sensors

Article

A Novel Recurrent Neural Network-Based Ultra-Fast, Robust, and Scalable Solver for Inverting a "Time-Varying Matrix"

Vahid Tavakkoli *, Jean Chamberlain Chedjou and Kyandoghere Kyamakya

Institute for Smart Systems Technologies, University Klagenfurt, A9020 Klagenfurt, Austria; jean.chedjou@aau.at (J.C.C.); kyandoghere.kyamakya@aau.at (K.K.)

* Correspondence: vtavakko@edu.aau.at; Tel.: +43-463-2700-3540

Received: 16 August 2019; Accepted: 11 September 2019; Published: 16 September 2019

Abstract: The concept presented in this paper is based on previous dynamical methods to realize a time-varying matrix inversion. It is essentially a set of coupled ordinary differential equations (ODEs) which does indeed constitute a recurrent neural network (RNN) model. The coupled ODEs constitute a universal modeling framework for realizing a matrix inversion provided the matrix is invertible. The proposed model does converge to the inverted matrix if the matrix is invertible, otherwise it converges to an approximated inverse. Although various methods exist to solve a matrix inversion in various areas of science and engineering, most of them do assume that either the time-varying matrix inversion is free of noise or they involve a denoising module before starting the matrix inversion computation. However, in the practice, the noise presence issue is a very serious problem. Also, the denoising process is computationally expensive and can lead to a violation of the real-time property of the system. Hence, the search for a new 'matrix inversion' solving method inherently integrating noise-cancelling is highly demanded. In this paper, a new combined/extended method for time-varying matrix inversion is proposed and investigated. The proposed method is extending both the gradient neural network (GNN) and the Zhang neural network (ZNN) concepts. Our new model has proven that it has exponential stability according to Lyapunov theory. Furthermore, when compared to the other previous related methods (namely GNN, ZNN, Chen neural network, and integration-enhanced Zhang neural network or IEZNN) it has a much better theoretical convergence speed. To finish, all named models (the new one versus the old ones) are compared through practical examples and both their respective convergence and error rates are measured. It is shown/observed that the novel/proposed method has a better practical convergence rate when compared to the other models. Regarding the amount of noise, it is proven that there is a very good approximation of the matrix inverse even in the presence of noise.

Keywords: matrix inversion; time-varying matrix; noise problem in time-varying matrix inversion; recurrent neural network (RNN); RNN-based solver; real-time fast computing

1. Introduction

Matrix inversion is extensively used in linear algebra (e.g., for solving linear equations). Although matrix inversion is already referred to in very ancient books, tremendous attention has been devoted to it (by scientists) mainly since the 17th century. The interest devoted to matrix inversion has led to the development of various methods, concepts, and algorithms for solving linear equations [1]. Solving matrix inversion is very useful in engineering, physics, and other natural sciences [2]. Solving a real-time/online matrix inversion is part of mathematics and control theory. It finds important applications in various areas such as traffic simulation and/or online control in the frame of intelligent transportation systems, robotics (e.g., for kinematics and inverse kinematics), communications [3], machine learning [4], smart/complex antennas (MIMO) [5–7], Field Programmable Gate Array (FPGA) [8,9], signal processing [10], image processing [11,12], and robotics [13–15], etc.

Also, the matrix inversion is very useful in several decision-making algorithms. There are plenty of different heuristics and goal-programming algorithms which are using linear relationships to solve problems and they often try to solve those problems via matrix inversion; this is the case, for example, in social media networks where it can help to speed up the ranking amongst different categories [16,17].

Matrix inversion is further widely used and is of critical importance in various processing contexts in smart sensors. In some cases, the accuracy of certain measurements can be significantly improved by involving matrix inversion. For example, a real-time matrix inversion is used to ensure high-precision in multi parameter sensing while using the so-called fiber Bragg grating (FBG)-based sensors [18]. Thereby, in that related approach, the sensor functionality is approximated with a linear system and the problem is thus solved through linear system of equations. Also, this last-mentioned approximation has other advantages like the capability of re-constructing lost information, a capability used, for example, in the context of so-called compressed sensing [19,20].

In general, the measurements of different physical quantities related to a dynamical system can be explained as linear measurement equations or multi-variate sensing processes [21]. The measurement of those/such mentioned quantities often requires a real-time computing of matrix inversions. Further, the relationship between different measurements can also be used for calibrating purposes for the related sensors [22].

Finally, matrix inversion used in sensors related processing processes does also provide the capability to significantly reduce noise in measurements. For example, just for illustration, matrix inversion is helping to ignore noisy measurement in spatially distributed sensors [23].

To fulfil the expected role in various applications, different methods have been developed to achieve both fast convergence and higher accuracy of the matrix inversion related calculations. Some of the most famous methods are the following: elimination of variables, Gaussian elimination (also known as row reduction), lower-upper (LU) decomposition, Newton's method, eigenvalue decomposition, Cholesky decomposition, and many other methods.

Generally, one can categorize matrix inversion methods into two different groups: (a) the recursive (or iterative) methods; and (b) the direct methods [7,24–26].

The first group encompasses methods like Gauss–Seidel or gradient descent. The initial condition (or starting point) is provided and each step uses the last value to calculate the new value. In each iteration, the solution approximation becomes better until the desired accuracy is reached [27,28].

On the other hand, direct methods like Cholesky or Gaussian elimination typically compute the solution in finite number of iterations. They can find the exact solution if there exists no rounding error [29]. Those analytical methods normally have a minimum arithmetic complexity of $O(n^3)$ (provided those algorithms are implemented on a single CPU). For parallel systems, this arithmetic complexity is different, as the algorithm should be changed in a way to fit for efficient processing on a parallel system architecture. Hereby, depending on the parallelizability potential of the algorithm, the present number of parallel cores will affect the final effective complexity. However, in real benchmarking implementations, most parallel implementations of algorithms do require the transfer of large amounts of data amongst processors and thereby this communication need does lead to a loss of efficiency of the algorithm with respect to speeding-up capability in presence of multiple cores. Therefore, implementing the same algorithm on a parallel system is very inefficient (i.e., speed-up). On the other hand, in practice, the noise problem is a very serious problem and the related denoising process is an expensive one, which can provoke a violation of the systems' real-time property. The need for developing a new concept to solve this concern is therefore sufficiently justified.

For answering the above described parallelizability problem, several parallel concepts have been introduced, one good example being the so-called cellular neural network (CNN). It has been introduced by Leon Chua in 1988 in the seminal paper entitled: "Cellular Neural Networks: Theory" [30]. In 1993, a further article entitled: "The CNN Universal Machine: An Analogic Array Computer", was published by Tamas Roska and Leon Chua, where the first analog CNN processor was presented [31]. Since then many different articles have been published to show the applicability of CNN processors for various usages.

In this article, we use a more generic concept which is called recurrent neural network (RNN), which is basically a more general neural network family to which CNN belongs. Essentially, they (i.e., RNN) are building elements of the so-called dynamic neural network (DNN) [32]. The parallel computing nature of RNN and the inherent fast processing speed while solving problems make it a very good basic brick of a novel concept for efficiently solving differential problems on multiple cores [33].

An RNN processor, similarly to its ancestors (artificial neural networks), is a set of cells, which do have mathematical relations (i.e., coupling) with their respective neighbors. The dynamic property of each cell is expressed by one differential equation. Furthermore, the dynamic property of the network can be customized for solving a given ODE (ordinary differential equation) by appropriate values of ODE's parameter settings (or coefficients) expressed in the form of matrices called "templates".

Playing with templates does provide the possibility to solve different kind of problems without changing the physical property and/or the architecture of the RNN machine or processor.

The inherent flexibility and the fast processing speed while solving problems with RNN does provide two advantages to solve problems when compared to traditional or competing computational methods or concepts [34–36]. First, we can solve different kind of problems by changing templates. Second, the problem solving can be significantly accelerated (see speed-up) without losing accuracy.

The above-mentioned good features of RNN (e.g., flexibility, speed-up (in presence of multiple cores), etc.) motivate the use of this promising paradigm for an accelerated solving of linear algebraic equations on either one-core or multi-core platforms. This is not a simple conversion as we do face completely different hardware computing frameworks with different parameters to be taken care of. All RNN parameters should be adjusted such that the resulting new dynamical system does converge to the target solution of the problem.

