Neurocomputing 74 (2011) 1696–1709

Contents lists available at ScienceDirect

Neurocomputing

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

Nonlinear system identification using memetic differential evolution trained neural networks

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

[a] Center for Industrial Electronics & Robotics, 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 18 June 2010
Received in revised form
23 November 2010
Accepted 7 February 2011
Communicated by A. Konar
Available online 17 March 2011

Keywords:
Back propagation
Differential evolution
Evolutionary computation
Nonlinear system identification
Particle swarm optimization

ABSTRACT

Several gradient-based approaches such as back propagation (BP) and Levenberg Marquardt (LM) methods have been developed for training the neural network (NN) based systems. But, for multimodal cost functions these procedures may lead to local minima, therefore, the evolutionary algorithms (EAs) based procedures are considered as promising alternatives. In this paper we focus on a memetic algorithm based approach for training the multilayer perceptron NN applied to nonlinear system identification. The proposed memetic algorithm is an alternative to gradient search methods, such as *back-propagation* and *back-propagation* with momentum which has inherent limitations of many local optima. Here we have proposed the identification of a nonlinear system using memetic differential evolution (DE) algorithm and compared the results with other six algorithms such as Back-propagation (BP), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Differential Evolution (DE), Genetic Algorithm Back-propagation (GABP), Particle Swarm Optimization combined with Back-propagation (PSOBP). In the proposed system identification scheme, we have exploited DE to be hybridized with the back propagation algorithm, i.e. differential evolution back-propagation (DEBP) where the local search BP algorithm is used as an operator to DE. These algorithms have been tested on a standard benchmark problem for nonlinear system identification to prove their efficacy. First examples shows the comparison of different algorithms which proves that the proposed DEBP is having better identification capability in comparison to other. In example 2 good behavior of the identification method is tested on an *one degree of freedom (1DOF) experimental aerodynamic test rig, a twin rotor multi-input–multi-output system (TRMS), finally it is applied to* Box and Jenkins Gas furnace benchmark identification problem and its efficacy has been tested through correlation analysis.

1. Introduction

SYSTEM identification using neural networks has been considered as a promising approach due to its function approximation properties [1] and for modeling nonlinear system dynamics. However, a lot more research is needed to achieve its faster convergence and obtaining global minima. Hence there has been a great interest in combining training and evolution with neural networks in recent years. The major disadvantage of the EANN [2] approach is that it is computationally expensive and has slow convergence. With a view to speed up the convergence of the search process, a number of different gradient methods such as LM and BP are combined with evolutionary algorithms. These new classes of hybrid algorithms, i.e. global evolutionary search supplemented by local search techniques are commonly known as memetic algorithms (MAs). It may be noted that the local search methods when used alone there may be problem for getting trapped in local minima. The hybridization of this local search with evolutionary techniques is useful to either accelerate the discovery of good solutions, for which evolution alone would take too long to discover, or to reach solutions that would otherwise be unreachable by evolution or a local method alone. It is assumed that the evolutionary search provides for a wide exploration of the search space while the local search can somehow zoom-in on the basin of attraction of promising solutions. MAs have been proven very successful across a wide range of problem domains such as combinatorial optimization [3], optimization of non-stationary functions [4], multi-objective optimization [5], bioinformatics [6], etc.

A good number of research investigations are directed in the automated design of the architecture of interconnection among neurons, which is regarded as a combinatorial optimization problem whilst in a continuous optimization problem the adjustment of the weights associated to the links between neurons, which is a continuous optimization problem. During the early 1990s, NNs were mostly trained using back-propagation, conjugate gradients or related methods. At the same time, work by

* Corresponding author.
E-mail address: bidyadharnitrkl@gmail.com (B. Subudhi).

0925-2312/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2011.02.006

Hinton and Nowlan [7] in the late 1980s provided much insight into the interplay between evolution and learning. Other works [8–11] followed similar trends, which reinforced the perception that, in order to distill an evolutionary algorithm that could achieve maximum performance on a real-world application, much domain knowledge needs to be incorporated. Domain knowledge is very often encoded by means of problem specific local search.

Research on Memetic Algorithms has progressed substantially, and several Ph.D. dissertations have been written analyzing this search framework and proposing various extensions to it [3,12,13,14].

A variant of evolutionary computing namely the Differential Evolution [15–19] is a population based stochastic optimization method similar to genetic algorithm [4] that finds an increasing interest in the recent year as an optimization technique in the identification of nonlinear systems due to its achievement of a global minimum. However, a little work has been reported on memetic differential evolution learning of neural network. Therefore, it attracts the attention of the present work for neural network training. In this work, a differential evolution hybridized with back propagation has been applied as an optimization method for feed-forward neural network. Differential Evolution (DE) is an effective, efficient and robust optimization method capable of handling nonlinear and multimodal objective functions. The beauty of DE is its simple and compact structure which uses a stochastic direct search approach and utilizes common concepts of EAs. Furthermore, DE uses few easily chosen parameters and provides excellent results for a wide set of benchmark and real-world problems. Experimental results have shown that DE has good convergence properties and outperforms other well known EAs [19]. Therefore, there is scope of using DE approach to neural weight optimization. In comparison to a gradient based method differential evolution seems to provide advantage in terms convergence speed and finding global optimum.

A nonlinear system as considered in [20–22] has been chosen in this work for demonstrating the efficacy of the proposed hybrid evolutionary system identification. In this work, the authors propose a hybrid approach in which the local search methods (BP) acts as an operator in the global search algorithm in view of achieving global minimum with good convergence speed. In this work, memetic genetic algorithm and particle swarm optimization are compared with differential evolution, which are individually combined with BP for training a feed-forward neural network. First two examples shows the comparison of different algorithms which proves that the proposed DEBP is having better identification capability in comparison to other. In the last example the proposed DEBP is applied to a Box–Jenkin's real time problem and its efficacy has been proved from the correlation analysis.

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. DE with a local search algorithm, i.e. BP for getting faster convergence in comparison to only evolutionary computation and to avoid the possibility of the search process being trapped in local minima which is the greatest disadvantage of local search optimization.
- BP has been integrated as an operator in global searches for optimizing the weights of the neural network training enabling faster convergence of the EANN employed for nonlinear system identification.
- The identification performance of the proposed DEBP scheme has been compared with other EANN and BPNN approaches to nonlinear system identification and found to be better in terms of identification performance and convergence speed.

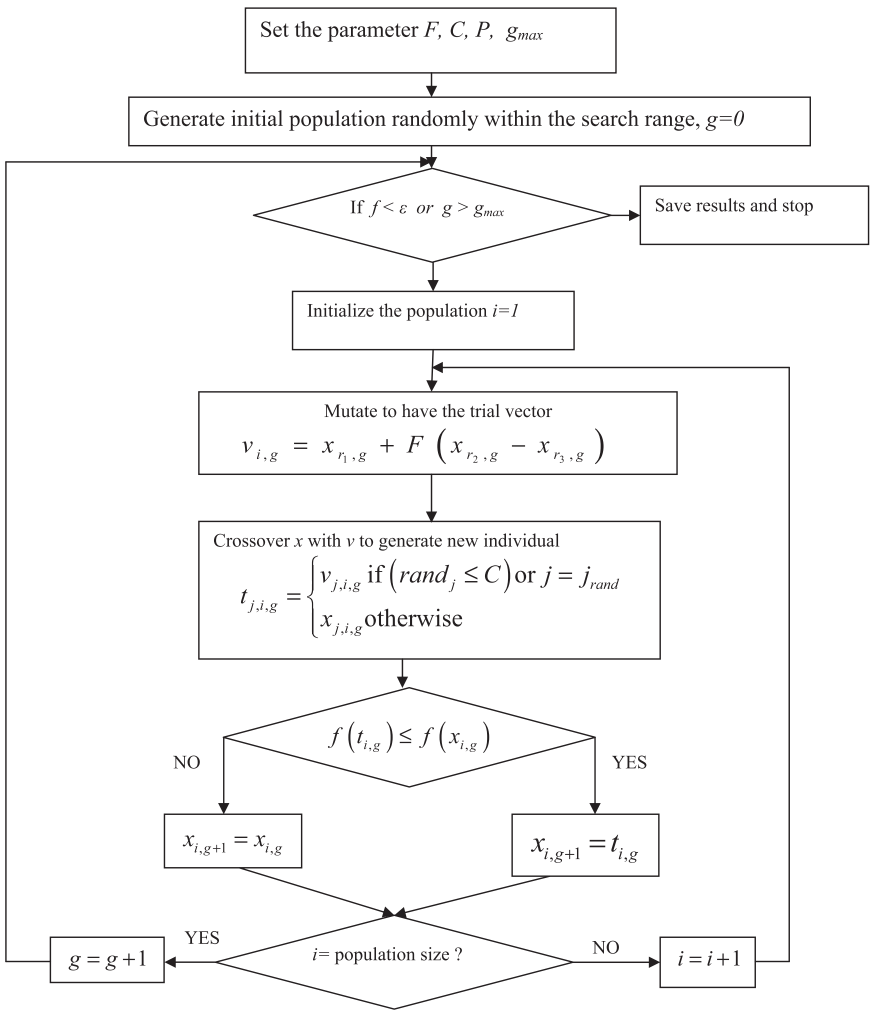
2. A brief review on differential evolution strategy

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 (DE) strategy 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 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, which differ from each other 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 $i=1,2,\ldots,P$ 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 a predefined criterion 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:

$$v_{i,g}=x_{r_1,g}+F(x_{r_2,g}-x_{r_3,g}) \tag{1}$$

In Eq. (1), F is the mutation factor, which provides the amplification to the difference between two individuals $(x_{r2,g}-x_{r3,g})$ so as to avoid search stagnation and it is usually taken in the range of [0,1]. Where $r_1,r_2,r_3\in\{1,2,\ldots,P\}$ 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, $(0\leq C\leq 1)$. 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,



Set the parameter F, C, P, g_{max}
Generate initial population randomly within the search range, $g=0$
If $f<\varepsilon$ or $g>g_{max}$
Save results and stop
Initialize the population $i=1$
Mutate to have the trial vector $v_{i,g} = x_{r_1,g} + F(x_{r_2,g} - x_{r_3,g})$
Crossover x with v to generate new individual $t_{j,i,g} = \begin{cases} v_{j,i,g} \text{ if } (rand_j \le C) \text{ or } j = j_{rand} \\ x_{j,i,g} \text{ otherwise} \end{cases}$
$f(t_{i,g}) \le f(x_{i,g})$
NO
YES
$x_{i,g+1} = x_{i,g}$
$x_{i,g+1} = t_{i,g}$
YES
$g=g+1$
$i=$ population size ?
NO
$i=i+1$

Fig. 1. Block diagram for DE algorithm.

