Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

ADAPTED AZNN METHODS FOR TIME-VARYING AND STATIC MATRIX PROBLEMS*

FRANK UHLIG†

Abstract. We present adapted Zhang neural networks (AZNN) in which the parameter settings for the exponential decay constant η and the length of the start-up phase of basic ZNN are adapted to the problem at hand. Specifically, we study experiments with AZNN for time-varying square matrix factorizations as a product of time-varying symmetric matrices and for the time-varying matrix square roots problem. Differing from generally used small η values and minimal start-up length phases in ZNN, we adapt the basic ZNN method to work with large or even gigantic η settings and arbitrary length start-ups using Euler's low accuracy finite difference formula. These adaptations improve the speed of AZNN's convergence and lower its solution error bounds for our chosen problems significantly to near machine constant or even lower levels. Parameter-varying AZNN also allows us to find full rank symmetrizers of static matrices reliably, for example, for the Kahan and Frank matrices and for matrices with highly ill-conditioned eigenvalues and complicated Jordan structures of dimensions from $n = 2$ on up. This helps in cases where full rank static matrix symmetrizers have never been successfully computed before.

Key words. Adapted ZNN method, Time-varying matrix problems, Numerical algorithm, Matrix factorization, Matrix symmetrizer.

AMS subject classifications. 15A99, 15B99, 65F99, 65F45, 15A21.

1. Introduction to AZNN methods for real or complex matrix flows $A(t)$. We consider time-varying matrix problems that we want to solve more speedily and accurately by using Zhang neural networks (ZNN) adaptively and judiciously, and varying the length of the initial start-up phase and also adjusting the exponential decay constant η in adapted ZNN (AZNN) to the three phases of ZNN, namely its start-up, middle, and final stages. Zeroing neural networks such as as ZNNs are computational workhorses in many branches of engineering. For functional inputs, discretized ZNN helps us to understand the feasibility of a given parametric matrix model's function equation over time or learn of its unfeasibility. If feasible, the chosen model can be implemented successfully in ZNN for sensor clocked discretized input data and used on-chip in robots, other machinery, and in other engineering processes.

ZNN goes back to Yunong Zhang's PhD work of 2000/2001 in Hong Kong [10]. ZNN's engineering applications have been described in almost 500 papers as well as in a handful of books. A theoretical numerical analysis of ZNN methods has never been attempted; see [8].

Starting from a time-varying matrix/vector problem

⓪ $f(x(t), A(t), ...) = g(t, ...)$,

discretized ZNN is designed to compute an approximate solution iteratively in time from a few start-up values. It does so predictively over time and with high accuracy. ZNN's success hinges on Zhang's idea of stipulating exponential decay for the associated error function:

① $E(t) = f(x(t), A(t), ...) - g(t, ...)$,

namely

② $\dot{E}(t) = -\eta E(t)$ for $\eta > 0$.

*Received by the editors on September 20, 2022. Accepted for publication on March 26, 2023. Handling Editor: Ren-Cang Li. Corresponding Author: Frank Uhlig.

†Department of Mathematics and Statistics, Auburn University, Auburn, AL 36849-5310, USA (uhligfd@auburn.edu).

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

③ Assuming a constant sampling gap $\tau = t_{k+1} - t_k = const$ throughout the ZNN process, the derivative of the unknown $\dot{x}(t_k)$ in the error differential equation ② is isolated algebraically if possible. This typically leads to an expression for $\dot{x}(t_k)$ as the solution of a system of linear equations:

$$\dot{x}(t_k) = P(t_k)\backslash q(t_k).$$

④ After summing several Taylor expansions around $x(t_k)$ that contain future $x(t_{k+1})$ and current or past solution $x(t_\ell)$ data for $\ell \leq k$ and the derivative $\dot{x}(t_k)$ of the unknown, we use a look-ahead and convergent finite difference formula of type `j_s`; see [7] for a list thereof, and solve for $\dot{x}(t)$. This leads to an expression for the derivative $\dot{x}(t_k)$ such as

$$\dot{x}_k = \frac{8x_{k+1} + x_k - 6x_{k-1} - 5x_{k-2} + 2x_{k-3}}{18\tau},$$

when for example using a five instance finite difference formula of type `j_s = 2_3` with global truncation error order $O(\tau^3)$.

⑤ Then the two expressions of the derivative $\dot{x}(t_k)$ in ③ and ④ of the unknown $x(t)$ are equated.

⑥ The resulting solution-derivative free equation of ⑤ is finally solved for

$$x(t_{k+1}) = \frac{9}{4}\tau(P(t_k)\backslash q(t_k)) - \frac{1}{8}x_k + \frac{3}{4}x_{k-1} + \frac{5}{8}x_{k-2} - \frac{1}{4}x_{k-3}.$$

This predicts the problem's solution at the next time step t_{k+1}.

⑦ Finally, ZNN iterates the process.

In step ⑥, the discretized ZNN method solves one linear equation and evaluates one short recursion of past solution data, and thereby it solves the underlying time-varying matrix/vector problem cheaply and predictively in time.

The above seven ⑦ steps of ZNN have recently been derived and explained in detail inside [8].

ZNN methods may seem connected to analytic continuation methods and ODE solvers when they start with the error differential equation ②, but when interpreting their operations and results and watching their phenomena, ZNN methods appear to follow quite different numerical principles and seem to belong to a different area of numerical matrix analysis. Further details, open questions, and an early assessment of ZNN methods are given [8].

This paper investigates both the role of the exponential decay constant η of ZNN and the actual length of ZNN's start-up phase. Our AZNN method adopts different strategies to improve the convergence behavior of ordinary ZNN. Adaptive η and other settings help AZNN compute the time-varying problem's predictive solution more accurately and much sooner than ZNN. This is specially useful for time-varying matrix problems for which no globally reliable static matrix algorithms exist such as for the factorization of real or complex square matrices into the product of two symmetric ones [2]. Note that ZNN methods are very tolerant of random, partially "little-sense" start-up values. But such start-up value settings often lead to divergence of the subsequent iterative process or they let ZNN needlessly wander through "solutions" with very low accuracy for a long time. A better way to obtain reasonable start-up values is to use Euler's look-ahead method of type `j_s = 1_2` with local truncation error order $O(\tau^2)$:

$$x(t_{k+1}) = x(t_k) + \tau \cdot \dot{x}(t_k) + O(\tau^2) \quad \text{that is based on the secant rule} \quad \dot{x}(t_k) = \frac{x(t_{k+1}) - x(t_k)}{\tau} + O(\tau),$$

and the problem formulation in tandem from an initial random first guess for $x(t_0)$ and with an adapted η setting. In contrast, the basic non-adapted ZNN method uses the identical value of η as the exponential decay constant in its start-up phase and in its main ZNN iterations ⑤ through ⑦. Likewise, frugality concerns for basic ZNN suggest to compute only the minimally needed number of start-up data before switching into iterative ZNN mode.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

In AZNN, we use different η settings in the start-up point generation phase and in the iterations phase and we test longer and shorter sequences of start-up value runs to find the best option for AZNN and the given matrix problem. The Euler start-up run should last until its errors reach the Euler method's truncation error plateau of around 10^{-5} or 10^{-6}, or a little bit beyond. Then , we switch to the AZNN iterations phase where we use a different η value. These two adaptations achieve the chosen finite difference formula's truncation error order plateau faster than standard ZNN does. The AZNN method helps us to solve our experimental example runs for finding time-varying matrix square roots and time-varying symmetric matrix factorizations in a better way.

Section 2 deals pragmatically with techniques to alter or adapt the basic ZNN method to two specific parametric matrix equations. Section 2 (A) deals with the time-varying matrix square root problem $A(t) = X(t) \cdot X(t)$ that generally has many solutions unless its Jordan structure is badly formed such as in the 2 by 2 flow matrix $A(t) = \begin{pmatrix} 0 & t \\ 0 & 0 \end{pmatrix}$ for which there exists no matrix square root at all for any $t \neq 0$, see [4] or [1]. In Section 2 (B), we consider the time-varying matrix symmetric factorization problem $A(t) = S(T) \cdot V(t)$ for two symmetric matrix flows $S(t) = S^T(t)$ and $V(t) = V^T(t) \in \mathbb{C}_{n,n}$ with $S(t)$ nonsingular for all t. In theory, the symmetrizer problem for static entry matrices $A \in \mathbb{C}_{n,n}$ can always be solved by solving the linear system in the entries of the symmetric matrix S with $S \cdot A = (S \cdot A)^T = A^T \cdot S^T$. But this intuitive approach generally leads to very ill conditioned linear systems. The study of symmetric matrix factorizations goes back more than a century to Frobenius [5, p. 421]. Frobenius showed that the symmetric factor S can always be chosen nonsingular, and that the symmetrizers of A form a linear subspace of dimension n for static $A_{n,n} \in \mathbb{C}_{n,n}$.

