Table 1 Eigenvalues in lateral-directional motions of a fighter aircraft

	Roll	Spiral	Dutch roll	Integrator
Open loop	−7.816	0.0076	$-0.757 \pm 5.806i$	N/A
Specified closed loop	−8.0	−0.05	$-4.88 \pm 3.66i$	−0.7
Resulting closed loop	−8.005	−0.0497	$-4.891 \pm 3.68i$	−0.6995

Numerical Example

To demonstrate this method, a linear model of a fighter aircraft is used. The aircraft is trimmed at Mach = 1.5 and h = 10,000 ft. The angle of attack $\alpha = 0.86$ deg, and the locally linearized lateral-directional equations of motion is

$$\begin{bmatrix} \dot{\beta} \\ \dot{p} \\ \dot{r} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} -0.493 & 0.015 & -1.000 & 0.020 \\ -61.176 & -7.835 & 4.991 & 0.000 \\ 31.804 & -0.235 & -0.994 & 0.000 \\ 0.000 & 1.000 & -0.015 & 0.000 \end{bmatrix} \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} + \begin{bmatrix} -0.002 & 0.002 \\ 8.246 & 1.849 \\ 0.249 & -0.436 \\ 0.000 & 0.000 \end{bmatrix} \begin{bmatrix} \delta_{df} \\ \delta_r \end{bmatrix} \tag{14}$$

where β, p, r, and ϕ represent the sideslip angle, pitch rate, yaw rate, and roll angle, whereas δ_{df} and δ_r are the deflection angles of differential flap and rudder, respectively.

For a zero steady-state error in sideslip, an integrator equation

$$\dot{\xi}_\beta = \beta - \beta_s \tag{15}$$

is added to Eq. (14), where β_s is the steady-state sideslip angle. The state and control vectors become

$$\boldsymbol{x} = (\beta, p, r, \phi, \xi_\beta)^T, \qquad \boldsymbol{u} = (\delta_{df}, \delta_r)^T$$

The elements of matrix R are chosen as

$$r_1 = 1/(\delta_{df,\max})^2 = 1/10^2, \qquad r_2 = 1/(\delta_{r,\max})^2 = 1/30^2$$

or $r_1 = 9$ and $r_2 = 1$ after multiplying by 900 (= 30^2), since only the relative magnitudes are needed.

If the specified closed-loop eigenvalues are −8.00 (roll), −0.05 (spiral), $-4.88 \pm 3.66i$ (dutch roll), and −0.7 (integrator), the elements of matrix Q are found from Eq. (10) to be

$$q_1 = -72.923, \qquad q_2 = 1.667, \qquad q_3 = 366.068$$
$$q_4 = 4.285, \qquad q_5 = 13.263$$

To ensure that Q is positive semidefinite, q_1 is set to 0. Solving the Ricatti equation (Eq. 12) with the computed weighting matrices Q and R, the full state control law can be determined as

$$\delta_r = 3.41(\beta - \beta_s) - 0.4126p + 15.947r - 0.865\phi - 3.536 \int (\beta - \beta_s)\, dt \tag{16}$$

$$\delta_{df} = -1.01(\beta - \beta_s) - 0.1576p - 0.8166r - 0.667\phi + 0.2904 \int (\beta - \beta_s)\, dt \tag{17}$$

As shown in Table 1, the resulting closed-loop eigenvalues are very close to the specified values.

Conclusions

A systematic method of determining weighting matrices for a linear quadratic regulator was proposed and studied. To obtain the specified closed-loop eigenvalues, the corresponding diagonal weighting matrices were numerically calculated. A tradeoff between penalties on the state and control inputs in the optimization of a cost function could also be analyzed. The computation normally needed less than 10 iterations and, thus, was very efficient.

References

[1]Bryson, A. E., and Ho, Y. C., *Applied Optimal Control*, Hemisphere, New York, 1975, Chap. 5.

[2]Stein, G., "Generalized Quadratic Weights for Asymptotic Regulator Properties," *IEEE Transactions on Automatic Control*, Vol. AC-24, Aug. 1979, pp. 559–566.

[3]Andry, A. N., Shapiro, E. Y., and Chung, J. C., "Eigenstructure Assignment for Linear Systems," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. AES-19, No. 5, 1983, pp. 711–728.

