Advanced Mathematical Models & Applications
Vol.4, No.3, 2019, pp.232-242

Jomard Publishing

GRAVITATIONAL SEARCH ALGORITHM SOLUTIONS OF INITIAL VALUE PROBLEMS FOR ORDINARY DIFFERENTIAL EQUATIONS

Korhan Günel*, **İclal Gör**
Department of Mathematics, Faculty of Arts and Sciences, Aydın Adnan Menderes University, Turkey

Abstract. In this study, a trial solution satisfying the initial conditions of a differential equation is defined. Next, a cost function is specified regarding with the trial solution depending on the parameters of a feed-forward neural network. In the training stage of the neural network, Gravitational Search Algorithm (GSA) is used as a global optimization method for minimizing the network cost. The main contributions of our paper are getting the numerical solution of Initial Value Problems (IVPs) without derivative and obtaining the solution at any point in the interval by taking advantages of using GSA and Neural Networks(NNs) together. GSA trains the NNs without derivation knowledge using the inputs obtained by the discretization of the domain. However, NNs produce the numerical solutions at any point of problem domain without depending on the step size, which is used for discretization stage. In numerical implementations of IVPs, the obtained results are discussed comparing with the analytical solution, forward Euler and Runge-Kutta 4 methods.

Keywords: feed-forward neural network, gravitational search algorithm, GSA, global optimization, initial value problems, IVPs.
AMS Subject Classification: 65L05, 68T05, 82C32, 82C80.
Corresponding author: Korhan, Günel, Aydın Adnan Menderes University, Faculty of Arts and Sciences, Department of Mathematics, Turkey, Tel: +90 (256) 218 20 00, *e-mail: kgunel@adu.edu.tr*
Received: 15 August 2019; Revised: 02 September 2019; Accepted: 23 November 2019;
Published: 21 December 2019.

1 Introduction

Differential Equations (DEs) are powerful tools for modeling real world problems facing the scientific and industrial community. However, some differential equations are complex and solutions of them could not be obtained analytically in a continuous space. In this case, the approximate solutions are looked for in an interval, which is discretized into a sequence of real numbers as nodes. Main disadvantages of the numerical methods are giving the discrete solution at the nodes of the interval. Interpolation methods are used to obtain the numerical solution at the points not belonging to the discretaziton set of the interval.

Artificial neural networks (ANNs) are alternative approaches to solve DEs at any points of the search space. In the literature, Lee and Kang (1990) presented the first study about solving DEs with ANNs (Lee & Kang, 1990). They used the finite difference techniques for the discretization of DEs, transformed the equations into a cost function and minimize the cost function with neural minimization algorithm utilizing Hopfield Neural Network. Since then, neural network models are varied for solving both Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs) in different forms with some initial and boundary conditions. In this field, a numerous studies are encountered in last two decades. Some of them are summarized in the following.

Meade and Fernandez (1994) demonstrated that feedforward neural network can solve lin-

ear and nonlinear ODEs using B_1-splines as basis function in ANN (Meade & Fernandez, 1994). Malek and Beidokhti (2006) constructed a hybrid method including feed forward network trained by optimization technique based on Nelder-Mead method in order to solve first and high order ODEs (Malek & Beidokhti, 2006). In their work, they got an approximated solution in the neigbourhood of search space. Li-ying, Hui and Zhe-zhao (2007) proposed a method for solving initial value problems (IVPs) by neural networks based on the cosine basis functions (Li-ying et al., 2007). They compared obtained solutions and results with using Euler and Heun method. Fojdl and Brause (2008) used the error as subjective error based on Total Least Square Error (TLSE) for the solution of ODEs (Fojdl & Brause, 2008).

Some researchers focus on the numerical solution of special types of DEs. Otadi and Mosleh (2011) solved Riccati differential equations by two layer feed forward neural network trained by Gradient Descent Algorithm (Otadi & Mosleh, 2011). Mall and Chakraverty (2014) used Chebyshev Neural Network model for the solution of homogeneous and nonhomogeneous Lane-Emden equations (Mall & Chakraverty, 2014). In their another article, Mall and Chakraverty (2015) solved singular IVPs of Emden-Fowler type equations with Chebyshev Neural Network (Mall & Chakraverty, 2015). Raja et al. (2014) presented articles about the numerical solution of one and two dimensional Bratu-type DEs (Raja & Ahmad, 2014; Raja et al., 2014).

