Journal of AL-Qadisiyah for computer science and mathematics Vol.11 No.1 Year 2019
ISSN (Print): 2074 – 0204 ISSN (Online): 2521 – 3504

Math Page 36 - 44 Khalid.M

Modification Of High Performance Training Algorithms for Solve Singularly Perturbed Volterra integro-differential and integral equation

Khalid. Mindeel. M. Al-Abrahemee

Department of Mathematics , College of Education , University of AL-Qadisiyha
Email: Khalid.mohammed@qu.edu.iq

Recived : 2\10\2018 **Revised : 7\10\2018** **Accepted : 14\10\2018**

Available online : 21/10/2018

DOI: 10.29304/jqcm.2019.11.1.448

Abstract:

In this paper, we apply neural network for solve singularly perturbed Volterra integro-differential equations (SPVIDE) and singularly perturbed Volterra integral equations (SPVIE). Using Modification Of High Performance Training Algorithms such as quase-Newton, Levenberge-Marqaurdt, and Baysian Regulation. The proposed method was compared with the standard training algorithms and analytical methods. We found that the proposed method is characterized by high accuracy in the results, a lower error rate and a speed that is much convergent to standard methods.

Key words: Singularly perturbed Volterra integro-differential equations , Singularly perturbed Volterra integral equations ,Singularly perturbed problems, neural network.

Mathematical subject classification : 34K28

Journal of AL-Qadisiyah for computer science and mathematics Vol.11 No.1 Year 2019
ISSN (Print): 2074 – 0204 ISSN (Online): 2521 – 3504

1. Introduction:

In this paper we consider the numerical discretization of (SPVIDE):

$\varepsilon \mho''(x)\, f\left(\mho, \mho', \varkappa\right) \int_0^x \mathcal{M}(\varkappa, t)\mho(t)dt\,, \varkappa \in [a,b]$ (1)

with boundary condition $\mho(a) = A\,, \mho(b) = B.$

And Volterra integral equations (SPVIE)

$\varepsilon \mho(x) = f(\varkappa) + \int_0^x \mathcal{M}(\varkappa, t)\mho(t)dt \quad , \quad \varkappa \in [a,b]$ (2)

with boundary condition $\mho(a) = A\,, \mho(b) = B$, where ε is a small parameter satisfying $0 \ll \varepsilon < 1$ called perturbation parameter , f and $\mathcal{M}$ are given.

An increasing interest in Voltaire's integrative and differential equations that have small parameters that stimulate this research. In the numerical solution to the problems of the value of the single hyper-limit of ordinary differential equations.

Nonlinear phenomena that appears in many applicationes in scientifec fields, such as fluids dynamics, solid state physics, plasma physics, mathematics biology and chemical, can be modeled by integral equations.. We are frequently faced with the problem of determining the solution of integral equations, one of these integral equations is SPVIE [1].

IN [2] have proposed the HPM for solving the SPVIEs, Alquran and Khair [3] solved the same problem by DTM and VIM. Finally, Dogan et al. [4] used DTM to solve the presented problem. Liao [5],[6],[7] successfully applied the HAM to solve many types of nonlinear problems.

This paper focus on building a new technique by using neural networks to arrive at an approximate solution to the integrated integrative and differential integrality equation. This structure of artificial neural networks (ANN) can calculate the corresponding production of vector inputs. The error function is now limited to the minimum in the selection points. Thus, the proposed ANN uses a training algorithm based on on quasi-Newton, Levenberg-Marquardt, and Bayesian Regulation algorithms used to modify the parameters (weights and biases) to any degree of accuracy required.

2. Modification Of High Performance Training Algorithms:

In this section we will explicate how to modify some of the training algorithms. And to avoid some disadvantage that occurs in LM algorithm, we imply singular value decompation(S V D)of Jacobian matrix ξ and ξ^{-1} if $\xi(w)$ is a rectanguler matrix or singular, then we use SVD of$\xi(w)$. To avoid some disadvantage that occurs in of quasi– Newton algorithm, we explicate Singular Value Decomposition of Hessian matrix Hand H^{-1}, which is the mainly extensively use technique, and specially, it is a high-quality technique for illconditioned problems and to calculate the pseudoinverse ofH and to calculate theminimum error. To over passing the drawbackof BayesianRegulation algorithm, we imply S V D for compute ξ and ξ^{-1}, which is a good quality technique to calculate the pseudoinverse of ξ and to calculate the minimumerror.

