

Memristive Linear Algebra

J. Lin^{1, 2,*} F. Barrows^{1, 2, †} and F. Caravelli^{1, ‡}

¹ *Theoretical Division (T-4), Los Alamos National Laboratory, New Mexico, 87545, USA*

² *Center for Nonlinear Studies, Los Alamos National Laboratory, New Mexico, 87545, USA*

The advent of memristive devices offers a promising avenue for efficient and scalable analog computing, particularly for linear algebra operations essential in various scientific and engineering applications. This paper investigates the potential of memristive crossbars in implementing matrix inversion algorithms. We explore both static and dynamic approaches, emphasizing the advantages of analog and in-memory computing for matrix operations beyond multiplication. Our results demonstrate that memristive arrays can significantly reduce computational complexity and power consumption compared to traditional digital methods for certain matrix tasks. Furthermore, we address the challenges of device variability, precision, and scalability, providing insights into the practical implementation of these algorithms.

I. INTRODUCTION

Matrix and matrix-vector operations are fundamental operations in various scientific and engineering applications, including solving systems of linear equations [1], signal processing [2], scientific computing [3, 4], machine learning [5], and control systems [6]. Various algorithms are available for inverting general matrices, such as Gaussian elimination, Gauss-Jordan elimination [7], Cholesky decomposition [8], QR decomposition, and LU decomposition [4]. These algorithms, while common, are computationally demanding and typically involve cubic complexity in terms of the number of matrix-vector operations. State-of-the-art algorithms for matrix inversion will be reviewed in Sec. II.

These methods, while well-established, face significant challenges related to memory consumption, algorithmic complexity, and power efficiency, particularly as the size of the matrices increases. High-performance GPUs are known for their significant power consumption (300–350 W), which is a critical factor in designing energy-efficient computing systems [9] [10].

In recent years, there has been growing interest in leveraging memristive crossbar arrays for computational tasks due to their potential for high-density integration, non-volatility, and low power consumption [11]. Memristive devices (e.g. resistors with memory) are resistive switching devices that can be used to perform analog matrix-vector multiplications efficiently. These are promising candidates for implementing analog, neuromorphic, and other unconventional computing paradigms.

Over the last few years, using the fact that Kirchhoff laws can be exploited to perform matrix-vector multiplication operations in one-shot, memristive crossbars have been shown to be a promising platform to implement matrix operation algorithms.

However, despite these promising developments, several challenges remain in realizing practical memristive crossbar-based matrix inversion. One major issue is the precision of the analog computations, which can be affected by device variability, non-idealities, and noise. Despite these challenges, research in crossbar arrays has flourished over the last decade [12–21], and state-of-the-art cross-point memristive memory has reached effectively 1024 (10 bits) states using a variety of noise-reduction techniques [22]. Although this precision is not enough yet for scientific computing, the methods currently implemented for noise reduction are scalable and promising for error mitigation in future technology.

Crossbars are typically thought of as accelerators of matrix-matrix and matrix-vector multiplication [23]. However, [24] present an innovative approach for matrix inversion utilizing cross-point resistive arrays. The process involves implementing the target matrix in a cross-point array circuit, where the input currents are applied, and the resulting output potentials are measured. This physical realization leverages the properties of resistive memory devices (RRAM) to perform matrix-vector multiplication (MVM) efficiently. The matrix inversion is achieved through a feedback mechanism with operational amplifiers that forces the output voltage to satisfy the equation $A \cdot V + I = 0$, thus obtaining $V = -A^{-1} \cdot I$. In practice, the authors solve linear equations $A\vec{x} = \vec{b}$ in one step, but this can be used to solve for matrix inverses in N steps by carefully choosing \vec{b} at each iteration, where N is the size of the matrix. The authors also demonstrated this method's accuracy and stability by comparing the experimentally measured inverse matrix with small matrices.

We ask whether it is possible to use a different method to solve a variety of problems at the same time. It is also crucial to note that the method in [24], while very fast, requires a cross-point array whose junction conductances are already driven to represent A . Our method addresses this requirement by instead driving the physical state of the cross-point array to A^{-1} using recursive iteration, which provides the desired answer but also yields an analog resource for $O(1)$ linear transformations of in-

* jlin1212@lanl.gov

† fbarrows@lanl.gov

‡ caravelli@lanl.gov

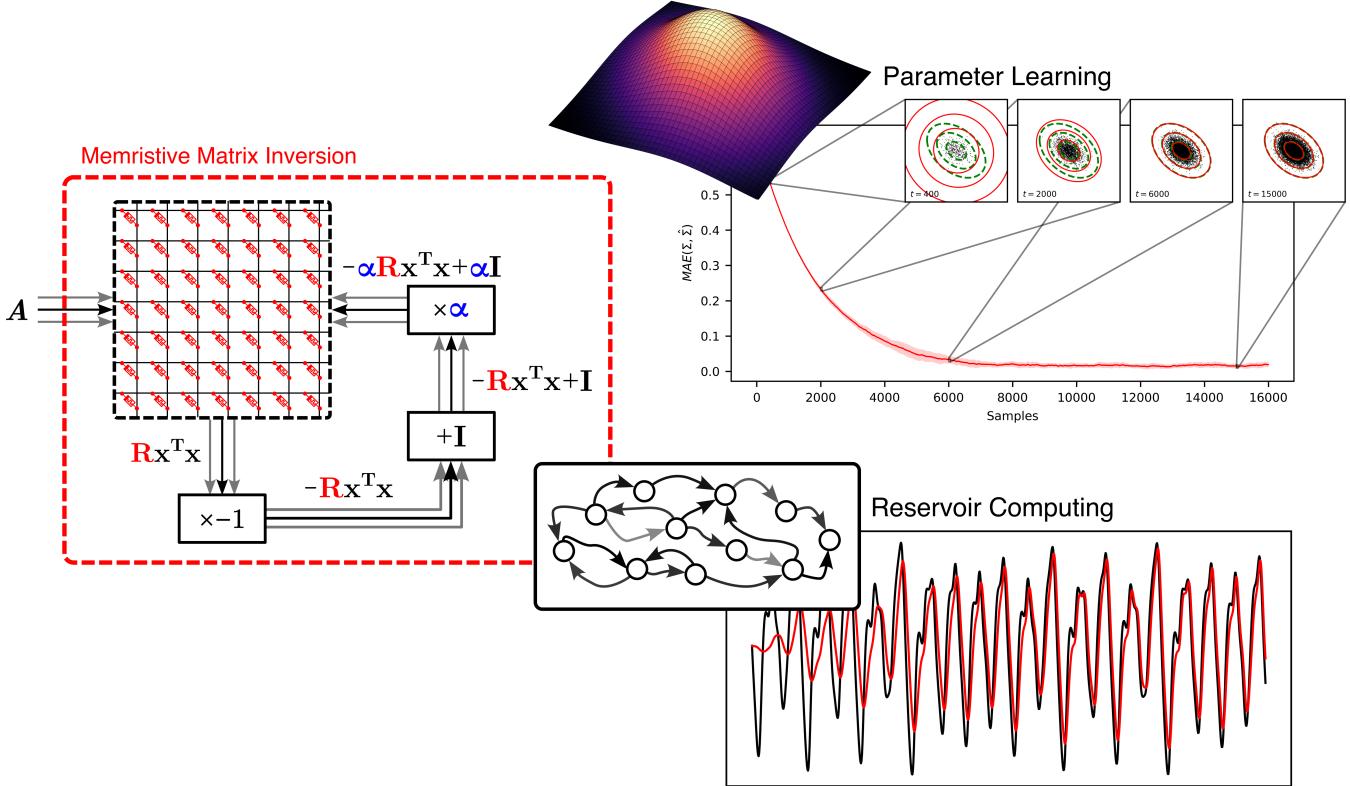


FIG. 1: The matrix inversion algorithm leverages Kirchhoff's laws to compute a feedback-based matrix inversion with computational advantage. At each iteration, a matrix \mathbf{A} is right-multiplied with the current crossbar state in $O(N)$ and used to compute an incremental update to the crossbar state until the crossbar state $\mathbf{R}^* = \mathbf{A}^{-1}$. Aside from the matrix-matrix multiplication, only three applications of simple arithmetic are required. The resulting process is simple and powerful, and power consumption and speed can be tuned with the control parameter α . This enables the implementation of output filters for applications like parameter learning and reservoir computing.

put signals/vectors using the resulting inverse matrix.