[4]Fahmy, M. M., and O'Reilly, J., "On Eigenstructure Assignment in Linear Multivariable Control Systems," *IEEE Transactions on Automatic Control*, Vol. AC-27, June 1982, pp. 690–693.

[5]Hovanessian, S. A., and Pipes, L. A., *Digital Computer Methods in Engineering*, McGraw–Hill, New York, 1969, pp. 6–10.

Recurrent Artificial Neural Network Simulation of a Chaotic System Without Training

Andrew J. Meade Jr.* and Rafael Moreno†
William Marsh Rice University,
Houston, Texas 77251-1892

I. Introduction

At present, an area of increasing interest is the emulation and control of nonlinear dynamic systems by the recurrent artificial neural network (RANN) architecture. These particular networks use supervised learning algorithms (training algorithms) such as recurrent backpropagation,[1] which allow the network to simulate a dynamic system without knowledge of the governing equations. However, since supervised learning algorithms require exposure to numerous training data sets, it can become memory intensive and time consuming without the guarantee of success. As a result, RANNs are not yet reliable tools for engineering.

Before RANNs can be used with confidence in the engineering community, several questions must be answered such as how to guarantee convergence, how to increase accuracy, and how to match a particular connection scheme to an application. We believe these questions can be answered by starting with two basic assumptions that force a slight paradigm shift: 1) RANN "learning" is actually function approximation, and 2) RANNs can be manipulated as well as conventional numerical techniques. Consequently, RANNs can be constructed to perform tasks with accuracy and computational efficiency, and therefore are as applicable to hard computing as they are to soft computing.

The second assumption follows naturally from the first since all numerical techniques used in computational mechanics can be considered to be methods of function approximation. However, in the majority of cases the numerical techniques construct a function from an integral or differential equation rather than data sets. A number of researchers[2] have published work operating under the first assumption for artificial neural networks (ANNs) in general, but few[3,4] have investigated the second. To prove the latter assumption, one must fully investigate the parameters that govern RANN performance such as connection weights, activation functions, types of neurons (additive or multiplicative), and connection schemes. The most straightforward and logical approach to the development of the mathematics of RANNs is the solution of well-posed problems of increasing complexity.

Received Oct. 3, 1994; revision received May 8, 1995; accepted for publication May 14, 1995.

*Assistant Professor, Department of Mechanical Engineering and Materials Science. Member AIAA.

†Research Assistant, Department of Mechanical Engineering and Materials Science. Student Member AIAA.

Currently our specific objective, and the topic of this paper, is to demonstrate through theory and numerical example that it is possible to construct a RANN to simulate a dynamic system, in the form of a nonlinear time-dependent ordinary differential equation, directly without training. In constructing the network we restricted ourselves only to commonly used activation functions, neurons, and connection schemes. The constructed network utilizes the explicit MacCormack finite difference method, piecewise linear transfer functions, and multiplicative and additive neurons to simulate the solution to the inhomogenous Duffing equation. Duffing's equation was chosen as the numerical example since its solution behaves both periodically and chaotically for certain parameter combinations. The network output, in the form of phase plane trajectories and Poincaré maps, compares well with the results of previous computational investigations for both chaotic and nonchaotic conditions.

II. Approach

A. RANN Architecture and Constraint of Parameters

The dynamic behavior of a RANN[5] is governed by the connection weight matrix, the manner in which the neurons in the network layers collect their inputs, and the activation functions applied to them.

As mentioned earlier, we have restricted ourselves to commonly used components. For the activation function we have chosen the first-order approximation to the hyperbolic tangent, the piecewise linear activation function (p. 48 of Ref. 6), since it is one of the simplest functions to implement on both digital computers and electronic hardware.[7] The neurons used were additive (Σ) and multiplicative (Π) (p. 237 of Ref. 6).

The piecewise linear activation function $\Upsilon_q^K(\xi_q)$ of the qth neuron of the Kth layer acting on input u_i can be modeled by the following equations:

$$\Upsilon_q^K(\xi_q) = \begin{cases} -1.0 & \text{for} \quad \xi_q < -1.0 \\ \xi_q & \text{for} \quad -1.0 \le \xi_q \le 1.0 \\ +1.0 & \text{for} \quad \xi_q > 1.0 \end{cases}$$

where

$$\xi_q = \begin{cases} \Sigma_i a_{iq} u_i & \text{for additive neurons} \\ \Pi_i a_{iq} u_i & \text{for multiplicative neurons} \end{cases}$$

and a_{iq} is the subset of connection weights linking the inputs u_i and the qth neuron in the Kth layer. We intend to show that by correctly assigning different values to the connection weights of a RANN, one can incorporate a system of nonlinear coupled algebraic recurrent relations into the architecture without the need of classical backpropagation or similar training. This will be accomplished by obtaining recurrent relations from a system of differential equations and designing a RANN suitable for the approximation of the solution.

