IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 22, Issue 2, Ser. 1 (Mar. – Apr. 2026), PP 34-42
www.iosrjournals.org

Solution Of 2nd Order Stiff Ordinary Differential Equations Using Feed Forward Neural Network And Bayesian Regularization Algorithm

Rootvesh Mehta
Department Of Mathematics, Indus University, Ahmedabad, Gujarat, India

Abstract:
In the present work, an advanced Feed Forward Neural Network (FFNN) and Bayesian regularization algorithm-based method is implemented to solve 2nd order stiff ordinary differential equations and system of ordinary differential equations. Using proposed method, the problems are solved for various time steps and comparisons are made with available analytical solutions and other existing methods. Various test problems have been simulated using proposed FFNN model and accuracy has been acquired with less calculation efforts and time. The outcome of the work is showing good hope to use artificial neural network methods to solve various types of higher order linear and nonlinear stiff differential equations in near future.

Keywords: *Feed Forward Neural Network, Multilayer Perceptron Neural Network, Stiff Ordinary Differential Equations, Back Propagation Algorithm, Bayesian Regularization Algorithm.*

Date of Submission: 14-03-2026 Date of Acceptance: 24-03-2026

I. Introduction

Differential equations originate from the fundamental balance laws governing science and engineering systems. Ordinary differential equations (ODEs) and partial differential equations (PDEs) naturally arise in the mathematical modeling of numerous engineering and physical processes. Among these, a special class known as stiff differential equations has received considerable attention due to their frequent occurrence in time-dependent physical systems and engineering applications [1,2].

In many practical situations, mathematical models lead to second-order stiff ordinary differential equations and systems of differential equations**,** particularly when modeling dynamic phenomena involving acceleration, oscillations, and diffusion processes. Such equations commonly appear in the analysis of electrical circuits, mechanical vibration systems, chemical reaction kinetics, atmospheric chemistry, control systems, and biological processes**.** The stiffness in these equations generally arises from the presence of multiple time scales, where certain components of the solution vary rapidly while others evolve more slowly.

A second-order stiff differential equation is typically characterized by the existence of solution components that decay exponentially as time progresses, while the derivatives associated with these components may exhibit significantly larger magnitudes compared to the solution itself. As a consequence, although the rapidly decaying components approach zero quickly, their derivatives remain relatively large, which introduces significant numerical challenges in obtaining stable and accurate solutions [3].

A variety of numerical techniques have been developed for solving differential equations; however, many of these methods face serious difficulties when applied to stiff problems. In particular, traditional explicit numerical schemes possess limited stability regions and therefore require extremely small step sizes to maintain numerical stability. This restriction substantially increases computational cost and makes explicit methods inefficient for solving stiff differential equations. Moreover, the presence of both transient and steady-state components in the solution often results in computational ill-conditioning, further complicating the numerical treatment of stiff systems [4].

To address these challenges, researchers have explored alternative computational approaches. One promising methodology involves the application of Artificial Neural Networks (ANNs) for the numerical solution of differential equations. Earlier attempts to solve stiff differential equations using neural networks encountered difficulties related to training efficiency and convergence. However, recent developments in advanced training algorithms, including the Levenberg–Marquardt algorithm**,** Bayesian Regularization algorithm**,** and Scaled Conjugate Gradient algorithm, have significantly improved the capability of neural network models to approximate solutions of stiff differential equations with satisfactory accuracy.

In the present study, an attempt is made to solve second-order stiff ordinary differential equations and systems of stiff differential equations using a Feed Forward Neural Network (FFNN) model integrated with

advanced training algorithms such as the Bayesian Regularization algorithm**,** Levenberg–Marquardt algorithm, and Scaled Conjugate Gradient algorithm. The performance of these training approaches is evaluated through comparative analysis of the obtained results, and conclusions are drawn based on their accuracy and computational effectiveness.

