and engineering, it is prudent to apply it first to the solution of simple equations of practical interest and known behavior. This same approach will be used in our evaluation of the noniterative method for FFANNs. A simple feedforward network using a single input and output neuron with a single hidden layer of processing elements utilizing the hard limit transfer function is constructed to accurately approximate the solution to a first and second order linear ordinary differential equation. It will also be demonstrated how the error of the constructed FFANNs can be predicted and controlled.

2 FUNCTION APPROXIMATION

Function approximation theory, which deals with the problem of approximating or interpolating a continuous, multivariate function, is an approach that has been considered by other researchers [7]-[9] to explain supervised learning and ANN behavior in general. There are four sets of parameters that influence the approximation performance of the simple FFANN architecture of Fig. 1; (1) the input weights, (2) the bias weights, (3) the transfer functions, and (4) the output weights. In an effort to provide guidelines in the interpretation of the mathematical roles of each set of parameters, function approximation theory is investigated.

The most common method of function approximation is through a functional basis expansion. A functional basis expansion represents an arbitrary function $y(x)$ as a weighted combination of linearly independent basis functions (Υ) that are functions of the variable x,

$$y(x) = \sum_{q=1}^{T} \Upsilon_q(x) c_q \tag{1}$$

where T is the total number of basis functions and c_q are the basis coefficients. This is analogous to representing arbitrary multidimensional vectors by a weighted sum of linearly independent basis vectors. A common example of a functional basis expansion is the Fourier series, which uses trigonometric basis functions. Other basis functions can be used as well; the values of the coefficients c_q will of course vary depending on the basis functions selected.

It is important to note that the basis expansion can also be viewed as an interpolation scheme, where the nature of the interpolation depends on the kind of basis function used. In fact, function approximation literature often uses the terms "basis functions" and "interpolation functions" interchangeably, with the former sometimes preferred by mathematicians and the latter by engineers. Therefore, a basis employing linear polynomials can be said to exhibit a linear interpolation of the approximated function.

Examination of Eq.(1) reveals it to be a scalar product of the basis functions and expansion coefficients. Note that the equation makes no mention of the dependence of the basis functions on any parameter other than the independent variable x. Similarly, the mathematical operations performed by the simple FFANN of Fig. 1 can be interpreted as a scalar product where the transfer functions act as basis functions and the output weights act as basis expansion coefficients. This insight is behind many of the attempts to prove that finite sums of transfer functions, such as sigmoids, can be used to approximate arbitrary continuous functions [10], [11].

The link between a FFANN and functional basis expansion assigns a mathematical role both to the output weights and to the transfer functions, two of the parameter sets mentioned earlier as affecting the performance of the network.

2.1 Transfer Functions as Basis Functions

Let us evaluate the potential usefulness of network transfer functions as basis functions. For the remainder of this paper we will use the hard limit of Fig. 2 as the transfer function. We have based this choice on the simplicity of the function and the ease by which it can be implemented in software and hardware [12].

The hard limit can be modelled by the following equations:

$$\begin{aligned} \Upsilon_q &= -1.0 \quad \text{for} \quad \xi_q < -1.0 \\ \Upsilon_q &= \xi_q(x) \quad \text{for} \quad -1.0 \leq \xi_q \leq 1.0 \\ \Upsilon_q &= +1.0 \quad \text{for} \quad \xi_q > 1.0 \end{aligned} \tag{2}$$

where ξ_q is some linear function of the independent variable x.

In the approximation of a function by basis expansion, the bases that are chosen for the expansion must either span the space inhabited by the function to be approximated or a subspace thereof. In theory this leads to a very large choice of possible bases. The commonly used basis functions in function approximation literature are said to be either local (exhibiting a nonzero value over a limited part of the domain only) or global (nonzero almost everywhere in the domain they are to span).

The hard limit transfer functions may be viewed as global linear polynomial basis functions defined in a piecewise manner with the aid of the transformation into ξ_q. An appeal to the interpolatory nature of the basis expansion explains the distribution of the ramp functions across the domain as being necessary in order to improve the quality of the interpolation. The distribution of the hard limits is accomplished using the input and bias weights, as will be shown later.

A question that must be answered is how effective are the hard limits as basis functions. To help in answering this question we use the Weierstrass Approximation Theorem which can be stated as follows: *Given any interval $[a, b]$, any real number $\epsilon > 0$, and any real valued function $f(x)$ continuous on $[a, b]$, there exists a polynomial $p(x)$ such that $||f - p|| < \epsilon$* , where $||\cdot||$ denotes some function norm. A constructive proof of this theorem is found in reference [13]. A welcome conclusion from this theorem is that the hard limits should at least theoretically allow representation of arbitrary functions and hence arbitrary mappings. Yet Weierstrass provides no clue as to the performance, let alone optimality, of the infinite number of polynomial functions that exist. The fact that it is possible to represent arbitrary functions with some primitive does not mean that it is either practical or desirable to do so.

Prenter [14] defines a good polynomial basis as one which allows for a unique solution to the expansion coefficient problem,

$$f(x_i) = p(x_i), \quad (i = 0, 1, ..., N), \tag{3}$$

where the N points used to satisfy the function to be approximated are commonly termed knots. A good polynomial basis should also provide smooth interpolation of the approximated function between the knots. A particular family of polynomials that provides these characteristics is the family of Lagrange polynomials.

Prenter examines the behavior of global Lagrange polynomials and discusses certain weaknesses shared by all global basis functions:

1. The algebraic systems they generate to solve for the expansion coefficients often suffer from linear dependence problems as N increases (ill conditioning).

2. They often suffer from poor approximation (so-called "snaking") between knots, particularly if the function to be approximated has high gradients, noise, or discontinuities.

2.2 Lagrange Polynomial Splines

Because of the disadvantages of global basis functions, piecewise polynomial splines based on the Lagrange polynomials are often used. Splines are polynomial curves that extend over a limited number of knots and which together provide varying degrees of interpolation between their knots. The most common polynomial spline is the Chapeau or "hat" function as shown in Fig. 3. This is sometimes referred to in the literature as a first-order Lagrange polynomial (also known as a first order Lagrangian interpolating function or shape function). Its formula is:

$$\Phi_i = \frac{(x - x_{i-1})}{(x_i - x_{i-1})}, \quad x_{i-1} \le x \le x_i$$

$$\Phi_i = \frac{(x_{i+1} - x)}{(x_{i+1} - x_i)}, \quad x_i \le x \le x_{i+1}$$
$$\text{otherwise} \quad \Phi_i = 0, \quad x < x_{i-1} \quad \text{or} \quad x > x_{i+1} \tag{4}$$

Note that the polynomial varies between the values of 0 and 1.

Figure 4 shows, for a problem domain $[x_1, x_N]$ discretized with $N = 3$ knots (x_1, x_2, and x_3), how the hat functions must be distributed to provide a linear interpolation. Notice that for any value of x in the domain, the sum of all hat functions will equal a value of 1. This means that the hat functions distributed in this manner will represent a constant accurately which is the first, and most important, test of a good interpolation scheme. An additional benefit of the distribution with respect to computational efficiency, is that the hat functions are linearly independent. This will be of importance in the determination of the basis expansion coefficients.

The distribution of the hat functions is justified also on purely mathematical grounds. Considering that the hat functions are being used as a functional basis, and that the function being represented has been discretized at N knots, it is clear there are N degrees of freedom for the function being represented. This implies N dimensions in the function subspace being used to represent the function, necessitating N hat functions in the domain, one centered at each knot.

This particular polynomial spline does not suffer from the common maladies of the global Lagrange polynomials. Namely:

1. The algebraic system generated to evaluate the expansion coefficients is very sparse and well-conditioned. In this sense a sparse system means that less than the total number (N) of unknowns appear in each of the N algebraic equations. This is because the splines exist only locally in the problem domain and so only a limited number of them are involved in representing the function at any one value of x.

2. The interpolation between knots is linear and well-behaved (no snaking) and can easily handle high gradients or noise.

3. Since the maximum value of the spline is 1, the values of the expansion coefficients are identical to the values of the approximated function at the knots.

One drawback, however, is that the approximation is discontinuous in the first derivative. In the language of functional analysis, this spline is found in $C^0[a, b]$, a subspace of $L_2[a, b]$.

3 CONSTRAINT OF NETWORK PARAMETERS

As previously mentioned, the efforts of other researchers in the application of ANNs to the solution of differential equations involve minimizing an energy function related to the differential equation on a Hopfield-type network. Supervised learning and solution of differential equations by minimization suffer from similar problems: long convergence times and the possibility of converging to a suboptimal state or not at all. It is crucial to realize that these difficulties are not exclusively due to the algorithms being employed. Part of the difficulty is that the problem as given is still underdetermined: there are not enough constraints on the parameters in the system to admit a reasonable number of solutions. As a result, the error surface created is likely to be complex, and the algorithms may experience problems traversing it.

One obvious solution to this difficulty is to impose constraints on the parameters, either arbitrarily or following some rationale, and changing the nature of the error surface into one that is more tractable but still approximates the desired function at the error minimum. Ideally, one would like to constrain the parameters to the extent that optimization is not necessary and the remaining unknown parameters can be determined uniquely.