B. Duffing's Equation

The inhomogeneous Duffing equation describes the forced motion of a particle between two equilibrium states and can be written as the following second-order nondimensional ordinary differential equation:

$$\frac{d^2x}{dt^2} + 2\mu\frac{dx}{dt} - \frac{1}{2}(x - x^3) = F_0\cos(\omega t) \tag{1}$$

where t, x, μ, F_0, and ω represent nondimensional forms of time, displacement, damping coefficient, force amplitude, and frequency of excitation, respectively. The importance of this equation is that its chaotic and nonchaotic behavior has been extensively examined by theoretical, experimental, and numerical methods.[8]

To apply the RANN programming method to Duffing's equation and to obtain dimensionless displacement, velocity, and acceleration, one reduces Eq. (1) to a system of two first-order equations by the following change of variables:

$$s = \frac{x}{\kappa_1}, \qquad y = \frac{1}{\kappa_2}\frac{dx}{dt} = \frac{1}{\kappa_2}\dot{x} \quad \text{and} \quad w = \frac{1}{\kappa_3}\frac{d^2x}{dt^2} = \frac{\kappa_2}{\kappa_3}\frac{dy}{dt} \tag{2}$$

where κ_1, κ_2, and κ_3 are dimensionless constants used to scale the new variables. Therefore, Eq. (1) becomes

$$\frac{dy}{dt} = -2\mu y + \frac{1}{2\kappa_4}\left(s - \kappa_1^2 s^3\right) + \frac{F_0}{\kappa_2}\cos(\omega t) = \frac{\kappa_3}{\kappa_2}w \tag{3}$$

$$\frac{ds}{dt} = \kappa_4 y \quad \text{where} \quad \kappa_4 = \frac{\kappa_2}{\kappa_1} \tag{4}$$

C. Network Approximation by the Explicit MacCormack Scheme

We will integrate the initial value problem of Eqs. (3) and (4) for arbitrary coefficients, using the explicit MacCormack method.[9] The MacCormack method is a finite difference, predictor-corrector scheme commonly used in the solution of time-dependent fluid dynamics equations. It should be noted that the RANN programming approach can use any time-marching technique that can be put into explicit form. The application of the MacCormack technique to Duffing's equation results in the following algebraic system:

$$p^n = (s^n)^3 \tag{5}$$

$$r^n = (s^*)^3 = \left(s^n + \kappa_4 \Delta t y^n\right)^3 \tag{6}$$

$$s^{n+1} = H_1 s^n + H_2 y^n + H_3 p^n + H_4\cos(n\omega\Delta t) \tag{7}$$

$$y^{n+1} = H_5 s^n + H_6 y^n + H_7 p^n + H_8 r^n + H_9\cos(n\omega\Delta t) + H_{10}\cos[(n+1)\omega\Delta t] \tag{8}$$

$$\sin[(n+1)\omega\Delta t] = H_{11}\sin(n\omega\Delta t) + H_{12}\cos(n\omega\Delta t) \tag{9}$$

$$\cos[(n+1)\omega\Delta t] = H_{11}\cos(n\omega\Delta t) + H_{13}\sin(n\omega\Delta t) \tag{10}$$

where the integer n represents the iteration count and Δt represents the dimensionless time step. Using Eq. (4), one can calculate the particle acceleration w from the nonrecurrent relation

$$w^n = H_{14} s^n + H_{15} y^n + H_{16} p^n + H_{17}\cos(n\omega\Delta t) \tag{11}$$