In this paper, we do need to formulate the central problem (i.e., realizing a "matrix inversion") in a manner such that the final state of the RNN system is the solution of the target linear algebraic system of equations; see Equation (1).

$$M\ (t)\ X = I \tag{1}$$

In more detail, we define the linear algebraic equations system as Equation (1). Hereby, $M(t)$ is a $n \times n$ matrix of smooth time varying real numbers and X is a $n \times n$ matrix, I is identity matrix of size $n \times n$. We would like to find X such that Equation (1) is satisfied. Our target is to find corresponding RNN templates, which are such that for any initial value problem (IVP), after a finite number of iterations, our dynamic system (see Equation (2)) converges to a solution X^*, which approximates X.

$$\begin{gathered} \dot{X} = F(X) \\ \forall X \in \mathbb{R}_{n\times n}, \lim_{t\to\infty} MX - I = 0 \end{gathered} \tag{2}$$

In Equation (2), $\dot{X} = F(X)$ is showing an ODE in a general form. The second part of Equation (2) is showing our problem that we wish to convert into the form of an ODE solving.

There exist several published works, in which one has tried to solve similar problems with dynamical systems like recurrent neural networks (RNN) [13,37–43] or artificial neural networks (ANN) [44–47], etc. Although most of those works are also related to RNN, our work and the novel concepts presented in this paper have more potential to reliably provide faster convergence towards the solution of Equation (1).

One of the main differences between our method and other related works, especially RNN concepts, is that the proposed model does need training like this required for most of RNN networks required. Our model is converging to the problem's solution if and only if a correct selection/setting of internal dynamic parameters is done. Therefore, this functionality makes our model completely unique amongst other types of RNN models, which are usually used in machine learning [48].

For training, most of the published related works use common traditional methods like gradient descent [39] to create dynamical systems and do then customize them for an implementation on a target platform.

In this paper we do compare the performance of the own novel method developed with those already implemented traditional methods from the relevant literature. This does provide the basis for a fair judgement of both advantages and disadvantages of each method, old ones versus our new one.

The overall structure of this paper is as follows. Section 2 contains both background and a comprehensive overview of previous dynamic system models used for matrix inversion. Besides, related requirements with respect to the target platform for implementing the dynamic system (RNN, ANN ...) are discussed. In Section 3, we do formulate our problem (i.e., inverting a time-varying matrix) with required restrictions related to RNN. The effectiveness of the developed model for solving the matrix inversion problem will be demonstrated in Section 4. The main proof and interesting result are the one showing the global (fast) convergence of the RNN-based dynamic system to the final solution of Equation (1).

In Section 5, the new method will be used to create a new dynamical system, i.e., an RNN model. The performance of this novel RNN will be compared to that of previous/original concepts.

In the last section (i.e., Section 7), some concluding remarks summarize the quintessence of the key achievements in this work.

2. Related Works of Dynamical Neural Networks

Solving Equation (1) using any dynamic method like the so-called dynamic neural network (DNN) requires creating a dynamic system with an attractor in that system (if and only if Equation (1) has a real solution), which is a fix point equal to the solution of Equation (1). Such systems can be implemented by different ways, but the most famous way of creating such dynamic systems is to use either the gradient descent method, the Zhang dynamics, or the Chen dynamics. All three named methods do essentially use the same concept, but both the second and the third named methods do use smaller step sizes and therefore more memory is required. This can improve/speed-up the convergence of the algorithms. We will be providing (in the next sections) a full explanation of the advantages of each method; afterwards a generic new method for solving Equation (1) will be provided.

2.1. The Gradient Method

This method involves a first-order optimization technique for finding the minimum point of a function. This technique takes steps in the direction of the negative gradient. It is based on the observation of where the multivariate function $F(X)$ decreases the fastest if we choose the negative gradient of $F(X)$ at point a: $-\nabla F(a)$.

$$b = a - \gamma \nabla F(a),\ \forall \gamma \in \mathbb{R}\ and\ \gamma > 0 \tag{3}$$

γ is the step size for updating decision neurons. For small γ, $F(a) \geq F(b)$; with this remark, we can extend the observations by Equation (4).

$$X_{n+1} = X_n - \gamma \nabla F(X_n) \tag{4}$$

That means, by iterating Equation (4), $F(X)$ will be converging to the minimum point of the function F.

The gradient descent (see Equation (5)) can also be described as the "Euler method" for solving ordinary differential equations.

$$\dot{X}(t) = -\gamma \nabla F(X(t)) \tag{5}$$

Gradient descent can be used to solve linear equations. In that case, the problem is reformulated as the quadratic minimization of a function which is then solving it with gradient descent method.

$$F(X) = \frac{1}{2}\|MX - I\|^2 \tag{6}$$

Then, based on Equation (6), we have:

$$\nabla F(X) = M^T(MX - I) \tag{7}$$

By substituting Equation (7) into Equation (5) the following dynamic model is obtained:

$$\dot{X}(t) = -\gamma M^T MX + \gamma M^T I \tag{8}$$

The exact analytical solution of Equation (8) is expressed as the following:

$$X(t) = \left(X(0) - M^{-1}\right) e^{-\gamma M^T M.t} + M^{-1} \tag{9}$$

Whereby the first derivative of $X(t)$ denoted by $\dot{X}(t)$ will be:

$$\dot{X}(t) = -\gamma\left(X(0) - M^{-1}\right)\left(M^T M + \dot{M}^T Mt\right) e^{-\gamma M^T M.t} \tag{10}$$

Equations (8) and (1) converge to the same solution described by the point M^{-1}.

$$\forall \gamma, a_{i,j} \in \mathbb{R}\ and\ \gamma > 0,\ then\ \gamma MM^T > 0$$

This method is also called "gradient-based" dynamics. It can be designed by norm-based energy functions [45,49]. The advantage of this model is its easiness of implementation, but due to the effective factor of convergence which can be seen in Equation (9) it takes (a long) time to converge to the solution of the problem, and this convergence rate has an effect on noise sensitivity of the model as it makes the model more sensitive to the noises. Also, this model is not appropriate for real-time matrixes.

2.2. Zhang Dynamics

There exists another method to create [13,50] a dynamic system converging to the required solution. In this method, the error function is defined in the following way:

$$E(t) = M(t)X(t) - I \tag{11}$$

This means the error will be increasing proportionally to the amount of divergence of X from its true solution. Larger values of the difference will create larger values of the error.

Equation (11) is changing over time in a way such that the result will be the same as Equation (2). This requires that one moves against errors to come closer to the solution. Therefore, one can define the derivation of this function as following:

$$\dot{E}(t) = -\gamma E(t) \tag{12}$$

The solution of Equation (12) can be expressed as Equation (13). This equation is showing how the error function is decreasing exponentially in a reverse direction to the errors. This will force the function to converge to the solution. Using this dynamic system is helping to create a new dynamic system for our problem.

$$E(t) = C\, e^{-\gamma t} \tag{13}$$

By substitution of $E(t)$ and $E'(t)$ into Equation (12) we will obtain a new dynamic system [11].

$$M\dot{X} + \dot{M}X = -\gamma(MX - I) \tag{14}$$

Whereby the solution of Equation (14) expressed as follows:

$$X(t) = Ce^{-\gamma t} + M^{-1} \tag{15}$$

When we make a first derivation of Equation (15) we will have:

$$\dot{X}(t) = -C\gamma e^{-\gamma t} \tag{16}$$

By combining Equations (15) and (16) we derive Equation (14). It is observed again that by increasing t to infinite our dynamic system will be converging to the solution of Equation (1). This model has a very good convergence rate and this will solve the problem of noise sensitivity. On the other hand, it is a model which is much more difficult to implement.

2.3. Chen Dynamics

This method is a combination of the two previously described methods. It assumes matrix M is not changing during time; therefore, the time-derivation of M is zero. If we multiply Equation (8) by M and then sum it with Equation (14) we will obtain a new dynamic system [39]:

$$M\dot{X}(t) = -\gamma M^T M\ (MX(t) - I)M\dot{X}(t) = -\gamma\Big(M^T M + I\Big)(MX - I) \tag{17}$$

Solving Equation (17) lead to the following solution:

$$X(t) = -C\, M^{-1} e^{-(M^T M + I)t} + M^{-1} \tag{18}$$

A derivation of Equation (18) leads to following dynamical system:

$$\dot{X}(t) = C\left(M^T + M^{-1}\right) e^{-(M^T M + I)t} \tag{19}$$

By combining Equation (17) in Equation (18) leads to Equation (19).

If $M^T M > 0$ we can state and confirm that this method has a better convergence rate than the first and second above mentioned ones [39]. As we see in Equation (19) t is multiplied with $(M^T M + I)$ in the exponent term, which produces a larger exponent value than the ones for the previous two methods. This model is better than the previous model. It has a very good convergence rate and its sensitivity to noise is very low. On the other hand, its implementation is difficult as it has more coefficient to calculate with respect to the other methods and it does not fit for a real-time matrix inversion (i.e., for inverting a time-varying matrix).

