Available Online at http://www.journalajst.com

ASIAN JOURNAL OF SCIENCE AND TECHNOLOGY

ISSN: 0976-3376

Asian Journal of Science and Technology
Vol.06, Issue, 09, pp. 1777-1787, September, 2015

RESEARCH ARTICLE

A COMPARATIVE STUDY TO SOLVE THE DIFFERENTIAL ALGEBRAIC EQUATION SYSTEMS BY USING GENETIC ALGORITHM AND NEURAL NETWORKS

***[1]Wahed, M.E., [2]Abdelslam, Z.A. and Eldaken, O.M.**

[1]Department of Computer Sciences, Faculty of computers and information, Suez Canal University, Suez Canal, Egypt

[2,3]Department of Mathematics, Faculty of Science, Suez Canal, University, Suez Canal, Egypt

ARTICLE INFO

Article History:
Received 22nd June, 2015
Received in revised form
07th July, 2015
Accepted 19th August, 2015
Published online 30th September, 2015

Key words:
Differential equations,
Genetic algorithms,
Grammatical evolution,
A domain decomposition,
Neural Networks.

ABSTRACT

In this paper, the solution system of differential algebraic equations (DAES). A novel hybrid method for the solution of ordinary and partial differential equations is presented here. The method creates trial solutions in Genetic Algorithms, neural network form using a scheme based on grammatical evolution. The trial solutions are enhanced periodically using a local optimization procedure. The proposed method is tested on a series of ordinary differential equations, systems of ordinary differential equations as well as on partial differential equations with boundary conditions and the results are reported. The goal is to obtain accurate solution with reduced calculus effort by comparing the solution of the DAES obtained from Genetic algorithms, A domian decomposition method and neural network approach , A domain decomposition method and pade approximation to this problem is very close to the exact solution. An illustrative numerical method is presented for the proposed method.

INTRODUCTION

A series of problems in many scientific field scan be modelled with the use of differential equations such as problems in physics (Byrne and Ponzi, 1988; Ercan Celik and Mustafa Bayram, 2004; Chuili Sun and Juergen Hahn, 2005; Deuflhard *et al.,* 1987; Hairer and Wanner, 1991), chemistry (Lagaris *et al.,* 1998; Narendra and Parathasarathy, 1990; El-sayed Wahed, 2008), biology (Ercan Çelik *et al.,* 2006; Susmita Mall and Chakraverty, 2013), economics (Ioannis G.Tsoulos *et al.,* 2009), etc. Due to the importance of differential equations many methods have been proposed in the relevant literature for their solution such as Runge Kutta methods (Ioannis G.Tsoulos *et al.,* 2009; Melike Karta and Ercan C. elik, 2012) Predictor–Corrector (El-sayed Wahed, 2008; Susmita Mall and Chakraverty, 2013), radial basis functions (Narendra and Parathasarathy, 1990; Chakraverty, 2013), artificial neural networks (Hairer and Wanner, 1991; Narendra and Parathasarathy, 1990), models based on genetic programming (Ioannis G.Tsoulos *et al.,* 2009; Melike Karta and Ercan C. elik, 2012), etc. In this article a hybrid method utilizing constructed feed-forward neural networks by grammatical evolution and a local optimization procedure is used in order to solve ordinary differential equations (ODEs),systems of ordinary differential equations(SODEs) and partial differential equations(PDEs). The constructed neural networks with gram- matical evolution have been recently introduced by Tsoulos *et al.* [30] and it utilizes the well-established grammatical evolution technique (Melike Karta and Ercan C. elik, 2012) to evolve the neural network topology along with the network parameters. The method has been tested with success on a series of data-fitting and classifications problems. In this article the constructed neural network methodology is applied on a series of differential equations while preserving the initial or boundary conditions using penalization. The proposed method does not require the user to enter any information regarding the topology of the network. Also, the new method can be used to solve either ODEs or PDEs and it can be easily parallelized. This idea is similar to the cascade correlation neural networks introduced by Fahlman and Lebiere [13] in which the user is not required to enter any topology in formation. However, the method for selecting the network topology differ since the proposed algorithms a stochastic one. In the proposed method, the advantage of using an evolutionary algorithms that the penalty function (use df or initial or boundary conditions) can be incorporated easily into the training process. Neural Networks have been employed before to solve DAES (Byrne and Ponzi, 1988) as well as eigen value problems (Ercan Celik and Mustafa Bayram, 2004).

Corresponsing author: Wahed, M.E.,
Department of Computer Sciences, Faculty of computers and information, Suez Canal University, Suez Canal, Egypt.

