Journal of Advanced Computer Science & Technology, 4 (1) (2015) 180-189
www.sciencepubco.com/index.php/JACST
©Science Publishing Corporation
doi: 10.14419/jacst.v4i1.4365
Research Paper

ANN-based modeling of third order runge kutta method

M. Dehghanpour [1]*, A. Rahati [2], E. Dehghanian [2]

[1] *M.S. Student, in Sistan and Baluchistan University, Zahedan, Iran*
[2] *Assistant Professor in Sistan and Baluchistan University, Zahedan, Iran*
**Corresponding author E-mail: dehghanpour_mahboube@yahoo.com*

Abstract

The world's common rules (Quantum Physics, Electronics, Computational Chemistry and Astronomy) find their normal mathematical explanation in language of differential equations, so finding optimum numerical solution methods for these equations are very important. In this paper, using an artificial neural network (ANN) a numerical approach is designed to solve a specific system of differential equations such that the training process of the ANN calculates the optimal values for the coefficients of third order Runge Kutta method. To validate our approach, we performed some experiments by solving two body problem using coefficients obtained by ANN and also two other well-known coefficients namely Classical and Heun. The results show that the ANN approach has a better performance in compare with two other approaches.

Keywords: *Differential Equations; Artificial Neural Network; Runge Kutta Method.*

1. Introduction

Ordinary differential equations (ODE), especially time-dependent ODEs have many applications in different areas such as astronomy, quantum physics, electronics and computational chemistry [1]. Often, finding an analytical and accurate solution is computationally complicate and time consuming. Therefore, many methods were invented that provide an approximation for solution of the ODEs. Some of these methods that are named after their inventors are: Euler, Taylor and Runge Kutta [2]. The simplest numerical method for solving OEDs is Euler method with the first order accuracy that numerically is not efficient [3]. On the other hand, the Taylor method has direct relationship with its order such that increasing the order requires more successive derivations to be calculated and consequently raises the complexity degree and the computational time [15].
The Runge kutta method in comparison to two other methods has more accuracy and efficiency. To apply this method some coefficients has to be calculated which have many computational ways with different numerical accuracy.
One of the methods in calculating the Runge Kutta coefficients is using artificial neural network (ANN) which is a simplified mathematical model of processing that occurs in biological neurons. One of the most common ANN architectures is Multiple Layer Perceptron (MLP) which often used for prediction problems. These kind of neural networks are mathematical models which map input data to output data.
Research for solving OED and Partial Differential Equations (PDE) by using ANN have been progressed significantly during the last decade. Some of these works are: converting PDE to ODE and solving them as an infinite objective function minimization [4], using Feed Forward Neural Network (FFNN) which can approximate linear and nonlinear OED [5], [6], creating cellular neural network which can approximate the solution of different PDEs [7], using FFNN for solving a special class of the linear first-order PDE[8], using of Radial Basis Function Neural Network (RBFNN) [9] for solving linear differential equations [10], presenting ANN with an embedded finite-element model, for the solution of PDE [11]. In all above mentioned researches, ANN acts as direct solution of differential equations.
In this study, we do not intend to use MLP network for prediction, but we use it as a tool for producing the coefficients of Runge Kutta method. To do so, we consider input and target data as start and end point for each integration step in Runge Kutta method then we design a MLP network in a manner that computing the output of network will simulate the

calculation that is performed in each step of the Runge Kutta method; therefore, the training algorithm will model the optimizing process for finding coefficients of Runge Kutta method.
In the following, the article is organized as follow. Section 2 and 3 describe Runge Kutta method and two-body problem correspondingly. Sections 4 elaborates modeling the process of finding coefficients of Runge Kutta by designing a MLP neural network for this purpose. Experiments and the result are discussed in Section 5. Finally, Section 6 present the conclusion of this work.

2. Runge kutta method

Suppose that $f(t, v)$ is a two-component real function which is defined on domain $D = \{(t, v): a \leq t \leq b, v \in R\}$ and v is a real function which is defined on domain $[a, b]$; then equation of $v' = (t, v)$ is a first order OED problem and each differentiable real function of v on domain $a \leq t \leq b$ which satisfies this equation will be a solution for this problem. Generally, due to a constant number which is created in integrating from $v' = f(t, v)$, the answer of OED is not unique. To unify the answer of problem, the condition $v(t_0) = v_0$ which $x_0 \in [a, b]$ will be added to the equation. The equation with mentioned condition will be named an initial value problem (IVP) and the condition $v(t_0) = v_0$ is called the initial condition [15].
The system of equations of first order OED includes some IVP which must be solved concurrently:

$$v'_1 = f(t, v_1, v_2, \dots, v_m)$$

$$v'_2 = f(t, v_1, v_2, \dots, v_m)$$

$$\vdots$$

$$v'_m = f(t, v_1, v_2, \dots, v_m) \quad (1)$$

Where $D = \{(t, v_1, v_2, \dots, v_m): a \leq t \leq b, v_i \in R\}$, $i = 1, 2, \dots, m$. Then the system of equations of first order OED will be defined as follow:

$$v' = f(t, v) \quad (2)$$

$$t \in [a, b], \qquad f: R \times R^m \to R^m, \qquad v = (v_1, v_2, \dots, v_m)^T$$

