Journal of Interdisciplinary Mathematics
ISSN: 0972-0502 (Print), ISSN: 2169-012X (Online)
Vol. 29 (2026), No. 2, pp. 225–234
DOI : 10.47974/JIM-2012

Efficient design of neural network based on modified LM training algorithm for solving nonlinear 4th order 3D-PDEs

Farah F. Ghazi *
Luma N. M. Tawfiq †
Department of Mathematics
College of Education for Pure Science (Ibn Al-Haitham)
University of Baghdad
Baghdad
Iraq

Abstract

Authors in this work design efficient neural networks, which are based on the modified Levenberg - Marquardt (LM) training algorithms to solve non-linear fourth - order three -dimensional partial differential equations in the two kinds in the periodic and in the non-periodic - Periodic. Software reliability growth models are essential tools for monitoring and evaluating the evolution of software reliability. Software defect detection events that occur during testing and operation are often treated as counting processes in many current models. However, when working with large software systems, the error detection process should be viewed as a random process with a continuous state space, since the number of faults found during testing is vast and the number of faults corrected by bug fixing changes only insignificantly. The suggested design addressing minimization problems employs a feed-forward approach to solve problems like these equations by converting the original problem into an optimization. Efficient design is achieved through a calculated parameter for learning with high precision. To clarify applicability, reliability, and accuracy for this design, some examples are provided. Additionally, to demonstrate the efficiency of the proposed design, comparisons were conducted with other designs.

Subject Classification: *Primary 93A30, Secondary 49K15.*

Keywords: *ANNs, BP-training algorithm, FFNNs, LM training algorithm, PDEs, Convergence.*

* *E-mail:* **farah.f.g@ihcoedu.uobaghdad.edu.iq (Corresponding Author)**
† *E-mail:* **luma.n.m@ihcoedu.uobaghdad.edu.iq**

1. Introduction

Many phenomenon in physics, finance, and other branches of science are modeled through equations or systems involving rates of change, i.e., derivatives. These equations are generally classified as: ordinary differential equations (ODEs), when an equation depends on the derivatives of a single independent variable; or partial differential equations (PDEs), when the equation depends on the derivatives of two or more independent variables [1]. Finding the solutions to these differential equations is of great importance to make predictions based on these models. ANN-based LM takes into account two different kinds of poor debugging that occur throughout the fault elimination phenomena [2]. They thought that a defect may be partially eliminated during a removal effort [3]. This is referred to as incomplete fault debugging when there are more failures than removals in the scenario [4]. If not, it might occur that a defect is created while eliminating another problem and the existence of the new fault is only discovered once the original fault has been perfectly eliminated [5]. The overall fault content rises as a result of mistake creation [6]. Many methods and techniques have been proposed to solve these nonlinearities to get a highly effective technique for solving the non-linear problems [7]. In this article, we use artificial neural networks (ANN) to solve nonlinear 4th order 3D-PDEs, and using an efficient design of ANN based on a new modification of Levenberg-Marquardt (LM) training algorithm, we avoid the difficulties of equations consisting of nonlinear terms. LM provides the accuracy solution for convergent series as a rapid series will lead to a closed form of the solution. The nonlinear terms can be easily processed by the using ANN. Given non-linear 4^{th} order 3D-PDEs with known coefficients, and we have conditions at the initial point or/and boundary point, the numerical approaches [8] will be used to obtain numerical solutions at elements (discrete case), in the domain to get results as shown in the table. To get solutions elsewhere in a domain, the obtained solutions have been interpolated. One drawback of this technique is that the domain must be discrete, such as meshes, thus the table of obtained results will be increased, hence requiring a big memory [1].ANNs provide an efficient acting technique for adaptive neural solutions by using a suitable learning algorithm [2], since their ability to approximate well non-linear functions. One advantage is that results are represented by some parameters, which reduces the required memory [7]. Another advantage that results is covering domain without need its interpolation.

2. Neural Networks

An artificial neural networks (ANNs) are networks for interconnected neurons with weights and biases. The typical structure of an ANN model is depicted in Fig. 1. After establishing the network structure, the subsequent step involves training the network. Finding the ideal settings for the network's different weights and biases is known as network training. Usually, a number of methods are used to determine appropriate values for the artificial neural networks (ANN) weights and biases. The Firefly Algorithm (FFA) has been used in this study to achieve optimum network training. Additional information

regarding FFA may be found in [2], offers a thorough description of the FFA method that was employed in this study6. Moreover, MATLAB implementation codes are available in [3]. The effectiveness of neural networks depends heavily on the quality of the weights and parameters used to train them. Traditionally, neural networks are trained using optimization algorithms such as gradient descent, herein we used efficient learning optimization algorithm that is LM algorithm.

