Modified Levenberg–Marquardt Training Algorithm for Solving Fuzzy Perturbed Integro-Differential Equations

Authors Names

Ahmed Raad Hassan [a]

Publication date: 29 / 10 /2026

Keywords: Fredholm Integro-Differential Equations, Levenberg–Marquardt training algorithm, feedforward neural network, fuzzy system.

ABSTRACT

In this paper, will develops a high-performance training algorithm based on the Levenberg–Marquardt (LM) algorithm by designing a partial fuzzy neural network (PFNN) with fuzzy weights, expressed through the ω-cut representation, and real biases. The main goal of this paper is to develop feedforward neural networks (FFNNs) for solving fuzzy singularly perturbed (FSPFI-DEs). The modified Levenberg–Marquardt (LM) algorithm designed for this purpose is called the Fuzzy ω-cut Levenberg–Marquardt (F ωLM) algorithm, and the sigmoid function is used as the activation (transfer) each unit's function. To demonstrate the effectiveness & precision of the suggested method, the results of numerical experiments for several examples are compared with the corresponding analytical solutions.

1. Introduction

The "Levenberg–Marquardt (LM) algorithm" is one of the most efficient second-order optimization techniques for training neural networks [13, 14]. In many practical problems, the model parameters involve uncertainty, and one way to represent this uncertainty is through fuzzy numbers. Fuzzy weights are represented here using the ω-cut representation, which transforms fuzzy arithmetic into interval arithmetic at different ω- levels. Based on this idea, we propose a modified LM training method called the Fuzzy ω-cut Levenberg–Marquardt (F ωLM) algorithm. Integral equations arise in numerous scientific disciplines and applications, including astronomy [1], ecology [2], and manufacturing [3, 4]. They also have many other applications, such as "fluid dynamics", "game theory", "medicine", "heat and mass transfer", "filtration theory", "oscillation theory" biomechanics, control, finance, bounce, electrodynamics, electrostatics, and more [5].

Research on coupled infrared radiation and molecular conduction [10], "elastic contact problems", "mechanics of fractures" [9], "airfoil theory" [6], and other subjects uses Fredholm and Volterra integral equations of the second kind.

(ANNs) was motivated by the functional principles of of the human brain; a typical network combines multiple hidden units with a single linear output neuron. Such networks are built from a large no. of artificial neurons that can be arranged in an ordered (layered) structure, so that many problems related to classification can now be addressed by designing a suitable ANN. According to their position in a network, there are 3 types of computational neurons:" input, output, and hidden neurons "[11, 12].

2. Basic Definitions

Definition 1 [8]: Let X be a nonempty set of points (referred to as the universal set). A fuzzy set $\mathcal{A}$ in X is characterized by its membership function

$U_{\mathcal{A}} = X \to I$, where I is the closed unit interval $[0, 1]$, and the fuzzy set $\mathcal{A}$ can be written as

[a]*Department of Mathematics, faculty of education , University of Al-Qadisiyah, 58006 Al Diwaniyah, Iraq., E-Mail: ahmed.hasan@qu.edu.iq*

$$\mathcal{A} = \{(\times, U_{\mathcal{A}}(\times))| \times \in X, 0 \leq U_{\mathcal{A}}(\times) \leq 1\}$$

Remark 1 [9]:

It is important to note that the two laws $\mathcal{A} \cup \mathcal{A}^c = X$ and $\mathcal{A} \cap \mathcal{A}^c = \emptyset$ do not hold for fuzzy sets, since $\mathcal{A} \cup \mathcal{A}^c \neq X$ and $\mathcal{A} \cap \mathcal{A}^c \neq \emptyset$. Indeed, for all $x \in X$, if $U_{\mathcal{A}}(\times) = r, 0 < r < 1$, then:

$$U_{\mathcal{A}\cup\mathcal{A}^c}(\times) = \max\{\tau, 1-\tau\} \neq 1 \quad (2.1)$$

$$U_{\mathcal{A}\cap\mathcal{A}^c}(\times) = \min\{\tau, 1-\tau\} \neq 0 \quad (2.2)$$

3. Architecture of the Partial Neuro-Fuzzy System (PN-FS)

A (N-FS) combines fuzzy systems with NNs: it uses machine learning, through the approximation capabilities of NNs, to ascertain a fuzzy system's parameters (such as fuzzy rules &fuzzy sets).

Fuzzy systems and NNs share many properties, and both are useful for solving problems such as fuzzy integral equations and fuzzy differential equations. A fuzzy neural network (FNN) is called a fully fuzzy neural network if all of its adjustable parameters(weights and biases) are fuzzy numbers; otherwise, it is called a partially fuzzy neural network (fuzzy weights and real biases).