The cases treated in the above mentioned articles were for simple finite or extended to infinity orthogonal box boundaries. However when one deals with realistic problems, as for instance in modeling the human head-neck system (Chuili Sun and Juergen Hahn, 2005) or the flow and mass transfer in chemical vapor deposition reactors (Deuflhard *et al.,* 1987), the DAES cannot be described in terms of simple geometrical shapes, that in turn would have allowed for a simple modeling scheme. In this article we propose a method capable of dealing with such kind of arbitrarily shaped boundaries. As before (Byrne and Ponzi, 1988; Celik and Mustafa Bayram, 2004), our approach is based on the use of feed forward artificial neural networks (ANNs) whose approximation capabilities have been widely acknowledged (Narendra and Parathasarathy, 1990; El-sayed Wahed, 2008). More specifically, the proposed approach is based on the synergy of two feed forward ANNs of different types: a multilayer perception (MLP) as the basic approximation element and a radial basis function (RBF) network for satisfying the BCs, at the selected boundary points. In addition, our approach relies on the availability of efficient software for multidimensional minimization (Hairer and Wanner, 1991) that is used for adjusting the parameters of the networks.

A solution to differential equation problems based on ANNs exhibits several desirable features:

• Differentiable closed analytic form.
• Superior interpolation properties.
• Small number of parameters.
• Implementable on existing specialized hardware (neuron processors).
• Also efficiently implementable on parallel computers.

In the next section we describe the proposed method for neural network, while in Section 2, described solving system of differential equations by using genetic algorithms. Section 3, we discuss implementation procedures of the Pade series method. In section 4 we illustrate Adomian decomposition method. Finally section 5 contains numerical example and we compare our results to analytically known ones and given a conclusion.

Genetic Algorithms

A differential algebraic equation has the form

$$\Psi (y, y', x) = 0, \qquad \text{…….. (1)}$$

With initial conditions (IC)

$$\Psi^* (x_0) = y_0, \qquad \left(\frac{d}{dx}\right) \Psi^*(x_0) = y_1,$$

Where Ψ and Ψ^* are both vector functions for which we assumed sufficient differentiability and the initial values to be consistent, i.e.

$$\Psi (y_0, y_0^1, x_0) = 0 \qquad \text{……..(2)}$$

The above DAE can be converted into a system of ODE as

$$\frac{d\psi_i^*}{dx} = f_i (x, y_i) \ (i = 1, 2, \text{……}, n) \qquad \text{……(3)}$$

With initial condition

$$\Psi_i (0) = A_i$$

Thus the solution of the system of ODE is equivalent to the solution of DAE system.

Method Description

In this section a brief description of the grammatical evolution algorithm is given**.** The main step sof the proposed algorithm are outlined with the steps for the fitness evaluation for the case sof ODEs.

Grammatical Evolution

Grammatical evolution is an evolutionary technique that can produce code in any programming language requiring the grammar of the target language in BNF syntax and some proper fitness function. This technique has been used with success in many

scientific fields such as symbolic regression [13], by replacing non terminal symbols with the right hand of the selected production rule. The selection is performed in two steps:

- We read an element from the chromosome (with value ***V***).
- We select the rule according to the scheme

Rule = *V* mod NR

Where NR is the number of rules for the specific non-terminal symbol. The process of replacing non terminal symbols with the right hand of production rules is continued until either a full program has been generated or the end of chromosome has been reached. In the latter case we can reject the entire chromosome or we can start over (wrapping event) from the first element of the chromosome. If the limit of the wrapping events is reached the chromosome is rejected by assigning to it a large fitness value, which prevents the chromosome to be used in the crossover procedure. In the proposed algorithm the limit of wrapping events wassetto2.As an example of the mapping procedure of the grammatical evolution consider the BNF grammar shown in Fig. 1. The number in parentheses denotes the sequence number of the corresponding production rule to be used in the mapping procedure. Consider the chromosome x=[9,8,7,6,16,10,17,23,8,14]. The step soft he mapping procedure are listed in Table1. The final outcome of these steps is the expression 3+sin(x).

Algorithm decription

The proposed method is baased on an evolutionary algorithm ,a stochastic process whose basis lies in the biological evolution
The grammar of the proposed method

S: : = <expr> (0)

<expr>: : = (<expr> <op> <expr>) (0)

| <func> (<expr>) (1)

| <terminal> (2)

<op> : := + (0)

| - (1)
| * (2)
| / (3)

<func> : := sin (0)
| cos (1)
| exp (2)
| log (3)

<digit >: := 0 (0)
| 1 (1)
| 2 (2)
| 3 (3)
| 4 (4)
| 5 (5)
| 6 (6)
| 7 (7)
| 8 (8)
| 9 (9)

<terminal> : :=<xlist> (0)
| <digitlist>.<digitlist> (1)

<xlist>: :=x1 (0)
| x2 (1)
| x3 (2)

<digitlist>: :=<digit> (0)
| <digit>.<digitlist> (1)

Table 1. An example of the mapping procedure

String	chromosome	operation
<expr >	9, 8, 7,6,16,10,17,23,8,14	9 mod 7 = 2
<func>(<expr>)	8,7,6,16,10,17,23,8,14	8 mod 4 = 0
sin(<expr >)	7,6,16,10,17,23,8,14	7mod 7 = 0
sin(<expr> <op> <expr>)	6, 16,10,17,23,8,14	6 mod 7 = 6
sin(x <op> <expr>)	16,10,17,23,8,14	16 mod 4 = 0
sin(x) + <expr>	10, 17,23,8,14	10 mod 7 = 3
sin(x) + < digit>	17, 23,8,14	17 mod 10=7
sin(x) + 3	23,8,14	

