Journal of Engineering and Applied Sciences 14 (23): 8576-8584, 2019
ISSN: 1816-949X

Solving Ordinary Differential Equations with Neural Networks and using PSO Algorithm

[1]Saadat Behzadi and [2]Maliheh Miri
[1]Department of Electrical Engineering, AmirKabir University, Tehran, Iran
[2]Department of Electrical Engineering, Faculty of Engineering,
Higher Educational Complex of Saravan, Saravan, Iran
BehzadiSaadat@aut.ac.ir

Abstract: In this study, a novel hybrid method is presented for the solution of Ordinary Differential Equations (ODEs) with neural network that is trained by using PSO algorithm. Although, many studies for solving ODEs are available now, this method has more advantages such as fast convergence and also little error. A solution of ODE is written as a sum of two parts. The first part involve no adjustable parameters that satisfies the initial condition and the second part contains a feed forward neural network containing adjustable parameters which use the PSO algorithm. Therefore, by using both parts satisfied the initial condition and also the neural network is train to solve ODEs. The proposed method is applicable to solve ordinary differential equations and systems of Ordinary Differential Equations (SODEs). Finally, there are several examples to analysis sensitivity of the convergence.

Key words: Neural network, differential equation, PSO algorithm, sensitivity, adjustable, convergnece

INTRODUCTION

Most of the problems in engineering can be transformed into a series of differential equations through mathematical modeling. In most of the cases, obtaining a precise solution for differential equations is not, so, simple and sometimes analytic solutions for these equations are not generally, available which makes us utilize numerical solutions.

In recent years because of the importance of differential equations, many methods have been proposed for solving these equations such as radial basis functions (Fasshauer, 1999; Franke, 1998), artificial neural networks (Lee and Kang, 1990; Ramuhalli *et al*., 2005; Lagaris *et al*., 1998; Puffer *et al*., 1995; Parisi *et al*., 2003; Yentis and Zaghloul, 1996; Meade and Fernandez, 1994a, b; Jianyu *et al*., 2002; He *et al*., 2000) and so forth. In this study, an artificial neural network which is based on orthogonal functions is used in order to solve linear and nonlinear Ordinary Differential Equations (ODEs) and systems of Ordinary Differential Equations (SODEs). Currently, there have been a variety of algorithms used to train the neural network such as Back-Propagation Algorithm (BPA), Genetic Algorithm (GA) (Angeline *et al*., 1994; Yao, 1993), Simulating Annealing Algorithm (SAA) (Koh *et al*., 1994; Shaw and Kinsner, 1996) and so forth.

Neural networks

Feedforward neural networks: A feedforward neural network was the simplest type of artificial neural network proposed. They are one of the most common models in many practical applications. The neural networks are mainly used in order to estimate or approximate functions and solving differential equations. The structure of a two layered feedforward neural network is shown in Fig. 1. Assuming that the hidden and output transport function is a linear activation function (Zhang *et al*., 2007).

From Fig. 1, n is the number of the input node, w_{ij} is the connection weight from the ith node of the input layer to the jth node of hidden layer; x_i is the ith input, p_i is the weight input sum in hidden and o is the number of output.

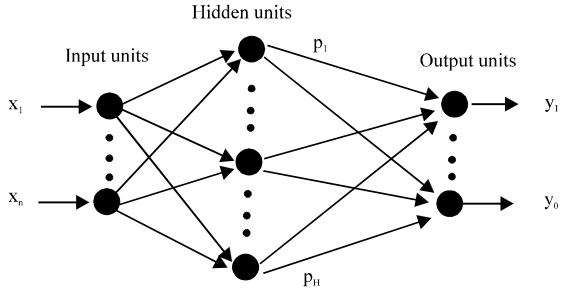

Fig. 1: A two-layered feed forward neural network structure (Zhang *et al*., 2007)

Corresponding Author: Saadat Behzadi, Department of Electrical Engineering, AmirKabir University, Tehran, Iran, BehzadiSaadat@aut.ac.ir