The treatment for the give data held by the income this data as weight input vectors x, as the form $W_j^T x$ to the enter in hidden layers. We present the relation as :

$$g(x) = \sum_{j=1}^{k} \quad \upsilon j \, \sigma(W_j^T x + b_j) \tag{1}$$

where b_j: bias, σ: activation function; g(x) represents output for ANN.

Note that, σ must be take sigmoidal function; herein take σ as [31]:

$$\sigma(n_i) = \frac{2}{e^{-2ni}+1} - 1 \tag{2}$$

Therefore, NN input − output equations are: $\hat{Y} = \upsilon^T \Phi(W^T x^T + b^T)$ when $W \epsilon R^{n \times r}$ is input weights will be adjustable. $\upsilon \epsilon R^{1 \times n}$ is the output weights adjustable and the biased is $b \epsilon R^{n \times r}$.

3. Proposed Modification for LM Learning Algorithm

Developing a framework for structural modeling using optimization techniques involves convex space concepts from both structural modeling and differential calculus; this will involve defining the structural model, formulating the differential equations, estimating parameters, and then validating the model performance. This section consists of suggested modifications for the LM learning algorithms denote as MLM , so we have:

Step 1: Let $x_0 \in R^n$ & if we have constant d_1 , d_2 , d_3 , μ , s with σ s.t $\mu > 0;\ 0 < d_1, d_2, d_3;\ s, \sigma \in (0,1)$. when k = 0.

Step 2: When $\|J_k^T E_k\| = 0$, so, we stop. Solve

$$(J_k^T J_k + \lambda_k I)p = -J_k^T E_k \text{ , with } \lambda_k = \mu \| E_k\|$$

To obtain p_k and set $y_k = x_k + p_k$,

Solve $(J_k^T J_k + \lambda_k I)p_k = -J_k^T E(y_k)$, To obtain $\hat{p}_k$ and set $l_k = p_k + \hat{p}_k$

Step 3: If $\| E(x_k + p_k + \hat{p}_k)\| \leq \sigma \| E_k\|, \| E(x_k + l_k)\| \leq \sigma \| E_k\|$,

Then take the $r_k = 1$ and go to step 5. Otherwise complete all steps.

Step 4: Compute r_k as $r_k = max\{1, s, s^2, s^3, ...\}$ with $r = s^i$, i =0, 1, 2, …

$\| E(x_k + r\, p_k + r^2 \hat{p}_k)\|^2 - \| E_k\|^2 \leq - d_1 r^2 \| p_k\|^2 - d_2 r^2 \| \hat{p}_k\|^2 - d_3 r^2 \| E_k\|^2 + \epsilon_k \| E_k\|^2$, Sequence which is $\sum_{k=0}^{\infty} \quad \epsilon_k < \infty$.

Step 5: $x_{k+1} = x_k + r_k p_k + r_k{}^2 \hat{p}_k$ Take k := k + 1 and go to Step 2.

4. Applications

The (ANNs) are considered one of the most important artificial intelligence techniques, which are used in many applications such as classification, prediction, and pattern recognition. The effectiveness of neural networks depends heavily on the quality of the weights and parameters used to train them. Traditionally, neural networks are trained using optimization algorithms such as gradient descent, which can struggle with complex loss function landscapes, leading them to converge to local minima. To overcome these challenges, propose the Firefly algorithm as an alternative, innovative approach to training artificial neural networks. The firefly algorithm is an optimization algorithm inspired by the natural behavior of fireflies, which attract one another based on the intensity of the light they emit, thereby facilitating the search. Here, we used a Feed Forward Neural Network (FFNN) with three fully interconnected layers; the input layer (first layer) consists of 4 neurons. But the hidden (second) layer consists of 9 neurons (depending on the error & trial) with tanh sig. activation functions (eq. (2)) and an output layer consisting of 3 neurons with Linsig. activation functions (see Figure 1). The train suggested NN by the backpropagation rule & choose the novel MLM algorithm. Many (SRGM) see the testing phase's fault detection procedure as a discrete counting process, based on the assumption of MLM. But according to recent observations, for large software systems, the number of faults found during testing increases significantly when compared to the original bug content at the start of the testing process, and the number of defects repaired by debugging operations decreases. Thus, a stochastic model with a continuous state space may be used to capture the stochastic nature of the defect-detection procedure.