In the literature, some articles are prepared using global optimization techniques for the numerical solutions of special types of DEs. Raja (2014) solved BVPs of the second order Pantography functional DEs using ANNs and optimization techniques as simulated annealing (SA), pattern search (PS), genetic algorithms (GAs), active set (AS) and their combinations (Raja, 2014a). Raja (2014) optimized the ANNs solution of Troesch's problem with particle swarm optimization (PSO), active set (AS) and PSO combined with AS (PSO - AS) optimization methods (Raja, 2014b).

Kumar and Yadav (2011) presented an extensive survey about the solution of differential equations with using Multi-Layer Perceptron (MLP) and Radial Basis Function Networks (RBFN) (Kumar & Yadav, 2011). Rudd, Muro and Ferrari (2014) presented a constrained backpropogation (CPROP) algorithm for the solution of nonlinear elliptic and parabolic Partial Differential Equations (PDEs) (Rudd et al., 2014). Rudd and Ferrari (2015) proposed a constrained integration (CINT) method so as to solve IVPs of PDEs (Rudd & Ferrari, 2015). Zjavka and Pedrycz (2016) constructed general PDEs with polynomial and ANNs for the solution of PDEs (Zjavka & Pedrycz, 2016). Mall and Chakraverty (2016) got the numerical solution of first, second ODEs and systems of first ODEs with ANNs constructed by Legendre polynomials (Mall & Chakraverty, 2016).

The studies including the types of differential equations such as fractional order Pakdaman et al. (2017); Qu (2017), Bagley-Torvik equation Raja et al. (2017), fractional differential equations of variable-order Zúñiga et al. (2017), high-dimensional parabolic partial differential equations Weinan et al. (2017), Backward Stochastic differential equations Weinan et al. (2017) and Navier-Stokes equations Sinchev et al. (2018) are samples that used the neural networks in the numerical solutions of them. The difference from others and the original contribution of our method is to use a population based method as heuristic optimization technique to train the neural network.

In this work, we solved the initial value problems with feed forward neural network trained by a global optimization technique as Gravitational Search Algorithm (GSA). In the next section, GSA is summarized. In the third section, the method based on a trial function depending on a feed forward neural network solution of initial value problems is explained in detail. The fourth section gives the experimental studies on different types of ODEs. The conclusions and further studies are discussed in last section.

2 Gravitational Search Algorithm

Gravitational Search Algorithm (GSA) is a heuristic method which utilizes the theory of Newton physics based on the law of gravity (Rashedi et al., 2009). In the search space, the solution of the problem is investigated. Simply, GSA is a global optimization method based on population. The population comprised agents having mass. The position of an agent shows the approximated solution, and GSA is used to get optimal solution. Agents interact each other with using gravity force and move another position according to their masses.

Now let have a system with agents having mass whose position is defined as given Eq. (1)

$$\mathbf{X}_i = (x_i^1, x_i^2, ..., x_i^d, ..., x_i^n), \text{ for } i = 1, 2, ..., N \tag{1}$$

where x_i^d is the position of i-th agent in the d-th dimension. In the time t, the force of i-th mass to j-th mass is defined as given in Eq. (2).

$$F_{ij}^d = G(t)\frac{M_{pi}(t) \times M_{ai}(t)}{R_{ij}(t) + \epsilon}(X_j^d(t) - X_i^d(t)) \tag{2}$$

where $M_{ai}(t)$ shows the active gravitational mass related to agent j, $M_{pi}(t)$ is the passive gravitational mass for the agent i, $G(t)$ is the gravitational constant at t-th time, ϵ is a small constant and $R_{ij}(t) = \|X_i^d(t), X_j^d(t)\|$ is the Euclidian distance between two agents i and j.

The total force, F_{ij}^d in Eq. (3), is defined as acting on agent i in a dimension d with using randomly generated value in the interval $[0, 1]$ defined as $rand_j$. The total force is composed of d-th forces exerted from other agents.

$$F_{ij}^d = \sum_{j=1, j\neq i}^{N} F_{ij}^d(t) \tag{3}$$

Thus with using the law of motion, the acceleration of the agent i at t-th time in d-th direction a_i^d is calculated in Eq. (4).

$$a_i^d = \frac{F_{ij}^d(t)}{M_{ii}(t)} \tag{4}$$

where $M_{ii}(t)$ is the inertial mass of i-th agent. Hence, the next velocity and position of the agent is calculated with using the acceleration of the agent as given in Eq. (5) and Eq. (6).