Moreover, the scalability of memristive crossbar arrays is a critical factor that influences their applicability in large-scale matrix computations. As the size of the crossbar array increases, issues related to interconnect resistance, crosstalk, and power distribution become more pronounced. Crosstalk and sneak paths can be reduced dramatically using the 1T1R approach, standing for 1 transistor 1 resistor. Although there are other techniques (1 selector 1 resistor 1S1R, and 1 diode 1 resistor 1D1R), the 1T1R is the standard approach to avoid crosstalk in the experimental setup. We will use this technique in our study.

In Sec. II we provide a summary of previous works and a summary of the results obtained in this paper. In Sec. III introduce all the paper's main results. In Sec. III A we introduce the algorithm we will be basing our results on as a dynamical system, and introduce the different variations to obtain different types of (pseudo-)inverses. In Sec. III B we discuss the simulation scheme and SPICE that we use to implement the inverses by simulating the crossbar. In Sec. III C we discuss analytical results and power consumption of the algorithm. In Sec.

IV we discuss applications, and in particular we consider parameter learning of Gaussian distributions, and online learning for reservoir computing. Conclusions follow. All the derivations of the results of this manuscript are provided in the Appendices.

II. SUMMARY OF PREVIOUS WORKS AND ANALOG METHODS

A. Previous work

Matrix operations are fundamental across many fields. For instance, in the traditional approach to matrix multiplication, multiplying two $N \times N$ matrices involves breaking one of the matrices into N column vectors and performing N matrix-vector multiplications. Each matrix-vector multiplication operation requires $O(N^2)$ operations, leading to an overall complexity of $O(N^3)$ for matrix multiplication. Significant research has focused on reducing this complexity. In 1969, Strassen was the first to break the $O(N^3)$ computational wall and introduced an algorithm that lowered the complexity from

$O(N^3)$ to $O(N^{2.808})$ [25]. Later, in 1978, Pan achieved a further reduction to $O(N^{2.796})$ [26]. Coppersmith and Winograd were able to bring it down to $O(N^{2.496})$ [27] (whose method settled to $O(N^{2.38})$ [28]), though reducing the complexity exponent below 2 has remained elusive. In 2003, Umans and Cohn [29] introduced a group theoretic method for matrix multiplication, followed by the result by Umans, Cohn, Kleinberg, and Szegedy which we briefly discuss. They proposed that embedding matrix multiplication into the group algebra of a finite group could yield faster algorithms. This approach hinges on finding groups satisfying specific structural properties, termed the “triple product property.” Two conjectures arising from their work suggest that if proven, they could establish the matrix multiplication exponent as 2 (i.e. $O(N^2)$). These algorithms still fundamentally rely on matrix-vector multiplications but optimize the total number of operations required through clever recursive strategies. Some comments are in order. First, as Higham puts it, “to numerical analysts, matrix inversion is a sin” [30]. For instance, it is not necessary to calculate the matrix inverse to solve $Ax = \vec{b}$, and tailored algorithms are designed to solve this specific problem (although not changing the scaling in N if not by a prefactor). This will be the case also for us later. Second, it is important to stress that the scaling in N is not the only important quantity to keep into account.

The conditioning number $\kappa(A)$ of a matrix A quantifies how sensitive the solution (or inverse) is to perturbations in A and is defined as $\kappa(A) = \|A^{-1}\|$ where $\|\cdot\|$ is a generic matrix norm. A high conditioning number indicates that A^{-1} can amplify errors due to small perturbations in A , potentially leading to less accurate solutions. Conversely, a low conditioning number indicates that A^{-1} is less sensitive to such perturbations. In our case, the conditioning number affects the relaxation of the solution. Solving linear algebra problems when dealing with matrices with high $\kappa(A)$ requires careful algorithm selection and possibly preconditioning to ensure accurate results. Direct methods may be suitable for well-conditioned matrices or smaller problems where computational efficiency is less critical. For large or ill-conditioned matrices, iterative methods with appropriate preconditioning are often preferred. Algorithms should be chosen based on the specific properties of A (e.g., symmetric, positive definite) to ensure numerical stability and efficiency. However, with the ever-increasing dimensions of matrices and the exponential growth in data, traditional methods and their improvements are becoming inadequate. Although quantum computers can achieve $O(\log(N)\kappa^2)$ in principle [31, 32], experimental tests of this algorithm have been obtained only for very small matrices.

From the perspective of classical devices, the advent of parallel computing has shifted the focus towards developing efficient algorithms for large-scale, distributed matrix inversion. For instance, the von Neumann-Ulam algorithm is a probabilistic method for matrix inversion,

leveraging the power of stochastic processes. The algorithm is based on iterative refinement and uses random sampling to approximate the inverse of a matrix [33–35]. The strength of the von Neumann-Ulam algorithm lies in its ability to handle large matrices more efficiently than deterministic methods via parallelization, especially when the matrix is sparse or has certain structural properties that can be exploited by the stochastic approach. It is also worth noticing that certain ensembles of dense matrices have well-defined matrix inverses, something that has been noticed recently, and that can be calculated in $O(N^2)$ square time if nothing is known about row and column sums, and in $O(1)$ if the row and column sums are known [36–38]. It is also worth mentioning recent efforts on stochastic hardware accelerators for matrix inversion [39].

To conclude, the Conjugate Gradient (CG) method [30] is widely employed for solving symmetric positive definite linear systems, and is the closest iterative approach to the one we implement. Adaptations for positive semi-definite matrices A that are also s -sparse extend its applicability. Given $A\vec{x} = \vec{b}$, where A is s -sparse and positive semi-definite, the CG method aims to find $\vec{x} \in \mathbb{R}^N$ by minimizing the quadratic form $|A\vec{x} - \vec{b}|^2$, starting from

$$\vec{x}^{(0)} = \vec{0} \quad \text{and} \quad \vec{r}^{(0)} = \vec{b} \quad (1)$$

For $k = 0, 1, 2, \dots$:

$$\alpha_k = \frac{\vec{r}^{(k)T} \vec{r}^{(k)}}{\vec{p}^{(k)T} A \vec{p}^{(k)}} \quad (2)$$

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \alpha_k \vec{p}^{(k)} \quad \vec{r}^{(k+1)} = \vec{r}^{(k)} - \alpha_k A \vec{p}^{(k)} \quad (3)$$

$$\beta_{k+1} = \frac{\vec{r}^{(k+1)T} \vec{r}^{(k+1)}}{\vec{r}^{(k)T} \vec{r}^{(k)}} \quad (4)$$

$$\vec{p}^{(k+1)} = \vec{r}^{(k+1)} + \beta_{k+1} \vec{p}^{(k)} \quad (5)$$

The algorithm terminates when a convergence criterion is met (e.g., residual tolerance or maximum iterations). For a s -sparse matrix A , the Conjugate Gradient method operates with $O(Ns\kappa)$ time complexity, where N is the matrix size and κ is the condition number of A [40]. This makes it suitable for large-scale problems where A is known to have a finite conditioning number. The drawback is that this method is not efficient on general-purpose processors, as it requires a sparse matrix-vector multiplication (SMVM) kernel [41], but can be implemented on a GPU [42]. In this paper, we ask whether a similar scaling can be obtained with an analog implementation on a cross-bar array.