3.1 Input Weights and Bias Weights

Using the notation of Fig. 1, where α_q represents the input weight and θ_q represents bias weight, the output (o_q) of a specific processing element q, can be written in equation form:

$$o_q = \Upsilon_q(\alpha_q x + \theta_q) = \Upsilon_q(\xi_q) \quad \text{where} \quad \xi_q = \alpha_q x + \theta_q \tag{5}$$

While α_q and θ_q are constants, ξ_q is a variable that is linearly dependent on the input x. By inspection of Eq.(5), the input weight α_q appears to act as a scaling coefficient for x. Alternately, the combination of the input weight and bias can be viewed as transforming the input variable to a new space ξ_q, where ξ_q is specific to the qth processing element. This linear transformation of x into ξ_q is similar to the unscaled shifted rotations of reference [11].

Define a point on the x-axis x_q as the "center" of the transfer function Υ_q such that

$$\alpha_q x_q + \theta_q = 0 \quad \text{or} \quad \theta_q = -\alpha_q x_q$$

From this equation it is seen that the bias weight allows the origin of each transfer function ($\xi_q = 0$) to be set in the independent variable space x. The input and bias weights allow each transfer function to be scaled differently and centered at different locations in the independent variable space x (Fig. 5).

3.2 Transforming the Transfer Functions of the Neural Network Into Splines

From a purely function approximation perspective, the aptitude of the hard limit as a basis function is relatively low. Since it is a global polynomial, it would be expected to produce a full system of algebraic equations for the solution of the basis coefficients and would be sensitive to noise. In addition, considering that the hard limit is zero at its center and has a nonzero value throughout the remainder of the problem domain, it is expected that the basis coefficients that will be generated will not have a direct relation to the values of the function being approximated. This property is often termed as nonlocal representation in the connectionist literature [15].

Non-local representation is a condition that is often viewed favorably in the connectionist literature, as indicating "redundancy," and is sometimes discussed as a characteristic unique to networks. In fact it is a common feature of any functional basis expansion utilizing global basis functions, and its net effect is often negative as it couples the values of all the basis coefficients in the approximation of a function, making the system more difficult to solve and complicating the imposition of boundary conditions.

A considerable number of advantages, as discussed in Section 2.2, may be obtained if the hard limit is transformed into the hat function. Consider a one-dimensional domain as in Fig. 6, with two hard limit functions centered in neighboring intervals as shown. If the function centered between x_i and x_{i+1} is multiplied by -1 and added to the second unaltered function (Fig. 7), it is clear by inspection that the result will be a hat function with a maximum value of 2. Additionally, if the hard limits in the figure are scaled between -0.5 and $+0.5$ by suitable output weights, the resulting hat function is indistinguishable from that used in the function approximation literature. The transformation of hard limits to hat functions can be described by the following equation:

$$\Upsilon_i^A - \Upsilon_i^B = 2\Phi_i, \quad (i = 1, 2, ..., N) \tag{6}$$

where the superscripts of the hard limits, A and B, refer to adjoining intervals $x_{i-1} \leq x \leq x_i$ and $x_i \leq x \leq x_{i+1}$ respectively. The functions Υ_i^A and Υ_i^B are defined as zero at the midpoints of their respective intervals. A consequence of this formulation is that the number of hard limits can be linked to the number of hat functions desired. We will require twice as many hard limits as hat functions ($T = 2N$).

3.3 Constraint of Input, Bias, and Output Weights

Let us now derive the remaining specific constraints required to equate the hard limit function representation (Eq.(1)) to one using the hat functions:

$$y_a(x) = \sum_{q=1}^{T} \Upsilon_q(x) c_q = \sum_{i=1}^{N} \Phi_i(x) w_i \tag{7}$$

where the hat functions Φ_i are defined as having a value of 1 at the knot x_i, as per Eq.(4) and w_i are the basis coefficients associated with the hat functions.

Let us discretize the problem domain using the knots x_i in the following manner:

$$a = x_1 < x_2 < \ldots < x_{N-1} < x_N = b$$

We will label this discretization as the problem mesh. To generate the appropriate hat functions at the boundaries, two auxiliary knots x_0 and x_{N+1} are required such that $x_0 < a$, and $x_{N+1} > b$.

Using the constraint, $T = 2N$, we may rewrite Eq.(1) as

$$\sum_{q=1}^{T=2N} \Upsilon_q(x) c_q = \sum_{q=1}^{N} \Upsilon_q(x) c_q + \sum_{q=N+1}^{2N} \Upsilon_q(x) c_q \tag{8}$$

where the summations of the right hand side can be rewritten without loss of generality as

$$\begin{aligned} \sum_{q=1}^{N} \Upsilon_q c_q &= \sum_{i=1}^{N} \Upsilon_i^A u_i \\ \sum_{q=N+1}^{2N} \Upsilon_q c_q &= \sum_{i=1}^{N} \Upsilon_i^B v_i \end{aligned} \tag{9}$$

therefore

$$\sum_{q=1}^{T} \Upsilon_q(x) c_q = \sum_{i=1}^{N} \Upsilon_i^A u_i + \sum_{i=1}^{N} \Upsilon_i^B v_i \tag{10}$$

where

$$\begin{aligned} \Upsilon_i^A &= -1.0 \quad \text{for} \quad x < x_{i-1} \\ \Upsilon_i^A &= \xi_i^A(x) \quad \text{for} \quad x_{i-1} \le x \le x_i \\ \Upsilon_i^A &= 1.0 \quad \text{for} \quad x > x_i \\ \Upsilon_i^A &= 0 \quad @ \quad x = \frac{x_{i-1} + x_i}{2} \end{aligned} \tag{11}$$

and where

$$\begin{aligned} \Upsilon_i^B &= -1.0 \quad \text{for} \quad x < x_i \\ \Upsilon_i^B &= \xi_i^A(x) \quad \text{for} \quad x_i \le x \le x_{i+1} \\ \Upsilon_i^B &= 1.0 \quad \text{for} \quad x > x_{i+1} \\ \Upsilon_i^B &= 0 \quad @ \quad x = \frac{x_i + x_{i+1}}{2} \end{aligned} \tag{12}$$

If we constrain the output weights such that

$$u_i = -v_i = \frac{w_i}{2} \tag{13}$$

then substituting this relation into Eq.(10) gives:

$$\sum_{i=1}^{N} \Upsilon_i^A u_i + \sum_{i=1}^{N} \Upsilon_i^B v_i = \left(\sum_{i=1}^{N} \Upsilon_i^A - \sum_{i=1}^{N} \Upsilon_i^B\right) \frac{w_i}{2} = \sum_{i=1}^{N} (\Upsilon_i^A - \Upsilon_i^B) \frac{w_i}{2}$$

But from Eq.(6), $\Upsilon_i^A - \Upsilon_i^B = 2\Phi_i$; therefore

$$\sum_{i=1}^{N}(\Upsilon_i^A - \Upsilon_i^B)\frac{w_i}{2} = \sum_{i=1}^{N}(2\Phi_i)\frac{w_i}{2} = \sum_{i=1}^{N}\Phi_i(x)w_i$$

which is the basis expansion for $y_a(x)$ in terms of hat functions (Eq.(7)).

It is relatively straightforward to derive actual formulae for the input, bias, and output weights. From Eq.(9) and Eq.(13) the output weights are given by

$$\begin{aligned} c_i = u_i &= \frac{w_i}{2}, \quad (i = 1, 2, ..., N) \\ c_{i+N} = v_i &= \frac{-w_i}{2}, \quad (i = 1, 2, ..., N) \end{aligned} \tag{14}$$

where the numbering of the weights with respect to the actual net architecture can be in any order. The result is simple. The ith pair of the $T = 2N$ output weights in the FFANN must be set to the values u_i and v_i, that is, $\frac{1}{2}$ and $-\frac{1}{2}$ respectively, of the ith basis expansion coefficient w_i for $(i = 1, 2, ..., N)$.

To derive the input and bias weight formulae refer again to Figs. 3, 5, 6 and 7 and also to Eqs.(4), (11), and (12). It is clear from Fig. 6 that the hard limits must be placed in the problem space in such a manner that their linear behavior ($\xi_i^A(x)$ and $\xi_i^B(x)$ respectively) occurs across the appropriate interval. Therefore by inspection,

$$\begin{aligned} \xi_i^A(x) &= \frac{2(x - x_{i-1})}{(x_i - x_{i-1})} - 1, \quad x_{i-1} \le x \le x_i \\ \xi_i^B(x) &= \frac{2(x - x_i)}{(x_{i+1} - x_i)} - 1, \quad x_i \le x \le x_{i+1}, \\ & \qquad\qquad\qquad\qquad i = 1, 2, ..., N \end{aligned} \tag{15}$$

Since $\xi_i^A(x) = \alpha_i^A x + \theta_i^A$ and $\xi_i^B(x) = \alpha_i^B x + \theta_i^B$, we can calculate the value of the input and bias weights using Eq.(15). Therefore

$$\alpha_i^A = \frac{2}{(x_i - x_{i-1})} \quad , \quad \theta_i^A = -\frac{2x_{i-1}}{(x_i - x_{i-1})} - 1 \tag{16}$$

and

$$\alpha_i^B = \frac{2}{(x_{i+1} - x_i)} \quad , \quad \theta_i^B = -\frac{2x_i}{(x_{i+1} - x_i)} - 1 \tag{17}$$

3.4 Constraint Results

The results of this section can be summarized as follows:

1. Two hard limits can be added to form a hat function spline with a maximum value of two. With appropriate scaling, the canonical hat function can be formed.