2.4. Summary of the Main Previous/Traditional Methods

By comparing the properties of the above presented methods, there is one big difference amongst them. Table 1 shows the major differences between those models by using 4 different criteria. The convergence rate refers to the convergence during time during the solving process. The "time-varying time matrix inversion" criteria refers to the ability of a given model to be usable for the case of the inversion of a time-varying matrix. The "implementation" criteria refer to how easy it is to implement a given model on RNN machines/processors. And the last criteria in Table 1 refers to how far a given model is sensitive to noise that is present in the time-varying matrix values. This last-mentioned criterion is very important because it does express the resilience of a given model to noise, which is always present in analog computing signals. Although noise is relatively low in digital systems, digital systems do however introduce a noise-equivalent signal distortion originating from computational rounding of numbers as they are digitally represented and processed with/in a fix-size digital arithmetic.

The Zhang model does have a fixed convergence rate over to time. But the gradient descent model does contain a coefficient which can be changed to influence (increase or decrease) the convergence rate. However, the Chen model is much better than the two previous ones as it does potentially provide a much better convergence rate.

The convergence rate has a direct impact on the noise sensitivity of a given model. Indeed, an increase of the convergence rate does decrease the noise sensitivity. Therefore, the Chen model has highest level of stability with respect to noise, although it is the more complex to implement.

Furthermore, amongst the 3 models listed in Table 1, only the Zhang model offers the capability of solving a time-varying matrix (i.e., a real-time matrix inversion).

Table 1. Comparison of different types of DNN (dynamic neural network) concepts (the traditional ones) for matrix inversion.

Criteria/Method	Gradient NN	Zhang NN	Chen
Convergence rate	$e^{-\gamma M^T M t}$	e^{-t}	$e^{-\gamma(M^T M+I)t}$
Time-varying matrix inversion	not available	Available	not available
Implementation	Easy	Hard	very hard/difficult
Noise sensitivity	High	Low	very low

3. Our Concept: The Novel RNN Method

According to Chen [39], his model is converging to the solution of Equation (1) under any initial value. One can however re-formulate the Chen model [39] as result of following goal function:

$$\min \{Z = \|\left(X - A^{-1}\right)\|^2 + \|(AX - I)\|^2\} \tag{20}$$

We can add another positive statement to Equation (20); thus, we multiply the last term of Equation (20), i.e., the term $((AX - I)^2)$, with the matrix A and we add it again to the function Z.

$$\min \{Z = \|\left(X - A^{-1}\right)\|^2 + \|(AX - I)\|^2 + \|\left(A^T AX - A^T\right)\|^2\} \tag{21}$$

After adding that new term to the right side of Equation (20) and solving Z (according to Equation (21), one does find/obtain the following dynamical system:

$$M\dot{X}(t) = -\gamma(\left(M^T M\right)^2 + M^T M + I)(MX - I) - \dot{M}X \tag{22}$$

The solution of this equation (see Equation (22)) can be expressed as follows:

$$X(t) = -C\cdot M(t)^{-1} e^{-\gamma \int_0^t ((M^T M)^2 + M^T M + I) dz} + M(t)^{-1} \tag{23}$$

In Equation (23), when times goes to infinite the limit of X will converge to $M(t)^{-1}$ and it thereby provides the solution for Equation (1). C is a constant value (a matrix), which is added during/while solving Equation (22); see Table 2 for an illustration. Newly added terms $\left(M^T M\right)^2 + M^T M$ produce a better convergence rate. The main reason of this convergence rate is the positive value of the integral and it provides additional factors when compared to the previous time-varying models. Therefore, by adding more coefficients to the right-hand side of Equation (22), we can obtain the following equation:

$$M\dot{X}(t) = -\gamma\left(\sum_{i=0}^{n}\left(M^T M\right)^i\right)(MX - I) - \dot{M}X \tag{24}$$

Equation (24) is more general and can create the model of Equation (22), which is a specific configuration of it. For this, we just need to set the value of parameter n to 2.

Theorem 1. *For any given nonsingular matrix $M \in \mathbb{R}^{n\times n}$, the state matrix $X(t) \in \mathbb{R}^{n\times n}$, while starting from any (initial value problem) IVP $X(0) \in \mathbb{R}^{n\times n}$, Equation (24) will achieve global convergence to $X^*(t) = M^{-1}(t)$.*

Proof of Theorem 1. Let define $E(t) = X(t) - X^*(t)$ be the error value during the process for finding the solution. If this equation is multiplied by M, it does lead to $M(t)E(t) = M(t)X(t) - M(t)X^*(t)$ or $M(t)E(t) = M(t)X(t) - I$. Thus, a derivation of the error function will lead to $M\dot{E}(t) + \dot{M}E(t) = M\dot{X}(t) - \dot{M}X(t)$. By replacing this in Equation (24) we obtain the following expression:

$$M\dot{E}(t) = -\gamma\left(\sum_{i=0}^{n}\left(M^TM\right)^i\right)M\,E(t) - \dot{M}\,E(t) \tag{25}$$

Let us define the Lyapunov function $\epsilon(t) = ME(t)$, which is always a positive function. The derivative of this function can be obtained as follows:

$$\dot{\epsilon}(t) = E(t)^T M^T M \cdot \frac{dE(t)}{dt} + E(t)^T M^T \frac{dM(t)}{dt} E(t) \tag{26}$$

By replacing Equation (25) into Equation (26) it leads to the following:

$$\dot{\epsilon}(t) = -\gamma E(t)^T \Big(\sum_{i=0}^{n}\left(M^TM\right)^{i+1}\Big)E(t) \tag{27}$$

Hence:

$$\dot{\epsilon}(t) = -\gamma E(t)^T M^T \Big(\sum_{i=0}^{n}\left(M^TM\right)^{i}\Big)ME(t) \tag{28}$$

One can replace middle term $\sum_{i=0}^{n}\left(M^TM\right)^i$ with large enough value of μ therefore:

$$\leq -\gamma\mu\ \|ME(t)\| \leq 0 \tag{29}$$

Thus, it appears that $\dot{\epsilon}(t)$ is always negative; furthers, $\dot{\epsilon}(t) = 0$ if and only if $X(t) = X(t)^*$ is satisfied. Therefore, our differential equation globally converges towards a point (matrix), which is the equilibrium point for this function. □

Equation (30) is the result of an analytical solving of Equation (24). Increasing t in this equation will lead to the solution of the algebraic equation (i.e., of Equation (1)).

$$X(t) = -C \cdot M^{-1} e^{-\gamma \sum_{i=0}^{n} \int_0^t (M^TM)^i dz} + M^{-1} \tag{30}$$

In this equation, C is a constant value (matrix) and it is added during solving differential equation. Obviously, this equation has a much better rate of convergence when compared to the previous implementations and, like previous solutions/concepts, the minimum value of the eigenvectors of M^TM should be positive.

Also, according to the Chen model, if Equation (24) is extended by introducing a monotonically increasing function F where $F(0) = 0$, here again our system will be converged to solution of

Equation (1). Thus, by introducing a function F in the Equation (24) the following new equation will be obtained, see Equation (31):

$$M\dot{X}(t) = -\gamma\left(\sum_{i=0}^{n}\left(M^{T}M\right)^{i}\right)F(MX - I) - \dot{M}X \tag{31}$$

Theorem 2. *For any given nonsingular matrix $M \in \mathbb{R}^{n\times n}$, the state matrix $X(t) \in \mathbb{R}^{n\times n}$, while starting from any IVP (initial value problem) $X(0) \in \mathbb{R}^{n\times n}$ and with a monotonically increasing function F where $F(0) = 0$, Equation (31) will achieve global convergence to $X^{*}(t) = M^{-1}(t)$.*

Proof of Theorem 2. Let define $E(t) = X(t) - X^{*}(t)$ for the error value during the process for finding the solution. If this equation is multiplied by M, it does lead to $M(t)E(t) = M(t)X(t) - M(t)X^{*}(t)$ or $M(t)E(t) = M(t)X(t) - I$. Thus, a derivation of the error function will lead to $M\dot{E}(t) + \dot{M}E(t) = M\dot{X}(t) - \dot{M}X(t)$. By replacing this in Equation (31), we obtain the following expression:

$$M\dot{E}(t) = -\gamma\left(\sum_{i=0}^{n}\left(M^{T}M\right)^{i}\right)F(ME(t)) - \dot{M}\,E(t) \tag{32}$$

Let's define the Lyapunov function $\epsilon(t) = ME(t)$, which is always a positive function. The derivative of this function can be obtained as follows:

$$\dot{\epsilon}(t) = E(t)^{T}M^{T}M\cdot\frac{dE(t)}{dt} + E(t)^{T}M^{T}\frac{dM(t)}{dt}E(t) \tag{33}$$

By replacing Equation (33) into Equation (32) it does lead to:

$$\dot{\epsilon}(t) = -\gamma E(t)^{T}M^{T}\Big(\sum_{i=0}^{n}\left(M^{T}M\right)^{i}\Big)F(ME(t)) \tag{34}$$

Hence:

$$\dot{\epsilon}(t) = -\gamma E(t)^{T}M^{T}\Big(\sum_{i=0}^{n}\left(M^{T}M\right)^{i}\Big)F(ME(t)) \tag{35}$$

One can replace middle term $\sum_{i=0}^{n}\left(M^{T}M\right)^{i}$ with large enough value of μ therefore:

$$\leq -\gamma\mu\; E(t)^{T}M^{T}F(ME(t)) \leq 0 \tag{36}$$

In the last equation, $E(t)^{T}M^{T}F(ME(t))$ is always positive because if $ME(t)$ becomes negative, $F(ME(t))$ also becomes negative and vise-versa.