B. Summary of results

This paper presents a novel approach to solving matrix equations using cross-point memristive arrays, commonly known as memristive crossbars. The main contributions and results of our study are summarized as follows:

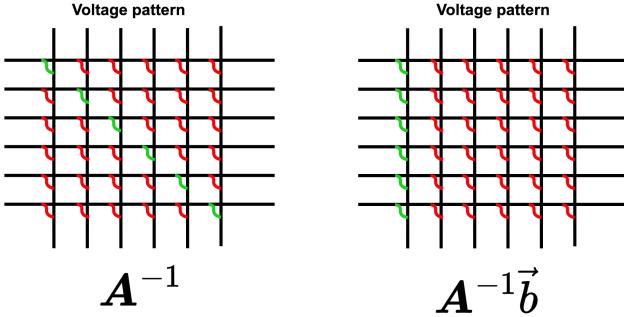


FIG. 2: The figure above shows the constant voltage pattern used to solve the two linear problems on the crossbars. On the left, the constant voltage pattern is proportional to the diagonal matrix, and thus the bias is located similarly on the crossbar elements. For the solution of the linear problem, we pick a column and place the vector \vec{b} . This implies that we can solve N parallel instances of the same problem using this scheme.

- We develop a method to leverage the properties of memristive devices to perform matrix inversion efficiently on crossbar arrays. The algorithm is described in Sec. III A. With this method, we can both solve for the matrix inverse when it exists, and we provided results for changing the approach to obtain the Moore-Penrose and Drazin pseudo-inverses. We also provide an algorithm to solve N parallel $A\vec{x} = \vec{b}$ as a slight modification of the same algorithm. This can be done by changing the voltage patterns on the crossbar as shown in Fig. 2. In particular, we have provided an online and offline implementation. We observe that it is unfair to claim a 1-step matrix inversion, as several operations need to be implemented on an analog device to reach the solution. We introduce the number of operations (these are a combination of read and write operations) as a metric, and show that these scale linearly with the matrix size. We show that this method can be easily implementable on non-volatile devices, both for positive and negative matrices, and for volatile devices a backtracking method can be implemented to apply the inversion algorithm. An interesting observation is that there is a tradeoff between precision and power consumption using this algorithm.
- A key finding of our work is the proof of convergence for the proposed algorithm under noise and with the constraints on applied quantized (in time and values) of the applied voltages. We demonstrate that the algorithm reliably converges to the correct matrix inverse, ensuring its practical applicability and robustness. A summary is provided in Sec. III D and the proofs in the Appendices.

- We validated our theoretical results through extensive simulations and on certain applications of interest. The empirical data supports our claims, showing that the memristive array-based method not only performs accurately but also offers considerable advantages in terms of speed and energy efficiency. We tested our algorithm on variational parameter learning, both online and offline, and tested a method for online reservoir computing in Sec. IV.

Despite these advancements, it is important to stress that algorithms whose inversion scales as $O(N^\alpha)$ with $\alpha \leq 2$ will have a scaling $O(N^2)$. The reason is that $O(N^2)$ is the price that one needs to pay for storing A in the memory in the first place. This will be the case for us too[43].

We provide the key results in the next section.

III. RESULTS

A. The main algorithm

The algorithm we employ for matrix inversion on a memristive crossbar array can be conceptualized as the emulation of a differential equation. This method uses voltage generators as drivers, applying voltages across the memristive elements to iteratively update their resistance values, effectively solving the system of linear equations.

In a manner reminiscent of the Babylonian method for finding square roots, our approach iteratively refines an initial guess to converge on the desired solution. Specifically, the Babylonian method repeatedly adjusts the approximation of the square root by averaging it with the quotient of the number and the current approximation. Analogously, our algorithm incrementally updates the resistance matrix by applying a sequence of voltages, driving the system towards the matrix inverse.

We begin with a resistance matrix $R(t)$ and apply voltages $V_{ij}(t)$ across the memristors. We start with the simplest model of memristive dynamics [44] to explain the basic idea, but this model can be implemented also by clamping the gating voltage of an array of diodes as in [45]. We will later discuss how this method is affected when window functions are introduced.

The dynamics of the system are governed by the differential equation:

$$\frac{dR_{ij}(t)}{dt} = -\alpha R_{ij}(t) + C_{ij} + \frac{R_d}{\beta} V_{ij}(t) \quad (6)$$

where $R_{ij}(t)$ represents the resistance values, α and β are constants, and C_{ij} is a correction factor. Above, $V_{ij}(t)$ are voltages applied to the devices. Each memristive device operates in isolation during a given time step, allowing us to sequentially scan through the entire matrix. If a single device (i, j) is operated at any time, then we can imagine that sequentially one scans through a matrix of

devices. This type of operation can be implemented on a crossbar architecture with gating 1T1R as described before, to avoid the presence of sneak paths. This will be the assumption going forward.

We now describe the algorithm. Take the matrix of junction resistances $\mathbf{R}(t)$, some matrix A , and potentially some vector \mathbf{b} , with $\mathbf{R}, A \in \mathbb{R}^{N \times N}$ and $\mathbf{b} \in \mathbb{R}^N$. With proper characterization, memristive elements may be driven to evolve according to the general form

$$\frac{dR_{ij}(t)}{dt} = \alpha(-\sum_k M_{ik}R_{ik}(t) + E_{ik}), \quad (7)$$

or, simply in matrix notation,

$$\frac{d\mathbf{R}(t)}{dt} = \alpha(-\mathbf{MR}(t) + \mathbf{E}), \quad (8)$$

with $\mathbf{M}, \mathbf{E} \in \mathbb{R}^{N \times N}$ and scalar α which we now choose. The steady state of the evolution is $R^* = \lim_{t \rightarrow \infty} R(t) = M^{-1}\mathbf{E}$. Proofs of these statements are provided in App. A, and are obtained by inserting these expressions into the analytical solution of the dynamical system, which can be obtained by vectorizing the matrix system of equations. We note that the applied voltage requires the matrix multiplication \mathbf{MR} , but this operation can be done in one step on the crossbar.