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t) \tag{5}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{6}$$

The random value, $rand_j$, is used for giving the randomized characteristic to the search.

The gravitational constant G is initialized at the beginning and it will be reduced. In other words, G is a function of initial value G_0 at the time t in Eq. (7).

$$G(t) = G(G_0, t) \tag{7}$$

Fitness evaluation is used to calculate gravitational and inertia masses. If a mass is heavier, it is more efficient than others. Furthermore, the heavier mass act slowly. The other masses move towards the heaviest one. The gravitational and inertial masses are updated for $M_{ai} = M_{pi} = M_{ii} = M_i$, $i = 1, 2, ..., N$ with using Eq. (8) and Eq. (9).

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \tag{8}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{n} m_j(t)} \tag{9}$$

where $fit_i(t)$ is the fitness value of the agent i at the time t. The best fitness function value, $best(t)$ and the worst fitness function value, $worst(t)$ are calculated for a minimization problem as in Eq. (10) and Eq. (11).

$$best(t) = min_{j\in 1,2,\ldots,N} fit_j(t) \tag{10}$$

$$worst(t) = max_{j\in 1,2,\ldots,N} fit_j(t) \tag{11}$$

Briefly, the method can be summarized as follows. First of all, the initial population is generated randomly. Then the fitness for each agent is evaluated. Gravitational constant, the best and worst value of the fitness function value of the population are calculated. The next step is the calculation mass and acceleration for each agents. Then, the velocities and positions are updated. If the desired fitness value is reached depending on the stopping criterion, it is assumed that the best solution is obtained. Otherwise, the fitness value is evaluated and the other steps of GSA are repeated again until meeting the goal.

3 Numerical Solution of IVPs Using GSA

The initial value problem for the first order differential equation is defined as in Eq. (12).

$$\begin{cases} y'(t) = f(t, y(t)), \\ y(t_0) = y_0 \end{cases} \tag{12}$$

In order to solve this IVP, the trial function approach is used. The trial function is defined as $y_T(t_j, \vec{p}) = y_0 + (t - t_0)N(t_j, \vec{p})$ is the feed forward neural network solution such that $\vec{p} = \vec{p}(\vec{\alpha}, \vec{\beta}, \vec{w})$ is the unknown parameters vector determined by GSA. $\vec{\alpha}, \vec{\beta}, \vec{w} \in \mathbb{R}^m$ where m is the number of neurons in the hidden layer. Then, the solution of neural network $N(t_j, \vec{p})$ is defined as $N(t_j, \vec{p}) = \sum_{i=1}^{m} \alpha_i \sigma(z_i)$ for the neuron $z_i = w_i t_j + \beta_i$ where w_i is the weight and β_i is the bias value for the input t_j for $1 \leq j \leq n$. The activation function is chosen as the sigmoid function defined as $\sigma(z) = \dfrac{1}{1 + \exp(-z)}$. The main reasons of this selection is that the sigmoid function is a bounded, differentiable and real valued function, and it imitates the probability distribution functions. Its first derivate has non-negative values at any point of problem domain. By this way, the neural networks output can be specified the linear combinations of sigmoid functions as a cumulative distribution function. In addition, it transforms the inputs to a bounded range of the interval $[0, 1]$. Thus, the parameters of neural networks can be easily determined at the training stage of neural net. Actually, a wide variety of sigmoid functions, such as logarithmic sigmoid, tangent hyperbolic, tangent sigmoid etc., is preferred as an activation function within different neural net topology based on the problem.

