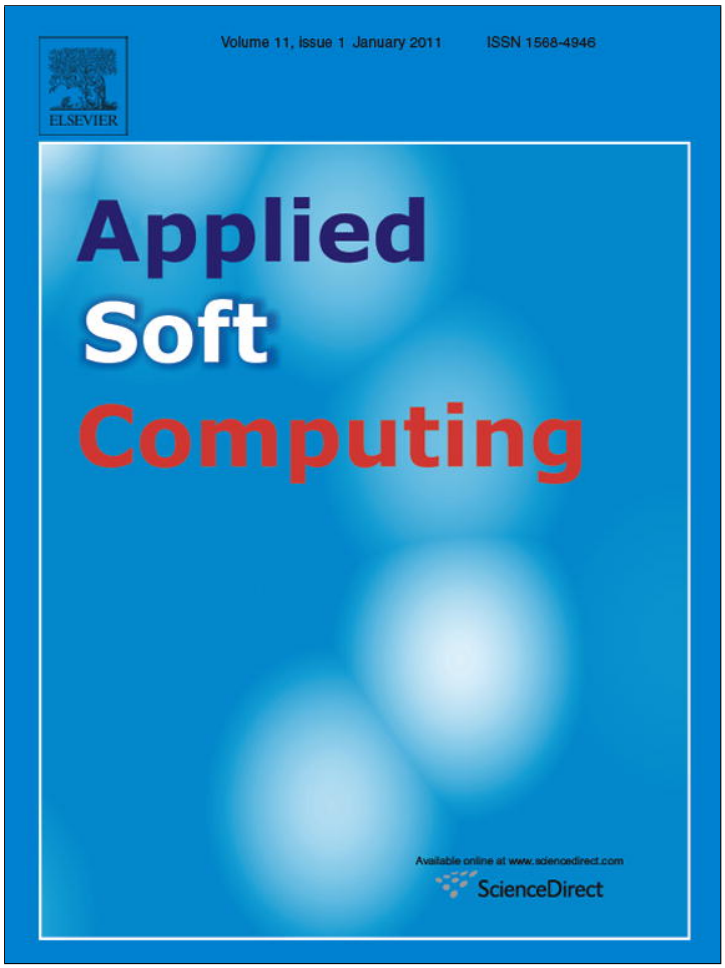

Applied Soft Computing 11 (2011) 861–871

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A differential evolution based neural network approach to nonlinear system identification

Bidyadhar Subudhi[a,*], Debashisha Jena[b,1]

[a] Department of Electrical Engineering, National Institute of Technology, Rourkela-769008, India
[b] Department of Electrical & Electronics Engineering, National Institute of Technology Karnataka, Surathkal-575025, India

ARTICLE INFO

Article history:
Received 27 December 2008
Received in revised form 7 January 2010
Accepted 17 January 2010
Available online 25 January 2010

Keywords:
Back propagation
Differential evolution
Evolutionary computation
Nonlinear system identification
Opposition based differential evolution

ABSTRACT

This paper addresses the effectiveness of soft computing approaches such as evolutionary computation (EC) and neural network (NN) to system identification of nonlinear systems. In this work, two evolutionary computing approaches namely differential evolution (DE) and opposition based differential evolution (ODE) combined with Levenberg Marquardt algorithm have been considered for training the feed-forward neural network applied for nonlinear system identification. Results obtained envisage that the proposed combined opposition based differential evolution neural network (ODE-NN) approach to identification of nonlinear system exhibits better model identification accuracy compared to differential evolution neural network (DE-NN) approach. The above method is finally tested on a one degree of freedom (1DOF) highly nonlinear twin rotor multi-input–multi-output system (TRMS) to verify the identification performance.

1. Introduction

System identification is widely used in a number of applications such as in control system [1], communication [2], signal processing [3], chemical process control [4], biological processes [5], etc. In the strict sense, all the real-world problems are nonlinear in nature. It is pertinent that there is less computational difficulty encountered for the identification of a linear system. However, in the nonlinear system identification, the scenario is not straightforward. There are some classical parameterized models such as Voltera series [6], Winner–Hammerstein model [7] and polynomial identification methods [8,9] which provide a reasonable degree of accuracy but nonetheless these methods involve computational complexities. Subsequently, neural network [10–13], wavelet networks [14] techniques were applied to system identification of nonlinear systems in which adaptive techniques such as back-propagation algorithm found to provide better accuracy compared to non-adaptive ones such as Voltera series, Winner–Hammerstein modeling and polynomial methods. A number of theoretical and practical system identification problems have been solved using neural network approach with multi-layered perceptron (MLP) with back-propagation approach.

The nice property of universal approximation of the neural networks make them suitable for modeling complex system dynamics in a systematic approach especially those which are hard to describe mathematically. It has been proven that any continuous function can be approximated by a feed-forward neural network trained with back-propagation learning algorithm to a reasonable degree of accuracy [10]. This function approximation property can be exploited to model a number of practical complex systems. Narendra and Parthasarathy [13] have shown that multi-layer neural networks trained with standard back-propagation algorithm can be used effectively for the identification of nonlinear dynamic plants.

Recently differential evolution (DE) algorithm has been considered as a novel evolutionary computation technique [15,16] used for optimization problems. The DE has been preferred to many other evolutionary techniques such as genetic algorithm (GA) [17–20] and particle swarm optimization (PSO) due to its attractive characteristics such as its simple concept, easy implementation and quick convergence [15,16]. Generally speaking, all population based optimization algorithms, including the DE, suffer from long computational times because of their evolutionary/stochastic nature. This crucial drawback sometimes limits their application to offline problems with little or no real-time prospective.

The concept of *opposition based learning* (OBL) was introduced by Tizhoosh [21]. It is applied to accelerate reinforcement learning [22] and back-propagation learning in neural networks [23]. The main idea behind OBL is the simultaneous consideration of an estimate and its corresponding opposite estimate (i.e., guess and opposite

* Corresponding author. Tel.: +91 6612462416; fax: +91 6612462999.
E-mail address: bidya2k@yahoo.com (B. Subudhi).
[1] Tel.: +91 824 2474000x3457; fax: +91 824 2474033.

1568-4946/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2010.01.006

guess) in order to achieve a better approximation for the current candidate solution. In this paper, OBL has been utilized to accelerate the convergence rate of DE. Hence, our proposed approach is called opposition based differential evolution (ODE). ODE uses opposite numbers during population initialization and also for generating new populations during the evolutionary process. Here opposite numbers have been utilized to speed up the convergence rate of DE optimization algorithm. Purely random resampling or selection of solutions from a given population has the chance of visiting or even revisiting unproductive regions of the search space. It has been demonstrated [21–28], the chance of this occurring is lower for opposite numbers than it is for purely random ones. In fact, a mathematical proof has already been proposed to show that, opposite numbers are more likely to be closer to the optimal solution than purely random ones [24]. In [29], the usefulness of opposite numbers is investigated by replacing them with random numbers and it is applied for population initialization and generation jumping for different versions of DE.

However, a little work has been reported on applying ODE to system identification and its use in training neural network employed as nonlinear system identifiers. Therefore, it attracts the attention of the present work for exploiting the use of OBL for effective neural network training. In this work, an opposition based differential evolution has been applied as a global optimization method for feed-forward neural network.

Nonlinear system as considered in [30,31] has been chosen in this paper for demonstrating the efficacy of the proposed ODE-NN system identification approach in comparison to DE-NN approach. In this work, an opposition based differential evolution method combined with LM has been applied as a global optimization method for training a feed-forward neural network. In the proposed scheme, the ODE is used to train the neural network that is chosen as a suitable candidate for nonlinear system identification. After observing the trends of training towards minimum through ODE, the network is then trained by LM. The role of the ODE here is to approach towards global minimum point and then LM is used to move forward achieving fast convergence. According to the proposed algorithm after reaching the basin of global minimum the algorithm is switched from global search of the evolutionary algorithm (ODE) to local search, LM. In differential evolution, at the moment of starting, the differential term is very high. As the solution approaches to global minimum the differential term automatically changes to a low value. So during the initial period, the convergence speed is faster and the search space is very large but in latter stages nearer to the optimum differential term is small, the algorithm becomes slower which takes more time to converge. As LM is a gradient based algorithm it can be exploited to increase the convergence speed for reaching the global minimum.