The PN-FS consists of "1 input layer, 1 hidden layer, and 1 output layer".

3.1. Training Algorithms

After a suitable initialization of the weights and biases, the model is then ready for the training phase. The network can be trained for a variety of tasks, including pattern association, a pattern identification, and approximation of functions (nonlinear regression). During the training process, the training set is a collection of input vectors $X_i$as well as their target outputs Φi, or the network behavior that should be learned.

The weights and biases are iteratively updated, minimizing a predetermined performance function, during training. In a feedforward neural network, the typical performance function is called a mean squared error (MSE) This is equivalent to the mean of the squared of the variations among the target outputs Φi and the network outputs Φt.

Accordingly, The following rule is used to update the biases and weights during training:

$w_{k+1} = w_k + \eta \Delta w_k$

where:

η: is the learning rate;

Δw_k : is determined by the training algorithm. In most training algorithms, the learning rate η : controls the magnitude of the weight update (step size) [7].

The (SSE) is used as a performance measure to evaluate the training process. It is computed over all training patterns and the corresponding network outputs as follows

$$E(x, w) = \frac{1}{2} \sum_{P=1}^{P} \sum_{m=1}^{M} e_{P,m}^2 \quad (3.2)$$

where

x: input vector;

w: weight vector;

$e_{P,m}$: training error at output m when pattern p is applied, defined as:

$$e_{p,m} = d_{p,m} - t_{p,m} \quad (3.3)$$

Where:

d: desired output vector;

t: actual output vector; and

g: gradient, defined as the first-order derivative of the total error function (3.2), i.e.,

$$g = \frac{\partial E(x,w)}{\partial w} \quad (3.4)$$

3.2. The Levenberg–Marquardt Algorithm (LM)

A class of optimization algorithms derived from Newton's method avoids the explicit computation of second-order derivatives; these algorithms are referred to as quasi-Newton methods. They iteratively update an approximation of the" Hessian matrix", where the update is based on gradient information. Among these methods, the Broyden–Fletcher–Goldfarb–Shanno (B.F.G.S) algorithm has shown superior performance in many reported studies [7], and it is implemented in the MATLAB trainbfg routine.

When Newton's method is applied to update the weights utilizing the value of total mistake, evaluating the "Hessian matrix H" requires the computation of second-order derivatives, which can be computationally demanding. To reduce this complexity, the "Jacobian matrix J" is presented below:

$$J = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \cdots & \frac{\partial e_{1,1}}{\partial w_N} \\ \frac{\partial e_{1,2}}{\partial w_1} & \frac{\partial e_{1,2}}{\partial w_2} & \cdots & \frac{\partial e_{1,2}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{1,M}}{\partial w_1} & \frac{\partial e_{1,M}}{\partial w_2} & \cdots & \frac{\partial e_{1,M}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P,1}}{\partial w_1} & \frac{\partial e_{P,1}}{\partial w_2} & \cdots & \frac{\partial e_{P,1}}{\partial w_N} \\ \frac{\partial e_{P,2}}{\partial w_1} & \frac{\partial e_{P,2}}{\partial w_2} & \cdots & \frac{\partial e_{P,2}}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{1,M}}{\partial w_1} & \frac{\partial e_{1,M}}{\partial w_2} & \cdots & \frac{\partial e_{1,M}}{\partial w_N} \end{bmatrix} \quad (3.5)$$

By combining Equations (3.2) and (3.4), gradient vector's components can be computed as:

$$g_i = \frac{\partial E}{\partial w_i} = \frac{\partial(\frac{1}{2}\sum_{P=1}^{P}\sum_{m=1}^{M} e_{p,m}^2)}{\partial w_i} = \sum_{p=1}^{P}\sum_{m=1}^{M}(\frac{\partial e_{p,m}}{\partial w_i} e_{p,m}) \quad (3.6)$$

Combining Equations (3.5) & (3.6), The gradient vector g and the "Jacobian matrix J" have the following relationship:

$$g = Je \quad (3.7)$$

where the form of the error vector e is:

$$e = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ \cdots \\ e_{1,M} \\ \cdots \\ e_{P,1} \\ e_{P,2} \\ \cdots \\ e_{P,M} \end{bmatrix} \quad (3.8)$$

By substituting Equations (3.2) and (3.7), the (i,j)-th entry of the "Hessian matrix "can be expressed as follows:

$$h_{i,j} = \frac{\partial^2 E}{\partial w_i \, \partial w_j} = \frac{\partial^2 (\frac{1}{2} \sum_{p=1}^{P} \sum_{m=1}^{M} e_{p,m}^2)}{\partial w_i \, \partial w_j} = \sum_{p=1}^{P} \sum_{m=1}^{M} \frac{\partial e_{p,m}}{\partial w_i} \frac{\partial e_{p,m}}{\partial w_j} + S_{i,j} \quad (3.9)$$