Where initial condition is $v(t_0) = (v_1(t_0), v_2(t_0), \dots, v_m(t_0))^T$.
The solution of such problems by Runge Kutta method is as following:

$$v_{i,n+1} = v_{i,n} + h(b_1 k_{1,i} + b_2 k_{2,i}) \quad (3)$$

$$k_{1,i} = f(t_n, v_{1,n}, v_{2,n}, \dots, v_{m,n}) \quad (4)$$

$$k_{2,i} = f(t_n + c_2 h, v_{1,n} + a_{21} h k_{1,1}, v_{2,n} + a_{21} h k_{1,2}, \dots, v_{m,n} + a_{21} h k_{1,m}) \quad (5)$$

$$k_{3,i} = f(t_n + c_3 h, v_{1,n} + a_{31} h k_{1,1} + a_{32} h k_{2,1}, v_{2,n} + a_{31} h k_{1,2} + a_{32} h k_{2,2}, \dots, v_{m,n} + a_{31} h k_{1,m} + a_{32} h k_{2,m}) \quad (6)$$

The equation number (3) approximates the solution of IVPs in t_n which $i = 1, 2, \dots, m$ and $t_{n+1} = t_n + h$, where h is the length of step and $a_{21}, a_{31}, a_{32}, b_1, b_2, b_3, c_2$ are coefficients of third order Runge Kutta method which must satisfy the following conditions [2]:

$$b_1 + b_2 + b_3 = 1, b_2 c_2 + b_3 c_3 = \frac{1}{2}, c_2 b_3 a_{32} = \frac{1}{6}, {c_2}^2 b_2 + {c_3}^2 b_3 = \frac{1}{3}, c_2 = a_{21}, c_3 = a_{31} + a_{32} \quad (7)$$

The conditions $a_{21} = c_2$ and $c_3 = a_{31} + a_{32}$ do not need direct satisfaction. Typically the Runge Kutta coefficients are showed as the following table which is named Butcher table:

Table 1: Representing the Runge Kutta Coefficients as Butcher Table

c_2	a_{21}		
c_3	a_{31}	a_{32}	
	b_1	b_2	b_3

The number iteration for solving IVP problem by Runge Kutta's is given by:

$$N = \frac{t_e - t_s}{h} \quad (8)$$

Where t_s and t_e are the beginning and the end of integration interval, respectively; h is the length of step and N is the iteration number of Runge Kutta algorithm [12].

3. Two-body problem

The two-body problem is related to two-body motion which they interact just with each other. Among these problems we can mention rotation of a planet around the sun, rotation of a satellite around a planet, two stars rotation around each other (double star) and the classical analysis for rotation of an electron around atom. To solve such problems, some differential equations are created that their solutions identify motion path of two-body.
System of equations for two-body problem studied in this study is as follow:

$$x'' = -\frac{x}{\sqrt{x^2+y^2}^3}\ ,\ y'' = -\frac{y}{\sqrt{x^2+y^2}^3} \quad (9)$$

In analytical solution variables x, y, x' and y' are substituted as follow:

$$x = cos(t),\quad y = sin(t),\ \ x' = -sin(t)\ ,\ y' = cos(t) \quad (10)$$

This system is second order OED and in general the second order OED is expressed as $v'' = f(t, v, v')$. Since the Runge Kutta method is just able to solve the first order OED, by choosing $v_1 = v$ and $v_2 = v'$, the second order OED can be expressed as $v_1'' = f(t, v_1, v_2)$ and by changing the variable of $v_1' = v_2$, we can convert the second order OED to two first orders [2]:

$$v_1' = v_2$$

$$(v_2)' = f(t, v_1, v_2) \quad (11)$$

In other words:

$$v'' = (v')' = f(t, v) = [v_1', v_2']^T \quad , v = [v_1, v_2]^T \quad (12)$$

The second order OED which is related to the system of equations of two-body problem is defined as follow:

$$v''(t) = f(t, v_1, v_2, v_3, v_4),\ \ v\ = [x, y, x', y'] \quad (13)$$

Therefore, the first order OED will be as follow:

$$v'(t) = f(t, v) = [v_1', v_2', v_3', v_4']^T = [x', y', x'', y'']^T \quad (14)$$

4. Artificial Neural Network

The artificial neural network is a data processing system and a very simplified mathematical model of human's brain. This model is constituted of a network of many small processors named neuron that are connected together. The neurons are able to operate in parallel. They acquire knowledge through a learning process called training and transfer it by synapses (electromagnetic connections) to solve a problem.
There are varieties of ANNs depending on their structure and their training algorithm. The most popular ANNs is MLP (figure 1) which consists of different layers such as, input layer, one or several hidden layer and output layer. This kind of network is in the group of FFNN which direction of all connections is toward the output of network and they do not have feedback connection. In other words, each layer has some connections to the next layer. Each of these connections has a weight (w) which determines the importance of those connections in calculation of output computation. The hidden and output layers can have some connections of some neurons called Bias (b) with a fixed weight 1. In each neuron, inputs are mapped to outputs by a transfer function (Y). In this mapping, operations $u = \sum_{i=1}^{n} w_i x_i + b$ and $Y = f(u)$ are done by two network input function (u) and activation function (f).