II. Materials And Methods

Artificial Neural Networks (ANNs) are computational frameworks inspired by the structure and functioning of biological neural systems. They consist of a large number of simple processing units known as neurons, which are interconnected through weighted links forming a directed network. In this network structure, the nodes represent neurons, while the connecting edges correspond to synaptic connections that transmit information between neurons [6]. Due to their ability to approximate complex nonlinear relationships, ANNs have become effective tools for solving a wide range of problems in science and engineering.

An ANN generally adopts a layered architecture consisting of three primary components: an input layer, one or more hidden layers, and an output layer. Each layer comprises several interconnected neurons that perform computations using specific activation functions. The input layer receives external information, which is then propagated through the hidden layers where nonlinear transformations are performed. Finally, the processed information is delivered to the output layer, which produces the network's predicted response.

The learning capability of an ANN is achieved through an adaptive training process in which the network parameters are adjusted to minimize the difference between the predicted outputs and the desired target values. During this training phase, the knowledge acquired from the input data is encoded within the network in the form of synaptic weights associated with the connections between neurons. These weights determine the strength of signal transmission and play a crucial role in the network's predictive performance.

In the computational process, input signals are initially fed into the input layer and are subsequently propagated through the network via weighted connections. At each neuron, the incoming signals are aggregated and transformed through an activation function before being forwarded to the subsequent layer. This feed-forward propagation mechanism allows the network to capture complex functional relationships between input and output variables.

Once the network has been adequately trained, the optimized weight parameters are used to evaluate new input data during the testing phase. The ability of ANN models to perform parallel processing and nonlinear approximation makes them particularly suitable for solving stiff differential equations and complex dynamical systems, where traditional numerical methods may encounter stability and computational limitations. The hidden layers then link to an output layer as shown in figure 1.

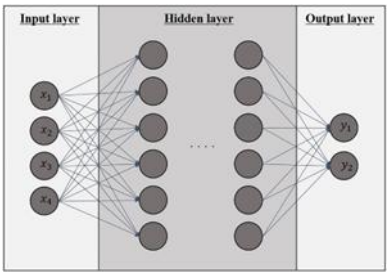

Figure 1: ANN Model

In the Feed Forward Neural Network architecture which is used in proposed work, the layers are formed using neurons. In a layer the neurons are receiving input from the previous layer and the next layer is fed by their output. Here, the data is strictly moving from input to output nodes in forward way. There is no feedback loops generated, that is why the output of any layer does not affect the same layer [7]. The FFNN architecture works on two models namely Single Layer Perceptron (SLP) Model and Multilayer Perceptron (MLP) Model. The Single-layer perceptron network is the simplest kind of neural network which consists of output nodes in a single layer only. The series of weights feed inputs to the outputs and in each of the neuron node the sum of products of the weights and the inputs are calculated. If $X= (x_1, x_2,\ldots, x_n)$ denotes the input vector, $W = w_{ij}$ denotes the weight matrix or connection matrix, and [WX] is the net input value, which is a scalar product of input vectors and weight vectors w_{ij}, f is the activation function and $O = (o_1, o_2, \ldots, o_m)$ is the output vector, then the block diagram of the single layer perceptron feed-forward ANN can be simply presented as figure 2.

Figure 2: The block diagram of the single layer perceptron feed-forward neural network

The Multilayer Perceptron (MLP) within the Feed Forward Neural Network (FFNN) framework represents a more generalized and powerful network architecture [8–10]. Unlike the single-layer perceptron, the MLP consists of one or more hidden layers positioned between the input and output layers, enabling the network to model complex nonlinear relationships.

In this architecture, the neurons in the hidden layers do not interact directly with the external environment; instead, they process information received from preceding layers and forward the transformed outputs to subsequent layers. Specifically, the neurons in the first hidden layer receive inputs from the source nodes of the input layer. The neurons in each subsequent hidden layer then receive inputs from the outputs of the previous hidden layer. This hierarchical processing continues until the signals reach the output layer.