Algorithm along with a penalty function which is used in order to represent the boundary or initial conditions of the ordinary differential equations, the main steps of the algorithm are as follows:

Algorithm
Input : function $f(x)$, lower , upper bound [l b], Number of variable N,ε Repeat Solve the differential equation by [t,sol]=ode45(f,[l b],N) [x , f]= Genetic_algorithm($sol, N, [L\ b]$) $Err = f - fold$ Until $Err < \varepsilon$ Plot the result with exact

Solution of DAES by using neural network

In this approach new feed forward neural network is used to transfer the transfer the trial solution of equation (3) to the neural network solution of (3). The trail solution is expressed as the difference of two terms as below[8].

$(\Psi_i)_a (x) = A_i + xN_i (x_j, h_{ij}),\quad (i, j = 1, 2, \ldots\ldots\ldots, n)$ ……..(4)

The first term satisfies the IC and contains no adjustable parameters. The second term employs a feed forward neural network and parameters h_{ij} correspond to the weights of the neural architecture. Consider a multilayer perception with n input units, one hidden layer with n sigmoidal units and a linear output unit. The extension to the case of more than one hidden layer can be obtained accordingly. For a given input vector, the output of the network is

$$N_{ij} = \sum_{i=1}^{n} (V_l)\sigma(Z_i) \qquad \text{where} \qquad Z_i = \sum_{j=1}^{n} (h_{ij})x_j + u_i$$

W_{ij} denotes the weight from the input unit j to the hidden unit i , V_i denotes the weight from the hidden unit i to the output, u_i denotes bias of the hidden unit i and σ(z) is the sigmoidal transfer function.

The error quantity to be minimized is given by

$$E_r = \sum_{i=1}^{n} \left(\frac{d\psi_a(x_i)}{dx} - f_i\left(x, \psi_a(x_i)\right) \right)^2 \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

The neural network is trained till the error function (5) becomes zero. Whenever E_r becomes zero, the trail solution (4) becomes the neural network solution of the equation (3).

Structure of the FFNN

The architecture consists of n input units, one hidden layer with n sigmoidal units and a linear output. Each neuron produces its output by computing the inner product of its input and its appropriate weight vector. During the training, the weights and biases of the network are iteratively adjusted by Nguyen and Widrow rule. The neural network architecture is given in the Fig. 1 for computing N_{ij}. The neural network algorithm was implemented in MATLAB on a PC, CPU 1.7 GHz for the neuro computing approach to solve DAES.

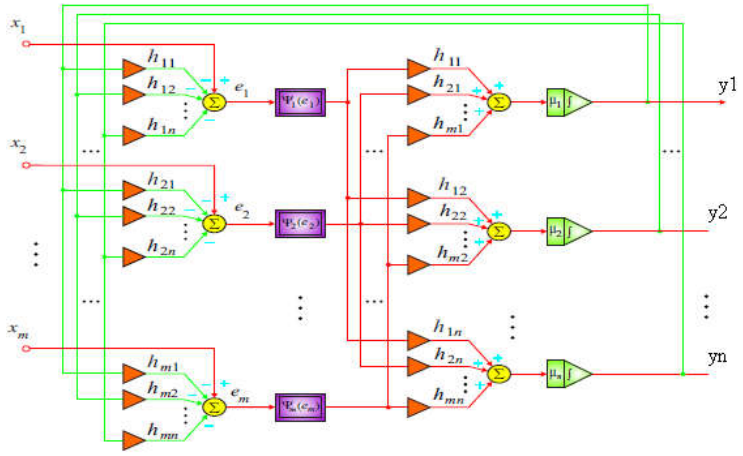


Figure 1. Neural network architecture

Neural network Algorithm

Step 1: Feed the input vector x_j.

Step 2: Initialize randomized weight matrix w_{ij} and bias u_i.

Step 3: Compute $Z_i = \sum_{j=1}^{n} h_{ij}\, x_j + u_i$

Step 4: Pass Z_i into n sigmoidal functions

Step 5: Initialize the weight vector V_i from the hidden unit to output unit.

Step 6: Calculate $N_{ij} = \sum_{i=1}^{n} V_i\, \sigma(Z_i)$

Step 7: Compute purelin function (N_{ij})

Step 8: Repeat the neural network training till he following error function

$$E_r = \sum_{i=1}^{n}\left(\frac{d\psi_a(x_i)}{dx} - f_i\left(x, \psi_a(x_i)\right)\right)^2$$

Description of the Pade Series Method

A system of initial value differential algebraic equations (DAEs) can be written in (1)

The solutions of (1) can be assumed that

$y = y_0 + ex,$ ……(6)

where e is a vector function which is the same size as y_0.

Power series of solution for DAEs

We define another type of Power series in the form

$f(x) = f_0 + f_1x + f_2x^2 + \ldots + (f_n + p_1e_1 + \ldots + p_me_m)x^n$ …… (7)

where $p_1, p_2, \ldots, p_m$ are constants. $e_1, e_2, \ldots, e_m$ are bases of vector e, m is the size of vector e. y is a vector with m elements in (6). Every element can be represented by the Power series in (7).