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.

$$t_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } (rand_j \le C) \text{ or } j = j_{rand} \\ x_{j,i,g} & \text{otherwise} \end{cases} \tag{2}$$

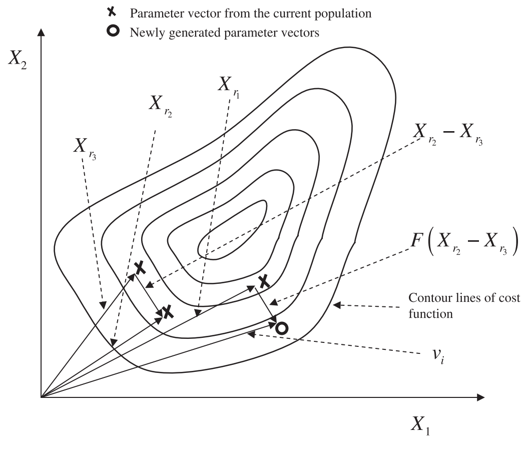
$j=1,2,\ldots,d$, where d is the number of parameters to be optimized.

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.

$$x_{i,g+1} = \begin{cases} t_{i,g} & \text{if } f(t_{i,g}) \le f(x_{i,g}) \\ x_{i,g} & \text{otherwise} \end{cases} \tag{3}$$

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 preset maximum g_{max}.



Parameter vector from the current population
Newly generated parameter vectors
X_2
X_{r_1}
X_{r_2}
X_{r_3}
$X_{r_2} - X_{r_3}$
$F(X_{r_2} - X_{r_3})$
Contour lines of cost function
v_i
X_1

Fig. 2. Minimization in a two dimensional objective function.

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. 1 shows a two dimensional objective function that illustrates the different vectors, x_i on which differential evolution is applied. It shows the process of generating trial vector for the scheme explained in Eq. (1). Fig. 2 shows a 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.$\mathbf{E}=(1/N)\sum_{k=1}^{N}[\mathbf{y}-f(\mathbf{x},\mathbf{w})]^2$, where N is the number of data considered. The above procedure of DE strategy is explained in Fig. 1 for clarity.

3. Memetic algorithm

Memetic algorithms have received various names throughout the literature and scientist not always agree what is and what is not an MA due to the large variety of implementations available. Some of the alternative names used for this search framework are hybrid GAs, Baldwinian EAs, Lamarckian EAs, genetic local search algorithms, etc. to cover a wide range of techniques where evolutionary-based search is augmented by the addition of one or more phases of local search Fig. 3.

The natural analogies between human evolution and learning, i.e. EAs and artificial neural networks (ANNs) prompted a great deal of research into the use of MAs to evolve the design of ANNs. Some research concentrated mainly in the automated design of the architecture of interconnection among neurons, which is a combinatorial optimization problem, and others on the adjustment of the weights associated to the links between neurons, which is a continuous optimization problem. During the 1980s and early 1990s, ANNs were trained using, for example, back-propagation, conjugate gradients or related methods. At the same time, seminal work by Hinton and Nowlan in the late 1980s [20] provided much insights into the interplay between evolution and learning. Other researchers [23–26] followed similar trends, which reinforced the perception that, in order to distill an evolutionary algorithm that could achieve maximum performance on a real world application, much domain knowledge needs to be incorporated. Domain knowledge was oftentimes encoded by means of problem specific local searchers.

Research in Memetic Algorithms has progressed substantially, and several Ph.D. dissertations have been written analyzing this search framework and proposing various extensions to it [12–14].

In [27] the authors have proposed an effective particle swarm optimization (PSO) based memetic algorithm for designing artificial neural network where an effective adaptive Meta-Lamarckian learning strategy is employed to decide which local search method to be used so as to prevent the premature convergence and concentrate computing effort on promising neighbor solutions. Delgado et al. [28] propose two hybrid evolutionary algorithms as an alternative to improve the training of dynamic recurrent neural networks.

3.1. Evolutionary algorithms + local search = memetic algorithms

There are a number of benefits that can be gained by combining the global search of EAs with local search or other methods for improving and refining an individual's solution. However, as there are no free lunches these benefits must be balanced against an increase in the complexity in the design of the algorithm. That is, a careful consideration must be place on exactly how the hybridization will be done. Consider for example the memetic algorithm template in figure given below. This a particular structure of memetic algorithm that has been considered in our work. The hybridization could be done in many ways of applying the local search inside the global algorithm. For example, the initial population could be seeded with solutions arising from sophisticated problem specific heuristics, the crossover (mutation) operator could be enhanced with domain specific and representation specific constrains as to provide better search ability to the EA. Moreover, local search could be applied to any or all of the intermediate sets of solutions. However, the most popular form of hybridization is to apply one or more phases of local search, based

```
Algorithm 1 DE Algorithm

Require: Initialization $pop$: Initial population, $F$: Scale factor, $C$: Cross over
  constant
while Convergence criteria not met do
  for $i=0$ to $P$ do
      $r_1 = rand(P)$
      $r_2 = rand(P)$
      $r_3 = rand(P)$
      $i \neq r_1 \neq r_2 \neq r_3$
      $v_{i,g} = x_{r_1,g} + F(x_{r_2,g} - x_{r_3,g})$
      $t_{j,i,g} = \begin{cases} v_{j,i,g} \text{ if } (rand_j \leq C) \text{ or } j = j_{rand} \\ x_{j,i,g} \text{ otherwise} \end{cases}$
    if $f(t_{i,g}) \leq f(x_{i,g})$ then
      $x_{i,g+1} = t_{i,g}$
    else
      $x_{i,g+1} = x_{i,g}$
    end if
  end for
end while
```

Fig. 3. Pseudo-code of the differential evolution (DE).

on some probability parameter, to individual members of the population in each generation. In our work as shown we have applied the local search the selection method, i.e. a fine search is applied to the offspring's before entering to the next generation.

3.2. Lamarckianism and Baldwinian effect

When integrating local search with evolutionary search we are faced with the dilemma of what to do with the improved solution that is produced by the local search. That is, suppose that individual i belongs to the population P in generation g and that the fitness of i is $f(i)$. Furthermore, suppose that the local search produces a new individual i^{new} with $f(i^{new})<f(i)$ for a minimization problem. The designer of the algorithm must now choose between two alternative options.

(1) Repacing i with i^{new}, in which case $P=P-\{i\}+\{i^{new}\}$ and the genetic information in i is lost and replaced with that of i'.
(2) The genetic information of i is kept but its fitness altered: $f(i)=f(i^{new})$.

The first option is commonly known as Lamarckian learning while the second option is referred to as Baldwinian Learning. It is a priori difficult to decide what method is best, and probably no one is better in all cases. In our study we have considered the Lamarckianism which tends to substantially accelerate the evolutionary process.

In the sequel we describe a novel memetic algorithm for the training of neural networks together with its main characteristics. Let us now discuss the hybrid scheme. Different approaches can be followed when combining a global optimizer and a local search (LS). In a hybrid algorithm, LS allows to efficiently explore the region of the fitness landscape in an individual's neighborhood.

To describe this mechanism with more details, we will introduce the following formal framework. Let us first introduce the population cardinality P and the population at the gth generation, $\Pi_g=\{\underline{c}_{1,g},\ldots,\underline{c}_{P,g}\}$. Let H be the operator may be mutation/crossover/reproduction defined as

$$H|\Pi_g\in\mathbb{R}^{d*P}\rightarrow\underline{x}(x_1,\ldots,x_P)\in\mathbb{R}^P$$

which associate to each population the fitness vector of its elements.

Let R, M be the recombination and the mutation operators respectively. These operators are called the reproduction operators as well and are defined as

$$R|\Pi\in\mathbb{R}^d\times\mathbb{R}^P\rightarrow\Pi'\in\mathbb{R}^d\times\mathbb{R}^P$$

$$M|\Pi'\in\mathbb{R}^d\times\mathbb{R}^P\rightarrow\Pi''\in\mathbb{R}^d\times\mathbb{R}^P$$

Let us denote with LS the local search operator, i.e., the operator which produces a new population by applying the LS with starting points equal to the individuals in the current population:

$$LS|\Pi''\in\mathbb{R}^d\times\mathbb{R}^P\rightarrow\Pi'''\in\mathbb{R}^d\times\mathbb{R}^P$$

where d is the number of parameters and P is the number of population. Then applying the selection operator the next generation population can be determined.

$$\Pi_{g+1}=selection\{R(M[LS(\Pi_g)])\}$$

In a Hybrid Evolutionary Algorithm, the role of the Evolutionary Algorithm is essentially to explore the searching space and locate the more promising regions. Fig. 4 shows how to produce next generation of the proposed algorithm whereas the Pseudo-code is given in Fig.5.