The hidden layers perform intermediate computations through weighted summation and activation functions before transmitting the processed information to the output layer. The connections between the input layer and the hidden layers are characterized by input–hidden layer weights, while the connections between the hidden layers and the output layer are defined by hidden–output layer weights. These weights play a crucial role in determining the network's ability to approximate complex functional mappings.

Thus, the MLP architecture enhances the representational capability of the FFNN by introducing multiple layers of nonlinear processing units. A typical schematic representation of the FFNN Multilayer Perceptron model is illustrated in Figure 3.

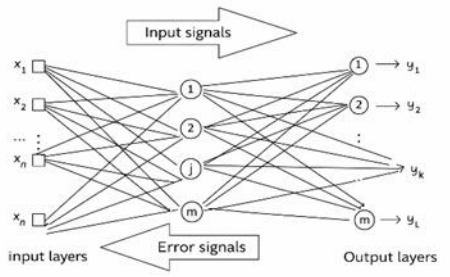

Figure 3: The block diagram of the multilayer perceptron feed-forward neural network

One of the most significant features of an Artificial Neural Network (ANN) is its ability to learn from data and generate appropriate outputs based on training. This learning capability enables the network to approximate complex functional relationships between input and output variables. Depending on the nature of the problem and the availability of target data, the learning process can be categorized into supervised learning, unsupervised learning, reinforcement learning, and competitive learning.

To transform the aggregated input signal into a meaningful output, each neuron employs an activation function. The activation function operates on the net input received by the neuron and introduces nonlinearity into the network, thereby enhancing its capability to model complex systems. It also ensures that the output of the neuron is constrained within a specific range, typically between 0 and 1 or −1 and 1, depending on the chosen function.

Commonly used activation functions include the sigmoid function, hyperbolic tangent (tanh) function, and rectified linear unit (ReLU), among others. These functions play a crucial role in determining the performance and convergence behavior of the neural network. A graphical representation of some widely used activation functions is presented in Figure 4.

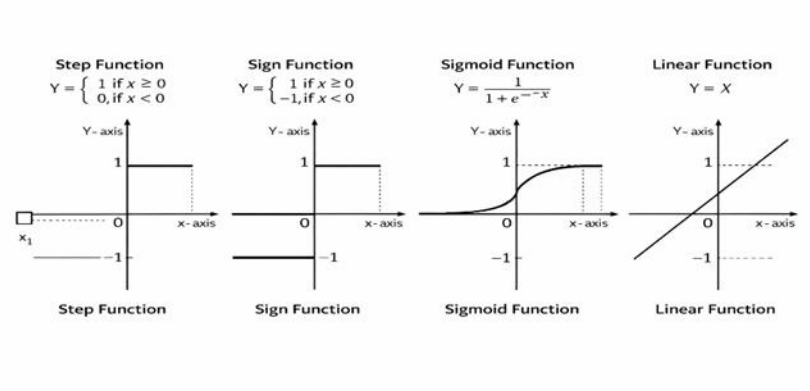

Figure 4: The list of activation functions

In the proposed work, a mathematical model based on a Feed Forward Neural Network (FFNN) is developed using the log-sigmoid activation function, followed by the training phase of the network. The training process involves adjusting the weights associated with the connections between neurons across different layers, with the objective of minimizing the error and achieving the desired output.

Training in neural networks can broadly be classified into two categories: supervised learning and unsupervised learning. In supervised learning, the network is trained using a set of input-output pairs, where the correct output (target) corresponding to each input is known. The learning process aims to minimize the difference between the predicted output and the target output, thereby improving the network's performance. This approach is particularly suitable for problems where labeled data is available.

In contrast, unsupervised learning does not rely on predefined output labels. Instead, the network identifies inherent patterns, structures, or regularities present in the input data. It organizes the data into clusters based on similarity, and new clusters may be formed when incoming patterns do not match existing groups. Since no explicit error signal is provided, the network undergoes a self-organizing process, during which its parameters are adjusted based on the input distribution.