The coefficients H_1–H_{17} are

$$H_1 = 1 + \frac{\Delta t^2}{4}, \qquad H_2 = \kappa_4\Delta t(1 - \mu\Delta t)$$

$$H_3 = -\frac{\kappa_1^2\Delta t^2}{4}, \qquad H_4 = \frac{F_0\Delta t^2}{2\kappa_1}, \qquad H_5 = \frac{\Delta t}{2\kappa_4}(1 - \mu\Delta t)$$

$$H_6 = 1 - 2\mu\Delta t + 2\mu^2\Delta t^2 + \frac{\Delta t^2}{4}, \qquad H_7 = -\frac{\kappa_1^3}{\kappa_2}\frac{\Delta t}{4}(1 - 2\mu\Delta t)$$

$$H_8 = -\frac{\kappa_1^3}{\kappa_2\kappa_5^3}\frac{\Delta t}{4}, \qquad H_9 = \frac{F_0}{\kappa_2}\frac{\Delta t}{2}(1 - 2\mu\Delta t) \tag{12}$$

$$H_{10} = \frac{F_0}{\kappa_2}\frac{\Delta t}{2}, \qquad H_{11} = \cos(\omega\Delta t), \qquad H_{12} = \sin(\omega\Delta t)$$

$$H_{13} = -\sin(\omega\Delta t), \qquad H_{14} = \frac{\kappa_1}{2\kappa_3}, \qquad H_{15} = -\frac{2\mu\kappa_2}{\kappa_3}$$

$$H_{16} = -\frac{\kappa_1^3}{2\kappa_3}, \qquad H_{17} = \frac{F_0}{\kappa_3}$$

If we require that Δt be constant, then the time-dependent coefficients of Eq. (12) become constants for specific values of μ, F_0, κ_1, κ_2, and κ_3. Equations (5–10) are the linear and nonlinear algebraic equations that approximate Duffing's equation and are modeled by the RANN.

Note that the magnitude of all inputs and outputs are scaled to lie in the $[-1, 1]$ closed interval. This is to ensure that the accumulated inputs of each neuron remain within the linear range of their activation functions. To satisfy this constraint on the magnitude of the dependent variables s^n, y^n, w^n, p^n, and r^n, it is required that

$$\kappa_1 > |x|_{\max}, \qquad \kappa_2 > \left|\frac{dx}{dt}\right|_{\max}$$

$$\kappa_3 > \left|\frac{d^2x}{dt^2}\right|_{\max}, \qquad \kappa_5 \ge 1 + \kappa_4\Delta t \tag{13}$$

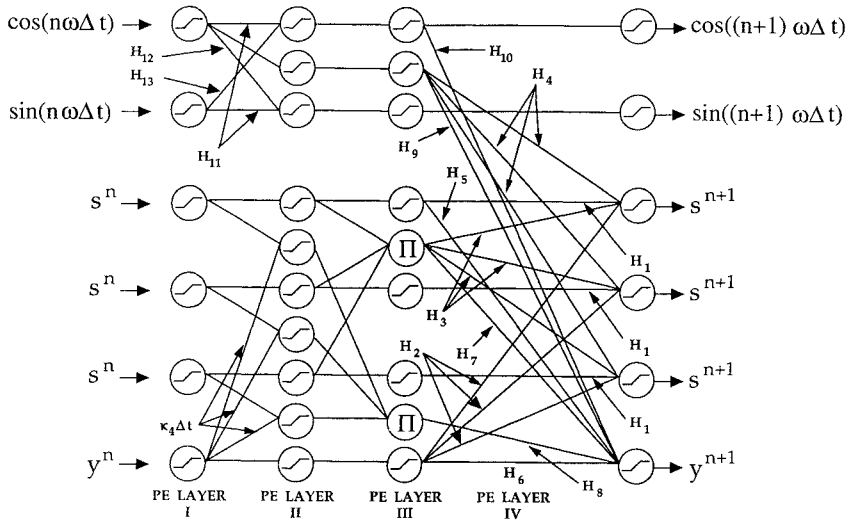

Fig. 1 RAAN constructed from the MacCormack scheme applied to Duffing's equation.