Static matrix symmetrizing algorithms that use linear equation solvers, iterative methods, or eigenvalue and singular value based methods were studied in depth in [2]. All of these methods were shown to be deficient in [2] when a given matrix A had degenerate eigen or Jordan structures. The analysis in [2] showed that eigen ill-conditioning prevented these static methods generally from finding invertible symmetrizers $S = S^T$ with $S \cdot A = V$ and $V = V^T$. Until now no symmetric factorizations $A = S^{-1} \cdot V$ could be computed reliably for such A. Section 3 of this paper finally shows how the adapted AZNN method can succeed via a simple parameterization in a matrix flow related to such A where static matrix theory and static numerical analysis have failed us before.

Note: The matrix symmetrizer problem deals exclusively with real or complex symmetric matrices $S = S^T$ and $V = V^T$ whose entries in position i, k are identical to entries in position k, i for all row and column indices $1 \leq i, k \leq n$. Matrices with complex conjugate entries in positions i, k and k, i are Hermitian and not the subject of symmetric matrix factorizations, nor of this paper.

2. Developing AZNN methods for time-varying square roots $X(t)$ and symmetric factorizations $S(t) \cdot V(t)$ of time-varying matrix flows $A(t)$.

(A) Time-varying square roots $X(t)$ for time-varying matrix flows $A(t)$. Fixed entry matrix $A_{n.n}$ and their square roots X with $A = X \cdot X$ have been studied for at least half a century, both numerically and theoretically; see [4, p. 466, 469, 506] and [1] for the existence or nonexistence of static matrix square roots. We recommend the *Wikipedia* entry on *Square root of a matrix* and a look at numerical and applied methods. Use Matlab's `sqrtm.m` file for computing X.

Computing time-varying matrix square root flows $X(t)$ for time-varying matrix flows $A(t)$ requires more effort.

Here, we first look at the basic ZNN method for a time-varying matrix square root problem and recreate [8, Fig. 1] for further study. In Fig. 1, the two graphs show the error function value at the start-up to full `j_s = 4_5` switching point

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

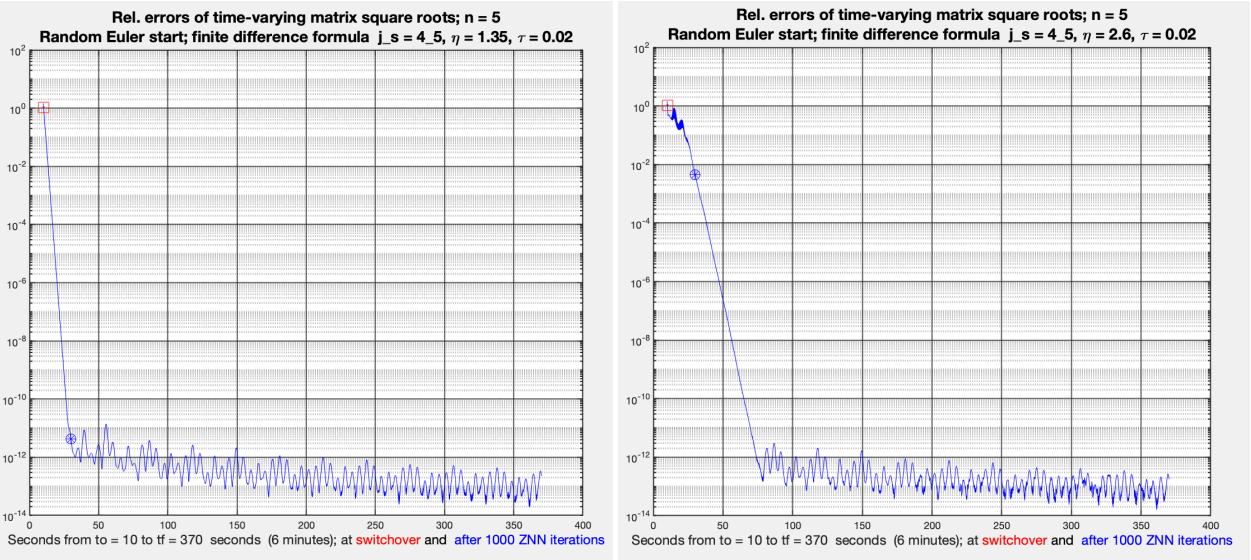

FIGURE 1.

displayed by a red square □ and the 1000th iterate's error function value displayed by a blue circle around a star ⊛; on the left-hand side with $\eta = 1.35$ as in [8, Fig. 1], and on the right side with $\eta = 2.6$ but for the higher truncation order finite difference formula of type `j_s = 4_5`. Both switching points between the start-up and iteration phases of ZNN occur at around 20 s, but with very different error function values, namely at around $4 \cdot 10^{-12}$ when $\eta = 1.35$ and at around $4 \cdot 10^{-3}$ when $\eta = 2.6$. With $\eta = 1.35$ in the left-hand graph of Fig. 1, the transition from the start-up phase that creates $j + s = 4 + 5 = 9$ data points via Euler's method is very smooth until the Euler method's error function hovers below 10^{-11}. For $\eta = 2.6$, the transition from Euler to basic $\eta = 2.6$ ZNN on the right of Fig. 1 is also rather smooth for 20 + iteration steps, but thereafter many steps are wasted with error function value going up and down until the `4_5`-based ZNN method stabilizes and takes over smoothly to converge well error-wise to below 10^{-11} after around 70 s from start as Fig. 1 right and the enlarged view in Fig. 2 clearly shows.

If we increase η further above 2.6, the error curve will move more or less horizontally around the 10^{-1} error level and basic ZNN becomes useless. The error function magnitudes may even diverge to 10^{150} and higher values over our 6 min run.

The aim of creating an adapted ZNN method is to try and extend the short region of rapidly decreasing errors after the switch from Euler to full j_s ZNN steps depicted in Fig. 2 and at the same time try to lower the terminal error function values to the theoretical $O(\tau^{j+s+1})$ local truncation error level for j_s based ZNN iterations.

To understand how this can be achieved, we shall explain the seven steps of ZNN briefly now.

These are the seven steps in basic discretized ZNN for the time-varying matrix square roots problem for a time-varying matrix flow $A(t)_{n,n}$ with $t \in [to, tf] \subset \mathbb{R}$ and a constant sampling gap $\tau = t_{\ell+1} - t_\ell$ for all $\ell = 1, 2, \ldots$.

⓪ Model equation $A(t) = X(t) \cdot X(t)$ for a given matrix flow $A(t)_{n,n}$ and its unknown square roots $X(t)_{n,n}$.

① Error equation $E(t) = A(t) - X(t) \cdot X(t)$.

② Differentiated error equation $\dot{E}(t) = -\eta E(t)$.
Error DE reordered $\dot{A}(t) - \dot{X}(t) \cdot X(t) - X(t) \cdot \dot{X}(t) = -\eta(A(t) - X(t) \cdot X(t))$; and by leaving off the time t dependence:

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

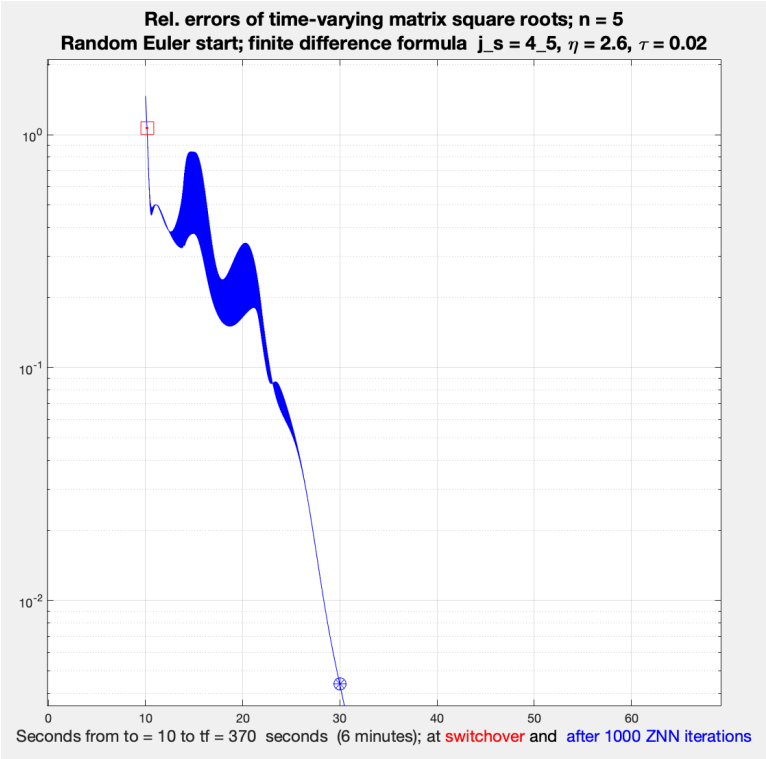

FIGURE 2.