However, for the Multilayer Perceptron (MLP) based FFNN model considered in this study, supervised learning is more appropriate due to its ability to accurately map input-output relationships. In the present work, backpropagation-based supervised training has been employed to optimize the network parameters. Specifically, advanced training algorithms such as the Levenberg–Marquardt algorithm [11,12], Bayesian Regularization algorithm [13,14], and Scaled Conjugate Gradient algorithm [15] are utilized to enhance convergence efficiency and obtain an accurate approximation of the solution in a closed analytical form.

Proposed Model

To understand the method, consider a first order ODE

$\frac{d\psi}{dx} = f(x,\psi)$ (1)

with $x \in [0,1]$ and the initial condition $\psi(0) = A$.

Now, the trial solution of Equation (1) is

$\psi(x) = A(x) + xN(x,\vec{p})$ (2)

where, $N(x,\vec{p}) = \sum_{i=1}^{h} v_i\sigma(z_i), z_i = w_i x + u_i$

is the output of a network with one node for x and $\vec{p}$.

Here, we assume trial function such that it satisfies the given initial condition. Now, differentiating both sides of equation (2), we get

$\frac{d\psi}{dx} = N(x,\vec{p}) + x\frac{dN(x,\vec{p})}{dx}$ (3)

We also know that

$$\frac{dN(x,\vec{p})}{dx} = \sum_{i=1}^{h} v_i w_i \sigma(z_i)$$

and hence, we get

$\frac{d\psi_i}{dx} = \sum_{i=1}^{h} v_i v_i \sigma(z_i) + x\sum_{i=1}^{h} v_i w_i v_i \sigma(z_i)$ (4)

Now, the error quantity to be minimized as

$E\overrightarrow{(p)} = \sum_k \left\{\frac{d\psi_k(x_k)}{dx} - f(x_k,\psi_k(x_k))\right\}^2, x_k \epsilon [0,1]$ (5)

By updating the parameters using advance algorithms like Levenberg Marquardt algorithm, Bayesian regularization algorithm and Scaled Conjugate Gradient algorithm, we can get results close to the analytic solution of the problem, and by comparing these results we can find appropriate method to solve 2nd order stiff Ordinary Differential Equation (ODE) as well as system of stiff ODEs.

Solution of second order linear stiff ODE.

In the present work attempt is made to solve 2nd order linear stiff ordinary differential equations and system of linear and non-linear stiff ordinary differential equations using neural network methods. The problems are solved for various time steps and their comparisons are presented with available analytical solutions and numerical solutions.

Second Order Linear Stiff ODE –Here we tried to solve Mass Spring Damper Problem (Applications in study of motion of spring and mass systems) which is Second Order Linear Stiff ODE

$\ddot{x} + c\dot{x} + kx = 0, x(0) = 0, \dot{x}(0) = 1, c = 300, k = 60$ (6)

The Analytical solution is

$$x = \frac{e^{-2t} - e^{-300t}}{298}$$

We have solved Linear, Non-linear stiff system of ODEs with boundary conditions by applying presented method. Once again comparison of generated ANN solution with analytical solution is presented which suggests ANN approach provides satisfactory result at required value of independent variable x with less computational work, by using less time and memory.