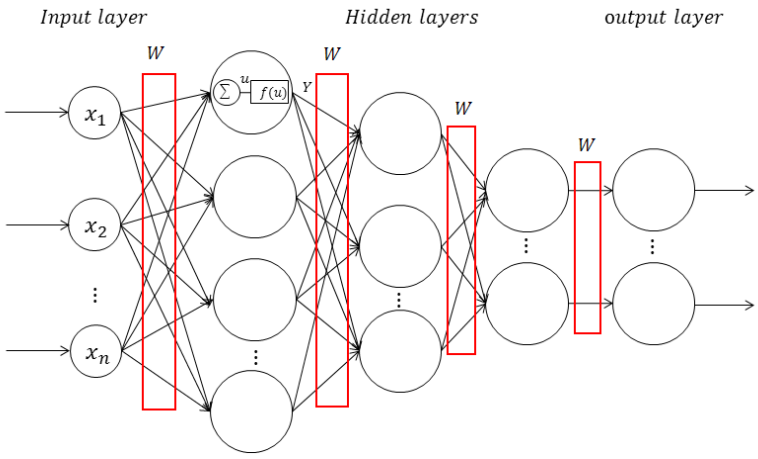

Fig. 1: Multi-Layer Perceptron (MLP) Neural Network

The ANN training means adjusting weights for producing outputs as close as possible to the targets for a specific set of input data. As mentioned, training algorithms are either supervised or unsupervised. In supervised training, in addition to input data there are some target data and the network task is to map the input data to a specific set of the target data. Unsupervised training is appropriate when there is no information about the target data.
Back propagation algorithms are the most popular training techniques for MLP [13]. One of these algorithms is gradient descent with momentum and adaptive learning rate (GDX). Given the momentum, GDX is able to escape of local minimum; this method has a faster convergence rate and shorter training time in comparison with other rival methods [14]. The training algorithms use an estimation of error in a function named performance function which estimates difference between target and network output during the training. In GDX, the first order derivation of performance function with respect to the weight variables is used for adjusting these variables and training the network. In other words, if w_{n+1} shows the weights of network in $n+1$ iteration the network's training, then the value of this weight will be computed as follow:

$$w_{n+1} = w_n + mc \times lr \times \frac{d(perf_{n-1})}{dW} + lr \times (1 - mc) \times \frac{d(perf_n)}{dW} \quad (15)$$

Where mc is the momentum factor, lr is the learning rate and $perf_n$ is the performance function in n-th iteration.
In each iteration, the performance function is computed before and after updating the weights and if the value of performance function after updating will be smaller than before updating, then the learning rate of the network will increase and if the value of performance function after updating divided by performance function before updating is larger than a threshold value (a predetermined value which defines the neurons' activities), then the network learning rate will increase [14].

4.1. Modeling runge kutta coefficients calculation by MLP

As mentioned before, MLP neural networks are used for solving prediction problems. To train these networks, weights are adjusted in a manner that difference between the outputs of the network and targets will be minimized. Therefore, in computing the Runge Kutta coefficients by using MLP neural network, input and target data are considered starting point and end point, respectively in each step of integration (v) and the values of analytical solution which are achieved by equations (10), are used as starting point for calculating each step.
The Runge Kutta will solve the system of third order equations (14) as follow:

$$k_{1r} = h\, f_r\left(t_n, x_n, y_n, {x'}_n, {y'}_n\right) \quad (16)$$

$$k_{2,r} = f_r(\, t_n + c_2 h\,, x_n + ha_{21}k_{1x}, y_n + ha_{21}k_{1y}\;\;, {x'}_n + ha_{21}k_{1x'}, {y'}_n + ha_{21}k_{1y'}) \quad (17)$$

$$k_{3,r} = f_r(\, t_n + c_3 h\,, x_n + ha_{31}k_{1x} + ha_{32}k_{2x}, y_n + ha_{31}k_{1y} + ha_{32}k_{2y}$$
$$, {x'}_n + ha_{31}k_{1x'} + ha_{32}k_{2x'}, {y'}_n + ha_{31}k_{1y'} + ha_{31}k_{2y'}) \quad (18)$$

$$v_{n+1} = v_n + h\sum_{i=1}^{3} b_i k_{i,r} \quad (19)$$

Where $v_n = [x_n, y_n, {x'}_n, {y'}_n]^T$ and $r = x, y, x', y'$.
Given equations (9), (10) and (14):

$$k_{1x} = h\, f_x\left(t_n, x_n, y_n, x'_n, y'_n\right) = x'_n \tag{20}$$

$$k_{1y} = f_y\left(t_n, x_n, y_n, x'_n, y'_n\right) = y'_n \tag{21}$$

$$k_{1x'} = f_{x'}\left(t_n, x_n, y_n, x'_n, y'_n\right) = x''_n = \frac{-x_n}{\sqrt{{x_n}^2+{y_n}^2}^3} \tag{22}$$

$$k_{1y'} = f_{y'}\left(t_n, x_n, y_n, x'_n, y'_n\right) = y''_n = \frac{-y_n}{\sqrt{{x_n}^2+{y_n}^2}^3} \tag{23}$$