The main contributions of the paper are as follows:

- The paper proposed a new training paradigm of neural networks combining an evolutionary algorithm, i.e. ODE with LM to avoid the possibility of being trapped in local minima.
- LM has been integrated to the search process of the neural network training enabling faster convergence of the ODE-ANN employed for nonlinear system identification.
- The identification performance of the proposed ODE-NN scheme has been compared with the DE-NN approach to nonlinear system identification and found to be better than the later.

The paper is organized as follows. Section 2 reviews the neural network approach to system identification. Section 3 presents the differential evolution technique used for system identification. In Section 4, we presented a new variant of the DE called opposition based DE. Subsequently in Section 5 and Section 6, we describe the algorithm for the proposed opposition based differential evolution combined with neural network approach to nonlinear system identification. Finally the paper discusses the results of all the aforesaid techniques in Section 7 to arrive at conclusions.

2. Identification using neural network

A neural network is a computational structure inspired by knowledge from neuroscience. In the past, most of the system identification problems exploit neural networks either multi-layer perceptron neural network (MLPNN) or radial basis function neural network. Typically, a MLPNN consists of at least two layers of neurons with weighted links connecting the output of neurons in one layer to the input of neurons in the next layer. The weights are updated as follows

$$w_{ji}(k+1) = w_{ji}(k) + \eta\delta_j(k)y_j(k) \tag{1}$$

where $w_{ji}(k)$ is the synaptic weight connecting the output of a neuron i to the input of neuron j at time k. η is the learning rate parameter and $\delta_j(k)$ is the local gradient of neuron j at time k. The learning parameter should be chosen to provide minimization of the total error function, ξ. However, for small η the learning process becomes slow and large value of η corresponds to fast learning but leads to oscillation that prevent the algorithm from converging to the desired solution.

In this section, we discuss a simple example of system identification using neural networks. A neural network approach to system identification involves learning mathematical description of the system, i.e. unknown functions describing the system dynamics. The nonlinear system can be described by the following model given in Eq. (2)

$$y(k) = N(y(k-1)\cdots y(k-n_y), u(k-1)\cdots u(k-n_u)) \tag{2}$$

where $u(k)$ and $y(k)$ are inputs and outputs, respectively. n_y is the maximum lag in output and n_u is the maximum lag in input. $N(.)$ is some nonlinear function between input and output. The input output relationship is dependent on the nonlinear function $N(.)$ which is generally much complex and not available. The system identification problem involves for approximating the unknown function $N(.)$ by using a neural network. There are different neural network identifier such as parallel identification model and series-parallel model [13]. However, for brevity we discuss here the series-parallel model only.

Fig. 1 represents a series-parallel configuration of neural network for nonlinear system identification. In series-parallel model the output of the plant (rather than the identification model) is fed back to the identification model [30]. In this figure, error $e_i(k)$ denotes the difference between plant actual output and estimated output. The delayed samples of the input vector are applied to the plant. Past values of input and output of the plant form the input vector to the neural network; $\hat{y}_p(k)$ corresponds to estimate the plant output given by the neural network at any instant of time k.

3. Differential evolution

In a population of potential solutions to an optimization problem within an n-dimensional search space, a fixed number of vectors are randomly initialized, and then new populations are evolved over time to explore the search space and locate the minima of the objective function. Differential evolutionary strategy (DES) uses a greedy and less stochastic approach in problem solving rather than the other evolutionary algorithms. DE combines simple arithmetical operators with the classical operators of recombination, mutation and selection to evolve from a randomly generated starting population to a final solution. The fundamental idea behind DE is a scheme whereby it generates the trial

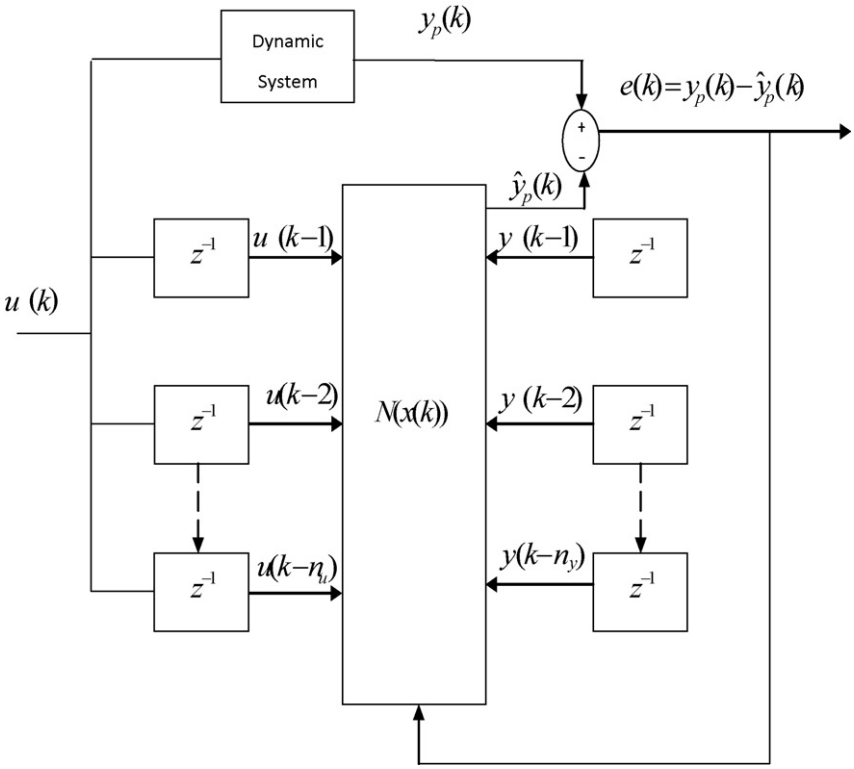

Fig. 1. Neural network (series-parallel model) system identification.

parameter vectors. In each step, the DE mutates vectors by adding weighted, random vector differentials to them. If the fitness of the trial vector is better than that of the target, the target vector is replaced by the trial vector in the next generation. There are many different variants of DE [16], which are as follows: the variants are DE/best/1/exp, DE/rand/1/exp, DE/rand-to-best/1/exp, DE/best/2/exp, DE/rand/2/exp, etc. Now we explain the working steps involved in employing a DE cycle.

- Step 1: parameter setup
  Choose the parameters of population size, the boundary constraints of optimization variables, the mutation factor (F), the crossover rate (C), and the stopping criterion of the maximum number of generations (g).
- Step 2: initialization of the population
  Set generation $g=0$. Initialize a population of $i = 1, \ldots, P$ individuals (real-valued d-dimensional solution vectors) with random values generated according to a uniform probability distribution in the d-dimensional problem space. These initial values are chosen randomly within user's defined bounds.
- Step 3: evaluation of the population
  Evaluate the fitness value of each individual of the population. If the fitness satisfies predefined criteria save the result and stop, otherwise go to step 4.
- Step 4: mutation operation (or differential operation)
  Mutation is an operation that adds a vector differential to a population vector of individuals. For each target vector $x_{i,g}$ a mutant vector is produced using the following relation

$$v_{i,g} = x_{r_1,g} + F(x_{r_2,g} - x_{r_3,g}) \tag{3}$$

  In Eq. (3), F is the mutation factor, which provides the amplification to the difference between two individuals $(x_{r_2,g} - x_{r_3,g})$ so as to avoid search stagnation and it is usually taken in the range of [0,1], where $i, r_1, r_2, r_3 \in \{1, 2, \ldots, P\}$ are randomly chosen numbers but they must be different from each other. P is the number of population.