Where:

$$S_{i,j} = \sum_{p=1}^{P} \sum_{m=1}^{M} \frac{e_{p,m}^2}{\partial w_i \, \partial w_j} e_{p,m} \quad (3.10)$$

According to Newton's method's fundamental premise that the term $S_{i,j}$ is negligible, The connection among the "Hessian matrix H and the Jacobian matrix J" can be expressed in the following form [7]:

$$H \approx J^T J \quad (3.11)$$

By applying update rule of "Newton's method"& substituting Equations (3.7) and (3.11), The following is the quasi-Newton algorithm's updating rule.:

$$w_{k+1} = w_k - (J_k^T J_k)^{-1} J_k e_k \quad (3.12)$$

Clearly, the primary benefit of the quasi-Newton technique over the conventional "Newton method" is that it eliminates the need to compute second-order derivatives of the total error function due to the presence of the Jacobian matrix J. However, when applied to optimization problems with complicated error surfaces, the quasi-Newton approach still confronts convergence

issues like to those of Newton's method. This challenge emerges mathematically because the matrix $J^T J$ might not be invertible.

A disadvantage of the quasi-Newton algorithm is its higher computational cost per iteration and larger memory requirement compared with other training algorithms, like conjugate gradient techniques, even though it usually takes less iterations to converge [7]. It is necessary to keep the approximate Hessian matrix, whose dimension is $n^2 \times n^2$, where n is the total number of weights and biases in the network.

Consequently, trainbfg can be an effective training function for smaller networks, but the robust backpropagation technique or one of the conjugate gradient algorithms may be better for very big networks.

Even more computationally demanding is the opposite of the Hessian H^{-1} computation. In this work, we suggest a new formulation for trainbfg to overcome these drawbacks.

3.3. Modified Levenberg–Marquardt Training Algorithm for PFNN Using ω-cut Representation

Let the neural network output be $\Phi(x, \overline{w}, b)$. Using ω-cut arithmetic, the output becomes the interval $\Phi^{\omega} = [\Phi_L^{\omega}, \Phi_U^{\omega}]$, where the subscripts L and U denote the lower and upper bounds of the ω-cut, respectively. The corresponding error vector is $e^{\omega} = \begin{bmatrix} e_L^{\omega} \\ e_U^{\omega} \end{bmatrix}^T$, and the fuzzy error function is:

$$\mathbb{E}(p) = \sum_{i=1}^{g} (\mathbb{E}_{i\omega}^{l}(p) + \mathbb{E}_{i\omega}^{u}(p)) \quad (3.13)$$

where the superscripts l and u refer to the error terms associated with the" lower and upper bounds", respectively. The Jacobian matrix with respect to the fuzzy weights is $J^{\omega} = \frac{\partial e^{\omega}}{\partial w^{\omega}}$, i.e.,

$$J^{\tau} = \begin{bmatrix} \frac{\partial e_U^{\tau}}{\partial w_L^{\tau}} & \frac{\partial e_L^{\tau}}{\partial w_U^{\tau}} \\ \frac{\partial e_U^{\tau}}{\partial w_L^{\tau}} & \frac{\partial e_U^{\tau}}{\partial w_U^{\tau}} \end{bmatrix} \quad (3.14)$$

Let the neural network output be $\Phi(x, \overline{w}, b)$. Using ω-cut arithmetic, the output becomes the interval $\Phi^{\omega} = [\Phi_L^{\omega}, \Phi_U^{\omega}]$, where the subscripts L and U denote the lower and upper bounds of the ω-cut, respectively. The corresponding error vector is $e^{\omega} = \begin{bmatrix} e_L^{\omega} \\ e_U^{\omega} \end{bmatrix}^T$, and the fuzzy error function is:

$$\mathbb{E}(p) = \sum_{i=1}^{g} (\mathbb{E}_{i\omega}^{l}(p) + \mathbb{E}_{i\omega}^{u}(p)) \quad (3.13)$$

where the superscripts l and u refer to the error terms associated with the" lower and upper bounds", respectively. The Jacobian matrix with respect to the fuzzy weights is $J^{\omega} = \frac{\partial e^{\omega}}{\partial w^{\omega}}$, i.e.,

$$J^{\tau} = \begin{bmatrix} \frac{\partial e_U^{\tau}}{\partial w_L^{\tau}} & \frac{\partial e_L^{\tau}}{\partial w_U^{\tau}} \\ \frac{\partial e_U^{\tau}}{\partial w_L^{\tau}} & \frac{\partial e_U^{\tau}}{\partial w_U^{\tau}} \end{bmatrix} \quad (3.14)$$

3.3.1. Modified LM Update Rule

The classical LM update is $\Delta w = -(J^T J + \lambda I)^{-1} J^T e$, where λ is the damping parameter.