2. By using $T = 2N$ hard limits, where N is the number of hat functions and knots in the problem domain $[a, b]$, it is shown that by appropriately constraining the input, bias, and output weights the feedforward network can be made to form a hat function based functional basis expansion everywhere in the problem domain.

3. The input and bias weights can be computed from Eq.(16) and Eq.(17) given the problem mesh.

4. The values of the expansion coefficients, w_i, can be evaluated using the hat function basis Φ. This eliminates all the problems associated with using the hard limits as basis functions directly, while preserving the architecture of the net. From Eq.(14) the values of the coefficients w_i are used to calculate the output weights c_q that will complete the construction of the feedforward network.

Figure 8 shows the resulting architecture for six processing elements used to form three hat functions for three knots plus two auxiliary knots, as in Figure 4. Figure 9 shows an alternate network architecture that can be constructed using two hidden layers, where the second hidden layer has linear transfer functions. Although the number of processing elements T is almost the same for this architecture ($T = 2N + 1$), only $N + 1$ of the processing elements have hard limit transfer functions associated with them. Note also the connections between the first and second hidden layer are local and repeating. The economy on the use of hard limits and the local connections between the first and second hidden layer should make this architecture particularly attractive for hardware implementation. For the remainder of this paper we will refer to the first architecture, with the understanding that the transformation to the second architecture is trivial.

The implication of Section 2 was that estimation of the weights of a FFANN ("training") could be viewed as the equivalent problem of finding a suitable basis expansion for the relationship to be modelled by the net. This section adds the additional consideration that for suitable transfer functions and constraints on the input and bias weights, the basis expansion being sought can be a classical expansion in polynomial splines. Thus training in the traditional sense is eliminated in this approach. The construction of the network has been reduced to a question of finding the appropriate basis expansion coefficients w_i and then setting the values of the pairs of output weights u_i and v_i from Eq.(14). The values of the input and bias weights are fixed for a given number of knots in the problem domain and are effectively uncoupled from the problem of finding the output weights.

The computation of the expansion coefficients w_i can be made from a variety of numerical methods. As mentioned in Section 1, we will use the method of weighted residuals [16] to determine the output weight values.

4 DETERMINATION OF OUTPUT WEIGHTS: THE METHOD OF WEIGHTED RESIDUALS

The problem of finding the expansion coefficients can be expressed in terms quite familiar to the connectionist: given a differential equation whose solution is a function y, and an approximation to the solution expressed as a basis expansion, y_a, find expansion coefficients such that the error of the approximation is small in terms of some norm in the domain $\mathcal{D}$.

To be more precise, if spatial variables are described by the vector $\mathbf{s}$ and $L(y)$ is some differential operator, then the substitution of Eq.(18)

$$y_a(\mathbf{s},t) = \sum_{i=1}^{N} \Phi_i(\mathbf{s})\ w_i(t) \qquad (\text{ where } \Phi_i(\mathbf{s}) \text{ are basis functions }) \tag{18}$$

into the time dependent governing differential equation $L(y) = g(\mathbf{s},t)$, results in the equation $L(y_a) - g(\mathbf{s},t) = R(w_1, ..., w_N, \mathbf{s}, t)$, where R is some function of nonzero value that can be described as the error or the differential equation residual. In addition, if Eq.(18) is subject to the initial and boundary conditions of the governing differential equation we may write

$$I(y_a) = R_I(w_1, ..., w_N, \mathbf{s}) \quad \text{and} \quad B(y_a) = R_B(w_1, ..., w_N, t) \text{ respectively.}$$

In principle, basis coefficients $w_i(t)$ can be found so that R becomes small in terms of some norm over the problem domain $\mathcal{D}$ as $N \to \infty$.

Let us require that y_a satisfy the initial and boundary conditions exactly so that $R_I = R_B = 0$. The basis coefficients $w_i(t)$ that satisfy the differential equation can be determined by requiring that the equation residual R be multiplied or weighed by a function $f(\mathbf{s})$ (often called the weight function or test function), integrated over the problem domain $\mathcal{D}$ and set to zero,

$$\int_{\mathcal{D}} fR\ d\mathcal{D} = (f, R) = 0 \tag{19}$$

where (f, R) is the inner product of functions f and R. It is from Eq.(19) that the method of weighted residuals derives its name. It may be noted that Eq.(19) is closely related to the weak form of the governing equation

$$\int_{\mathcal{D}} fR\ d\mathcal{D} = \int_{\mathcal{D}} f(L(y_a) - g(\mathbf{s},t))\ d\mathcal{D} = 0 \tag{20}$$

This relation to the weak form has the benefit of allowing discontinuities in the exact solution [17], which is particularly advantageous when approximating the solution to problems with large gradients or discontinuities.

Since linearly independent relationships are needed to solve for the basis coefficients of Eq.(19) it is clear that f must be made up of linearly independent functions, f_k. By letting $k = 1, ..., N$ a system of N equations for the basis coefficients is generated. For a time dependent case a system of ordinary differential equations in t result. For the steady-state case the basis coefficients are constants and a system of algebraic equations is generated.

Different choices to the weight function f_k give rise to different computational methods that are subclasses of MWR. Some of these methods are:

1. The subdomain method. The computational domain is discretized into N subdomains $\mathcal{D}_k$, which may overlap where

$$f_k = 1 \text{ within } \mathcal{D}_k \quad \text{and} \quad f_k = 0 \text{ outside } \mathcal{D}_k \ , \quad \text{for} \ \ k = 1, ..., N.$$

   The subdomain method is identical to the finite volume method when evaluating Eq.(19). With the subdomain method, Eqs.(18) and (20) provide the appropriate framework for enforcing conservation properties in the governing equation, both locally and globally.

2. The collocation method. The weight functions are set to

$$f_k = \delta(\mathbf{s} - \mathbf{s}_k), \quad \text{for} \ \ k = 1, ..., N$$

   where δ is the Dirac delta function. Substitution of this relation into Eq.(19) gives

$$\int_{\mathcal{D}} \delta(\mathbf{s} - \mathbf{s}_k) \ R \ d\mathcal{D} = R(w_1, w_2, ..., \mathbf{s}_k) = 0$$

   Since the finite difference method requires the solution of the differential equation only at nodal points, one can interpret the finite difference method as a collocation method without the use of an approximate solution y_a.

3. The method of moments. The weight functions are chosen from a set of linearly independent functions such that successively higher moments of the equation residual are required to be zero. For a one dimensional problem we have

$$f_k = x^k, \quad \text{for} \ \ k = 0, ..., N-1$$

   Substitution of this relation into a one dimensional form of Eq.(19) gives

$$\int_a^b x^k R \ dx = 0, \quad \text{for} \ \ k = 0, ..., N-1$$

4. The least-square method. The weight functions are set to

$$f_k = \frac{\partial R}{\partial w_k}, \quad \text{for} \ \ k = 1, ..., N$$

where w_k are the basis coefficients. The application of f_k to Eq.(19) is identical to finding the minimum to the square of the residual summed over the problem domain i.e.,

$$\frac{\partial}{\partial w_k}\int_{\mathcal{D}} R^2 \; d\mathcal{D} = 0$$

5. The generalized Galerkin method. The weight functions are set to

$$f_k = \Psi_k(\mathbf{s}), \quad \text{for} \quad k = 1, ..., N$$

where $\Psi_k(\mathbf{s})$ are analytic functions similar to the bases but modified with additional terms to satisfy the boundary and/or initial conditions. The finite element and spectral methods can be considered subclasses of the generalized Galerkin method. Galerkin based methods are considered particularly accurate if basis functions are the first N members of a complete set since Eq.(19) indicates that the residual is orthogonal to every member of the complete set. Consequently as N tends to infinity the approximate solution y_a will converge to the exact solution y.

It should also be noted that MWR is not limited to cases where initial and boundary conditions are satisfied exactly ($R_I = R_B = 0$). If we require that only the differential equation be satisfied exactly ($R = 0$) we may use MWR to derive boundary methods such as the panel and boundary element methods. The various subclasses of the method of weighted residuals are discussed and compared at great length by Finlayson [16] and Fletcher [18].

By approaching the evaluation of the output weights through the method of weighted residuals we have access to both theoretical and computational results from the most commonly used techniques in the fields of scientific and engineering computing. Each method has advantages and disadvantages depending on the particular application. For this paper we will make use of the generalized Galerkin technique following the observation made by Fletcher that " . . . the Galerkin method produces results of consistently high accuracy and has a breadth of application as wide as any method of weighted residuals " (p. 38 [18]).