3. Description of the Method

In this part we will explicate how our appear be able to use find the approximate solution of equations (1) and (2). To enter y (x) to the converter to be calculated, $\mho_\tau(\varkappa_i, \rho)$Refers to the analytical solution. In the proposed approach, the FFNF experimental solution is used and the parameter p correspond to the W_i and B_i of the neural architecture. We choose a model for a pilot function $\mho_\tau(\varkappa)$ to meet BC requirements. This is acheived by writing it as two parts :

$$\mho_\tau(\varkappa_i, \rho) = \mathrm{A}(\varkappa) + \mathcal{F}(\varkappa, \mathbb{N}(\varkappa, \rho, \varepsilon)) \quad (3)$$

where $\mathbb{N}(\varkappa, \rho, \varepsilon)$ is the output of the neural network with one input vector $\varkappa$.

Journal of AL-Qadisiyah for computer science and mathematics Vol.11 No.1 Year 2019
ISSN (Print): 2074 – 0204 ISSN (Online): 2521 – 3504

4. Illustration of the Method

To show the technique, we will consider the equations ((1) and (2)) ,where x ∈ [0, 1] and the BC $\mho$ (0) = A and $\mho$ (1) = B . The approximate solution can be written:

$$\mho_\tau(\varkappa_i,\rho) = \mathrm{A} + (\mathrm{A}+\mathcal{B})\varkappa + \varkappa(\varkappa - 1)\mathbb{N}(\varkappa,\rho,\varepsilon) \quad (4)\ ,$$

And the error quantety to be minimzed is given by

$$E[p] = \{\varepsilon \frac{d^2\mho_\tau}{d\varkappa^2} \sum_{i=1}^{n} f\left(\mho_i, \mho_t^{'}, \varkappa_i\right) + \int_0^x \mathcal{M}(\varkappa_i,t)\mho_\tau(t)dt\}^2 \ , \quad (5)$$

where the xi's are points in [a, b].

5. Numerical examples

In this section we will present the numerical results of some mathematical models from the various examples of some cases of numerical conditions of the proposed neural network where the structure of the network consists of three layers. . In hidden layers we used sigmoid(logsig) as an activation function that is $\sigma(\varkappa) = \frac{1}{1+e^{-\varkappa}}$. or each test problem, the analytical solution $y_a(x)$was defined in proceed. The accuracy of the approximate solutions can be tested by using the following equation $\Delta\mho(x) = |\mho_t(\varkappa) - \mho_a(x)|$.

In this section, there are three examples of different cases and each example contains four tables.

The first table shows the experimental and approximate solution of the high-level training algorithms with the analytical solution. The second table shows the error of the modified method and the third table represents the accuracy of the proposed method and the number of iterative needed to reach the target.

And the fourth table represents the initial value of W_i *and* B_is of the of the neural network. The figure illustrates the exact and approximation solutions in the training set

Example 1:

We consider the following linear SPVIE [8]

$$\varepsilon y(x) = \int_0^x [1 + t - y(t)\,dt$$ and BC:

y(0)=0 ,y(1)= $y(x) = 2 - e^{-\frac{1}{\varepsilon}} - \varepsilon\left(1 - e^{-\frac{1}{\varepsilon}}\right)$

which has the exact solution $y(x) = x + 1 - e^{-\frac{x}{\varepsilon}} - \varepsilon\left(1 - e^{-\frac{x}{\varepsilon}}\right)$, $\varepsilon = 0.25$.

Example 2:

We consider the following nonlinear SPVIE [9]

$$y(x) = \frac{1}{\varepsilon}(1 - e^x) + \frac{1}{\varepsilon}\int_0^x e^{x-t}\, y^2(t)dt$$

and the boundary conditions y(0)=0 , y(1)=$\frac{2(1-e^{\frac{1}{\varepsilon}(\sqrt{1+4\varepsilon^2})}}{\varepsilon(\frac{1}{\varepsilon}(\sqrt{1+4\varepsilon^2})-1)e^{\frac{1}{\varepsilon}(\sqrt{1+4\varepsilon^2})}+\frac{1}{\varepsilon}(\sqrt{1+4\varepsilon^2})+1}$

which has the exact solution $y(x) = \frac{2(1-e^{\varphi x})}{\varepsilon(\varphi-1)e^{\varphi x}+\varphi+1}$

where the parameters φ are defined as $\varphi = \frac{1}{\varepsilon}(\sqrt{1+4\varepsilon^2})$ and ε=0.75.