For the fuzzy weights, $\Delta w^{\omega} = -(J^{\omega T} J^{\omega} + \lambda I)^{-1} J^{\omega T} e^{\omega}$, and the lower (L) and upper (U) weights are updated at each level ω

$$w_{L,new}^{\omega} = w_L^{\tau} + \Delta w_L^{\tau} \quad (3.15)$$

$$w_{U,new}^{\omega} = w_U^{\omega} + \Delta w_U^{\omega} \quad (3.16)$$

The bias update follows the classical LM rule:

$$\Delta b = -\left(J_b^T J_b + \lambda I\right)^{-1} J_b^T e \quad (3.17)$$

$$b_{new} = b + \Delta b \quad (3.18)$$

To improve stability, we define:

$$\lambda^{\omega} = \lambda_0 (1 + \sigma_w^{\tau}), \text{ where } \sigma_w^{\omega} = w_U^{\omega} - w_L^{\omega} \quad (3.19)$$

This quantity represents the uncertainty width of the fuzzy weight, i.e., the difference between its upper and lower bounds.

Algorithm 3.1 (F ωLM):

Step (1) Initialization: Initialize the lower and upper bounds of the fuzzy weights (w_L^{ω}, w_U^{ω}), the real biases b, the damping parameter λ, and the ω levels.

Step (2) Forward Propagation: For each training sample and each $\omega -$ level, propagate the interval weights and compute the interval outputs $[\Phi_L^{\omega}, \Phi_U^{\omega}]$.

Step (3) Error Evaluation: Compute the lower and upper errors e_L^{ω} and e_U^{ω}, and evaluate the total error.

Step (4) Jacobian Computation: Compute the Jacobian matrix with respect to the lower and upper weight bounds.

Step (5) Parameter Update: Compute the LM update and adjust the lower and upper weights.

Step (6) Bias Update: Update the biases using the classical LM formula.

Step (7) Damping Adjustment. If the error decreases, reduce λ; otherwise, increase λ.

Step (8) Convergence Check: Repeat until the convergence criteria are satisfied.

Theorem 3.1 (Convergence Theorem)

Assume that:

1. The error function $E^{\omega(w)}$ is twice continuously differentiable.
2. The Jacobian matrix J^{ω} has full rank near the optimal solution.
3. The damping parameter satisfies $\lambda > 0$.

Then the proposed Fuzzy ω-cut Levenberg–Marquardt algorithm converges locally to a stationary point of the fuzzy error function. Moreover, as λ approaches zero, the convergence rate approaches second-order convergence.

Proof:

Using the first-order Taylor expansion:

$$e(w + \Delta w) \approx e(w) + J\Delta w \tag{3.20}$$

The objective function is

$$E(w) = \frac{1}{2} e(w)^T e(w) \tag{3.21}$$

Substituting, we obtain

$$E(w + \Delta w) \approx \frac{1}{2} (e + J\Delta w)^T (e + J\Delta w) \tag{3.22}$$

Expanding gives

$$E(w+\Delta w) = \frac{1}{2} e^T e + e^T J\Delta w + \frac{1}{2} \Delta w^T J^T J\Delta w \tag{3.23}$$

The gradient is

$$\nabla E = J^T e + J^T J \Delta w \tag{3.24}$$

Setting $\nabla E = 0$ gives the Gauss–Newton equation:

$$J^T J \Delta w = - J^T e \tag{3.25}$$

Adding the damping term gives the LM equation:

$$(J^T J + \lambda I) \Delta w = - J^T e \tag{3.26}$$

Extending the parameter space to the lower and upper fuzzy bounds (w_L^{ω}, w_U^{ω}) preserves the descent property. Therefore, the algorithm inherits the local convergence properties of the classical LM algorithm.