$$k_{2x} = h\, f_x\left(t_n + c_2h\,, x_n + ha_{21}k_{1x}, y_n + ha_{21}k_{1y}, x'_n + ha_{21}k_{1x'}, y'_n + ha_{21}k_{1y'}\right) = x'_n + ha_{21}k_{1x} \tag{24}$$

$$k_{2y} = h\, f_y\left(t_n + c_2h\,, x_n + ha_{21}k_{1x}, y_n + ha_{21}k_{1y}, x'_n + ha_{21}k_{1x'}, y'_n + ha_{21}k_{1y'}\right) = y'_j + ha_{21}k_{1y} \tag{25}$$

$$\begin{aligned} k_{2x'} &= h\, f_{x'}\left(t_n + c_2h\,, x_n + ha_{21}k_{1x}, y_n + ha_{21}k_{1y}, x'_n + ha_{21}k_{1x'}, y'_n + ha_{21}k_{1y'}\right) \\ &= \frac{-(x_n+ha_{21}k_{1x'})}{\sqrt{(x_n+ha_{21}k_{1x'})^2+(y_n+ha_{21}k_{1y'})^2}^3} \end{aligned} \tag{26}$$

$$\begin{aligned} k_{2y'} &= h\, f_{y'}\left(t_n + c_2h\,, x_n + ha_{21}k_{1x}, y_n + ha_{21}k_{1y}, x'_n + ha_{21}k_{1x'}, y'_n + ha_{21}k_{1y'}\right) \\ &= \frac{-(y_n+ha_{21}k_{1y'})}{\sqrt{(x_n+ha_{21}k_{1x'})^2+(y_n+ha_{21}k_{1y'})^2}^3} \end{aligned} \tag{27}$$

$$\begin{aligned} k_{3x} &= h\, f_x(\, t_n + c_3h\,, x_n + ha_{31}k_{1x} + ha_{32}k_{2x}, y_n + ha_{31}k_{1y} + ha_{32}k_{2y} \\ &\quad , x'_n + ha_{31}k_{1x'} + ha_{32}k_{2x'}, y'_n + ha_{31}k_{1y'} + ha_{31}k_{2y'}) \\ &= x'_n + ha_{31}k_{1x} + ha_{32}k_{2x} \end{aligned} \tag{28}$$

$$\begin{aligned} k_{3y} &= h\, f_y(\, t_n + c_3h\,, x_n + ha_{31}k_{1x} + ha_{32}k_{2x}, y_n + ha_{31}k_{1y} + ha_{32}k_{2y} \\ &\quad , x'_n + ha_{31}k_{1x'} + ha_{32}k_{2x'}, y'_n + ha_{31}k_{1y'} + ha_{31}k_{2y'}) \\ &= y'_n + ha_{31}k_{1y} + ha_{31}k_{2y} \end{aligned} \tag{29}$$

$$\begin{aligned} k_{3x'} &= h\, f_{x'}(\, t_n + c_3h\,, x_n + ha_{31}k_{1\mathrm{x}} + ha_{32}k_{2x}, y_n + ha_{31}k_{1y} + ha_{32}k_{2y} \\ &\quad , x'_n + ha_{31}k_{1x'} + ha_{32}k_{2x'}, y'_n + ha_{31}k_{1y'} + ha_{31}k_{2y'}) \\ &= \frac{-(x_n+ha_{31}k_{1x'}+ha_{32}k_{2x'})}{\sqrt{(x_n+ha_{31}k_{1x'}+ha_{32}k_{2x'})^2+(y_n+ha_{31}k_{1y'}+ha_{32}k_{2y'})^2}^3} \end{aligned} \tag{30}$$

$$\begin{aligned} k_{3y'} &= h\, f_{y'}(\, t_n + c_3h\,, x_n + ha_{31}k_{1x} + ha_{32}k_{2x}, y_n + ha_{31}k_{1y} + ha_{32}k_{2y} \\ &\quad , x'_n + ha_{31}k_{1x'} + ha_{32}k_{2x'}, y'_n + ha_{31}k_{1y'} + ha_{31}k_{2y'}) \\ &= \\ &\frac{-(y_n+ha_{31}k_{1y'}+ha_{32}k_{2y'})}{\sqrt{(x_n+ha_{31}k_{1x'}+ha_{32}k_{2x'})^2+(y_n+ha_{31}k_{1y'}+ha_{32}k_{2y'})^2}^3} \end{aligned} \tag{31}$$

Assuming equation (19) we have:

$$x_{n+1} = x_n + h(b_1k_{1x} + b_2k_{2x} + b_3k_{3x}) \tag{32}$$

$$y_{n+1} = y_n + h(b_1k_{1y} + b_2k_{2y} + b_3k_{3y}) \tag{33}$$

$$x'_{n+1} = x'_n + h(b_1k_{1x'} + b_2k_{2x'} + b_3k_{3x'}) \tag{34}$$

$$y'_{n+1} = y'_n + h(b_1k_{1y'} + b_2k_{2y'} + b_3k_{3y'}) \tag{35}$$

Therefore, the solution of Runge Kutta for two-body problem will be given by equations (32) to (35). In each step of Runge Kutta method, values of x_n, y_n, x'_n, y'_n,h, a_{21}, a_{31}, a_{32}, b_1, b_2, b_3 must be clear. a_{21}, a_{31}, a_{32}, b_1, $b_2$and $b_3$are the Runge Kutta coefficients which their values are determined by the weights of network. Therefore, in each stepx_n, y_n, x'_n, y'_n and h along with bias input 1 must be given to neural network as input data. As mentioned before, input data in each step are analytical solution amounts; therefore, network input data will be equal to a matrix with $6 \times N$ dimensions where each column is calculated as below:

$$[x_n, y_n, x'_n, y'_n, h, 1] = [cos\ t_n\ , sin\ t_n, -sin\ t_n, cos\ t_n\ , h, 1] \quad (36)$$