③ $$\dot{X}X + X\dot{X} = \dot{A} + \eta\,(A - XX).$$

Rephrased using Kronecker products $A \otimes B$ and the matrix column vector notation $X(:)$

③$_K$ $$(X^T \otimes I_n + I_n \otimes X)_{n^2,n^2} \cdot \dot{X}(:)_{n^2,1} \in \mathbb{C}^{n^2} =$$
$$= \dot{A}(:)_{n^2,1} + \eta\, A(:)_{n^2,1} - \eta\,(X^T \otimes I_n)_{n^2,n^2} \cdot X(:)_{n^2,1} \in \mathbb{C}^{n^2}\,.$$

With $P(t) = (X^T(t) \otimes I_n + I_n \otimes X(t)) \in \mathbb{C}_{n^2,n^2}$ assumed nonsingular for all t and $q(t) = \dot{A}(t)(:) + \eta\, A(t)(:) - \eta\,(X^T(t) \otimes I_n) \cdot X(t)(:) \in \mathbb{C}^{n^2}$, we finally have separated

③$_{nonsing}$ $$\dot{X}(t) = P(t) \backslash q(t).$$

If $P(t)$ is singular for some t, we cannot invert $P(t)$ and need to use P's pseudoinverse instead. For more details on this process, see [8, Section 3, (III), (IV), (VII), and (VII start-up)].

④ Choose a convergent look-ahead finite difference formula of type `j_s` that involves $X(t_{k+1}), X(t_{k-1}), X(t_{k-2}), \ldots$ and $\dot{X}(t_k)$.

⑤ Solve the difference formula in ④ algebraically for $\dot{X}(t_k)$.

⑥ Equate both expressions of $\dot{X}(t_k)$ in steps ③$_{nonsing}$ and ⑤ and solve the resulting solution-derivative free equation for $X(t_{k+1})$ numerically by solving the linear system in ③$_{nonsing}$ and adding the recursion in ⑤.

⑦ Iterate step ⑥ for $t_{k+2}, t_{k+3}, \ldots$ until you reach the final time tf.

Discretized ZNN benefits from two important innovations. The first is the exponential error decay stipulation and the setting of τ in ②. The second are the combined workings of ③ through ⑤ where ZNN places a barrier, a membrane between the derivative-based formulation and their conditioning problems and the linear equations solving

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

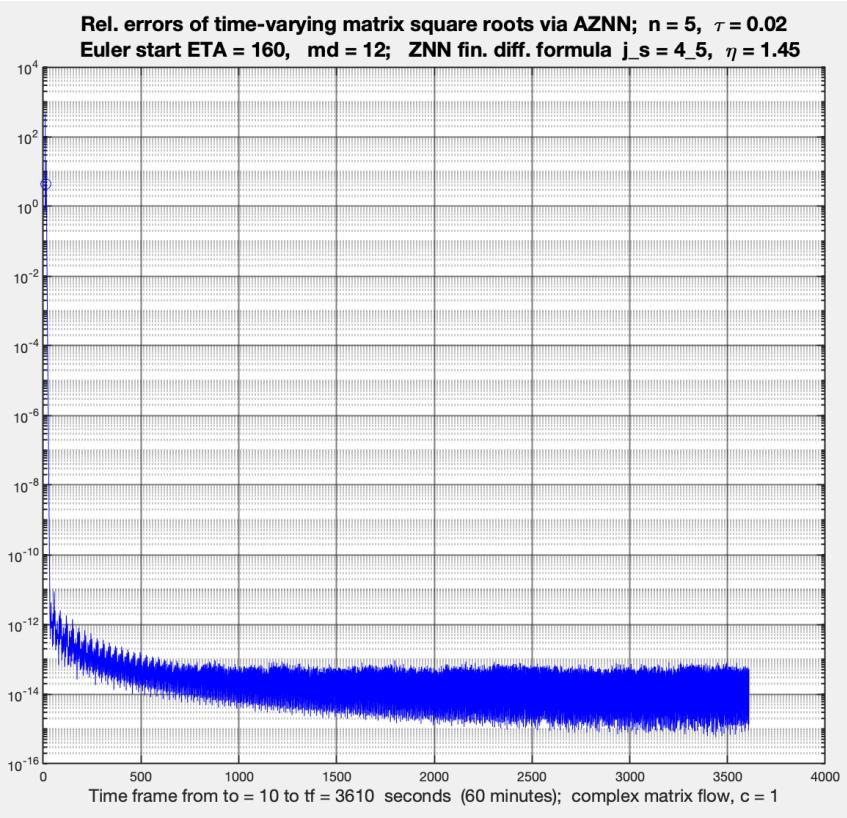

FIGURE 3.

plus recursion step ⑥. Most simply said: conditioning problems at the error equation or the error equation ODE level are not passed into or onto the actual solving part of ZNN and do not affect the linear system conditioning in ⑥. The two – quite different – conditioning limitations of ODE solvers and of linear systems' solvability are not shared, and thereby ZNN's output and efficiency are immune to original problem instabilities and stiffnesses, etc., in the initial problem setup ⓪, ①, and of the ODE setup in ②, ③.

Our first desire was to learn how to assign separate values to the decay constants η used in the start-up phase and in the regular ZNN iteration phase. After some experiments and trial and error discoveries, we have settled on two different exponential decay setting of η, namely at 160 for the start-up phase that we run for 12 steps with Euler while our chosen finite difference formula of type `j_s = 4_5` requires only $j + s = 9$ actual solution data points. We have adjusted η in the iteration phase of ZNN to $\eta = 1.45$ which differs slightly from the earlier global setting of $\eta = 1.35$ for Fig. 1 left. Our time interval for Fig. 3 is 1 h or 3600 s. The error drops below 10^{-11} and stays below after 23 s and levels out around 10^{-13} at the end of the run. Our simulation run took less than 14 s, and it computed 18,000 points of the solution $X(t_k)$ using around $8 \cdot 10^{-4}$ s per time step $t_k \in [10, 3610]$, or in less than 4 % of the constant sampling gap of $tau = 0.02$ that simulates a discrete 50 Hz sensor input.

The error function graphs of Fig. 1 left and Fig. 3 look similar, but the AZNN method reaches a tenfold lower error plateau when compared to basic ZNN. In Fig. 3, AZNN's η values were altered between phases, and the number of initial Euler steps was increased from the necessary start-up data length $j + s = 4 + 5 = 9$ to 12 for this time-varying matrix flow problem.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

We start Euler from the entry-wise square root matrix `A(to).^{0.5}` in Matlab notation for ease with on-chip implementation.

All of our finite difference formulas are convergent and as convergent difference formulas they obey certain rules. For example, with the 5-IFD formula

$$\dot{z}_k = \frac{8z_{k+1} + z_k - 6z_{k-1} - 5z_{k-2} + 2z_{k-3}}{18\tau} \in \mathbb{C}^{n+1},$$

of type `2_3`, its global truncation order $O(\tau^3)$ and its scaled numerator above expressed as:

$$z(t_{k+1}) + \alpha_k z(t_k) + \alpha_{k-1} z(t_{k-1}) + \ \cdots \ + \alpha_{k-\ell} z(t_{k-\ell}),$$

has the characteristic polynomial $p(x) = x^{k+1} + \alpha_k x^k + \alpha_{k-1} x^{k-1} + \ \cdots \ + \alpha_{k-\ell} x^{k-\ell}$. Convergence now requires that $p(1) = 1 + \alpha_k + \alpha_{k-1} + ... + \alpha_{k-\ell} = 0$, see [3, Section 17]. Plugging $z_{..}$ into the 5-IFD difference formula, we realize that

$$z(t_{k+1}) + \frac{1}{8}z(t_k) - \frac{3}{4}z(t_{k-1}) - \frac{5}{8}z(t_{k-2}) + \frac{1}{4}z(t_{k-3}) \approx o_{n+1}, \qquad (*)$$

is approximately the zero vector due to unavoidable truncation and rounding errors. The recursion part of the ZNN computations is split by an equal sign in the final computational equation in step ⑥. Thus,

$$||z(t_{k+1})|| \approx \left|\left| -\frac{1}{8}z(t_k) + \frac{3}{4}z(t_{k-1}) + \frac{5}{8}z(t_{k-2}) - \frac{1}{4}z(t_{k-3}) \right|\right|.$$

And the two nearly equal recursion parts above become ever closer to each other in magnitude due to the convergence to zero condition $(*)$ of all convergent finite difference schemes. Thus, the first linear equations solution term $(P(t_k)\backslash q(t_k))$ in ⑥ eventually has only a comparatively small magnitude. It adjusts the predicted value of $z(t_{k+1})$ only slightly according to the current system data inputs while the remaining finite difference formula term:

$$-\frac{1}{8}z_k + \frac{3}{4}z_{k-1} + \frac{5}{8}z_{k-2} - \frac{1}{4}z_{k-3},$$

has nearly the same magnitude as the computed solution vector z_{k+1} for large indexed time steps t_{k+1}.