Specifically, $f_k(x)$ will be identical to the basis functions used to describe the approximation y_a. This is known as the Bubnov-Galerkin technique [18]:

$$f_k(x) = \Phi_k(x), \quad \text{where} \quad k = 1, 2, ..., N \tag{21}$$

With $f(x)$ so defined, evaluation of the integrals leads to a system of algebraic equations that is usually linear if the differential equation to be approximated is linear. Otherwise the algebraic equations are nonlinear, though still tractable. These equations can be evaluated for a unique solution through standard techniques [19], allowing the evaluation of both linear and nonlinear differential equations.

5 MODEL PROBLEMS

5.1 Example 1: A First Order Linear Ordinary Differential Equation

The most logical and practical model problem in which to first test the accuracy and convergence properties of the network methodology is a first order linear ordinary differential equation with its associated initial condition:

$$\frac{dy}{dx} - y = 0, \quad y(x=0) = 1, \quad 0 \le x \le 1 \tag{22}$$

This equation was also used as an example by reference [5]. A simple feedforward network is constructed to approximate the nontrivial solution to Eq.(22). We select:

1. A single input processing element using a linear transfer function for the independent variable x .
2. A single output processing element using a linear transfer function for the dependent variable approximation y_a.
3. A number of hidden layer processing elements using the hard limit transfer function.
4. A single bias node connected to each of the hidden layer processing elements.

Applying the results of the discussion in Section 3.3, we select the number of knots N to distribute in the problem domain and determine the values of the input and bias weight sets α_i^A, α_i^B, θ_i^A, and θ_i^B respectively. Using hat basis functions Φ, we can now represent the approximation of the dependent function,

$$y_a = \sum_{i=1}^{N} \Phi_i(x) w_i \tag{23}$$

What remains to complete the network are the values of the output weights and consequently, the values of the coefficients w_i.

Substituting Eq.(23) into the model equation results in

$$\frac{dy_a}{dx} - y_a = R \tag{24}$$

The Bubnov-Galerkin technique ($f_k = \Phi_k$) is applied to Eq.(24).

$$(\Phi_k, R) = (\Phi_k, \frac{dy_a}{dx}) - (\Phi_k, y_a) = 0 \quad \text{for } k = 1, ..., N \tag{25}$$

Since

$$\frac{dy_a}{dx} = \sum_{i=1}^{N} \frac{d\Phi_i}{dx} w_i \tag{26}$$

then Eq.(25) becomes

$$\sum_{i=1}^{N}(\Phi_k, \frac{d\Phi_i}{dx}w_i) - \sum_{i=1}^{N}(\Phi_k, \Phi_i w_i) = 0$$

Since the problem is steady, w_i are constants and we can write

$$\sum_{i=1}^{N}(\Phi_k, \frac{d\Phi_i}{dx})w_i = \sum_{i=1}^{N}(\Phi_k, \Phi_i)w_i \quad \text{for } k = 1, ..., N$$

Evaluating the inner products we arrive at the following system of linear algebraic equations:

$$\sum_{i=1}^{N} M_{ki}w_i = b_k, \quad \text{for } k = 1, ..., N \tag{27}$$

where

$$M_{ki} = (\Phi_k, \frac{d\Phi_i}{dx} - \Phi_i) \text{ and } b_k = 0 \quad \text{for } k = 1, ..., N$$

An initial condition is required for a nontrivial solution to the model equation Eq.(22). Using our approximation of y_a

$$y_a(0) = \sum_{i=1}^{N} \Phi_i(0)w_i = y(0) = 1$$

which yields the equation

$$w_1 = y(0) = 1$$

This equation can be incorporated into the coefficient matrix $\mathbf{M}$ and the vector $\mathbf{b}$ of Eq.(27) and therefore

$$M_{11} = 1 \text{ and } M_{1i} = 0 \quad \text{for } i = 2, ..., N$$
$$\text{and } b_1 = y(0) = 1$$

Solution of the modified algebraic system provides the basis expansion coefficients w_i for the differential equation being approximated.

It was mentioned earlier that the type of coefficient matrix created by the weighted residual method (or specifically the Bubnov-Galerkin method) depended on the type of basis functions employed. Analysis of the integral forming the coefficient matrix $\mathbf{M}$ of Eq.(27) shows that

$$M_{ki} = (\Phi_k, \frac{d\Phi_i}{dx} - \Phi_i) = 0, \quad \text{for} \quad i < k - 1 \text{ and } i > k + 1$$

This is due to the local nature of the hat functions as well as their distribution in the problem domain and linear independence. The original matrix $\mathbf{M}$ of Eq.(27) is thus positive definite and tridiagonal. Modification of the matrix by incorporation of the initial condition does not alter these advantages. As a result, the modified algebraic system can be solved in $O(N)$ operations by the Thomas algorithm [20]. With the expansion coefficients known, the output weights of the neural network can be easily computed by Eq.(14).

The determination of the output weights completes the specification of the parameters for the feedforward network that models Eq.(22). The accuracy of the output can be controlled by increasing the number of knots in the problem domain thereby creating additional hat functions. This then translates into additional neurons.

5.2 Results

The exact solution of Eq.(22) with the boundary condition $y(0) = 1$ is

$$y = e^x$$

The feedforward network was constructed using 12 processing elements with a single hidden layer, corresponding to an even spacing of six hat functions from six knots within the problem domain and two auxiliary knots. Figure 10 compares the output of the constructed feedforward neural network (denoted by triangles) with the exact solution (solid line) at 21 equally spaced sample points.

The RMS error of the network is 5.60×10^{-5}, where the RMS error is defined as

$$(E)_{RMS} = \sqrt{\frac{\sum_{j=1}^{S}[y(x_j) - y_a(x_j)]^2}{S}}$$

and where S is the number of samples for the evaluation of the error. The lack of snaking of the results about the exact solution can be attributed to the constraints on the input and bias weights to guarantee linear interpolation, and to the use of the Bubnov-Galerkin method that minimizes the error across the problem domain and not just at discrete points.

The input, bias, and output weights for the network were computed via a conventional Fortran code. All computations were done in double precision. The IMSL subroutine DQDAG [21] was used for numerical integration of the coefficient matrix components and the Thomas algorithm was used for the solution of the tridiagonal linear system of algebraic equations. The program ran on a standard SUN Microsystems Sparc 2 workstation with a run time of approximately 1 second. When optimized for speed, similar programs used in computational physics and engineering applications can run even faster for this problem size on comparable hardware.

5.3 Convergence

The error bounds derived for the Bubnov-Galerkin method lead us to expect that the L_2 norm of the error will be bounded by a quadratic term in the spacing of the knots (grid spacing) [22]. For uniform grid spacing

$$x_{i+1} - x_i = x_i - x_{i-1} = h = \frac{1}{(N-1)}$$

So, for the error $E = y - y_a$ and with the L_2 norm defined as

$$||E|| = \sqrt{\int (y - y_a)^2 dx} \tag{28}$$

we have

$$||E|| \leq Ch^2 \quad \text{where} \quad C \quad \text{is a constant.} \tag{29}$$

In practice the discrete L_2 norm of the error is often used to avoid having to perform the actual integration in Eq.(28):

$$||E|| = \sqrt{h[\sum_{i=1}^{N}(y(x_i) - y_a(x_i))^2]}$$

where the integral has been approximated by a scaled summation; integration schemes of higher accuracy could have been used if deemed necessary. The similarity in the definition of the discrete L_2 error and the RMS error, when the network is sampled at the knots ($S = N$), allows the following relation to be established:

$$(E)_{RMS} = \frac{||E||_{L_2}}{\sqrt{hN}} = \sqrt{\frac{N-1}{N}}\ ||E||_{L_2}$$

Substitution of this relation into Eq.(28) would yield an expected convergence behavior in the RMS norm for the network; it should be slightly greater than quadratic. The L_2 norm is used here due to its wide use in function approximation and the simpler expression of Eq.(30) in terms of the L_2 norm.

If we take the log of $||E||$ then

$$\log(||E||) \leq \log(C) + 2\log(h) \tag{30}$$

Considering the form of Eq.(30), it would be reasonable to expect that a log plot of the L_2 norm of the error versus the log of the grid spacing would yield a straight line with slope of 2.0. A suitable convergence plot of this type is shown in Fig. 11. The size of the mesh spacing was halved starting at $h = 0.2$ to a value of $h = 0.00625$, with the values marked by circles in the figure. The line is found to have an actual slope closer to

2.03, showing slightly greater than expected convergence. This is not completely unusual for some problems, although a slope of 2.0 is the most that can normally be expected from Bubnov-Galerkin utilizing a hat function basis on an arbitrary problem, barring any attempts to force superconvergence at specific points through special techniques outside the scope of this investigation.