Example 1: Consider the following 4 thorder nonlinear 3D- breaking soliton equation

$$u_{xxxy} + u_{xxxz} - u_{xt} + 2u_{xx}u_y + 2u_{xx}u_z + 4u_xu_{xy} + 4u_xu_{xz} = 0 \qquad (3)$$

With IC: $u_x(z, x, y, 0) = 2\big((z + x + y)\big)$

From IC: $u_{0x}(z, x, y, 0) = 2(z + x + y) \Longrightarrow u_0 = tanh\ tanh\ (z + x + y)$

The exact solutions are $[u(z, x, y, t) = 2\ tanh\ tanh\ (z + x + y + 8t)$

Neural Solution: For each given (input) data *x, z, y, t*, a process from the first (input) layers to the second (hidden) layers is:

$$n_i = \sum_{i=1}^{9} \quad (W_{x_i}x + W_{y_i}y + W_{z_i}z + W_{t_i}t) + b_1$$

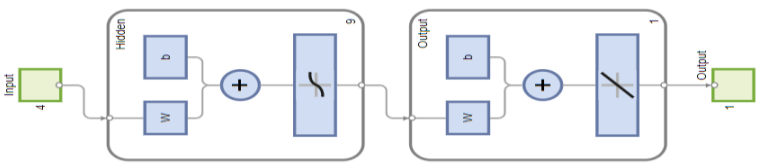

Figure 1
Architecture of neural network

where W_{x_i} , Wy_i , W_{z_i} , W_{t_i} are weight interrelate for *inputs z,x, y,* t to hidden layers respective, so b_1 biases for the hidden layers. The next step be process of interrelating hidden layers to output layers that base on next formula:

$$h_i = \sum_{j=1}^{9} \quad \upsilon_{ij}\ \sigma(n_i) + b_2 \tag{4}$$

Where each of υ_{i_j} is weight connect hidden layers with output, and b2 is biases. That is output for NN have a next form:

$$u_{net}\ (x, y, z\ , t;\ \theta) = \sum_{j=1}^{9} \quad \upsilon_i\ \sigma(h_i) \tag{5}$$

The performance for network solutions $\mho_{net}(x, y, z, t; \theta)$ is measured between the standard LM and modified MLM by computational train and test, and validation cases see Tables 1 &2 it shown the mean square error (MSE), best epoch, time, Mu max, Min grad, best test perf, best validation perf, best train perf and learning rate, we see that the results of suggested NN with MLM is good and convergent to exact solution faster than NN trained by LM. Also, the gradient of MLM is 1.01e-16 smaller than that of LM which is equal to 5.48e-8. The MSE computed to the check accuracy for suggested design which are obtain in these cases of different value for epochs, present in Figure 2, but Figure 3 illustrates the NN solution for different times. The errors between exact & NN solutions are illustrated in Figure 4. In [3], Mohammed and Mohammed solved this equation using the linear superposition (LS) method, along with a series of specific techniques, which is considered one of the classic methods that require many calculations, as well as the resulting solution contains some coefficients must be found, herein we use NN to reduce the calculation and time.

Table 1

The comparison between LM & MLM for Example 1

	MSE	Test perf	Vol.perf	Train perf	lr
LM	2.75623e-16	2.8557e-16	2.82041e-16	2.721e-16	0.001
MLM	6.55446e-32	6.57456e-32	6.42883e-32	6.5771e-32	0.01

Table 2

The comparison for accuracy LM & MLM for epoch & time, for example 1

Algorithms	LM			MLM		
Learning results	Reached minimum gradient			Reached maximum mu		
Unit	Initial value	Stopped value	Target value	Initial value	Stopped value	Target value
Epoch	0	15	1000	0	12	1000
Elapsed time	-	00:00:01	-	-	00:00:03	-
performance	72	2.72e-16	0	9.26	6.58 e-32	0
Gradient	67.2	5.48e-08	1e-07	20.9	1.01e-15	1e-100
Mu	0.001	1e-15	1e+10	0.001	1e+308	1e+308
Validation checks	0	0	6	0	0	10