This heuristic analysis explains the small, but persistent error magnitude wiggles that set in after a few dozen seconds in Figs. 1 and 3 above when the error level has dropped to well below 10^{-10}. Toward the end of our hour long simulation, the linear equations term that is computed in the AZNN iterative matrix square root example for z_{k+1} has only about 1/10,000 of the magnitude of the finite difference formula term while our trial matrix flow norms have grown to around 10^6. The linear equations "correction term" in z_{k+1} then affects the average number of 13 accurate digits only slightly. These variations by around 1 digit up and down are depicted in Figs. 1 and 3 as "wiggles," and they assure us of between 12 and 14 accurate digits in all sufficiently late computed solutions.

Following Fig. 4 for a matrix symmetrizer problem, we include a data table that describes the same theoretical magnitude disparity. Our Matlab codes for AZNN computations of time-varying matrix square roots are tvMatrSquareRootwEulerStartva.m and AZNNtvMatrSquRootEuler5.m. Both are available at [9] together with the auxiliary codes Polyksrestcoeff3.m, formAnnm.m, and formAnnmc.m.

(B) Time-varying symmetrizers $S(t)$ for time-varying matrix flows $A(t)$ via AZNN. Matrix flow symmetrizers are much easier to handle in AZNN as shown in Figs. 4 and 5. Here, the separate exponential decay parameters of AZNN can apparently be set rather high, or even astronomically high when compared with the rather moderate decay

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

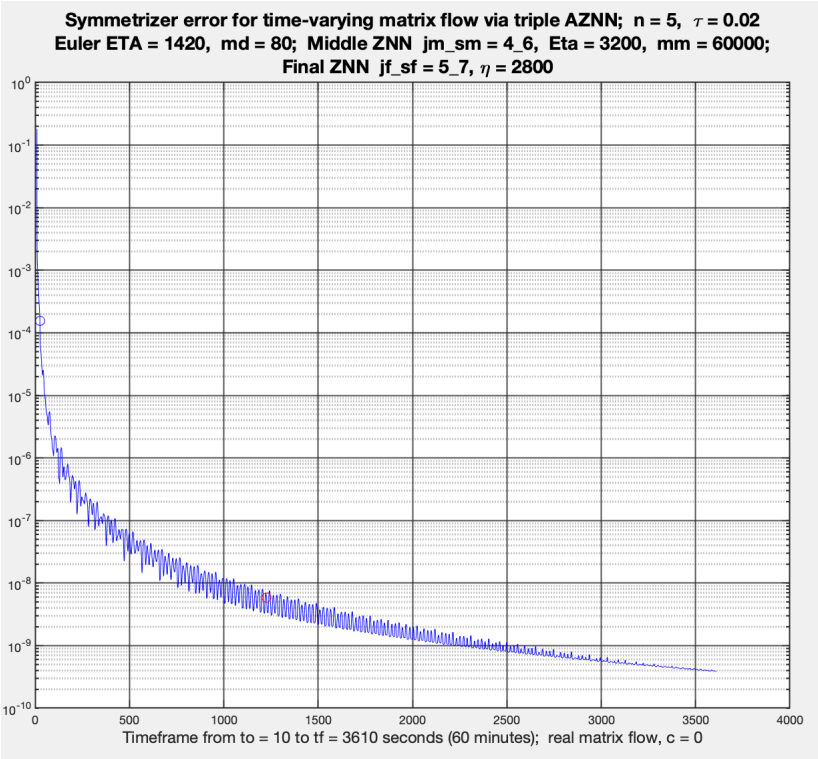

FIGURE 4.

constants that are suitable for convergence in the same matrix flow example $A(t)$ in Section 2 (A) for finding time-varying matrix square roots in Fig. 3. The data or linear solver term-induced wiggles of $z(t_{k+1})$ are much smaller and less noticeable here. They occur in a relatively small time interval from about 2 min to about 45 min after start-up. In this time interval, the magnitudes of the linear solver term in the solution remain around 4 digits below that of the finite difference formula's term. After around 6 h, the magnitude disparity of the two terms increases to 5 digits making the contribution of the linear equations term solve almost irrelevant. But the exponential truncation errors decrease is much slower even with ten or hundred times larger decay constants than those that we found for the time-varying matrix square roots problem.

Individual matrix problems for the same data may behave quite differently under AZNN.innovations

Here is a table of convergence data and magnitude comparisons for the linear equations solution part and the partial difference recurrence part that drive the iterations for the symmetrizer example in Figs. 4 and 5.

The columns in Table 1 list computational data for several discrete times t_k. The second column contains the matrix norms of the time-varying input matrices $A(t_k)$, followed by the norm of the linear equations' system matrices P and of their right-hand sides $q(t_k)$ when computing the next solution at t_{k+1}. The two framed columns 5 and 6 contain the magnitudes of the two terms that predict the symmetrizer $S(t_{k+1})$ of $A(t_{k+1})$. The final two columns measure the actual symmetrization error $||A(t_{k+1})\cdot S(t_{k+1}) - S^T(t_{k+1})\cdot A^T(t_{k+1})||$ for $A(t_{k+1})$ and the unsymmetry of $S(t_{k+1})$. Clearly the two additive terms in the ZNN formulation of $S(t_{k+1})$ have very different magnitudes, and the finite difference term is approximately 10,000 times larger than the linear equations term. These data corroborate our earlier theoretical analysis very well.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

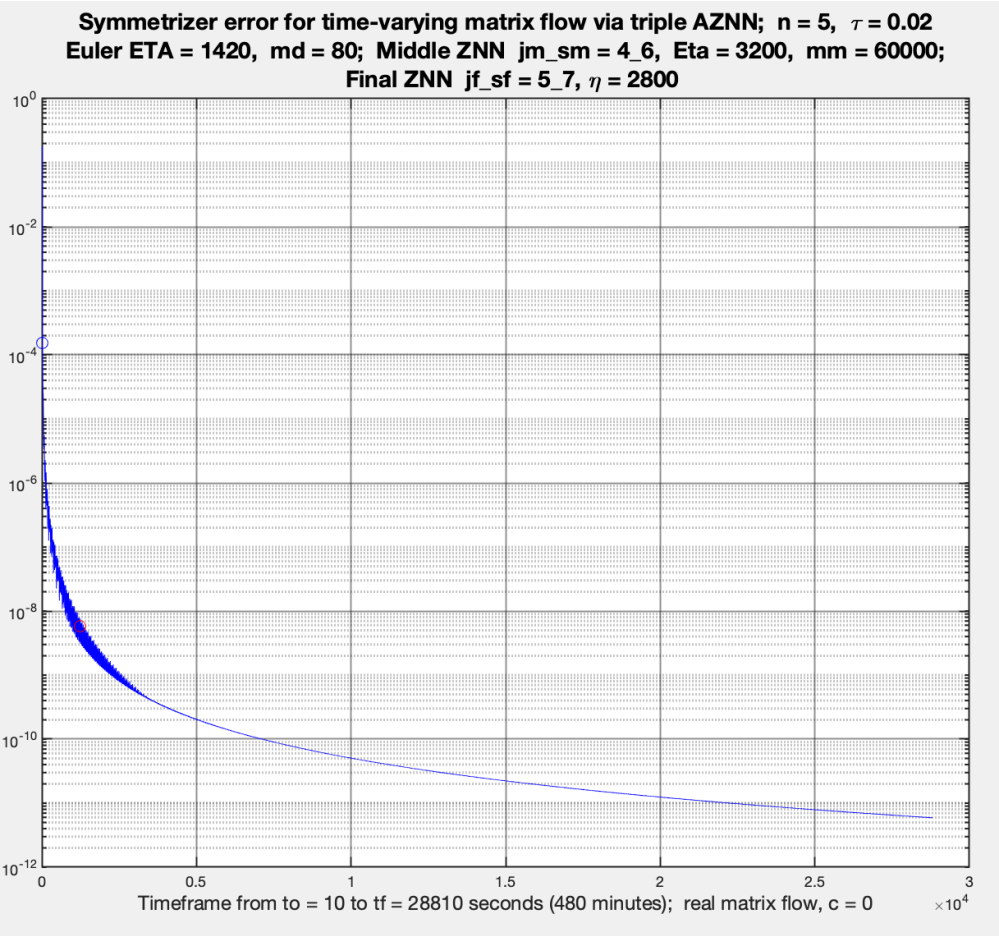

FIGURE 5.