4. Solving FSPFI-DEs with Boundary Conditions

To solve an FSPFI-DE using the PFNN, we consider a 3-layer N.N with 2 input units $\varkappa$ and y, 1 hidden layer consisting of m activation functions, & 1 output unit OUT($\varkappa$, y). Thus, the dimension of the PFNN is (2×m×1). input–output relations of each unit of the NN can be expressed for the ω-level set as follows:

Input unit

$$\varkappa = \varkappa, \mathrm{y}=\mathrm{y} \tag{4.1}$$

Hidden unit:

$$[\mathcal{H}id_{\mathrm{j}}]_{\omega} = [\,[\mathcal{H}id_{\mathrm{j}}]^{\mathrm{l}}_{\omega}, [\mathcal{H}id_{\mathrm{j}}]^{\mathrm{u}}_{\omega}] = [\,\Omega[net_{\mathrm{j}}]^{\mathrm{l}}_{\omega}, \Omega[net_{\mathrm{j}}]^{\mathrm{u}}_{\omega}] \tag{4.2}$$

$$[net_{\mathrm{j}}]^{\mathrm{l}}_{\omega} = \varkappa\left[\mathrm{w}_{\mathrm{j}1}\right]^{\mathrm{l}}_{\omega} + \mathrm{y}\left[\mathrm{w}_{\mathrm{j}2}\right]^{\mathrm{l}}_{\omega} + \mathfrak{b}_{\mathrm{j}} \tag{4.3}$$

$$[net_{\mathrm{j}}]^{\mathrm{u}}_{\omega} = \varkappa\left[\mathrm{w}_{\mathrm{j}1}\right]^{\mathrm{u}}_{\omega} + \mathrm{y}\left[\mathrm{w}_{\mathrm{j}2}\right]^{\mathrm{u}}_{\omega} + \mathfrak{b}_{\mathrm{j}} \tag{4.4}$$

Output unit:

$$[\mathrm{OUT}_{\mathrm{k}}]_{\omega} = \left[[\mathrm{OUT}_{\mathrm{k}}]^{\mathrm{l}}_{\omega}, [\mathrm{OUT}_{\mathrm{k}}]^{\mathrm{u}}_{\omega}\right] \tag{4.5}$$

$$[\mathrm{OUT}_{\mathrm{k}}]^{\mathrm{l}}_{\omega} = \sum^{\mathrm{m}}_{\mathrm{j}\in\mathrm{a}}\left[\mathrm{v}_{\mathrm{j}}\right]^{\mathrm{l}}_{\omega}[\mathcal{H}id_{\mathrm{j}}]^{\mathrm{l}}_{\omega} + \sum^{\mathrm{m}}_{\mathrm{j}\in\mathrm{b}}\left[\mathrm{v}_{\mathrm{j}}\right]^{\mathrm{l}}_{\omega}[\mathcal{H}id_{\mathrm{j}}]^{\mathrm{u}}_{\omega} \tag{4.6}$$

$$[\mathrm{OUT}_{\mathrm{k}}]^{\mathrm{u}}_{\omega} = \sum^{\mathrm{m}}_{\mathrm{j}\in\mathrm{c}}\left[\mathrm{v}_{\mathrm{j}}\right]^{\mathrm{u}}_{\omega}[\mathcal{H}id_{\mathrm{j}}]^{\mathrm{u}}_{\omega} + \sum^{\mathrm{m}}_{\mathrm{j}\in\mathrm{d}}\left[\mathrm{v}_{\mathrm{j}}\right]^{\mathrm{u}}_{\omega}[\mathcal{H}id_{\mathrm{j}}]^{\mathrm{l}}_{\omega} \tag{4.7}$$

$$\text{for } [\mathcal{H}id_{\mathrm{j}}]^{\mathrm{u}}_{\omega} \geq [\mathcal{H}id_{\mathrm{j}}]^{\mathrm{l}}_{\omega} \geq 0 \text{ , where :}$$

$$\mathrm{a} = \left\{\mathrm{j} : \left[\mathrm{v}_{\mathrm{j}}\right]^{\mathrm{l}}_{\omega} \geq 0\right\}, \qquad \mathrm{b} = \left\{\mathrm{j} : \left[\mathrm{v}_{\mathrm{j}}\right]^{\mathrm{l}}_{\omega} < 0\right\},$$

$$\mathrm{c} = \left\{\mathrm{j} : \left[\mathrm{v}_{\mathrm{j}}\right]^{\mathrm{u}}_{\omega} \geq 0\right\} \qquad , \qquad \mathrm{d} = \left\{\mathrm{j} : \left[\mathrm{v}_{\mathrm{j}}\right]^{\mathrm{u}}_{\omega} < 0\right\}$$

$$\mathrm{a} \cup \mathrm{b} = \{1,2,3, \ldots \mathrm{m}\} \text{ and } \mathrm{c} \cup \mathrm{d} = \{1,2,3, \ldots \mathrm{m}\}$$

5. Numerical Illustrations

This section depicts some examples that explain the proposed technique by using MATLAB 7.12 programs, in which the characteristics and features of the suggested design are shown. The PN-FS, which solves FSPFI-DEs, had several layers – (1) input layer, and (1) hidden layer with 9 hidden units/neurons. As demonstrated by the numerical experiments presented in the NFS, the proposed approach greatly enhances the accuracy and convergence rate while decreasing the number of iterations needed to attain the goal.