- Step 5: recombination operation
  Following the mutation operation, recombination is applied to the population. Recombination is employed to generate a trial vector by replacing certain parameters of the target vector with the corresponding parameters of a randomly generated donor (mutant) vector. There are two methods of recombination in DE, namely, binomial recombination and exponential recombination.
  In binomial recombination, a series of binomial experiments are conducted to determine which parent contributes which parameter to the offspring. Each experiment is mediated by a crossover constant, C $(0 \le C \le 1)$. Starting at a randomly selected parameter, the source of each parameter is determined by comparing C to a uniformly distributed random number from the interval [0,1) which indicates the value of C can exceed the value 1. If the random number is greater than C, the offspring gets its parameter from the target individual; otherwise, the parameter comes from the mutant individual. In exponential recombination, a single contiguous block of parameters of random size and location is copied from the mutant individual to a copy of the target individual to produce an offspring. A vector of solutions are selected randomly from the mutant individuals when $rand_j (rand_j \in [0, 1]$, is a random number), is less than C

$$t_{j,i,g} = \begin{cases} v_{j,i,g} \text{ if} (rand_j \le C) \text{ or } j = j_{rand} \\ x_{j,i,g} \text{ otherwise} \end{cases} \tag{4}$$

  $j = 1, 2, \ldots, d$, where d is the number of parameters to be optimized.
- Step 6: selection operation
  Selection is the procedure of producing better offspring. If the trial vector $t_{i,g}$ has an equal or lower value than that of its target vector, $x_{i,g}$ it replaces the target vector in the next generation; otherwise the target retains its place in the population for at least

one more generation

$$x_{i,g+1} = \begin{cases} t_{i,g}, \text{ if} f(t_{i,g}) \le f(x_{i,g}) \\ x_{i,g}, \text{ otherwise} \end{cases} \quad (5)$$

Once new population is installed, the process of mutation, recombination and selection is replaced until the optimum is located, or a specified termination criterion is satisfied, e.g., the number of generations reaches a predefined maximum g_{max}.

At each generation, new vectors are generated by the combination of vectors randomly chosen from the current population (mutation). The upcoming vectors are then mixed with a predetermined target vector. This operation is called recombination and produces the trial vector. Finally, the trial vector is accepted for the next generation if it yields a reduction in the value of the objective function. This last operator is referred to as a selection. Fig. 2 shows the pseudo code for differential evolution algorithm. The most commonly used method for validation is to utilize the sum-squared error and mean-squared error between the actual output $y(n)$ of the system and the predicted output $\hat{y}(n)$. In this work we have taken the cost function as mean-squared error, i.e. $\mathbf{E} = (1/N)\sum_{k=1}^{N}[\boldsymbol{y} - f(\mathbf{x}, \mathbf{w})]^2$, where N is the number of data considered.

4. Opposition based differential evolution

In evolutionary algorithm optimization approaches, generally a uniform random guess is considered for the initial population. In each generation the solution obtained moves towards the optimal solution and the search process terminates when some predefined criteria is satisfied. The time of computation generally depends on the initial guess, i.e. more is the distance between the initial guess to optimal solution more time it will take to terminate and vice versa. Opposition based learning improves the chance of starting with better initial population by checking the opposite solutions. According to the probability theory 50% of the time a guess is further from the solution than its opposite guess starting with the closer of the two guesses has the ability to have faster convergence. Therefore, starting with the closer of the two guesses (as judged by its fitness) has the potential to accelerate convergence. The same approach can be applied not only to initial solutions but also continuously to each solution in the current population. However, before concentrating on OBL, we need to define the concept of opposite numbers [21].

4.1. Definition of opposite number

Let $x \in [a, b]$ be a real number. The opposite number is $\tilde{x}$ which is defined by $\tilde{x} = a + b - x$.

Definition of opposite point:

Let $p = (x_1, x_2, \ldots, x_d)$ be a point in the D-dimensional space where $x_1, x_2, \ldots, x_d \in R$ and $x_i \in [a_i, b_i]$. The opposite point $\tilde{p} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_d)$ where $\tilde{x}_i = a_i + b_i - x_i$.

4.2. Opposition based optimization

Let $p = (x_1, x_2, \ldots, x_d)$ be a point in the d-dimensional, i.e. a candidate solution. Assume $f(.)$ is the fitness function which is used to measure the candidates' fitness. According to the defini-

```
//Initialization of population//
//Generate a initial random population pop

    while //(convergence criterion not yet met)
    //x_{i,g} defines a vector of the current vector population
    //x_{i,g+1} defines a vector of the new vector population in the
next generation
    for (i=0; i<P; i++)
        r1 = rand (P); //select a random index from 1, 2,…,P
        r2 = rand (P); //select a random index from 1, 2,…,P
        r3 = rand (P); //select a random index from 1, 2,…,P
        v_{i,g} = x_{r1,g} + F (x_{r2,g} − x_{r3,g})
        t_{j,i,g} = { v_{j,i,g} if (rand_j ≤ C) or j = j_rand
                    { x_{j,i,g} otherwise
            if f(t_{i,g}) ≤ f(x_{i,g})
            x_{i,g+1} = t_{i,g}
            else
            x_{i,g+1} = x_{i,g}
            end
    end
```

Fig. 2. Pseudo code of the differential evolution (DE).

```
/*Initialization of opposition based population*/
//Generate a initial random population ipop
     for (i=0; i<P; i++)
          for (j=0; j<d; i++)
          iopop_{i,j} =a_j+b_j - pop_{i,j}
          end
     end
//Select P fittest individual from the set (ipop,iopop) as
initial population
     while //(convergence criterion not yet met)
     //x_{i,g} defines a vector of the current vector population
     //x_{i,g+1} defines a vector of the new vector population in the
next generation
          for (i=0; i<P; i++)
          r_1 = rand (P); //select a random index from 1, 2,…, P
          r_2 = rand (P); //select a random index from 1, 2,…, P
          r_3 = rand (P); //select a random index from 1, 2,…, P
          v_{i,g} = x_{r_1,g} + F (x_{r_2,g} - x_{r_3,g} )
                    { v_{j,i,g}  if (rand_j ≤ C ) or j = j_rand
          t_{j,i,g} = {
                    { x_{j,i,g}  otherwise
                    if f(t_{i,g}) ≤ f(x_{i,g})
                    x_{i,g+1}=t_{i,g}
                    else
                    x_{i,g+1}=x_{i,g}
                    end
               end
//Store in new population npop
/*Initialization of opposition based generation jumping starts*/
               If (rand < J_r)
                    for (i=0; i<P; i++)
                         for (j=0; j<d; i++)
                    onpop_{i,j} = min(npop_j)+ max(npop_j) - mpop_{i,j}
                         end
                    end
               end
//Select P fittest individual from the set (onpop_{i,j}, npop_{i,j})

          end   % while
```

Fig. 3. Pseudo code of the ODE.

tion of the opposite point $\tilde{p} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_d)$ is the opposite of $p = (x_1, x_2, \ldots, x_d)$. Now if $f(\tilde{p}) \geq f(p)$ then point p can be replaced by $\tilde{p}$ otherwise we will continue with p. Hence the point and its opposite point are evaluated simultaneously in order to continue with the more fit ones.

5. Proposed ODE-NN algorithm

Similar to all population based optimization algorithms, two main steps are distinguishable for DE, namely, population initialization and producing new generations by evolutionary operations such as mutation, crossover, and selection. We will enhance these two steps using the OBL scheme. The original DE is chosen as a parent algorithm and the proposed opposition based ideas are embedded in DE to accelerate its convergence speed. The pseudo code for the proposed approach (ODE) is presented in Fig. 3.

5.1. Opposition based population initialization

According to our review of optimization literature, random number generation, in absence of *a priori* knowledge, is the common choice to create an initial population. Therefore, by utilizing OBL, we can obtain fitter starting candidate solutions even when there is not *a priori* knowledge about the solution(s). The following steps present opposition based initialization for ODE that procedure. Initialize the population pop (P) randomly Calculate opposite population

$$opop_{i,j} = a_j + b_j - pop_{i,j}, \quad i = 1, 2, \ldots, P, \ \ j = 1, 2, \ldots, d$$

where $pop_{i,j}$ and $opop_{i,j}$ denote the jth variable of the ith vector of the population and opposite population, respectively. Select P fittest individual from ($pop\ U\ opop$) as initial population.

5.2. Opposition based generation jumping

By applying a similar approach to the current population, the evolutionary process can be forced to jump to a new solution candidate, which ideally is fitter than the current one. Based on a jumping rate (i.e., jumping probability), after generating new populations by mutation, crossover, and selection, the opposite population is calculated and the fittest individuals are selected from the union of the current population and the opposite population. Unlike opposition based initialization, generation jumping calculates the opposite population dynamically. In each generation the search space is reduced so that we have to calculate the opposite points by using variables current interval in the population

$$onpop_{i,j} = \min(npop_j) + \max(npop_j) - npop_{i,j}$$

By staying within variables' interval static boundaries, we would jump outside of the already shrunken search space and the knowledge of the current reduced space (converged population) would be lost. Hence, we calculate opposite points by using variables' current interval in the population which is, as the search

does progress, increasingly smaller than the corresponding initial range.