Then for different choices of M, E , we may implement different operations. If A is known to be invertible, we may simply choose M, E in the following ways:

- $\mathbf{M} = \mathbf{A}, \mathbf{E} = \mathbf{I}$. $R^* = A^D$, the Drazin inverse of A , with $A^D = A^{-1}$ for invertible A .
- $\mathbf{M} = \mathbf{A}, \mathbf{E} = \mathbf{B}$. For a single \mathbf{b} , $B = [\mathbf{b} \ \mathbf{0}]$ and $R^* = [\mathbf{x} \ \mathbf{0}]$, where \mathbf{x} is the solution of $A\mathbf{x} = \mathbf{b}$ (etc. for different choices of input column). We note also that for $\mathbf{b}_1, \dots, \mathbf{b}_N$, choosing $B = [\mathbf{b}_1 \ \dots \ \mathbf{b}_N]$ would yield $R^* = [\mathbf{x}_1 \ \dots \ \mathbf{x}_N]$, i.e. a “parallel” solution of N problems.

For general A we may compute the Gram matrix $G = A^T A$ in $\Theta(N^2 + N)$ using crossbar dynamics, and then choose M, E in the following ways:

- $\mathbf{M} = \mathbf{G}, \mathbf{E} = \mathbf{A}^T$. R^* converges to the Moore-Penrose pseudoinverse of A , A^\dagger , for injective A , i.e. $A^\dagger = (A^T A)^{-1} A^T$.
- $\mathbf{M} = \mathbf{G}, \mathbf{E} = \mathbf{A}^T \mathbf{b}$. This is included for completeness. R^* converges to the least-norm solution of the linear system $A\mathbf{x} = \mathbf{b}$. In practice, one would drive $R^* = A^\dagger$ as above and then compute $A^\dagger \mathbf{b}$ as needed.

Online formulation. The fixed points of (7) are reached when $\mathbf{MR} = \mathbf{RM} = \mathbf{I}$. In particular, there exists a matrix \mathbf{R} such that $\mathbf{MR} = \mathbf{I}$ when \mathbf{M} is square. In these cases we can instead right-multiply, in contrast

to left-multiply, the input matrix \mathbf{M} with the crossbar state $\mathbf{R}(t)$:

$$\frac{d\mathbf{R}(t)}{dt} = \alpha(-\mathbf{R}(t)\mathbf{M} + \mathbf{E}) \quad (9)$$

When this is true, only one crossbar is required for inversion iteration—the current state $\mathbf{R}(t)$ is both stored and used to compute the forcing for the next iteration, an example of so-called “in-memory compute”. The iteration circuit may therefore be implemented as a feedback loop (Figure 1), which will be useful for the online algorithmic applications we discuss. The crossbar array enables us to implement and left- and right-multiply by applying bias on the left and right ends of the horizontal buses, respectively.

For square A , one may choose M, E in (9) in the following ways:

- $\mathbf{M} = \mathbf{A}, \mathbf{E} = \mathbf{I}$. If A is invertible, again $R^* = A^D$, the Drazin inverse of A , with $A^D = A^{-1}$ for invertible A . This is true because right and left inverses are identical for invertible square matrices.
- $\mathbf{M} = \mathbf{A}\mathbf{A}^T, \mathbf{E} = \mathbf{A}^T$. $R^* = A^\dagger$, the Moore-Penrose pseudoinverse of A .

B. SPICE implementation

In this section, we discuss the methods and the obstacles that come from implementing such an algorithm on chip, and their possible solution.

A standard crossbar requires memristive elements to be packaged in so-called 1T1R cells to allow for per-element read/write. In particular, the most efficient (naïve) scheme restricts concurrent read/write access to diagonals of the crossbar matrix, meaning that $O(N)$ distinct operations of some duration τ are required to fully access the crossbar state.

In addition to this restriction, we wish to evaluate the effects of common practical issues like tuning error, read noise, and response time limitations in devices, which also introduce variability and delay. We investigate these issues in a SPICE implementation of our algorithm, performing a parameter sweep to capture the dynamics. Simulations are performed with the PySpice library, a wrapper over the Ngspice simulator.

We demonstrate general convergence in nonvolatile crossbar arrays of Joglekar-windowed TiO_2 memristor [44, 46] model. Memristors are initialized with memory parameters uniformly drawn from the range $[0, 0.5]$. The voltage $V_{ij}(t)$ applied to every memristor takes the form $V_{ij}(t) = -\alpha(\mathbf{AR})_{ij}(t - \tau) + \alpha\delta_{ij}$, where τ represents the delay introduced by component limitations. The input matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ has entries drawn from $I - \frac{0.1}{N} \cdot U(N)$, where $U(N)$ denotes a matrix with entries drawn uniformly from $[0, 1]$. We observe that the array state $R(t)$ converges exponentially (Figure 3) towards the inverse

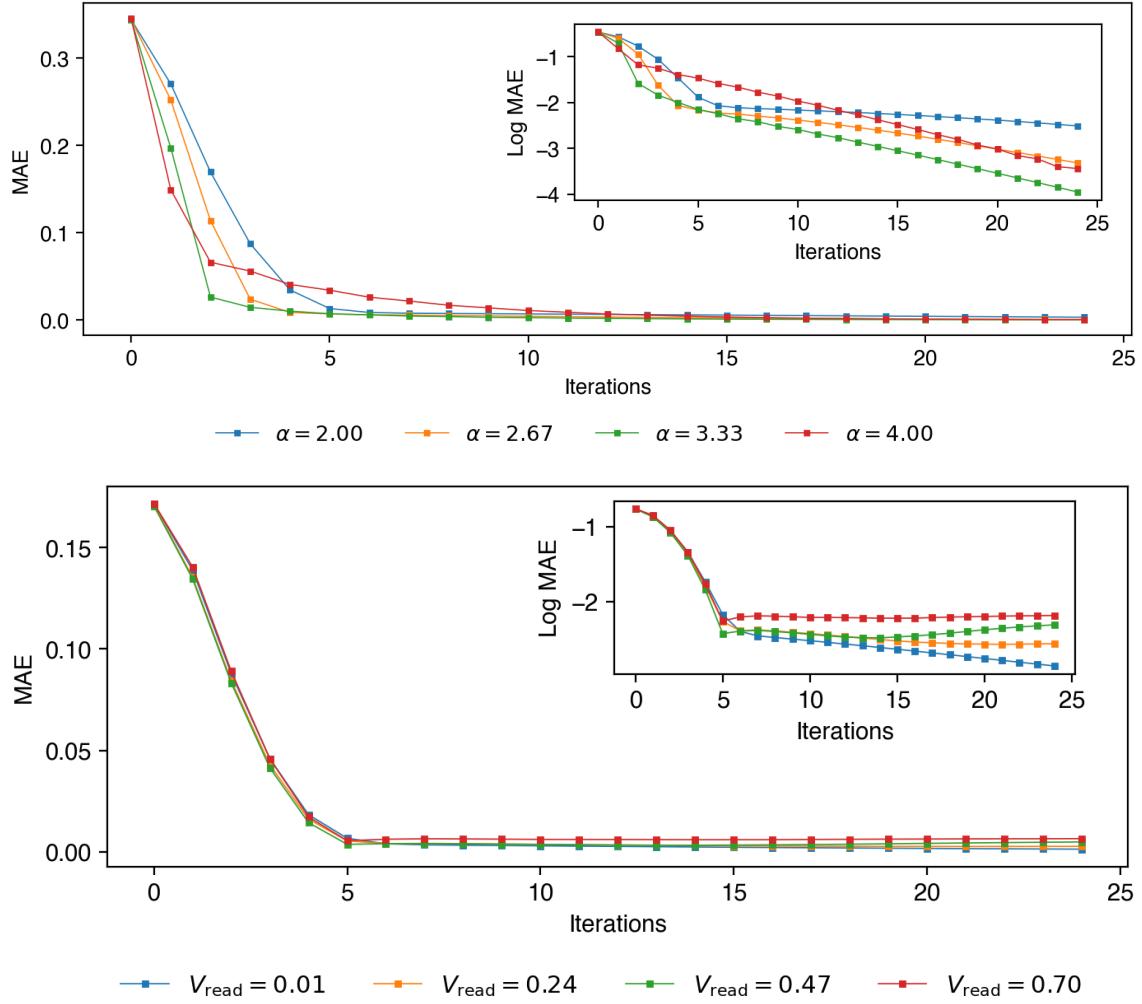


FIG. 3: SPICE simulation of convergence performance of basic linear memristor implementation for various α . The mean absolute error between the crossbar state $R(t)$ and the inverse A^{-1} converges exponentially to ϵ in α . Joglekar-windowed HP memristors with $p = 7$ are used in SPICE. **Inset:** Log convergence error (base 10).