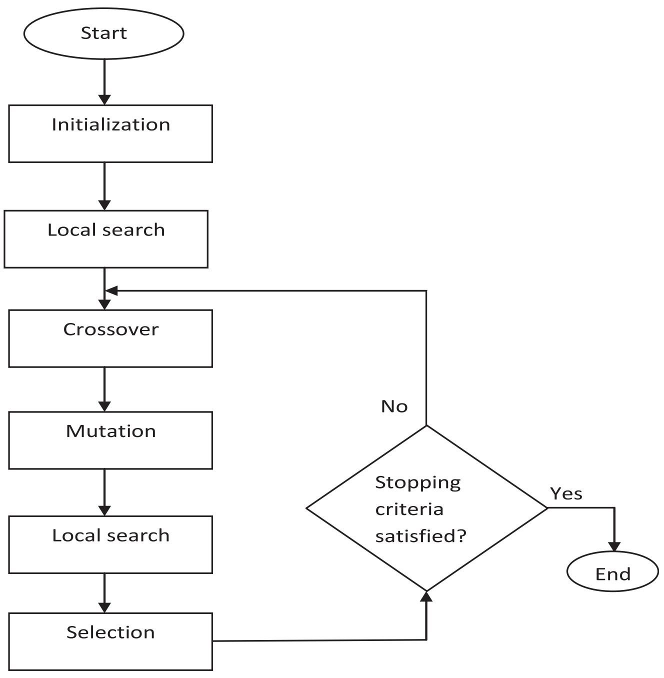


Fig. 4. Scheme of the memetic algorithm..

```
Memetic Procedure
Generate an Initial Population of Solutions
Repeat
    • Apply the mutation operator to the initial population set
    • Apply DE recombination
    • Perform local search with starting from each member of the offspring set
    • Replace the worst solutions in the population with the new offspring
Until reaching some stopping criterion
```

Fig. 5. Pseudo-code of the proposed memetic algorithm.

4. Proposed differential evolution back-propagation training algorithm for nonlinear system identification

In the sequel, we describe how a memetic differential evolution (DE) is applied for training neural network in the framework of system identification (see Algorithm 1). DE [30–36] 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 $\mathbf{w}$ and input values $\mathbf{x}$, i.e. $\mathbf{y}=f(\mathbf{x},\mathbf{w})$. The role of BP in the proposed algorithm has been described in Section 1. In the training process, both the input vector $\mathbf{x}$ and the output vector $\mathbf{y}$ are known and the synaptic weights in $\mathbf{w}$ are adapted to obtain appropriate functional mappings from the input $\mathbf{x}$ to the output $\mathbf{y}$. Generally, the adaptation can be carried out by minimizing the network error function $\mathbf{E}$ which is of the form $\mathbf{E}(\mathbf{y},f(\mathbf{x},\mathbf{w}))$. In this work we have taken $\mathbf{E}$ 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 $\mathbf{E}$ by optimizing the values of the network weights, $\mathbf{w}=(w_1,\ldots,w_d)$.

Algorithm 1. Differential Evolution Back-Propagation (DEBP) Identification Algorithm:

Step 1:
Initialize population **pop**: Create a population from randomly chosen object vectors with dimension *P*, where *P* is the number of population

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

where *d* is the number of weights in the weight vector. In $\mathbf{w}_{i,g}$, *i* is index to the population and *g* is the generation to which the population belongs.
Step 2:
Evaluate all the candidate solutions inside the **pop** for a specified number of iterations.
Step 3:
For each *i*th candidate in **pop**, select the random population members, $r_1,r_2,r_3 \in \{1,2,\ldots,P\}$
Step 4:
Apply a mutation operator to each candidate in a population to yield a mutant vector, i.e.

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

where "*F*" denotes the scale factor.
Step 5:
Apply crossover, i.e. each vector in the current population is recombined with a mutant vector to produce trial vector.

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

Step 6:
Apply Local Search (back propagation algorithm), i.e. each trial vector will produce a lst-trial vector $lst_{j,i,g+1} = bp(t_{j,i,g+1})$
Step 7.
Apply selection, i.e. between the local search trial (lst-trial) vector and the target vector. If the lst-trial vector has an equal or lower objective function value than that of its target vector, it replaces the target vector in the next generation; otherwise, the target retains its place in the population for at least one more generation

$$\mathbf{w}_{i,g+1} = \begin{cases} lst_{i,g+1} & \text{if } \mathbf{E}(\mathbf{y},f(\mathbf{x},\mathbf{w}_{i,g+1})) \le \mathbf{E}(\mathbf{y},f(\mathbf{x},\mathbf{w}_{i,g})) \\ \mathbf{w}_{i,g} & \text{otherwise} \end{cases}$$

Step 8:
Repeat steps 1–7 until stopping criteria (i.e. maximum number of generation) is reached

5. Results and discussion

In this section we present the performance of the proposed Differential Evolution Back-Propagation (DEBP) Identification Algorithm using the simulation studies on a benchmark problem for the identification of a nonlinear discrete system [16] expressed by

$$y_p(k+1) = \frac{y_p(k)[y_p(k-1)+2][y_p(k)+2.5]}{8.5+[y_p(k)]^2+[y_p(k-1)]^2} + u(k) \tag{4}$$

where $y_p(k)$ is the output of the system at the *k*th time step and $u(k)$ is the plant input which is uniformly bounded function of time. The plant is stable for $u(k) \in [-2\ 2]$.

For the identification of the plant described in Eq. (4), let the neural model be in the form of

$$\hat{y}(k+1) = f(y_p(k),y_p(k-1)) + u(k) \tag{5}$$

where $f(y_p(k),y_p(k-1))$ is the nonlinear function of $y_p(k)$ and $y_p(k-1)$. The inputs to the neural network are $u(k)$, $y_p(k)$ and $y_p(k-1)$.The output from neural network is $\hat{y}(k+1)$. The goal is to train the neural network such that when an input $u(k)$ is simultaneously presented to the nonlinear system (4) as well as to neural network (5), the neural network outputs $\hat{y}(k+1)$ will finally approach the nonlinear system output $y_p(k+1)$ as close as possible. In the following discussions, we will present our observation on nonlinear system identification schemes using seven different identification algorithms with comparison of their identification performances.

Fig. 6 shows the neural network based system identification scheme for the given plant employing the proposed memetic algorithm. The role of MA here is to train the weights of the neural network optimally.

In case of MLPNN architecture one hidden layer is sufficient to guarantee the universal approximation feature. Fig. 7 illustrates this kind of network.

The two-layer feed-forward neural network with sigmoid activation function in the hidden layer and linear activation function in output layer has the ability to approximate nonlinear function if the number of neurons in the hidden layer is sufficiently large. The Feed-forward neural network (FNN) used in this work is shown in Fig. 7. The inputs $u(k-1)$, $u(k-2)$, …, $u(k-n_u)$ and outputs $y(k-1)$,$y(k-2)$,…,$y(k-n_y)$ are multiplied with the weights $w_{u(i,j)}$ and $w_{y(i,j)}$, respectively, and summed at each hidden node. Then the summed signal at a node activates a nonlinear function (sigmoid function). Thus, the output $\hat{y}(k)$ at a linear output node can be

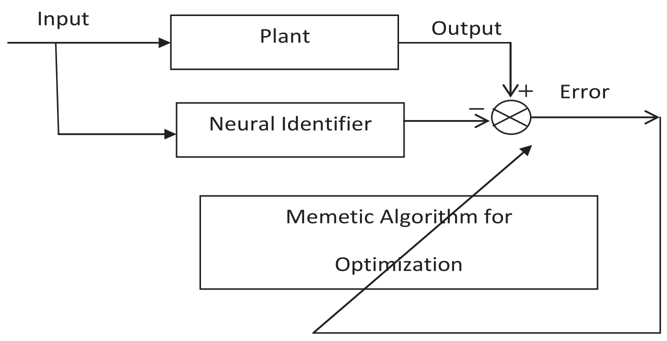


Fig. 6. Neural network based identification scheme.

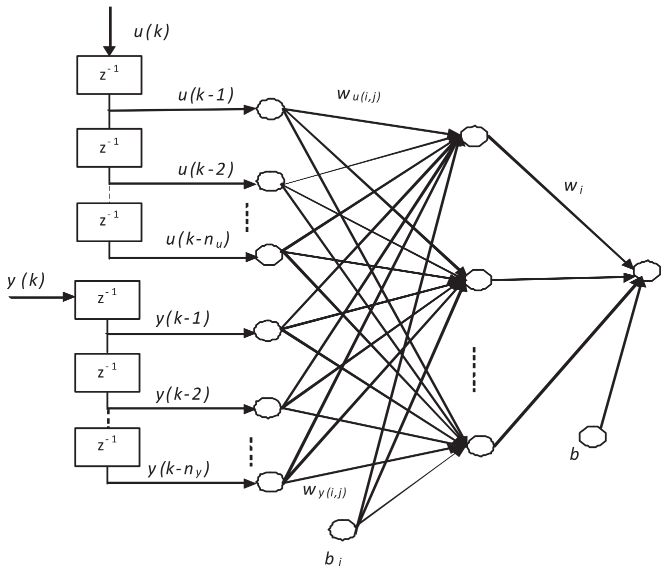


Fig. 7. NN identifier with external dynamics.

calculated from its inputs as follows:

$$\hat{y}(k)=\sum_{i=1}^{nH} w_{k,i}\frac{1}{1+e^{-\left(\sum_{j=1}^{n_u} u(k-j)w_{u(i,j)}+\sum_{j=1}^{ny} y(k-j)w_{y(i,j)}+b_i\right)}}+b \tag{6}$$

where n_u+n_y is the number of inputs n_H is the number of hidden neurons, $w_{u(i,j)}$ is the first layer weight between the input $u\ (k-j)$ and the ith hidden neuron, $w_{y(i,j)}$ is the first layer weight between the input $y\ (k-j)$ and the ith hidden neuron, w_i is the second layer weight between the ith hidden neuron and output neuron, b_i is a biased weight for the ith hidden neuron and b is a biased weight for the output neuron. It can be seen from Fig. 7 that the FNN is a realization of the Nonlinear Auto Regressive Exogenous (NARX) model. The difference between the output of the plant $y_p(k)$ and the output of the network $\hat{y}(k)$ is called the prediction error: $e_i(k)=y_p(k)-\hat{y}(k)$. This error is used to adjust the weights and biases in the network via the minimization of the error function

$$E=\tfrac{1}{2}\left[y_p(k)-\hat{y}(k)\right]. \tag{7}$$

In applying different system identification techniques to nonlinear systems considered in Eq. (4) we conducted several sets of simulation experiments with different number of hidden units. During these experiments we observed a MLP neural network model of $3\times21\times1$ configuration has been used for identifying the given nonlinear system. In other words, the NN-based model has 3 inputs, 21 neurons in hidden layer and 1 neuron in output layer. To find a suitable configuration it is common to start from a simple configuration, usually only one hidden layer, and then increase the number of neurons and even the number of layers if necessary. After 100 epochs the training of the neural identifier has been stopped. During training period, input $u(k)$ was a random white noise signal, but after the training is over, its prediction capability were tested for input given by

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

5.1. *Identification using back-propagation algorithm (BP) (Figs. 8 and 9)*

Fig. 8 shows the system identification results obtained using the back propagation algorithm for training the feed-forward neural network. Fig. 9 shows the identification error plot obtained with BP identification.