$y_i = y_{i,0} + y_{i,1}x + y_{i,2}x^2 + \ldots + e_i x^n$. ……..(8)

where y_i is the ith element of y. Substituting (8) into (1), we can get the following:

$f_i = (f_{i,n} + p_{i,1}e_1 + \ldots + p_{i,m}e_m)x^{n-j} + Q(x^{n-j+1})$, ……..(9)

where f_i is the ith element of f (y, y', x) in (1) and j is 0 if f(y, y', x) have y', 1 if do not. From (9) , we can determine the linear equation in (6) as follows:

$A_{i,j} = p_{i,j}$, $B_i = -f_{i,n}$. ……(10)

Solving this linear equation, we have e_i, i = 1,…,m. Substituting e_i into (2.2), we have y_i, i = 1,…,m. which are polynomials of degree n. Repeating this procedure from (7) –(9), we can get the arbitrary order Power series of the solution for DAEs in (1). Let stepsize of x be h and substituting it into the Power series of y and y', we have y and y' at $x = x_0 + h$. If we repeat the above procedure, we have numerical solution of DAEs in (1) (Byrne and Ponzi, 1988; Ercan Celik and Mustafa Bayram, 2004; El-sayed Wahed, 2008).

Pade series

The Power series can be transformed into Pade' series easily. Pade' series is defined in the following:

$$a_0 + a_1x + a_2x^2 + \ldots = \frac{p_0 + p_1x + \ldots + p_M x^M}{1 + q_0 + q_1x + \ldots + q_L x^L} \quad \ldots\ldots (11)$$

Multiply both sides of (11) by the denominator of right-hand side in (11) and compare the coefficients of both sides in (11). We have

$$a_l + \sum_{k=1}^{M} a_{l-k}q_k = p_l \text{ , } l=1,\ldots,M, \quad \ldots\ldots(12)$$

$$a_l + \sum_{k=1}^{L} a_{l-k}q_k = p_l \text{ , } l=M+1,\ldots,M+L, \quad \ldots\ldots(13)$$

Using Adomian Decomposition Method

A system of differential equations can be considered as (1) We can present the system (1), by using the i^{th} equation as:

$Ly_i = f_i (x, y_1, \ldots, y_n)$ $\quad i = 1, 2, \ldots, n$ ……..(14)

where L is the linear operator d/dx with the inverse $L^{-1} = \int_0^x (.)dx$. Applying the inverse operator on (14) we get the following canonical form, which is suitable for applying Adomian decomposition method.

$$y_i = y_i(0) + \int_0^x f_i(x, y_1, \ldots, y_n)\, dx \text{ . } i = 1, 2, \ldots, n \quad \ldots\ldots(15)$$

As usual in Adomian decomposition method the solution of Eq.(15) is considered to be as the sum of a series:

$$y_i = \sum_{j=0}^{\infty} f_{i,j} \quad \ldots\ldots(16)$$

And the integrand in the Eq.(16), as the sum of the following series:

$$f_i(x_i, y_1, \ldots, y_n) = \sum_{j=0}^{\infty} A_{i,j}\left(f_{i,0}, f_{i,1}, \ldots, f_{i,j}\right) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(17)$$

Where $A_{i,j}(f_{i,0}, f_{i,1}, \ldots, f_{i,n})$ are called Adomian polynomials [6]. Substituting (16) and (17) into (15). We get

$$\sum_{j=0}^{\infty} f_{i,j} = y_i(0) + \int_0^x \sum_{j=0}^{\infty} A_{i,j}\left(f_{i,0}, f_{i.1}, \ldots\ldots, f_{i,j}\right)$$

$$= y_i(0) + \sum_{j=0}^{\infty} A_{i,j}\left(f_{i,0}, f_{i.1}, \ldots\ldots, f_{i,j}\right) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(18)$$

From which we define:

$f_{i,0} = y_i(0)$

$$f_{i,n+1} = \int_0^x A_{i,n}\left(f_{i,0}, f_{i,1}, \ldots\ldots, f_{i,n}\right) dx \quad n = 0, 1, 2, \ldots. \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(19)$$

Numerical example

Consider the following differential equation system

$$\begin{pmatrix} 1 & -x & x^2 \\ 0 & 1 & -x \\ 0 & 0 & 0 \end{pmatrix} y' + \begin{pmatrix} 1 & -(x+1) & x^2+2x \\ 0 & -1 & x-1 \\ 0 & 0 & 1 \end{pmatrix} y = \begin{pmatrix} 0 \\ 0 \\ \sin(x) \end{pmatrix}$$

With initial condition $y_1(0)=1$, $y_2(0)=1$ and $y_3(0)=0$

The exact solution is $y_1(x) = e^{-x} + xe^x$, $y_2(x) = e^x + x\sin(x)$ and $y_3(x) = \sin(x)$