The feed forward neural network produces network error for each input in each iteration. In order to minimize the total error, the network parameters are updated in the training stage. After updating these parameters, the error is propagated over the whole network. In general, the error of the network is calculated as $E = \dfrac{1}{2}\sum_{i=1}^{n}(d_j - y_j)^2$ where d_j is the desired output and y_j is the output of the network for the input t_j. In this work, we use the cost function as $E = \dfrac{1}{n}\sum_{i=1}^{n}\left(\dfrac{\partial y_T}{\partial t_j} - f(t_j, y_T(t_j))\right)^2$ to calculate the Mean Squared Error (MSE) where n is the number of inputs belonging to the training set. To minimize the cost function, we must determine $\dfrac{\partial y_T}{\partial t_j}$ for the first order differential equations.

$$y_T(t_j,\vec{p}) = y_0 + (t-t_0)N(t_j,\vec{p}) \Rightarrow \frac{\partial y_T}{\partial t_j} = N(t_j,\vec{p}) + t_j\frac{\partial N(t_j,\vec{p})}{\partial t_j}$$

where $\dfrac{\partial N(t_j,\vec{p})}{\partial t_j} = \sum_{i=1}^{m} \alpha_i w_i \sigma(z_i)(1-\sigma(z_i))$.

The second order differential equation is given in the following form.

$$\begin{cases} y''(t) = f(t, y(t), y'(t)), \\ y(t_0) = A \\ y'(t_0) = B \end{cases} \tag{13}$$

Then the cost function defined as $E = \dfrac{1}{n}\sum_{i=1}^{n}\left(\dfrac{\partial^2 y_T}{\partial t_j^2} - f\left(t_j, y_T(t_j), \dfrac{\partial y_T}{\partial t_j}\right)\right)^2$ to calculate MSE. In order to get MSE, the second order partial derivatives of the trail function with respect to t_j must be determined as given in the following.

$$\frac{\partial y_T}{\partial t_j} = N(t_j,\vec{p}) + t_j\frac{\partial N(t_j,\vec{p})}{\partial t_j} \Rightarrow \frac{\partial^2 y_T}{\partial^2 t_j} = 2\frac{\partial N(t_j,\vec{p})}{\partial t_j} + t_j\frac{\partial^2 N(t_j,\vec{p})}{\partial t_j^2}$$

where $\dfrac{\partial^2 N(t_j,\vec{p})}{\partial t_j^2} = \sum_{i=1}^{m} \alpha_i w_i \sigma(z_i)(1-\sigma(z_i))(1-2\sigma(z_i))$.

For the solution of higher order initial value problems, the higher order partial derivations of $y_T(t_j)$ and $N(t_j,\vec{p})$ with respect to t_j must be evaluated in similar way. After constructing the cost function, it is tried to minimize using GSA to determine the unknown parameters of the network.

In the next section, we demonstrate how the method works with some examples. In the numerical implementation, we solve different types of linear differential equations as homogenous, nonhomogenous, and second order homogenous.

4 Numerical Implementation

In this section, we give some examples in order to show the numerical solution of initial value problems using feed forward neural network trained by GSA. In the experiments, the elements of the training set are generated with a fix step size, $h > 0$. The step size is halved as $\dfrac{h}{2}$ for the test set construction. In addition, obtained quadrature nodes belonging to the training set are removed from the test set. Only the initial values are added to test set. Furthermore, the neural network gives the solution at any points in the search space after training.

In our experiments, we use the parameters of GSA as shown in Table 1.

Table 1: Parameters in GSA algorithm

Number of agents	20
Number of neurons	5
Lower Bound	-10
Upper Bound	10
Dimension of Search Space	15
Maximum iteration	1000, 3000, 5000

In order to get the performance of our approach, we compare the results with Euler and Runge Kutta 4 (RK4) methods.

Example 1.

$$\begin{cases} y'(x) + \dfrac{y}{x+1} = 0, \quad x \in [2,3], \\ y(2) = 3 \end{cases} \tag{14}$$