5.2. *Identification using genetic algorithm (GA) (Figs. 10 and 11)*

Fig. 10 shows the identification performance of the system using genetic algorithm as learning algorithm for the given neural network. The same neural network configuration, i.e. twenty one neurons are taken into account. After 100 epochs it was found that the squared error is more than conventional back propagation also taking more time to converge. Fig. 11 shows the error between actual and GA identification.

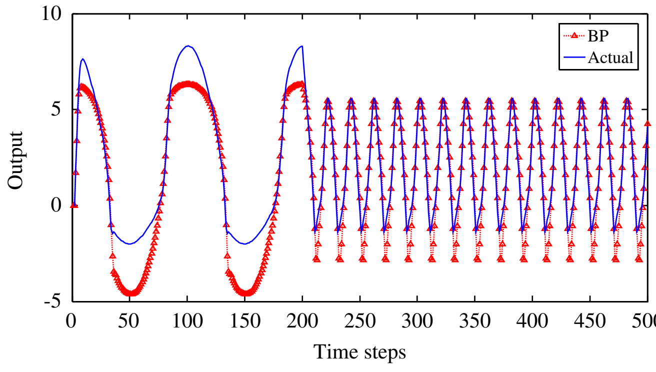


Fig. 8. BP identification performance.

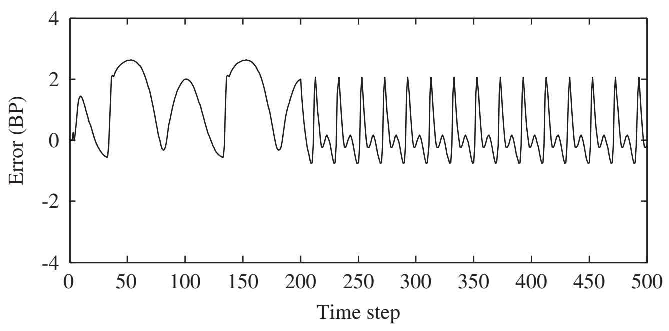


Fig. 9. Error in modeling (BP identification).

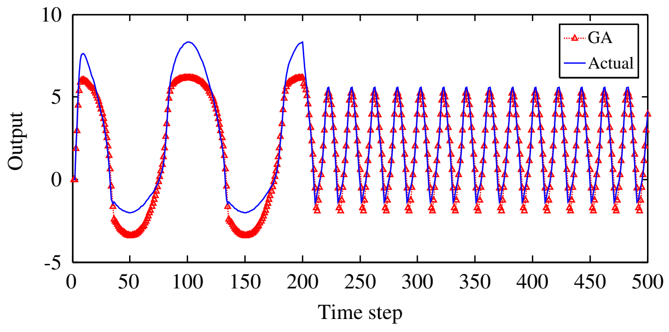


Fig. 10. GA identification performance.

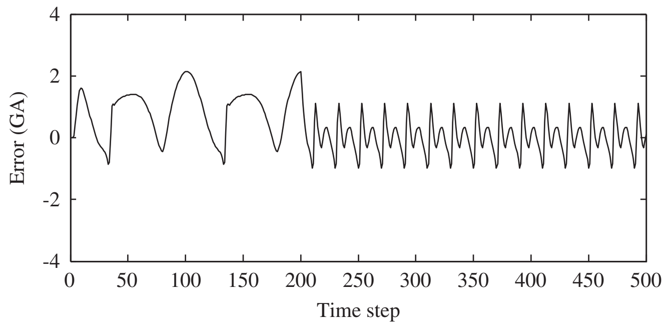


Fig. 11. Error in modeling (GA identification).

5.3. *Genetic algorithm back-propagation (GABP) identification (Figs. 12 and 13)*

Fig. 12 shows the comparison of identification performance between the GABP identification scheme and the actual plant output. The identification error is shown in Fig. 13.

5.4. *Identification using particle swarm optimization (PSO) (Figs. 14 and 15)*

Fig. 14 shows the identification performance between the particle swarm optimization and the actual output. The identification error between the actual output and the PSO output is shown in Fig. 15.

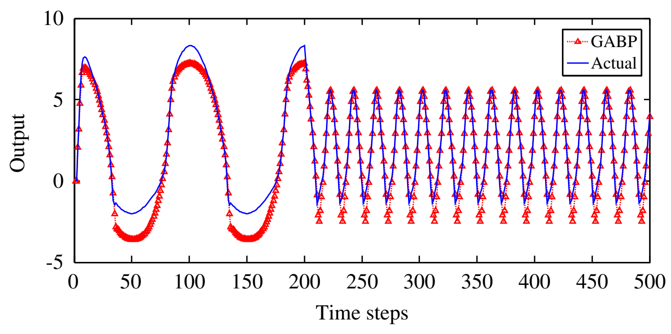


Fig. 12. GABP identification. performance.

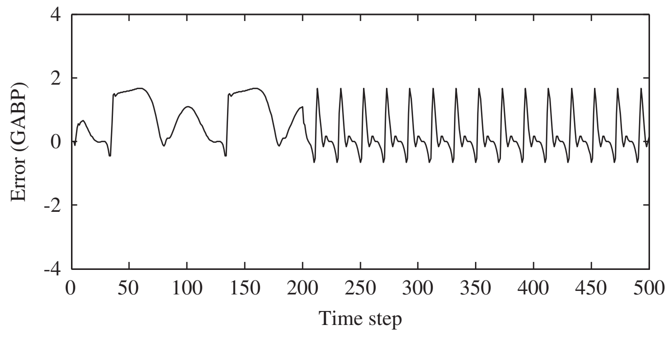


Fig. 13. Error in modeling (GABP identification).

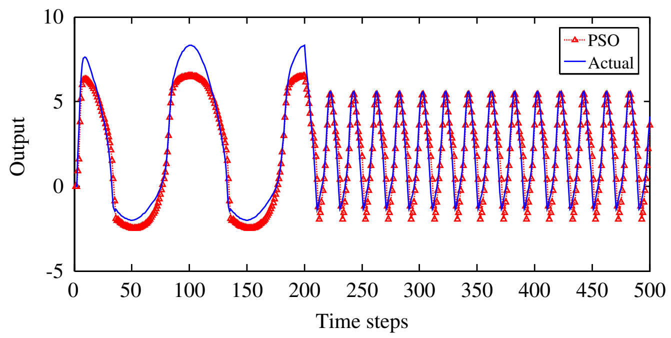


Fig. 14. PSO identification performance.

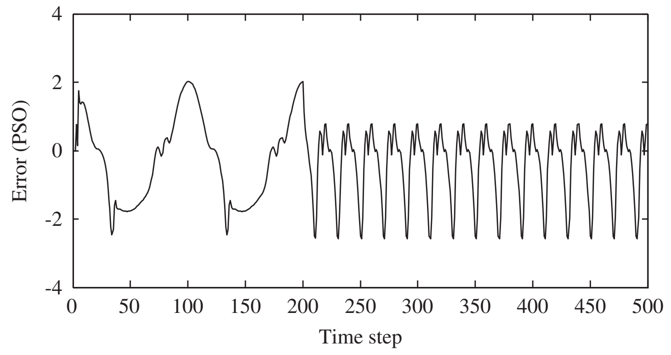


Fig. 15. Error in modeling (PSO identification).

5.5. Particle swarm optimization combined with back-propagation (PSOBP) identification (Figs. 16 and 17)

Fig. 16 shows the result of memetic scheme (PSOBP) where particle swarm optimization is hybridized with back propagation. The result clearly indicates even if the above scheme gives less sum squared error than PSO at the moment of testing, but does not give better identification of nonlinear system at validation

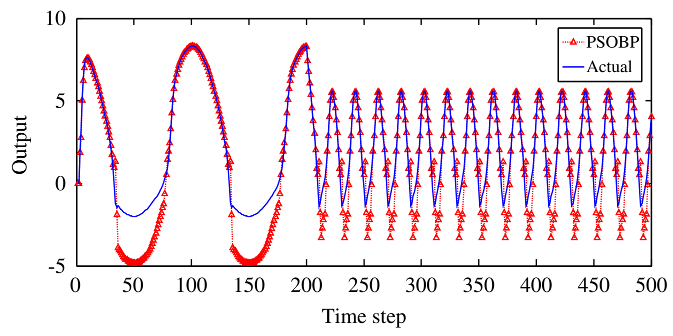


Fig. 16. PSOBP identification performance.

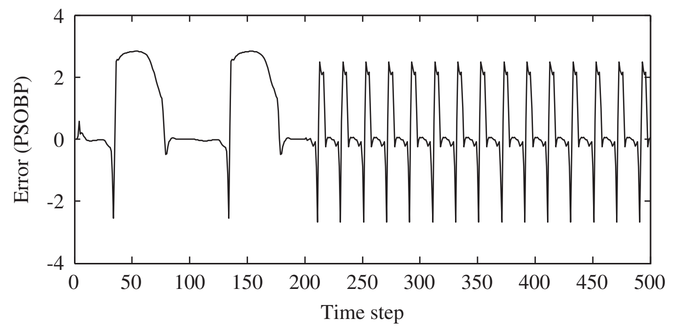


Fig. 17. Error in modeling (PSOBP identification).