On the other hand, based on the equations (32) to (35), output of Runge Kutta is equal to x_{n+1} ،y_{n+1} ،x'_{n+1} and y'_{n+1}, therefore, the target data corresponding to them will be a matrix with$4 \times N$ dimensions where each column is expressed as follow:

$$[x_{n+1}, y_{n+1}, x'_{n+1}, y'_{n+1}] = [cos\ t_{n+1}\ , sin\ t_{n+1}, -sin\ t_{n+1}, cos\ t_{n+1}] \quad (37)$$

where x and y are coordination of body, x' and y' are their derivations with respect to independent variable t, h is the step length, 1 is the artificial input with scalar value and $t_{n+1} = t_n + h, n = 1 \dots N$ and N is iteration number of the Runge Kutta algorithm.
In this method, at first, we identify the integration interval $[t_s, t_e]$, value of x_0, y_0, x'_0 ،$y'_0(v_0)$ and the scalar step length, then we will calculate total number of steps for Runge Kutta method by formula number (8) and we will construct the real input and real output matrices.

4.2. The third order runge kutta network structure design

Based on the equations of (16) to (19), we need to calculate the values of $k_{1x'}$, $k_{1y'}$ ، k_{2x} ، k_{2y} ، $k_{2x'}$ ، $k_{2y'}$, k_{3x} ،k_{3y} ، $k_{3x'}$ ، $k_{3y'}$ ، x_{j+1} ، y_{j+1}،x'_{j+1}،y'_{j+1} for solving two-body problem. Therefore, to model this problem by a neural network, we need 14 layers for calculating these parameters and also 6 layers for calculating the values of ha_{21}, ha_{31}, ha_{32}, b_1, b_2, b_3.
The MLP which is designed for the third order Runge Kutta method has 1 input layer, 16 hidden layers and 4 output layers as you can be seen in Figure 2. The neurons of hidden and output layers have specific input function and identity activating function ($Y = f(u) = u$). Input functions are listed in Table 2. Tables 3 and 4 which depict connections between input layer and hidden layers and also between hidden layers and output layers, respectively. The symbol □, in these tables indicates a connection between mentioned layer in its row and column. Most of the connections have a constant weight value 1 that do not take part in training. The 16 hidden layers are connected to inputs and other layers such that they model the operations expressed by formulas (20) to (30). Number of 4 output layers were created corresponding to formulas (32) to (35). There are some connections between output and hidden layers which have adjustable weights and take part in training and are related to the Runge Kutta's coefficients. Therefore, instead of the standard performance function, we use a performance function indicated by following formula:

$$\|Output - Target\|_\infty + |b_1 + b_2 + b_3 - 1| + \left|c_2 b_2 + c_3 b_3 - \frac{1}{2}\right| + \left|c_2 b_3 a_{32} - \frac{1}{6}\right| + \left|c_2^2 b_2 + c_3^2 b_3 - \frac{1}{3}\right| \quad (38)$$

In this function conditions related to the Runge Kutta coefficients expressed by equation (7) is included.
ANN training models the iteration of Ruge Kutta method which calculates the numerical solution of IVP in each step of Runge Kutta and the values of analytical solution are used as the starting point for calculating each step.

Table 2: The Input Functions of ANN Layers in the Third Order Runge Kutta

Layer's number	Layer's type	Input function
1	Hidden	$f_{x'}(x_n, y_n) = -\frac{x_n}{(\sqrt{x_n^2 + y_n^2}\ )^3}$
2	Hidden	$f_{y'}(x_n, y_n) = -\frac{y_n}{(\sqrt{x_n^2 + y_n^2}\ )^3}$
3,4,5,6,7,8	Hidden	$\sum$
9	Hidden	$f_{2x'} = f_{x'}(x_n + ha_{21}k_{1x'}, y_n + h\ a_{21}k_{1y'})$
10	Hidden	$f_{2y'} = f_{y'}(x_n + h\ \ a_{21}k_{1x'}, y_n + h\ a_{21}k_{1y'})$
11	Hidden	$f_2 = x'_n\ + h\ a_{21}k_{1x} = x'_n\,.(1 + ha_{21})$
12	Hidden	$f_2 = y'_n\ + h\ a_{21}k_{1y} = y'_n\,.(1 + ha_{21})$
13	Hidden	$f_{3x'} = f_{x'}(x_n + ha_{31}k_{1x'} + ha_{32}k_{2x'}, y_n + h\ a_{31}k_{1y'} + ha_{32}k_{2y'})$
14	Hidden	$f_{3y'} = f_{y'}(x_n + ha_{31}k_{1x'} + ha_{32}k_{2x'}, y_n + h\ a_{31}k_{1y'} + ha_{32}k_{2y'})$
15	Hidden	$f_3 = x'_n\ + h\ a_{31}k_{1x} + h\ a_{32}k_{2x}$
16	Hidden	$f_3 = y'_n\ + h\ a_{31}k_{1y} + h\ a_{32}k_{2y}$
17	Hidden	$f_4 = x_n\ + h.(b_1k_{1x} + b_2k_{2x} + b_3k_{3x})$
18	Hidden	$f_4 = y_n\ + h.(b_1k_{1y} + b_2k_{2y} + b_3k_{3y})$
19	Hidden	$f_4 = x'_n\ + h.(b_1k_{1x'} + b_2k_{2x'} + b_3k_{3x'})$
20	Hidden	$f_4 = y'_n\ + h.(b_1k_{1y'} + b_2k_{2y'} + b_3k_{3y'})$