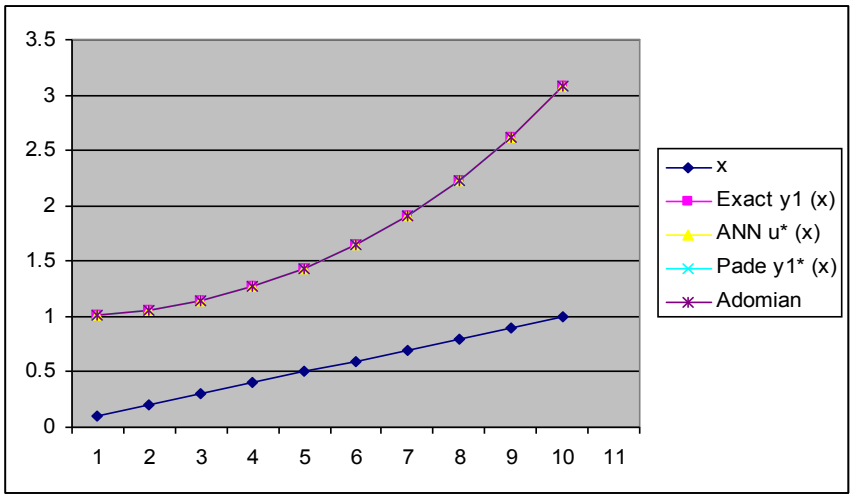
Solution curves using Neural networks. The solution of DAE and the error between the solution by Genetic Algorithm, neural network , Adomian decomposition method and Pade approximation [3] are displayed in Figures 2,3 and 4. The numerical values of the required solution are listed in the Tables 1,2 and 3.

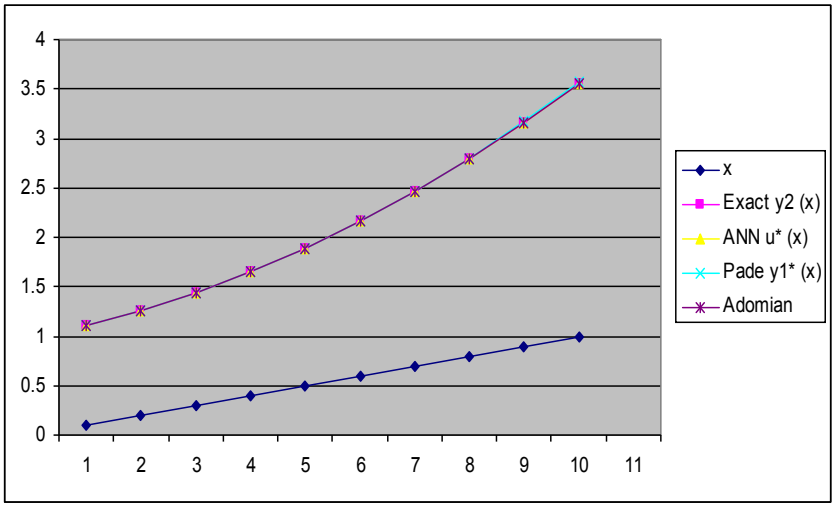
x	Exact $y_1(x)$	ANN $u^*(x)$	$u(x) - u^*(x)$	Pade $y_1^*(x)$	$y_2(x) - y^*_2(x)$	*Adomian*	$y(x) - y_1^*(x)$
0.1	1.015354510	1.015354510	0.000000000	1.01353545	0.00181906	1.015354521	0.11×10^{-7}
0.2	1.063011305	1.063011305	0.000000000	1.0630113	5E-09	1.063011200	0.105×10^{-6}
0.3	1.145775863	1.145775863	0.000000000	1.1457758	6.3E-08	1.145774763	0.11×10^{-5}
0.4	1.267049925	1.267049925	0.000000000	1.2670499	2.5E-08	1.267043625	0.63×10^{-5}
0.5	1.430891295	1.430891295	0.000000000	1.4308912	9.5E-08	1.430866344	0.24951×10^{-4}
0.6	1.642082916	1.642082916	0.000000000	1.6420828	1.16E-07	1.642005198	0.77718×10^{-4}
0.7	1.906212199	1.906212199	0.000000000	1.9062119	2.99E-07	1.906006967	0.205232×10^{-3}
0.8	2.229761707	2.229761707	0.000000000	2.2297606	1.107E-06	2.229281110	0.480596×10^{-3}
0.9	2.620212460	2.620212460	0.000000000	2.6202090	3.46E-06	2.619185426	0.1027034×10^{-2}
1.0	3.086161270	3.086161269	0.000000000	3.0861515	9.77E-06	3.084118661	0.2042608×10^{-2}

$y1^*(x)$ is approximation solution of $y_1(x)$.