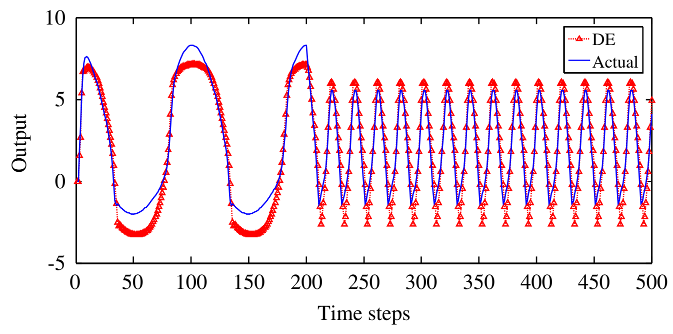


Fig. 18. DE identification performance.

stage. Fig. 17 shows its identification error curve between actual and GABP system identification.

5.6. Differential evolution (DE) identification (Figs. 18 and 19)

Fig. 18 shows the identification performance between the differential evolution and the actual output. It was found that the performance is better than GA and PSO but worst than GABP. The identification error between the actual output and the DE output is shown in Fig. 19.

5.7. Differential evolution plus the back-propagation (DEBP) identification (Figs. 20 and 21)

From Fig. 20 it is clear that the proposed method, i.e. DEBP identification is more effective than other mentioned approaches as per as identification performance and speed of convergence is concerned. Fig. 21 shows the error between the plant output and NN identified model output.

5.8. Performance comparison of all the seven identification methods cited in this paper

Fig. 22 depicts the mean square error (MSE) profiles for all the seven different identification methods (BP, GA, GABP, PSO, PSOBP, DE and DEBP). In these seven methods, we have proposed a new identification scheme, namely the Differential Evolution plus the Back-propagation (DEBP) identification approach. From this figure it is clear that the SSE with the proposed method DEBP converges to zero very fast taking only about 20th iteration while the error curves with the other system identification methods (BP, GA,GABP, PSO, PSOBP and DE) converges to zero taking over 70th iterations. Hence it is important to note that the proposed DEBP system identification exhibits better convergence characteristics Fig. 23.

All the simulations have been performed in an Intel® core (TM) DUO CPU with 3 GHz speed 3 GB RAM and 32 bit operating system. The programming language used is MATLAB with same set of parameters, i.e. population size, number of generations, upper and lower bounds of weights and number of hidden layer neurons.

Table 1 gives the value of parameters used in all the seven identification schemes. Table 2 gives the comparison of performance of all the seven methods in terms of mean squared error (MSE). From the results it is clear that for a particular number of generations, i.e. 100, the proposed DEBP algorithm has a mean squared error (MSE) of 0.0625. It is found that out of all the seven methods the memetic approaches, i.e. GABP, DEBP, and PSOBP are having faster convergence in comparison to other local search and evolutionary computing approaches. Finally it is concluded that the proposed memetic DEBP is having better identification performance and faster convergence in comparison to memetic GABP and PSOBP algorithm which indicates DE is outperforming than its counterpart GA and PSO.

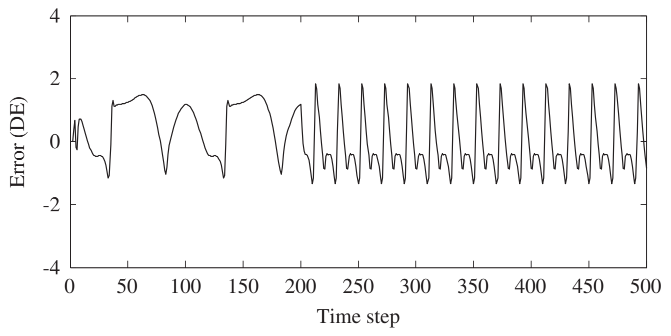


Fig. 19. Error in modeling (DE identification).

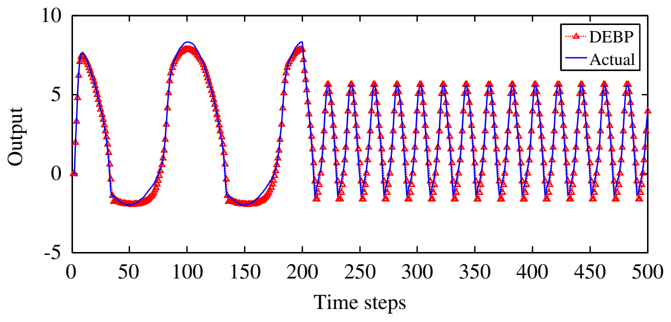


Fig. 20. DEBP identification performance.

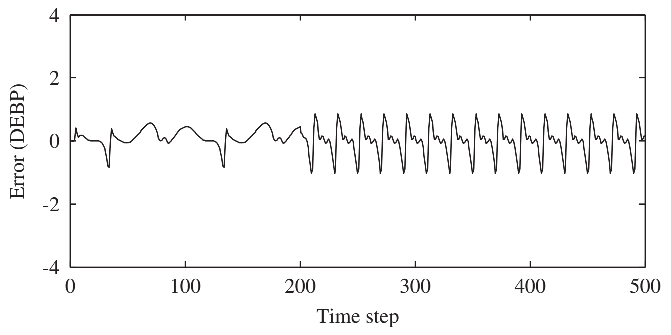


Fig. 21. Error in modeling (DEBP identification).

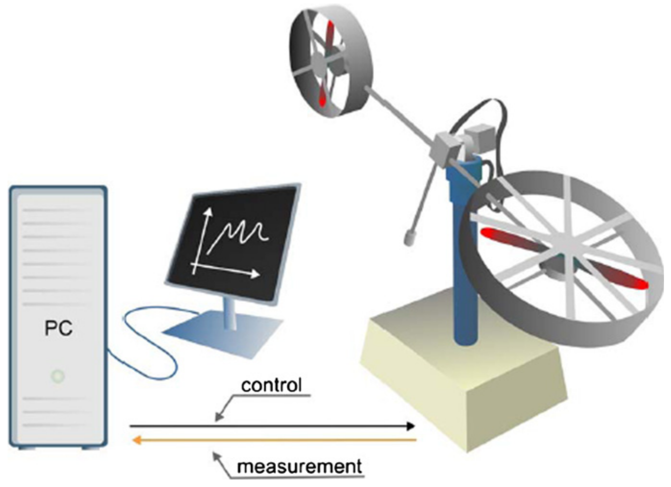


Fig. 23. The laboratory setup: TRMS system.

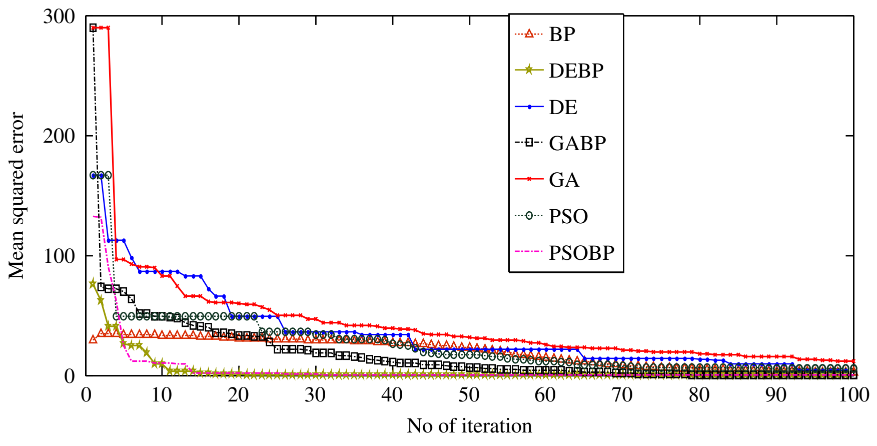


Fig. 22. A comparisons on the convergence on the mean squared error (MSE) for all the seven methods.

Table 1
Parameters used in simulation studies.

Total sampling period, T	500
Population size, P	50
Number of generations	100
Upper and lower bound of weights	$[-1, 1]$
BP learning parameter, η	0.55
Number of hidden layer neurons	21
Parameters for DE and DEBP algorithms	
Mutation constant factor, F	0.6
Cross over constatnt, C	0.5
Bp learning parameter, η	0.55
Parameters for GA and GABP algorithms	0.002
Mutation probability, p_m	1
BP learning parameter, η	0.55
Parameters for PSO and PSOBP algorithms	
Learning factor, C_1	1.9
Learning factor, C_2	1.9
BP learning parameter, η	0.55

Table 2
Comparison of performances of seven methods.

SL no.	Identification method	Computation time in seconds (s)	MSE	Number of generation at which the MSE converges to 0.05
1	BP	4.76	2.6086	>100
2	GA	40.42	11.4156	>100
3	GABP	131.42	0.2852	70
4	PSO	42.15	5.49	>100
5	PSOBP	142.79	0.2074	50
6	DE	42.19	3.9645	>100
7	DEBP	136.73	0.0625	20

***Example* 2**: The TRMS used in this work is supplied by Feedback Instruments designed for control experiments. This TRMS setup serves as a model of a 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. 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 instant's$(t-1)$.

5.9. Differential evolution (DE) and differential evolution back propagation (DEBP) identification

Figs. 24–28 shows the identification performance of 1 degree of freedom (DOF) vertical DE and DEBP based model. Fig. 24 compares the actual output, $y(t)$ and identified plant output $\hat{y}(t)$ within the time step of 0–500. As the identification performances shown in Fig. 24 are overlapping each other, in Fig. 25 we have shown the results within the time step of 88–96. From this it is clear that the proposed DEBP exhibits better identification ability compared to DE approach. Figs. 26 and 27 shows the error between the actual and identified model. Fig. 28 gives the sum

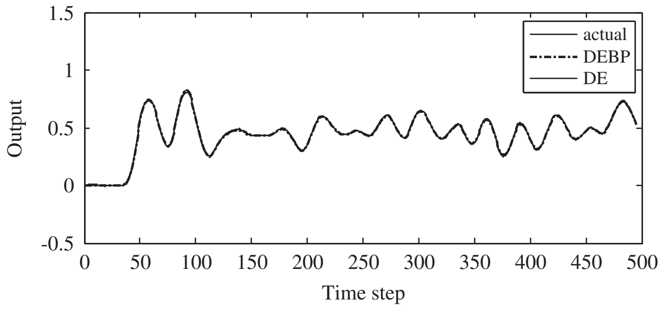


Fig. 24. DE and DEBP identification performance.