Particle swarm optimization

Particle Swarm Optimization (PSO): Is a population-based stochastic method for solving optimization problems which was introduced by Kennedy and Eberhart (1995). PSO took originally inspiration from movement and flocks of birds, fish school or swarms of bees. Every bird, fish or bee is called as a "particle". These "particles" move with a certain velocity and search for the global best position after some iteration in the searching space. Supposing D is the dimension for a searching space and n is total number of particles. The position of particle i is represented as vector $X_i = [x_{i1}, x_{i2}, ..., x_{iD}]$, the previous best position of particle i is denoted as vector $P_{ib} = [P_{i1}, P_{i2}, ..., P_{iD}]$ and the previous best position of the total particle is denoted as $P_{gb} = [P_{g1}, P_{g2}, ..., P_{gD}]$. A particle is moving and has a velocity. That's why the velocity of particle i can be expressed as vector $V_i = [v_{i1}, v_{i2}, ..., v_{iD}]$. For the t+1 particle movement, the velocity of each particle can be modified by Eq. 1:

$$V_{id}(t+1) = w * V_{id}(t) + c_1 * r_1 * (P_{ib} - X_{iD}) + c_2 * r_2 * (P_{gb} - X_{iD}) \quad (1)$$

In the above equation, w is called "inertial weight" and controls the influence of old velocity. c_1 and c_2 are positive constants that called as coefficient of the self-recognition component and coefficient of the social component, respectively. r_1 and r_2 are the random numbers in the interval [0, 1]. According to this information the new location of particle i is described as Eq. 2:

$$X_{id}(t+1) = X_{id}(t) + V_{id}(t+1) \quad (2)$$

Parameter values is very important to obtain best result. Different inertial weights and other constants have been proposed by a variety of authors. In this study, I used the value of 2 for the constants c_1 and c_2 and for the inertial weight w the following selection strategy is used Eq. 3:

$$w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} \times iter \quad (3)$$

Where:

w_{max} = The initial weight
w_{min} = The final weight
$iter_{max}$ = The maximum iteration number
iter = The current iteration number

From (Eq. 3), the inertial weight is gradually decreased. So, after a number of iterations the current searching point close to the global best result (gbest) and its own best result (pbest).

Shifted legendre polynomials: The legendre polynomials $\varphi_i(t)$, i = 0, 1, ..., n-1 on the interval $t \in [-1, 1]$ are defined and related as (Razzaghi and Shafiee, 1998; Perng, 1986):

$$\varphi_{i+1}(t) = \left(\frac{2i+1}{i+1}\right) t\varphi_i(t) - \left(\frac{i}{i+1}\right)\varphi_{i-1}(t)$$

i is order Legendre polynomial. The zero and the first order Legendre polynomials are given by:

$$\begin{cases} \varphi_0(t) = 1 \\ \varphi_1(t) = 1 \end{cases}$$

That's why some of the order polynomials are defined by:

$$\varphi_2(t) = \frac{3t^2 - 1}{2}$$

$$\varphi_3(t) = \frac{5t^3 - 3t}{2}$$

$$\varphi_4(t) = \frac{35t^4 - 30t^2 + 3}{8}$$

In order to use of Legendre polynomials on the interval of x = [0, T], it is necessary to shift it. The shifted Legendre polynomials are defined as (Perng, 1986):

$$\varphi_{i+1}(x) = \frac{2i+1}{i+1}\left(\frac{2x}{T} - 1\right)\varphi_i(x) - \left(\frac{i}{i+1}\right)\varphi_{i-1}(x)$$

$$\varphi_0(x) = 1$$

$$\varphi_1(x) = \frac{2x}{T} - 1$$

Here, the "shifting" function is chosen by t = 2x/T-1.

MATERIALS AND METHODS

Illustration of the method

Solution of single ODE's: An Ordinary Differential Equation (ODE) can be represented in Eq. 4:

$$F\left(x, y, \dot{y}, ..., y^{(n)}\right) = 0, \quad x \in [a, b] \quad (4)$$

Equation 4 is an ordinary differential equation of order n where $y^{(n)}$ is the nth-order derivative of y and [a, b] denotes the problem domain. The n initial conditions needs for solving (Eq. 4) which can be expressed in Eq. 5:

$$y(x_0) = y_{x_0}, \; y(x_1) = y_{x_1}, ..., y(x_{n-1}) = y_{x_{n-1}} \quad (5)$$