A^{-1} with a speed proportional to α , measuring the distance between $\mathbf{R}(t)$ and \mathbf{A}^{-1} by computing the mean absolute error $\text{MAE}(\mathbf{R}, \mathbf{A}^{-1}) = \frac{1}{N^2} \sum_{ij} |R_{ij}(t) - A_{ij}^{-1}(t)|$.

We assume that signals to the crossbar array are sent as square wave pulses of some duration τ , with two relevant durations τ_{read} and τ_{write} for the respective stage within an iteration.

We next show the impact of read voltage V_{read} on the convergence of the algorithm, as higher V_{read} increases robustness to read noise but introduces an observer effect (Figure 3).

To gauge the time complexity of our algorithm, we fix a convergence error bound $\epsilon = 5e-3$ and consider the number of distinct read/writes required to drive the crossbar MAE to ϵ for increasing $N \in [2 \dots 10]$. We note that in crossbars each read/write is (in our scaling assumption) N atomic operations. Using this fact to obtain a loose analog to standard big- O computational complexity T

for a given N . We also fix $\alpha = 15$, $\tau_{\text{read}} = 4 \text{ ms}$, $\tau_{\text{write}} = 10 \text{ ms}$. The resulting scaling is shown in Figure 4.

We observe in simulation that for a certain class of diagonally-dominant asymmetric monotone matrices with fixed minimum eigenvalue 0.5, the iteration count converges to a constant C at large N ; the complexity of the algorithm is thus in $\Theta(3CN)$ or $O(N)$ for these matrices (Figure 4). Thus the algorithm is in $\Omega(N)$. For more general matrices, we observe an average-case complexity in $O(N^2)$ for random matrices (Fig. 4).

C. Analytical results

Computational complexity and advantages. Iterative matrix inversion methods become indispensable in the solution of large linear systems and often prove to be the only option compared to direct methods. Such

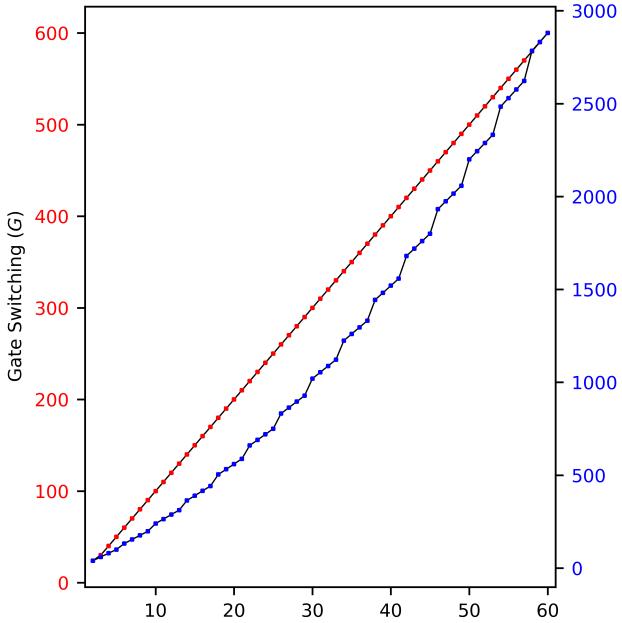


FIG. 4: Total number of gate switching operations G required for convergence to successfully invert matrices $\mathbf{A} \in \mathbb{R}^{N \times N}$. **Red.** Inverting random monotone matrices drawn from the distribution $A = \mathbf{I} - \frac{0.1}{N}U(N)$ results in a linear G in $O(N)$. **Blue.** Inverting random matrices with fixed minimum eigenvalue 1.2 and $\kappa \approx 3$ reveals a weak dependence of iteration count on matrix size, giving a general operation complexity in $O(N^2)$.

algorithms rely extensively on recursive applications of matrix multiplication, which is normally in $O(N^3)$. By contrast, we note that the dynamics of memristor crossbars allow for matrix multiplication in $O(N)$ via $N O(1)$ matrix-vector products. We therefore expect some computational advantage in memristive iterative matrix inversion.

We precisely define an atomic crossbar operation in Appendix D 4. By this definition, a single iteration of the algorithm has a complexity in $\Theta(3N)$. Our algorithm thus has an advantage as long as it converges to some error ϵ in less than $O(N^2)$ iterations. As with all iterative methods, this condition is predicated on $\kappa(\mathbf{A})$ and the extrema of the eigenspectrum of A , rather than its size N .

Consider an invertible matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ s.t. $A = Q\Lambda Q^{-1}$, with eigenvalues $\lambda_i = \{\lambda_1, \dots, \lambda_N\}$, $\lambda_i \in \mathbb{C}$ and $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_N)$. Then the rate of convergence $\tau\mu$ is bounded only by $\min_i |\Re(\lambda_i)|$. To see this, note from (A15) that convergence depends on the evolution of the matrix exponential integral $I(t) = \int_0^t \alpha e^{-\alpha t \Lambda} dt$ as $t \rightarrow \infty$. Since Λ is diagonal, each nonzero element λ_i evolves independently according to $\int_0^t \alpha e^{-\alpha t \lambda_i} dt$. If we fix a convergence bound ϵ , for a given λ_i the integral reaches $\frac{1}{\lambda_i} \pm \epsilon$ at time $t_i = -\ln|\epsilon|/\alpha\lambda_i$. Thus

over all independent convergence times $\{t_1, \dots, t_N\}$, we see that $t_{\max} = \text{argmax}_i t_i = \text{argmin}_i |\Re(\lambda_i)|$, and $I(t_{\max}) = \Lambda^{-1}$. Such a result is expected as matrices with near-zero eigenvalues are near-singular.

We then expect that for matrices with fixed minimum eigenvalues, the iteration count is independent of N , and thus that $\tau\mu \ll N$ for increasing problem sizes. Asymptotically, then, we expect that for matrices with amenable spectra, the time complexity is $\Theta(\tau\mu N) \in O(N)$ for the iterative portion of the algorithm. Indeed, we observe this effect in simulation (Figure 4). However, an underlying assumption is that the evolution of the dynamics can be integrated stably by our system, determined by the condition number $\kappa(A)$. Thus the convergence rate depends also on the maximum eigenvalue of A , in the sense that above some $\kappa_{\text{crit}}(A)$, the error will never decrease below ϵ .