Example 3:

In this problem we consider the SPVIDE [10–12]

$\varepsilon\frac{d}{dx}y(x) = (1+\varepsilon)e^{-1} - \varepsilon - y(x) + \int_0^x (1+\varepsilon)y(t)dt$,

with the boundary condition $y(0) = 1 + e^{-1}$, $y(1) = 1 + e^{\frac{-1}{\varepsilon(1+\varepsilon)}}$ and the analytic solution is $y(x) = e^{x-1} + e^{\frac{-x}{\varepsilon(1+\varepsilon)}}$, we get ε =$2^{-10}$.

6. Conclusion

This paper present new technique to solve 2nd singularly perturbed Volterra integro-differential equations (SPVIDE) and singularly perturbed Volterra integral equations (SPVIE). Using artificial neural network which have the singularly perturbed, using modification of high performance training algorethms such as quase-Newton, Levenberge-Marquard, and Baysian Regulation. The projected construction of ANN is more professional and accurate than the other approximat method. convenient outcome with a few hundred wieghts prove that the Levenberge-Marquardt (trainlm) algorethm will have the highest convergence, then trainbfg and trainbr. The performance of different algorethms can be affect by the accuracyrequired for rounding.

References :

[1] M.H. Alnasr. Numerical treatment of singularly perturbed volterra integral equations, Ph.D thesis, Egypt, 1997.

[2] M.H. Alnasr, S. Momani. Application of homotopy perturbation method to singularly perurbed volterra integral equations. J. Appl. Sci, 8(2008): 1073-1078.

[3] M. Alquran, B. Khair. Algorithms to solve singularly perturbed volterra integral equations. Appl. Appl. Math, 6(2011): 2110-2124.

[4] N. Dogan, V. Erturk, S. Momani, O. Akin, A. Yildirim. Differential transform method for solving singularly perturbed volterra integral equations. J. King Saud Univ. Sci, 23(2011): 223-228.

[5] S. J. Liao. Beyond perturbation: Introduction to homotopy analysis method, Chapman & Hall/CRC Press, Boca Raton, 2003.

[6] Hristev, R. M. ," The ANN Book ", Edition 1, 1998.

[7] Yamashita N., Fukushima M., "On the rate of convergence of the Levenberg-Marquardt method", Computing Suppl. J., Vol. 15,(2001), pp: 237–249.

[8] M.H.Alnaser , Shaher Momani, " Application of Homotopy Perturbation Metood to Singularly Perturbed Volterra Integral Equation" 8(6) (2008):1073-1078.

[9] Ahmad Jafarian, Pariya Ghaderi," Application of Homotopy Analysis Method for the Singularly Perturbed Volterra Integral Equations", 7(1) 2013,: 41-54.

[10] V. Horvat and M. Rogina. Tension spline collocation methods for singularly perturbed Volterra integro-differential and Volterra integral equations. J. Comput. Appl. Math., 140(2002):381–402.

[11] A. Salama and S.A. Bakr. Difference schemes of exponential type for singularly perturbed Volterra integrodifferential problems. Appl. Math. Model., 31(2007):866–879.

[12] J.I. Ramos. Piecewise-quasilinearization techniques for singularly perturbed Volterra integro-differential equations.Appl. Math. Comput., 188(2007):1221–1233.

[13] K. Parand , J.A. Rad," An Approximation Algorithm for the Solution of the Singularly Perturbed Volterra Integro-differential and Volterra Integral Equations", Vol.12(2011):430-441.