Our next example deals with finding a complex symmetric matrix factorization $S(t) \cdot V(t)$ of a complex matrix flow $A(t)$ via AZNN in a "wiggle-free" way if possible.

Here, we use one fixed complex matrix flow $A(t)$ and determine the η intervals that ensure wiggle-free truncation error curves over long time intervals. In our tests with AZNN below, we keep the Euler initial part value of η and the length of the start-up iterations fixed while we vary the value of η in the subsequent basic ZNN iterations.

Figure 6 depicts the extreme lower and upper values for η that induce wiggly truncation error curves for this example flow. Figure 7 shows upper and lower bounds for non-wiggly truncation error curves, except possibly right near the Euler to ZNN transit point in time, in between which any η can be chosen for smooth output.

In Fig. 7, the lowest relative symmetrizer errors are recorded in the left graph as $8.4872 \cdot 10^{-9}$ at the end of the 30 min run when $\eta = 0.9$ and as $4.1151 \cdot 10^{-8}$ in the right graph for $\eta = 4.2$. Here, the slower exponential decay constant gives us a five times better convergence result. This seems to turn our intuitively expected result on its head. And we know of no explanation for this mystery.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

TABLE 1

```
>>
>> tic, AZNNtvMatrSymmwEulerStartm5(5,0,10,3610,0.02,1420,80,4,6,3200,6000,5,7,2800,2), toc

Norms =

   1.2020e+01   1.9386e+02   2.8242e+02   5.4652e+02   1.5946e-01            0   1.3296e-03   4.4874e-17
   1.4040e+01   2.6816e+02   3.9634e+02   4.7403e+02   1.0660e-01            0   8.5343e-04   5.1171e-17
   1.6060e+01   3.5406e+02   5.3285e+02   4.0925e+02   7.6063e-02            0   6.1513e-04   5.2722e-17
   1.8080e+01   4.5153e+02   6.9353e+02   3.7270e+02   5.5834e-02            0   4.6618e-04   5.5002e-17
   2.0100e+01   5.7282e+02   8.8016e+02   3.5398e+02   4.3063e-02            0   3.5868e-04   6.2674e-17
   2.2120e+01   7.2958e+02   1.0946e+03   3.3851e+02   3.8589e-02            0   2.6930e-04   7.2014e-17
   2.4140e+01   9.1342e+02   1.3389e+03   3.2206e+02   2.6123e-02            0   1.9997e-04   7.0594e-17
   2.6020e+01   1.1110e+03   1.5949e+03   3.1077e+02   1.5206e-02            0   1.5539e-04   6.5465e-17
   3.3726e+02   1.9214e+06   2.7149e+06   5.4012e+02   4.8229e-03   6.6257e+02   8.1507e-08   1.7322e-15
   6.6452e+02   1.4679e+07   2.0754e+07   9.6852e+02   1.6787e-02   6.6256e+02   1.5536e-08   4.1499e-15
   9.9178e+02   4.8787e+07   6.8988e+07   1.9967e+03   9.5725e-03   6.6256e+02   1.1196e-08   5.5233e-14
   1.3190e+03   1.1476e+08   1.6229e+08   2.4593e+03   2.5076e-03   6.6255e+02   5.4348e-09   4.2533e-15
   1.6463e+03   2.2312e+08   3.1552e+08   2.8245e+03   1.0011e-02   6.6255e+02   2.9790e-09   3.0788e-14
   1.9736e+03   3.8436e+08   5.4356e+08   2.5418e+03   7.5087e-03   6.6255e+02   1.2583e-09   2.6304e-14
   2.3008e+03   6.0902e+08   8.6127e+08   2.8895e+03   4.2729e-03   6.6254e+02   9.5952e-10   7.7823e-14
   2.6281e+03   9.0761e+08   1.2835e+09   3.6381e+03   8.9629e-03   6.6254e+02   7.3366e-10   5.2116e-14
   2.9553e+03   1.2906e+09   1.8252e+09   4.3027e+03   3.4317e-03   6.6254e+02   5.9813e-10   1.7531e-14
   3.2826e+03   1.7686e+09   2.5012e+09   4.7778e+03   5.6178e-03   6.6254e+02   4.7676e-10   3.2487e-14
   3.6099e+03   2.3521e+09   3.3263e+09   4.5535e+03   7.5962e-03   6.6254e+02   3.8461e-10   9.6514e-14
   3.6100e+03   2.3523e+09   3.3267e+09   4.5399e+03   7.4442e-03   6.6254e+02   3.8438e-10   9.6206e-14

        tk          ||Am||        ||P||         ||q||     tautau||P/q|| ||fin diff||   Symm error   unsymm_Sm
Elapsed time is 29.171018 seconds.
>>
```

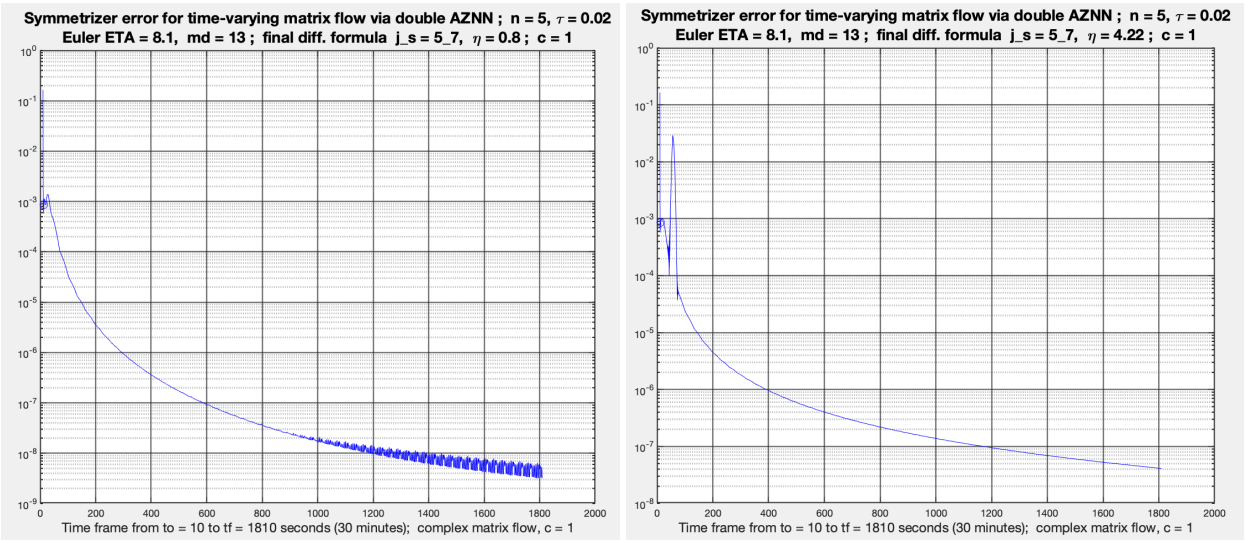

FIGURE 6.

For 5-by-5 real matrix flows, the AZNN computations of the flow's symmetrizers take around 28 s per 1 h of simulation or 224 s for the 8 h plot in Fig. 8. It reached a minimum relative symmetrization error of $5.9095 \cdot 10^{-8}$ at the end of the run.

Our 5-by-5 complex matrix flow experiment for time-varying matrix symmetrizers takes around 48 s per 1 h of simulation or 385 s for the 8 h plot in Fig. 8 that reached a minimum relative symmetrization error of $7.2301 \cdot 10^{-11}$ at the end of its run.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

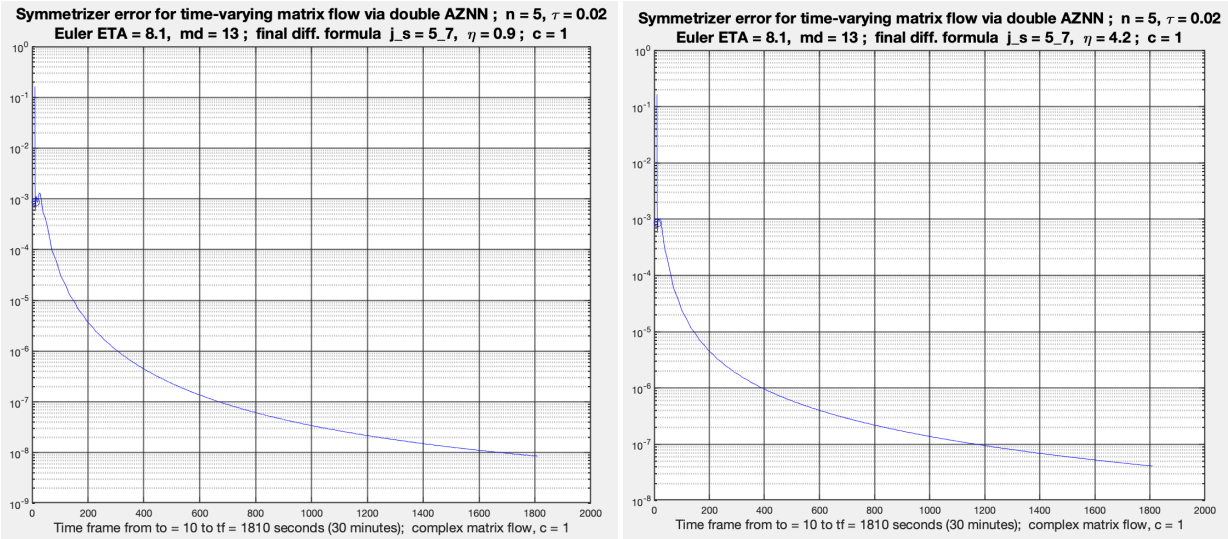

FIGURE 7.