System size dependence. In a physical crossbar with gated elements, one must also consider the finitely high impedance of nominally “open” switches, which introduce a small amount of crosstalk between elements which depends on size. This effect introduces a dependence on N that varies in the magnitude of the gating impedances, which we may call $R_{\text{open}}, R_{\text{closed}}$. In simulation, we find that $R_{\text{open}} \approx 10^9 \Omega$ is necessary to invert large matrices with the algorithm (i.e. matrices with $N \gtrsim 30$). We assume $R_{\text{closed}} \approx 0.05 \Omega$ without loss of generality, as different magnitudes may be absorbed into the rate coefficient α .

Optimal initialization. The closed-form solution of the dynamics (A12) contains a transient term $e^{-\alpha t \mathbf{A}} \mathbf{R}(0)$, where $\mathbf{R}(0)$ is the initial matrix of resistances in the crossbar. This transient term must go to zero during convergence, introducing additional time complexity. However, we can simply apply a strong reset voltage to all elements in $O(1)$ such that $\mathbf{R}(0) \approx \mathbf{0}$ before the start of the iteration. Initializing $\mathbf{R}(0)$ in this manner results in exponential convergence, with a rate independent of N when considering matrices with fixed spectral properties. This effect is expected given that the lower bounds on speed result from the eigenspectrum of A and our convergence bound ϵ , not N .

Voltage requirements. The element-wise magnitude of V_{write} is proportional to the error, meaning that it reaches its maximum value early in iteration and decreases exponentially thereafter. We thus have an upper bound on V_{write} for all steps a priori.

If we consider the maximum element $\max(\mathbf{M}) = \max_{ij} |M_{ij}|$ for some matrix \mathbf{M} , then we can find $\max(V_{\text{write}}(i))$ for a given iteration i . With $\mathbf{R}(0) \approx \mathbf{0}$ as described, clearly $V_{\text{write}}(0) \approx \alpha \mathbf{I}$ and therefore $\max(V_{\text{write}}(0)) \approx \alpha$, $\mathbf{R}(1) = \alpha \mathbf{I}$. Then $V_{\text{write}}(1) \approx -\alpha^2 \mathbf{A} + \alpha \mathbf{I}$ and $\max(V_{\text{write}}(1)) = |\alpha - \alpha^2 \max(\mathbf{A})|$. This is the maximum required voltage for inversion; afterwards $\max(V_{\text{write}}(t)) \sim e^{-\alpha t / \kappa(\mathbf{A})}$. The maximum voltage required to invert A is therefore proportional to the maximum element of \mathbf{A} and the speed with which we wish to invert it.

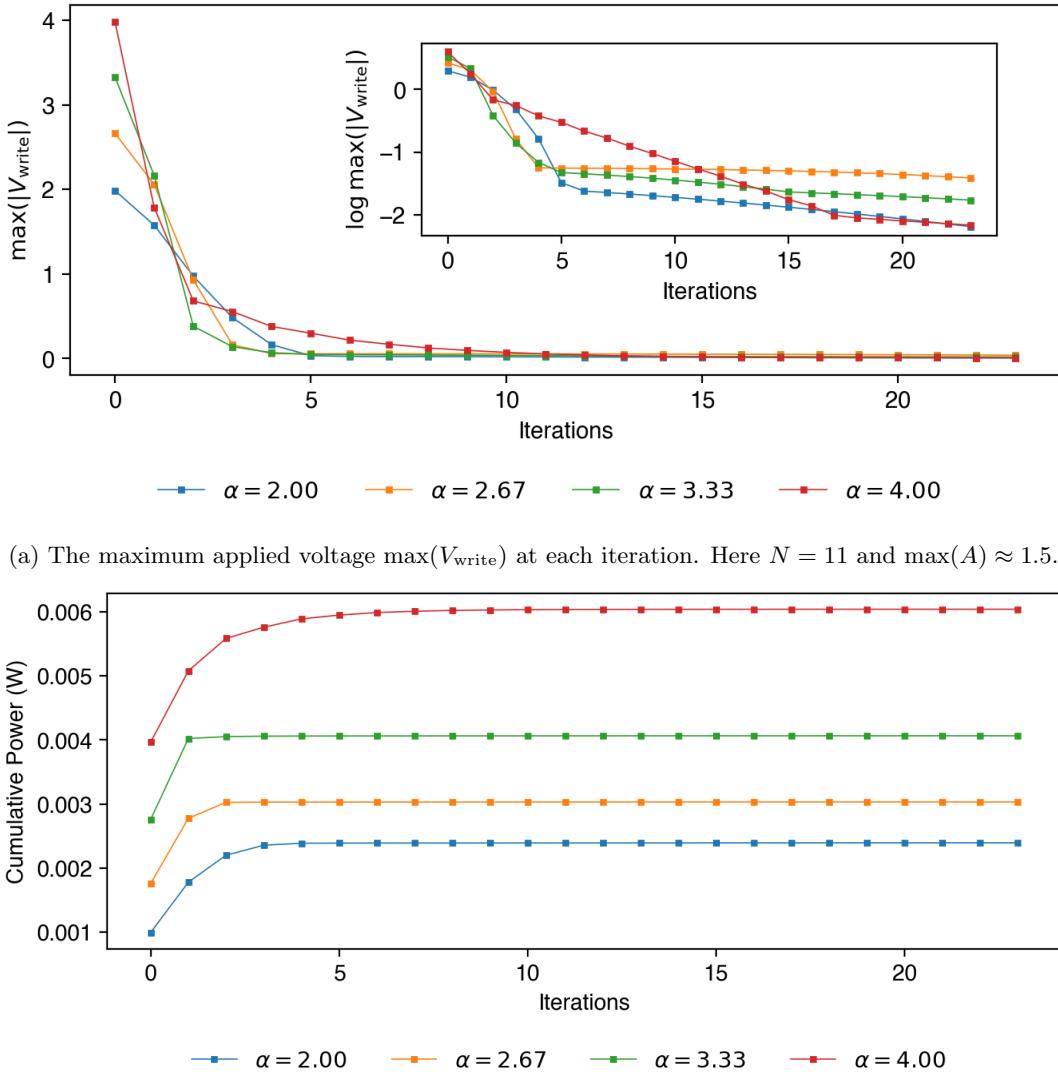


FIG. 5: Power consumption curves for matrix A with $N = 11$, $\max(A) \approx 1.5$ for various α .

Power consumption. In our simulations, the power consumption is well-approximated by considering the iteration sequence $\mathbf{R}_0, \mathbf{R}_1, \mathbf{R}_2, \dots$ and the input matrix \mathbf{A} . We may then compute $\mathbf{V}_i = -\alpha(-\mathbf{A}\mathbf{R}_{i-1} + \mathbf{I})$, taking $\mathbf{V}_0 = \alpha\mathbf{I}$ given $\mathbf{R}_0 \approx \mathbf{0}$. The cumulative power consumption at some iteration i is then $\sum_i \sum_{jk} \mathbf{V}_{jk}^2 / \mathbf{R}_{jk}$.

Because the algorithm converges exponentially to the state \mathbf{A}^{-1} , the elementwise magnitude of \mathbf{V} , which may be thought of as an error or correction term, decreases exponentially. Thus the cumulative power consumption converges exponentially to an upper bound, as shown in Fig. 5.