The System is

$$\begin{aligned} y_1' &= -100y_1 + 9.901y_2 \\ y_2' &= 0.1y_1 - y_2, x \in [0,10] \\ with\ y_1(0) &= 1, y_2(0) = 10 \end{aligned} \quad (7)$$

The Analytical Solution is

$$y_1(x) = e^{-0.99x}, y_2(x) = 10e^{-0.99x} \quad (8)$$

III. Results

Table - 1 ANN solution Mass-Damping System (h=0.1) Equation (6)

Analytical Sol. h=0.1		ANN solution	
t	x	output	Error
0	0	0.002632779	-0.002632779
0.1	0.002747419	0.002417469	0.00032995
0.2	0.002249396	0.002176996	7.24E-05
0.3	0.00184165	0.001919378	-7.77E-05
0.4	0.001507815	0.001655423	-0.000147607
0.5	0.001234495	0.001397037	-0.000162543
0.6	0.001010719	0.001155161	-0.000144442
0.7	0.000827507	0.000938029	-0.000110523
0.8	0.000677505	0.000750321	-7.28E-05
0.9	0.000554694	0.000593263	-3.86E-05
1	0.000454145	0.000465404	-1.13E-05
1.1	0.000371823	0.000363617	8.21E-06
1.2	0.000304423	0.000284019	2.04E-05
1.3	0.00024924	0.000222637	2.66E-05
1.4	0.000204061	0.000175812	2.82E-05
1.5	0.000167071	0.000140383	2.67E-05
1.6	0.000136786	0.000113746	2.30E-05
1.7	0.000111991	9.38E-05	1.82E-05
1.8	9.17E-05	7.89E-05	1.27E-05
1.9	7.51E-05	6.79E-05	7.18E-06
2	6.15E-05	5.97E-05	1.78E-06

Table 2 ANN solution Mass-Damping System (h=0.05) Equation (6)

ANN Sol. of Mass Damping problem			
h=0.05		ANN solution	
t	x	output	Error
0	0	0.001719766	-0.001719766
0.05	0.003036366	0.001677302	0.001359064
0.1	0.002747419	0.001631935	0.001115483
0.15	0.002485967	0.001583693	0.000902275
0.2	0.002249396	0.001532646	0.00071675
0.25	0.002035338	0.001478915	0.000556423
0.3	0.00184165	0.001422671	0.000418978
0.35	0.001666394	0.001364137	0.000302257
0.4	0.001507815	0.001303584	0.000204232
0.45	0.001364328	0.001241332	0.000122996
0.5	0.001234495	0.001177742	5.68E-05
0.55	0.001117017	0.001113208	3.81E-06
0.6	0.001010719	0.00104815	-3.74E-05
0.65	0.000914536	0.000983002	-6.85E-05

ANN Sol. of Mass Damping problem			
h=0.05		ANN solution	
t	x	output	Error
0.7	0.000827507	0.000918198	-9.07E-05
0.75	0.000748759	0.000854165	-0.000105406
0.8	0.000677505	0.000791309	-0.000113804
0.85	0.000613032	0.000730007	-0.000116975
0.9	0.000554694	0.000670595	-0.000115901
0.95	0.000501908	0.000613366	-0.000111458
1	0.000454145	0.000558565	-0.000104419
1.05	0.000410928	0.000506382	-9.55E-05
1.1	0.000371823	0.000456959	-8.51E-05
1.15	0.000336439	0.000410388	-7.39E-05
1.2	0.000304423	0.000366714	-6.23E-05
1.25	0.000275453	0.00032594	-5.05E-05
1.3	0.00024924	0.000288033	-3.88E-05
1.35	0.000225522	0.00025293	-2.74E-05