5.4 Example 2: A Second Order Linear Differential Equation

A slightly more challenging problem of practical interest is the eigenvalue problem [23]. The eigenvalue problem, familiar to workers in the dynamic systems analysis field, is described by the following second order linear ordinary differential equation with associated boundary conditions

$$\begin{aligned} \frac{d^2y}{dx^2} + \lambda_j y = 0, \quad 0 \leq x \leq 1 \\ y(0) = y(1) = 0 \end{aligned} \tag{31}$$

where the eigenvalues (λ_j) are given by

$$\lambda_j = (j\pi)^2 \quad \text{for} \quad j = 1, 2, 3, ... \tag{32}$$

The solutions $y(\lambda_j)$ corresponding to the eigenvalues are known as the eigenfunctions. An additional condition we impose on the eigenfunction is that they be orthonormalized, which is described using the inner product

$$\begin{aligned} (y(\lambda_j), y(\lambda_k)) = 1 \quad \text{if} \quad j = k \quad \text{and} \\ (y(\lambda_j), y(\lambda_k)) = 0 \quad \text{if} \quad j \neq k \end{aligned} \tag{33}$$

The eigenvalue problem is a good test of the noniterative method not only because of the higher order of differentiation, but also because its homogeneous boundary conditions can easily force a trivial solution $(y_a = 0)$.

Application of the Bubnov-Galerkin method to the problem produces

$$(f_k, \frac{d^2y_a}{dx^2}) + (f_k, \lambda_j y_a) = 0 \quad \text{for} \quad k = 1, ..., N \tag{34}$$

Note that since the basis functions being used are linear, the second derivative is uniformly zero. Therefore, we must reduce the order of the derivatives using integration by parts. We can then rewrite Eq.(34) as

$$(\frac{df_k}{dx}, \frac{dy_a}{dx}) - \lambda_j(f_k, y_a) = f_k \frac{dy_a}{dx}\bigg|_0^1 \quad \text{for} \quad k = 1, ..., N \tag{35}$$

Note that the use of integration by parts brings in boundary derivative information (Neumann conditions). In this example the value of the dependent variable is known at the boundaries (Dirichlet conditions). In using hat functions the boundary derivative term is nonzero only for $k = 1$ and N.

Unlike Example 1, the approximate solution y_a of the eigenvalue problem requires a basis expansion that uses the independent variable x explicitly. This is because the problem with its associated boundary conditions is such that the substitution of a simple expansion like Eq.(23) will lead to a zero known vector ($\mathbf{b}$) and yield only a trivial solution for a nonsingular matrix [19].

The following expansion is used for y_a:

$$y_a(x) = \sum_{i=1}^{N} \Phi_i(x) w_i = C1 \sum_{i=1}^{N} \Phi_i(x_i - \tau_i) \tag{36}$$

where

$$w_i = C1(x_i - \tau_i) \tag{37}$$

Since x_i are the locations of the knots and Φ_i act as linear interpolation functions

$$\sum_{i=1}^{N} \Phi_i x_i = x$$

Therefore Eq.(36) becomes

$$C1(\sum_{i=1}^{N} \Phi_i x_i - \sum_{i=1}^{N} \Phi_i \tau_i) = C1(x - \sum_{i=1}^{N} \Phi_i \tau_i) \tag{38}$$

where $C1$ is some scaling coefficient to be determined when the approximate solution is orthonormalized as in Eq.(33).

Note that with this expansion:

$$y_a(0) = C1(0 - \sum_{i=1}^{N} \Phi_i(0)\tau_i) = 0 \quad \text{and} \quad y_a(1) = C1(1 - \sum_{i=1}^{N} \Phi_i(1)\tau_i) = 0 \tag{39}$$

and therefore

$$\tau_1 = 0 = b_1 \quad \text{and} \quad \tau_N = 1 = b_N$$

This has the effect of creating a nonzero vector $\mathbf{b}$. As a result, a nonsingular coefficient matrix will yield a unique, nontrivial solution.

Applying the expansions to Eq.(35) we obtain the following linear system:

$$\sum_{i=1}^{N} M_{ki}\tau_i = b_k, \quad \text{for} \quad k = 1, ..., N \tag{40}$$

where

$$M_{ki} = (\frac{d\Phi_k}{dx}, (1 - \frac{d\Phi_i}{dx})) - \lambda_j(\Phi_k, (x - \Phi_i))$$
$$\text{and} \quad b_k = 0 \quad \text{for} \quad k = 2, ..., N-1 \tag{41}$$

$$M_{11} = 1 \text{ and } M_{1i} = 0 \quad \text{for } i = 2, ..., N$$
$$\text{and } b_1 = \tau_1 = 0$$

with

$$M_{NN} = 1 \text{ and } M_{Ni} = 0 \quad \text{for } i = 1, 2, ..., N-1$$
$$\text{and } b_N = \tau_N = 1$$

Solution of Eq.(40) yields the values of τ_i.

Applying the orthonormalization requirement to the Bubnov-Galerkin solution yields the following formula:

$$\int_0^1 [C1 \sum_{i=1}^{N} \Phi_i w_i \ \cdot \ C1 \sum_{i=1}^{N} \Phi_i w_i] dx = 1 \tag{42}$$

or

$$\int_0^1 (C1 \sum_{i=1}^{N} \Phi_i w_i)^2 dx = 1 \tag{43}$$

We can replace the integral with an approximate sum,

$$C1^2 \sum_{r=1}^{N} h (\sum_{i=1}^{N} \Phi_i(x_r) w_i)^2 dx = 1 \tag{44}$$

But

$$\Phi_i(x_r) = 1 \quad \text{for} \quad r = i \tag{45}$$

Thus, solving for $C1$, we find that

$$C1 = \frac{1}{\sqrt{h \sum_{i=1}^{N} (x_i - \tau_i)^2}}$$

With τ_i and C_1 known then w_i can be determined. Computing the network output weights from w_i completes the construction of the network.

Results

Eq.(31) with its associated boundary and orthonormality conditions has the following exact solution

$$y(x) = \sqrt{2} \sin(j \pi x)$$

Figures 12 - 17 compare the sampled outputs of generated feedforward neural networks (denoted by triangles) with the exact eigenfunctions (solid line) for integers $j = 1$ to $j = 6$. The number of sampled points vary from 15 to 45. The size of the networks in terms of the number of processing elements T was kept as low as possible while keeping $(E)_{RMS} \leq 0.01$. For instance, for $j = 1$, the RMS error was 7.23×10^{-5} with $T = 24$, while for $j = 6$, the RMS error was 9.84×10^{-3} with $T = 90$.

The program ran on a standard SUN Microsystems Sparc 2 workstation with a run time of approximately 1 minute for the largest case of $j = 6$. Again, this run time could be significantly shortened with properly optimized code.

Convergence Plots

Following the same arguments as for example 1, we would again expect a logarithmic convergence plot of the error versus the grid spacing to be quadratic in the L_2 norm. As Fig. 18 shows, this is indeed the case. The slope of the line is 1.945, or approximately 2.0. The slightly subquadratic convergence rate is due to the inaccuracies introduced by the process of orthnormalizing the solution. The convergence plot shown was produced for an eigennumber $j = 1$ and for a sequence of mesh spacings between $h = 0.1$ and $h = 0.00625$. As before, the values are marked with circles on the convergence plot.

6 CONCLUSION

In an effort to both develop more sophisticated engineering analysis software and enhance understanding of connectionist systems, the popular feedforward artificial neural network was applied to the solution of linear ordinary differential equations. Observing that even the most basic FFANN architecture involved a large number of interacting parameters of uncertain effects, an effort was made to impose constraints on the network system by attempting to assign specific roles to the various parameters.

An analogy was made between supervised learning and function approximation. Following this analogy, concepts from function approximation theory were brought to bear on the problem of parameter determination in the net. It was observed that a clear analogy could be made between the basis functions of approximation theory and the transfer functions of the network. The output weights of the network were viewed as basis expansion coefficients, and the input and bias weights were seen as controlling the size and location of the interpolation functions in the problem space.

Further analysis revealed that hard-limit transfer functions could be easily and economically employed to represent the first order spline basis function, more commonly known as the hat function in the function approximation literature. This allows nets to be viewed as straightforward functional basis expansions for the relationships they are to model, using commonplace basis functions.

The method of weighted residuals, and one of its variations, the Bubnov-Galerkin method, were introduced as mathematical algorithms to determine the values of basis expansion coefficients when approximating the solution to differential equations.

The analogies and equivalences thus uncovered allowed explicit formulae for the input and bias weights to be formulated. Additionally, they revealed how to transform the results of applying the Bubnov-Galerkin method into the output weights of a neural network.

Example problems were approached in the following manner: linear ordinary differential equations were solved using the Bubnov-Galerkin method with hat basis functions and the results used to construct neural networks. Results of the output of the networks were shown to demonstrate the accuracy of the approximation.

Thus, it is possible to construct directly and noniteratively a feedforward neural network to approximate arbitrary linear ordinary differential equations. The methods used are all linear ($O(N)$) in storage and processing time. The L_2 norm of the network approximation error decreases quadratically with the increasing number of hidden layer neurons. The construction requires imposing certain constraints on the values of the input, bias, and output weights, and the attribution of certain roles to each of these parameters.

All results presented used the hard limit transfer function. However, the noniterative approach should also be applicable to the use of hyperbolic tangents, sigmoids, and radial basis functions.

7 CURRENT AND FUTURE RESEARCH

Since the quality of the basis function affects the interpolation, storage, speed, and convergence properties of the noniterative method, work is progressing on generating higher order splines using the hyperbolic tangent, sigmoid, and radial basis functions. The higher order splines being investigated include Lagrange, Hermite, and B-splines.