D. Stability and Error

In this section, we present a detailed analysis of the algorithms used in our study, highlighting the results presented in the appendix for further insights. One comment to make is that purely analog (continuous time) control does not exist. In practice, voltage is controlled in steps, and thus in many ways, the effective algorithm being implemented is a delayed differential equation. We consider the reduced dynamics of the crossbar inversion algorithm, a gradient flow that follows the dynamics

$$\frac{d\mathbf{R}(t)}{dt} = -\alpha \mathbf{A} \hat{\mathbf{R}}(t - \tau) + \alpha \mathbf{I} \quad (10)$$

where $\hat{\mathbf{R}}(t)$ represents the estimated value of the crossbar state from a read and $\tau \geq 0$ is a delay in the estimation of the state R . In systems where the error in $\hat{\mathbf{R}}(t)$ grows with time due to volatility ($\tau > 0$), such delay-induced error results in oscillations in the evolution of the crossbar state about the true inverse.

We can also consider the effects of observational noise (ξ_O) and process noise (ξ_P), writing

$$\frac{d\mathbf{R}(t)}{dt} = -\alpha \mathbf{A}(\mathbf{R}(t - \tau) + \xi_O) + \alpha \mathbf{I} + \xi_P \quad (11)$$

While process noise ξ_P can be averaged out or predicted with standard techniques, ξ_O enters as a nonlinear term. Its influence is proportional to the spectral radius $\rho(A)$. In practice, however, if the RMS noise magnitude is negligible compared to the magnitude of V_{read} , ξ_P may be ignored.

More specifically, the appendix contains detailed derivations and proofs of the key results used in our analysis. This proof demonstrates that under certain conditions, the algorithm converges to the true inverse of the matrix, leveraging the inherent properties of memristive devices, see App. A) for a detailed proof. The convergence proof is central to our algorithmic analysis, as it establishes the foundational guarantee that our method reliably produces the correct inverse under specified conditions. This proof utilizes techniques from dynamical systems theory to show that the feedback mechanism used in the memristive array ensures stability and convergence.

In App. B, the error analysis provides insights into how device imperfections affect the algorithm's performance asymptotically. By modeling the variability in memristive devices, we derive upper bounds on the error, which are crucial for understanding the practical limitations of our approach, including the Moore-Penrose and Drazin inverses. These two converge to the same matrix if the matrix is invertible, but the response to noise is different. If we assume that the right-hand side is perturbed by an i.i.d. noise $\xi(t)$ such that σ is the strength of the noise $\langle \xi^2 \rangle = \sigma^2$, then we obtain the elementwise error is given by

$$\langle \mathbf{R}^2(t) \rangle - \langle \mathbf{R}(t) \rangle^2 = \begin{cases} \frac{2\sigma^2}{\alpha} \int_0^t e^{-s\mathbf{A}} ds & \text{Drazin} \\ \frac{2\sigma^2}{\alpha} \int_0^t e^{-s\mathbf{A}^t \mathbf{A}} ds & \text{Moore-Penrose} \end{cases}$$

which, as we can see, depends on time and is element dependent, but it is proportional in both cases to $2\sigma^2/\alpha$. This implies that noise can be mitigated by running the analog at a slower pace, determined by the constant α .

In addition, we have studied the case in which volatility is present in the devices, and delays are present in the algorithm. Specifically, a comprehensive stability analysis is presented in App. C 2, quantifying the stability to non-idealities and volatility in the control, and on the accuracy of the computed inverse, both for Moore-Penrose

and Drazin inverses using stochastic analysis. In particular, we assume that the applied voltage has a delay τ with respect to the readout. We show the classification of the instability according to the location of the pole. In particular, for small values of the delay and for volatile devices stable oscillations occur. The oscillations become unstable for longer delays.

IV. APPLICATIONS

We now describe two applications where we can implement matrix inversion both offline and online. To be clear, in the offline version, all data is at the beginning of the algorithm, while in the online version, data arrives over time. In both cases we describe below, we can implement the matrix inversion.

A. Variational parameter learning

As an application of the analog inverse method, we consider the parameter learning of a Gaussian distribution from samples [47, 48]. The Gaussian distribution we consider can be parametrized in the form

$$p_\Sigma(\vec{x}) = \frac{1}{Z(\Sigma)} e^{-\frac{1}{2}\vec{x}^t \Sigma \vec{x}} \quad (12)$$

We assume that we want to learn the parameters Σ_{ij} from samples of the distribution, x_i^k where k represents the sampled values and i are the vector elements. A common method used to infer these parameters is the KL divergence minimization via gradient descent (see App. D 6), which leads to the equation

$$\frac{d(\Sigma_t)_{ab}}{dt} = \xi (\langle x_a x_b \rangle_{\text{emp}} - (\Sigma_t^{-1})_{ab}) \quad (13)$$

or rather, its Euler discretization

$$(\Sigma_{t+1})_{ab} = (\Sigma_t)_{ab} + dt \xi (\langle x_a x_b \rangle_{\text{emp}} - (\Sigma_t^{-1})_{ab}), \quad (14)$$

where Σ_t is the matrix at time t , ξ is the learning rate and $\langle x_a x_b \rangle_{\text{emp}}$ is the empirical average, e.g.

$$\langle x_a x_b \rangle_{\text{emp}} = \frac{1}{M} \sum_k x_a^k x_b^k, \quad (15)$$

where M is the total number of samples. With some sample matrix $X \in \mathbb{R}^{M \times D}$ at each iteration, this is also the averaged Gram matrix $\langle x_a x_b \rangle_{\text{emp}} = \frac{1}{M} X^T X$.

Notice that the gradient flow in (14) is essentially computing the quantity $\hat{\Sigma}_{ij} = \left\langle \left(\frac{1}{M} X^T X \right)^{-1} \right\rangle_{\text{emp}}$. Because the estimator $\hat{\Sigma}$ is square and invertible, we may thus slightly modify (9) to the form

$$\dot{R}(t) = \alpha (-R(t) \langle x_a x_b \rangle_{\text{emp}} + I) \quad (16)$$

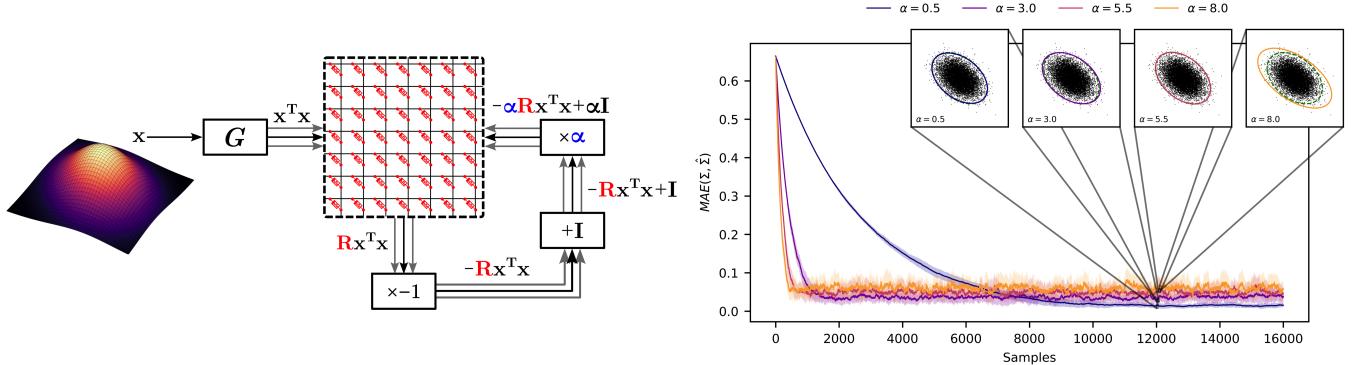


FIG. 6: With $M = 1$ —a single sample per iteration—we already achieve a good estimate $\hat{\Sigma}$ for Σ when the crossbar state evolves according to (16). The real-time equivalent of convergence is about 14 seconds, but it should be noted that this estimate evolves from a complete lack of knowledge of Σ . “Online” usage after initial convergence should converge very quickly, with only incremental updates to the crossbar state. The **FB: right** plot gives MAE curves for online inference with select values of $\alpha \in [0.5, 8]$, showing that higher α yield faster convergence to an estimate, albeit with larger fluctuations and less accuracy. The true distribution is marked by a dashed green line.