The trial solution $\hat{y}(x)$ can be written by such a neural network as Eq. 6:

$$\hat{y}(x) = A(x, \phi) + H(x, N(x, w)) \quad (6)$$

where, Eq. 7:

$$\phi = y_{x_0}, y_{x_1}, ..., y_{x_{n-1}} \quad (7)$$

ϕ describes the initial conditions and A(x, ϕ) satisfies the initial conditions. N(x, w) is a feed forward neural network with one input vector x and weights w and H(x, N(x, w)) is a function which is zero in initial points. That's why H(x, N (x, w)) function can be expressed in Eq. 8:

$$H(x, N(x, w)) = \Psi \times N(x, w) \quad (8)$$

Ψ is used to reset to zero the H(x, N (x, w)) in initial points. All in all, Ψ and A(x, ϕ) are depending on the initial conditions and subtend no adjustable parameters. For several types of ODEs, the section of Ψ and A(x, ϕ) is represented in. In this study, the proposed neural network for solving ODEs is defined as Eq. 9:

$$N(x, w) = \sum_{i=1}^{m} w_i \varphi_i(x) \quad (9)$$

where, the element $\varphi_i(x)$ is the basis function, m is the number of these basis and w_i are adjustable parameters. If the solution is defined as $\hat{y}(x)$, it must satisfy Eq. 10:

$$F(x, \hat{y}, \dot{\hat{y}}, ..., \hat{y}^{(n)}) = 0 \quad (10)$$

That's why, the error function that must be minimized is Eq. 11:

$$e = F(x, \hat{y}, \dot{\hat{y}}, ..., \hat{y}^{(n)}) \quad (11)$$

Neural network training using PSO: Different researchers have proposed several strategies for training neural networks. In this study, we used the PSO algorithm. The advantages of PSO-NN defined as:

There are no constrains for the PSO method which is important for the Back Propagation (BP) algorithm. For example, in the BP the transfer function in hidden layer should be differentiable. So, more transfer functions can be selected to make different requirements.

PSO training algorithm is not easy to be trapped into local minimal. As a disadvantage, PSO method suffers from the "curse of dimensionality" in the other word the performance of PSO method deteriorates as the dimensionality of the search space increases.

Optimal variables in training of the neural network and PSO algorithm consist of weights, biases and number of basis function. The number of these variables is m which presented the number of necessary terms of polynomial. For this reason, at first generate the stochastic N initial population for algorithm. The network is running with these initial weights and error vector will be produced. The cost function for this algorithm is defined as Eq. 12:

$$J = \frac{1}{2} e^2 \quad (12)$$

P_{bi} and P_{gb} in PSO algorithm are computed and new position vector obtain. This algorithm iterated until the stopping criteria is met. The stopping criteria consists the minimum amount of error vector or a large number of iteration. The structure of the proposed approach for solving the ODEs is shown in Fig. 2 and 3.

Solution of system ODEs: Consider the following system of nonlinear Eq. 13:

$$\begin{cases} \dot{y}_1 = f_1(x, y_1, y_2, ..., y_n) \\ \\ \dot{y}_n = f_n(x, y_1, y_2, ..., y_n) \end{cases}, x \in [a, b] \rightarrow \dot{Y} = F(x, Y) \quad (13)$$