Table 3: The Connections between Input Layer and the Third Order Runge Kutta ANN Hidden and Output Layers

$\frac{Input}{Layer}$	IN1 (x_n)	$IN2$ (y_n)	$IN3$ (x'_n)	$IN4$ (y'_n)	$IN5$ (h)	$IN6$ (1)
L1 ($k_{1x'}$)	☐	☐				
L2 ($k_{1y'}$)	☐	☐				
L3 ($h\ a_{21}$)					☐	
L4 ($h\ a_{31}$)					☐	
L5 ($h\ a_{32}$)					☐	
L6 (b_1)						☐
L7 (b_2)						☐
L8 (b_3)						☐
L9 ($k_{2x'}$)	☐	☐				
L10 ($k_{2y'}$)	☐	☐				
L11 (k_{2x})			☐			
L12 (k_{2y})				☐		
L13 ($k_{3x'}$)	☐	☐				
L14 ($k_{3y'}$)	☐	☐				
L15 (k_{3x})			☐			
L16 (k_{3y})				☐		
L17 (x_{n+1})	☐		☐		☐	
L18 (y_{n+1})		☐		☐	☐	
L19 (x'_{n+1})			☐		☐	
L20 (y'_{n+1})				☐	☐	

Table 4: The Connections between the Third Order Runge Kutta ANN Hidden and Output Layers

$\frac{Layer}{Layer}$	L9 ($k_{2x'}$)	L10 ($k_{2y'}$)	L11 (k_{2x})	L12 (k_{2y})	L13 ($k_{3x'}$)	L14 ($k_{3y'}$)	L15 (k_{3x})	L16 (k_{3y})	L17 (x_{n+1})	L18 (y_{n+1})	L19 (x'_{n+1})	L20 (y'_{n+1})
L1 ($k_{1x'}$)	☐	☐			☐	☐					☐	
L2 ($k_{1y'}$)	☐	☐			☐	☐						☐
L3($h\ a_{21}$)	☐	☐	☐	☐								
L4($h\ a_{31}$)					☐	☐	☐	☐				
L5 ($h\ a_{32}$)					☐	☐	☐	☐				
L6 (b_1)									☐	☐	☐	☐
L7 (b_2)									☐	☐	☐	☐
L8 (b_3)									☐	☐	☐	☐
L9 $k_{2x'}$)					☐	☐					☐	
L10 ($k_{2y'}$)					☐	☐						☐
L11 (k_{2x})							☐		☐			
L12 (k_{2y})								☐		☐		
L13 ($k_{3x'}$)											☐	
L14 ($k_{3y'}$)												☐
L15 (k_{3x})									☐			
L16 (k_{3y})										☐		

5. Data analysis and results

We implemented the third-order Runge Kutta neural network that was described so far by MATLAB development environment. Some experiments were conducted by different integration interval and step length. In all experiments, the maximum training iteration was 10000; this number of iteration was obtained empirically by trial and error and more than that do not improve the answer. The best result was achieved in the integration interval $[0, 2\pi]$ and the step length $h = \frac{1}{32}$ is created. The integration interval $[0, 2\pi]$ is a complete fluctuation of solution of a two-body problem which has a analytical solution of cos and sin. After training, the Runge Kutta coefficients satisfy the third-order Runge Kutta conditions with accuracy up to 4 decimal digits and also the output of network, which is the solution of Runge Kutta, is equal to the analytical answer of the two-body problem with accuracy up to four decimal digits.

Furthermore, we have compared the results obtained by the third-order Runge Kutta with coefficients obtained by neural network and two other well-known coefficients in the literatures namely Classical and Heun. Coefficients calculated by neural network are shown in Table 5 and coefficients of Classical and Heun methods are depicted in Tables 6 and 7 respectively.