Table 1: presents a comparison between the exact and approximated solutions of example 1, ϵ =0.25

Input	*Analytic solution*	*Solution of FFNN yt(x) for training algorithm*		
x	*ya(x)*	*Trainlm*	*Trainbfg*	*Trainbr*
0.0	0	8.63846704779123e-06	6.241609519574316-06	3.33127199026609e-06
0.1	0.347259965473271	0.347262822894421	0.347549233468808	0.347241687278199
0.2	0.613003276912084	0.613012683806799	0.612995182065023	0.613047107476904
0.3	0.824104341065849	0.823855506383785	0.824088405150215	0.824059458872570
0.4	0.998577611504008	0.998385206583588	0.998584506284787	0.998578563013621
0.5	1.14848853757254	1.14847942445457	1.14848027740797	1.14853704015194
0.6	1.28196153503294	1.28199362598582	1.28192438254163	1.28193562817729
0.7	1.40439245303109	1.40437009337511	1.40437701556679	1.40428258136672
0.8	1.51942834701623	1.51937926612973	1.51943739391913	1.51933687735538
0.9	1.62950720816453	1.62950369108927	1.62944608756291	1.62951250794293
1.0	1.73626327083345	1.73621733919038	1.73591117829189	1.73626101112071

Table 2: Accuracy of solutions of example 1

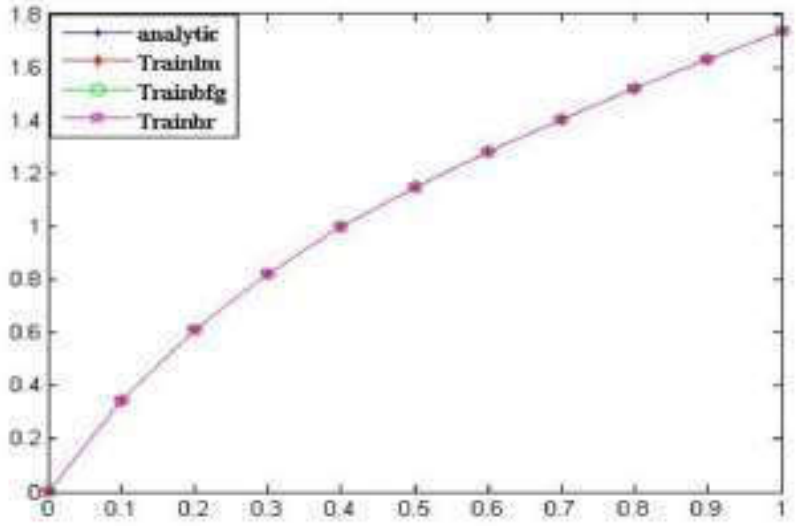
The error E(x)=| yt(x)-ya(x)|where yt(x)computed by the following training algorithm		
Trainlm	*Trainbfg*	*Trainbr*
8.63846704779123e-06	6.241609519574316-06	3.33127199026609e-06
2.85742115052612e-06	0.000289267995537579	1.82781950711641e-05
9.40689471529144e-06	8.09454706110197e-06	4.38305648197135e-05
0.000248834682063381	1.59359156334249e-05	4.48821932786947e-05
0.000192404920420497	6.89478077875450e-06	9.51509612434620e-07
1.91131179658743e-05	1.82601645739577e-05	3.85025794009675e-05
3.20909528779279e-05	3.71524913145294e-05	2.59068556500708e-05
2.23596559758565e-05	1.54374642924449e-05	0.000109871664370598
4.90808864939130e-05	9.04696290393084e-06	9.74696608420089e-05
3.51707526458078e-06	6.11206016205568e-05	5.29977839969220e-06
4.59316430732049e-05	0.000352092541554994	2.25971274048220e-06

Table 3:The accuracy of the train of suggested FFNN.

Train function	*Performance of train*	*Epoch*	*Time*	*Msereg.*
Trainlm	3.09e-33	95	0:00:01	8.634742679417933e-09
Trainbfg	1.60e-15	442	0:00:08	1.7494423987738 95e-08
Trainbr	7.19e-10	301	0:00:04	2.294226818490657e-09

Table4: Initial weight and bias of FFNN

Initial weights and bias for trainlm			*Initial weights and bias for trainbfg*			*Initialweights and bias for trainbr*		
Net.IW {1,1}	Net.LW {2,1}	Net.B{1}	Net.IW {1,1}	Net.LW {2,1}	Net.B{1}	Net.IW {1,1}	Net.LW {2,1}	Net.B{1}
0.2691	0.9831	0.6981	0.7702	0.1759	0.6074	0.5523	0.0495	0.1465
0.4228	0.3015	0.6665	0.3225	0.7218	0.1917	0.6299	0.4896	0.1891
0.5479	0.7011	0.1781	0.7847	0.4735	0.7584	0.0320	0.1925	0.0427
0.9427	0.6663	0.1250	0.4714	0.1527	0.2428	0.6147	0.1231	0.6352
0.4177	0.5391	0.9991	0.0358	0.3411	0.9174	0.3624	0.2055	0.2819



Figure(1) : shows a comparison between the e×act solution and the appro×imate solution of the problem which is presented in e×ample 1 , with ε =0.25.