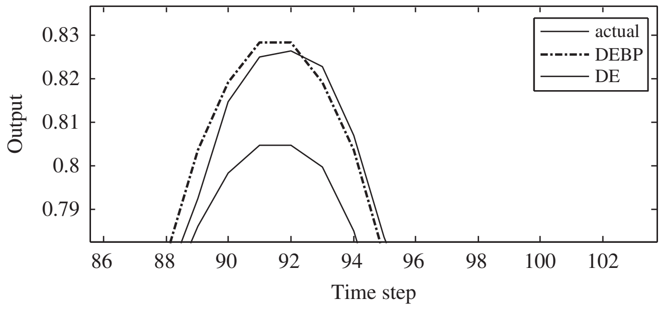


Fig. 25. DE and DEBP identification performance.

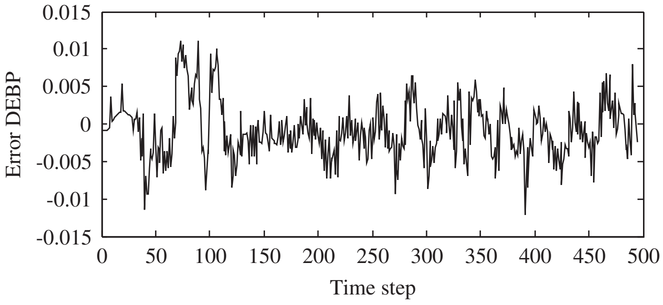


Fig. 26. Error in modeling (DEBP identification).

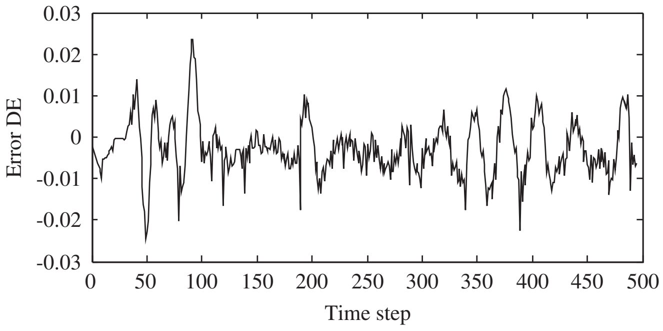


Fig. 27. Error in modeling (DE identification).

squared error (SSE) where it is found that the value of SSE for DEBP is 0.0036 whereas for DE identification is 0.0110.

5.10. Genetic algorithm (GA) and genetic algorithm back propagation (GABP) identification

Figs. 29–33 shows the identification performance of 1 degree of freedom (DOF) vertical GA and GABP based model. Fig. 18

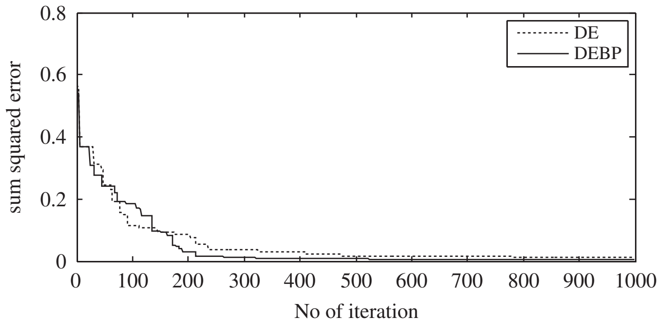


Fig. 28. A comparisons on the convergence on the sum squared error (SSE) (DE, DEBP).

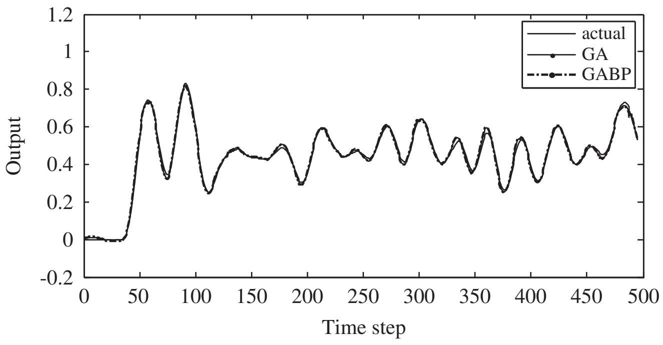


Fig. 29. GA and GABP identification performance.

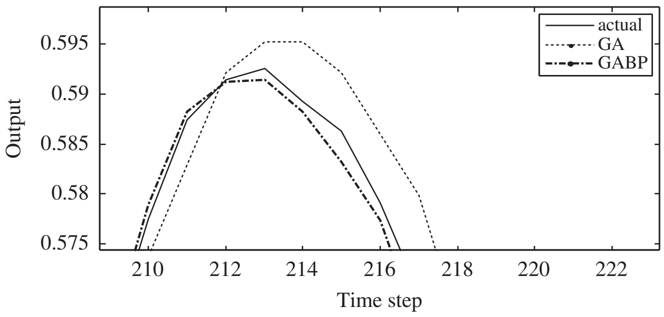


Fig. 30. GA and GABP identification performance.

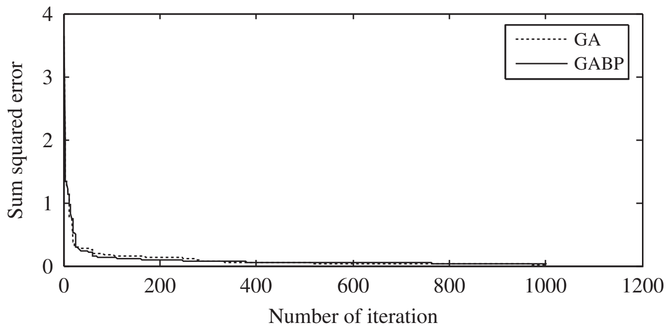


Fig. 31. A comparisons on the convergence on the sum squared error (SSE) (GA, GABP).

compares the actual output, $y(t)$ and identified plant output $\hat{y}(t)$ within the time step of 0–500. As the identification performances shown in Fig. 29 are overlapping each other, in Fig. 30 we have shown the results within the time step of 208–218. From this it is clear that the GABP identification approach exhibits better identification ability compared to GA approach. Fig. 31 gives the sum squared error (SSE) where it is found that the value of SSE for GABP is 0.0.0197 whereas for GA identification is 0.0327. Figs. 32

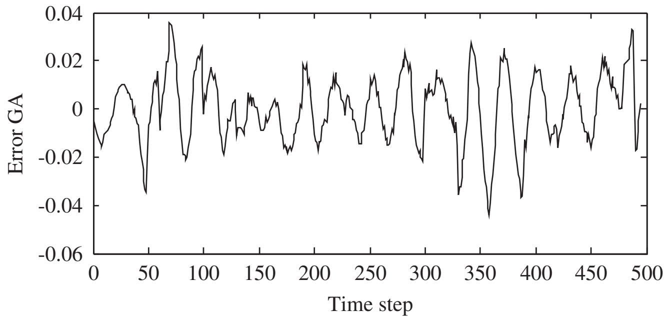


Fig. 32. Error in modeling (GA identification).

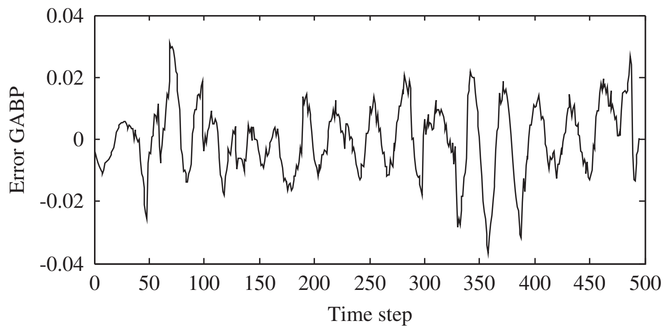


Fig. 33. Error in modeling (GABP identification).

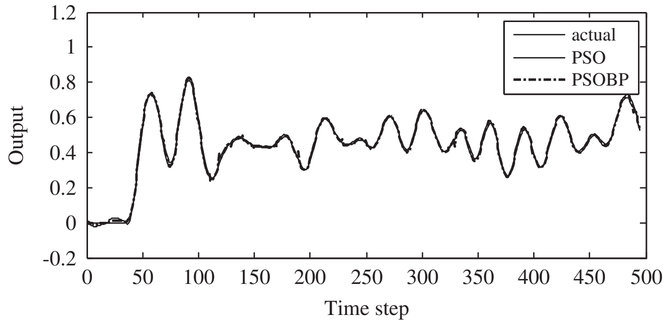


Fig. 34. PSO and PSOBP identification performance.

and 33 shows the error between the actual and identified model for both the identification scheme.

5.11. *Particle swarm optimization (PSO) and particle swarm optimization (PSOBP) identification*

Fig. 34–38 shows the identification performance of 1 degree of freedom (DOF) vertical PSO and PSOBP based model. Fig. 34 compares the actual output, $y(t)$ and identified plant output $\hat{y}(t)$ within the time step of 0–500. As the identification performances shown in Fig. 34 are overlapping each other, in Fig. 35 we have shown the results within the time step of 87–96. From this it is clear that the PSOBP approach exhibits better identification ability compared to PSO approach. Fig. 36 gives the sum squared error (SSE) where it is found that the value of SSE for PSOBP is 0.0235 whereas for PSO identification is 0.0505. Figs. 37 and 38 shows the error between the actual and identified model for both the identification scheme.

Finally it has been seen that among all the methods the proposed DEBP method is having lowest SSE, i.e. 0.0036 amongst all the methods disused.