6. A combined ODE-NN approach to system identification

In the sequel, we describe how an ODE is applied for training neural network in the frame work of system identification (see Algorithm 1). According to step 10 in the proposed algorithm, the value of the cost function after reaching a particular value of ε, the algorithm is switched from global search such of the evolutionary algorithm (ODE) to local search, LM. In opposition based differential evolution, at the moment of starting, the differential term is very high. As the solution approaches to global minimum the differential term automatically changes to a low value. So at initial period the convergence speed is faster and search space is very large but in latter stages nearer to the optimum due to small differential term the algorithm becomes slower which will take more time to converge. As LM is a gradient based algorithm at that point the role of LM is to increase the convergence speed for reaching the global minimum. ODE can be applied to global searches within the weight space of a typical feed-forward neural network. Output of a feed-forward neural network is a function of synaptic weights **w** and input values **x**, i.e. $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{w})$. The role of LM in the proposed algorithm has been described in section I. In the training processes, both the input vector **x** and the output vector **y** are known and the synaptic weights in **w** are adapted to obtain appropriate functional mappings from the input **x** to the output **y**. Generally, the adaptation can be carried out by minimizing the network error function **E** which is of the form $\mathbf{E}(\mathbf{y}, \mathbf{f}(\mathbf{x}, \mathbf{w}))$. In this work we have taken **E** as mean-squared error i.e. $\mathbf{E} = (1/N)\sum_{k=1}^{N}[\mathbf{y} - f(\mathbf{x}, \mathbf{w})]^2$, where N is the number of data considered. The optimization goal is to minimize the objective function **E** by optimizing the values of the network weights **w**. Where $\mathbf{w} = (w_1, \ldots, w_d)$

Algorithm 1 *(ODE-NN Identification Algorithm).*

Step 1

Initialize population ***pop***: Create a population from randomly chosen object vectors.

Step 2

Find out the opposite population ***opop***: Create an opposite population from the population ***pop***.

Step 3

Create a fittest population ***npop*** from both ***pop*** U ***opop*** with dimension P

$$\mathbf{P}_g = (\mathbf{w}_{1,g}, \ldots, \mathbf{w}_{P,g})^T, \quad g = 1, \ldots, g_{\max}$$

$$\mathbf{w}_{i,g} = (w_{1,i,g}, \ldots, w_{D,i,g}), \quad i = 1, \ldots, P$$

where D is the number of weights in the weight vector and in $\mathbf{w}_{i,g}$, i is index to the population and g is the generation to which the population belongs.

Step 4

Evaluate all the candidate solution inside ***npop*** for a specified number of generations.

Step 5

For each ith candidate in ***npop*** select the random variables $r_1, r_2, r_3 \in \{1, 2, \ldots, P\}$.

Step 6

Apply mutation operator to each candidate in population to yield a mutant vector, i.e.

$$v_{j,i,g} = w_{j,r_1,g} + F(w_{j,r_2,g} - w_{j,r_3,g}), \quad \text{for } j = 1, \ldots, d$$
$$(i \neq r_1 \neq r_2 \neq r_3) \in \{1, \ldots, P\} \quad \text{and} \quad F \in (0, 1+]$$

Step 7

Apply crossover, i.e. each vector in the current population is recombined with a mutant vector to produce trial vector

$$t_{j,i,g} = \begin{cases} v_{j,i,g} \text{ if } rand_j[0, 1) \leq C \\ w_{j,i,j} \text{ otherwise} \end{cases}$$
where $C \in [0, 1]$

Step 8

Apply selection, i.e. between the trial vector and target vector. If the target vector has an equal or lower objective function value than that of its target vector, it replaces target vector; otherwise, the target retains its place in the population.

$$\boldsymbol{w}_{i,g} = \begin{cases} t_{i,g} \textit{ if } \boldsymbol{E}(\boldsymbol{y}, f(\boldsymbol{x}, t_{i,g})) \leq \boldsymbol{E}(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{w}_{i,g})) \\ \boldsymbol{w}_{i,g} \text{ otherwise} \end{cases}$$

Step 9

If $rand_j < J_r$ Find the opposite population of $\mathbf{w}_{i,g}$, i.e. $\mathbf{ow}_{i,g}$

Select P fittest individuals from $\boldsymbol{w}_{i,g} \cup \boldsymbol{ow}_{i,g}$ which gives the populations for the next generation which is represented by $\boldsymbol{w}_{i,g+1}$. Else $\boldsymbol{w}_{i,g+1} = \boldsymbol{w}_{i,g}$

Step 10

Evaluate **E** for the weights obtained from step-9 ***If*** $\boldsymbol{E} \leq \varepsilon$ where $\varepsilon > 0$ go to step-8 ***Else*** go to step 5.

Step 11

Initialize the weight matrix of Levenberg Marquardt algorithm taking the values of weights obtained after the fixed number of iterations. Find out the value of **E**.

Step 12

Compute the Jacobian matrix $\boldsymbol{J}(w)$.

Step 13

Find Δw using the following equation

$$\Delta w = [\mathbf{J}^T(w)J(w) + \mu I]^{-1}\mathbf{J}^T(w)\mathbf{E}$$

Step 14

Recompute **E** using $(w + \Delta w)$. if this new **E** is smaller than that computed in step 7 then reduce μ and go to step 1 where μ is the damping factor.

Step 15

The algorithm is assumed to have converged when the norm of the gradient, i.e. $||\nabla E|| = ||\mathbf{J}^T(w)\mathbf{y} - f(\mathbf{x}, w)||$ is less than some predetermined value, or when the sum of squares of errors has been reduced to some error goal.

7. Results and discussion

We present here the system identification results of different approaches such as DE-NN and ODE-NN applied to the systems given in equation (6) and Box and G.M. Jenkins, Time Series Analysis, and a real-time TRMS system.

Example 1 *(A Bench Mark Problem).* The nonlinear system to be identified is expressed by

$$y_p(k+1) = \frac{y_p(k)[y_p(k-1)+2][y_p(k)+2.5]}{8.5+[y_p(k)]^2+[y_p(k-1)]^2} + u(k) \quad (6)$$

where $y_p(k)$ is the output of the system at the kth time step and $u(k)$ is the plant input which is uniformly bounded function of time. The plant is stable at $u(k) \in [-2, 2]$, i.e. the input signal $u(k)$ is bounded in the region $[-2, 2]$. The identification model be in the form of

$$y_{pi}(k+1) = f(y_p(k), y_p(k-1)) + u(k) \quad (7)$$

where $f(y_p(k), y_p(k-1))$ is the nonlinear function of $y_p(k)$ and $y_p(k-1)$ which will be the inputs for DE-NN and ODE-NN neural system identifier. The output from neural network will be $y_{pi}(k+1)$. The goal is to train the above three networks such that when an input $u(k)$ is presented to the network and to the nonlinear system, the network outputs $y_{pi}(k)$ and the actual system output $y_p(k)$ are very close.

The neural network identifier structure consisted of eleven numbers of neurons in the hidden layer. After 500 epochs the training of the neural identifier has been stopped. After the training is over, its prediction capability has been tested for input

$$u(k) = \begin{cases} 2\cos(2\pi k/100) \text{ if } k \le 200 \\ 1.2\ \sin(2\pi k/20) \text{ if } 200\ <\ k \le 500 \end{cases} \quad (8)$$

Table 1
Parameters for DE and ODE.

Total number of iterations	1000
Population size, N	50
Upper and lower bound of weights	[0 1]
Mutation constant factor, F	0.6
Cross over constant, C	0.5
Random number J_r	0.3

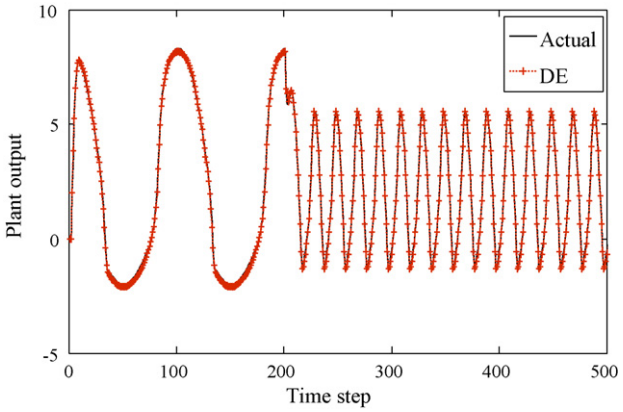

Fig. 4. DE-NN identification performance.