Table 5: : presents a comparison between the exact and approximated solutions of example 2, ε =0.75

input	Analytic solution	Solution of FFNN $y_a(x)$ for training algorithm		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	0	-1.23296639564785e-09	-2.29553043240571e-10	-6.76384614983760e-09
0.1	-0.129682372351998	-0.129682372416046	-0.129682331724809	-0.129682382846623
0.2	-0.279753006539272	-0.279753006437191	-0.279753106383285	-0.279753010727454
0.3	-0.448074072192466	-0.448074072225477	-0.448074029369055	-0.448074072108653
0.4	-0.630374312628864	-0.630374312671338	-0.630374165443910	-0.630374312666036
0.5	-0.820483778242515	-0.820483778253259	-0.820483734588987	-0.820483779223842
0.6	-1.01107274618158	-1.01107274623763	-1.01107281604895	-1.01107274632745
0.7	-1.19473797678062	-1.19473797682557	-1.19473800796783	-1.19473797555232
0.8	-1.36511190858948	-1.36511190848251	-1.36511196236237	-1.36511190753393
0.9	-1.51765877577841	-1.51765877584827	-1.51765912453648	-1.51765877552818
1.0	-1.64997435974891	-1.64997436156016	-1.64997435305496	-1.64997435715362

Table 6: Accuracy of solutions of example 2, ε =0.75

The error $E(x)=\|yt(x)-ya(x)\|$ where $yt(x)$ computed by the following training algorithm		
Trainlm	*Trainbfg*	*Trainbr*
1.23296639564785e-09	2.29553043240571e-10	6.76384614983760e-09
6.40480168900837e-11	4.06271892816852e-08	1.04946255297111e-08
1.02081509911756e-10	9.98440130017819e-08	4.18818218994588e-09
3.30114824365069e-11	4.28234107063830e-08	8.38128455526999e-11
4.24735802084797e-11	1.47184954446544e-07	3.71715991320798e-11
1.07445163877173e-11	4.36535279035866e-08	9.81327130666898e-10
5.60449464614976e-11	6.98673674470740e-08	1.45869538670240e-10
4.49544845793071e-11	3.11872150327019e-08	1.22829435511562e-09
1.06969100244214e-10	5.37728923610814e-08	1.05554631701921e-09
6.98625601813774e-11	3.48758070423116e-07	2.50229170717375e-10
1.81124848452896e-09	6.69394806251944e-09	2.59529575608042e-09

Table 7:The accuracy of the train of suggested FFNN, ε =0.75

Train function	*Performance of train*	*Epoch*	*Time*	*Msereg.*
Trainlm	5.03e-21	225	0:00:02	3.959872612109448e-19
Trainbfg	1.60e-15	442	0:00:08	1.370004135922276e-14
Trainbr	8.19e-18	286	0:00:03	1.504156882367969e-17

Table 8: Initial weight and bias of FFNN, ε =0.75

Initial weights and bias for trainlm			*Initial weights and bias for trainbfg*			*Initialweights and bias for trainbr*		
Net.IW{1}	*Net.LW{2}*	*Net.B{1}*	*Net.IW{1}*	*Net.LW{2}*	*Net.B{1}*	*Net.IW{1}*	*Net.LW{2}*	*Net.B{1}*
0.1478	0.4674	0.4278	0.4504	0.4470	0.7462	0.5825	0.6714	0.2794
0.0198	0.6567	0.2672	0.4736	0.5876	0.4679	0.6866	0.8372	0.9462
0.9643	0.2902	0.7537	0.9497	0.8776	0.8608	0.7194	0.9715	0.9064
0.9704	0.7545	0.8984	0.0835	0.4691	0.4665	0.6500	0.0569	0.3927
0.1239	0.5581	0.7284	0.2798	0.4374	0.4981	0.7269	0.4503	0.0249

