Applied Mathematics 2019, 9(2): 25-48
DOI: 10.5923/j.am.20190902.01

Estimation of Tinkerbell Dynamical Map by Using Neural Network with FFT as Transfer Function

Salah H. Abid*, Saad S. Mahmood, Yaseen A. Oraibi

Department of Mathematics, College of Education, AL-Mustansiriyah University, Baghdad, Iraq

Abstract The aim of this paper is to design a feed forward artificial neural network (Ann) to estimate two dimensional Tinkerbell dynamical map by selecting an appropriate network, transfer function and node weights. The proposed network side by side with using Fast Fourier Transform (FFT) as transfer function is used. For different cases of the system, chaotic and noisy, the experimental results of proposed algorithm will compared empirically, by means of the mean square error (MSE) with the results of the same network but with traditional transfer functions, Logsig and Tansig. The performance of proposed algorithm is best from others in all cases from Both sides, speed and accuracy.

Keywords FFT, Logsig, Tansig, Feed Forward neural network, Transfer function, Tinkerbell map, Logistic noise, Normal noise

1. Introduction

Ann is a simplified mathematical model of the human brain. It can be implemented by both electric elements and computer software. It is a parallel distributed processor with large numbers of connections; it is an information processing system that has certain performance characters in common with biological neural networks. Ann has been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

1- Information processing occurs at many simple elements called neurons that are fundamental to the operation of Ann's.
2- Signals are passed between neurons over connection links.
3- Each connection link has an associated weight which, in a typical neural net, multiplies the signal transmitted.
4- Each neuron applies an action function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal [15].

The units in a network are organized into a given topology by a set of connections, or weights.

Ann is characterized by [28]:

1- Architecture: its pattern of connections between the neurons.
2- Training Algorithm: its method of determining the weights on the connections.
3- Activation function.

Ann are often classified as single layer or multilayer. In determining the number of layers, the input units are not counted as a layer, because they perform no computation. Equivalently, the number of layers in the net can be defined to be the number of layers of weighted interconnects links between the slabs of neurons [44].

1.1. Multilayer Feed Forward Architecture [21]

In a layered neural network the neurons are organized in the form of layers. We have at least two layers: an input and an output layer. The layers between the input and the output layer (if any) are called hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units. Extra hidden neurons raise the network's ability to extract higher-order statistics from (input) data.

The Ann is said to be fully connected in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer; otherwise the network is called partially connected. Each layer consists of a certain number of neurons; each neuron is connected to other neurons of the previous layer through adaptable synaptic weights w and biases b.

1.2. Literature Review

Pan and Duraisamy in 2018 [31] studied the use of feedforward neural networks (FNN) to develop models of non-linear dynamical systems from data. Emphasis is placed on predictions at long times, with limited data availability. Inspired by global stability analysis, and the observation of strong correlation between the local error and the maximal

* Corresponding author:
abidsalah@uomustansiriyah.edu.iq (Salah H. Abid)
Published online at http://journal.sapub.org/am

singular value of the Jacobian of the ANN, they introduced Jacobian regularization in the loss function. This regularization suppresses the sensitivity of the prediction to the local error and is shown to improve accuracy and robustness. Comparison between the proposed approach and sparse polynomial regression is presented in numerical examples ranging from simple ODE systems to nonlinear PDE systems including vortex shedding behind a cylinder, and instability-driven buoyant mixing ow. Furthermore, limitations of feedforward neural networks are highlighted, especially when the training data does not include a low dimensional attractor. The need to model dynamical behavior from data is pervasive across science and engineering. Applications are found in diverse fields such as in control systems [41], time series modeling [37], and describing the evolution of coherent structures [12]. While data-driven modeling of dynamical systems can be broadly classified as a special case of system identification [23], it is important to note certain distinguishing qualities: the learning process may be performed off-line, physical systems may involve very high dimensions, and the goal may involve the prediction of long-time behavior from limited training data. Artificial neural networks (ANN) have attracted considerable attention in recent years in domains such as image recognition in computer vision [18, 35] and in control applications [12]. The success of ANNs arises from their ability to effectively learn low-dimensional representations from complex data and in building relationships between features and outputs. Neural networks with a single hidden layer and nonlinear activation function are guaranteed to be able to predict any Borel measurable function to any degree of accuracy on a compact domain [16]. The idea of leveraging neural networks to model dynamical systems has been explored since the 1990s. ANNs are prevalent in the system identification and time series modeling community [20, 26, 27, 33], where the mapping between inputs and outputs is of prime interest. Billings et al. [5] explored connections between neural networks and the nonlinear autoregressive moving average model (NARMAX) with exogenous inputs. It was shown that neural networks with one hidden layer and sigmoid activation function represent an infinite series consisting of polynomials of the input and state units. Elanayar and Shin [13] proposed the approximation of nonlinear stochastic dynamical systems using radial basis feedforward neural networks. Early work using neural networks to forecast multivariate time series of commodity prices [9] demonstrated its ability to model stochastic systems without knowledge of the underlying governing equations. Tsung and Cottrell [42] proposed learning the dynamics in phase space using a feedforward neural network with time-delayed coordinates. Paez and Urbina [29, 30, 43] modeled a nonlinear hardening oscillator using a neural network-based model combined with dimension reduction using canonical variate analysis (CVA). Smaoui [38, 39, 40] pioneered the use of neural networks to predict fluid dynamic systems such as the unstable manifold model for bursting behavior in the 2-D Navier-Stokes and the Kuramoto-Sivashinsky equations. The dimensionality of the original PDE system is reduced by considering a small number of proper orthogonal decomposition (POD) coefficients [4]. Interestingly, similar ideas of using principal component analysis for dimension reduction can be traced back to work in cognitive science by Elman [14]. Elman also showed that knowledge of the intrinsic dimensions of the system can be very helpful in determining the structure of the neural network. However, in the majority of the results [38, 39, 40], the neural network model is only evaluated a few time steps from the training set, which might not be a stringent performance test if longer time predictions are of interest. ANNs have also been applied to chaotic nonlinear systems that are challenging from a data-driven modeling perspective, especially if long time predictions are desired. Instead of minimizing the pointwise prediction error, Bakker et al. [3] satisfied the Diks'criterion in learning the chaotic attractor. Later, Lin et al. [21] demonstrated that even the simplest feedforward neural network for nonlinear chaotic hydrodynamics can show consistency in the time-averaged characteristics, power spectra, and Lyapunov exponent between the measurements and the model. A major difficulty in modeling dynamical systems is the issue of memory. It is known that even for a Markovian system, the corresponding reduced-dimensional system could be non-Markovian [10, 32]. In general, there are two main ways of introducing memory effects in neural networks. First, a simple workaround for feedforward neural networks (FNN) is to introduce time delayed states in the inputs [11]. However, the drawback is that this could potentially lead to an unnecessarily large number of parameters [19]. To mitigate this, Bakker [3] considered following Broomhead and King [6] in reducing the dimension of the delay vector using weighted principal component analysis (PCA). The second approach uses output or hidden units as additional feedback. As an example, Elman's network [19] is a recurrent neural network (RNN) that incorporates memory in a dynamic fashion. Miyoshi et al. [24] demonstrated that recurrent RBF networks have the ability to reconstruct simple chaotic dynamics. Sato and Nagaya [36] showed that evolutionary algorithms can be used to train recurrent neural networks to capture the Lorenz system. Bailer-Jones et al. [2] used a standard RNN to predict the time derivative in discrete or continuous form for simple dynamical systems; this can be considered an RNN extension to Tsung's phase space learning [42]. Wang et al. [45] proposed a framework combining POD for dimension reduction and long-short-term memory (LSTM) recurrent neural networks and applied it to a fluid dynamic system.

1.3. Fast Fourier Transform

The first to propose the techniques that we now call the fast Fourier transform (FFT) for calculating the coefficients in a trigonometric expansion of an asteroid's orbit in 1805 [8]. However, Fast Fourier transform is an algorithm that calculates the value of the discrete Fourier transform in faster. the speed this algorithm Is due to the fact that it does not

calculate the parts that are equal to zero .the algorithm is discovered by James W. Cooley and John W. Tukey who published the algorithm in 1965 [11]

As know today

$$x(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}(a_n cos\frac{\pi n}{L}t + b_n sin\frac{\pi n}{L}t) \quad (1)$$

The coefficients can be determined as follows:

$$a_n = \frac{1}{L}\int_{-L}^{L} x(t)cos\frac{\pi n}{L}t\,dt \quad (2)$$

$$b_n = \frac{1}{L}\int_{-L}^{L} x(t)sin\frac{\pi n}{L}t\,dt \quad (3)$$

$$a_0 = \frac{1}{L}\int_{-L}^{L} x(t)dt \quad (4)$$

The discrete Fourier transform (DFT) is one of the most powerful tools in Digital signal processing. The DFT enables us to conveniently analyze and Design systems in frequency domain [1] and the formal as:

$$X_k = \sum_{n=1}^{N-1} x_n e^{-j\frac{2\pi}{N}nk} \quad (5)$$

This paper contains six sections, the first one related with general concepts and some literature review. In the second section, we introduce some properties of Tinkerbill map. The suggested network in details is discussed in section three, while description of the training process for Tinkerbell map is presented in section four. Section five contains the results discussion. Finally, the summary of the research is in section six.

2. Tinkerbell Map [17]

The Tinkerbell two dimensional map is expressed through the following equation:

$$\begin{cases} x_{n+1} = x_n^2 - y_n^2 + ax_n + by_n \\ y_{n+1} = 2x_n y_n + cx_n + dy_n \end{cases} \quad (6)$$

Where subscript n represents iteration steps corresponding to the discrete time evolution of the system, and a, b, c, d are parameters. The origin of the name Tinkerbell is unknown; however, the graphical picture of the system shows a similarity to the movement of the Tinkerbell over the Cinderella Castle, as shown at the beginning of the familiar Disney cartoons. The study of map is mathematically fundamental with many practical interests.

The Tinkerbell map for some special cases has been studied; for example, Nusse and Yorke [1997] reported the finding of 64 period-10 unstable periodic orbits and one strange chaotic with a fractal basin boundary at a = 0.9, b = − 0.6013, c = 2, d = 0.5 using a quasi-Newton method on map. Davidchack et al. [2001] implemented the Schmelcher–Diakonos method on map and observed a drastic increase in the number of seeds as the period is increased. In addition, Aulbach and Colonius [1996] provided a figure of pseudo-orbit of length 1 00 000 at a = 0.9, b = − 0.6013, c = 2, d = 0.5 of the map by using the Finite-time Shadowing Theorem [Aulbach & Colonius, 1996].

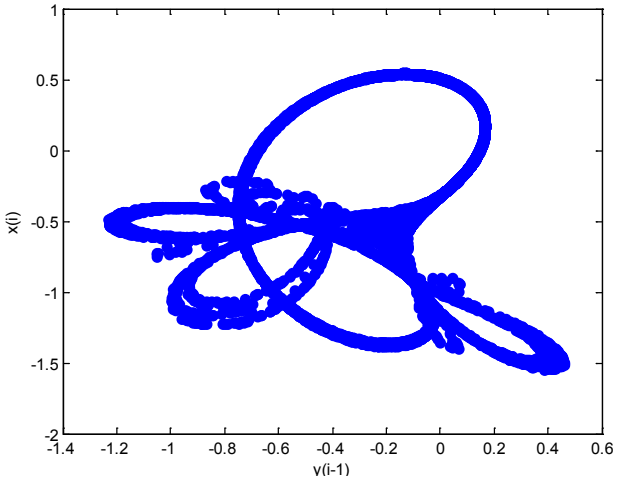

Figure 1. Tinkerbell attractor for a = 0.9 and b = -0.6103, c=2, d=0.5

2.1. Tinkerbell Map Solution

In this section we will explain how this approach can be used to find the approximate solution of the Tinkerbell map that is stated in equation (6).

T(x) is the solution to be computed. Let $y_t(x, p)$ denotes a trial solution with adjustable parameters p.

In the proposed approach, the trial solution y_t employs a FFNN and the parameters p corresponding to the weights and biases of the neural architecture. We choose a form for the trial function $y_t(x,p)$ such that $y_t(x,p) = N(x, p)$ where N(x, p) is a single-output FFNN with parameters (weights) p and n input units fed with the input vector x.

2.2. Computation of the Gradient

The error corresponding to each input vector x_i is the value E (x_i) which has to force near zero. Computation of this error value involves not only the FFNN output but also the derivatives of the output with respect to any of its inputs. Therefore, for computing the gradient of the error with respect to the network weights, consider a multilayer FFNN with n input units (where n is the dimensions of the domain), two hidden layer with H sigmoid units,q hidden layer and a linear output unit .

For a given input vector x = (x_1, x_2, …, x_n) the output of the FFNN is :

$$N = \sum_{i=1}^{H} v_k \sigma(u_i),\ z_i = \sum_{j=1}^{n} w_{ij}x_j + b_i \quad \text{and}$$

$$u_k = \sum_{k=1}^{q} s_{ik}\,\sigma(z_j) + b_{ik}$$

w_{ij} denotes the weight connecting the input unit j to the hidden unit i

s_{ik} denotes the weight connecting the hidden unit i to the hidden unit k

v_k denotes the weight connecting the hidden unit k to the out put unit,

b_i denotes the bias of hidden unit i,

b_{ik} denotes the bias of hidden unit i to the hidden unit k, and

σ is the transfer function

The gradient of suggest FFNN, with respect to the coefficients of the FFNN can be computed as:

$$\frac{\partial N}{\partial v_k} = \sigma(u_i), \quad (7)$$

$$\frac{\partial N}{\partial b_i} = v_k \sigma'(u_i), \quad (8)$$

$$\frac{\partial N}{\partial wij} = v_k \sigma'(u_i) x_j, \quad (9)$$

$$\frac{\partial N}{\partial s_{ik}} = v_k \, \sigma(z_j) \sigma'(u_i) x_j, \quad (10)$$

$$\frac{\partial N}{\partial b_{ik}} = v_k s_{ik} \sigma'(u_i) \sigma'(z_j), \quad (11)$$

The derivative of the error performance with respect to the FFNN coefficients can be defined and then it is easy to find minimization solution.