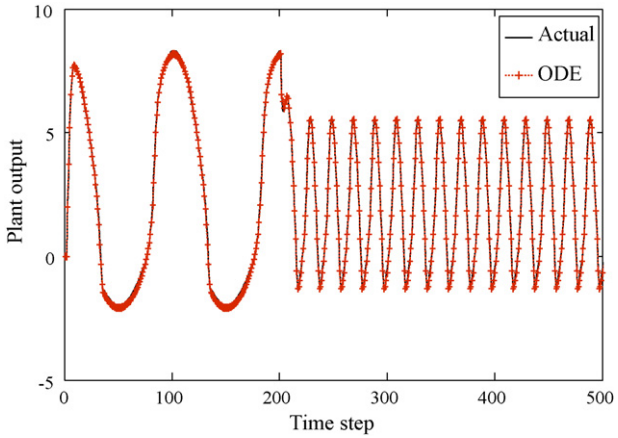

Fig. 5. ODE-NN identification performance.

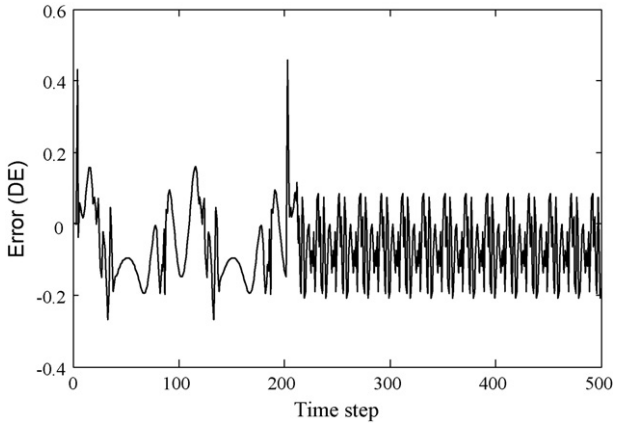

Fig. 6. DE-NN identification error.

Table 1 gives the parameters of DE and ODE. Figs. 4 and 5 give the identification performance between actual and identified model for DE-NN and ODE-NN, respectively. Figs. 6 and 7 give the identification error and Fig. 8 gives the comparison of mean-squared error for both the system identification scheme. From the figures it is clear that both the results are nearly same. The value of mean-squared error is given in Table 1. From this it is clear that the prediction error is slightly less in case of ODE-NN approach in comparison to the proposed DE-NN system identification technique.

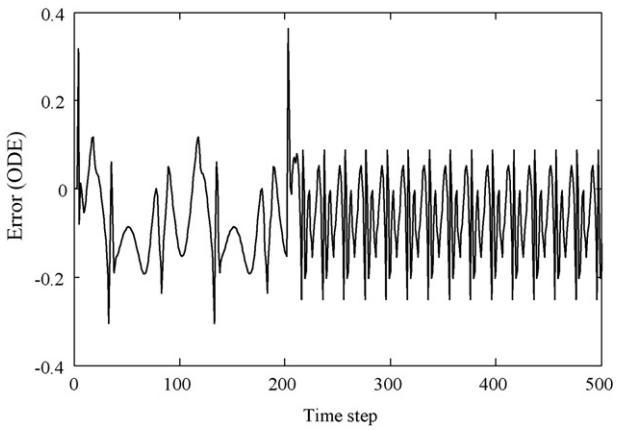

Fig. 7. ODE-NN identification error.

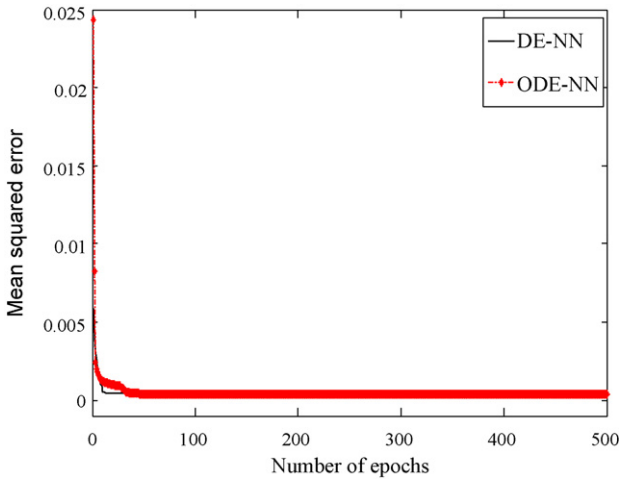

Fig. 8. Mean-squared error.

Example 2 *(Box and Jenkins' Gas Furnace Problem).* Box and Jenkins' gas furnace data are frequently used in performance evaluation of system identification methods. This is a time series data set for a gas furnace. The data consists of 296 input–output samples recorded with a sampling period of 9 s. The gas combustion process has one variable, gas flow $u(t)$, and one output variable, the concentration of carbon dioxide (CO_2), $y(t)$. The instantaneous values of output $y(t)$ have been regarded as being influenced by ten variables $y(t-1)$, $y(t-2)$, $y(t-3)$, $y(t-4)$, $y(t-5)$ $u(t-1)$, $u(t-2)$, $u(t-3)$, $u(t-4)$, $u(t-6)$. The original data set contains 296 $[u(k),y(k)]$ data pairs. By converting the data so that each training data consists of $[y(t-1)\ldots y(t-4),u(t-1)\ldots u(t-6)]$ reduces the number of effective data points to 290. The number of training data was taken as 100 for all the cases and rest 190 data were the test data. Here we have taken two inputs for simplicity one is from furnace output and other is from furnace input so we have build 24 models of different input and output. Table 1 gives the training and testing performances of these 24 models. For all the methods eleven number of hidden layer neurons were taken and the results obtained after 100 epochs. From Table 2 we can conclude that model with $y(t-1)$ and $u(t-3)$ as input has the smallest training and testing error for both the DE-NN and ODE-NN identification schemes. It is also clear that ODE is having less training and testing errors in comparison to its DE counterpart. The RMSE for testing turned out to be the least for 20 cases in ODE-NN approach whereas it was found to be better only for four cases for DE-NN approach. Similarly it was found that the training RMSE is least for sixteen cases for ODE-NN and for rest eight cases DE-NN was found to be better. In some cases even if the training error is less for DE-NN but the testing error is better for ODE-NN.

As it is not possible to show the identification performance and error curve for all the 24 cases given in Table 2, we have taken three cases to analyze the mean-squared error and their performances. From Fig. 9 it is clear that MSE of the proposed ODE-NN approach is converging faster than DE-NN approach at the same time the MSE of ODE-NN starting from a lower value, i.e. around 2.2 where as for DE-NN it is starting from 2.9. Fig. 10 gives the identification performance for DE-NN, ODE-NN and the actual output. As the performance in Fig. 10 is not clear Fig. 11 gives a zoomed version within time step 111–116 where it clearly shows that the ODE-NN is having better identification capability than DE-NN approach.

Fig. 12 gives the mean-squared error for the input $y(t-4)$ and $u(t-5)$. Here the we have considered 20 epochs because there was no change in MSE after 20 epochs. In this case even if the ODE-NN MSE starts from a higher value but it is converging to a lower value

Table 2
Comparison of training and testing errors.