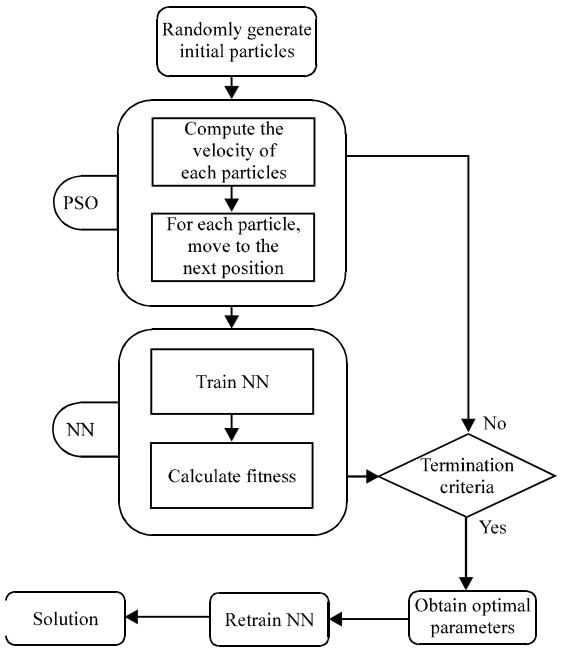

Fig. 2: Schematic diagram of solving differential equations

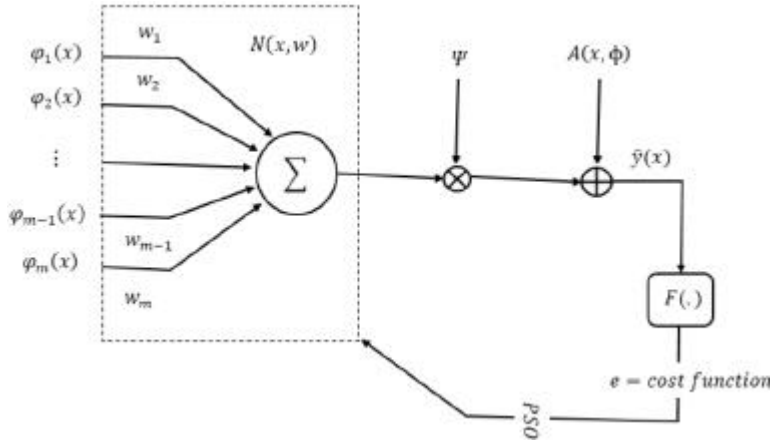

Fig. 3: Schematic diagram of PSO-legendre neural network

where, Eq. 14:

$$Y=[y_1 \quad y_2 \quad ,\cdots, \quad y_n]^T \qquad F=[f_1 \quad f_2 \quad ,\cdots, \quad f_n]^T \tag{14}$$

The initial conditions for solving SODEs (Eq. 13) can be expressed in Eq. 15

$$y_1(c)=y_{1c},\ \ y_2(c)=y_{2c},\ ...,\ y_n(c)=y_{nc} \tag{15}$$

The trial solution $\hat{y}(x)$ can be written by such a neural network as Eq. 16:

$$\hat{y}(x)=A_i(x,\phi)+H_i(x,N_i(x,w)) \quad i=1,2,...,n \tag{16}$$

The proposed neural network for solving SODEs is defined as Eq. 17:

$$N_i(x,w)=\sum_{j=1}^{m} w_{ij}\varphi_i(x) \quad j=1,2,...,n \tag{17}$$

The error function that must be minimized is given as Eq. 18:

$$\begin{cases} \dot{\hat{y}}_1=f_1(x,\hat{y}_1,\hat{y}_2,...,\hat{y}_n) \ \rightarrow \ e_1=\dot{\hat{y}}_1-f_1(\hat{y}_1,\hat{y}_2,...,\hat{y}_n) \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ \dot{\hat{y}}_n=f_n(x,\hat{y}_1,\hat{y}_2,...,\hat{y}_n) \ \rightarrow \ e_n=\dot{\hat{y}}_n-f_n(\hat{y}_1,\hat{y}_2,...,\hat{y}_n) \end{cases} \tag{18}$$

The error function can be rewritten as Eq. 19:

$$E=\dot{\hat{Y}}-F(\hat{Y}) \tag{19}$$

where:

$$E=[e_1 \quad e_2 \quad \cdots \quad e_n]^T \tag{20}$$

The cost function for PSO algorithm and solving SODEs is defined as Eq. 21:

$$J=\frac{1}{2}E^TE \tag{21}$$

The procedure for solving SODEs with PSO-gendre neural network algorithm can be summarized as follows:

PSO–Legendre neural network algorithm:

Step 1: Initialize the positive and arbitrary value of m
Step 2: Create the trial solution according to the (Eq. 16)
Step 3: Calculate the number of optimal parameter (P = m×n)
Step 4: Training Legendre neural network using PSO. Initialize the positions and velocities of particles randomly. Evaluate initialized particle's fitness value and determine P_{ib} and P_{gb}
Step 5: Evaluate the vector of estimation error
Step 6: update the positions and velocities and create new particles according to (Eq. 1 and 2)
Step 7: If cost function or P_{gb} is unchanged for fifteen iteration, go to step 9; else go to next step.
Step 8: if the maximal iteration number are arrived, go to step 10; else go to step 5
Step 9: m = m+1 and go to step 2
Step 10: output the global optimum

Figure 4 shows a schematic diagram of solving SODEs using PSO-Legendre neural network.

RESULTS AND DISCUSSION

Numerical simulations: In this study, the solution of a number of ODEs, non-linear ODEs and SODEs are reported. For each of the problems there is discussions about maximum iteration number and the initial value of number of basis function. The number of particles in all the examples is 70. Example 1, consider the following first order differential Eq. 22:

$$\dot{y}=\frac{2x-y}{x} \tag{22}$$

with y(1) = 3 and xϵ [1, 8]. Hence,

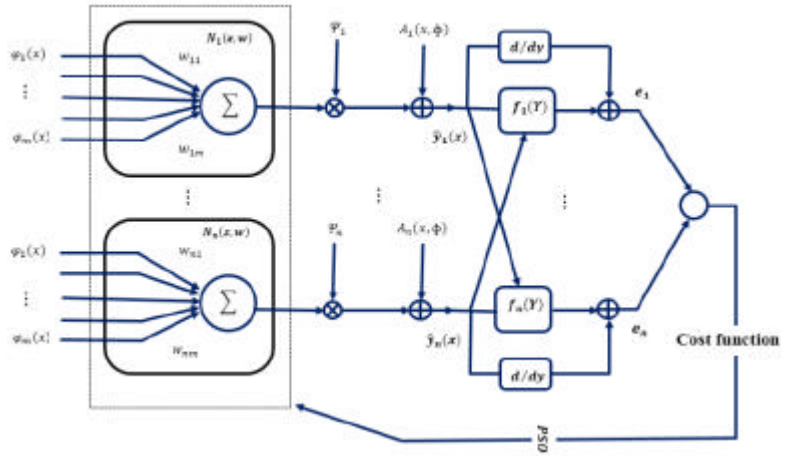

Fig. 4: Schematic diagram of solving SODEs using PSO-Legendre NN

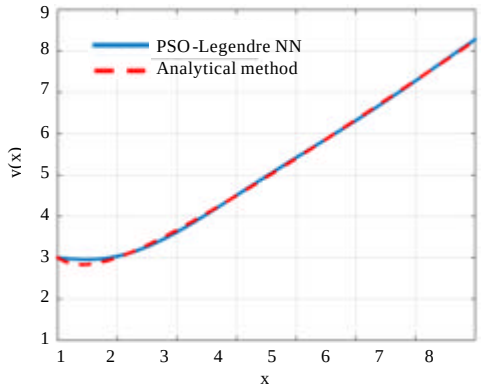

Fig. 5: Solution of example 1

$$A(x,\phi)=3,\ \Psi(x)=(x\text{-}1)$$

The number of iteration is set 75. That's way the trail solution can be written in the following form:

$$\hat{y}(x)=3+(x\text{-}1)\times\sum_{i=1}^{m}w_i\varphi_i(x)$$

The analytical solution is x+2/x. Now we can solve (Eq. 22) with PSO-Legendre NN. Analytical solution and obtained solution $(\hat{y}(x))$ are shown in Fig. 5. The cost function is plotted in Fig. 6. In this experiment, the interval x were discretized into 700 samples. Example 2, consider the following ordinary differential equation:

$$\dot{y}(x)=e^{\left(-\frac{x}{5}\right)}\cos(x)-\frac{1}{2}y(x)$$

with y (0) = 0 and xϵ[0, 10]. Therefore,

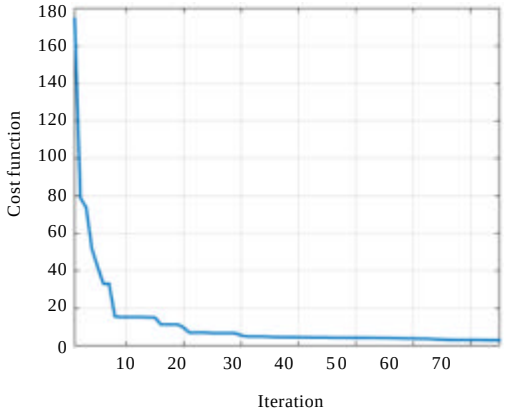

Fig. 6: Cost function for example 1

$$A(x,\phi)=0,\ \Psi(x)=(x)$$

The analytical solution $e^{(-x/5)}\sin(x)$.The number of iteration is set 100. The trail solution of this ODE can be written in the following form:

$$\hat{y}(x)=0+(x)\times\sum_{i=1}^{m}w_i\varphi_i(x)$$

Analytical solution and the solution that obtained with PSO-Legendre NN are plotted in Fig. 7. The cost function is plotted in Fig. 8. In this experiment, the interval x were discretized into 2000 samples. Example 3, consider the second ordinary differential equation:

$$\ddot{y}=2$$

with y(0) = 2 and y(1) = 14/5 and xϵ[0, 10]. Therefore,

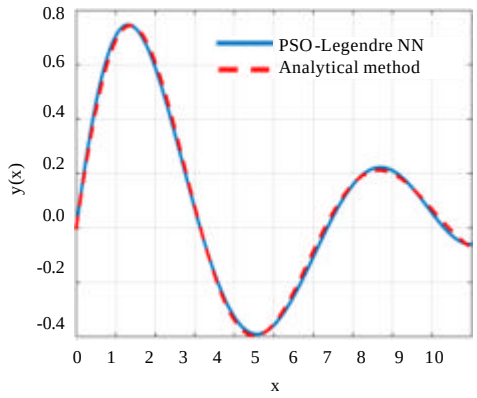

Fig. 7: Solution of example 2

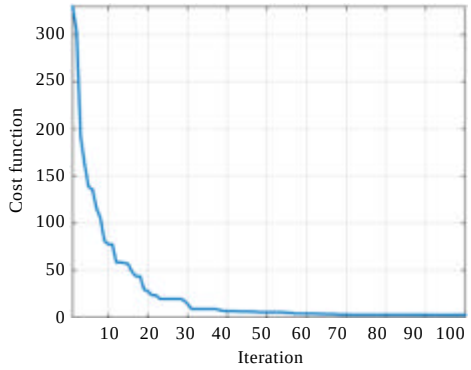

Fig. 8: Cost function for example 2

$$A(x,\phi)=2(1-x)+\frac{14}{5}x,\ \Psi(x)=(1-x)$$

The analytical solution is x^2-i/5x+2. The number of iteration is set 100. The trail solution of this ODE can be written in the following form:

$$\hat{y}(x)=2(1-x)+\frac{14}{5}x+x(1-x)\times\sum_{i=1}^{m}w_i\varphi_i(x)$$

Analytical solution and obtained solution with PSO-Legendre NN are illustrated in Fig. 9. The cost function is shown in Fig. 10. It is interesting to note that in this problem, the interval x were discretized into 500 samples. Example 4, consider the following non-linear ordinary differential equation:

$$(\dot{y})^2+\log(y)-\cos^2(x)-2\cos(x)-\log(x+\sin(x))-1=0$$

with y(0) = 2 and $x\epsilon[0, 10]$. Therefore:

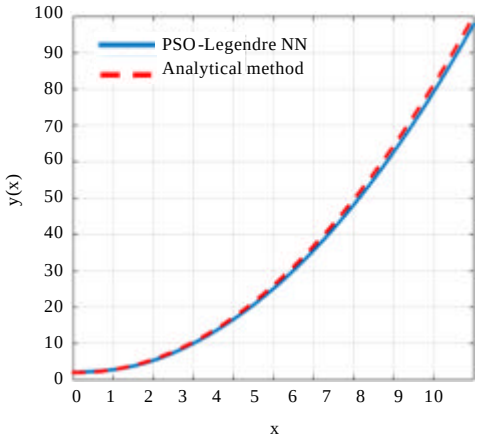

Fig. 9: Solution of example 3

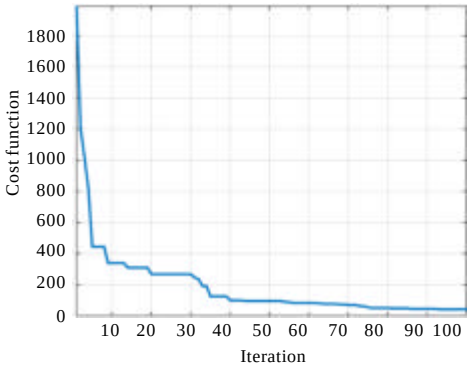

Fig. 10: Cost function for example 3

$$A(x,\phi)=0,\ \Psi(x)=(x)$$

The analytical solution is x+sin(x). The number of iteration is set 50. The trail solution of this non-linear ODE can be written in the following form:

$$\hat{y}(x)=0+(x)\times\sum_{i=1}^{m}w_i\varphi_i(x)$$

Analytical solution and obtained solution via. PSO-Legendre NN are compared in Fig. 11. The cost function is shown in Fig. 12. It is interesting to note that in this problem, the interval x were discretized into 1000 samples. Example 5, consider the following systems of ordinary differential equation:

$$\begin{cases}\dot{y}_1(x)=y_2(x)\\ \dot{y}_2(x)=\dfrac{\left(1+(y_2(x))\right)^2}{2y_1(x)}\end{cases}$$