3. Suggested Network

It is well known that a multilayer FFNN consist one hidden layer can approximate any function to any accuracy [25], but dynamical maps they have more completed behavior than other functions, thus, we suggest FFNN contains two hidden layer, one input and one output to estimate a solution for dynamic maps.

The suggested network divided the inputs in to two parts 60% for training and 40% for testing. The error quantity to be minimized is given by:

$$E[p] = \sum_{i=1}^{n} \{T(xi) - y_t(x_i)\}^2, \quad (12)$$

Where $x_i \in [0, 1]$. It is easy to evaluate the gradient of the performance with respect to the coefficient. Using (7) – (11). The training progress algorithm of FFNN with supervises training and BFGS algorithm is given as follows. Assume that there are one node in the First layer (input layer), the second layer (hidden layer) consist with ten nodes, the third layer (hidden layer) consist with five nodes and the fourth layer (output layer) consist with one node.

Following the steps of the technique as an algorithm,

Step 0: ***input and target.***

Insert the input (***x***: $x_1, x_2, x_3, \ldots, x_n$) and the target.

Step 1: ***allocating inputs.***

Each input goes to each neuron with a hidden layer.

Step 2: ***initialization weights.***

We initial weights and bias from the uniform distribution respectively for all connection in the neural network.

Step 3: ***select the following.***

- parameter epoch
- parameter goal (ε)
- performance function (MSE)

Step 4: ***calculations in each node in the first hidden layer.***

In each node in hidden layer, compute the sum of the product of weights and inputs and adding the result to the bias.

Step 5: ***compute the output of each node for the first hidden layer.***

Take the active function for sum value in step4, then its output is sent to the second hidden layer as input.

Step 6: ***calculations in each node in the second layer.***

In each node in second hidden layer, compute the sum of the product of weights and inputs and add the result to the bias.

Step 7: ***compute the output of each node for the second hidden layer.***

Take the active function for sum value in step6, then its output is sent to the output layer as input.

Step 8: ***calculations in output layer.***

There is only one neuron (node) in the output layer. The node sum is the product of weights by inputs.

Step 9: ***compute the output of node in output layer***

The value of active function for node output is also considered as the output of overall network.

Step 10: ***compute the mean square error (MSE).***

The mean square error is computed as follows

MSE $= \frac{1}{n} \sum_{r=0}^{n} e_r^2$ where $e_r = t_r - y_r$

the MSE is a measure of performance.

Step 11: ***The checking.***

When $MSE \leq \varepsilon$ such that ε is small value close to zero, then stop the training and the bias and weights are sent. Otherwise training process goes to the next step.

Step 12: when select the training rule, the low for update weights and bias between the hidden layer and the output layer are calculated

Step 13: ***the update weights and bias in output layer.***

At end for each iteration, the weights and bias are updating as follows:

$$v_{k\,new} = v_{k\,old} - \alpha H_k^{-1} g_k$$

$$b_{k\,new} = b_{k\,old} - \alpha H_k^{-1} g_k$$

When (new) means the current iteration and (old) means the previous iteration,

g_k represents the gradient for weights and bias, α is the parameter selected to minimize the performance function along the search direction, H_k^{-1} represent the invers hessian matrix,v is the weight in the output layer and b is the bias.

Step 14: ***the update of weights and bias in the first hidden layer.***

Each hidden node in the first hidden layer updates the weights and bias as follow:

$$w_{k\,new} = w_{k\,old} - \alpha H_k^{-1} g_k$$

$$b_{k\,new} = b_{k\,old} - \alpha H_k^{-1} g_k$$

Where w is the weight of hidden layer and b is the bias.

Step 15: ***the update of weights and bias in the second hidden layer as follow.***

$$s_{k\,new} = s_{k\,old} - \alpha H_k^{-1} g_k$$

$$b_{k\,new} = b_{k\,old} - \alpha H_k^{-1} g_k$$

Where s is the weight of hidden layer and b is the bias.

Step 16: return to step2 for next iteration.

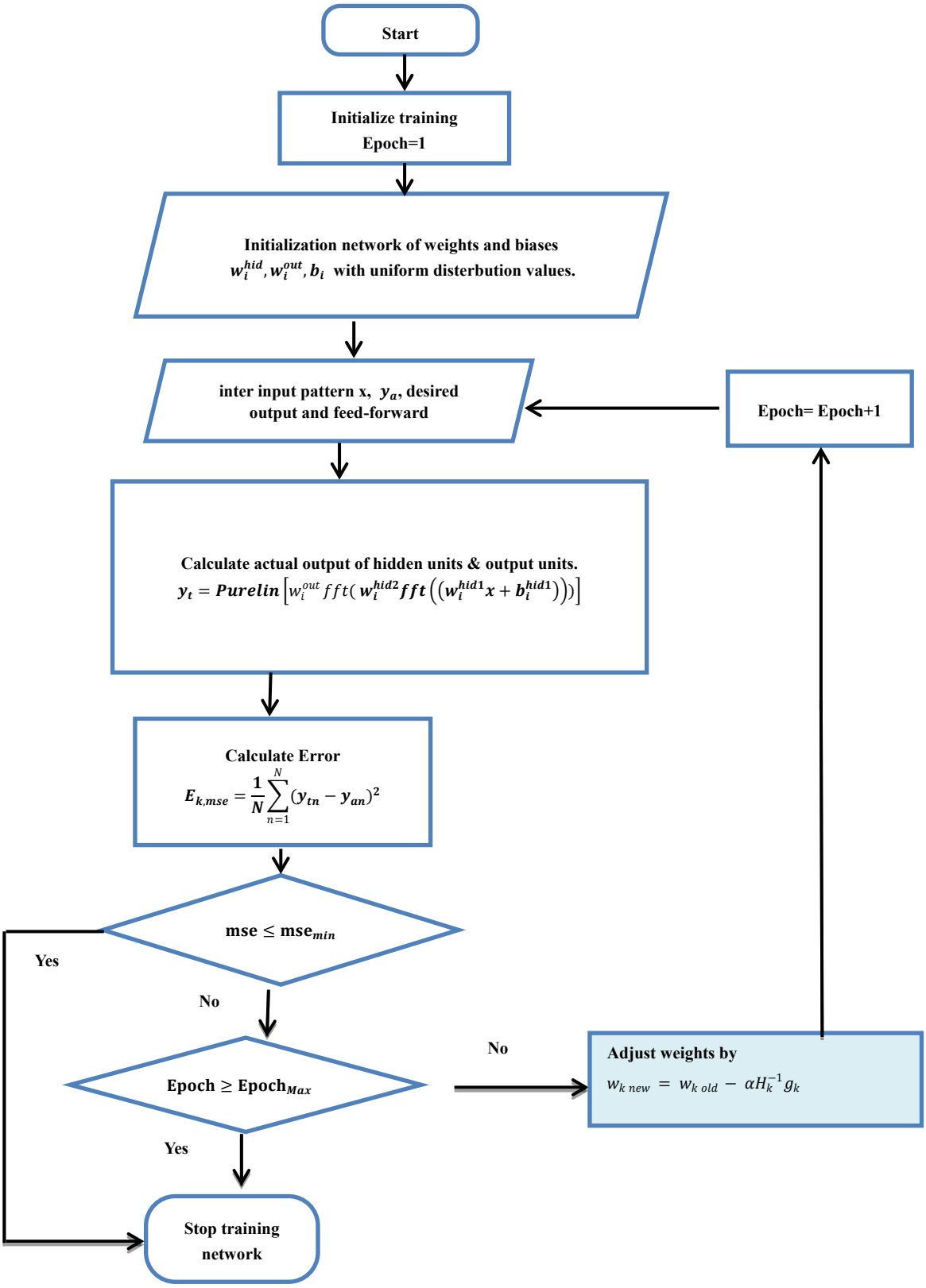

Figure 2. Flowchart for training algorithm with BFGS

4. Description of the Training Process for Tinkerbell Map

We will train the case when the Tinkerbell map is chaotic a=0.9, b= -0.6103, c=2 and d=0.5 and train the input data (x and y) separately using the proposed FFNNs with one input, one output and tow hidden layers and we train this case with three types of transfer functions Tansig, logsig, and FFT.

Table (1) contains the results. Figures from (3) to (14) show the Real and approximate Tinkerbell dynamical map with different situations.

The weighted MSE is used as performance function according to x and y performance functions weights.

Table (1). Time and MSE of approximate solution after many training trials by using different transfer functions

Transfer function	Initial point								Number points
	-0.1		-0.4		-0.7		-0.9		
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	18
FFT	2.27e-14	0:00:05	1.32e-14	0:00:08	5.23e-15	0:00:05	1.92e-14	0:00:07	
Tansig	1.94e-13	0:00:09	5.95e-14	0:00:12	2.93e-15	0:00:10	1.53e-14	0:00:11	
Logsig	1.47e-14	0:00:13	2.84e-14	0:00:12	8.4e-14	0:00:10	9.4e-15	0:00:13	

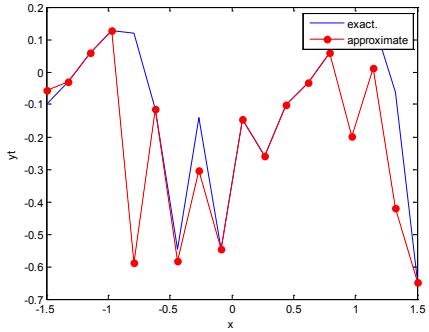

Figure 3. Real and approximate Tinkerbell dynamical map with initial point x=-0.1 using FFT transfer function

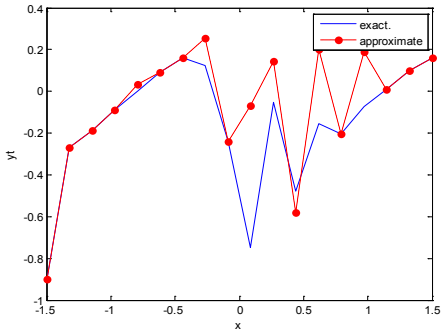

Figure 6. Real and approximate Tinkerbell dynamical map with initial point x=-0.9 using FFT transfer function

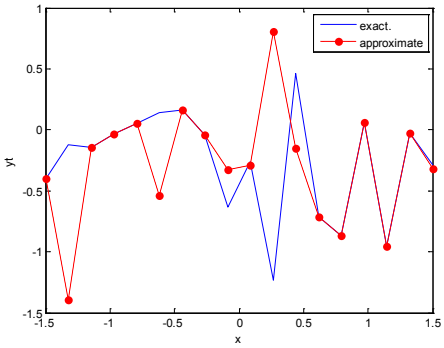

Figure 4. Real and approximate Tinkerbell dynamical map with initial point x=-0.4 using FFT transfer function

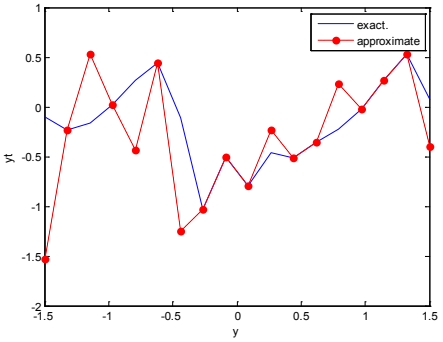

Figure 7. Real and approximate Tinkerbell dynamical map with initial point y=-0.1 using FFT transfer function

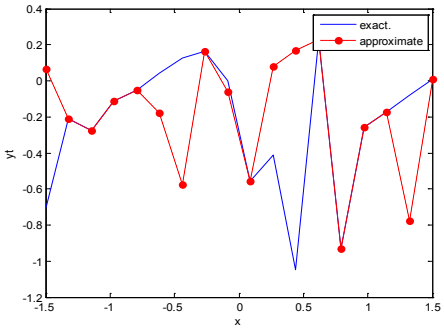

Figure 5. Real and approximate Tinkerbell dynamical map with initial point x=-0.7 using FFT transfer function

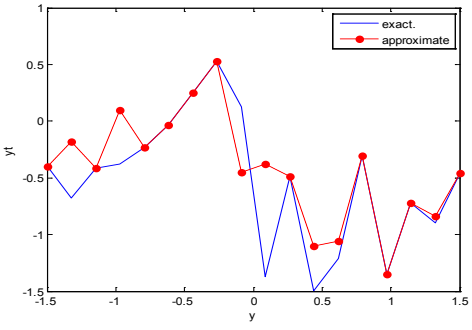

Figure 8. Real and approximate Tinkerbell dynamical map with initial point y=-0.4 using FFT transfer function

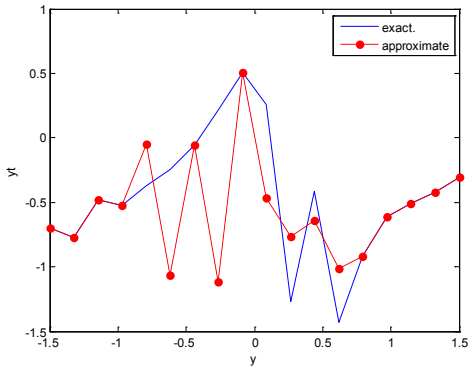

Figure 9. Real and approximate Tinkerbell dynamical map with initial point y=-0.7 using FFT transfer function

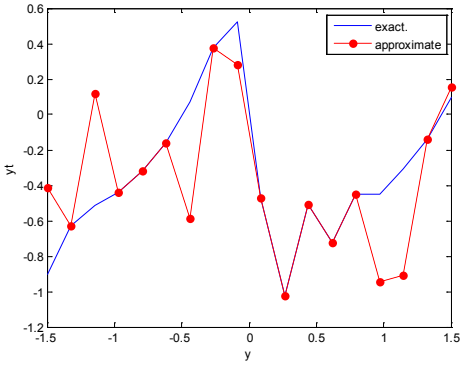

Figure 10. Real and approximate Tinkerbell dynamical map with initial point y=-0.9 using FFT transfer function