Input	Testing error (RMSE)		Training error (RMSE)	
	DE	ODE	DE	ODE
Example 1				
$y(k), y(k-1)\ u(k)$	0.0207	**0.0190**	0.1186	**0.1137**
Example 2				
$y(t-1), u(t-3)$	0.4400	**0.4194**	0.1501	**0.1411**
$y(t-3), u(t-4)$	0.7838	**0.7773**	0.3402	**0.2850**
$y(t-2), u(t-4)$	0.6733	**0.6602**	0.3256	**0.2898**
$y(t-1), u(t-2)$	**0.4906**	0.6801	**0.2909**	0.2924
$y(t-1), u(t-4)$	0.5430	**0.5132**	0.2991	**0.2926**
$y(t-4), u(t-4)$	12.259	**0.8894**	**0.3274**	0.3428
$y(t-2), u(t-3)$	1.1340	**0.7199**	**0.2968**	0.3051
$y(t-1), u(t-1)$	0.6183	**0.6056**	0.4638	**0.4151**
$y(t-4), u(t-3)$	**1.2405**	1.2771	0.7266	**0.4301**
$y(t-1), u(t-6)$	0.8469	**0.8410**	0.6012	**0.5661**
$y(t-3), u(t-3)$	**1.0067**	1.0347	**0.5172**	0.5176
$y(t-2), u(t-2)$	0.9889	**0.9753**	0.6314	**0.6261**
$y(t-1), u(t-5)$	0.6873	**0.6518**	**0.6220**	0.6303
$y(t-4), u(t-5)$	1.0149	**0.9698**	0.7038	**0.6373**
$y(t-2), u(t-1)$	1.8368	**1.2726**	0.8934	**0.6844**
$y(t-2), u(t-5)$	**0.9176**	1.1808	0.7222	**0.6804**
$y(t-3), u(t-5)$	**0.9536**	1.0470	**0.7138**	0.7338
$y(t-3), u(t-2)$	1.8184	**1.4138**	0.8766	**0.8600**
$y(t-4), u(t-6)$	1.7628	**1.4677**	1.3988	**1.1126**
$y(t-2), u(t-6)$	1.3352	**1.2639**	1.6264	**1.1945**
$y(t-4), u(t-2)$	1.6725	**1.6377**	**1.1799**	1.1963
$y(t-3), u(t-6)$	27.468	**1.4641**	**1.2063**	1.2424
$y(t-3), u(t-1)$	1.7123	**1.6475**	1.5725	**1.2702**
$y(t-4), u(t-1)$	2.0821	**2.0217**	**1.4250**	1.4352

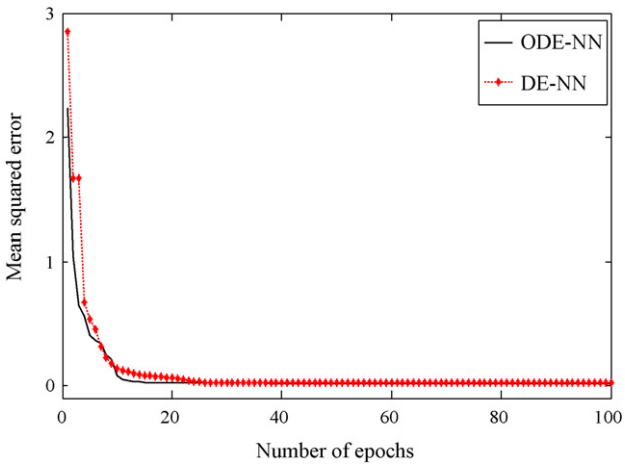

Fig. 9. Mean-squared error ($y(t-1)$, $u(t-3)$).

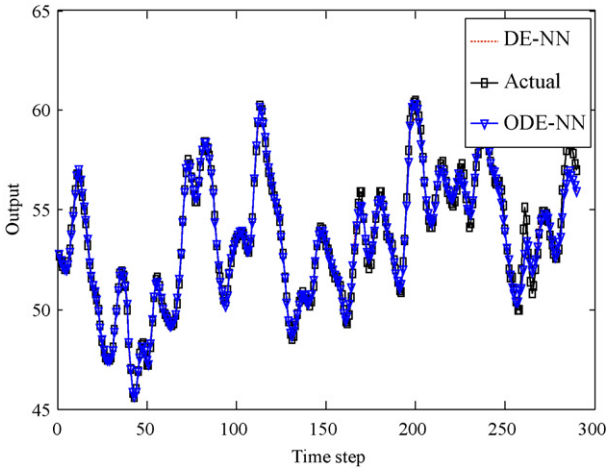

Fig. 10. Identification performance ($y((t-1), u(t-3))$.

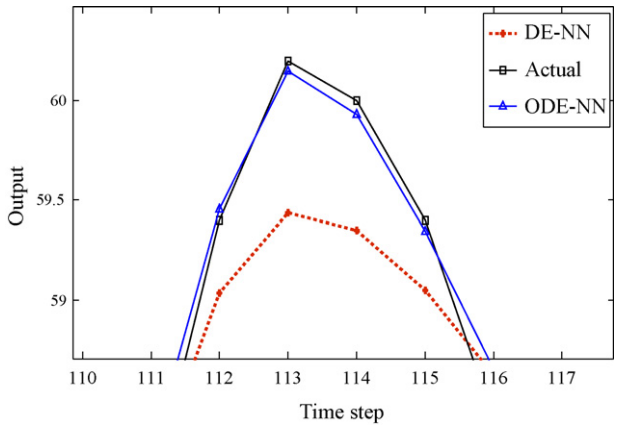

Fig. 11. Identification performance ($y(t-1)$, $u(t-3)$).

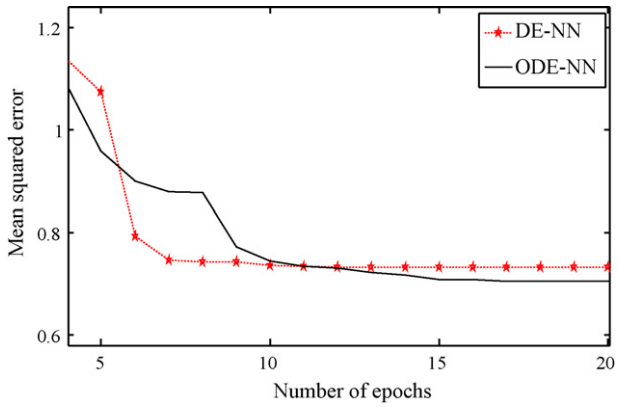

Fig. 12. Mean-squared error ($y(t-4)$, $u(t-5)$).

in comparison to DE-NN approach. Fig. 13 gives the identification performance for the input $y(t-4)$ and $u(t-5)$. The zoomed version of the identification performance within time step 54–58 is shown in Fig. 14.

Fig. 15 shows the MSE for the input $y(t-4)$ and $u(t-4)$. From the figure it is clear that the MSE for ODE-NN is having higher convergence speed but attending a slight lower value in comparison to DE-NN approach, the numerical values are mentioned in Table 2. Fig. 16 shows the identification performance for the input $y(t-4)$ and $u(t-4)$ from which it is found even if the training error for ODE-NN approach is slightly higfer than the DE-NN approach but having much better identification capability in comparison to DE-NN approach.

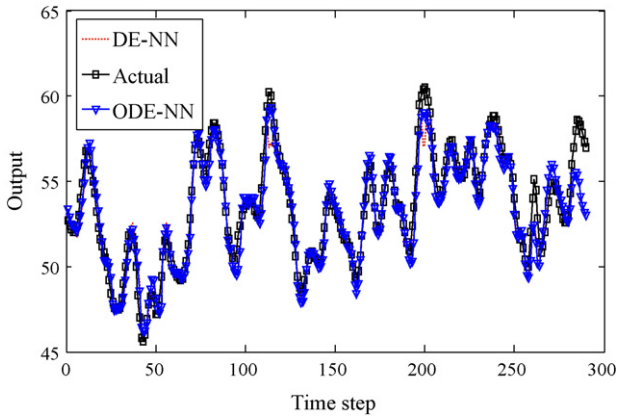

Fig. 13. Identification performance ($y(t-4)$, $u(t-5)$).

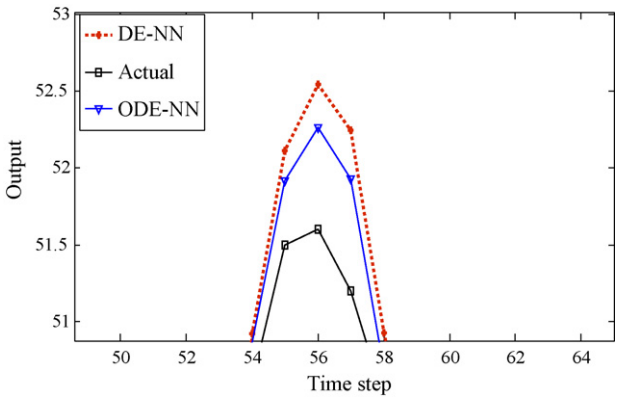

Fig. 14. Identification performance ($y(t-4)$, $u(t-5)$).