Thus, it appears that $\dot{\epsilon}(t)$ is always negative; and $\dot{\epsilon}(t) = 0$ if and only if $X(t) = X(t)^{*}$ is satisfied. Therefore, our differential Equation (31) or (32) globally converges towards a point (matrix), which is the equilibrium point for this function. □

By choosing different forms of the function F, one can create various dynamical properties to be expressed by this model.

Examples of functions for F are: sigmoid, linear, square, cubic, arcos, etc. All these functions are suitable to be used in Equation (31) as all of them are increasing monotonic functions and they do all satisfy the $F(0) = 0$ condition (see Figure 1).

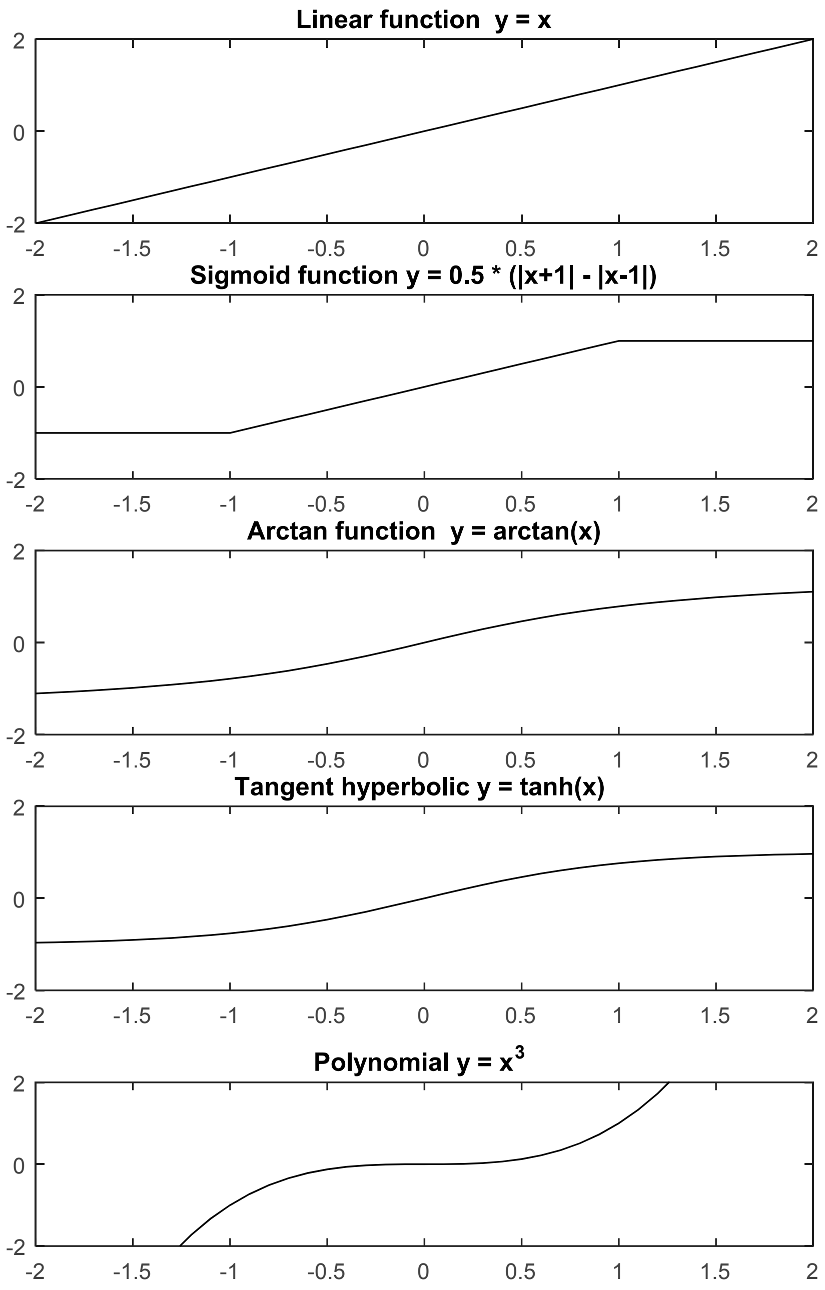


Figure 1. Illustrative examples of monotonic functions which can be used for solving the inversion of a time-varying matrix through Equation (31).

4. Model Implementation in SIMULINK

Equation (24) or Equation (31) can be implemented directly into SIMULINK (see Figure 2). This dynamic model has the components shown in Table 2.

If M is not a time-varying matrix we can simply put zero values in the M′ matrix, otherwise M′ model should be a corresponding derivative of matrix M. Executing the model will result in the following output in SIMULINK, see Figure 2, Figure 3, and Table 2, which is the solution of Equation (1). Therefore, it works well as we expected and gives the solution of Equation (24).

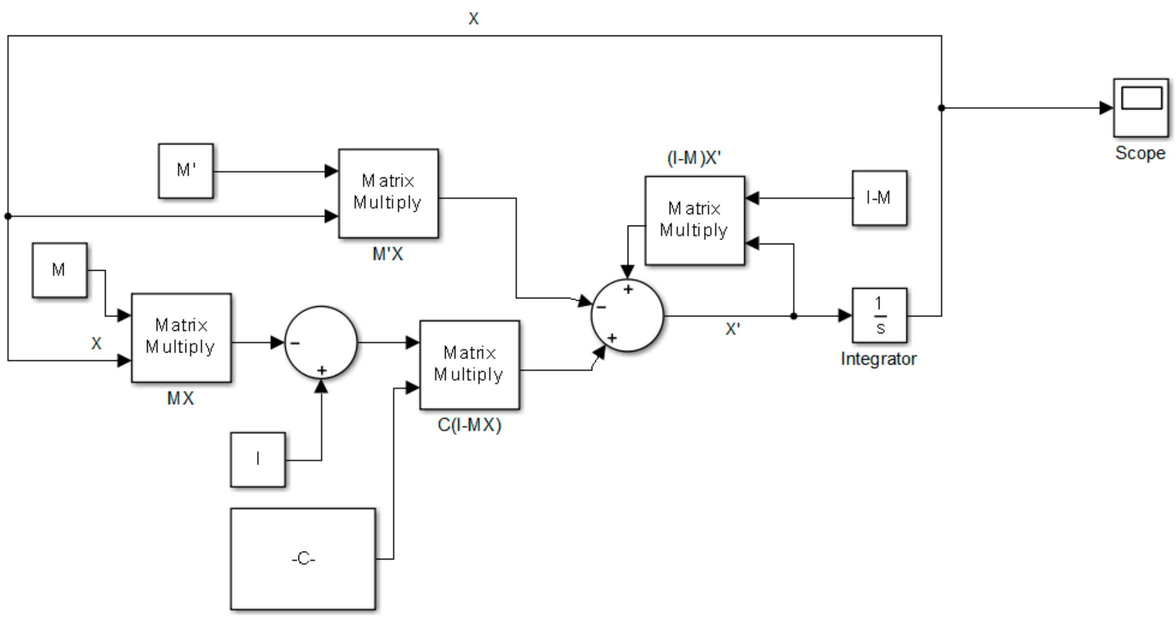
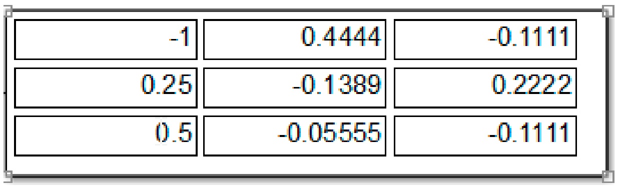


Figure 2. The RNN block diagram corresponding to Equation (24). Note that the matrix to be inverted is M.