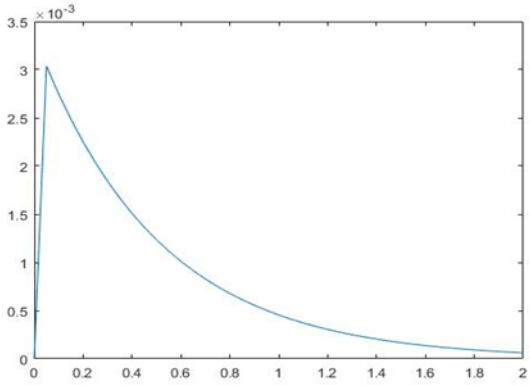

Figure 5 Analytical solution of Mass Spring Damper Problem

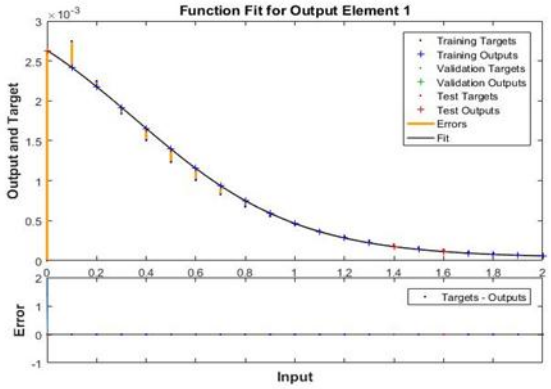

Figure 6 ANN solution of Mass Spring Damper Problem h=0.1

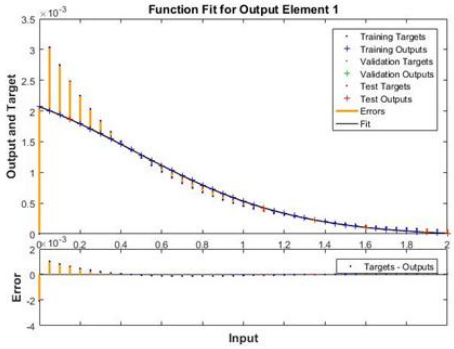

Figure 7 ANN solution of Mass Spring Damper Problem h=0.05

We applied MLP neural network approach with ANN Curve fitting toolbox in Matlab with 3 layers FFNN. One input layer, one hidden layer and one output layer, in training Bayesian Regularization Algorithm is used for obtaining required accuracy.

Table 3 Analytical Sol.(h=1) & ANN Sol. of System of ODEs (7)

Analytical Sol.(h=1)			ANN Sol.(h=1)			
x	y1	y2	y1	y2	Error (y1)	Error (y2)
0	1	10	0.999986272	9.999708273	1.37E-05	2.92E-04
1	0.371576691	3.71576691	0.371624065	3.717075347	-4.74E-05	-1.31E-03
2	0.138069237	1.380692373	0.137604731	1.379622089	0.000464507	1.07E-03
3	0.05130331	0.513033103	0.050908895	0.511966459	3.94E-04	1.07E-03
4	0.019063114	0.190631143	0.018854098	0.190169771	2.09E-04	4.61E-04
5	0.007083409	0.070834089	0.007010798	0.070859348	7.26E-05	-2.53E-05
6	0.00263203	0.026320297	0.002636222	0.026628602	-4.19E-06	-3.08E-04
7	0.000978001	0.009780009	0.001020539	0.010232097	-4.25E-05	-4.52E-04
8	0.000363402	0.003634023	0.000423833	0.00415395	-6.04E-05	-5.20E-04
9	0.000135032	0.001350318	0.000203459	0.001900807	-6.84E-05	-5.50E-04
10	5.02E-05	0.000501747	0.000122072	0.001065579	-7.19E-05	-5.64E-04