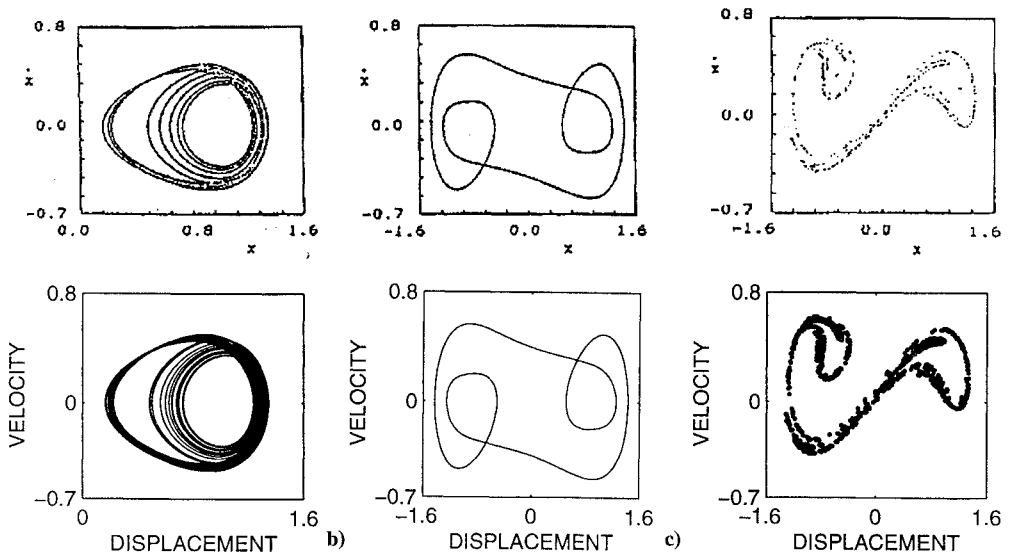

Fig. 2 Results from Masoud and Asfar (top set) and the RANN simulation of Duffing's equation (bottom set). a) Phase space trajectories for ω = 0.957 period-8, coexisting with b) ω = 0.957 global period-5. c) Poincaré maps in the first chaotic region for ω = 0.865.

The cubic time-dependent unknowns of Eqs. (5) and (6) are approximated in the RANN by Π neurons. In addition, the trigonometric nonlinearities in Eqs. (9) and (10) are updated recursively. Incorporating Eqs. (5–10) into a RANN architecture results in the four-layer neural network shown in Fig. 1. Connection weights that are not indicated in the figure have a value 1.

III. Results

The RANN of Fig. 1 was simulated with fixed values of μ = 0.084 and F_0 = 0.178. Previous results of the two-well potential system[8] indicate nondimensional displacement and velocity ranging between ±2.0 and ±1.0, respectively. To satisfy the conditions of Eq. (13) we used $\kappa_1 = 4.0, \kappa_2 = 2.0$, and $\kappa_3 = 1.0$. Initial conditions were set for $x(0) = 1.0$ and $\dot{x}(0) = 0.0$ for all cases. To ensure that steady-state solutions were displayed, approximations were run for 900 periods (T) based on the nondimensional forcing frequency ω ($T = 2\pi/\omega$). All computations were done in double precision, and all test cases were run in less than 40 s of real time on a Sparc 10 workstation.

Figures 2a–2c compare the simulated RANN output ($x = \kappa_1 s$ and $\dot{x} = \kappa_2 y$) with the numerical results of Masoud and Asfar[8] for the same parameter values and initial conditions. Throughout the simulation the dimensionless time step used in the RANN was identical to that used in the Masoud and Asfar study, $\Delta t = T/100$.

Figures 2a and 2b compare the phase plane trajectories of period-8 oscillations at ω = 0.957 and a coexisting global period-5 motion at which the system oscillates between the two equilibrium states $(\dot{x}, x)$ of (0, −1) and (0, 1). Note that the motion of the oscillator can be characterized by the number of times a cycle of oscillation is completed (k) in one period T of the forcing function about the total number of equilibrium states of the system. Such an oscillation is known as global period-k oscillation. If the oscillator completes k cycles of motion in one period of excitation T about a number of equilibrium states less than the total number, the oscillator is said to have completed a period-k oscillation. Figure 2c compares the Poincaré maps of the first region of chaotic oscillations at ω = 0.865.

IV. Conclusions

The engineer, as a RANN user, is usually capable of casting basic knowledge of a dynamic system in the form of algebraic and/or differential model equations. Training RANNs with experimental observations using arbitrary initial connection weight configurations when theoretical models are available can be time consuming, without the guarantee of convergence. It would be more convenient and efficient to incorporate mathematical models directly into the RANN architecture. The technique presented in this paper is such a method.