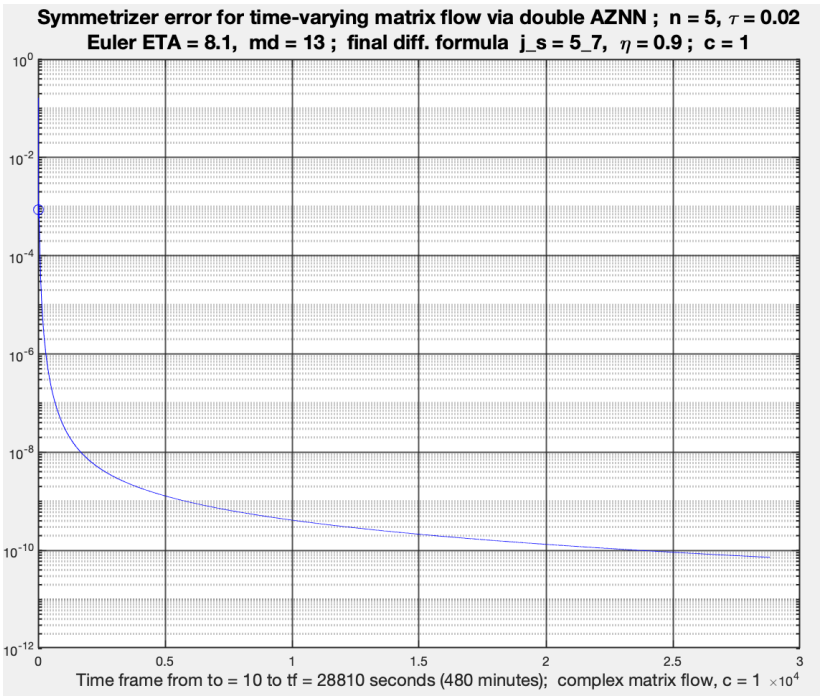

FIGURE 8.

Exploring some phenomena of AZNN in this paper has lead to several new open problems with neural network-based time-varying matrix algorithms: parameter optimizations in AZZN appear to be problem-dependent; for any one problem large magnitude variations are seemingly possible for decay constant intervals with comparative results, even thousand-fold η variations; why are there wiggles in some relative AZNN error graphs and so forth.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

Matlab codes for AZNN computations of time-vaying matrix symmetrizers are AZNNtvMatrSymmwEulerStartm5.m for triple-phase parameter settings and AZNNtvMatrSymmwEulerStartm4.m for double-phase parameter settings for the start-up Euler phase and the ZNN iterations phase only. They are available together with trial matrix flow generating codes at [9].

A new application of AZNN methods uses time-varying neural networks directly to solve some previously unsolvable static matrix symmetrizer problems in Section 3.

3. Computing static matrix symmetrizers via time-varying AZNN. We start with general observations of matrix factorizing methods such as LR, QR, or SV.

For a given fixed entry matrix $A_{n,m}$, real or complex, almost every matrix factorization $A = X \cdot Y$ with specified qualities in X and Y is achieved by operating on the given matrix A from the left with a matrix U of desired type to obtain the right factor Y as $U \cdot A = Y$ with Y as desired and then inverting U to obtain $A = X \cdot Y$ with $X = U^{-1}$ if the inversion of U preserves its special type. To factor $A = L \cdot R$ for lower and upper triangular matrices L and R for example, we obtain the upper triangular R by repeatedly multiplying A and its updates by lower triangular Gaussian elimination matrices from the left. Their product is again lower triangular, and it can be easily inverted while keeping the lower triangular form to give us the LR factorization of A. Likewise for orthogonal Householder matrices that reduce $A_{n,m}$ from the left to upper triangular form in the QR matrix factorization.

The key issue for successfully expressing any given real or complex square matrix A as a product of two symmetric matrices S and V as $A = S \cdot V$ is to find a nonsingular symmetric matrix $U = U^T$ with $U \cdot A = V = V^T = A^T \cdot U$. Then a symmetric matrix factorization of A is $A = S \cdot V$ with $S = U^{-1}$. In 1902, Frobenius [5] proved theoretically that such a factorization exists for each square matrix A. But finding *nonsingular* left-side symmetrizers numerically for all static matrices $A_{n,n}$ proved quite elusive, see [2, p. 609-612], especially for defective or derogatory matrices A with repeated eigenvalues and certain Jordan structured ones inside a recent extensive comparison of eight iterative and eigen or svd based symmetrizer algorithms [2].

An application of time-varying AZNN to compute matrix symmetrizers for a given fixed entry matrix A is relatively simple. We use AZNN for the associated matrix flow $\mathtt{A(t)} = \mathtt{t} * \mathtt{A} + (1 - \mathtt{t})^{\mathtt{a}} * \mathtt{BB}$ here with an exponent $a > 0$ that we choose heuristically through experimentation with $A(t)$ and a random small entry matrix BB. We start the AZNN process with Euler and a chosen constant sampling gap $\tau = t_{k+1} - t_k$ from $t_o < 1$. Typically $t_o = 0.985$ when using small η and τ parameters and $t_o = 0.9985$ for large η and τ parameters. We stop the AZNN iterations at $t_k = 1$ because then we have computed a symmetrizer S of $A(1)$ via AZNN. Clearly this symmetrizer also symmetrizes A since $A(1) = A$ by construction. Here, it is important to synchronize t_o and τ so that in the iteration process $\mathtt{A(t_k)} = \mathtt{t_k} * \mathtt{A} + (1 - \mathtt{t_k})^{\mathtt{a}} * \mathtt{BB}$, the factor $1 - t_k$ reaches zero precisely for some index k.

Now we test AZNN for the fixed entry matrix examples in [2] for which none of its eight symmetrizer algorithms could find a nonsingular symmetrizer. The first one is the Kahan matrix of dimension 35, see [2, p.609, first table] and the *gallery* in Matlab.

The left plot of Fig.3 9 shows the relative error $||S(1) \cdot A - A^T \cdot S(1)||/||A||$ for the computed Kahan (20) matrix symmetrizer $S = S(1)$ at the end point $t_k = 1$ of AZNN for a relatively small exponential decay parameter ETA and a relatively large first BB perturbation for $A(0, 985)$. The value of the *approach exponent* a in $(1 - t_k)^a$ in the definition of $A(t_k)$ here is set to 1.1 for all iterates, indicating a convex, slightly quadratic function approach to $t_k = 1$ from below a linear approach with $a = 1$. Sixteen AZNN steps take us into the 10^{-9} error territory here. Our computed symmetrizers $S(t_k)$ for 35-by-35 Kahan all have condition numbers around 10^8 or 10^{10} and full rank, while the ones computed in [2, p. 609] in six different static matrix ways have condition numbers 10^9 or 10^{10} with full numerical