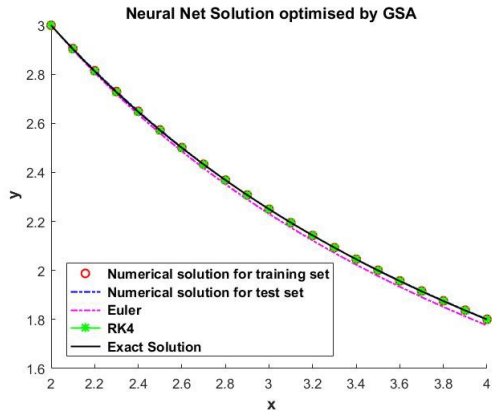

Figure 1: Numerical Solutions for training and test set of Example 1

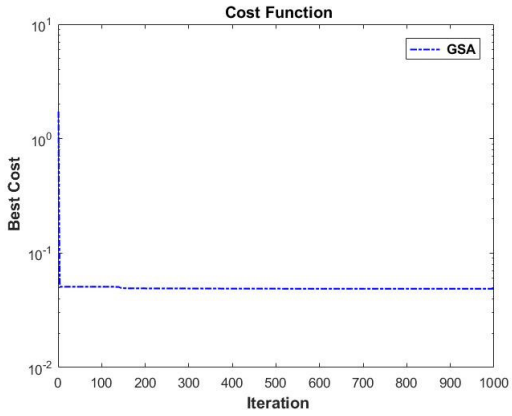

Figure 2: Best cost function value obtained by GSA for Example Example 1

The exact solution of First Order Homogenous Linear Differential Equation given in Eq. (14) is $y = \dfrac{9}{x+1}$. *We get the numerical solution of Eq. (14) illustrated in Fig. 1 after 1000 iterations where the step size chosen as* $h = 0.1$. *Also, the best cost function value obtained by GSA is given in Fig. 2.*

The mean of MSEs with standard deviation for different step size is given for training set in Table 2 and for test set in Table 3.

Table 2: Mean of MSEs for training set for Example 1.

Step size (h)	Number of iteration		
	1000	3000	5000
0.1	$1.574 \times 10^{-5} \pm 3.879 \times 10^{-5}$	$1.498 \times 10^{-6} \pm 1.301 \times 10^{-6}$	$5.191 \times 10^{-5} \pm 1.596 \times 10^{-4}$
0.05	$8.609 \times 10^{-6} \pm 1.082 \times 10^{-5}$	$2.776 \times 10^{-6} \pm 2.877 \times 10^{-6}$	$\mathbf{1.288 \times 10^{-6} \pm 1.677 \times 10^{-6}}$
0.01	$5.475 \times 10^{-5} \pm 8.516 \times 10^{-5}$	$2.355 \times 10^{-5} \pm 7.063 \times 10^{-5}$	$1.687 \times 10^{-6} \pm 1.451 \times 10^{-6}$

Example 2.

$$\begin{cases} y'(x) + \tan y = \cos^2(x), & x \in [0,1], \\ y(0) = 0 \end{cases} \tag{15}$$

The exact solution of First Order Nonhomogenous Linear Differential Equation given in Eq. (15) is
$y = \sin x \cos x$. *After 1000 iterations using the step size* $h = 0.1$, *the numerical solution illustrated in Fig. 3 is obtained. Also, the best cost function value obtained by GSA can be seen in Fig. 4.*

Table 3: Mean of MSEs for test set for Example 1.

Step size (h)	Number of iteration		
	1000	3000	5000
0.1	$1.508 \times 10^{-5} \pm 3.701 \times 10^{-5}$	$1.464 \times 10^{-6} \pm 1.271 \times 10^{-6}$	$4.995 \times 10^{-5} \pm 1.536 \times 10^{-4}$
0.05	$8.454 \times 10^{-6} \pm 1.060 \times 10^{-5}$	$2.720 \times 10^{-6} \pm 2.818 \times 10^{-6}$	$\mathbf{1.262 \times 10^{-6} \pm 1.643 \times 10^{-6}}$
0.01	$5.488 \times 10^{-5} \pm 8.552 \times 10^{-5}$	$2.370 \times 10^{-5} \pm 7.111 \times 10^{-5}$	$1.679 \times 10^{-6} \pm 1.444 \times 10^{-6}$