By theory and numerical example, we have shown that it may be possible to assemble a RANN from commonly used components and select the connection pattern and values of connection weights to approximate the solution to a time-dependent nonlinear ordinary differential equation. Specifically, we have shown that given a system of coupled differential equations describing periodic and chaotic motion we can obtain a set of nonlinear recurrent algebraic equations utilizing the explicit MacCormack finite difference scheme. The set of nonlinear recurrent algebraic equations can then be represented in RANN architecture.

Once the available information about a system of interest is incorporated, conventional training can be used to correct and refine the possibly inadequate initial theoretical model. An additional and important benefit of a RANN programming approach is that an analog simulation of the physical system of interest or arbitrary ordinary differential equations can be constructed in hardware for rapid real-time computation.

We have also shown that it is possible to describe the operation of a RANN with an explicit finite difference marching technique. As a result, in light of the extensive literature that exists on finite difference techniques, it may be possible to better understand the stability and convergence properties of RANNs in general. For example, error bounds can be derived for the explicit MacCormack method that indicate that in the integration of a linear ordinary differential equation the L_2 norm of the error will be bounded by a quadratic term in the time step Δt, whereas less than quadratically in a nonlinear case.[9] In our numerical example a user can then control the accuracy of the RANN output by decreasing the value of Δt in the connection weights of Eq. (12).

It may seem that the programming approach is severely limited by the use of the piecewise linear transfer function and the scaling of all variables to its linear range. However, for applications in which speed of real-time computation is important, the piecewise linear function is one of the most computationally efficient functions to use in hardware.[7] It is possible to use other transfer functions reported in the literature (e.g., hyperbolic tangents) by using the scaling constants to fit variables within the linear range of nonlinear transfer functions. Moreover, by linearly combining piecewise linear activation functions one can generate hat functions,[10] which are also used as one-dimensional interpolation functions in finite elements.[11] Using the hat function in a basis expansion, it is then possible to represent and incorporate into the RANN a required nonlinear function of a polynomial or nonpolynomial nature.[4] Depending on the characteristics of the nonlinearity and the required precision of its approximation, the number of neurons required for such a representation can vary enormously and increase the size of the RANN to hundreds of neurons. For reasons of clarity and brevity, we have limited our approximation of polynomial nonlinearities by using Π neurons and updating trigonometric functions from their recursive nature. Work will continue in investigating the computational capabilities of the more commonly used two-layer RANN with sigmoidal activation functions and additive neurons.

Acknowledgments

This work was supported under NASA Grant NAG 9-719. The authors would like to thank Robert O. Shelton of the NASA Johnson Space Flight Center for his many helpful suggestions.

References

[1]Pineda, F. J., "Recurrent Backpropagation and the Dynamical Approach to Adaptive Neural Computation," *Neural Computation*, Vol. 1, 1989, pp. 161–172.

[2]Hornik, K., Stinchcombe, M., and White, H., "Multi-Layer Feedforward Networks Are Universal Approximators," *Neural Networks*, Vol. 2, No. 5, 1989, pp. 359–366.

[3]Lee, H., and Kang, I., "Neural Algorithms for Solving Differential Equations," *Journal of Computational Physics*, Vol. 91, No. 1, 1990, pp. 110–131.

[4]Meade, A. J., and Fernandez, A. A., "Solution of Nonlinear Ordinary Differential Equations by Feedforward Neural Networks," *Mathematical and Computer Modelling*, Vol. 20, No. 9, 1994, pp. 19–44.

[5]Hecht-Nielsen, R., *Neurocomputing*, Addison–Wesley, Reading, MA, 1990, pp. 183–186.

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

[7]Chua, L. O., Desoer, C. A., and Kuh, E. S., *Linear and Nonlinear Circuits*, McGraw–Hill, New York, 1987, pp. 171–212.

[8]Masoud, K. K., and Asfar, K. R., "Period Doublings, Bifurcations and Strange Attractors in a Two-Well Potential Oscillator," *European Journal of Mechanics, A/Solids*, Vol. 12, No. 3, 1993, pp. 417–428.

[9]Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, Hemisphere, New York, 1984, pp. 163–164.

[10]Prenter, P. M., *Splines and Variational Methods*, Wiley, New York, 1989, p. 91.

[11]Fletcher, C. A. J., *Computational Galerkin Methods*, Springer–Verlag, New York, 1984, p. 87.