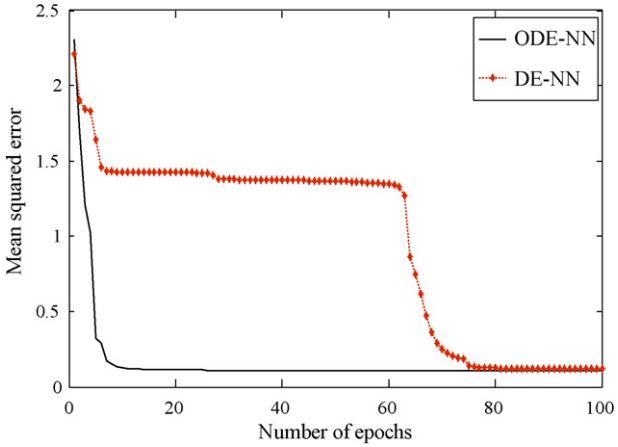

Fig. 15. Mean-squared error ($y(t-4)$, $u(t-4)$).

Table 2 gives the comparison of training and testing root mean square error (RMSE) for DE-NN and ODE-NN approaches. For the first example it is found that the training and testing errors are less for the proposed ODE-NN approach. From the table the results marked in bold indicates less training and testing error for the corresponding input combinations. In Example 2 we have considered all the possible input combinations, i.e. 24. It is found that the testing error is less for 19 combinations of ODE-NN approach in comparison to DE-NN approach. This shows the better identification capability of the proposed hybrid approach.

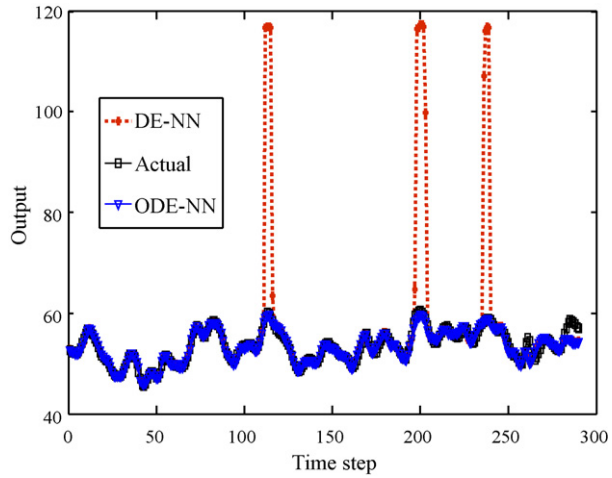

Fig. 16. Identification performance ($y(t-4)$, $u(t-4)$).

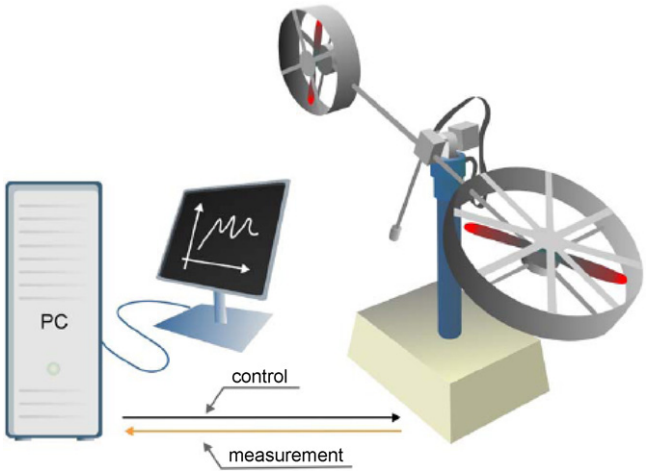

Fig. 17. The laboratory set-up: TRMS system.

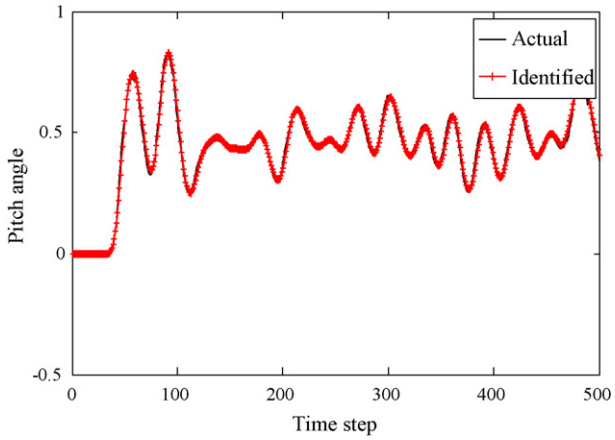

Fig. 18. Identification performance.

Example 3 *(Twin Rotor MIMO System).* The TRMS used in this work is supplied by Feedback Instruments designed for control experiments. Fig. 17 shows the TRMS setup which serves as a model of the helicopter.

It consists of two rotors placed on a beam with a counterbalance. These two rotors are driven by two D.C. motors. The main rotor produces a lifting force allowing the beam to rise vertically making the rotation around the pitch axis. The tail rotor which is smaller than the main rotor is used to make the beam turn left or right around the yaw axis. Both the axis of either or both axis of rotation can be locked by means of two locking screws provided for physically restricting the horizontal and or vertical plane of the TRMS rotation. Thus, the system permits both 1 and 2 DOF experiments. In this work we have taken only the 1 DOF around the pitch axis and identified the system using proposed method discussed in Section 6. The model has three inputs and eleven neurons in the hidden layer. The inputs are the main rotor voltage at the present time $v(t)$, main rotor voltage at previous time $v(t-1)$ and the pitch angle of the beam at previous time instants $(t-1)$. Fig. 18 shows the identi-

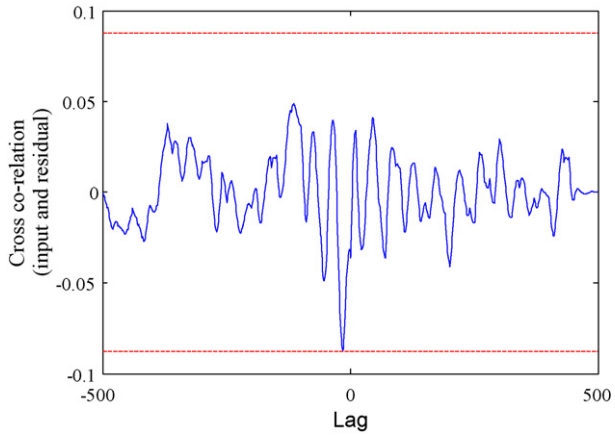

Fig. 19. Cross-correlation of input and residuals.

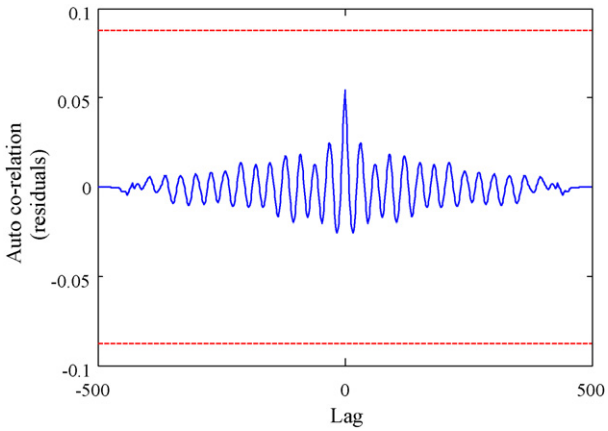

Fig. 20. Auto-correlation of residuals.

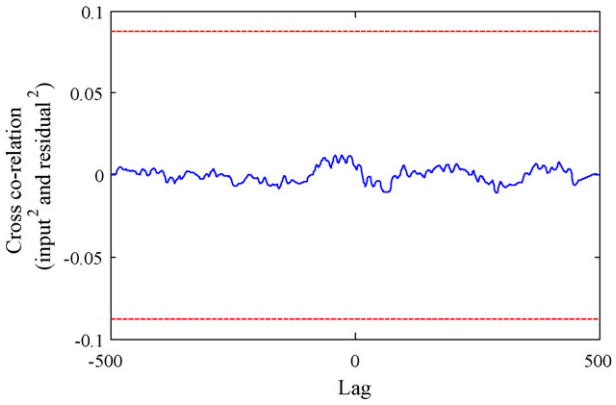

Fig. 21. Cross-correlation of input square and residuals square.