Table 2. Components of SIMULINK model to implement Equation (24).

Component	Description
M	$\begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 2 & 6 & 1 \end{bmatrix}$
I	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
I-M	I-M
M'	Time-derivative of M; if M is constant, just put zero values in the matrix M′: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
-C-	C is a weight value matrix equal to $\sum_{i=0}^{n}\left(M^TM\right)^i$ *(see Equations* (24) *and* (31)) For the setting n= 2, we do have the following value for C: $M^TMM^TM + M^TM + I$ Note: If the matrix M is time-varying, C will no more be constant and will also be time-varying.



-1	0.4444	-0.1111
0.25	-0.1389	0.2222
0.5	-0.05555	-0.1111

Figure 3. Result of model simulation (just for illustration) for the matrix M indicated in Table 2 (see first row of Table 2), SIMULINK output.

5. Illustrative Examples

In this section we consider some numerical examples to demonstrate the performance of our RNN implementation for solving a system of linear algebraic equations.

5.1. Illustrative Example 1

Let us define M and V as follows: Here, instead of using the identity matrix in Equation (1) we use a vector V. This does result in a linear system of equations.

$$M = \begin{bmatrix} -3.0755 & 0.5004 & 3.8135 \\ 0.0739 & -2.2382 & -4.7532 \\ -4.7576 & 1.9624 & -1.5879 \end{bmatrix}, V = \begin{bmatrix} 7.920 \\ 7.983 \\ 9.633 \end{bmatrix} \tag{37}$$

Thus, one can find X as following:

$$X = M^{-1}V = \begin{bmatrix} -3.330 \\ -3.305 \\ -0.175 \end{bmatrix} \tag{38}$$

This above given value of X is the target value (ground truth). In the following, we shall see how far how fast the models developed (Equation (24) and/or Equation (31)) do reach well this target value.

M is not singular and therefore can be inverted. Thus, our system of equations has one unique solution. C, as explained in the previous section, is a weight values-matrix which is calculated using the following formula:

$$C = \sum_{i=0}^{n} \left(M^T M\right)^i \tag{39}$$

Increasing the value of n will increase the convergence rate of Equation (24) or Equation (31) or (29). Here, we choose the value n = 3. The corresponding RNN parameters are defined as follows (see Figure 3):

$$\begin{gathered} C = M^T M M^T M M^T M + M^T M M^T M + M^T M + I \\ C = 10,000 \times \begin{bmatrix} 4.6283 & -2.2035 & -2.7439 \\ -2.2035 & 1.3283 & 2.5763 \\ -2.7439 & 2.5763 & 7.5178 \end{bmatrix} \\ X_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{gathered} \tag{40}$$

These parameters are implemented in the previously described model in Simulink (see Figure 3) for numerical simulations and F (for simplicity) is taken as a linear function (Figure 1).

Figure 4 is showing the system simulation results during the time interval $t = [0,\ 0.5]$. The output (value of state vector) at the end of this time interval is the following:

$$X(0.5) = \begin{bmatrix} -3.329 \\ -3.304 \\ -0.1749 \end{bmatrix} \tag{41}$$

This above obtained result shows that the output of our system model is very close to the analytical solution and the difference is small and approximately 0.001, which can furthermore be corrected but adjusting the step size.

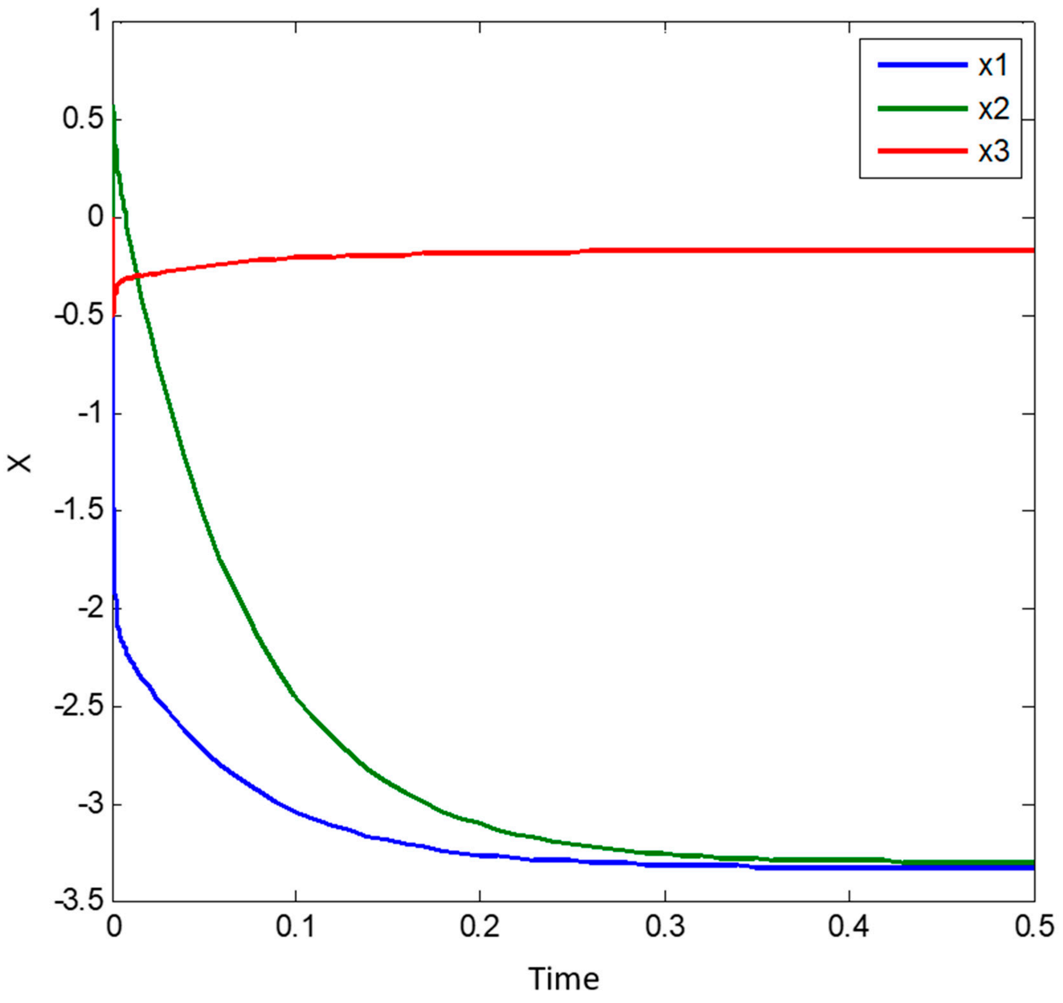


Figure 4. Result of our model simulation for solving the illustrative example with respect to solving an algebraic system of linear equations. Please note in this case we have taken n = 3 in Equation (24) or Equation (31).

5.2. Illustrative Example 2

Now we do the same experience for calculating the inverse of a time-varying matrix M. The matrix M is defined as follows:

$$M = \begin{bmatrix} Sin(t) & -Cos(t) \\ Cos(t) & Sin(t) \end{bmatrix} \tag{42}$$

the matrix M is always invertible and it does satisfy the conditions of Equation (1). Therefore, we can use it for testing our dynamic system model.

$$M^T = M^{-1} = \begin{bmatrix} Sin(t) & Cos(t) \\ -Cos(t) & Sin(t) \end{bmatrix} \tag{43}$$

Also, we define F as a linear function (Figure 1).

As we have explained previously (see Table 2), we calculate the weight C again for n = 3 as the following:

$$C = M^T M M^T M M^T M + M^T M M^T M + M^T M + I \tag{44}$$

Now all parameters are introduced in the Simulink model to simulate our system model (see Equation (24)). The simulation is done for $t = [0,\ 2.0]$. Figure 5 does contain benchmarking results for the first element of the time-varying matrix (i.e., the matrix element in first row, first column). Here (see Figure 5) the speed of convergence is compared to that of the original Zhang model. Thereby, for our model we consider different values of the parameter n. One can clearly see that our novel model strongly outperforms the original Zhang model. Table 3 shows the difference very clearly: Here, it is clear that by choosing n = 3 our model will converge to the problem solution 4 times faster than the ZNN model; notice that the ZNN model corresponds to n = 0 in Table 3. Amount of memory for saving this model is very small as we need to save main templates parameter of dynamic model in Equation (24) or Equation (31). Therefore, as see in Table 3, the memory usage is very small and it can be implemented on small computers.