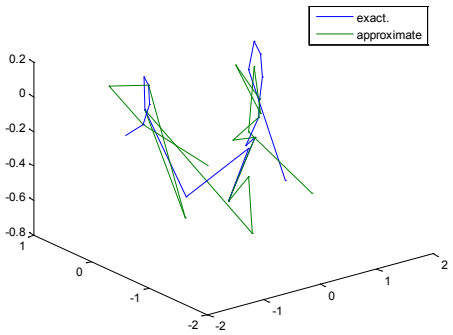

Figure 11. Real and approximate Tinkerbell dynamical map with initial points x=-0.1 & y=-0.1 using FFT transfer function

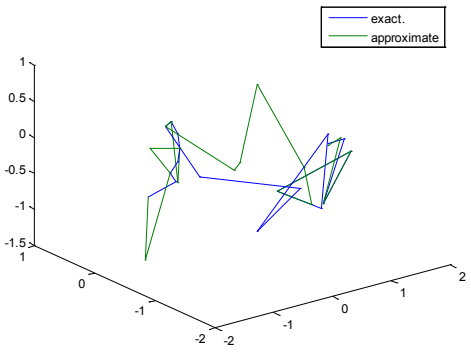

Figure 12. Real and approximate Tinkerbell dynamical map with initial points x=-0.4 & y=-0.4 using FFT transfer function

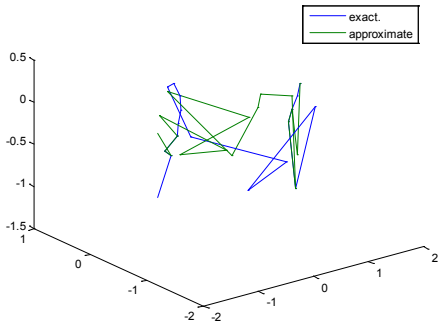

Figure 13. Real and approximate Tinkerbell dynamical map with initial points x=-0.7 & y=-0.7 using FFT transfer function

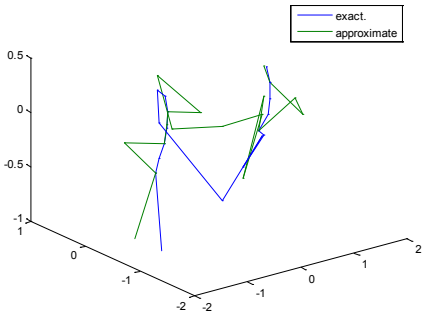

Figure 14. Real and approximate Tinkerbell dynamical map with initial points x=-0.9 & y=-0.9 using FFT transfer function

4.1. Results Discussion

The experimental results from above table (1) show that the logsig, tansig and FFT transfer functions with proposed FFNN have a very good performance when the Tinkerbell map is chaotic. But the FFT transfer function is faster than logsig and tansig transfer functions to get results for all initial points.

5. Description of Training Process for Tinkerbell Map with Noise

Here we train the normal and logistic noisy data of Tinkerbell map by using the suggested network with tansig, logsig and FFT transfer functions. It is suitable to choose the maximum number of epochs to reach high performance. It is worth to mention that the run size is k=1000.

5.1. Normal Distribution [34]

Let x be a normally distributed random variable with mean $-\infty < \mu < \infty$ and variance $0 < \sigma^2$, where $x \in (-\infty, \infty)$ with probability density function and cumulative distribution function are respectively,

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} * e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{13}$$

$$D(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-(x'-\mu)^2}{2\sigma^2}} dx' \tag{14}$$

One can generate values from Normal random variable as follows,

$$N = (\sqrt{-2 * \log(rand(1:q))} * \sin(2\pi * rand(1:q))) * v \quad (15)$$

Where q is represents the number point and v is represent the variance.

5.2. Logistic Distribution [34]

Let x be a Logistic distributed random variable with parameters $-\infty < m < \infty$ and $b > 0$, where $x \in (-\infty, \infty)$ with probability density function and cumulative distribution function are respectively,

$$P(x) = \frac{e^{-(x-m)/b}}{b[1+e^{-\frac{x-m}{b}}]^2} \quad (16)$$

$$D(x) = \frac{1}{1+e^{-(x-m)/b}} \quad (17)$$

One can generate values from Logistic random variable as follows,

$$L = \log\left(\frac{1}{rand\ (1:q)} - 1\right) * \frac{(3v)^{1/2}}{\pi} \quad (18)$$

Where q is represents the number point and v is represent the variance.

5.3. Training the Case with Normal Noise

The training process is conducted with variances 0.05, 0.5, 40 and 60.

Table (2) contains the results. Figures from (15) to (62) show the Real and approximate Tinkerbell dynamical map with different situation.

Table (2). Time and MSE of approximate solution after many training trials with normal noise by using different transfer functions when the variance equal to 0.05

Transfer function	Initial point								Number points
	-0.1		-0.4		-0.7		-0.9		
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	18
FFT	2.86e-14	0:00:04	4.38e-14	0:00:03	1.83e-15	0:00:06	4.34e-14	0:00:06	
Tansig	2.39e-14	0:00:04	3.21e-14	0:00:05	8.4e-14	0:00:06	5.4e-14	0:00:04	
Logsig	1.73e-14	0:00:04	1.94e-15	0:00:06	4.39e-15	0:00:06	9.4e-14	0:00:05	

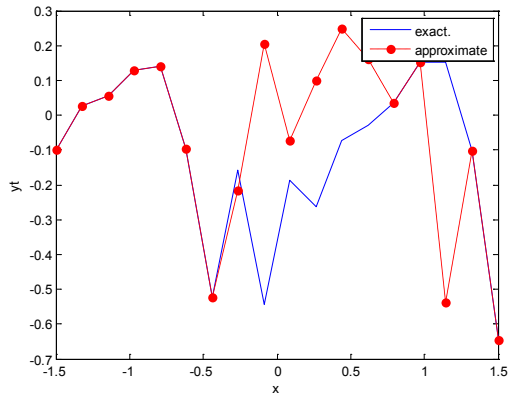

Figure 15. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial point x=-0.1 using FFT transfer function

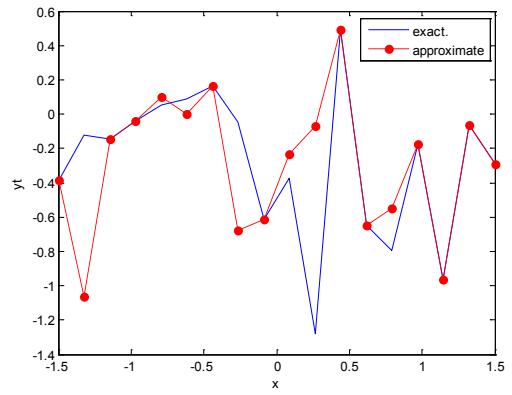

Figure 16. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial point x=-0.4 using FFT transfer function

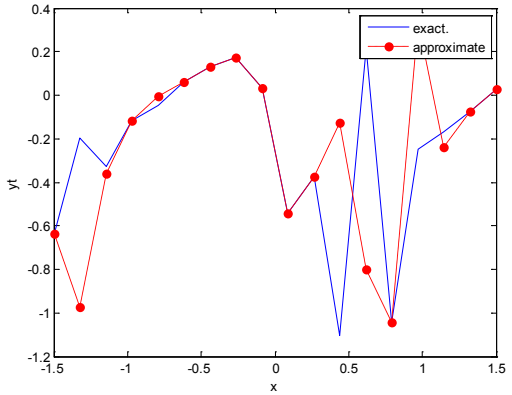

Figure 17. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial point x=-0.7 using FFT transfer function

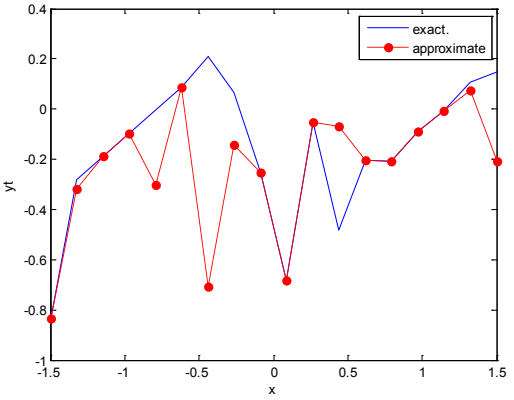

Figure 18. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial point x=-0.9 using FFT transfer function

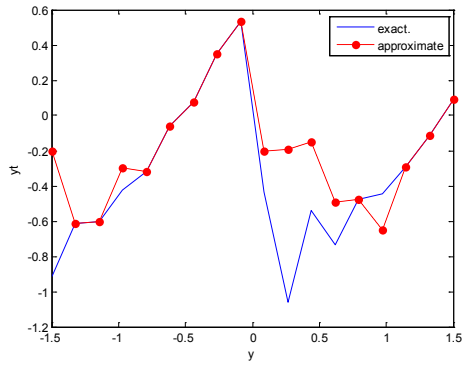

Figure 19. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial point y=-0.1 using FFT transfer function

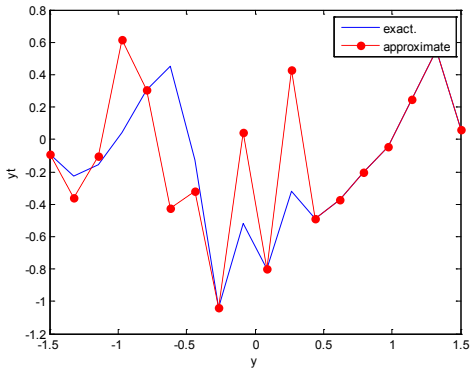

Figure 20. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial point y=-0.4 using FFT transfer function

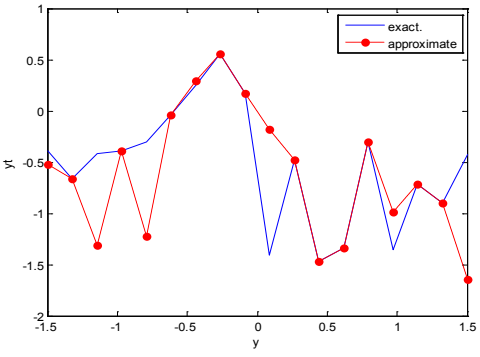

Figure 21. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial point y=-0.7 using FFT transfer function

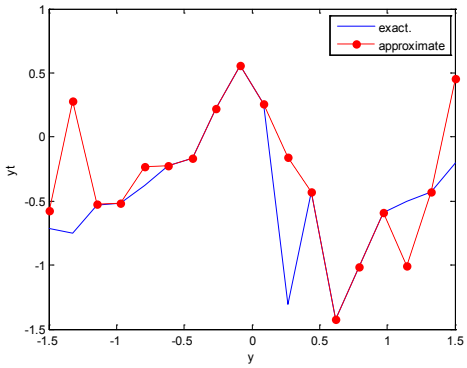

Figure 22. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial point y=-0.9 using FFT transfer function

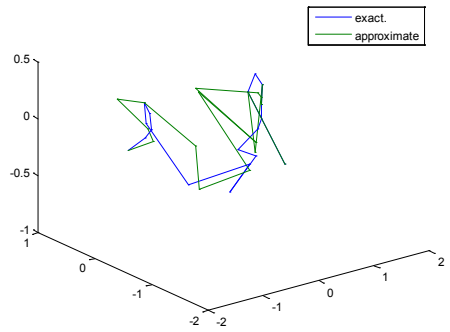

Figure 23. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial points x=-0.1 & y=-0.1 using FFT transfer function

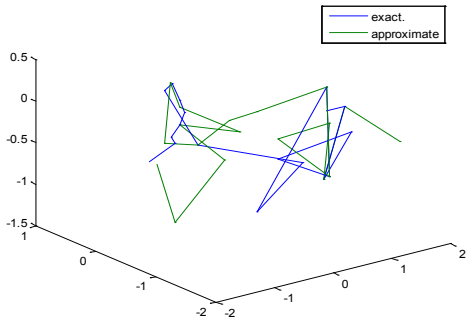

Figure 24. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial points x=-0.4 & y=-0.4 using FFT transfer function

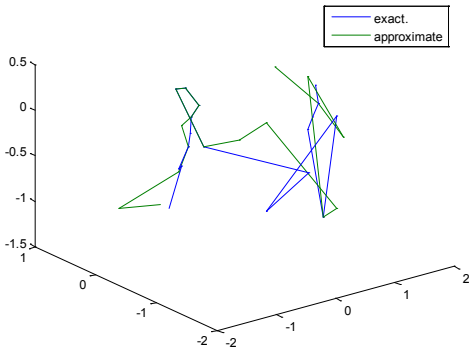

Figure 25. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial points x=-0.7 & y=-0.7 using FFT transfer function

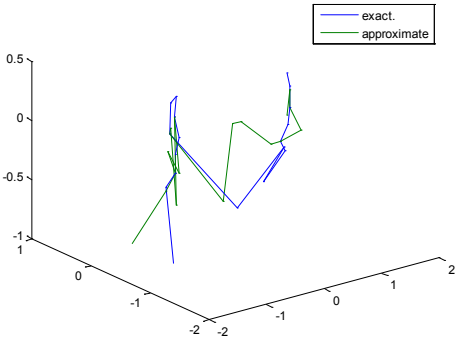

Figure 26. Real and approximate Tinkerbell dynamical map with normal noise (v=0.05) and initial points x=-0.9 & y=-0.9 using FFT transfer function

Table (3). Time and MSE of approximate solution after many training trials with normalnoise by using different transfer functions when the variance equal to 0.5

Transfer function	Initial point								Number points
	-0.1		-0.4		-0.7		-0.9		
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	18
FFT	1.83e-15	0:00:05	1.94e-15	0:00:06	4.39e-14	0:00:04	1.99e-14	0:00:06	
Tansig	1.27e-15	0:00:05	2.93e-16	0:00:06	3.82e-15	0:00:06	4.23e-15	0:00:05	
Logsig	1.83e-15	0:00:05	1.94e-15	0:00:06	4.39e-14	0:00:04	1.99e-14	0:00:06	