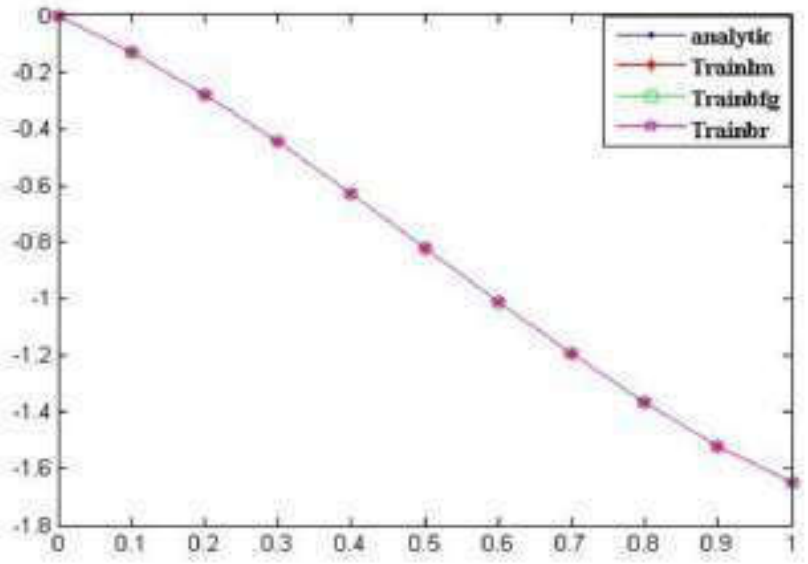


Figure2: shows a comparison between the e×act solution and the appro×imate solution of the problem which is presented in e×ample 2, with ε =0.75.

Table 9: presents a comparison between the e×act and approximated solutions of example 3, $\varepsilon = 10^{-4}$

input	Analytic solution	Solution of FFNN $yt(x)$ for training algorithm		
x	$ya(x)$	Trainlm	Trainbfg	Trainbr
0.0	1.36787944117144	1.36788005912683	1.36658579902393	1.36787348745286
0.1	0.408403141605383	0.408403158192725	0.409648772717205	0.408555958854668
0.2	0.449332325772970	0.447220714806900	0.446917028590653	0.449072090350214
0.3	0.496585309954944	0.496585884132547	0.497291215677891	0.496705926669505
0.4	0.548811636105327	0.549525400628744	0.550290995079382	0.548833632292113
0.5	0.606530659712654	0.606598111288221	0.607065207795657	0.605991033696597
0.6	0.670320046035639	0.670332186873078	0.669343251141215	0.669735307851162
0.7	0.740818220681718	0.740783737266799	0.739191349597689	0.744369056450970
0.8	0.818730753077982	0.818759261704545	0.818267940190760	0.818711070837418
0.9	0.904837418035960	0.904826537817461	0.905932626518221	0.904812403869590
1.0	1	0.999131812534284	0.998764074542181	1.00337320163045

Table 10: Accuracy of solutions of example 3, $\varepsilon = 10^{-4}$

The error $E(x)=\lvert yt(x)-ya(x)\rvert$ where $yt(x)$ computed by the following training algorithm		
Trainlm	Trainbfg	Trainbr
6.479553913418881e-07	0.0012936421474634 1	5.953718585471 89e-06
1.658734138931 81e-08	0.0012456311118 2108	0.000152817249284620
0.002111610966070 54	0.0024152971823 1664	0.000260235422756383
5.741776029632 99e-07	0.0007059057229 46654	0.000120616714560984
0.000713764523417204	0.0014793589740 5427	2.199618678544 46e-05
6.745157556642 40e-05	0.0005345480830 02684	0.000539626016057282
1.214083743827 92e-05	0.0009767948942 4593	0.000584738184477796
3.448341491896 21e-05	0.0016268710840 2837	0.00355083576925175
2.850862656256 23e-05	0.0004628128872 22275	1.968224056381 20e-05
1.088021849871 00e-05	0.0010952084822 6119	2.501416636924 16e-05
0.000868187465716153	0.0012359254578 1870	0.00337320163044530