Table 3. Comparison of performance of time varying matrix inversion with change of parameter n of Equation (24) or Equation (31). Please note n = 0 is ZNN model.

Criteria/n	0	1	2	3
Convergence Time to MSE 0.01	8.4	3.8	2.8	1.9
MSE in t = 2.0	3.4	0.4	0.06	0.008
Approximated estimation of memory usage in Bytes	96	128	128	128

5.3. Illustrative Example 3

In the last illustrative experiment, we try to test and analyze the effect of changing the function F in Equation (31) on the convergence rate of our model.

For this test, we take the same parameter setting as in the first experience (see illustrative example 1), but we change the function F and calculate coefficients for $n = 3$. The selected functions for this test are the following ones: linear function ($Y = X$), sigmoid function ($Y = \frac{1}{2}(|X+1| - |X-1|)$, arctangent function ($Y = \arctan(X)$), tangent hyperbolic function ($Y = \tan\mathrm{h}(X)$), and polynomial function ($Y = \mathrm{X}^3$). All other parameters are fixed such to show the correct properties of the functions. As we can see in the results of this simulation experiment (see Figure 6) the polynomial function is showing the very best convergence amongst all considered functions. Therefore, it is evident that selecting the appropriate function can help our model to converge faster to the solution of the matrix inversion problem. It is also evident that higher values of n will perform much better.

Table 4 is showing our simulation result until t = 0.05 for this illustrative example 3. It does clearly show the effect of using different functions on the system/model convergence. Both polynomial and linear functions are the best functions for this task, while both sigmoid and tanh functions are the worst functions to be used in this context for matrix inversion.

Table 4. Performance comparison of different function types used in Equation (31), whereby all are monotonic increasing functions. The polynomial function shows the better convergence compared to the other functions.

Criteria/ Function used in Equation (31) with n = 3	**Linear**	**Sigmoid**	**Arctan**	**Tanh**	**X^3**
MSE at t = 0.05	0.075	7.90	4.58	7.83	0.062

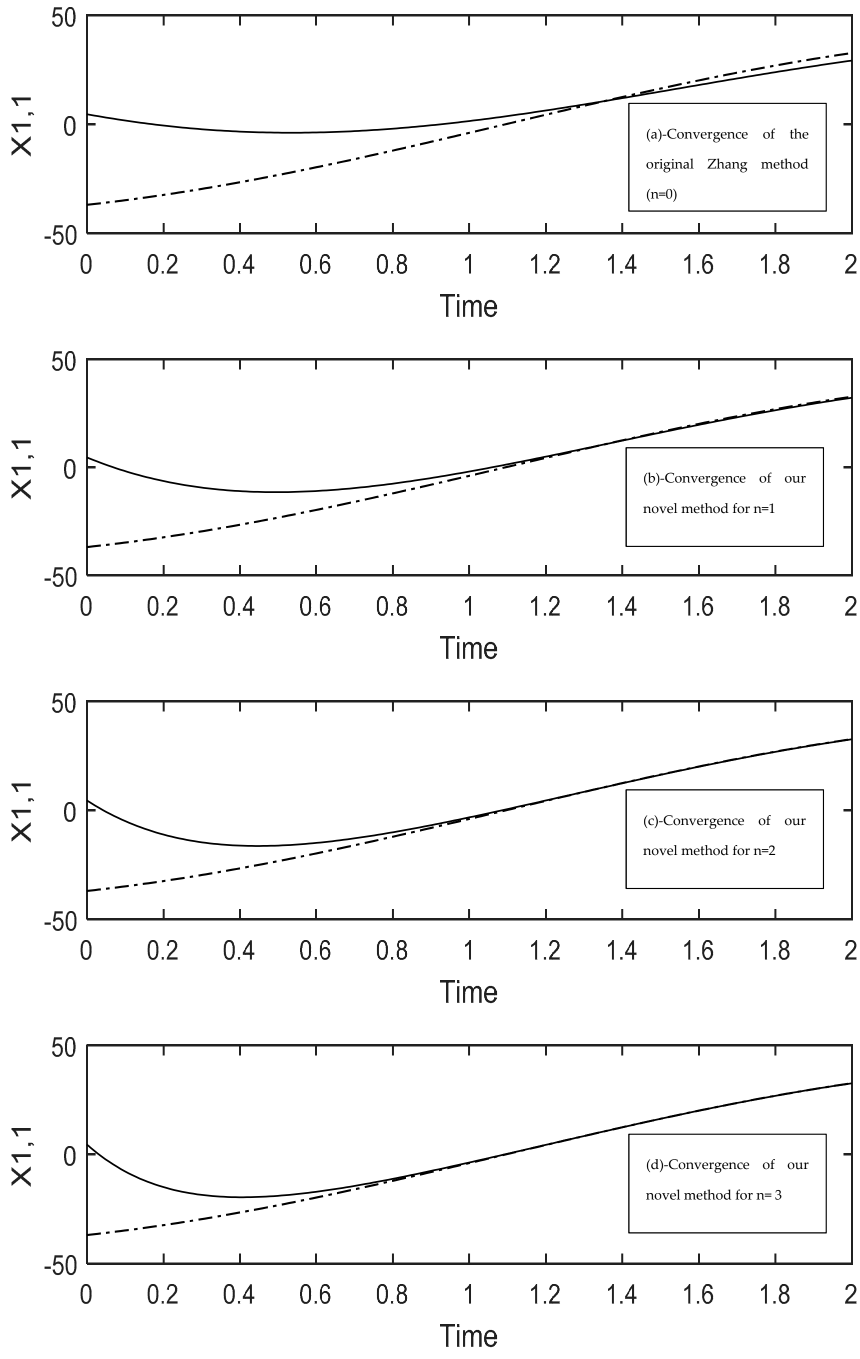


Figure 5. Showing the difference in convergence speed between the original Zhang method and our new developed method (see Equation (24)). The solid lines are the one obtained from the model solving and the dashed lines are the target values. (**a**) convergence of the original Zhang model; (**b**) until (**d**): convergence profiles of our novel method for different values of n. All graphs (a) until (d) are plotted for the first element of the time-varying matrix M (i.e., first row, first column).

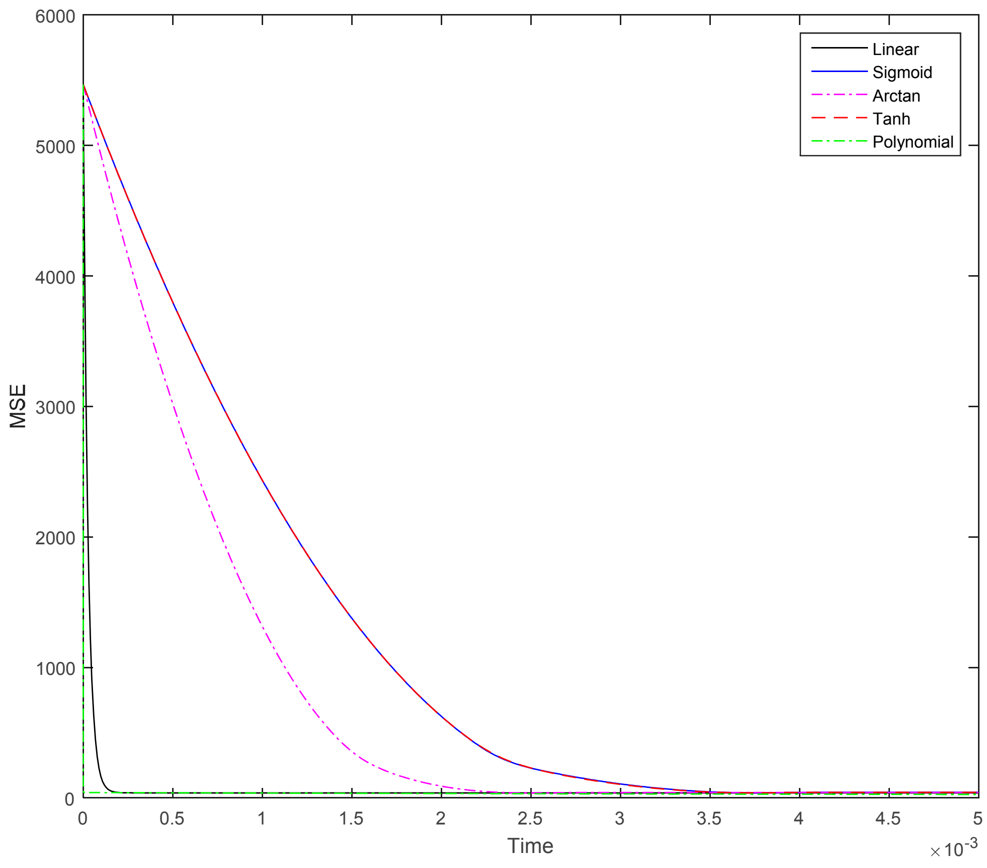


Figure 6. Showing differences in convergence speed while considering different function types for F in Equation (31). Please note that n is 3 for the polynomial function F.