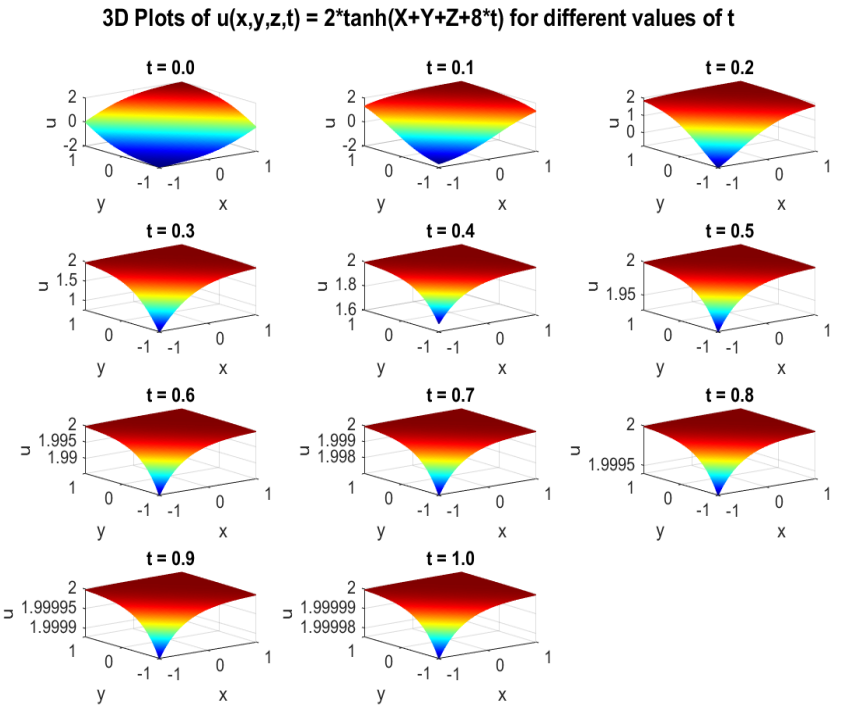

Figure 2
Results of suggested design of Example 1 for different times.

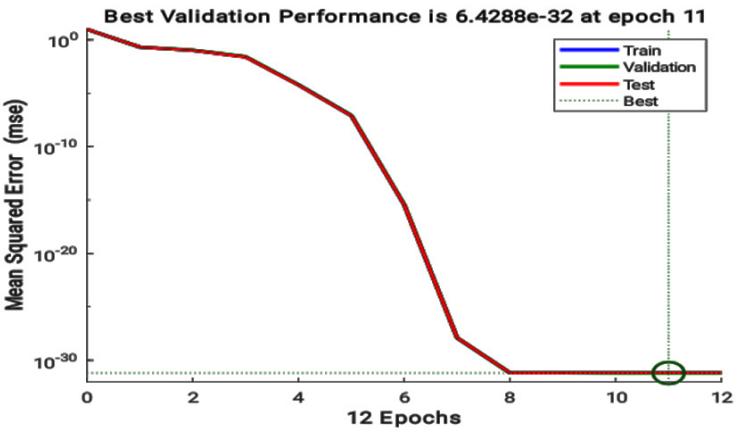

Figure 3
Mean square error of results for training, testing, and val. Example 1.

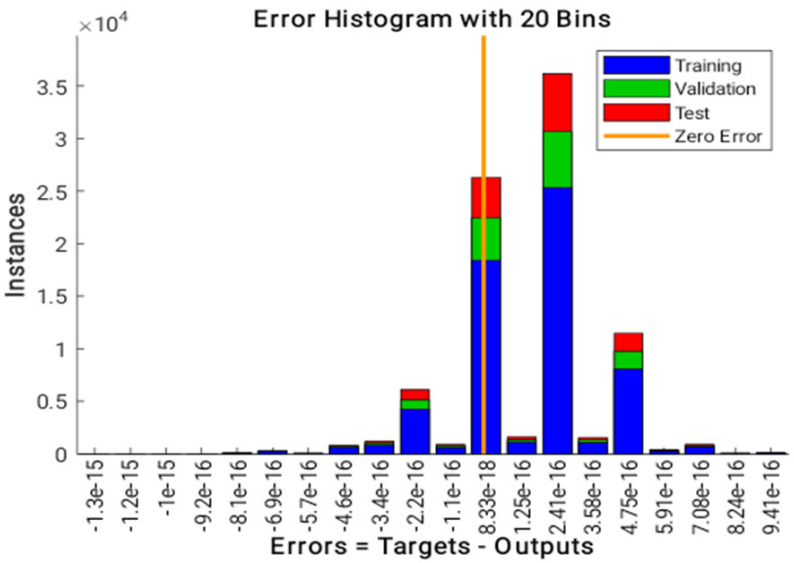

Figure 4
The error among exacts & MLM neural solutions of Example1.