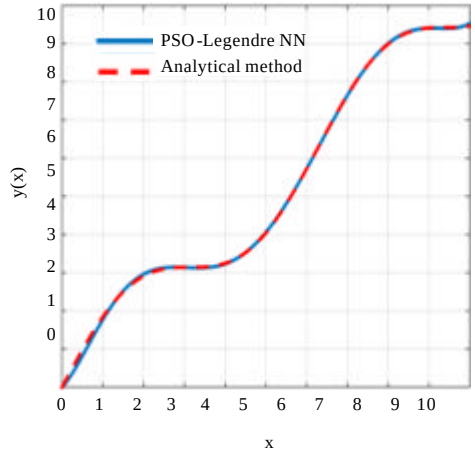

Fig. 11: Solution of example 4

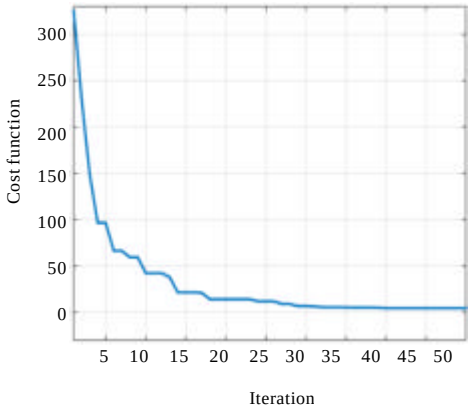

Fig. 12: Cost function for example 4

with $y_{10} = (1.25)$ and $y_{20} = (0.5)$ and $x\epsilon[0, 10]$. Therefore:

$$A_1(x, \phi) = 1.25,\ \Psi_1(x) = (x)$$
$$A_2(x, \phi) = 0.5,\ \Psi_2(x) = (x)$$

The analytical solution is:

$$\begin{cases} y_1(x) = 1 + \frac{1}{4}(1+x)^2 \\ y_2(x) = \frac{1}{2}(1+x) \end{cases}$$

The number of iteration is set 100. The trail solution of this ODE can be written in the following form:

$$\hat{y}_1(x) = 1.25 + (x) \times \sum_{i=1}^{m} w_i \varphi_i(x)$$

$$\hat{y}_2(x) = 0.5 + (x) \times \sum_{i=1}^{m} w_i \varphi_i(x)$$

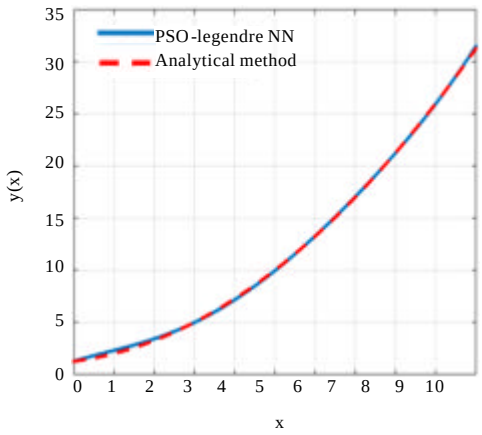

Fig. 13: Solution of example 5-$y_1(x)$

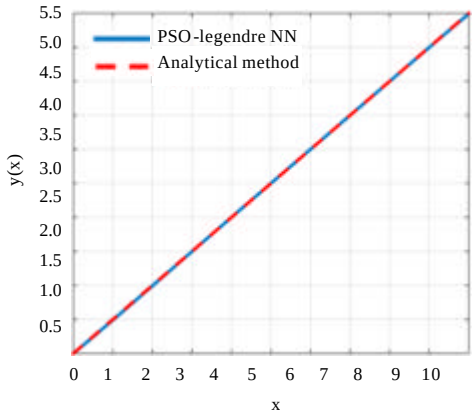

Fig. 14: Solution of example 5-$y_2(x)$

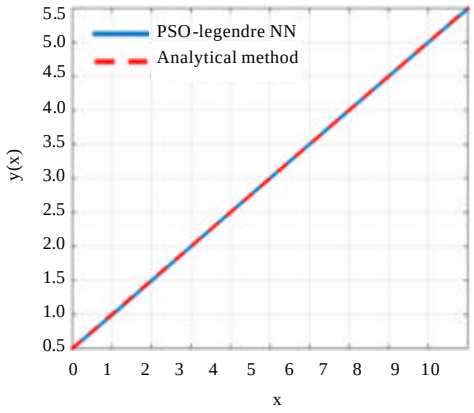

Fig. 15: Cost function for example 5

Analytical solutions and obtained solutions via. PSO-Legendre NN are compared in Fig. 13 and 14. The cost function is shown in Fig 15. It is interesting to note that in this problem, the interval x were discretized into 1000 samples.