Example 1

Consider the following nonlinear FSPFI-DE:

$$\varepsilon\Phi''(\varkappa) = (\omega + 2{,}4 - \omega)\varkappa + \int_0^1 (\varkappa - \mathrm{t})\,\Phi(\mathrm{t})\,\mathrm{dt} \tag{5.1}$$

with the initial conditions (I.C.):

$$\Phi(0) = (\omega + 1{,}3 - \omega)\,, \Phi'(0) = (\omega, 2 - \omega) \tag{5.2}$$

where $(\omega + 1{,}3 - \omega)$ and $(\omega, 2 - \omega)$ are fuzzy numbers in $\mathbb{E}^1$ under the $\omega-$cut sets, and ε is the perturbation parameter, i.e., $0 < \varepsilon \ll 1$.

$$[\mathrm{A}]_{\omega} = [3 - \omega, \omega + 1], \omega \in [0,1]\,,\ [\mathrm{A}']_{\omega} = \left[\overline{\mathrm{A}'}, \underline{\mathrm{A}'}\right],\ \omega \in [0,1] \tag{5.3}$$

A fuzzy trial solution Φ_{t} can be written as:

$$[\Phi_t(\varkappa, p, \varepsilon)]_\omega = [A]_\omega + [A']_\omega(\varkappa - a) + (\varkappa - a)^2[Out(\Phi(\varkappa), \Phi(\omega), p, \varepsilon)]_\omega \quad (5.4)$$

$$[\Phi_t(\varkappa, p, \varepsilon)]_\omega = [3 - \omega, \omega + 1] + [A']_\omega(\varkappa - a) + (\varkappa - a)^2[Out(\Phi(\varkappa), \Phi(\omega), p, \varepsilon)]_\omega \quad (5.5)$$

From this, we obtain:

$$\overline{\Phi}_t(\varkappa, \overline{p}, \varepsilon) = 3 - \omega + \varkappa(2 - \omega) + \varkappa^2\overline{Out}(\Phi(\varkappa), \Phi(\omega), \overline{p}, \varepsilon) \quad (5.6)$$

$$\underline{\Phi}_t\left(\varkappa, \underline{p}, \varepsilon\right) = \omega + 1 + \varkappa\omega + \varkappa^2\underline{Out}\left(\Phi(\varkappa), \Phi(\omega), \underline{p}, \varepsilon\right) \quad (5.7)$$

where $Out(\Phi(\varkappa), \Phi(\omega), p, \varepsilon)$ is the output of the feedforward NFS with one input unit for $\varkappa$ and parameter p.

$$\frac{\partial\overline{\Phi}_t(\varkappa,\overline{p},\varepsilon)}{\partial\varkappa} = (2 - \omega) + \varkappa^2\frac{\partial\overline{Out}(\Phi(\varkappa),\Phi(\omega),\overline{p},\varepsilon)}{\partial\varkappa} + 2\varkappa\overline{Out}(\Phi(\varkappa), \Phi(\omega), \overline{p}, \varepsilon) \quad (5.8)$$

$$\frac{\partial\underline{\Phi}_t\left(\varkappa,\underline{p},\varepsilon\right)}{\partial\varkappa} = \omega + \varkappa^2\frac{\partial\underline{Out}\left(\Phi(\varkappa),\Phi(\omega),\underline{p},\varepsilon\right)}{\partial\varkappa} + 2\varkappa\underline{Out}\left(\Phi(\varkappa), \Phi(\omega), \underline{p}, \varepsilon\right) \quad (5.9)$$

$$\frac{\partial^2\overline{\Phi}_t(\varkappa,\overline{p},\varepsilon)}{\partial\varkappa^2} = \varkappa^2\frac{\partial^2\overline{Out}(\Phi(x),\Phi(\omega),\overline{p},\varepsilon)}{\partial\varkappa^2} + 2\overline{Out}(\Phi(\varkappa), \Phi(\omega), \overline{p}, \varepsilon) + 4\varkappa\frac{\partial\overline{Out}(\Phi(\varkappa),\Phi(\omega),\overline{p},\varepsilon)}{\partial\varkappa} \quad (5.10)$$

$$\frac{\partial^2\underline{\Phi}_t\left(\varkappa,\underline{p},\varepsilon\right)}{\partial\varkappa^2} = \varkappa^2\frac{d^2\underline{Out}\left(\Phi(\varkappa),\Phi(\omega),\underline{p},\varepsilon\right)}{\partial\varkappa^2} + 2\underline{Out}\left(\Phi(\varkappa), \Phi(\omega), \underline{p}, \varepsilon\right) + 4\varkappa\frac{\partial\underline{Out}\left(\Phi(\varkappa),\Phi(r),\underline{p},\varepsilon\right)}{\partial\varkappa} \quad (5.11)$$