Example 2: Consider a following 4^{th}order nonlinear 3D –Boiti-Leon-Mmanna-Pempinelli equation

$$u_{yt} + u_{zt} + u_{xxxy} + u_{xxxz} - 3u_x(u_{xy} + u_{xz}) - 3u_{xx}(u_y + u_z) = 0 \qquad (10)$$

With ICs: $u_0 = -2\ tanh(x + y + z)$; $u_y(x, y, z, 0) = -2(x + y + z)$

Exact solution $u(x, y, z, t) = -2tanh(x + y + z - 4t)$

Neural Solution: In the same manner in Example 1, the performance for NN solutions $\Psi_{net}(x, y, z, t; \theta)$ is measured between LM and modify MLM by computational train, test, and validation case see Tables 3 & 4 and Figure 5, it shown the MSE, best epoch, time, Mu max, Min grad, best test perf, best validation perf, best train perf and learning rate, we see that the results of suggested NN with MLM is better than NN results trained by LM and fastest convergent to exact solution. Figure 6 illustrates the neural results for different values of time. The MSE is computed to check the accuracy for the suggested design obtained in these cases of the different values for epochs, as shown in Figure 7. Although error among exact and neural solutions of MLM kind. In [7], the researcher solved this problem by transforming the equation from PDE to a non-linear fractional ODE type, and then used a numerical method to solve the resulting equation, and got the solution as a series consisting of an unknown constant. The method requires many calculations and time. Herein, we used ANN, which has many advantages to solve such problems, in addition, we got a high-accuracy solution.

Table 3

The comparison between NN results trained by LM &MLM Example2

	MSE	Test perf	Vol.perf	Train perf	lr
LM	8.4338e-19	8.7122e-19	7.98752e-19	8.4698e-19	0.001
MLM	9.771e-32	7.180e-32	7.2227e-32	7.260e-32	0.01

Table 4

The comparison for accuracy of LM&MLM for epoch &time for example 2

Algorithms	LM			MLM		
Results	Reached minimum gradient			Reached maximum mu		
Unit	Initial value	Stopped value	Target value	Initial value	Stopped value	Target value
Epoch	0	8	1000	0	30	1000
Elapsed time	-	00:00:01	-	-	00:00:01	-
performance	20	8.47e-19	0	10.2	7.26e32	0
Gradient	29.4	1.61e-10	1e-07	20	1.11e-16	1e-100
Mu	0.001	1e-07	1e+10	0.001	1e+308	1e+308
Validation checks	0	0	6	0	3	10

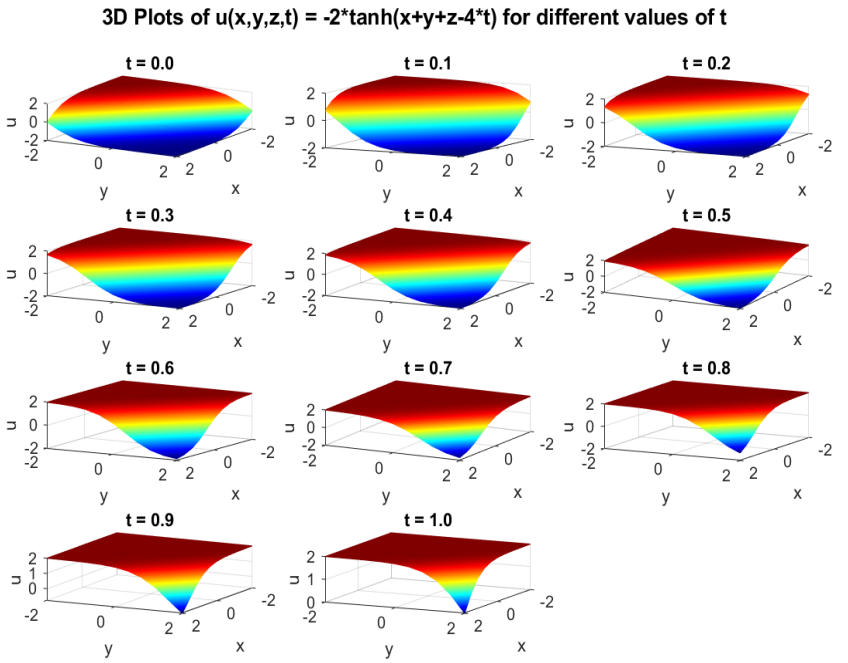

Figure 5

Results of the suggested design for Example 2, at different times.