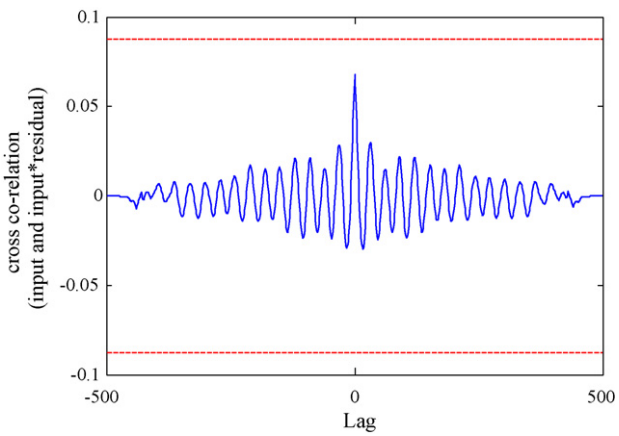

Fig. 22. Cross-correlation of residuals and input*residuals.

fication performance of 1 degree of freedom (DOF) vertical ODE-NN based model. The correlation analysis of the above model is given in Figs. 19–22. If the residuals (model errors) contain no information about past residuals or about the dynamics of the system, it is likely that all information has been extracted from the training set and the model approximates the system well. It is found that all four correlation functions; cross-correlation of input and residuals (Fig. 19), auto-correlation of residuals (Fig. 20), cross-correlation of input square and residuals square (Fig. 21), cross-correlation of residuals and input*residuals (Fig. 22) are within 95% of the confidence band indicating that the model is adequate, i.e. the model behavior is closed to the real system performance.

8. Conclusions

The paper has described the scope of improving system identification of nonlinear systems by using proposed ODE-NN approach. In the proposed identification frame work, ODE is used only to find approximate values in the vicinity of the global minimum. These approximate weight values are then used as starting values for a faster convergence algorithm, i.e. Levenberg Marquardt algorithm. From the results presented in Section 7, it is clear that there is certainly an improvement in identification performance for nonlinear systems over the existing DE-NN approach. In comparison to use of DE-NN approach proposed ODE-NN approach provides better system identification performance in terms of speed of convergence and identification capability. In comparison to DE method ODE seems to provide advantage in terms of faster convergence speed. The above-proposed ODE-NN approach is tested on a real-time TRMS system. It is shown that the models obtained ODE-NN methods can generally be considered adequate in representing the system.

References

[1] S. Chen, S.A. Billings, Representation of non-linear systems: the NARMAX model, Int. J. Control 49 (1989) 1013–1032.
[2] H. Hujiberts, H. Nijmeijer, R. Willems, System identification in communication with chaotic systems, IEEE Trans. Circuits Syst. I 47 (6) (2000) 800–808.
[3] M. Adjrad, A. Belouchrani, Estimation of multi component polynomial phase signals impinging on a multisensor array using state-space modeling, IEEE Trans. Signal Process. 55 (1) (2007) 32–45.
[4] K. Watanbe, I. Matsuura, M. Abe, M. Kebota, D.M. Himelblau, Incipient fault diagnosis of chemical processes via artificial neural networks, AICHE J. 35 (11) (1989) 1803–1812.
[5] Y. Xie, B. Guo, L. Xu, J. Li, P. Stoica, Multistatic adaptive microwave imaging for early breast cancer detection, IEEE Trans. Biomed. Eng. 53 (8) (2006) 1647–1657.
[6] M. Schetzmen, The Voltera and Winner Theories on Nonlinear Systems, Wiley, New York, 1980.
[7] F. Dinga, T. Chenb, Identification of Hammerstein nonlinear ARMAX systems, Automatica 41 (2005) 1479–1489.
[8] E. Hernandez, Y. Arkun, Control of nonlinear systems using polynomial ARMA models, AICHE J. 39 (3) (1993) 446–460.
[9] T.T. Lee, J.T. Jeng, The Chebyshev polynomial-based unified model neural networks for functional approximation, IEEE Trans. Syst. Man Cybern. B 28 (1998) 925–935.
[10] N. Sadegh, A perceptron based neural network for identification and control of nonlinear systems, IEEE Trans. Neural Networks 4 (1993) 982–988.
[11] K.H. Chon, R.J. Cohen, Linear and nonlinear ARMA model parameter estimation using an artificial neural network, IEEE Trans. Biomed. Eng. 44 (1997) 168–174.
[12] Xianfeng Ni Stef, J.R. Simons, Nonlinear dynamic system identification using radial basis function networks, in: Proceedings of IEEE on Decision and Control, Kobe, Japan, 1996, pp. 935–936.
[13] K.S. Narendra, K. Parthaasarathy, Identification and control of dynamical systems using neural networks, IEEE Trans. Neural Networks 1 (1990) 4–27.
[14] Q. Zhang, Using wavelet network in nonparametric estimation, IEEE Trans. Neural Networks 8 (1997) 227–236.
[15] R. Storn, System design by constraint adaptation and differential evolution, IEEE Trans. Evol. Comput. 3 (1999) 22–34.
[16] J. Ilonen, J.K. Kamarainen, J. Lampinen, Differential evolution training algorithm for feed forward neural networks, Neural Proc. Lett. 17 (2003) 93–105.
[17] E. Goldberg, J. Richardson, Genetic algorithms with sharing for multimodal function optimization, in: J. Richardson (Ed.), Genetic Algorithms and their Applications (ICGA'87)., 1987, pp. 41–49.
[18] L.D. Montana, Training feedforward neural networks using genetic algorithms, in: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, San Mateo, CA, Morgan Kaufmann, 1989, pp. 762–767.
[19] E. Goldberg, Genetic algorithms and walsh functions: Part 2, deception and its analysis, Complex Syst. 3 (1989) 153–171.
[20] K. Kristinsson, G.A. Dumont, System identification and control using genetic algorithms, IEEE Trans. Syst. Man Cybernet. 22 (1992) 1033–1046.
[21] H.R. Tizhoosh, Opposition-based learning: a new scheme for machine intelligence, in: Proc. Int. Conf. Comput. Intell. Modeling Control and Autom, vol. I, Vienna, Austria, 2005, pp. 695–701.
[22] M. Shokri, H.R. Tizhoosh, M. Kamel, Opposition-based Q(λ) algorithm, in: Proc. IEEE World Congr. Comput. Intell., Vancouver, BC, Canada, 2006, pp. 646–653.
[23] M. Ventresca, H.R. Tizhoosh, Improving the convergence of back-propagation by opposite transfer functions, in: Proc. EEE World Congr. Comput. Intell., Vancouver, BC, Canada, 2006, pp. 9527–9534.
[24] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition Versus Randomness in Soft Computing Techniques, Elsevier J. Appl. Soft Comput. 8 (March (2)) (2008) 906–918.
[25] H. Tizhoosh, M. Ventresca, Oppositional Concepts in Computational Intelligence. Studies in Computational Intelligence, Springer-Verlag, 2008.
[26] M. Ventresca, H.R. Tizhoosh, Numerical condition of feed-forward networks with opposite transfer functions, in: International Joint Conference on Neural Networks (IJCNN), Hong Kong, China, 2008, pp. 3232–3239.
[27] M. Ventresca, H. Tizhoosh, Simulated annealing with opposite neighbors, in: IEEE Symposium on Foundations of Computational Intelligence, Honolulu, USA, 2007, pp. 186–192.
[28] M. Ventresca, H. Tizhoosh, Opposite transfer functions and back-propagation through time, in: IEEE Symposium on Foundations of Computational Intelligence, Honolulu, USA, 2007, pp. 570–577.
[29] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution, IEEE Trans. Evol. Comput. 12 (1) (2008).
[30] C.-T. Lin, C.S. George Lee, Neural Fuzzy Systems: A Neuro-fuzzy Synergism to Intelligent Systems, Prentice Hall International, Inc., New Jersey, 1996.
[31] G.E.P. Box, G.M. Jenkins, Time Series Analysis. Forecasting and Control, Holden Day, San Francisco, 1970.