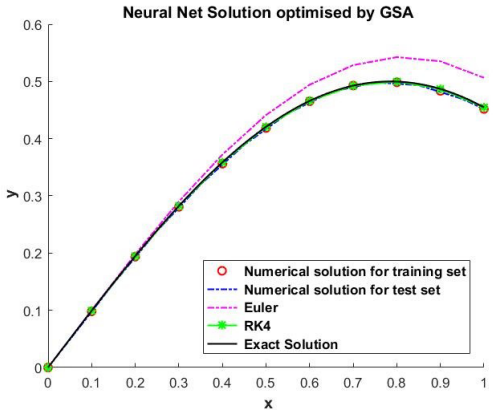

Figure 3: Numerical Solutions for training and test set of Example 2

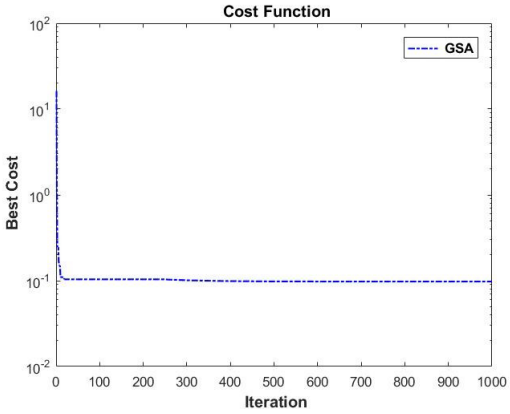

Figure 4: Best cost function value obtained by GSA for Example Example 2

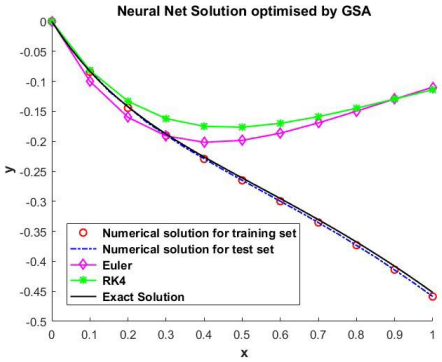

Figure 5: Numerical Solutions for training and test set of Example 3

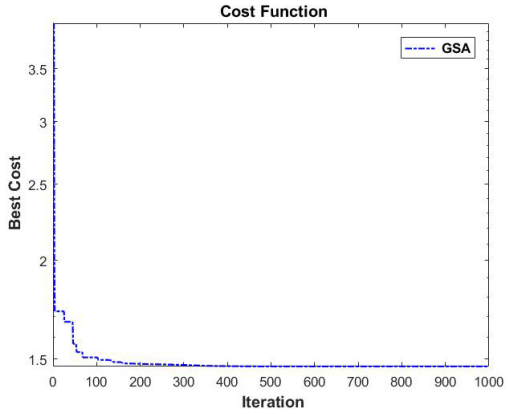

Figure 6: Best cost function value obtained by GSA for Example Example 3

The mean of MSEs with standard deviation with different h values is given in Table 4 and in Table 5 for training and test set, respectively.

Table 4: Mean of MSEs for training set for Example 2.

Step size (h)	Number of iteration 1000	3000	5000
0.1	$1.172 \times 10^{-4} \pm 2.108 \times 10^{-4}$	$7.841 \times 10^{-6} \pm 1.073 \times 10^{-5}$	$1.642 \times 10^{-5} \pm 2.014 \times 10^{-5}$
0.05	$4.694 \times 10^{-5} \pm 9.315 \times 10^{-5}$	$7.038 \times 10^{-6} \pm 9.557 \times 10^{-6}$	$2.246 \times 10^{-5} \pm 4.717 \times 10^{-5}$
0.01	$1.162 \times 10^{-5} \pm 2.718 \times 10^{-5}$	$3.051 \times 10^{-6} \pm 5.374 \times 10^{-6}$	$\mathbf{1.955 \times 10^{-6} \pm 3.249 \times 10^{-6}}$

Table 5: Mean of MSEs for test set for Example 2.

Step size (h)	Number of iteration 1000	3000	5000
0.1	$1.099 \times 10^{-4} \pm 1.961 \times 10^{-4}$	$7.576 \times 10^{-6} \pm 1.054 \times 10^{-5}$	$1.549 \times 10^{-5} \pm 1.878 \times 10^{-5}$
0.05	$4.493 \times 10^{-5} \pm 8.885 \times 10^{-5}$	$6.772 \times 10^{-6} \pm 9.184 \times 10^{-6}$	$2.150 \times 10^{-5} \pm 4.504 \times 10^{-5}$
0.01	$1.151 \times 10^{-5} \pm 2.692 \times 10^{-5}$	$3.022 \times 10^{-6} \pm 5.323 \times 10^{-6}$	$\mathbf{1.936 \times 10^{-6} \pm 3.218 \times 10^{-6}}$

Example 3.

$$\begin{cases} y''(x) + 4y'(x) - 5y(x) = 0, \quad x \in [0,1], \\ y(0) = 1 \\ y'(0) = -1 \end{cases} \tag{16}$$

The exact solution of Second Order Homogenous Linear Differential Equation is

$$y = \frac{1}{6}\left(\exp(-5x) - \exp(x)\right).$$

The numerical solution of Eq. (16) is seen in Fig. 5 after 1000 iterations with step size $h = 0.1$. Furthermore, the best cost function value obtained by GSA is given in Fig. 6.