The noniterative approach outlined in this paper has allowed suitable algorithms to be devised for the synthesis of FFANNs that approximate nonlinear ordinary differential equations. In addition, the noniterative algorithm has allowed work to progress in the construction of FFANNs, Sigma-Pi, and recurrent networks, to approximate linear and nonlinear partial differential equations.

In closing, work is progressing in applying the noniterative approach to the problem of supervised learning. A noniterative "training" algorithm has been developed which allows the supervised learning problem to be solved deterministically for the needed weights and number of hidden layers. The algorithm requires only the solution of systems of linear algebraic equations.

8 ACKNOWLEDGEMENTS

The authors would like to thank technical monitor Ms. Maureen Tracy of NASA Langley Research Center and Mr. Gregory Lind of Rice University for their helpful suggestions and assistance. This work was supported under NASA grant NAG 1-1433.

REFERENCES

[1] J. Freeman and D. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison Wesley, New York, (1991).

[2] M. Takeda and J. Goodman, Neural Networks for Computation: number representation and programming complexity, *Applied Optics*, **25**(18), 3033, (1986).

[3] E. Barnard and D. Casasent, New Optical Neural System Architectures and Applications, *Optical Computing 88*, **963**, 537 Toulon, France, (1988).

[4] L. Wang and J.M. Mendel, Structured Trainable Networks for Matrix Algebra, *IEEE International Joint Conference on Neural Networks*, **2**, 125 San Diego, June (1990).

[5] H. Lee and I. Kang, Neural Algorithms for Solving Differential Equations, *Journal of Computational Physics*, **91**, 110, (1990).

[6] A.J. Meade, Jr., An Application of Artificial Neural Networks to Experimental Data Approximation, AIAA-93-0408, AIAA Aerospace Sciences Meeting, Reno, Nevada, Jan. (1993).

[7] S. Omohundro, Efficient Algorithms with Neural Network Behaviour, *Complex Systems*, **1**, 237, (1987).

[8] T. Poggio and F. Girosi, A Theory for Networks for Approximation and Learning, *Artificial Intelligence Laboratory, Massachusetts Institute of Technology*, A.I. Memo No. 1140, July (1989).

[9] F. Girosi and T. Poggio, Networks for learning: a view from the theory of approximation of functions, *Proceedings of the Genoa Summer School on Neural Networks and Their Applications*, Prentice Hall, (1989).

[10] G. Cybenko, Approximation by Superposition of a Sigmoidal Function, *Math. Control Signals Systems*, **2**, 303, (1989).

[11] Y. Ito, Approximation of Functions on a Compact Set by Finite Sums of a Sigmoid Function Without Scaling, *Neural Networks*, **4**, 817, (1991).

[12] L.O. Chuo, C.A. Desoer and E.S. Kuh, *Linear and Nonlinear Circuits*, McGraw Hill, New York, (1987).

[13] P.J. Davis, *Interpolation and Approximation*, Blaisdell, New York, (1963).

[14] P.M. Prenter, *Splines and Variational Methods*, Wiley, New York, (1989).

[15] A. Maren, C. Harston and R. Pap, *Handbook of Neural Computing Applications*, Academic Press, New York, (1990).

[16] B.A. Finlayson, *The Method of Weighted Residuals and Variational Principles*, Academic Press, New York, (1972).

[17] P. Lax and B. Wendroff, Systems of conservation laws, *Comm. Pure and Applied Mathematics*, **13**, 217, (1960).

[18] C.A.J. Fletcher, *Computational Galerkin Methods*, Springer-Verlag, New York, (1984).

[19] G. Strang, *Linear Algebra and Its Applications*, Second Edition, Academic Press, New York, (1980).

[20] D.A. Anderson, J.C. Tannehill and R.H. Pletcher, *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Corporation, New York, (1984).

[21] *IMSL User's Manual, MATH/LIBRARY*, Version 2, (1991).

[22] C. Johnson, *Numerical Solution of Partial Differential Equations by The Finite Element Method*, Cambridge University Press, Cambridge, (1990).

[23] D. Trim, *Applied Partial Differential Equations*, PWS-KENT, Boston, (1990).

FIGURES

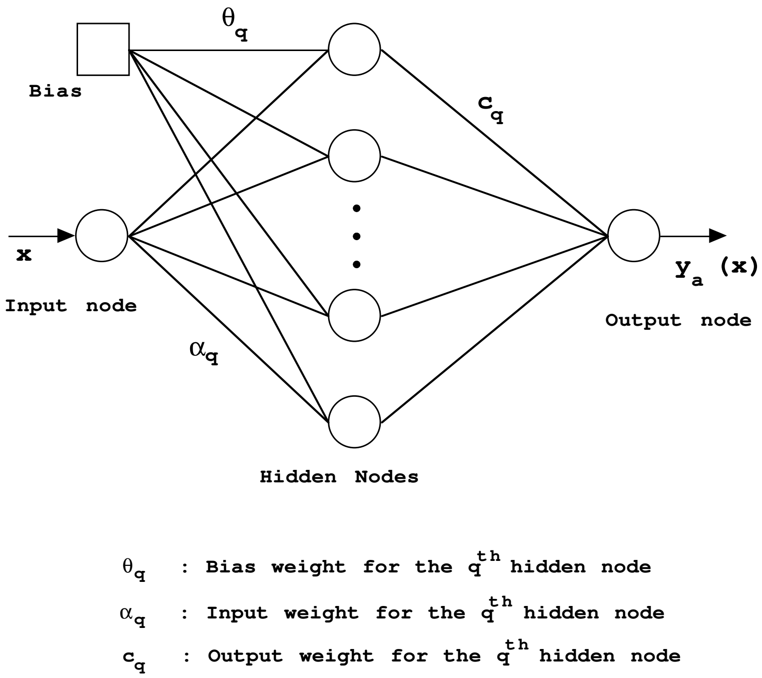

Figure 1: Standard feedforward neural network architecture.

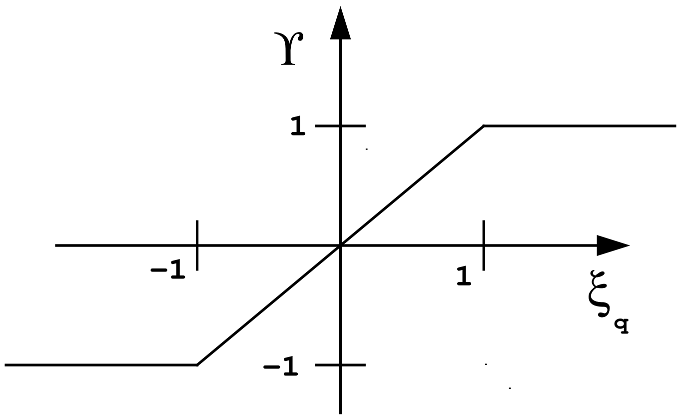

Figure 2: Hard limit transfer function.

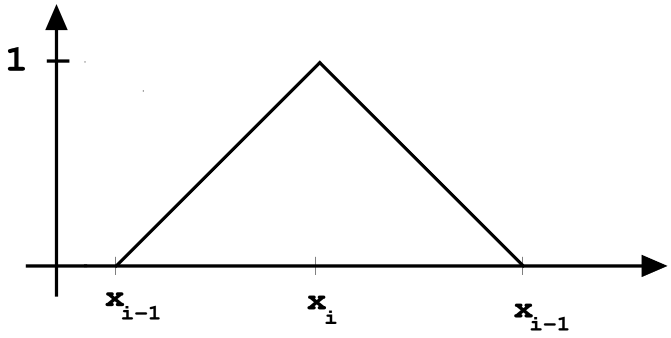

Figure 3: The Chapeau or "hat" function.

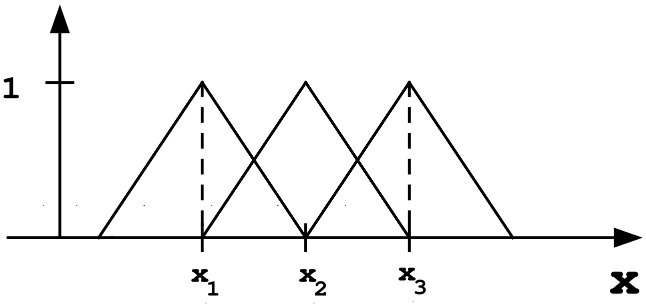

Figure 4: Proper distribution of hat functions along the problem domain ($x_1 \leq x \leq x_3$).

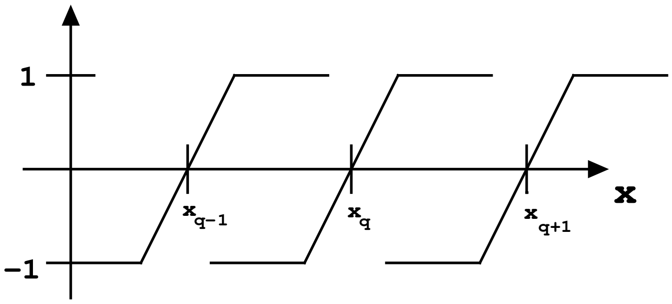

Figure 5: Arbitrary distribution of hard limits along x axis.