The error function to be min are given by:

$$\mathbb{E}(p) = \sum_{i=1}^{g}(\mathbb{E}^l_{i\omega}(p) + \mathbb{E}^u_{i\omega}(p)) \quad (5.12)$$

where

$$\mathbb{E}^l_{i\omega}(p, \varepsilon) = \left[\left[\frac{\partial^2\underline{\Phi}_t\left(\varkappa,\underline{p},\varepsilon\right)}{\partial x^2}\right]^l_{\boldsymbol{\omega}} - \frac{1}{\varepsilon}\left[\sum_{\varkappa_i\in D}(\omega + 2)\varkappa + \int_0^1(\varkappa - t)\,\Phi(t)\,dt\right]^l_\omega\right]^2 \quad (5.13)$$

$$\mathbb{E}^u_{i\omega}(p, \varepsilon) =$$

$$\left[\left[\frac{\partial^2\overline{\Phi}_t(\varkappa,\overline{p},\varepsilon)}{\partial x^2}\right]^u_{\boldsymbol{\omega}} - \frac{1}{\varepsilon}\left[\sum_{\varkappa_i\in D}(4 - \omega)\varkappa + \int_0^1(\varkappa - t)\,\Phi(t)\,dt\right]^u_\omega\right]^2 \quad (5.14)$$

The error function to be minimized in this case is simple to evaluate:

$$\mathbb{E} = \sum_{i=1}^{11}(\mathbb{E}^l_{i\omega} + \mathbb{E}^u_{i\omega}) \quad (5.15)$$

The following tables and figure present the approximate solution obtained by the propose network and the error table which illustrates the accuracy and convergence speed of the proposed method with the modified QN and modified LM algorithm for a set of training points.

Table 1: The analytic and PFNN solutions obtained with the modified LM algorithm for Example 1, where $\varepsilon = 10^{-8}$, $\omega = 0.3$

Input	**Analytic solution**		**PFNN solution $\Phi_t(\varkappa)$**	
$\varkappa$	$[\Phi_a(\varkappa)]^l_\omega$	$[\Phi_a(\varkappa)]^u_\omega$	$[\Phi_t(\varkappa)]^l_\omega$	$[\Phi_t(\varkappa)]^u_\omega$
0	1.300000000000	2.700000000000	1.300000000000	2.700000000000
0.1	1.330388775046	2.870628058407	1.330388776743	2.870628905438
0.2	1.363154139258	3.045117876237	1.363154139784	3.045117878759
0.3	1.400794926935	3.227595840039	1.400794926567	3.227595800674
0.4	1.445946369032	3.422492981245	1.445946003456	3.422492905567
0.5	1.501383901151	3.634563297810	1.501383909054	3.634563999711
0.6	1.570027724946	3.868903323160	1.570027726734	3.868903390660
0.7	1.654948400549	4.130973418165	1.654948659865	4.130973467986
0.8	1.759373762118	4.426621296137	1.759373113765	4.426621997865
0.9	1.886697463711	4.762108327306	1.886698895411	4.762100078766
1	2.040489478373	5.144139207953	2.040489476599	5.144139207907

Table2: Accuracy of the solutions of Ex 1, where $\varepsilon = 10^{-8}$, $\omega = 0.3$

The error $[\mathbb{E}(\varkappa)]_\omega = \lvert[\Phi_a(\varkappa)]_\omega - \Phi_t(\varkappa)]_\omega\rvert$	
$[\mathbf{error}]^l_\omega$	$[\mathbf{error}]^u_\omega$
0	0
1.69724E-09	8.4703E-07
5.26863E-10	2.5218E-09
3.68029E-10	3.93652E-08
3.65576E-07	7.56773E-08
7.90356E-09	7.01901E-07
1.78723E-09	6.74995E-08
2.59317E-07	4.98208E-08
6.48353E-07	7.01728E-07
1.4317E-06	8.24854E-06
1.77459E-09	4.63647E-11
MSE=2.67109E-12	MSE=6.9103E-11