The mean of MSEs with standard deviation is given in Table 6 for training set and in Table 7 for test set for different step size.

Table 6: Mean of MSEs for training set for Example 3.

Step size (h)	Number of iteration 1000	3000	5000
0.1	$2.133 \times 10^{-5} \pm 2.498 \times 10^{-5}$	$2.958 \times 10^{-6} \pm 4.835 \times 10^{-5}$	$3.695 \times 10^{-6} \pm 4.705 \times 10^{-6}$
0.05	$2.448 \times 10^{-6} \pm 4.040 \times 10^{-6}$	$5.261 \times 10^{-6} \pm 7.000 \times 10^{-6}$	$3.667 \times 10^{-7} \pm 3.070 \times 10^{-7}$
0.01	$1.903 \times 10^{-6} \pm 4.083 \times 10^{-6}$	$1.909 \times 10^{-8} \pm 1.737 \times 10^{-8}$	$\mathbf{5.258 \times 10^{-9} \pm 5.823 \times 10^{-9}}$

Table 7: Mean of MSEs for test set for Example 3.

Step size (h)	Number of iteration 1000	3000	5000
0.1	$2.178 \times 10^{-5} \pm 2.563 \times 10^{-5}$	$3.006 \times 10^{-6} \pm 4.905 \times 10^{-6}$	$3.760 \times 10^{-6} \pm 4.795 \times 10^{-6}$
0.05	$2.488 \times 10^{-6} \pm 4.137 \times 10^{-6}$	$5.330 \times 10^{-6} \pm 7.118 \times 10^{-6}$	$3.717 \times 10^{-7} \pm 3.166 \times 10^{-7}$
0.01	$1.920 \times 10^{-6} \pm 4.121 \times 10^{-6}$	$1.902 \times 10^{-8} \pm 1.729 \times 10^{-8}$	$\mathbf{5.245 \times 10^{-9} \pm 5.795 \times 10^{-7}}$

5 Conclusion

In this paper, we get the numerical solution of IVPs with utilizing feed forward neural network trained by Gravitational Search Algorithm. We solve three types of ODEs including first and second order. In the experiments, for the first order ODEs, the neural network approximated solution is quite close the exact solution. For the second order ODEs, we obtain more acceptable result with proposed approach rather than the other numerical methods as Euler and RK4. In all experiments, we get the decreasing cost function value independent from the step size.

The classical numerical methods as Euler and RK4 gives the solution at the quadrature nodes in the interval. However, neural networks are trained using the quadrature nodes and give the solution at any points in the interval. In other words, the advantage of our approach is to obtain the solution without depending on step size.

For the future work, the different types of differential equations will be solved by feed forward neural networks including nonlinear ODEs. The other work will be about solving linear and nonlinear ODEs with another ANNs method. Furthermore, some global optimization techniques will be investigated for obtaining better solutions.

6 Acknowledgement

The research has been supported by the Council of Higher Education in Turkey (YÖK), Coordination of Academic Member Training Program (ÖYP) in Adnan Menderes University, under Grant no. ADÜ-ÖYP-14011.

References

Weinan, E., Han, J., & Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. C*ommunications in Mathematics and Statistics, 5*(4), 349-380.

Fojdl, J., Brause, R.W. (2008, November). The performance of approximating ordinary differential equations by neural nets. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence* (Vol. 2, pp. 457-464). IEEE.

Kumar, M. & Yadav, N.(2011). Multilayer perceptrons and radial basis function network methods for the solution of differential equations: A survey. *Computer and Mathematics with Applications*, *62*(10), 3796 - 3811.

Lee, H., Kang I.S. (1990). Neural algorithms for solving differential equations, *Journal of Computational Physics*, *91*, 110-131.