Table 11:The accuracy of the train of suggested FFNN, $\varepsilon = 10^{-5}$

Train function	*Performance of train*	*Epoch*	*Time*	*Msereg*	*MSE of Numerical Method in [13]*
Trainlm	3.24e-10	369	0:00:06	4.687304918869537e-07	$5.70\varepsilon-003$
Trainbfg	7.84e-8	209	0:00:03	1.519637055012368e-06	
Trainbr	1.20e-22	275	0:00:04	2.023134155861496e-06	

Table 12: Initial weight and bias of FFNN, $\varepsilon = 10^{-6}$

Initial weights and bias for trainlm			*Initial weights and bias for trainbfg*			*Initialweights and bias for trainbr*		
Net. IW[	*Net. LW[*	*Net. B[1*	*Net. IW[*	*Net. LW[*	*Net. B[1*	*Net. IW[*	*Net. LW[*	*Net. B[1*
0.9541	0.1820	0.6427	0.6393	0.4333	0.6240	0.6109	0.4186	0.8051
0.5428	0.0930	0.0014	0.9173	0.8842	0.3279	0.9000	0.1557	0.0672
0.5401	0.4635	0.0304	0.1616	0.3931	0.8030	0.1934	0.8190	0.9508
0.3111	0.0093	0.2085	0.7156	0.1790	0.9995	0.7544	0.6249	0.4976
0.0712	0.9150	0.4550	0.5777	0.6333	0.9810	0.3463	0.7386	0.7551

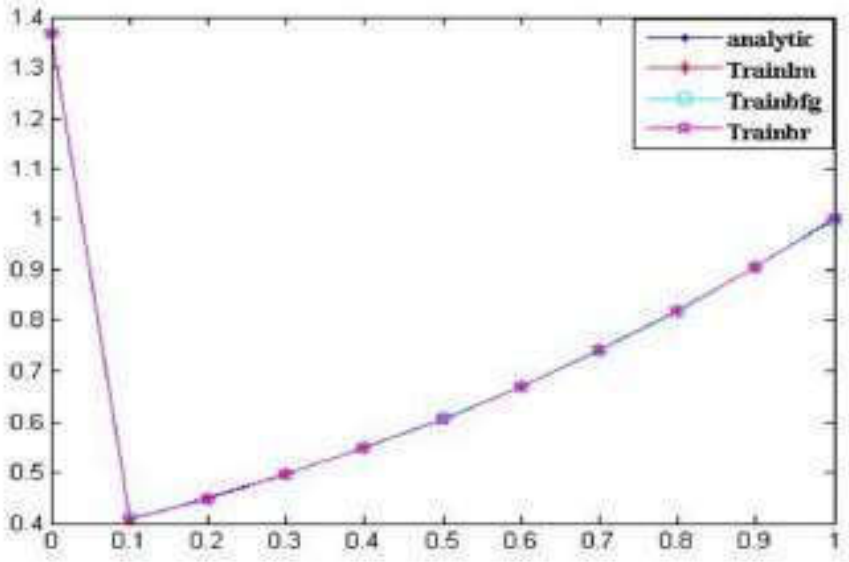


Figure 3.: shows a comparison between the e×act solution and the appro×imate solution of the problem which is presented in e×ample 3, with $\varepsilon = 2^{-6}$.

Journal of AL-Qadisiyah for computer science and mathematics Vol.11 No.1 Year 2019
ISSN (Print): 2074 – 0204 ISSN (Online): 2521 – 3504

خوارزميات التدريب ذات الاداء العالي المحسنة لحل معادلة فولتيرا التكاملية التفاضلية و التكاملية المضطربة الشاذة

خالد منديل محمد
قسم الرياضيات ، كلية التربية ، جامعة القادسية

المستخلص :

في هذا البحث، نستخدم الشبكات العصبية لحل معادلة فولتيرا التكاملية التفاضلية ذات الاضطراب المنفردة ومعادلة فولتيرا التكاملية المنفردة ذات الاضطراب . استخدمنا تحسين خوارزميات التدريب ذات الاداء العالي مثل كوازي نيوتن ، ليفنبيرك - ماركواردت وبايسن ركوليشن . تمت مقارنة الطريقة المقترحة مع خوارزميات التدريب القياسية والحلول التحليلية وجدنا أن الطريقة المقترحة تتميز بالدقة العالية في النتائج ، وانخفاض معدل الخطأ وسرعة التقارب كثيرًا مع الطرق القياسية .