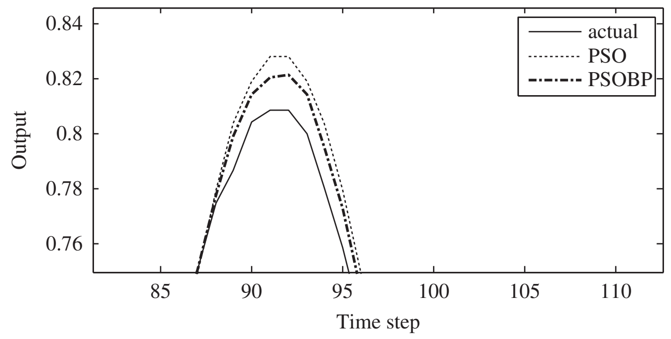


Fig. 35. PSO and PSOBP identification performance.

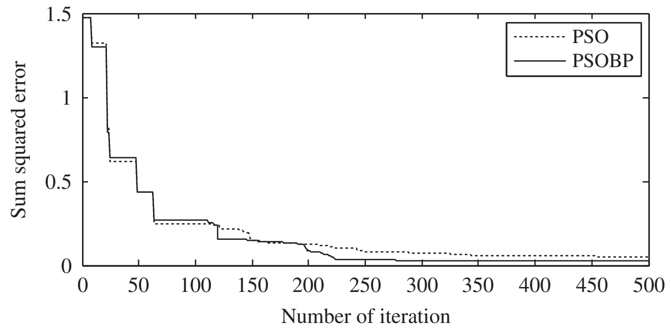


Fig. 36. A comparisons on the convergence on the sum squared error (SSE) (PSO, PSOBP).

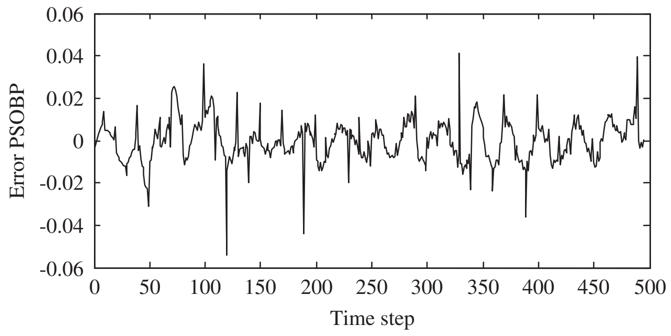


Fig. 37. Error in modeling (PSOBP identification).

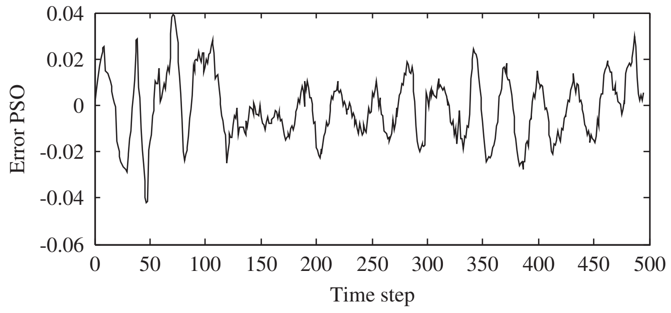


Fig. 38. Error in modeling (PSO identification).

Example 3 (Box Jenkin's Gas Furnace Modeling): The time series data set for a gas furnace consists of 296 input–output samples recorded with a sampling period of 9 s. The instantaneous values of output $y(t)$ have been regarded as being influenced by ten variables mainly the past values of $y(t)$ *for past four* sampling times and $u(t)$ for past six sampling times, i.e. $y(t-1)$, $y(t-2)$, $y(t-3)$, $y(t-4)$, $u(t-1)$, $u(t-2)$, $u(t-3)$, $u(t-4)$, $u(t-5)$, $u(t-6)$. The original data set contains 296 $[u(t), y(t)]$ data pairs. But, by converting the data set to previous sampling instants 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 data points to effectively 290 data pairs. The number of training data was taken as 100 for the three identification schemes (DE-NN, OBDE-NN and NF) and the rest 190 data pairs were considered as the test data. It may be noted that, for dynamic system modeling, the inputs selected must contain elements from both set of historical furnace outputs $\{y(t-1,),\ldots,y(t-4)\}$ and the set of historical furnace inputs $\{u(t-1),\ldots,u(t-6)\}$. In this study we assumed six inputs are fed to the neural networks namely, $y(t-1)$, $y(t-2)$, $y(t-3)$, $u(t-1)$, $u(t-2)$, $u(t-3)$. During the experiment, we observed the pattern of estimation errors corresponding to the number of hidden nodes taken. By this process we end up with choice of eleven numbers of hidden units, leading to the lowest estimation error. For all the methods eleven number of hidden layer neurons were taken and the results obtained after 1000 epochs. We have tried for more number of neurons for the same problem which took more computational time without achieving appreciable amount of accuracy.

5.11.1. Correlation test

A more convincing method of the identification model validation is to use correlation tests. If the model of a system is adequate then the residuals should be unpredictable from (uncorrelated with) all linear and nonlinear combinations of past inputs and outputs. A number of auto-correlation and cross-correlation tests given below has been recommended by the authors in [29]

$$\xi_{\varepsilon\varepsilon} = E[\varepsilon(t-\tau)\varepsilon(t)] = \delta(\tau)$$
$$\xi_{u\varepsilon} = E[u(t-\tau)\varepsilon(t)] = 0 \quad \forall\tau$$
$$\xi_{u^2\varepsilon^2} = E[u^2(t-\tau)-\overline{u}^2\varepsilon^2(t)] = 0 \quad \forall\tau$$
$$\xi_{\varepsilon(\varepsilon u)} = E[\varepsilon(t)\varepsilon(t-1-\tau)u(t-1-\tau)] = 0 \quad \tau \geq 0$$

where $\xi_{u\varepsilon}$ indicates the cross-correlation between $u(t)$ and $\varepsilon(t)$ and $\delta(t)$ is an impulse function. The test results are given below. In general, if the correlation functions are within the 95% confidence intervals, $\pm 1.96/N$, where N is the total number of data points, the model is regarded as adequate.

From Fig. 39 a close match, perceived from physical observation, between the neural model and the actual system response reveals that the obtained model represents the system adequately. The identification error is shown in Fig. 40. However, the effectiveness of the model is further tested by carrying the above mentioned correlation tests. It is found that all four correlation functions; cross-correlation of input and residuals (Fig. 41), auto-correlation of residuals (Fig. 42), cross-correlation of input square and residuals square (Fig. 43), cross-correlation of residuals and (input × residuals) (Fig. 44) 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.

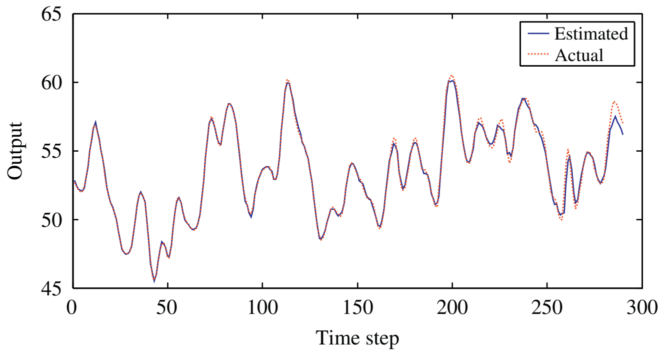


Fig. 39. Identification performance.

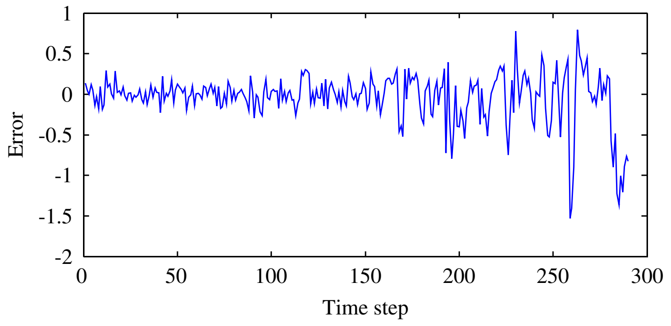


Fig. 40. Identification error.

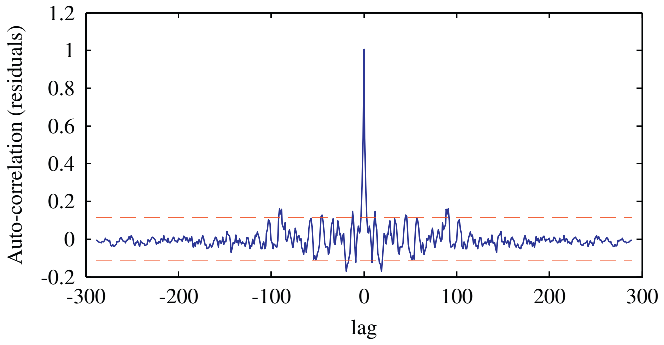


Fig. 44. Cross-correlation of residuals and input residuals.

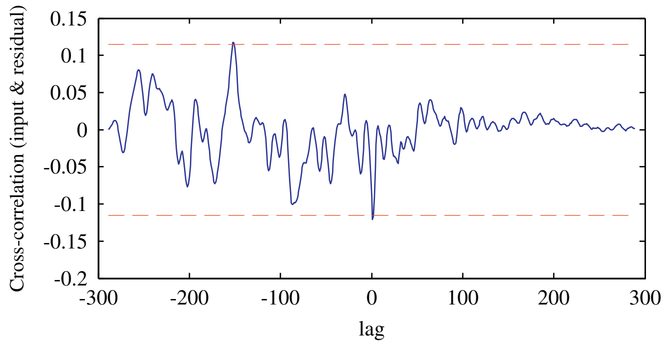


Fig. 41. Cross-correlation of input and residuals.