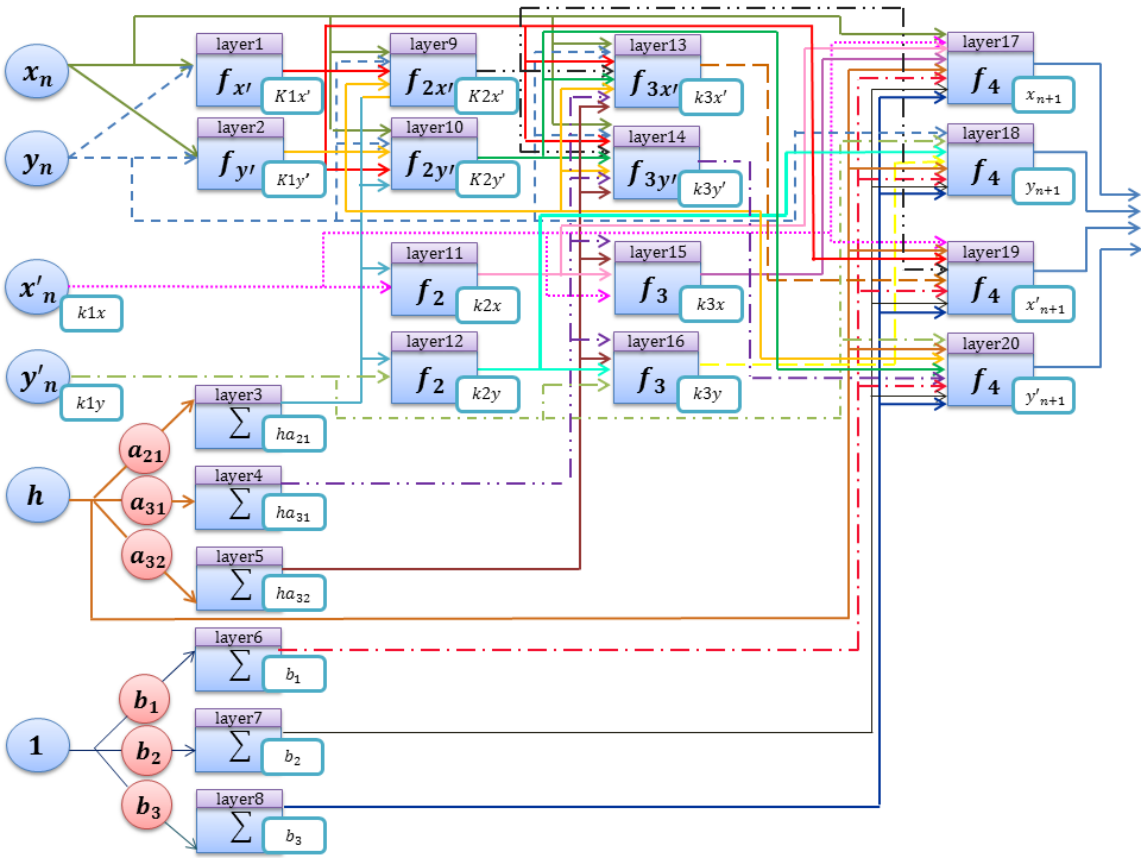

Fig. 2: The Third Order Runge Kutta Neural Network

Table 5: Butcher Table of the ANN Method

0.6206744	0.6206744		
0.6777417	0.2256413	0.4521004	
	0.2490056	0.1570639	0.5939214

Table 6: Butcher Table of Heun Method

$\frac{1}{3}$	$\frac{1}{3}$		
$\frac{2}{3}$	0	$\frac{2}{3}$	
	$\frac{1}{4}$	0	$\frac{3}{4}$

Table 7: Butcher Table of Classical Method

$\frac{1}{2}$	$\frac{1}{2}$		
1	-1	2	
	$\frac{1}{6}$	$\frac{4}{6}$	$\frac{1}{6}$

Comparison of these methods for solving two-body problem is shown on Figure 3 and Table 8. The horizontal axis of Figure 3 shows the total number of function evaluation for two-body problem which is calculated by formula $F = s.N$, where s=2 is the total number of steps and N is the number of iterations for the Runge Kutta algorithm for solving two-body problem. The vertical axis shows the maximum absolute error (MAE) where error is the difference between output and target. As we can see in Figure 3, for less than 320 times of function evaluation MAE of neural network is more than of Heun and Classical methods. After this number of function evaluation, MAE of the classic method will raise significantly. For function evaluation between 320 to 490, the number of MAE of neural network is less than Classic method but more than Heun. For function evaluation more than 490, the MAE of neural network method is less than two other methods. Therefore, we can say that neural network is superior to two other methods with respect to MAE criterion.

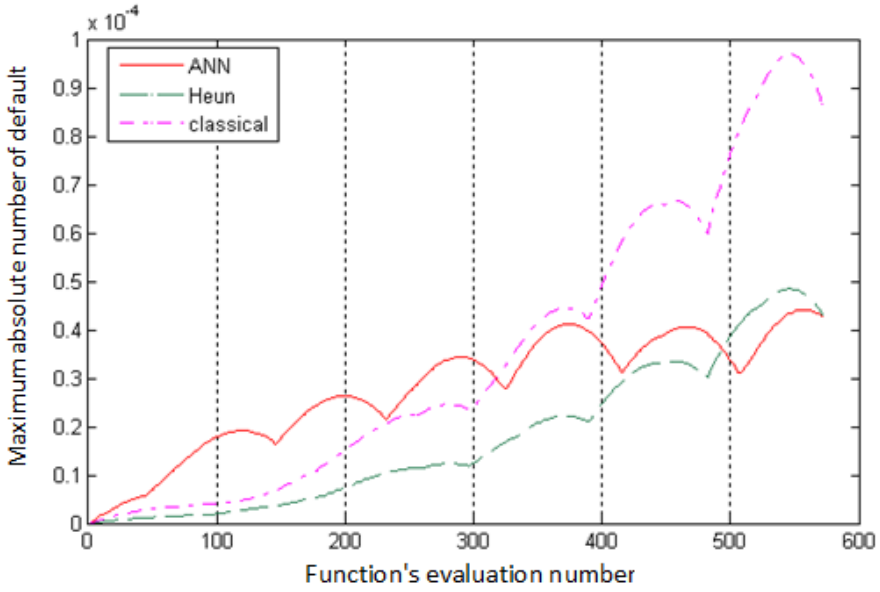

Fig. 3: Efficiency of the Third Order Runge Kutta Method

Table 8: Maximum Absolute Error Value Obtained by Application Three Different Methods of Coefficients