Li-ying, X., Hui, W., & Zhe-zhao, Z. (2007, May). The algorithm of neural networks on the initial value problems in ordinary differential equations. *In 2007 2nd IEEE Conference on Industrial Electronics and Applications* (pp. 813-816). IEEE.

Malek, A., Beidokhti, R.S. (2006). Numerical solution for high order differential equations using a hybrid neural network-optimization method. *Applied Mathematics and Computation*, *183*(1), 260-271.

Myneni, S., Patel, V.L. (2010). Chebyshev Neural Network based model for solving Lane-Emden type equations. *Applied Mathematics and Computation* , *247*, 100-114.

Mall, S., Chakraverty S. (2015). Numerical solution of nonlinear singular initial value prob. of Emden-Fowler type using Chebyshev Neural Network method *Neurocomputing*, *149*, 975-982.

Mall, S., Chakraverty, S. (2016). Application of Legendre Neural Network for solving ordinary differential equations, Applied Soft Computing. *International Journal of Information Management*, *43*, 347-356.

Meade, A.J., Fernandez, A.A. (1994). The numerical solution of linear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, *19*(12), 1-25.

Otadi, M., Mosleh, M. (2011). Numerical solution of quadratic Riccati differential equation by neural network. *Mathematical Sciences* , *5*(3), 249-257.

Pakdaman, M., Ahmadian, A., Effati, S., Salahshour, S. & Baleanu, D. (2017). Solving differential equations of fractional order using an optimization technique based on training artificial neural network. *Applied Mathematics and Computation*, *293*, 81-95.

Qu, H. (2017). Cosine Radial Basis Function Neural Networks for Solving Fractional Differential Equations. *Advances in Applied Mathematics and Mechanics*, *9*(3), 667-679.

Raja, M.A.Z. (2014). Numerical treatment for boundary value problems Pantograph functional differential equation using computational intelligence algorithms.*Applied Soft Computing*, *24*, 806 - 821.

Raja, M.A.Z., Ahmad, S. (2014). Numerical treatment for solving one-dimensional Bratu problem using neural networks. *Neural Computing and Applications* , *24*(3), 49-561.

Raja, M.A.Z., Ahmad, S. & Raza, S.(2014). Solution of the 2-dimensional Bratu problem using neural network, swarm intelligence and sequential quadratic programming. *Neural Computing and Applications*, *25*(7-8), 1723-1739.

Raja, M.A.Z. (2014). Stochastic numerical treatment for solving Troesch's problem. *Information Sciences*, *279*(3), 860-873.

Raja, M.A.Z., Samar, R., Manzar, M.A. & Shah, S.M. (2017). Design of unsupervised fractional neural network model optimized with interior point algorithm for solving Bagley-Torvik equation. *International Journal of Information Management*, *132*, 139-158.

Rashedi, E., Nezamabadai-pour, H. & Saryazdi, S., GSA: A Gravitational Search Algorithm, Information Sciences. *Information Sciences*, *179*(13), 2232-2248.

Rudd, K. Muro, G. D. & Ferrari, S. (2014), A Constrained Backpropagation Approach for the Adaptive Solution of Partial Differential Equations. *International Journal of Information Management*, *25*(3), 571-584.

Rudd, K., Ferrari, S. (2015). A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks. *Neurocomputing*, *155*, 277-285.

Sinchev, B., Sibanbayeva, S.E., Mukhanova,A.M., Nurgulzhanova, A.N., Zaurbekov, N.S., Imanbayev, K.S., Gagarina, N.L. & Baibolova,L.K. (2018). Some methods of training radial basis neural networks in solving the Navier-Stokes equations. *International Journal for Numerical Methods in Fluids.*, *86*(10), 625-636.

Zjavka, L., Pedrycz, W. (2016). Constructing general partial differential equations using polynomial and neural networks. *Neural Networks*, *70*, 58-69.

Zúñiga-Aguilar, C.J., Romero-Ugalde, H.M., Gómez-Aguilar, J.F., Escobar-Jiménez, R.F. & Valtierra-Rodríguez, M. (2017) Solving fractional differential equations of variable-order involving operators with Mittag-Leffler kernel using artificial neural networks. *Chaos, Solitons & Fractals.*, *103*, 382-403.