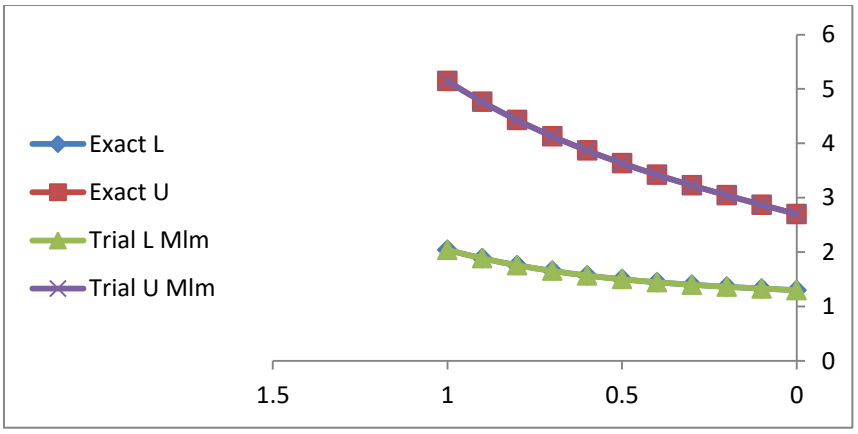


Figure 1: Analytic &neural solutions of Ex 1 with $\varepsilon = 10^{-8}$

References

[1] Al-Abrahemee, K.M.M., Efficient Algorithm for Solving Fuzzy Singularly Perturbed Volterra Integro-Differential Equation, Iraqi Journal of Science, 64(11), pp. 5851–5865, 2023. https://orcid.org/0000-0002-4705-6349.

[2] Ghassan Salih Salman, K.M.M. Al-Abrahemee, Designing feed forward fully fuzzy neural network to solve fuzzy singular perturbed Volterra integro-differential equations, Journal of Interdisciplinary Mathematics, Vol. 28 (2025), No. 1, pp. 139–146, DOI: 10.47974/JIM-1804

[3] A. Rosenfeld, Fuzzy graphs. In Fuzzy Sets and their Applications; Zadeh, L.A., Fu, K.S., Shimur, M., Eds.; Academic Press: New York, NY, USA, (1975), 77–95.

[4] I. A. Ali, A. F. Hassan, The Independent Incompatible Edges Topology on Di- graphs, Journal of Physics: Conference Series, 2322, (1) (2022), 012010.

[5] Al-Abrahemee, K.M., High Performance Training Algorithm with Strong Local Convergence Properties for Solving Some Fractional Integral Equation, AIP Conference Proceedings, Volume 2414, Issue 1,2023. https://doi.org/10.1063/5.0115034.

[6] Tabark Aqeel Al-Janabi, Khalid Mindeel Mohammed Al-Abrahemee, Modification fuzzy artificial neural networks for solving partial differential equations, AIP Conf. Proc. 3097, 080017 (2024). https://doi.org/10.1063/5.0210093.

[7] Khalid Mindeel Mohammed Al-Abrahemee, Alaa Kamil Jabber, A parallel numerical training algorithm for solving of a class of delay partial differential equations, AIP Conf. Proc. 2845, 050041 (2023). https://doi.org/10.1063/5.0157086.

[8] Tabark Aqeel Al-Janabi, Khalid Mindeel Mohammed Al-Abrahemee, Modification neural networks for solving singular perturbation problems based on polynomial of second degree, AIP Conf. Proc. 3097, 080006 (2024). https://doi.org/10.1063/5.0210091.

[9] Al-Abrahemee, K.M.M., A novel hybridized neuro-fuzzy model for solving fuzzy singular perturbation problems with initial conditions, Journal of Interdisciplinary Mathematics, 26(6), pp. 1287–1301, 2023, https://doi.org/10.47974/JIM-1627.

[10] Al-Abrahemee, K.M.M., Yousif, M.S. , A new parallel numerical algorithm for solving singular Perturbation problems in partial differential equations, Journal of Prime Research in Mathematics, 17(2), (2021).

[11] Tabark Aqeel Al-Janabi*, Khalid Mindeel Mohammed Al-Abrahemee, "Solving Singular Perturbation Problems With Initial and Boundary Conditions By Using Modified Neuro System," Iraqi Journal of Science, Vol. 65, No. 11, pp. 6566–6575, 2024. https://doi.org/10.24996/ijs.2024.65.11.31.

[12] L. Hooshangian., "Nonlinear Fuzzy Volterra Integro-diferential Equation of N-th Order: Analytic Solution and Existence and Uniqueness of Solution," Int. J. Industrial Mathematics, vol. 11, no. 1,

(2019).

[13] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," Journal of the Society for Industrial and Applied Mathematics, vol. 11, no. 2, pp. 431–441, 1963. https://doi.org/10.1137/0111030.

[14] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," IEEE Transactions on Neural Networks, vol. 5, no. 6, pp. 989–993, 1994. https://doi.org/10.1109/72.329697.