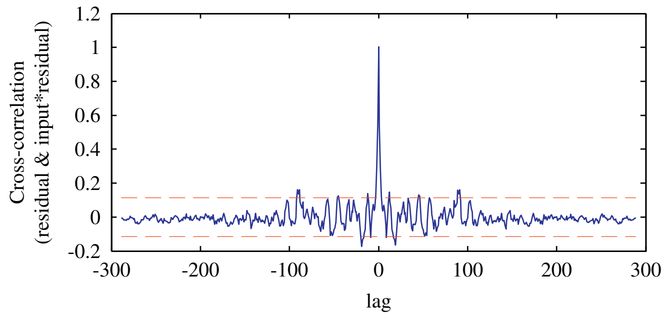


Fig. 42. Auto-correlation of residuals.

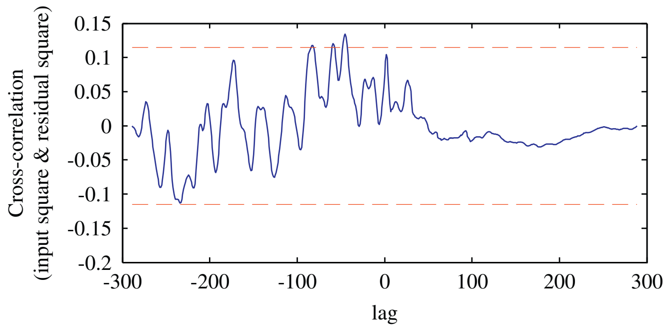


Fig. 43. Cross-correlation of input square and residuals square.

6. Conclusions

In this paper we have provided an extensive study of memetic algorithms (MAs) applied to nonlinear system identification. From the results presented in this paper it has been found that the proposed DEBP memetic algorithm applied to neural network learning exhibits better result in terms of faster convergence and lowest mean squared error (MSE) amongst all the seven methods (i.e. BP, GA, GABP, PSO, PSOBP, DE, and DEBP). The proposed method DEBP exploits the advantages of both the local search and global search. It is interesting to note that the local search pursued after the mutation and crossover operation that helps in intensifying the region of search space which leads to faster convergence. We investigated the performance of the proposed version of the DEBP algorithm using a benchmark nonlinear system identification problem, a real time Box–Jenkin's time series model and a multi-input multi-output highly nonlinear TRMS system. The simulation studies showed that the proposed algorithm of DEBP outperforms in terms of convergence velocity among all the seven discussed algorithms. The overall performance of the DEBP scheme was better than the other approaches and the overall performance of the newly proposed DEBP algorithm was superior to other methods, i.e. GABP and PSOBP. This shows it is advantageous to use DEBP over other evolutionary computation such as GA and PSO in nonlinear system identification.

References

[1] K.S. Narendra, K. Parthaasarathy, Identification and control of dynamical systems using neural networks, IEEE Trans. Neural Networks 1 (1990) 4–27.
[2] X. Yao, Evolutionary artificial neural networks, Int. J. Neural Systems 4 (1993) 203–222.
[3] P. Merz, Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 2000.
[4] F. Vavak, T. Fogarty, K. Jukes, A genetic algorithm with variable range of local search for tracking changing environments, in: Proceedings of the Fourth Conference on Parallel Problem Solving from Nature, 1996.
[5] J. Knowles, D. Corne, A comparative assessment of memetic, evolutionary and constructive algorithms for the multi-objective d-msat problem, in: Genetic and Evolutionary Computation Workshop Proceeding, 2001.
[6] N. Krasnogor, Self-generating metaheuristics in bioinformatics: the protein structure comparison case, in: Genetic Programming and Evolvable Machines, Kluwer Academic Publishers, vol. 5, 2004, pp 181–201.
[7] G. Hinton, S. Nowland, How learning can guide evolution, Complex Systems 1 (1987) 495–502.
[8] L. Whitley, S. Gordon, K. Mathias, Lamarkian evolution, the Baldwin effect, and function optimization, in: Proceedings of the Third Conference on Parallel Problem Solving from Nature, 1994.
[9] G. Mayaley, Landscapes, learning costs and genetic assimilation, Evol. Comput. 4 (1996) 213–234.
[10] P. Turney, How to shift bias: lessons from the Baldwin effect, Evol. Comput. 4 (1996) 271–295.
[11] C. Houck, J. Joines, M. Kay, J. Wilson, Empirical investigation of the benefits of partial Lamarckianism, Evol. Comput. 5 (1997) 31–60.
[12] W. Hart, Adaptive Global Optimization with Local Search, Ph.D. Thesis, University of California, San Diego, USA, 1994.
[13] M. Land, Evolutionary Algorithms with Local Search for Combinatorial Optimization, Ph.D. Thesis, University of California, San Diego, USA, 1998.
[14] N., Krasnogor, Studies in the Theory and Design Space of Memetic Algorithms, Ph.D. Thesis, University of the West of England, Bristol, UK, 2002.
[15] R. Storn, K. Price, Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.

[16] R. Storn, System design by constraint adaptation and differential evolution, IEEE Trans. Evol. Comput. 3 (1999) 22–34.
[17] J. Ilonen, J.K. Kamarainen, J. Lampinen, Differential evolution training algorithm for feed forward neural networks, Neurol. Proc. Lett. 17 (2003) 93–105.
[18] H.-K. Kim, J.-K. Chong, K.-Y. Park, Differential evolution strategy for constrained global optimization and application to practical engineering problems, IEEE Trans. Magn. 43 (3) (2007) 1565–1568.
[19] N. Noman, H. Iba, Accelerating differential evolution using an adaptive local search, IEEE Trans. Evol. Comput. 12 (1) (2008) 107–125.
[20] C.-T. Lin, C.S. George Lee, Neural Fuzzy Systems: A Neuro-fuzzy Synergism to Intelligent Systems, Prentice Hall International, Inc., New Jersey, 1996.
[21] G.E.P. Box, G.M. Jenkins, Time Series Analysis, Forecasting and Control, Holden Day, San Francisco, 1970.
[22] S.M. Ahmad, M.H. Shaheed, A.J. Chipperfield, M.O. Tokhi, Nonlinear modelling of a twin rotor MIMO system using radial basis function networks, in: Proceedings of the 2000 IEEE International Conference on National Aerospace and Electronics, 2000, pp. 313–320.
[23] L. Whitley, S. Gordon, K. Mathias, Lamarkian evolution, the Baldwin effect, and function optimization, in: Y. Davidor, H.P. Schwefel, R. Manner (Eds.), Proceedings of the Third Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science, vol. 866, Springer, , 1994.
[24] G. Mayaley, Landscapes, learning costs and genetic assimilation, Evol. Comput. 4 (1996) 213–234.
[25] P. Turney, How to shift bias: lessons from the Baldwin effect, Evol. Comput. 4 (1996) 271–295.
[26] C. Houck, J. Joines, M. Kay, J. Wilson, Empirical investigation of the benefits of partial Lamarckianism, Evol. Comput. 5 (1997) 31–60.
[27] B. Liu, L. Wang, Y. Jin, D. Huang, Designing neural networks using PSO based memetic algorithm, Adv. Neural Networks (2007) 219–224.
[28] M. Delgado, M.C. Pegalajar, M.P. Cuéllar, Memetic evolutionary training for recurrent neural networks: an application to time-series prediction, Expert Systems 23 (2) (2006) 99–115.
[29] S.A. Billings, W.S.F. Voon, Correlation based validity tests for nonlinear models, Int. J. Control 44 (1) (1986) 235–244.
[30] S. Das , A. Konar, U.K. Chakraborty, Two improved differential evolution schemes for faster global search, ACM-SIGEVO Proceedings of GECCO, Washington, DC, June 2005, pp. 991–998.
[31] S. Das, A. Abraham, U.K. Chakraborty, A. Konar, Differential evolution using a neighborhood based mutation operator, IEEE Trans. Evol. Comput. 13 (3) (2009) 526–553.
[32] S. Das, A. Konar, U.K. Chakraborty, Annealed differential evolution, IEEE Congr. Evol. Comput (2007).
[33] S. Dasgupta, S. Das, A. Biswas, A. Abraham, On stability and convergence of the population-dynamics in differential evolution, AI Commun. 22 (1) (2009) 1–20.
[34] S. Das, A. Abraham, A. Konar, Automatic clustering using an improved differential evolution algorithm, IEEE Trans. Systems Man Cybern.—Part A 38 (1) (2008) 218–236.
[35] A. Biswas, S. Das, A. Abraham, S. Dasgupta, Design of fractional-order PI ⟨lambda⟩D⟨mu⟩ controllers with an improved differential evolution, Eng. Appl. Artif. Intell. 22 (2) (2009) 343–350.
[36] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2011) 4–31 doi:10.1109/TEVC.2010.2059031.

Bidyadhar Subudhi has received a Bachelor Degree in Electrical Engineering from Regional Engineering College Rourkela (presently National Institute of Technology Rourkela, India), Master of Technology in Control & Instrumentation from Indian Institute of Technology, Delhi in 1994 and Ph.D. degree in Control System Engineering from University of Sheffield, United Kingdom in 2003. He worked as a Post Doctoral Research Fellow in the Department of Electrical & Computer Engineering, NUS, Singapore, during May–November 2005. Currently he is working as Professor and Head of the Department, Electrical Engineering in the National Institute of Technology, Rourkela, India. He is serving as the Coordinator of Centre for Industrial Electronics & Robotics, in the National Institute of Technology, Rourkela. His research interests include System Identification, Intelligent Control, Networked Control System and Photovoltaic System. He is a Fellow of the Institution of Engineers (India), Life Member of Systems Society of India and Senior Member IEEE.

Debashisha Jena has received a Bachelor of Electrical Engineering degree from University College of Engineering, Burla, India, in 1996 and Master of Technology in Electrical Engineering in 2004 and Ph.D. degree in Control System Engineering from the Department of Electrical Engineering, National Institute of Technology, Rourkela, India 2010. He was awarded a GSEP fellowship in 2008 from Canada for research in control and automation. Currently he is an Assistant Professor in the Department of Electrical & Electronics Engineering in the National Institute of Technology Karnataka, Surathkal, Mangalore, India. His research interests include Evolutionary Computation, System Identification and Neuro-evolutionary computation.