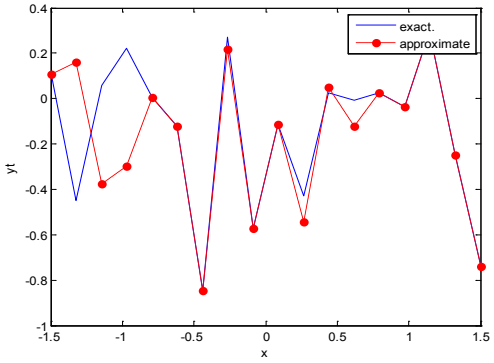

Figure 27. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial point x=-0.1 using FFT transfer function

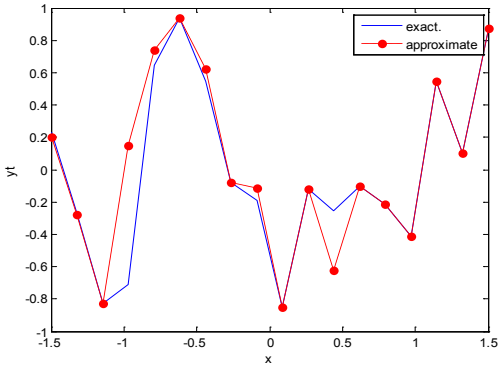

Figure 30. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial point x=-0.9 using FFT transfer function

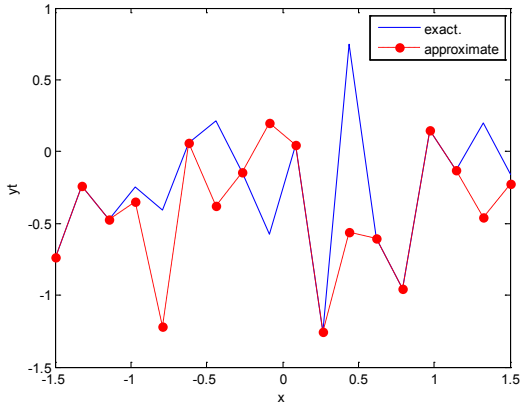

Figure 28. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial point x=-0.4 using FFT transfer function

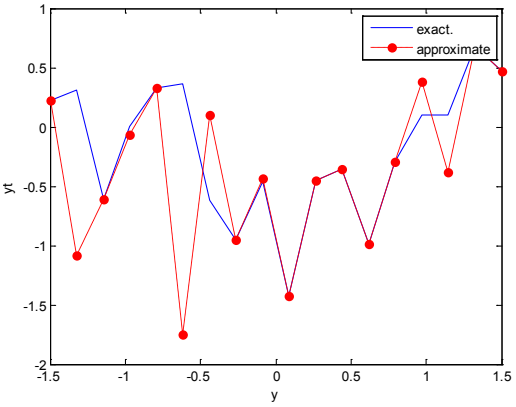

Figure 31. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial point y=-0.1 using FFT transfer function

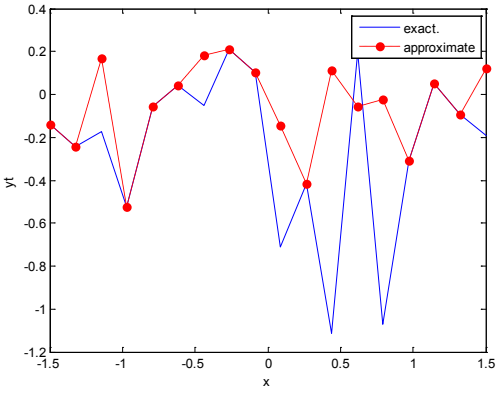

Figure 29. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial point x=-0.7 using FFT transfer function

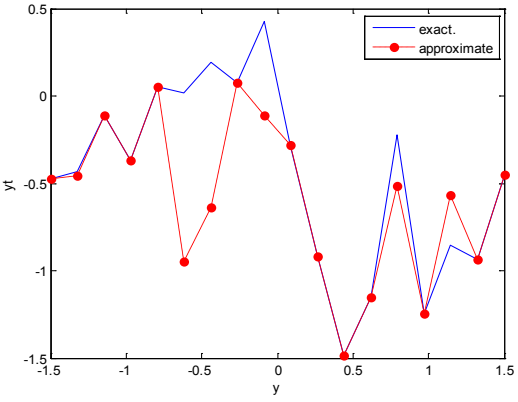

Figure 32. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial point y=-0.4 using FFT transfer function

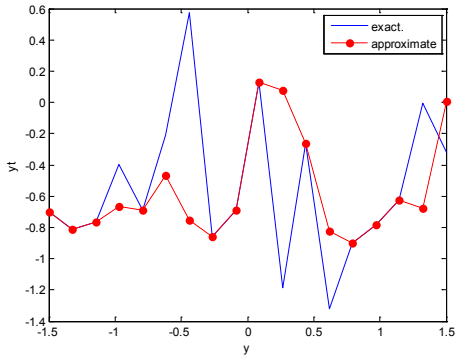

Figure 33. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial point y=-0.7 using FFT transfer function

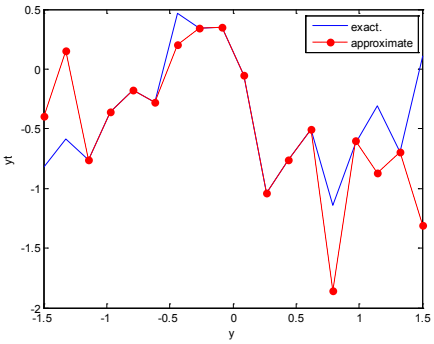

Figure 34. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial point y=-0.9 using FFT transfer function

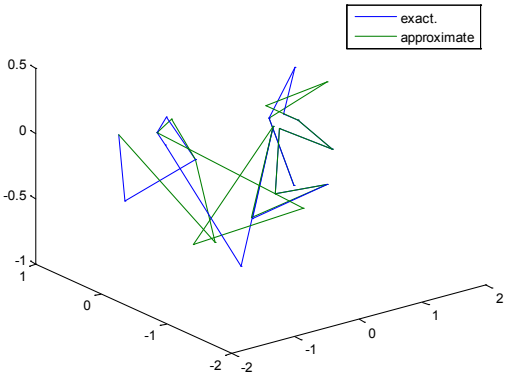

Figure 35. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial points x=-0.1 & y=-0.1 using FFT transfer function

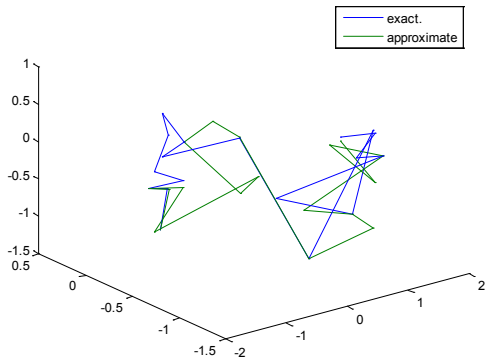

Figure 36. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial points x=-0.4 & y=-0.4 using FFT transfer function

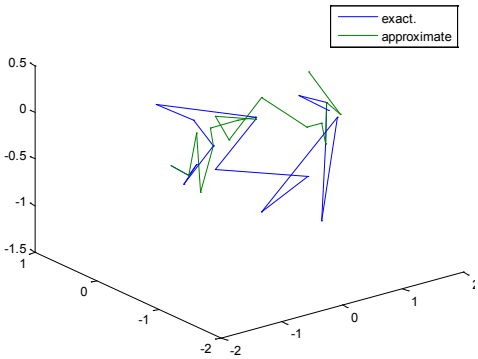

Figure 37. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial points x=-0.7 & y=-0.7 using FFT transfer function

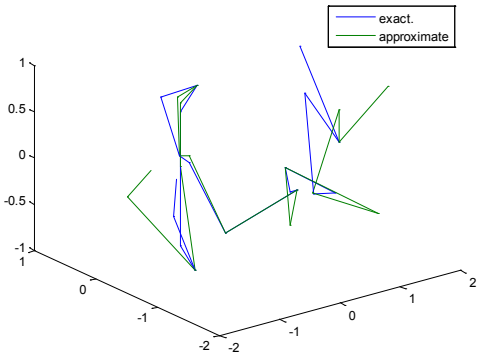

Figure 38. Real and approximate Tinkerbell dynamical map with normal noise (v=0.5) and initial points x=-0.9 & y=-0.9 using FFT transfer function

Table (4). Time and MSE of approximate solution after many training trials with normal noise by using different transfer functions when the variance equal to 40

Transfer function	Initial point								Number points
	-0.1		-0.4		-0.7		-0.9		
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	18
FFT	10.2	3:29:03	14.8	4:39:00	8.32	5:32:05	11.2	3:21:55	
Tansig	11.2	3:05:02	10.1	4:21:29	15.2	3:28:05	9.10	4:34:02	
Logsig	0.0147	5:02:01	0.213	6:30:01	2.91	9:34:31	0.296	5:10:02	

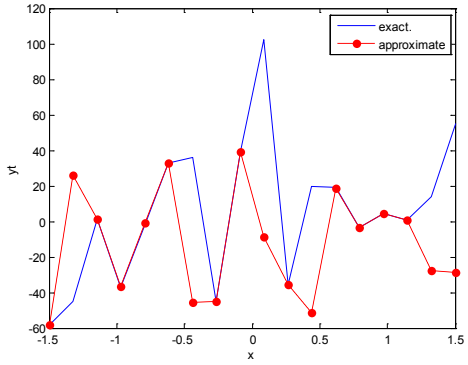

Figure 39. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial point x=-0.1 using FFT transfer function

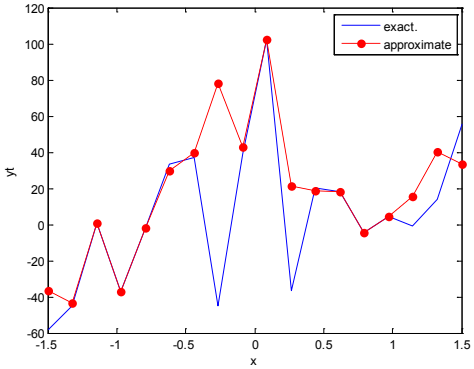

Figure 40. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial point x=-0.4 using FFT transfer function

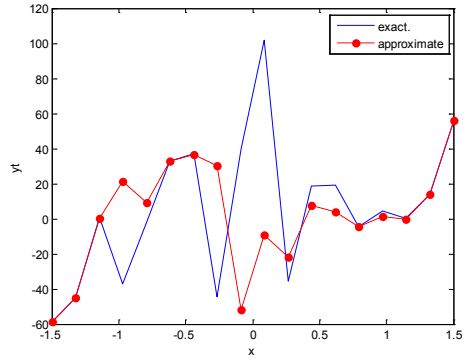

Figure 41. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial point x=-0.7 using FFT transfer function

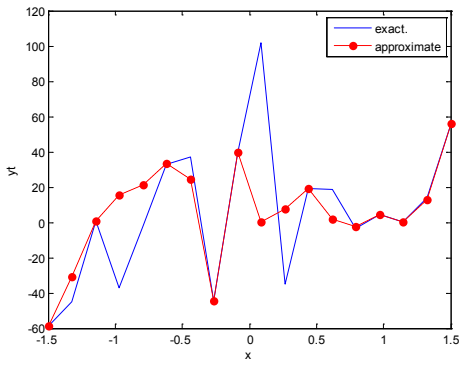

Figure 42. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial point x=-0.9 using FFT transfer function

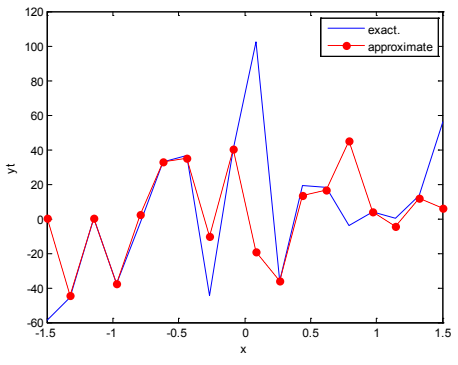

Figure 43. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial point y=-0.1 using FFT transfer function

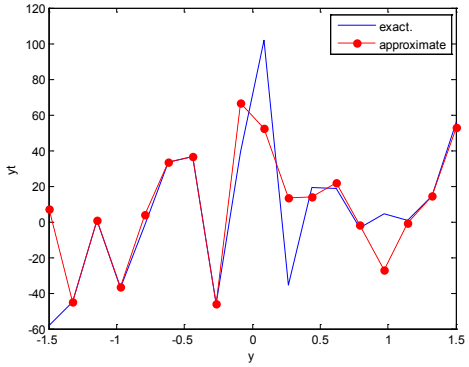

Figure 44. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial point y=-0.4 using FFT transfer function

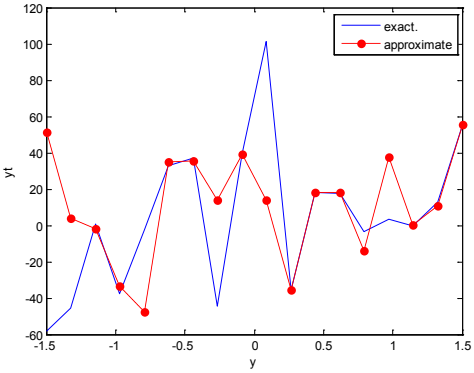

Figure 45. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial point y=-0.7 using FFT transfer function

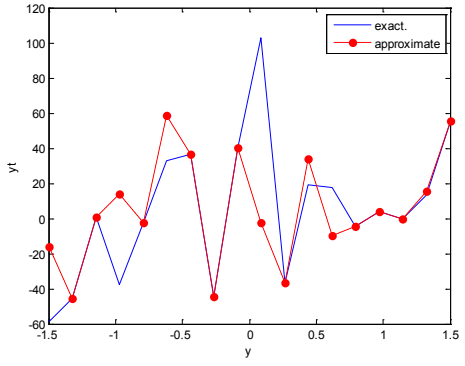

Figure 46. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial point y=-0.9 using FFT transfer function