The superiority with respect to convergence speed of the polynomial function F is very evident.

6. Comparison of Our Novel Method with Previous Studies

In this section we compare our novel model with the following related methods which are well-known from the relevant literature: gradient descent method, Zhang dynamics, and Chen dynamics. Hereby, 1000 different static random matrices are generated for use in the experiment. Finally, we then sum up the results obtained as shown in Figure 7. Figure 7 does display how the error converges towards zero when the different models are executed over time; the intention is to hereby illustrate the convergence speed of all considered models. Hereby, the error function is calculated by using the formula $||MX - I||$.

By comparing the different methods with our method (Figure 7), the fastest convergence of our method to the exact solution is observed. Specifically, our method is at least 3 times faster than the Chen method while implemented on CPU (the results of Figure 7 were obtained on CPU). An implementation on a multiple-core platform should display a much higher relative and comparative speed-up performance of our model. Obviously, by using more coefficients (i.e., higher values of n) in Equation (24) we can reach a much better convergence rate.

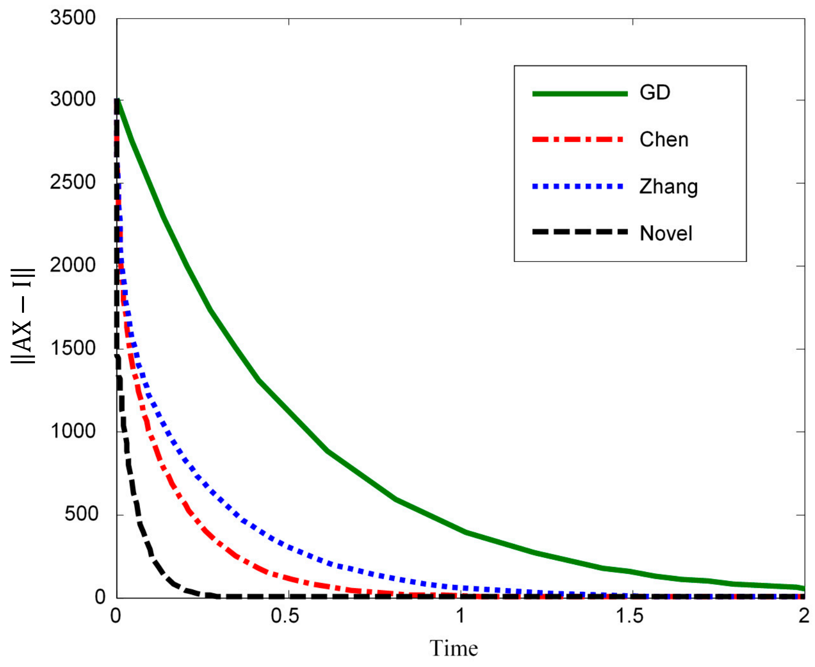


Figure 7. Comparison of different DNN models with respect to convergence speed under the same conditions—benchmarking. In this figure, GD refers to the "gradient descent" method. Our novel method (see novel in the figure; Equation (24)) is calculated for n = 3. The fast convergence of our method to the exact solution is clearly demonstrated. It is also clear that this convergence would be much faster if higher values of n are taken (n > 3).

7. Conclusions

A novel method for solving the very important "matrix inversion" problem has been developed and validated in this paper. Consequently, it can also solve linear algebraic systems of equations. The concept developed is essentially an RNN-based analog computing machine.

We have compared this novel method with other dynamical systems methods from the relevant literature. It has been demonstrated that our novel method does theoretically have an exponential convergence rate towards the exact solution of the problem for any IVP (initial value problem). Also, the convergence rate of this method is much higher than those of other related competing concepts.

It is further possible to customize this novel model in order to enable its implementation on a cellular neural network processor machine. This has previously been done in our previous works.

For validation we have extensively compared our method with other relevant competing methods like gradient descent, Zhang, and Chen methods in one large simulation experiment. Hereby, we have considered the following scenarios: 1000 random matrixes of M are generated and applied in the different dynamical system models for matrix inversion. For each time-point, errors reached are summed-up using the formula $||MX - I||$.

It has been clear that while using our novel method we do visibly reach a significant speed-up even on one single CPU, this compared to the other methods. A use of more coefficients (i.e., higher values for n in model Equations (24) and (31)) will surely result in an even much higher convergence rate. On the other hand, however, if we use more coefficients, we would need more preparation time to create the templates and consume more computing resources for running the (our novel) RNN processor dynamical model. Therefore, it is recommended to reach a balance/tradeoff between convergence rate and number of required coefficients (i.e., value of n).

To finish, the last experiments have demonstrated that using the polynomial function for F in Equation (31) does lead to a clearly much higher convergence rate when compared to the other types of function. Further, the higher the polynomial order, the best.

Author Contributions: Conceptualization, V.T. and K.K.; Methodology, J.C.C. and K.K.; Software, V.T.; Validation, V.T., J.C.C. and K.K.; Formal Analysis, V.T.; Investigation, V.T.; Resources, V.T.; Data Curation, V.T.; Writing-Original Draft Preparation, V.T.; Writing-Review & Editing, J.C.C. and K.K; Visualization, V.T.; Supervision, J.C.C. and K.K.; Project Administration, K.K.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest

References

1. Lumpkin, B. *Algebra Activities from Many Cultures*; J. Weston Walch: Portland, ME, USA, 1997.
2. Song, W.; Wang, Y. Locating Multiple Optimal Solutions of Nonlinear Equation Systems Based on Multiobjective Optimization. *IEEE Trans. Evol. Comput.* **2015**, *19*, 414–431. [CrossRef]
3. Wang, Y.; Leib, H. Sphere Decoding for MIMO Systems with Newton Iterative Matrix Inversion. *IEEE Commun.* **2013**, *17*, 389–392. [CrossRef]
4. Gu, B.; Sheng, V. Feasibility and Finite Convergence Analysis for Accurate On-Line ν-Support Vector Machine. *IEEE Trans. Neural Netw. Learn. Syst.* **2013**, *24*, 1304–1315.
5. Eilert, J.; Wu, D.; Liu, D. Efficient Complex Matrix Inversion for MIMO Software Defined Radio. In Proceedings of the IEEE International Symposium on Circuits and Systems, New Orleans, LA, USA, 27–30 May 2007.
6. Wu, M.; Yin, B.; Vosoughi, A.; Studer, C.; Cavallaro, J.R.; Dick, C. Approximate matrix inversion for high-throughput data detection in the large-scale MIMO uplink. In Proceedings of the ISCAS2013, Beijing, China, 19–23 May 2013; pp. 2155–2158.
7. Ma, L.; Dickson, K.; McAllister, J.; McCanny, J. QR Decomposition-Based Matrix Inversion for High Performance Embedded MIMO Receivers. *IEEE Trans. Signal Process.* **2011**, *59*, 1858–1867. [CrossRef]
8. Arias-García, J.; Jacobi, R.P.; Llanos, C.H.; Ayala-Rincón, M. A suitable FPGA implementation of floating-point matrix inversion based on Gauss-Jordan elimination. In Proceedings of the VII Southern Conference on Programmable Logic (SPL), Cordoba, Argentina, 13–15 April 2011; pp. 263–268.
9. Irturk, A.; Benson, B.; Mirzaei, S.; Kastner, R. An FPGA Design Space Exploration Tool for Matrix Inversion Architectures. In Proceedings of the Symposium on Application Specific Processors, Anaheim, CA, USA, 8–9 June 2008; pp. 42–47.
10. Benesty, J. Adaptive eigenvalue decomposition algorithm for passive acoustic source localization. *J. Acoust. Soc. Am.* **2001**, *107*, 384–391. [CrossRef]
11. Warp, R.J.; Godfrey, D.; Dobbins, J.T. Applications of matrix inversion tomosynthesis. *Med. Imaging* **2000**, *3977*. [CrossRef]
12. Godfrey, D.; McAdams, H.P.; Dobbins, J.T. The effect of averaging adjacent planes for artifact reduction in matrix inversion tomosynthesis. *Med Phys.* **2013**, *40*, 021907. [CrossRef]
13. Zhang, Y.; Ge, S. Design and analysis of a general recurrent neural network model for time-varying matrix inversion. *IEEE Trans. Neural Netw.* **2005**, *16*, 1477–1490. [CrossRef]
14. Guo, D.; Zhang, Y. Zhang neural network, Getz–Marsden dynamic system, and discrete-time algorithms for time-varying matrix inversion with application to robots' kinematic control. *Neurocomputing* **2012**, *97*, 22–32. [CrossRef]
15. Guo, D.; Li, K.; Yan, L.; Nie, Z.; Jin, F. The application of Li-function activated RNN to acceleration-level robots' kinematic control via time-varying matrix inversion. In Proceedings of the Chinese Control and Decision Conference (CCDC), Yinchuan, China, 28–30 May 2016; pp. 3455–3460.
16. Amato, F.; Moscato, V.; Picariello, A.; Sperlí, G. Recommendation in Social Media Networks. In Proceedings of the IEEE Third International Conference on Multimedia Big Data (BigMM), Laguna Hills, CA, USA, 19–21 April 2017; pp. 213–216.
17. Amato, F.; Castiglione, A.; Moscato, V.; Picariello, A.; Sperli, G. Multimedia summarization using social media content. *Multimed. Tools Appl.* **2018**, *77*. [CrossRef]
18. Hopf, B.; Dutz, F.; Bosselmann, T.; Willsch, M.; Koch, A.; Roths, J. Iterative matrix algorithm for high precision temperature and force decoupling in multi-parameter FBG sensing. *Opt. Express* **2018**, *26*, 12092–12105. [CrossRef] [PubMed]