5. Conclusions

The results in this work are compared with other methods, and the performance of a suggested design is tested on an actual software defect dataset. The MSE indicates that the proposed design fits the data better and has a wider range of applications. Subsequent studies can investigate the links between the proposed model and the error generation techniques. This discovery has important implications for mathematics and computer science, as it provides an ideal framework for simulating the increasing reliability of software.

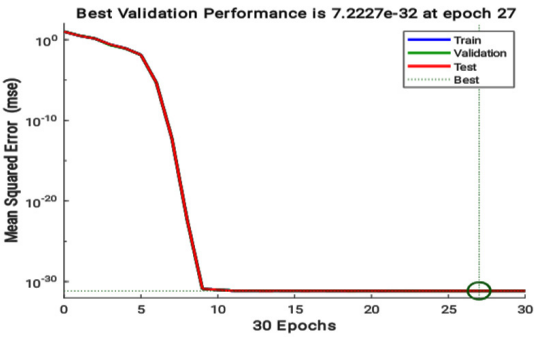

Figure 6

The error between the exact & neural solutions of the MLM of example 2.

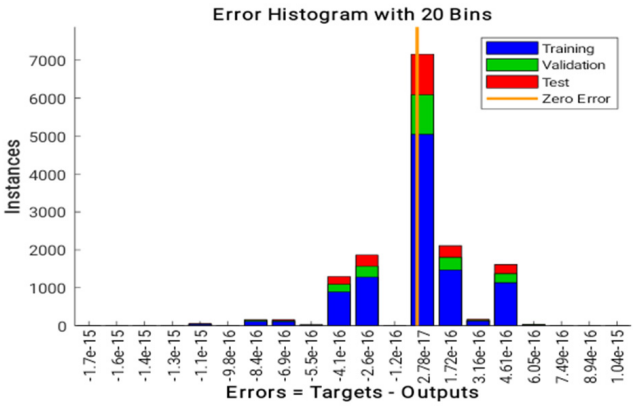

Figure 7

MSE of results NN for train, test, & validation of Example 2

Utilizing one genuine software failure datasets, performed parameter estimation to assess the efficacy of the suggested neural network model. The goodness-of fit a term used to describe how well software reliability growth models match historical software failure data is used to assess the models' performance. The issue of "How well does a model align with the available data" is raised by goodness of fit.

References

[1] R. K. K. Ajeena and H. Kamarulhaili, "On the distribution of scalar kfor elliptic scalar multiplication," *AIP Conf. Proc.*, vol. 1682, no. 1, Art. no. 020052 (2015).

[2] S. J. Mohammed and M. A. Mohammed, "Runge–Kutta numerical method for solving a nonlinear influenza model," *J. Phys.: Conf. Ser.*, vol. 1879, Art. no. 032040, pp. 1–15 (2021).

[3] S. J. Mohammed and M. A. Mohammed, "Mean Latin hypercube Runge–Kutta method to solve the influenza model," *Iraqi J. Sci.*, vol. 63, no. 3, pp. 1158–1177 (2022).

[4] F. F. Ghazi, "Modeling the contamination of soil adjacent to Mohammed Al-Qassim Highway in Baghdad," *Iraqi J. Sci.*, vol. 61, no. 10, pp. 2663–2670 (2020).

[5] M. H. Hashem and R. K. K. Ajeena, "The tensor product bipartite graph for a symmetric encryption scheme," *AIP Conf. Proc.*, vol. 2591, no. 1, Art. no. 050003 (2023).

[6] R. K. K. Ajeena, "Integer matrix size 2×2sub-decomposition method for elliptic curve cryptography," *Comput. Sci.*, vol. 18, no. 4, pp. 599–606 (2023).

[7] H. J. Jamil and M. R. A. Albahri, "Hookah smoking with health risk perception of different types of tobacco," *J. Phys.: Conf. Ser.*, vol. 1664, no. 1, Art. no. 012127 (2020).

[8] M. H. Ali and L. N. Tawfiq, "Design of an optimal neural network for solving the unsteady-state confined aquifer problem," *Math. Model. Eng. Probl.*, vol. 10, no. 2 (2023).

Received March, 2025