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

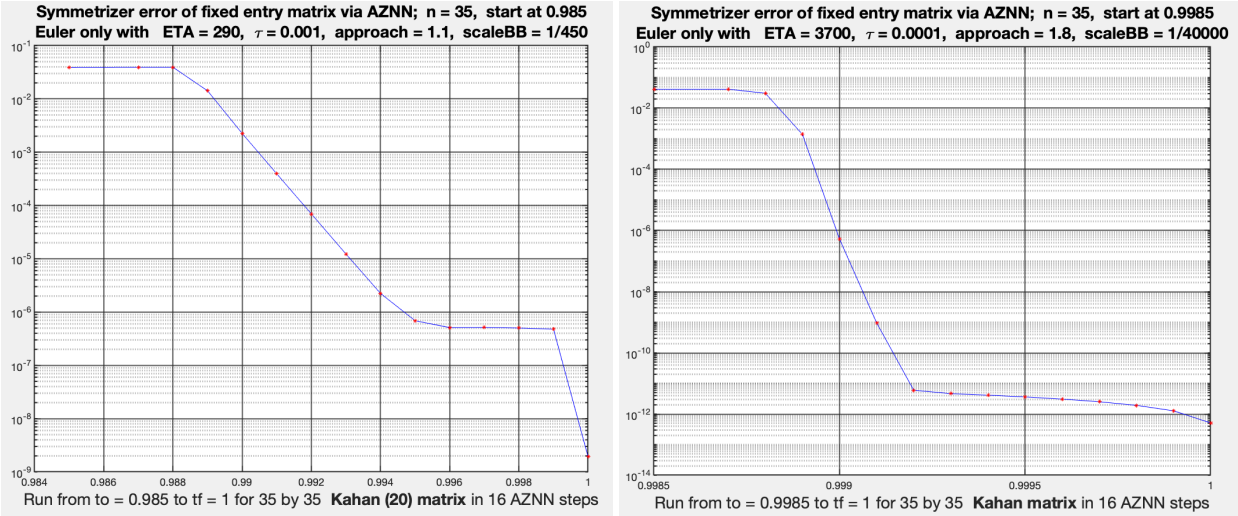

FIGURE 9.

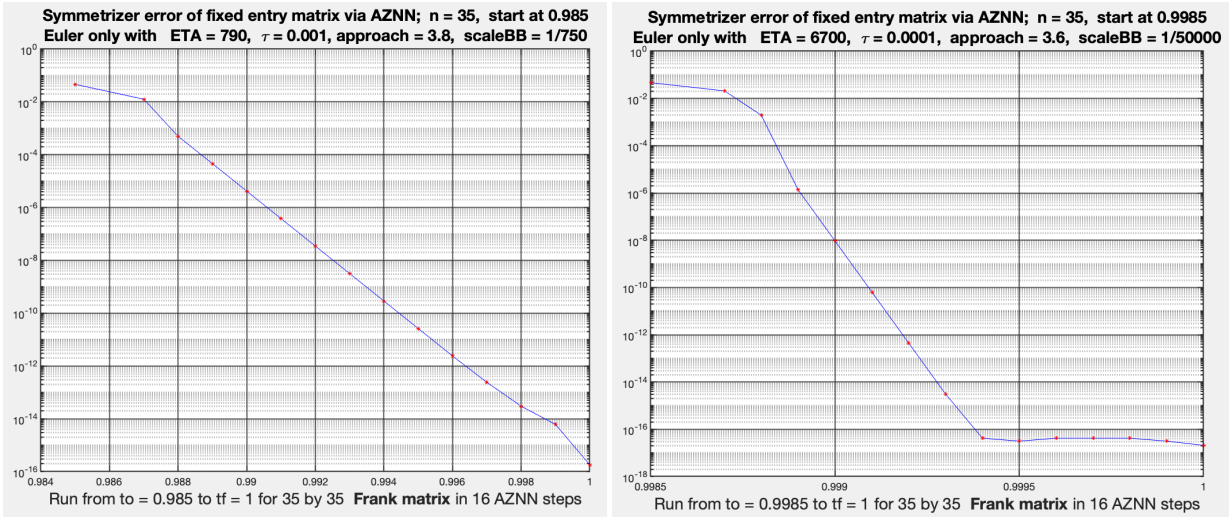

FIGURE 10.

rank according to Matlab's criteria and four eigen method-based solutions have symmetrizer condition numbers of 10^{15} and deficient numerical rank 32 only. The 2-norm of the 35-dimensional Kahan matrix is 4.8.

The right side plot with increased parameters for Eta and the approach constant a, both by factors of around 10, and with a 100-fold decreased perturbation BB size and a 10-fold smaller sampling gap τ, that is, with a countervalent change of parameter pairs, brings the relative error for AZNN down into 10^{-13} territory and computes a full-rank symmetrizer for the Kahan matrix.

Next in [2, p.409, second table] comes the Frank matrix of size 35-by-35 and 2-norm 340. The plots in Fig. 10 show quite similar error behavior as the Kahan matrix did with or without countervalently set parameters, except for the much smaller relative symmetrizing errors that fall well below the machine constant in the right graph of Fig. 10.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

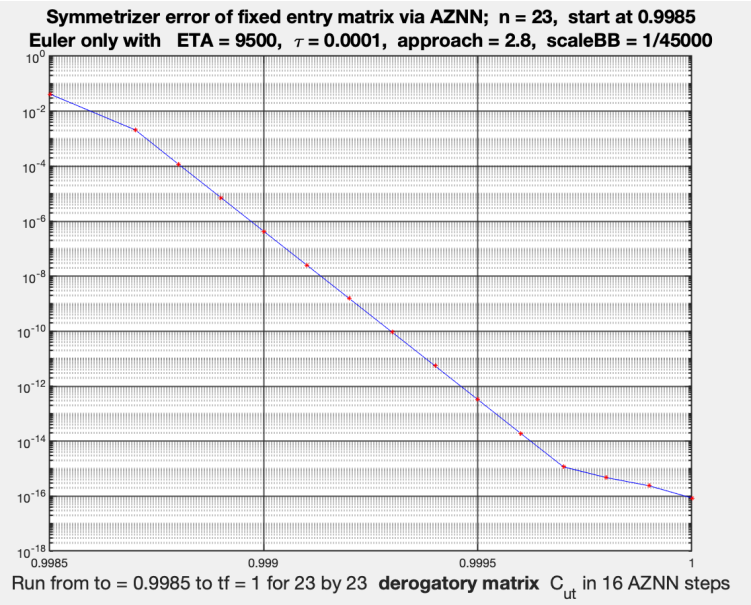

FIGURE 11.

A derogatory 23-by-23 upper triangular matrix C_{ut} with 2-norm 281 was studied in [2]. Here are our AZNN results for this static matrix symetrizer problem with our standard large and small parameter value pairs.

In Fig. 11, we have again used relatively large and small countervalent parameter pairs together to compute a nonsingular symmetrizer of C_{ut} with symmetrizing error below the machine constant and condition number 362. In [2, p. 410, second table], all six iterative and eigen-based methods compute C_{ut} symmetrizers with condition numbers $10^{10}, 10^{11}, 10^{14}, 10^{15}, 10^{18}$, and 10^{19}, ranks below 23 (four times) and full rank 23 only twice.

The above examples all deal with sparse matrices $A_{n,n}$. We have also transformed each of these matrices separately via random dense unitary matrix similarities to dense form and then applied AZNN using the same parameter settings to obtain quite similar results numerically.

The main cost of AZNN methods for matrix square root and matrix symmetrizer computations lies in the necessary transformation of the respective n-by-n matrix model equations to Kronecker n^2 by n^2 matrix form and solving the ensuing n^2 by n^2 linear system for each t_k. Note that $35^2 = 1225$ and the cost of solving each Kronecker matrix linear system now is $O(1225^3/3) = O(1.8383 \cdot 10^9/3) \approx O(6 \cdot 10^8)$. Here, the Kronecker system matrices of the unknown $X(t_k)$ both have the form $(X^T(t_k) \otimes I_n) + (I_n \otimes X(t_k))$, see [8, Sections 2 (VIII) and (VII) start-up)], whose sparsity structure may be exploited for sparse A and X and thus help us to expedite the linear system solves significantly. We have not studied this idea any further.