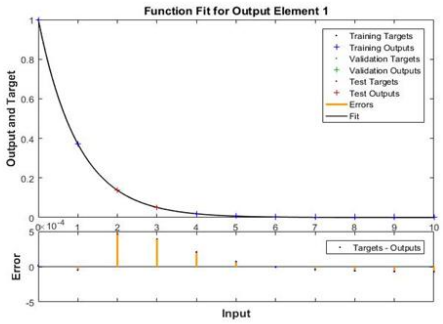

Figure 8 ANN Sol. Sys. (7), h=1, y1

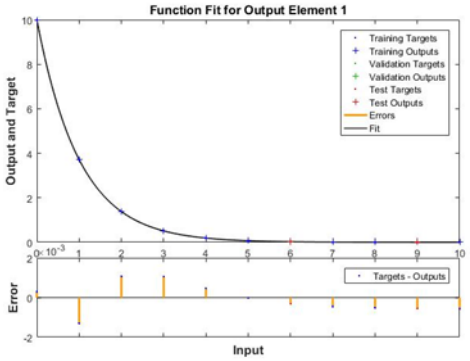

Figure 9 ANN Sol. Sys. (7), h=1, y2

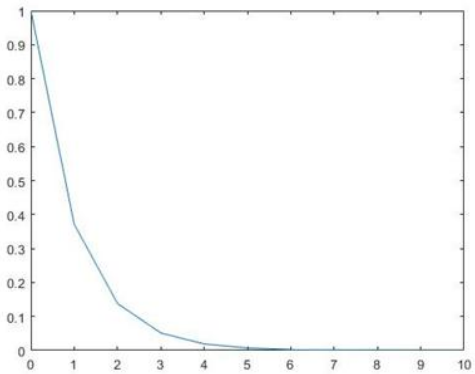

Figure 10 Analytical Sol. Sys. (7), h=1, y1

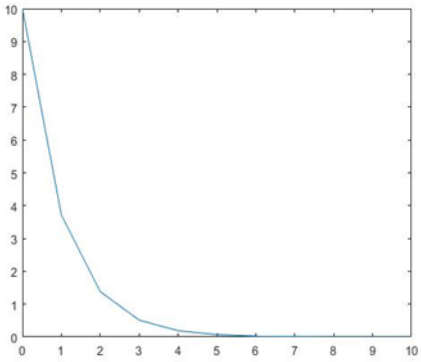

Figure 11 Analytical Sol. Sys. (7), h=1, y2

We solved the stiff system with larger step-size and using ANN approach we find here better solution of system of ODEs (7)

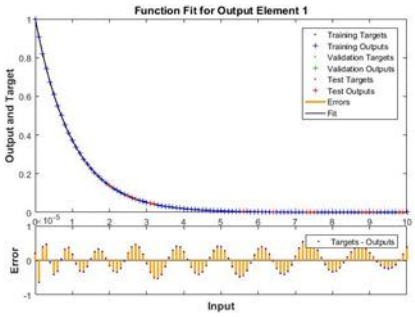

Figure 12 ANN Sol. Sys. (7), h=0.1, y1

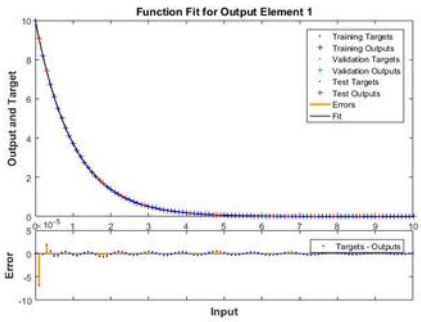

Figure 13 ANN Sol. Sys. (7), h=0.1, y2

IV. Conclusion

In the present study, an attempt has been made to solve second-order stiff ordinary differential equations and systems of stiff ODEs with boundary conditions using a Feedforward Neural Network (FFNN) based Multilayer Perceptron (MLP) model. The model integrates advanced training algorithms, namely the Bayesian Regularization algorithm, Levenberg–Marquardt algorithm, and Scaled Conjugate Gradient algorithm.

The primary objective of this work is to obtain accurate approximate solutions while minimizing computational complexity, memory requirements, and time consumption. Among the considered algorithms, the Bayesian Regularization approach, although comparatively more time-consuming than the Levenberg–Marquardt and Scaled Conjugate Gradient methods, demonstrates superior performance in terms of accuracy, stability, and reduced computational error, while also requiring relatively less memory and fewer effective parameters.

Furthermore, extensive experimentation has been carried out by varying key network parameters such as the number of neurons in hidden layers, step size, dataset size, and choice of training algorithm. A comparative analysis of these variations highlight that the FFNN model trained with the Bayesian Regularization algorithm yields more reliable and efficient results for solving stiff differential equations.