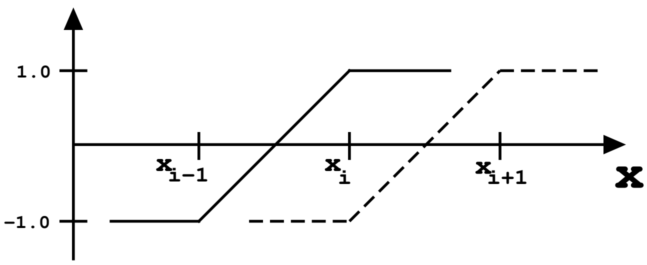

Figure 6: Constrained distribution of hard limits along x axis using the input and bias weights.

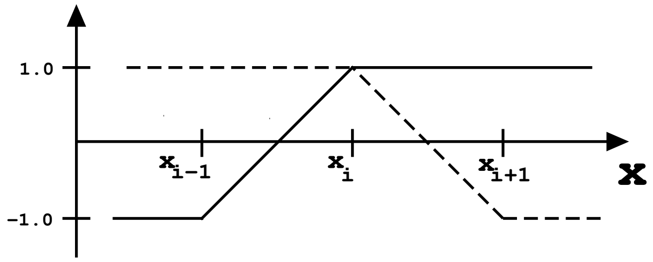

Figure 7: Right side hard limit (dashed) flipped by multiplying by negative output weight. Addition of the two hard limits creates a hat function. Refer to figure 3.

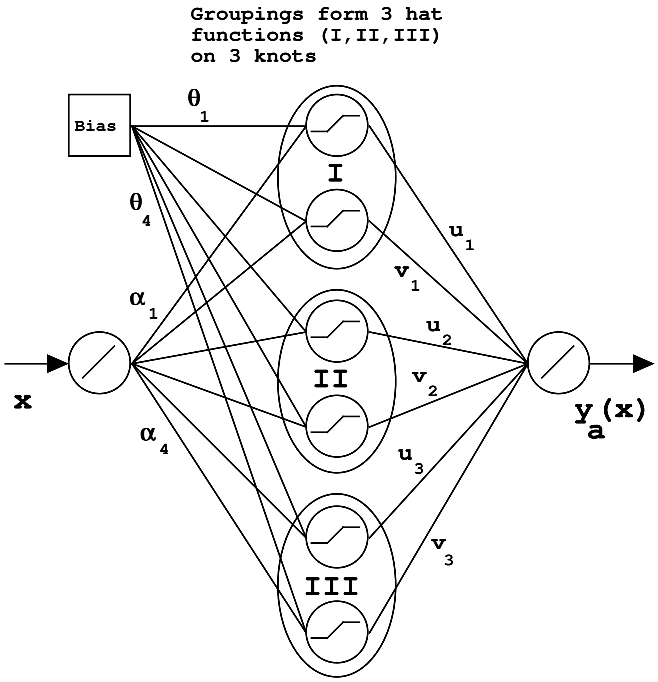

Figure 8: Single hidden layer feedforward network with constrained weights ($T = 6$, $N = 3$).

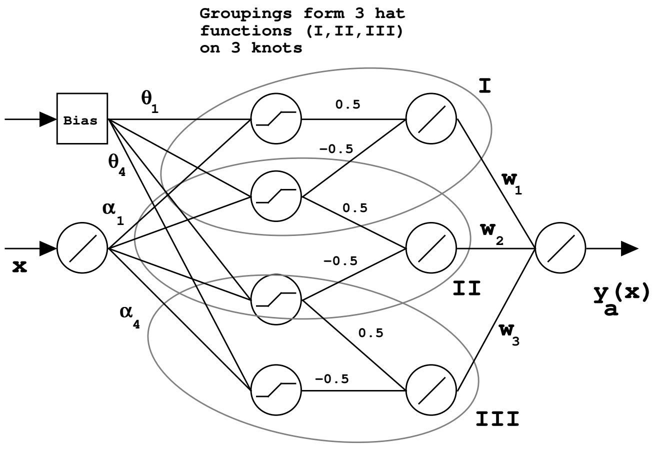

Figure 9: Two hidden layer feedforward network with constrained weights ($T = 7$, $N = 3$).

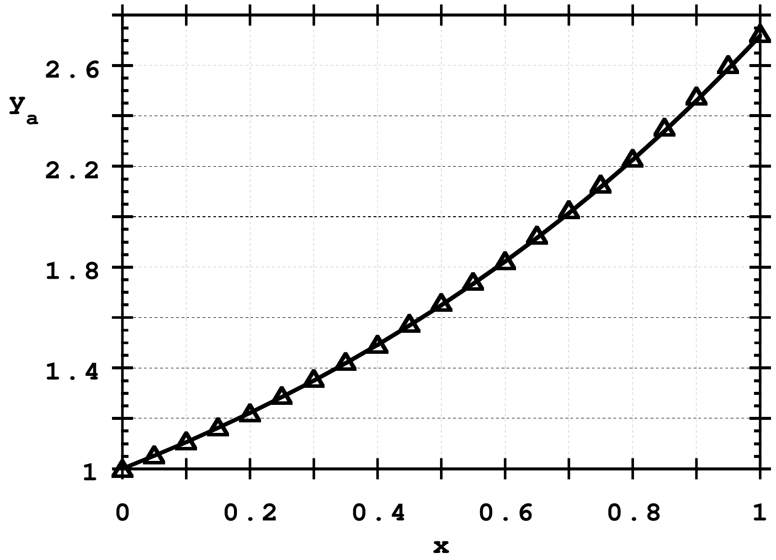

Figure 10: Comparison of exact solution and network output ($\triangle$) at $S = 21$ sample points with $T = 12$ processing elements. $RMS = 5.60 \times 10^{-5}$

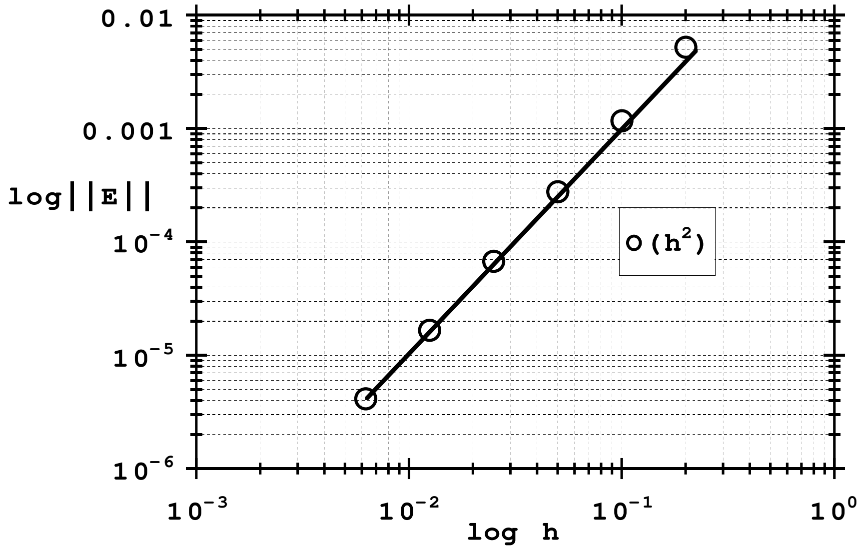

Figure 11: Logarithmic convergence plot. Slope = 2.03.

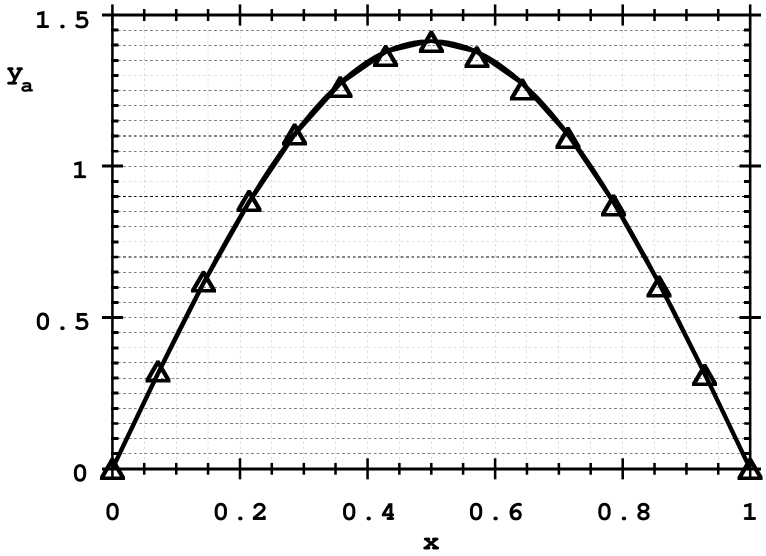

Figure 12: Comparison of exact eigenfunctions and network output ($\triangle$) for j = 1 with $T = 24$, $S = 15$, and $RMS = 7.23 \times 10^{-5}$.