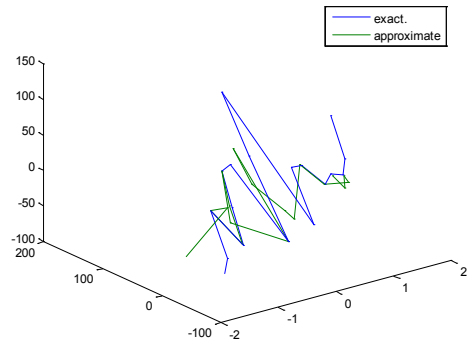

Figure 47. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial points x=-0.1 & y=-0.1 using FFT transfer function

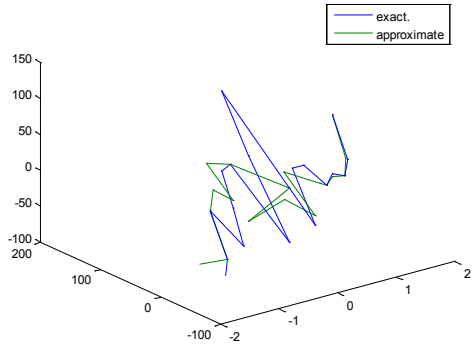

Figure 49. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial points x=0.7 & y=0.7 using FFT transfer function

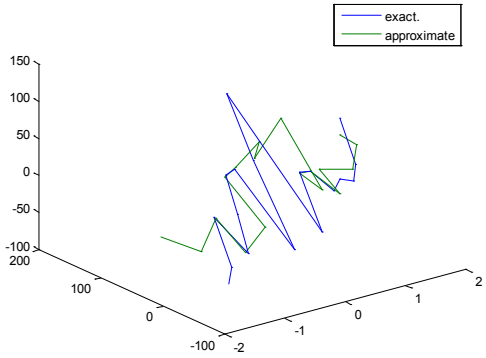

Figure 48. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial points x=-0.4 & y=-0.4 using FFT transfer function

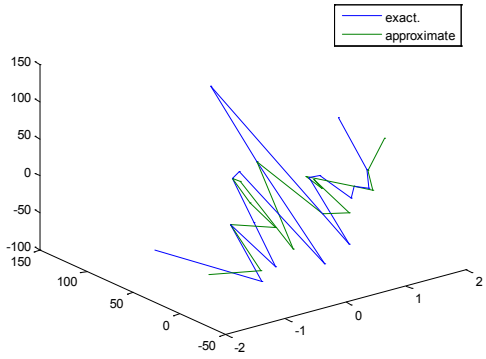

Figure 50. Real and approximate Tinkerbell dynamical map with normal noise (v=40) and initial points x=-0.9 & y=-0.9 using FFT transfer function

Table (5). Time and MSE of approximate solution after many training trials with normal noise by using different transfer functions when the variance equal to 60

Transfer function	Initial point								Number points
	-0.1		-0.4		-0.7		-0.9		18
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	
FFT	0.581	10:28:25	1.73	12:20:05	0.000677	2:30:04	0.26	10:23:43	
Tansig	4.56	2:25:55	3.29	5:23:01	8.3	4:12:40	7.9	3:55:03	
logsig	11.7	3:20:48	4.3	3:55:39	10.3	5:23:10	4.83	4:29:04	

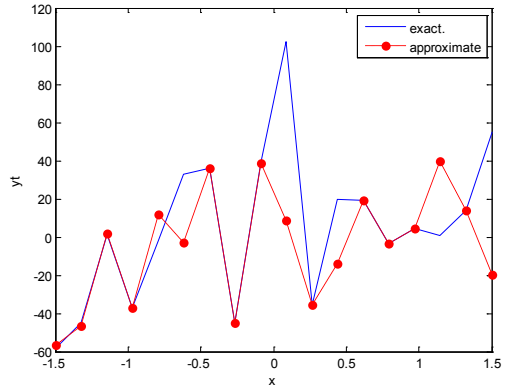

Figure 51. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial point x=-0.1 using FFT transfer function

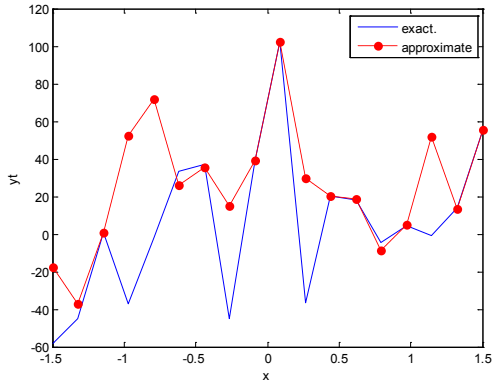

Figure 52. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial point x=-0.4 using FFT transfer function

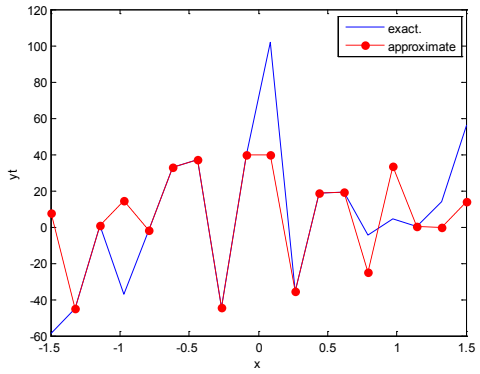

Figure 53. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial point x=-0.7 using FFT transfer function

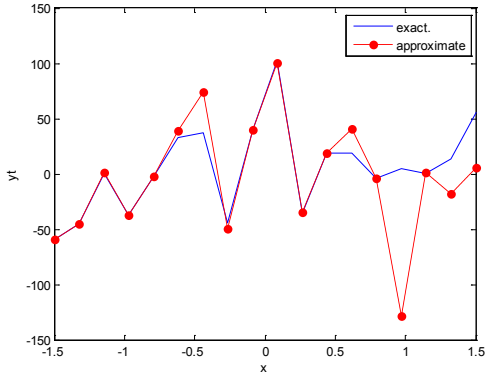

Figure 54. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial point x=-0.9 using FFT transfer function

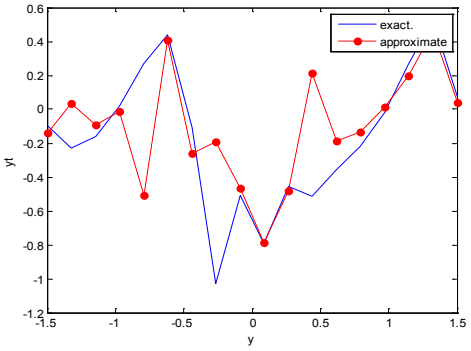

Figure 55. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial point y=-0.1 using FFT transfer function

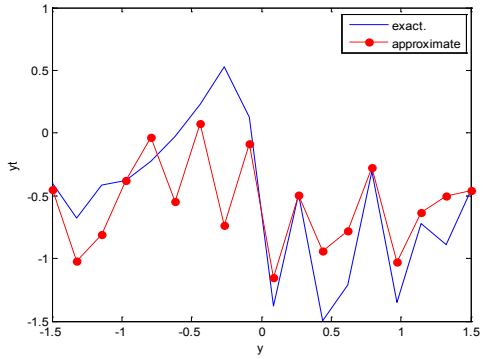

Figure 56. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial point y=-0.4 using FFT transfer function

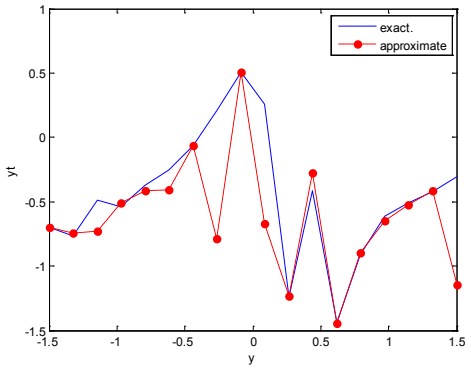

Figure 57. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial point y=-0.7 using FFT transfer function

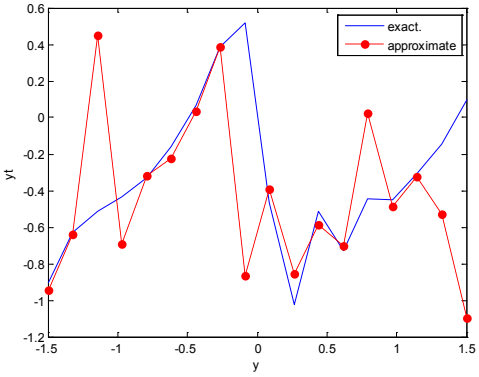

Figure 58. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial point y=-0.9 using FFT transfer function

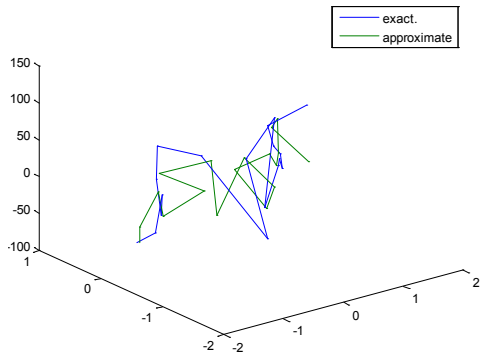

Figure 59. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial points x=-0.1 & y=-0.1 using FFT transfer function

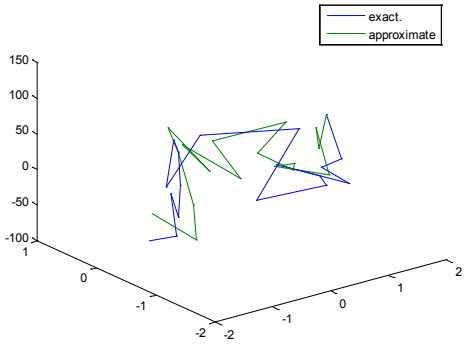

Figure 60. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial points x=-0.4 & y=-0.4 using FFT transfer function

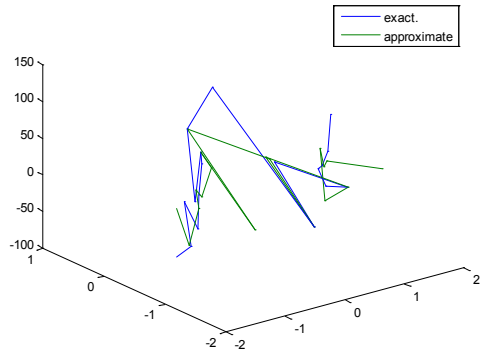

Figure 61. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial points x=-0.7 & y=-0.7 using FFT transfer function

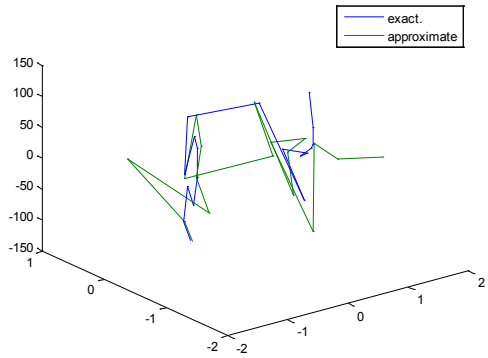

Figure 62. Real and approximate Tinkerbell dynamical map with normal noise (v=60) and initial points x=-0.9 & y=-0.9 using FFT transfer function

5.4. Training with Logistic Noise

The training process is conducted with variances 0.05, 0.5, 40 and 60.

Tables from (6) to (9) contain the results. Figures from (63) to (110) show the Real and approximate Tinkerbell dynamical map with different situation.

Table (6). Time and MSE of approximate solution after many training trials with logistic noise by using different transfer functions when the variance equal to 0.05

Transfer function	Initial point								Number points
	-0.1		-0.4		-0.7		-0.9		
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	18
FFT	6.8e-14	0:00:07	2.39e-14	0:00:04	3.31e-14	0:00:04	1.12e-15	0:00:04	
Tansig	1.05e-13	0:00:15	5.2e-14	0:00:33	5.36e-14	0:00:05	9.4e-15	0:00:07	
Logsig	4.91e-15	0:00:29	6.6e-13	0:00:10	2.19e-14	0:00:08	3.2e-14	0:00:07	

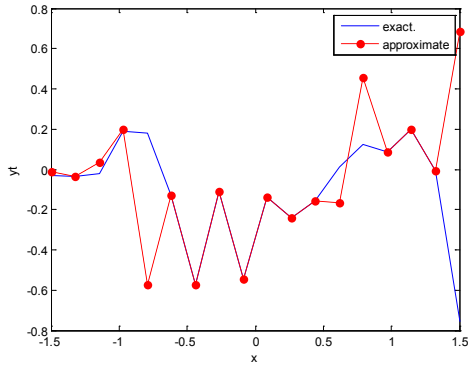

Figure 63. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial point x=-0.1 using FFT transfer function

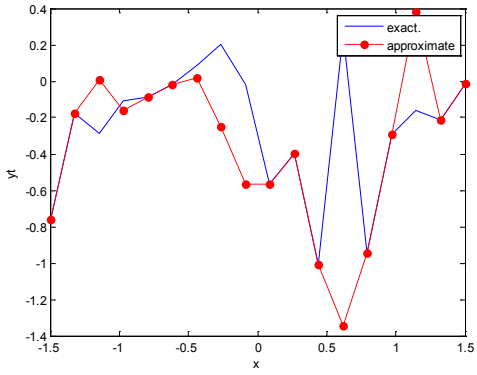

Figure 65. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial point x=-0.7 using FFT transfer function

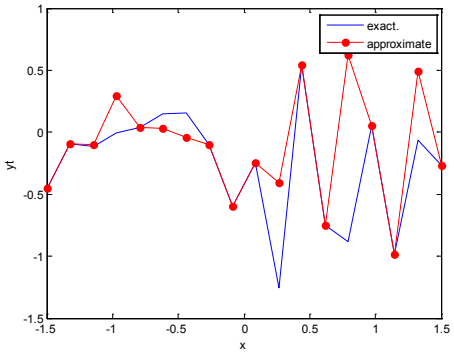

Figure 64. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial point x=-0.4 using FFT transfer function