Our last static matrix symmetrizer problem from [2, p. 617 formula (9) through p. 619] deals with the singular 2-by-2 matrix for varying $0 \leq \alpha \leq 1$:

$$A = \begin{pmatrix} 0 & 1 \\ 0 & \alpha \end{pmatrix} .$$

Here, we receive excellent full-rank results with symmetrizer condition numbers below 2.7 and relative symmetrizer errors well below the machine constant via AZNN for the identical parameter settings as used before.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

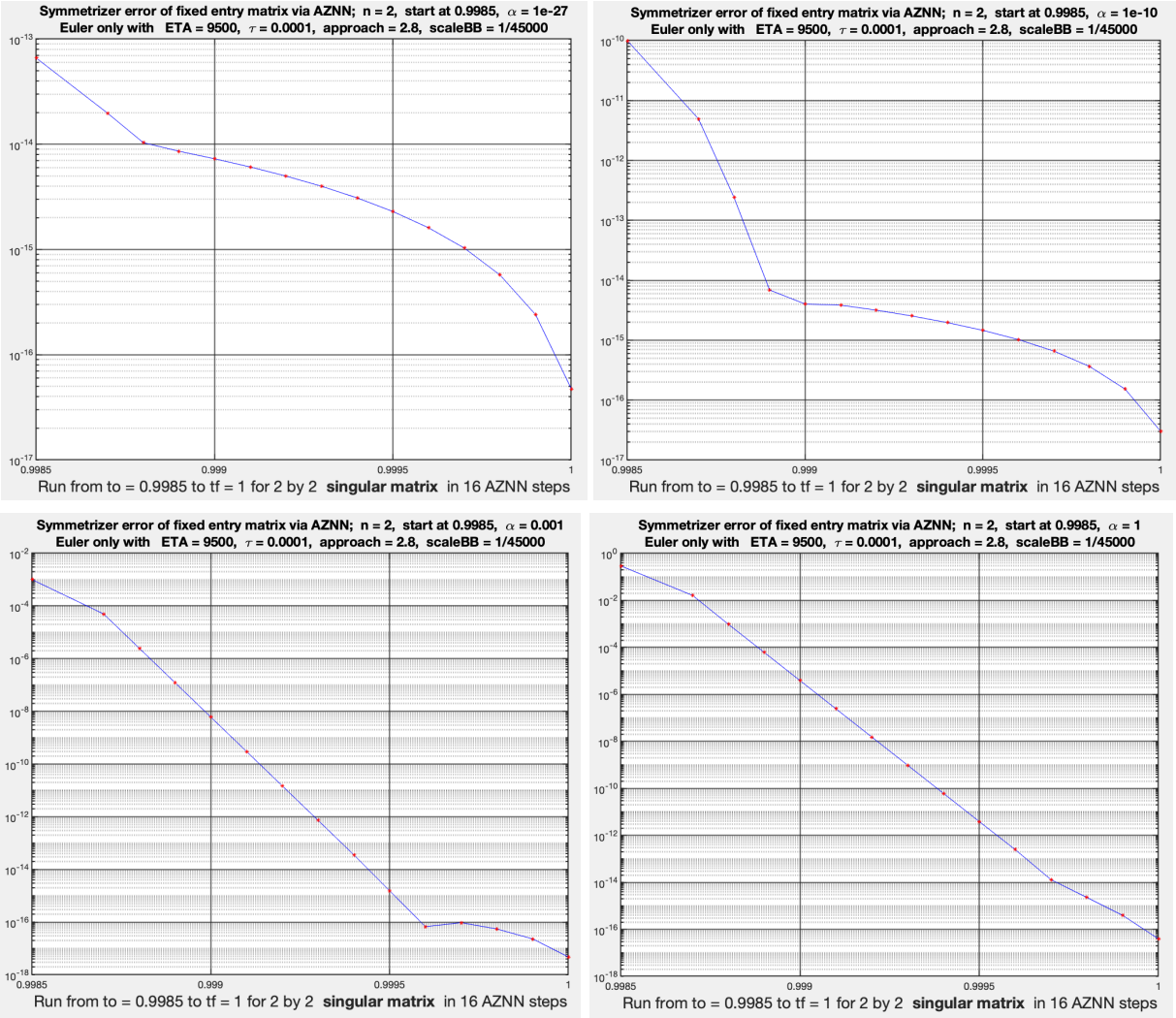

FIGURE 12.

Our static matrix symmetrizer method is AZNNsingleMatrSymmEuler3.m. It uses Matlab's "gallery" matrices gallery('frank',35) or gallery('kahan',35) and Jordan block matrices of varying dimensions built in AJordunisym.m of [9].

4. Conclusions. We have shown how to obtain faster more accurate solutions for time-varying matrix problem through an adapted version AZNN of basic ZNN, where we define the exponential decay constants η in the start-up phase and in the ZNN iterations phase independently as the problem requires and also free the start-up phase to last as long as is needed to ensure a smooth transition between AZNN's two phases.

In the process, we have also learnt how and when to use "astronomically high" η settings for improved theoretical results. These results were obtained when investigating two complicated time-varying n-by-n matrix factorizations that require a model equation switch to Kronecker matrix form due to their very formulation.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

TABLE 2
Table of $h = \eta \cdot \tau$ values for Figures 1 through 12.

	Left side graph	Right side graph
Figure 1 (t.-v. square root)	h = 0.027	h = 0.056
Figure 3 (t.-v. square root)	h = 3.2	h = 0.029
Figures 4 and 5 (t.-v. symmetrizer)	Start h = 28.4 / Middle h = 64 / End h = 56	
Figures 4 and 5 (t.-v. symmetrizer)	Start h = 28.4 / Middle h = 64 / End h = 56	
Figure 6 (t.-v. symmetrizer)	Start h = 0.162 / End h = 0.016	Start h = 0.162 / End h = 0.084
Figure 7 (t.-v. symmetrizer)	Start h = 0.162 / End h = 0.018	Start h = 0.162 / End h = 0.084
Figure 8 (t.-v. symmetrizer)	Start h = 0.162 / End h = 0.018	
Figure 9 (Static symmetrizer)	(Euler) h = 0.29	(Euler) h = 0.37
Figure 10 (Static symmetrizer)	(Euler) h = 0.79	(Euler) h = 0.67
Figure 11 (Static symmetrizer)	(Euler) h = 0.95	
Figure 12 (Static symmetrizer)	Top and bottom left graphs (Euler) with α at 10^{-27} and 10^{-3} h = 0.95	Top and bottom right graphs (Euler) with α at 10^{-10} and 1 h = 0.95

For n-by-n matrix flows $A(t)_{n,n}$ and the unknown matrix $X(t)$, Kronecker-based matrix problems increase the linear equations solving dimensions for ZNN iterates from $O(n^3/3)$ to $O((n^2)^3/3) = O(n^6/3)$ operations. This slows down time-varying matrix factorizations from typically fractions of a second to possibly dozens of seconds. But a study of the sparsity structure of the Kronecker matrices $(X^T(t) \otimes I_n) + (I_2 \otimes X(t))$ may improve the speed of time-varying matrix flow factorizations in the future. And really high dimensions n are not generally encountered even in complex biological and chemical reaction models as demonstrated in [6] in numerous applications.

This paper is not about *Theory* or the numerical analysis of time-varying matrix computations, which – at this moment – has not even begone for Zhang neural networks. Here, we rather have shown some phenomena and behavior that appear with AZNN methods when altering the set of parameters of ZNN in each of its phases experimentally. One commonly noted event with basic ZNN is the relative constancy of the product $h = \eta \cdot \tau$ for ZNN over wide ranges of τ and η for any one problem. Table 2 lists the η and τ data for all our experiments which – for AZNN – puts this assumption to rest with wide h variations and factors of 2 to 10 (Fig. 6 left) and even 110 (Fig. 3) for identical problems.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 39, pp. 164-180, April 2023.

REFERENCES

[1] G.W. Cross and P. Lancaster. Square roots of complex matrices, *Lin. Multilin. Alg.*, 1:289–293, 1974. http://doi.org/10.1080/03081087408817029 .

[2] F. Dopico and F. Uhlig. Computing matrix symmetrizers, Part 2: New methods using eigendata and linear means, a comparison. *Lin. Alg. Appl.*, 504:590–622, 2016. http://dx.doi.org/10.1016/j.laa.2015.06.031 .

[3] G. Engeln–Müllges and F. Uhlig. *Numerical Algorithms with C, with CD-ROM*. Springer, 596 p., 1996, (MR 97i:65001) (Zbl 857.65003).

[4] J.-C. Evard and F. Uhlig. On the matrix equation $f(X) = A$. *Lin. Alg. Appl.*, 162:447–519, 1992.

[5] F.G. Frobenius. Über die mit einer Matrix vertauschbaren Matrizen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, 3–15, 1910. Also in: F. Frobenius and G. Abhandlungen, vol. 3, Springer, 415–427, 1968.

[6] S. Elnashaie, F. Uhlig, and C. Affane. *Numerical Techniques for Chemical and Biological Engineers using MATLAB, A Simple Bifurcation Approach*. Springer, 2007, ISBN-10: 0-387-34433-0, ISBN-13: 978-0-387-34433-1, 590 p + xxiv p, with 95 MATLAB codes on CD.

[7] F. Uhlig. The construction of high order convergent look-ahead finite difference formulas for Zhang Neural Networks. *J. Diff. Equat. Appl.*, 25:930–941, 2019. https://doi.org/10.1080/10236198.2019.1627343 .

[8] F. Uhlig. Zeroing neural networks: An introduction to predictive computations for time-varying matrix problems via ZNN, 34 p, submitted, http://arxiv.org/abs/2008.02724.

[9] F. Uhlig. MATLAB codes for plotting and assessing matrix flows under AZNN. Available at http://www.auburn.edu/~uhligfd/m_files/AZNN/.

[10] Y. Zhang and J. Wang. Recurrent neural networks for nonlinear output regulation. *Automatica*, 37:1161–1173, 2001.