step length h	ANN	Heun	Classical
2	2.2881	**0.5609**	1.5366
1	**0.1312**	0.4777	0.5045
$\frac{1}{2}$	**0.0174**	0.0595	0.1312
$\frac{1}{4}$	**0.0021**	0.0078	0.0161
$\frac{1}{6}$	**0.00058**	0.0023	0.0047
$\frac{1}{8}$	**0.00024**	0.00099	0.0020
$\frac{1}{32}$	0.000042	**0.000015**	0.000031

Table 8 shows MAE values for solving two-body problem by three coefficient methods for different step lengths. To simplify the comparison, the minimum MAE value was bolded for each step length. As can be seen, in step lengths of 2 and $\frac{1}{32}$, the Heun method and for all other step lengths, the performance of ANN is better than the other methods.

6. Conclusion

In this study, we analyzed production of the third order Runge Kutta method by MLP neural network. The network was designed in a way that its training gives the optimum coefficient of third order Runge Kutta with accuracy up to 4 decimal digits.

References

[1] Anastassi A.,"Constructing Runge–Kutta methods with the use of artificial neural networks", Neural Comput & Applic, Vol. 25, (2013), pp.229- 236, available online: http:// arxiv.org/pdf/1106.1194

[2] Burden L, Faires J, Numerical Analisis, Cengage Learning, (2001), pp: 328-335.

[3] Atkinson K, Han W, Stewart D, Numerical solution of ordinary differential equations, John Wiley & Sons, (2009), PA: 15. http://dx.doi.org/10.1002/9781118164495.

[4] Dissanayake M.W.M.G, Phan-Thien N, "Neural-network based approximations for solving partial differential equations", Communicational in Numerical Methods in Engineering, Vol. 25, (1994), PP. 195–201.

[5] Meade AJ. Jr, Fernandez A.A, "Solution of nonlinear ordinary differential equations by feedforward neural networks", Mathematical and Computer Modelling, Vol. 20, (1994), pp. 19 – 44, available online: http://www.sciencedirect.com/science/article/pii/089571779400160X. http://dx.doi.org/10.1016/0895-7177(94)00160-X.

[6] Meade AJ.Jr, Fernandez A.A, "The numerical solution of linear ordinary differential equations by feedforward neural networks", Mathematical and Computer Modelling, Vol. 19, No. 12, (1994), pp. 1–25, available online: http://www.sciencedirect.com/science/article/pii/0895717794900957. http://dx.doi.org/10.1016/0895-7177(94)90095-7.

[7] Puffer F, Tetzlaff R, Wolf D, "Learning algorithm for cellular neural networks (CNN) solving nonlinear partial differential equations", proceedings international symposium on singnals, systems and Electronics, (1995), pp. 501–504.
[8] He S, Reif K, Unbehauen RS, "Multilayer neural networks for solving a class of partial differential equations", Neural Networks, Vol. 13, (2000), pp. 385–396, available online: http://www.sciencedirect.com/science/article/pii/S0893608000000137. http://dx.doi.org/10.1109/ISSSE.1995.498041.
[9] Alexandridis A, Chondrodima E, "Sarimveis H, Radial basis function network training using a nonsymmetric partition of the input space and particle swarm optimization", IEEE Transactions on Neural Networks and Learning Sysemst, Vol. 24, No. 2, (2013), pp. 219 –230, available online: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6395832.
[10] Jianyu L, Siwei L, Yingjian, Q., Yaping, H., "Numerical solution of differential equations by radial basis function neural networks", Proceedings of International Joint Conference on Neural Networks, (2002),2 pp: 773–777, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1005571.
[11] Ramuhalli P, Udpa L, Udpa S.S, "Finite-element neural networks for solving differential equations", IEEE Transactions on Neural Networks, Vol. 16, (2005), pp. 1381–1392, available online: http://people.sc.fsu.edu/~jburkardt/classes/sem_2015/ramuhalli.pdf. http://dx.doi.org/10.1109/TNN.2005.857945.
[12] Butcher J.C, Numerical methods for ordinary differential equations, Wiley, (2003). http://dx.doi.org/10.1002/0470868279.
[13] Haykin S, Neural networks: a comprehensive foundation, Prentice Hall Englewood Cliffs, (1999).
[14] Yu, C.C, Liu, B.D, "A back-propagation algorithm with adaptive learning rate and momentum coefficient", Proceedings of 2002 International Jointt Conference on Neural Networks, (2002) pp: 1218-1223, http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1007668&url
[15] Vahidi J, Qasem Pour S, Numerical calculations Methods, Computer Science, (1969), pp: 247-257.