Based on the promising outcomes obtained, the proposed methodology can be effectively extended to solve higher-order stiff ordinary differential equations and more complex real-world applications involving stiff systems. Therefore, this approach opens new avenues for future research in the numerical and computational treatment of stiff differential equations.

References

[1]. Suyong K., Weiqi J., Sili D., Yingbo M., Christopher R., "Stiff Neural Ordinary Differential Equations," Chaos: An Interdisciplinary Journal Of Nonlinear Science, Vol. 31, No. 9, Pp. 093122, 2021. DOI: 10.1063/5.0060697

[2]. Weiqi J., Weilun Q., Zhiyu S., Shaowu P., Sili D., "Stiff-PINN: Physics-Informed Neural Network For Stiff Chemical Kinetics," Journal Of Physical Chemistry, Vol. 125, No. 36, Pp. 8098-8106, 2021. DOI: 10.1021/Acs.Jpca.1c05102

[3]. Richard L. Burden J., Douglas F, "Numerical Analysis", Cengage Learning, USA, 2011.

[4]. Miranker W.L.,"The Computational Theory Of Stiff Differential Equations", Publicazioni IAC Roma Ser. III N. 102, 1975.

[5]. Malhotra S., "Simulation Of Navier Stokes Equation Represents The Transient Model Of Single Tube Heat Exchanger With Vapor-Liquid Two Phase Flow Inside", International Journal Of Emerging Technologies In Computational And Applied Sciences, Vol. 5, No. 5, Pp. 540-547, 2013. Http://Citeseerx.Ist.Psu.Edu/Viewdoc/Download?Doi=10.1.1.380.7741&Rep=Rep1&Type=Pdf

[6]. M. Minsky And S. Papert. Perceptrons. MIT Press, Cambridge, MA, 1969.

[7]. T. Khanna. Foundations Of Neural Networks. Addison-Wesley Press, Boston, USA, 1990.

[8]. Neha Y., Anupam Y., Manoj K., "An Introduction Of Neural Network Methods For Differential Equations" Springer Briefs In Applied Sciences And Technology, 2015.

[9]. Kumar M, Yadav N "Multilayer Perceptrons And Radial Basis Function Neural Network Methods For The Solution Of Differential Equations: A Survey," Computers And Mathematics With Applications, Vol. 62, No. 10, Pp. 3796–3811, 2011. DOI: 10.1016/J.Camwa.2011.09.028

[10]. Kumar M, Yadav N "Numerical Solution Of Bratu's Problem Using Multilayer Perceptron Neural Network Method" National Academy Science Letters. Volume 38, No 5, 425–428. 2015 DOI: 10.1007/S40009-015-0359-3.

[11]. Marquardt, D., "An Algorithm For Least-Squares Estimation Of Nonlinear Parameters," SIAM Journal On Applied Mathematics, Vol. 11, No. 2, Pp. 431–441, 1963. DOI: 10.1137/0111030

[12]. Hagan, M.T., And M. Menhaj, "Training Feed-Forward Networks With The Marquardt Algorithm," IEEE Transactions On Neural Networks, Vol. 5, No. 6, Pp. 989–993, 1994. DOI: 10.1109/72.329697

[13]. Mackay, David J. C. "Bayesian Interpolation." Neural Computation. Vol. 4, No. 3, Pp. 415–447, 1992. DOI: 10.1162/Neco.1992.4.3.415

[14]. Foresee, F. Dan, And Martin T. Hagan. "Gauss-Newton Approximation To Bayesian Learning." Proceedings Of The International Joint Conference On Neural Networks, Vol. 3, No. 3, Pp. 1930-1935, 1997. DOI: 10.1109/ICNN.1997.614194

[15]. Moller M.F., "A Scaled Conjugate Gradient Algorithm For Fast Supervised Learning" Neural Networks, Elsevier, Vol. 6, No. 4, Pp. 525–533, 1993. DOI: 10.1016/S0893-6080(05)80056-5