CONCLUSION

In this study, a novel hybrid method for solving ordinary differential equations and systems of ordinary differential equations is proposed. This method uses an orthogonal neural network trained by PSO algorithm. By solving 5 well-known examples of linear and non-linear first-order and second-order differential equations the accuracy of this method was examined. Although, there are several methods for solving differential equations, the proposed method does not have the problems as traditional neural networks have such as slow convergence speed.

REFERENCES

Angeline, P.J., G.M. Saunders and J.B. Pollack, 1994. An evolutionary algorithm that constructs recurrent neural networks. IEEE. Trans. Neural Netw., 5: 54-65.

Fasshauer, G.E., 1999. Solving differential equations with radial basis functions: Multilevel methods and smoothing. Adv. Comput. Math., 11: 139-159.

Franke, C., 1998. Solving partial differential equations by collocation using radial basis functions. Appl. Math. Comput., 93: 73-82.

He, S., K. Reif and R. Unbehauen, 2000. Multilayer neural networks for solving a class of partial differential equations. Neural Netw., 13: 385-396.

Jianyu, L., L. Siwei, Q. Yingjian and H. Yaping, 2002. Numerical solution of differential equations by radial basis function neural networks. Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN'02) (Cat. No.02CH37290) Vol. 1, May 12-17, 2002, IEEE, Honolulu, Hawaii, USA., pp: 773-777.

Kennedy, J. and R. Eberhart, 1995. Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks (ICNN'95) Vol. 4, November 27-December 01, 1995, IEEE, Perth, Western, ISBN:0-7803-2768-3, pp: 1942-1948.

Koh, C.S., O.A. Mohammed and S.Y. Hahn, 1994. Detection of magnetic body using artificial neural network with modified simulated annealing. IEEE. Trans. Magn., 30: 3644-3647.

Lagaris, I.E., A. Likas and D.I. Fotiadis, 1998. Artificial neural networks for solving ordinary and partial differential equations. IEEE. Trans. Neural Netw., 9: 987-1000.

Lee, H. and I.S. Kang, 1990. Neural algorithm for solving differential equations. J. Comput. Phys., 91: 110-131.

Meade Jr, A.J. and A.A. Fernandez, 1994b. Solution of nonlinear ordinary differential equations by feedforward neural networks. Math. Comput. Modell., 20: 19-44.

Meade Jr., A.J. and A.A. Fernandez, 1994a. The numerical solution of linear ordinary differential equations by feedforward neural networks. Math. Comput. Modell., 19: 1-25.

Parisi, D.R., M.C. Mariani and M.A. Laborde, 2003. Solving differential equations with unsupervised neural networks. Chem. Eng. Process. Process Intensif., 42: 715-721.

Perng, M.H., 1986. Direct approach for the optimal control of linear time-delay systems via shifted legend re polynomials. Intl. J. Control, 43: 1897-1904.

Puffer, F., R. Tetzlaff and D. Wolf, 1995. A learning algorithm for Cellular Neural Networks (CNN) solving nonlinear partial differential equations. Proceedings of the International Symposium on Signals, Systems and Electronics (ISSE'95), October 25-27, 1995, IEEE, San Francisco, California, USA., pp: 501-504.

Ramuhalli, P., L. Udpa and S.S. Udpa, 2005. Finite-element neural networks for solving differential equations. IEEE. Trans. Neural Netw., 16: 1381-1392.

Razzaghi, M. and M. Shafiee, 1998. Optimal control of singular systems VIA legendre series. Intl. J. Comput. Math., 70: 241-250.

Shaw, D. and W. Kinsner, 1996. Chaotic simulated annealing in multilayer feedforward networks. Proceedings of the 1996 Canadian Conference on Electrical and Computer Engineering Vol. 1, May 26-29, 1996, IEEE, Calgary, Canada, pp: 265-269.

Yao, X., 1993. A review of evolutionary artificial neural networks. Intl. J. Intell. Syst., 8: 539-567.

Yazdi, H.S., M. Pakdaman and H. Modaghegh, 2011. Unsupervised kernel least mean square algorithm for solving ordinary differential equations. Neurocomp., 74: 2062-2071.

Yentis, R. and M.E. Zaghloul, 1996. VLSI implementation of locally connected neural network for solving partial differential equations. IEEE. Trans. Circuits Syst. I. Fundam. Theor. Appl., 43: 687-690.

Zhang, J.R., J. Zhang, T.M. Lok and M.R. Lyu, 2007. A hybrid particle swarm optimization-back-propagation algorithm for feedforward neural network training. Appl. Math. Comput., 185: 1026-1037.