x	Exact $y_2(x)$	ANN $u^*(x)$	$u(x) - u^*(x)$		$y_2(x) - y^*_2(x)$	*Adomian Approx.*	$y(x) - y_1^*(x)$
0.1	1.115154260	1.115154260	0.000000000	1.115154260	0.2×10^{-8}	1.115154258	0.2×10^{-8}
0.2	1.261136624	1.261136624	0.000000000	1.261136628	0.92×10^{-7}	1.261136532	0.92×10^{-7}
0.3	1.438514870	1.438514869	0.000000000	1.438515059	0.1061×10^{-5}	1.438513809	0.1061×10^{-5}
0.4	1.647592035	1.647592035	0.000000000	1.647594033	0.6065×10^{-5}	1.647585970	0.6065×10^{-5}
0.5	1.888434040	1.888434040	0.000000000	1.888446092	0.23551×10^{-4}	1.888410489	0.23551×10^{-4}
0.6	2.160904284	2.160904284	0.000000000	2.160957091	0.71650×10^{-4}	2.160832634	0.71650×10^{-4}
0.7	2.464705088	2.464705089	0.000000000	2.464891938	0.184228×10^{-3}	2.464520860	0.184228×10^{-3}
0.8	2.799425801	2.799425801	0.000000000	2.799991522	0.418878×10^{-3}	2.799006923	0.418878×10^{-3}
0.9	3.164597330	3.164597330	0.000000000	3.166116460	0.867168×10^{-3}	3.163730162	0.867168×10^{-3}
1.0	3.559752813	3.559752813	0.000000000	3.563459231	0.1667496×10^{-2}	3.558085317	0.1667496×10^{-2}

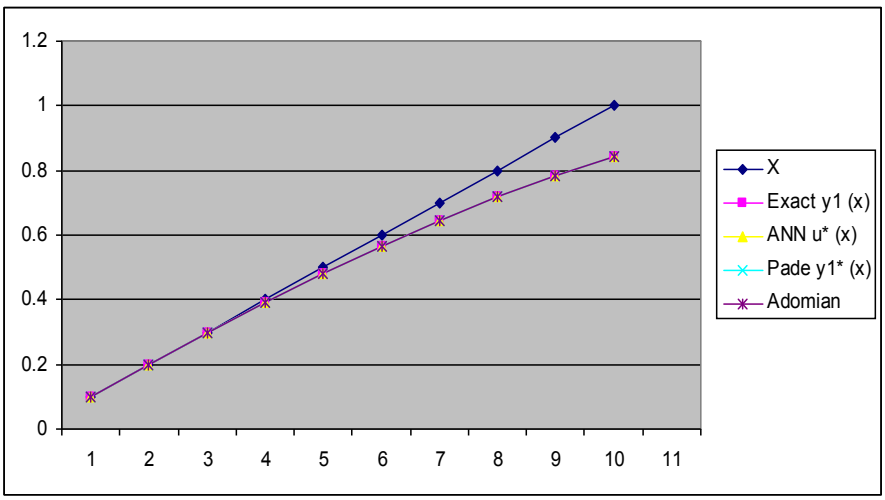
$y_2^*(x)$ is approximation solution of $y_2(x)$





X	Exact y_1 (*x*)	ANN *u** (*x*)	*u* (*x*) −*u**(*x*)	Pade y_1* (*x*)	y_1 (*x*) − y_1*(*x*)	*Adomian*	*Y1(x)-y1*(x)*
0.1	0.099833417	0.099833417	0	0.099833416	-1E-09	0.099833416	$0.2\text{x}10^{-8}$
0.2	0.198669331	0.198669331	0	0.1986693321	1.1E-09	0.198669331	$0.92\text{x}10^{-7}$
0.3	0.295520207	0.295520207	0	0.295520179	-2.8E-08	0.295520205	$0.1061\text{x}10^{-5}$
0.4	0.389418342	0.389418342	0	0.389418721	3.79E-07	0.389418393	0.6065x10−5
0.5	0.479425539	0.479425538	1E-09	0.479429870	4.33E-06	0.479425529	$0.23551\text{x}10^{-4}$
0.6	0.564642473	0.564642474	-1E-09	0.564667618	2.51E-05	0.564642453	$0.71650\text{x}10^{-4}$
0.7	0.644217687	0.644217687	0	0.644323167	0.000105	0.644217657	$0.184228\text{x}10^{-3}$
0.8	0.717356091	0.717356091	0	0.717714654	0.000359	0.717356001	$0.418878\text{x}10^{-3}$
0.9	0.783326910	0.783326910	0	0.784373775	0.001047	0.783325091	$0.867168\text{x}10^{-3}$
1.0	0.841470985	0.841470985	0	0.844190631	0.00272	0.841470098	$0.1667496\text{x}10^{-2}$