19. Cheng, Y.; Tsai, P.; Huang, M. Matrix-Inversion-Free Compressed Sensing with Variable Orthogonal Multi-Matching Pursuit Based on Prior Information for ECG Signals. *IEEE Trans. Biomed. Circuits Syst.* **2016**, *10*, 864–873. [CrossRef] [PubMed]
20. Ji, S.; Dunson, D.; Carin, L. Multitask Compressive Sensing. *IEEE Trans. Signal Process.* **2009**, *57*, 92–106. [CrossRef]
21. Bicchi, A.; Canepa, G. Optimal design of multivariate sensors. *Meas. Sci. Technol.* **1994**, *5*, 319–332. [CrossRef]
22. Mach, D.; Koshak, W.J. General matrix inversion technique for the calibration of electric field sensor arrays on aircraft platforms. *J. Atmos. Ocean. Technol.* **2007**, *24*, 1576–1587. [CrossRef]
23. Liu, S.; Chepuri, S.P.; Fardad, M.; Maşazade, E.; Leus, G.; Varshney, P.K. Sensor Selection for Estimation with Correlated Measurement Noise. *IEEE Trans. Signal Process.* **2016**, *64*, 3509–3522. [CrossRef]
24. Zhang, Y.; Chen, K.; Tan, H. Performance Analysis of Gradient Neural Network Exploited for Online Time-Varying Matrix Inversion. *IEEE Trans. Autom. Control* **2009**, *54*, 1940–1945. [CrossRef]
25. Zhang, Y.; Ma, W.; Cai, B. From Zhang Neural Network to Newton Iteration for Matrix Inversion. *IEEE Trans. Circuits Syst.* **2009**, *56*, 1405–1415. [CrossRef]
26. Kandasamy, W.; Smarandache, F. *Exploring the Extension of Natural Operations on Intervals, Matrices and Complex Numbers*; ZIP Publishing: Columbus, OH, USA, 2012.
27. Chen, Y.; Yi, C.; Qiao, D. Improved neural solution for the Lyapunov matrix equation based on gradient search. *Inf. Process. Lett.* **2013**, *113*, 876–881. [CrossRef]
28. Yi, C.; Chen, Y.; Lu, Z. Improved gradient-based neural networks for online solution of Lyapunov matrix equation. *Inf. Process. Lett.* **2011**, *111*, 780–786. [CrossRef]
29. Wilkinson, J. Error Analysis of Direct Methods of Matrix Inversion. *JACM* **1961**, *8*, 281–330. [CrossRef]
30. Chua, L.; Yang, L. Cellular Neural Networks: Theory. *IEEE Trans. Circuits Syst.* **1988**, *35*, 1257–1272. [CrossRef]
31. Roska, T.; Chua, L. The CNN universal machine: An analogic array computer. *IEEE Trans. Circuits Syst.* **1993**, *40*, 163–173. [CrossRef]
32. Endisch, C.; Stolze, P.; Hackl, C.M.; Schröder, D. Comments on Backpropagation Algorithms for a Broad Class of Dynamic Networks. *IEEE Trans. Neural Netw.* **2009**, *20*, 540–541. [CrossRef]
33. Potluri, S.; Fasih, A.; Kishore, L.; Machot, F.A.; Kyamakya, K. CNN Based High Performance Computing for Real Time Image Processing on GPU. In Proceedings of the Nonlinear Dynamics and Synchronization (INDS) & 16th Int'l Symposium on Theoretical Electrical Engineering (ISTET), Klagenfurt, Austria, 25–27 July 2011; pp. 1–7.
34. Mainzer, K. CNN and the Evoulution of Complex Information Systems in Nature and Technology. In Proceedings of the 7th IEEE Cellular Neural Networks and Their Applications, Frankfurt, Germany, 24 July 2002; pp. 480–485.
35. Chedjou, J.; Kyamakya, K.; Khan, U.; Latif, M. Potential Contribution of CNN-based Solving of Stiff ODEs & PDEs to Enabling Real-Time Computational Engineering. In Proceedings of the 2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010), Berkeley, CA, USA, 3–5 February 2010; pp. 1–6.
36. Chedjou, J.; Kyamakya, K. A Universal Concept Based on Cellular Neural Networks for Ultrafast and Flexible Solving of Differential Equations. *IEEE Trans. Neural Netw. Learn. Syst.* **2015**, *26*, 749–762. [CrossRef]
37. Stanimirovic, P. Recurrent Neural Network for Computing the Drazin Inverse. *IEEE Trans. Neural Netw. Learn. Syst.* **2015**, *26*, 2830–2843. [CrossRef]
38. Wang, J. Recurrent Neural Networks for Computing Pseudoinverses of Rank-Deficient Matrices. *Siam J. Sci. Comput.* **1997**, *5*, 1479–1493. [CrossRef]
39. Chen, K. Recurrent Implicit Dynamics for Online Matrix Inversion. *Appl. Math. Comput.* **2013**, *219*, 10218–10224. [CrossRef]
40. Feng, F.; Zhang, Q.; Liu, H. A Recurrent Neural Network for Extreme Eigenvalue Problem. In Proceedings of the ICIC 2005, Fuzhou, China, 20–23 August 2015; pp. 787–796.
41. Tang, Y.; Li, J. Another neural network based approach for commuting eigenvalues and eigenvectors of real skew-symmetric matrices. *Comput. Math. Appl.* **2010**, *60*, 1385–1392. [CrossRef]
42. Xu, L.; King, I. A PCA approach for fast retrieval of structural patterns in attributed graphs. *IEEE Trans. Syst. Man Cybern.* **2001**, *31*, 812–817.

43. Liu, Y.; You, Z.; Cao, L. A recurrent neural network computing the largest imaginary or real part of eigenvalues of real matrices. *Comput. Math. Appl.* **2007**, *53*, 41–53. [CrossRef]
44. Bouzerdorm, A.; Pattison, T. Neural Network for Quadratic Optimization with Bound Constraints. *IEEE Trans. Neural Netw.* **1993**, *4*, 293–304. [CrossRef]
45. Zhang, Y. Towards piecewise-linear primal neural networks for optimization and redundant robotics. In Proceedings of the IEEE International Conference on Networking, Sensing and Control, Ft. Lauderdale, FL, USA, 23–25 April 2006.
46. Hopfield, J.J.; Tank, D. Neural Computation of Decisions in Optimization Problems. *Cybernetics* **1984**, *52*, 141–152.
47. Kenndey, M.P. Neural Networks for Nonlinear Programming. *IEEE Trans Circuits Syst.* **1988**, *35*, 554–562. [CrossRef]
48. He, Z.; Gao, S.; Xiao, L.; Liu, D.; He, H.; Barber, D. Wider and deeper, cheaper and faster: Tensorized LSTMs for sequence learning. In Proceedings of the Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 1–11.
49. Jang, J.-S.; Lee, S.-Y.; Shin, S.-Y. An Optimization Network for Matrix Inversion. In Proceedings of the 1987 IEEE Conference on Neural Information Processing Systems, Denver, CO, USA, 8–12 November 1987.
50. Zhang, Y.; Li, Z.; Li, K. Complex-valued Zhang neural network for online complex-valued time-varying matrix inversion. *Appl. Math. Comput.* **2011**, *217*, 10066–10073. [CrossRef]