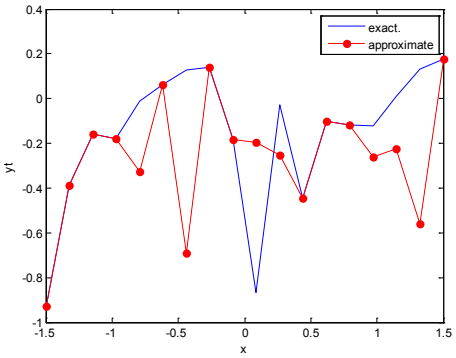

Figure 66. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial point x=-0.9 using FFT transfer function

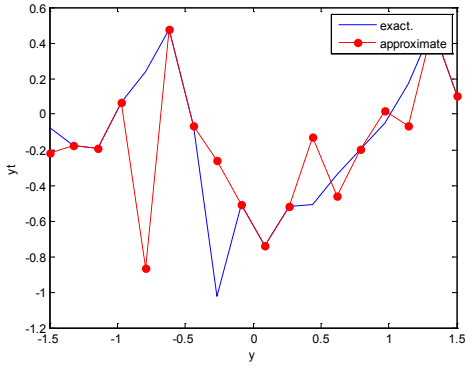

Figure 67. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial point y=-0.1 using FFT transfer function

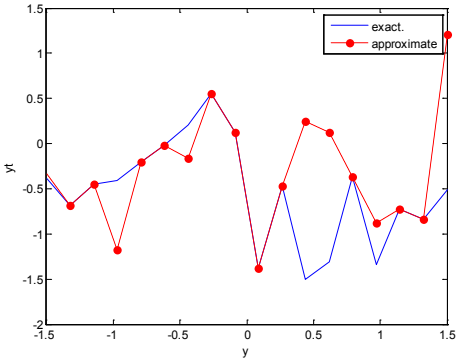

Figure 68. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial point y=-0.4 using FFT transfer function

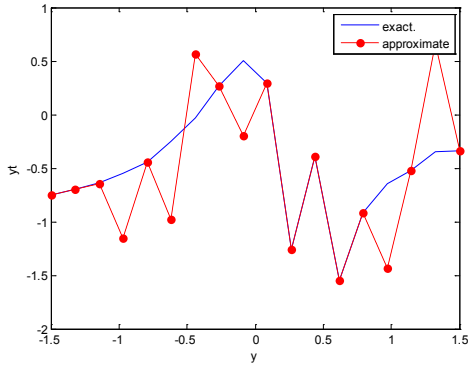

Figure 69. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial point y=-0.7 using FFT transfer function

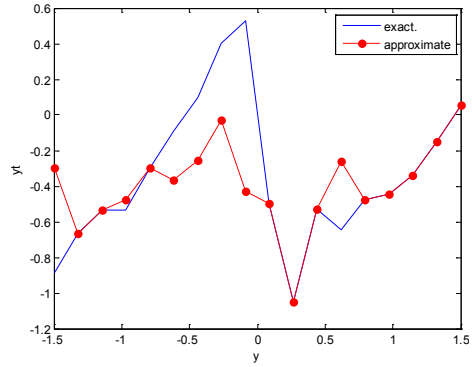

Figure 70. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial point y=-0.9 using FFT transfer function

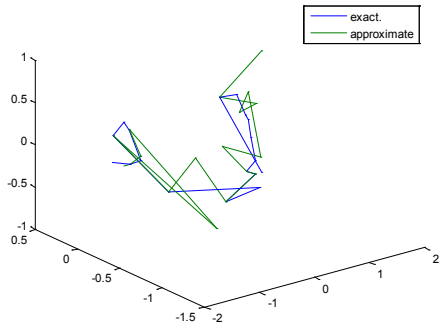

Figure 71. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial points x=-0.1 & y=-0.1 using FFT transfer function

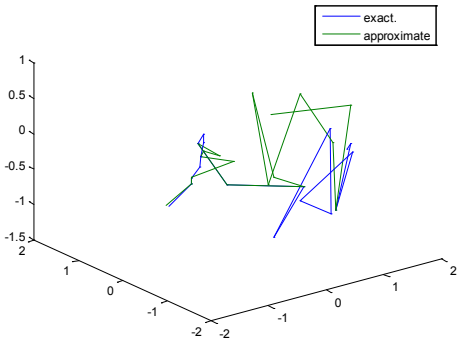

Figure 72. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial points x=-0.4 & y=-0.4 using FFT transfer function

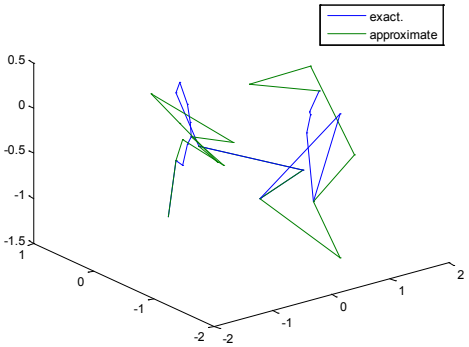

Figure 73. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial points x=-0.7 & y=-0.7 using FFT transfer function

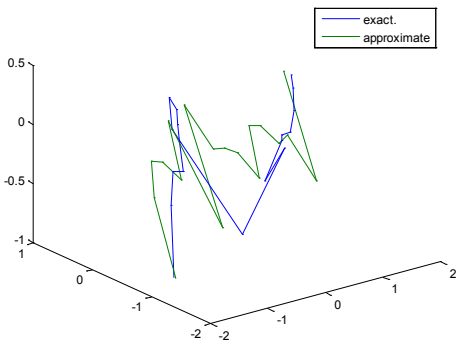

Figure 74. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.05) and initial points x=-0.9 & y=-0.9 using FFT transfer function

Table (7). Time and MSE of approximate solution after many training trials with logistic noise by using different transfer functions when the variance equal to 0.5

Transfer function	Initial point								Number points
	0.1		0.4		0.7		0.9		
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	18
FFT	2.48e-14	0:00:17	6.51e-15	0:00:05	1.04e-14	0:00:07	1.8e-16	0:00:09	
Tansig	1.9e-14	0:00:07	5.32e-16	0:00:07	9.57e-14	0:00:06	1.07e-13	0:00:10	
Logsig	1.05e-14	0:00:10	2.83e-15	0:00:09	1.84e-14	0:00:10	4.3e-15	0:00:07	

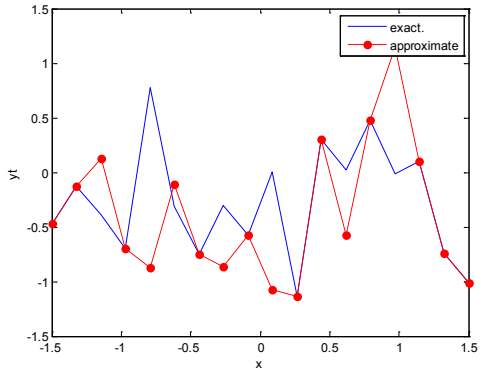

Figure 75. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial point x=-0.1 using FFT transfer function

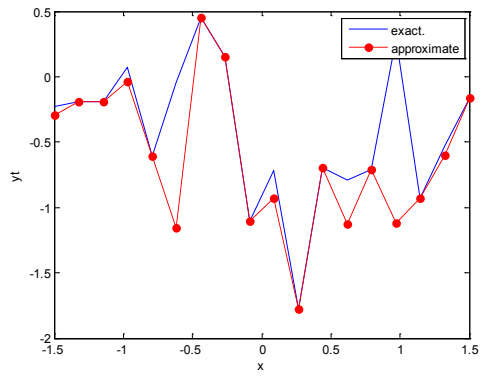

Figure 76. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial point x=-0.4 using FFT transfer function

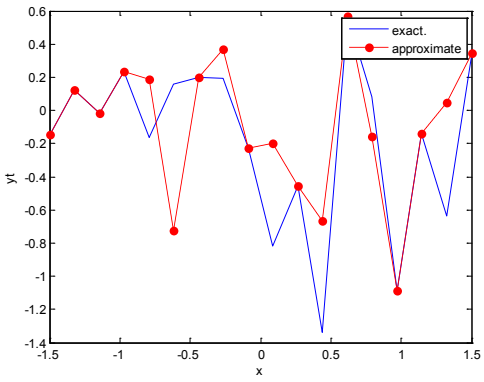

Figure 77. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial point x=-0.7 using FFT transfer function

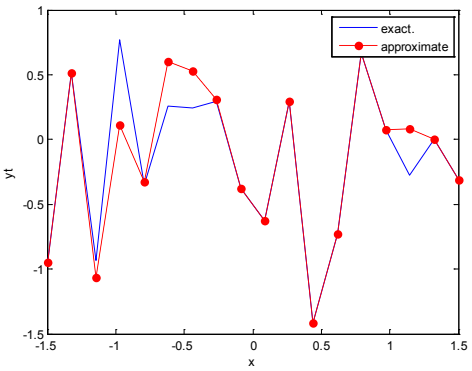

Figure 78. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial point x=-0.9 using FFT transfer function

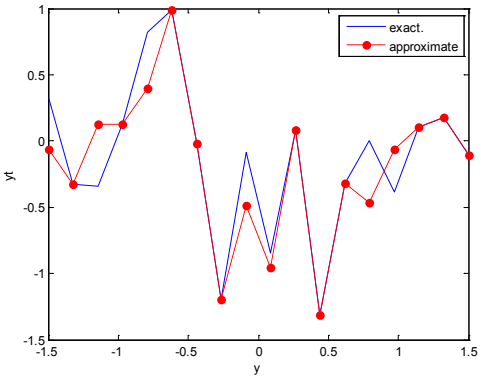

Figure 79. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial point y=-0.1 using FFT transfer function

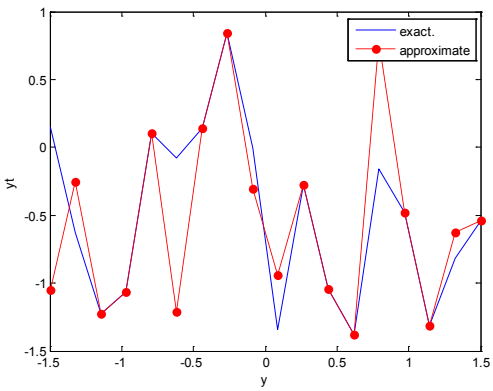

Figure 80. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial point y=-0.4 using FFT transfer function

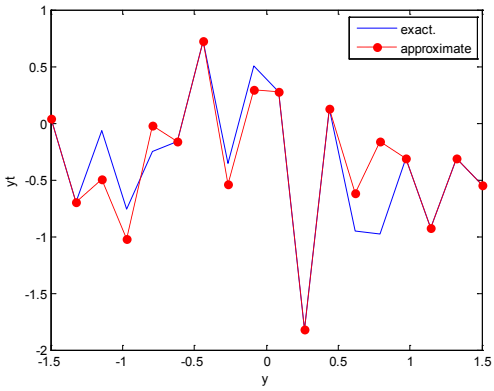

Figure 81. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial point y=-0.7 using FFT transfer function

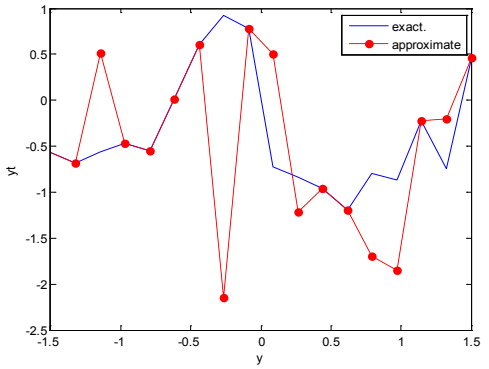

Figure 82. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial point y=-0.9 using FFT transfer function

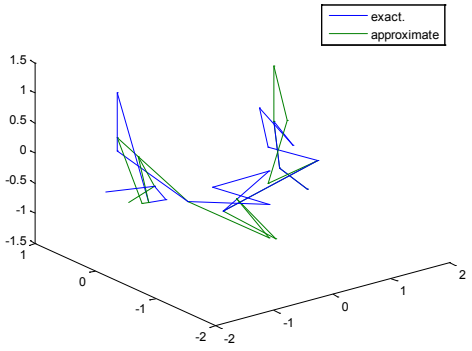

Figure 83. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial points x=-0.1 & y=-0.1 using FFT transfer function

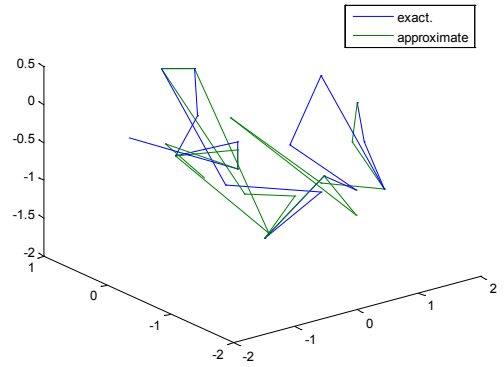

Figure 84. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial points x=-0.4 & y=-0.4 using FFT transfer function

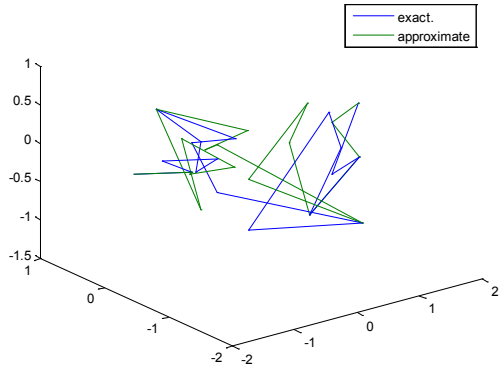

Figure 85. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial points x=-0.7 & y=-0.7 using FFT transfer function

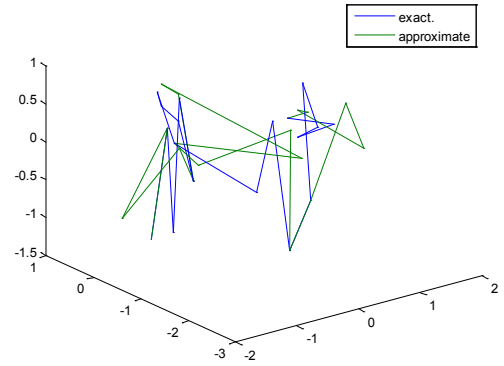

Figure 86. Real and approximate Tinkerbell dynamical map with logistic noise (v=0.5) and initial points x=-0.9 & y=-0.9 using FFT transfer function