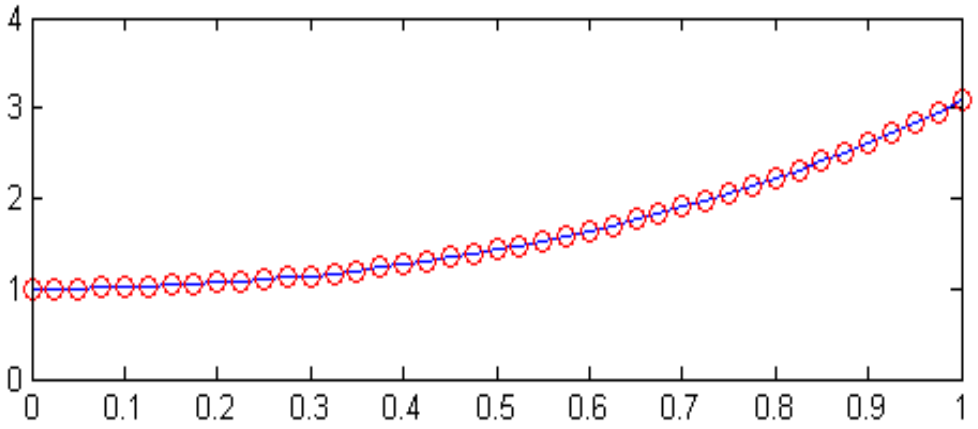
Y_3*(*x*) is approximation solution of y_3(*x*)



Solution of Differential-Algebraic Equations by using Genetic Algorithm

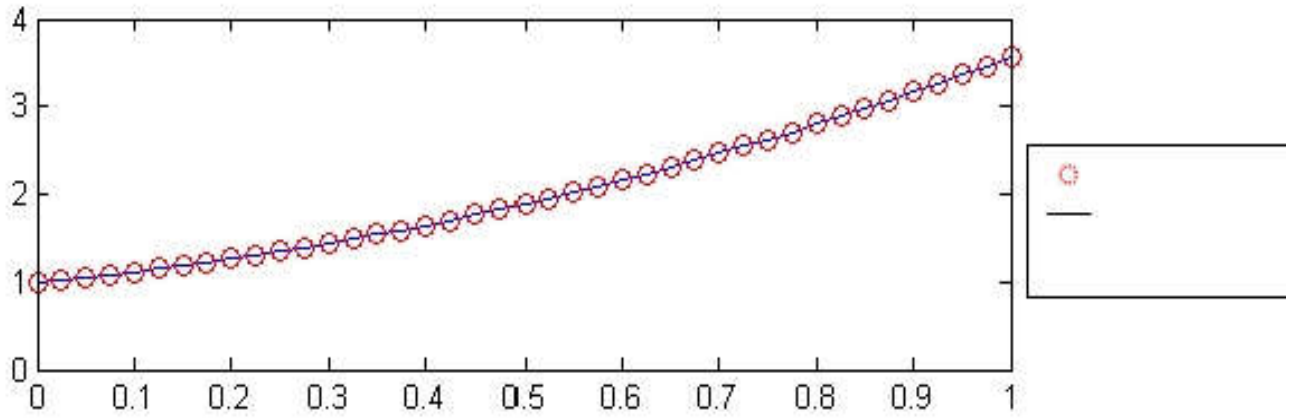
X	App y1by using Genetic	Exact solution	Error 1.0e-07
0	1.0000	1.0000	-0.1000
0.0250	1.0009	1.0009	-0.1000
0.0500	1.0038	1.0038	-0.1000
0.0750	1.0086	1.0086	-0.1000
0.1000	1.0154	1.0154	-0.1000
0.1250	1.0241	1.0241	-0.1000
0.1500	1.0350	1.0350	-0.1000
0.1750	1.0479	1.0479	-0.1000
0.2000	1.0630	1.0630	-0.1000
0.2250	1.0803	1.0803	-0.1000
0.2500	1.0998	1.0998	-0.1000
0.2750	1.1216	1.1216	-0.1000
0.3000	1.1458	1.1458	-0.1000
0.3250	1.1723	1.1723	-0.1000
0.3500	1.2014	1.2014	-0.1000
0.3750	1.2329	1.2329	-0.1000
0.4000	1.2670	1.2670	-0.1000
0.4250	1.3038	1.3038	-0.1000
0.4500	1.3434	1.3434	-0.1000
0.4750	1.3857	1.3857	-0.1000
0.5000	1.4309	1.4309	-0.1000
0.5250	1.4790	1.4790	-0.1000
0.5500	1.5302	1.5302	-0.1000
0.5750	1.5846	1.5846	-0.1000
0.6000	1.6421	1.6421	-0.1000
0.6250	1.7029	1.7029	-0.1000
0.6500	1.7671	1.7671	-0.1000
0.6750	1.8349	1.8349	-0.1000
0.7000	1.9062	1.9062	-0.1000
0.7250	1.9813	1.9813	-0.1000
0.7500	2.0601	2.0601	-0.1000
0.7750	2.1429	2.1429	-0.1000
0.8000	2.2298	2.2298	-0.1000
0.8250	2.3208	2.3208	-0.1000
0.8500	2.4161	2.4161	-0.1000
0.8750	2.5159	2.5159	-0.1000
0.9000	2.6202	2.6202	-0.1000
0.9250	2.7293	2.7293	-0.1000
0.9500	2.8432	2.8432	-0.1000
0.9750	2.9621	2.9621	-0.1000
1.0000	3.0862	3.0862	-0.1000