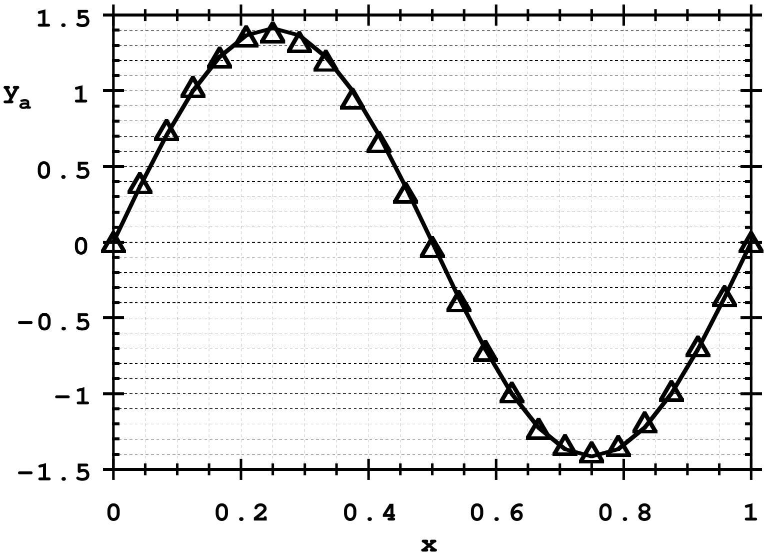

Figure 13: Comparison of exact eigenfunctions and network output ($\triangle$) for (b) j = 2 with $T = 32$, $S = 15$, and $RMS = 8.71 \times 10^{-5}$

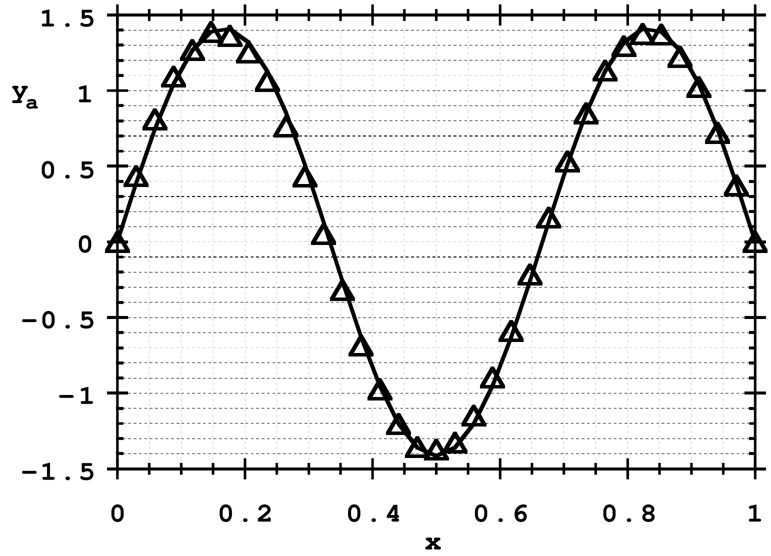

Figure 14: Comparison of exact eigenfunctions and network output ($\triangle$) for j = 3 with $T = 44$, $S = 35$, and $RMS = 2.70 \times 10^{-3}$.

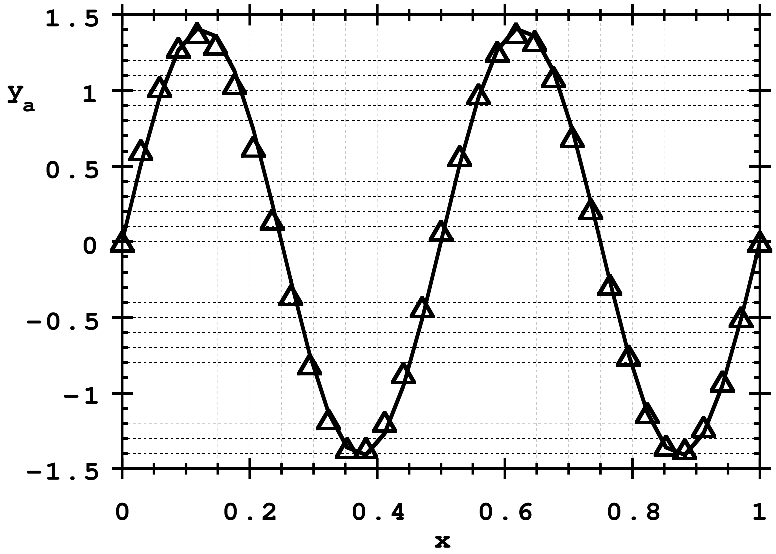

Figure 15: Comparison of exact eigenfunctions and network output ($\triangle$) for j = 4 with $T = 58$, $S = 35$, and $RMS = 5.00 \times 10^{-3}$.

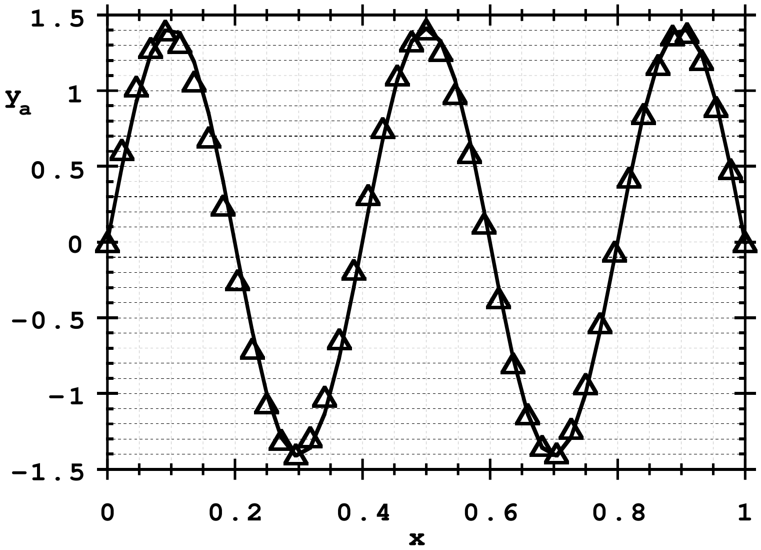

Figure 16: Comparison of exact eigenfunctions and network output ($\triangle$) for j = 5 with $T = 70$, $S = 35$, and $RMS = 8.93 \times 10^{-3}$.

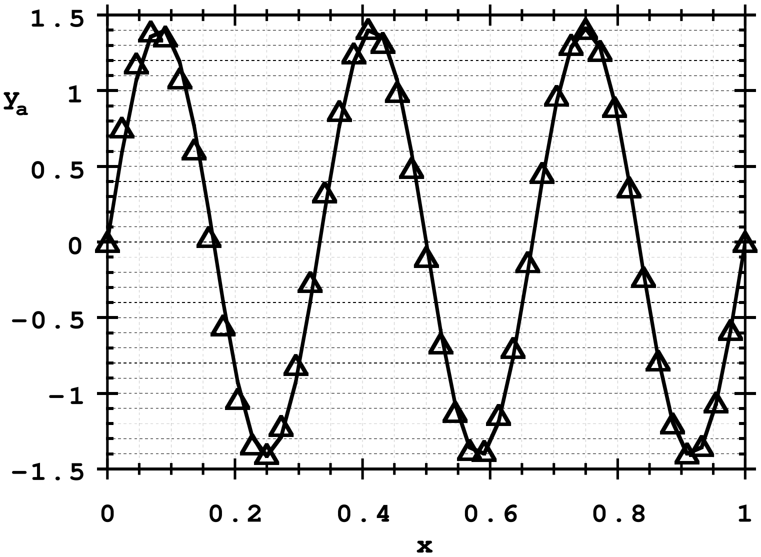

Figure 17: Comparison of exact eigenfunctions and network output ($\triangle$) for j = 6 with $T = 90$, $S = 45$, and $RMS = 9.84 \times 10^{-3}$

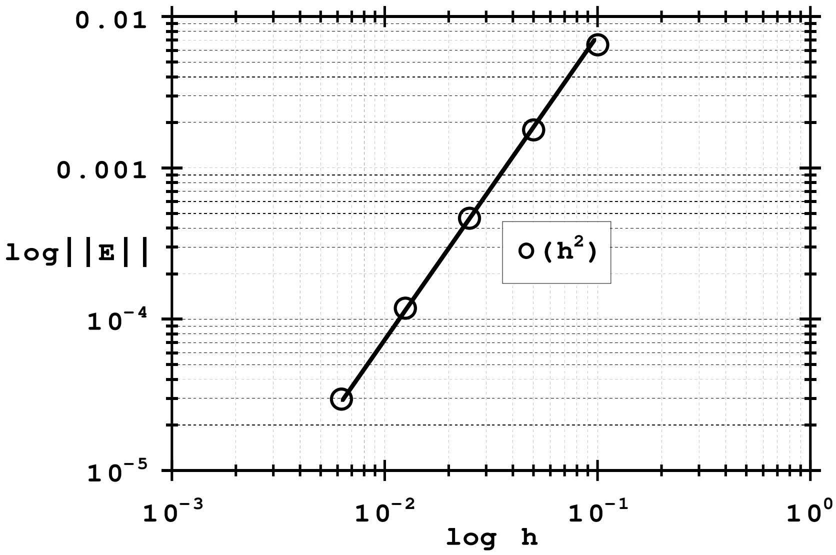

Figure 18: Convergence plot for j = 1. Slope = 1.94