Table (8). Time and MSE of approximate solution after many training trials with logistic noise by using different transfer functions when the variance equal to 40

Transfer function	Initial point								Number points
	0.1		0.4		0.7		0.9		
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	18
FFT	0.00213	3:02:12	0.00940	10:07:13	0.028	6:04:30	0.00184	8:45:03	
Tansig	0.104	7:39:28	2.01	6:43:02	0.28	8:43:09	0.13	7:03:56	
Logsig	1.23	6:46:21	0.106	5:34:09	0.849	4:51:49	0.483	4:32:08	

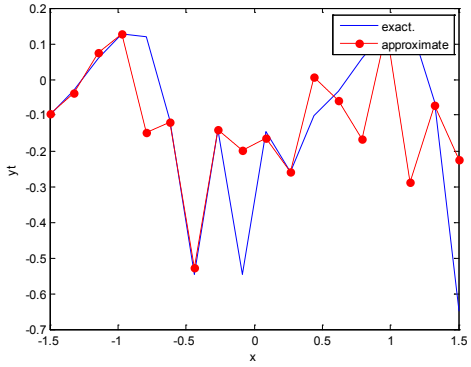

Figure 87. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial point x=-0.1 using FFT transfer function

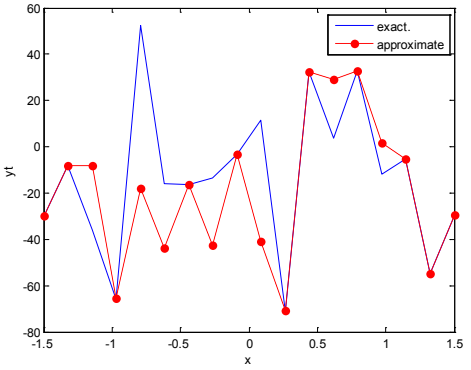

Figure 88. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial point x=-0.4 using FFT transfer function

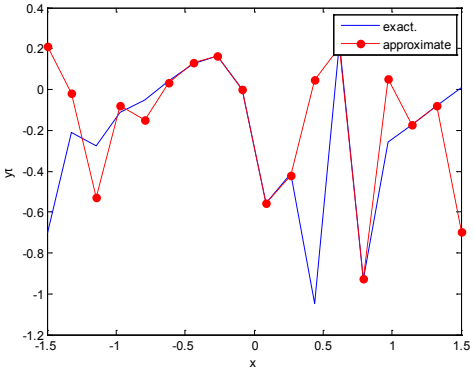

Figure 89. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial point x=-0.7 using FFT transfer function

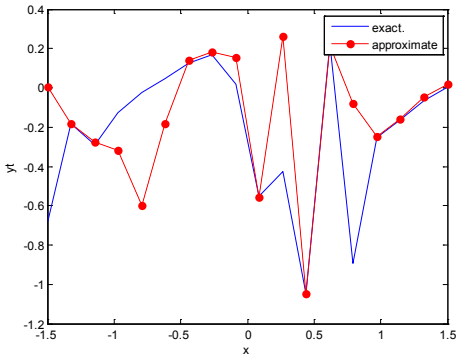

Figure 90. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial point x=-0.9 using FFT transfer function

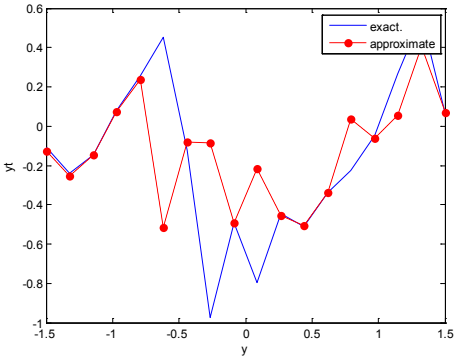

Figure 91. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial point y=-0.1 using FFT transfer function

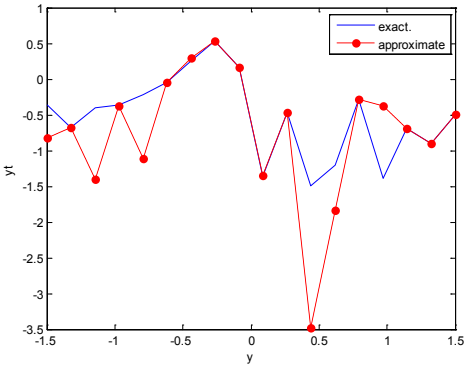

Figure 92. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial point y=-0.4 using FFT transfer function

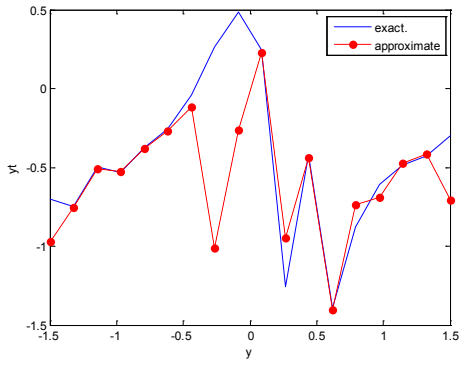

Figure 93. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial point y=-0.7 using FFT transfer function

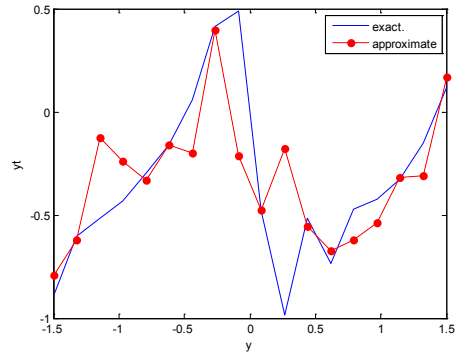

Figure 94. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial point y=-0.9 using FFT transfer function

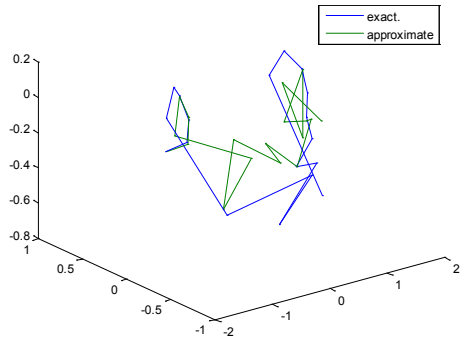

Figure 95. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial points x=-0.1 & y=-0.1 using FFT transfer function

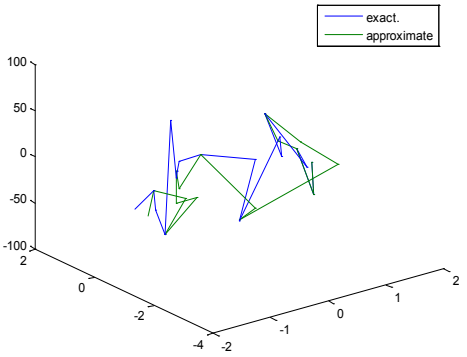

Figure 96. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial points x=-0.4 & y=-0.4 using FFT transfer function

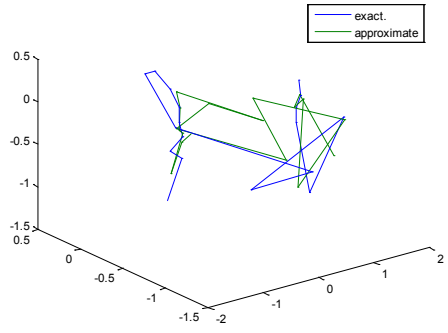

Figure 97. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial points x=-0.7 & y=-0.7 using FFT transfer function

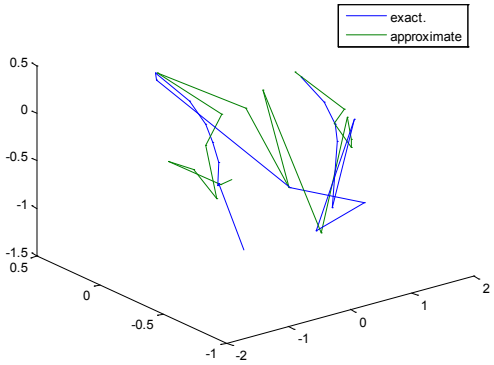

Figure 98. Real and approximate Tinkerbell dynamical map with logistic noise (v=40) and initial points x=-0.9 & y=-0.9 using FFT transfer function

Table (9). Time and MSE of approximate solution after many training trials with logistic noise by using different transfer functions when the variance equal to 60

Transfer function	Initial point								Number points
	-0.1		-0.4		-0.7		-0.9		18
	MSE	Time	MSE	Time	MSE	Time	MSE	Time	
FFT	0.00289	12:49:54	0.00281	10:03:23	0.0098	8:49:37	0.00184	11:28:00	
Tansig	24.3	8:23:01	20.1	7:34:02	15.9	8:49:55	16.3	8:45:02	
Logsig	2.93	7:39:02	0.135	7:26:12	1.28	20:24:19	3.21	6:32:16	

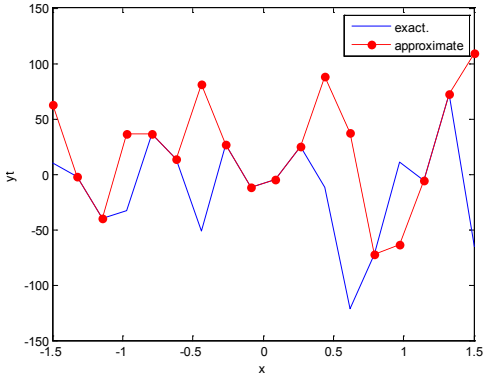

Figure 99. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial point x=-0.1 using FFT transfer function

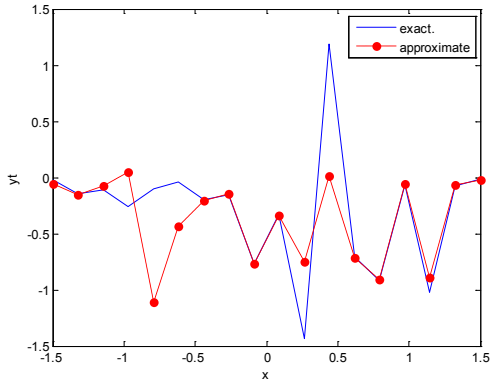

Figure 100. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial point x=-0.4 using FFT transfer function

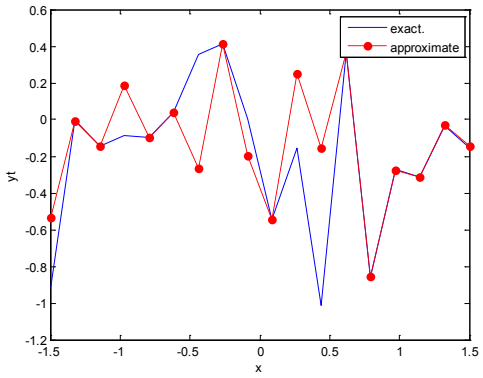

Figure 101. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial point x=-0.7 using FFT transfer function

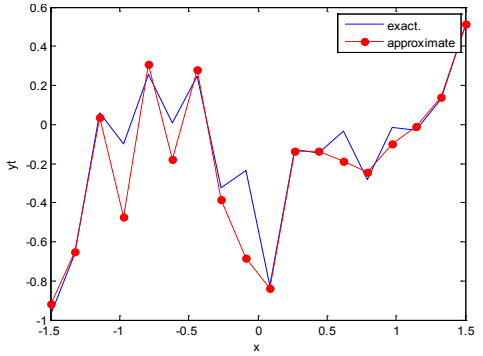

Figure 102. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial point x=-0.9 using FFT transfer function

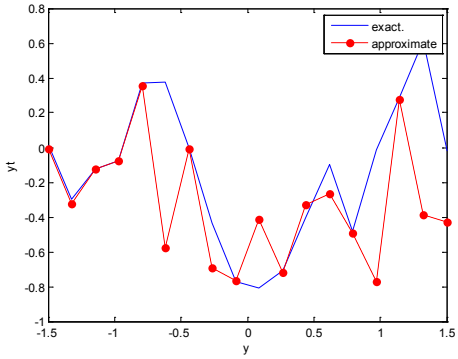

Figure 103. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial point y=-0.1 using FFT transfer function

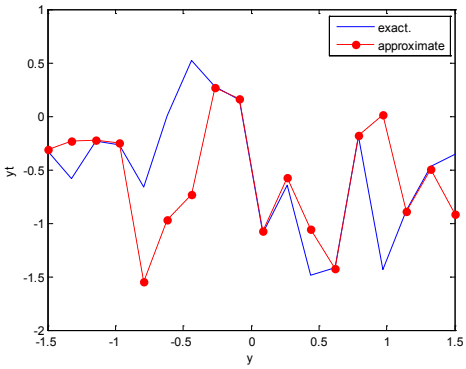

Figure 104. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial point y=-0.4 using FFT transfer function

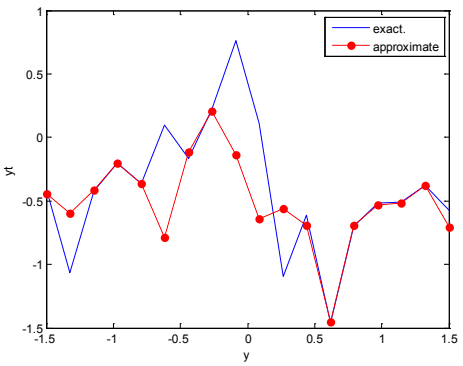

Figure 105. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial point y=-0.7 using FFT transfer function

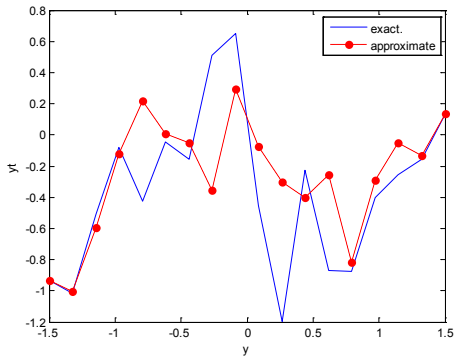

Figure 106. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial point y=-0.9 using FFT transfer function