$y1\ (x)$ is approximation solution of $\mathbf{y}^1$



X	App $y_2$by using Genetic	Exact	Error 1.0e-07
0	1.0000	1.0000	-0.1000
0.0250	1.0259	1.0259	-0.1000
0.0500	1.0538	1.0538	-0.1000
0.0750	1.0835	1.0835	-0.1000
0.1000	1.1152	1.1152	-0.1000
0.1250	1.1487	1.1487	-0.1000
0.1500	1.1842	1.1842	-0.1000
0.1750	1.2217	1.2217	-0.1000
0.2000	1.2611	1.2611	-0.1000
0.2250	1.3025	1.3025	-0.1000
0.2500	1.3459	1.3459	-0.1000
0.2750	1.3912	1.3912	-0.1000
0.3000	1.4385	1.4385	-0.1000
0.3250	1.4878	1.4878	-0.1000
0.3500	1.5391	1.5391	-0.1000
0.3750	1.5923	1.5923	-0.1000
0.4000	1.6476	1.6476	-0.1000
0.4250	1.7048	1.7048	-0.1000
0.4500	1.7640	1.7640	-0.1000
0.4750	1.8252	1.8252	-0.1000
0.5000	1.8884	1.8884	-0.1000
0.5250	1.9536	1.9536	-0.1000
0.5500	2.0207	2.0207	-0.1000
0.5750	2.0898	2.0898	-0.1000
0.6000	2.1609	2.1609	-0.1000
0.6250	2.2339	2.2339	-0.1000
0.6500	2.3089	2.3089	-0.1000
0.6750	2.3858	2.3858	-0.1000
0.7000	2.4647	2.4647	-0.1000
0.7250	2.5455	2.5455	-0.1000
0.7500	2.6282	2.6282	-0.1000
0.7750	2.7129	2.7129	-0.1000
0.8000	2.7994	2.7994	-0.1000
0.8250	2.8879	2.8879	-0.1000
0.8500	2.9782	2.9782	-0.1000
0.8750	3.0705	3.0705	-0.1000
0.9000	3.1646	3.1646	-0.1000
0.9250	3.2606	3.2606	-0.1000
0.9500	3.3585	3.3585	-0.1000
0.9750	3.4582	3.4582	-0.1000
1.0000	3.5598	3.5598	-0.1000

$y_2(x)$ is approximation solution of $y_2(x)$



Conclusion

The solution of DAES can be obtained by Genetic Algorithms, neural network approach. The numerical results of DAES indicate that the neural networks solutions are must more efficient, compared with the solutions of Adomian decomposition method and pade approximation method. A neural computing approach discussed in this paper can yield the best solution of DAE than pade approximation method derived by the authors Celik and Bayram[3]. A numerical example is given to illustrate the derived results. The efficient approximations of the DAES are done in MATLAB on PC, CPU 1.7 GHz.

REFERENCES

Byrne, G.D. and Ponzi, P.R. 1988. "Differential algebraic systems, their application and solution", *Computers Chem, Engng.,* 12(5), pp. 377- 382.

Chuili Sun and Juergen Hahn, 2005. "Reduction of stable differential algebraic equation systems via projections and system identification", *Journal of process control.,* 15, pp. 639- 650.

Deuflhard, P., Hairer, E. and zugek, J. 1987. "One step and extrapolation methods for differential algebraic systems", Numer. Ath., 51, pp. 501- 516.

El-sayed Wahed, M. 2008. " Neural-Network Methods for Irregular Boundaries with Robin Boundary Conditions " *Kuwait J. Sci. Eng.,* 35 (2A), pp. 1-14.

Ercan Celik and Mustafa Bayram, 2004. "Numerical solution of differential algebraic equation systems and applications", *Appl Math. Comput.,* 154, pp. 405- 413.

Ercan Çelik[a*], Mustafa Bayram[b] and Turgut yeloglu[a] " 2006. Solution of Differential-Algebraic Equations(DAEs) by Adomian Decomposition Method" *International Journal Pure & Applied Mathematical Sciences,* Vol.3 No.1, pp. 93-100.

Hairer, E. and Wanner, G. 1991. "Solving ordinary differential equations II: Stiff and Differential Algebraic problems", Springer-Verlag, New York, 1991.

Ioannis G. Tsoulos [a,*] , DimitrisGavrilis[b], EuripidisGlavas[a] " 2009. Solving differential equations with constructed neural networks " Neurocomputing 72, pp. 2385–2391.

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

Melike Karta[1] and Ercan C. elik[2] , 2012." On the Numerical Solution of Differential-Algebraic Equations with Hessenberg Index-3" Hindawi Publishing Corporation Discrete Dynamics in Nature and Society Volume, Article ID 147240, 12 pages

Narendra, K.S. and Parathasarathy, K. 1990. "Identification and control of dynamical system using neural networks", *IEEE* Trans. Neural networks, 1, pp. 4-27.

Susmita Mall and Chakraverty, S. 2013. "Comparison of Artificial Neural Network Architecture in Solving Ordinary Differential Equations " Hindawi Publishing Corporation Advances in Artificial Neural Systems, Volume, Article ID 181895, 12 pages,http://dx.doi.org/10.1155/2013/181895.

****