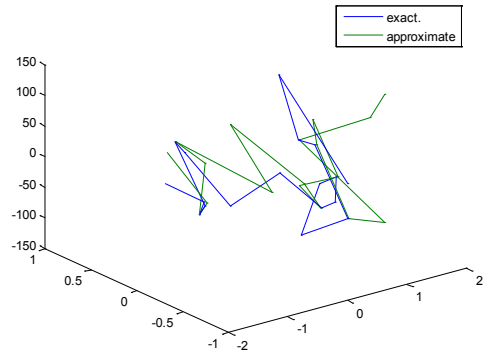

Figure 107. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial points x=-0.1 & y=-0.1 using FFT transfer function

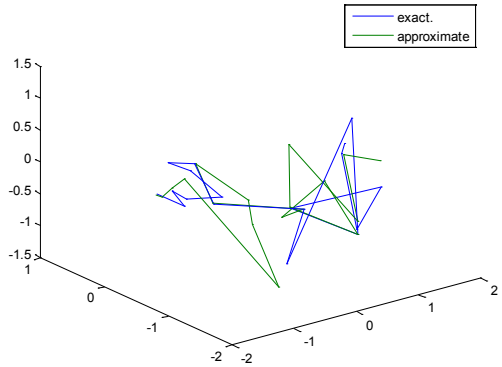

Figure 108. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial points x=-0.4 & y=-0.4 using FFT transfer function

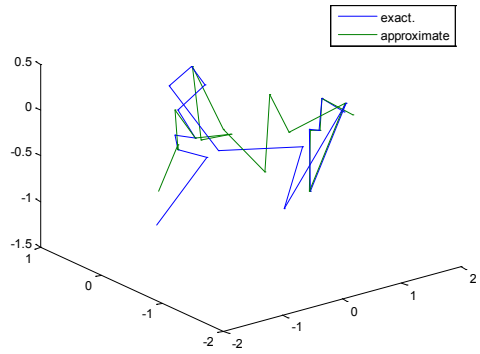

Figure 109. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial points x=-0.7 & y=-0.7 using FFT transfer function

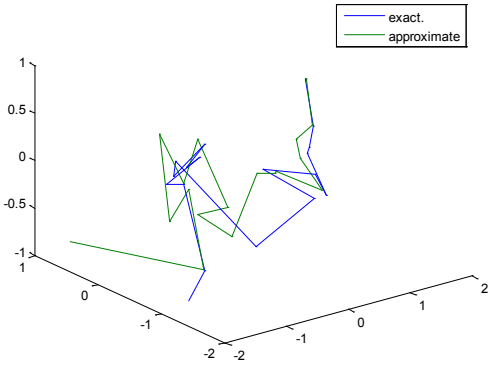

Figure 110. Real and approximate Tinkerbell dynamical map with logistic noise (v=60) and initial points x=-0.9 & y=-0.9 using FFT transfer function

5.5. Results Discussion

The two dimension Tinkerbell with noise normal and logistic, we see that the results of logsig transfer function when the variance (0.05 and 0.5), results of tansig transfer function and the results FFT in tables (2, 3, 6 and 7) are near in Performance and time. When the variance is increased to be 40 and 60, we need more time to get a good performance sometimes up to Hours. The performance FFT transfer functions is better than other transfer Functions see tables (4, 5, 8 and 9).

6. Summary

From the above cases, it is clear that the suggestion of FFT as transfer function in artificial neural network gives excellent results and good accuracy with and without noise in compare with traditional transfer functions (see Tables (1) – (9)) for two dimension dynamical map (Tinkerbell). Therefore, we can conclude that the FFNN with FFT as transfer function which we proposed can handle effectively on the two dimension dynamical maps and provide accurate approximate solution throughout the whole domain and not only at the training set. As well, one can use the interpolation techniques to find the approximate solution at points between the training points or at points outside the training set. The estimation of Tinkerbell dynamical map obtained by trained Ann's offer some advantages, such as:

1. Complexity of computations increases with the increase of the number of sampling points in Tinkerbell dynamical map.
2. The FFNNs with FFT transfer function provides a solution with very good performance function in compare with other traditional transfer functions
3. The proposed FFNNs with FFT transfer function can be applied to two dimension dynamical maps
4. The proposed transfer function FFT gave best rustles especially when the Tinkerbell dynamical map is chaotic and noisy chaotic.
5. In general, the experimental results show that the FFNN side by side FFT transfer function which proposed can handle effectively Tinkerbell dynamical map and provide accurate approximate solution throughout the whole domain, because three points the first point neural network computations is parallel the second point FFT analysis of data and computations are parallel and third point FFT transfer function returns differences in data to sources original related homogeneity data and sectors of the work.

Some future works can be recommended. These works is as follows

1. Using networks with three or more hidden layers.
2. Using the architecture feedback neural network.
3. Increase the neurons in each hidden layer.
4. Use anther random variables as noise on Tinkerbell dynamical map.
5. Choose initial weights to be distributed as different random variables.
6. We recommended using FFT as transfer function in practical application.

REFERENCES

[1] Arar, S. (2017) “An Introduction to the Fast Fourier Transform”, https://www.allaboutcircuits.com/technical -articles/anintroduction-to-the-fast-fourier-transform/ 2017.

[2] Bailer-Jones, C., MacKay, D. and Withers, P. J. (1998) “A recurrent neural network for modelling dynamical systems,” Network: Computation in Neural Systems, vol. 9, no. 4, pp. 531–547.

[3] Bakker, R., Schouten, J., Giles, C. Takens, F. and van den Bleek, C. (2000) “Learning chaotic attractors by neural networks”, Neural Computation, vol. 12, no. 10, pp. 2355–2383.

[4] Berkooz, G., Holmes, P. and Lumley, J. (1993) “The proper orthogonal decomposition in the analysis of turbulent flows,” Annual Review of Fluid Mechanics, vol. 25, no. 1, pp. 539–575.

[5] Billings, S., Jamaluddin, H. and Chen, S. (1992) “Properties of neural networks with applications to modelling nonlinear dynamical systems,” International Journal of Control, vol. 55, no. 1, pp. 193–224.

[6] Broomhead, D. and King, G. (1986) “Extracting qualitative dynamics from experimental data,” Physica D: Nonlinear Phenomena, vol. 20, no. 2-3, pp. 217–236.

[7] Brunton, S. Proctor, J. and Kutz, J. (2016) “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”, Proceedings of the National Academy of Sciences of the United States of America, vol. 113, no. 15, pp. 3932–3937.

[8] Burrus, C. and Johnson, S. (2012) “Fast Fourier Transforms”, http://cnx.org/content/col10550/1.22.

[9] Chakraborty, K., Mehrotra, K., Mohan, C. and Ranka, S. (1992) “Forecasting the behavior of multivariate time series using neural networks,” Neural Networks, vol. 5, no. 6, pp. 961–970.

[10] Chorin, A. and Hald, O. (2009) “Stochastic Tools in Mathematics and Science, vol. 3 of Surveys and Tutorials in the Applied Mathematical Sciences”, Springer.

[11] Cooly, J. and Tukey, J.W. (1965) “An Algorithm for the Machine Calculation of Complex Fourier Series”, Mathematics of Computation, 19 (90), 297-301.

[12] Duriez, T., Brunton, S. and Noack, B. (2017) “Machine Learning Control – Taming Nonlinear Dynamics and Turbulence”, Springer.

[13] Elanayar, V. and Shin, Y. (1994) “Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems,” IEEE Transactions on Neural Networks, vol. 5, no. 4, pp. 594–603.

[14] Elman, J. (1990) “Finding structure in time,” Cognitive Science, vol. 14, no. 2, pp. 179–211.

[15] Galushkin, I. (2007) "Neural Networks Theory", Berlin Heidelberg.

[16] Hornik, K. (1991) “Approximation capabilities of multilayer feedforward networks,” Neural Networks, vol. 4, no. 2, pp. 251–257, 1991.

[17] Jing Z., Yuan S. and Jiang T. " Bifurcation and chaos in the Tinkerbell map", International Journal of Bifurcation and Chaos, 2011.

[18] Karpathy, A.,Toderici, G., Shetty, S., Leung, T., Sukthankar, R. and Fei-Fei, L. (2014) “Largescale video classification with convolutional neural networks,” in Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 1725–1732, Columbus, OH, USA.

[19] Koskela, T. Lehtokangas, M., Saarinen, M. and Kaski, K. (1996) “Time series prediction with multilayer perceptron, FIR and Elman neural networks,” in Proceedings of the World Congress on Neural Networks, pp. 491–496, Citeseer.

[20] Kuschewski, J., Hui, S. and Zak, S. H. (1993) “Application of feedforward neural networks to dynamical system identification and control,” IEEE Transactions on Control Systems Technology, vol. 1, no. 1, pp. 37–49.

[21] Lin, H., Chen, W. and Tsutsumi, A. (2003) “Long-term prediction of nonlinear hydrodynamics in bubble columns by using artificial neural networks,” Chemical Engineering and Processing: Process Intensification, vol. 42, no. 8-9, pp. 611–620.

[22] Mahdi, O. and Tawfiq, L. (2015) "Design Suitable Neural Networks to Solve EigenValue Problems and It′s Application", M.Sc. Thesis, College of Education Ibn AL-Haitham, University of Baghdad, Iraq.

[23] Mangan, N. Brunton, S., Proctor, J. and Kutz, J. (2016) “Inferring biological networks by sparse identification of nonlinear dynamics”, IEEE Transactions on Molecular, Biological and Multi-Scale Communications, vol. 2, no. 1, pp. 52–63.

[24] Miyoshi, T., Ichihashi, H., Okamoto, S. and Hayakawa, T. (1995) “Learning chaotic dynamics in recurrent RBF network,” in Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 1, pp. 588–593, Perth, WA, Australia.

[25] MacKay, (1992) "Neural Computation", Vol. 4, No. 3, pp 415-447.

[26] Narendra, K., and Parthasarathy, K. (1990) “Identification and control of dynamical systems using neural networks,” IEEE Transactions on Neural Networks, vol. 1, no. 1, pp. 4–27.

[27] Narendra, K. and Parthasarathy, K. (1992) “Neural networks and dynamical systems,” International Journal of Approximate Reasoning, vol. 6, no. 2, pp. 109–131.

[28] Oraibi, Y. and Tawfiq, L. (2013) "Fast Training Algorithms for Feed Forward Neural Networks", Ibn Al-Haitham Jour. for Pure & Appl. Sci., Vol. 26, No. 1, pp.276.

[29] Paez, T. and Hunter, N. (1997) “Dynamical system modeling via signal reduction and neural network simulation,” Sandia National Labs, Albuquerque, NM (United States).

[30] Paez, T. and Hunter, N. (2000) “Nonlinear system modeling based on experimental data,” Technical report, Sandia National Labs., Albuquerque, NM(US); Sandia National Labs., Livermore, CA (US).

[31] Pan, S. and Duraisamy, K. (2018) “Long-Time Predictive Modeling of Nonlinear Dynamical Systems Using Neural Networks”, Hindawi, Complexity, Volume 2018, pp. 1–26.

[32] Parish, E. and Duraisamy, K. (2016) “Reduced order modeling of turbulent flows using statistical coarse-graining,” in 46th AIAA Fluid Dynamics Conference, Washington, D.C., USA.

[33] Polycarpou, M. and Ioannou, P. (1991) “Identification and control of nonlinear systems using neural network models: design and stability analysis,” University of Southern California.

[34] Rubinstein, Y. and Kroese, D. (2007), “Simulation And The Monte Carlo Method”,Wiley, second Edition.

[35] Russakovsky, O., Deng, J., Su, H. (2015) “Imagenet large scale visual recognition challenge,” International Journal of Computer Vision, vol. 115, no. 3, pp. 211–252.

[36] Sato, Y. and Nagaya, S. (1996) “Evolutionary algorithms that generate recurrent neural networks for learning chaos dynamics,” in Proceedings of IEEE International Conference on Evolutionary Computation, pp. 144–149, Nagoya, Japan.

[37] Shumway, R. and Stoffer, D. (2000) “Time series analysis and its applications,” Studies in Informatics and Control, vol. 9, no. 4, pp. 375-376.

[38] Smaoui, N. (1997) "Artificial neural network-based low-dimensional model for spatio-temporally varying cellular flames," Applied Mathematical Modelling, vol. 21, no. 12, pp. 739–748.

[39] Smaoui, N. (2001) "A model for the unstable manifold of the bursting behavior in the 2D navier–stokes flow," SIAM Journal on Scientific Computing, vol. 23, no. 3, pp. 824–839.

[40] Smaoui, N. and Al-Enezi, S. (2004) "Modelling the dynamics of nonlinear partial differential equations using neural networks,"Journal of Computational and Applied Mathematics, vol. 170, no. 1, pp. 27–58.

[41] Tanaskovic, M., Fagiano, L., Novara, C. and Morari, M. (2017) "Data driven control of nonlinear systems: an on-line direct approach," Automatica, vol. 75, pp. 1–10.

[42] Tsung, F. and Cottrell, G. (1995) "Phase-space learning", in Advances in Neural Information Processing Systems, pp. 481–488, MIT Press.

[43] Urbina, A., Hunter, N. and Paez, T. (1998) "Characterization of nonlinear dynamic systems using artificial neural networks,"Technical report, Sandia National Labs, Albuquerque, NM (United States).

[44] Villmann, T., Seiffert, U. and Wismöller, A. (2004) "Theory and Applications of Neural maps", ESANN2004 PROCEEDINGS - European Symposium on Ann, pp.25 - 38.

[45] Wang, Z., Xiao, D., Fang, F., Govindan, R., Pain, C. and Guo, Y. (2018) "Model identification of reduced order fluid dynamics systems using deep learning," International Journal for Numerical Methods in Fluids, vol. 86, no. 4, pp. 255–268.