i.e. the “online” form of (7), such that $\langle R^* \rangle = \Sigma_{ij}$ with large enough M and small enough α . We find that in practice, excellent performance can already be achieved with $M = 2, \alpha = 1$ (Figure 6), making this approach a viable candidate for an online implementation of variational inference. A crossbar driven to such an estimator of A can then be used to transform another white noise source into a similar distribution, a useful operation in applications like noise cancellation.

B. From Offline to Online Reservoir Computing

The previous section described an online, iterative computation of the matrix inverse. We now consider its potential application to on-chip reservoir computing [49, 50]. In particular, we ask if this algorithm can be used for online learning—and computation—of the transformation on the reservoir via iterative pseudoinverse computation.

We give a brief overview of reservoir computing. A reservoir computer is a driven dynamical system that is trained to transform an input driving signal into an output trajectory. Given an input driving signal $\mathbf{u}(t)$ and target trajectory $\mathbf{z}(t) \in \mathbb{R}^O$, and assuming a system with state readout $\mathbf{s}(t) \in \mathbb{R}^D$ and sufficiently nonlinear F that evolves according to

$$\dot{\mathbf{s}}(t) = F(\mathbf{s}(t)) + \mathbf{u}(t) \quad (17)$$

we ask for a linear transformation $\mathbf{A} \in \mathbb{R}^{O \times D}$ such that $\mathbf{As}(t) \approx \mathbf{z}(t)$.

Commonly, the output states and corresponding target trajectories are discretely sampled at T times such that we have matrices $\mathbf{S} \in \mathbb{R}^{T \times D}, \mathbf{Z} \in \mathbb{R}^{T \times O}$. Then by standard linear least-squares arguments, an approximation for the transformation \mathbf{A} is given by the Moore-Penrose

pseudoinverse, i.e.

$$\arg \min_{\mathbf{A}} \|\mathbf{As}(t) - \mathbf{z}(t)\|^2 = [(\mathbf{S}^T \mathbf{S})^{-1} \mathbf{S}^T \mathbf{Z}]^T \quad (18)$$

We put the approximation in the form above $([\dots]^T)$ to ask if an online computation of (18) can be achieved with our method. By similar arguments as in the previous sections, we arrive at

$$\dot{\mathbf{R}}(t) = \alpha(-\mathbf{R}(t)(\mathbf{s}^T \mathbf{s} + \lambda \mathbf{I}) + \mathbf{z}^T \mathbf{s}) \quad (19)$$

where $\mathbf{s} \in \mathbb{R}^{1 \times D}, \mathbf{z} \in \mathbb{R}^{1 \times O}$ are now single samples of output states and target trajectories respectively, and λ is the familiar ridge regression control parameter. Given a suitably distributed sequence of paired (\mathbf{s}, \mathbf{z}) , this iterative least-squares regression indeed converges to (18) in an online fashion (Figure 7). We use the paired-crossbar architecture discussed in Appendix D 5 to address the mixed signs in the estimator $\hat{\mathbf{A}}$.

The algorithm also works when the desired mapping is rectangular, i.e. $A \in \mathbb{R}^{O \times D}$ and $O \ll D$. In such cases the gating transistors on the lower $D - O$ rows of the crossbar can be switched off for the full duration of execution, effectively zeroing those matrix elements. We use this fact to map the full reservoir state to a far smaller output dimensionality in an online fashion.

We note that (19) converges under similar assumptions as (16), i.e. that the samples $(\mathbf{s}, \mathbf{z}) \leftarrow \text{span}(A)$ arrive roughly i.i.d. in time. To see this, we fix $\lambda = 0$ and note that a fixed point of (19) exists when $\mathbf{R}^*(\mathbf{s}^T \mathbf{s}) = \mathbf{z}^T \mathbf{s}$. Suppose that the pairs (\mathbf{s}, \mathbf{z}) are drawn uniformly and randomly from the span of \mathbf{A} , such that we have random variables $\langle \mathbf{s}^T \mathbf{s} \rangle, \langle \mathbf{z}^T \mathbf{s} \rangle$. Assuming the existence of \mathbf{A} , the fixed point condition thus becomes $\mathbf{R}^*(\mathbf{s}^T \mathbf{s}) = \mathbf{A}(\mathbf{s}^T \mathbf{s})$, and we see that a fixed point is reached when the time average of the outer product $\langle \mathbf{s}^T \mathbf{s} \rangle$ becomes stationary. This requirement also becomes clear if one notes that

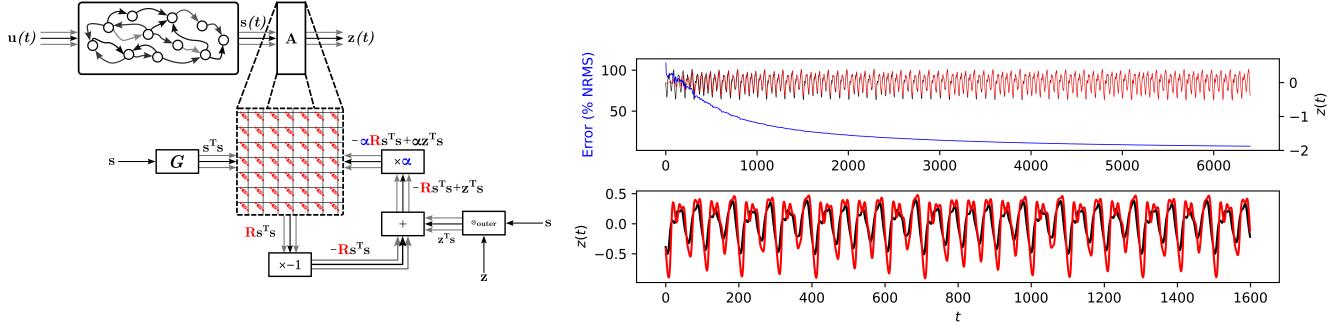


FIG. 7: Online reservoir computing on a constant stream of sampled (\mathbf{s}, \mathbf{z}) from an ESN. Training error converges to 1.679% as samples arrive. In the left figure, we plot the evolution of the error as a function of time, using an online learning scheme for a Mackey-Glass time series. At the beginning, the system is not tuned to fit the time series. The regression matrix is being learned as data arrives, and the cumulative NRMSE goes down accordingly. We can see a zoom of the final fit in the right plot, for the prediction mapping $\mathbf{s}(t) \mapsto \mathbf{z}(t + 5)$.

(19) essentially implements a first-order recursive least squares (RLS) filter.

These conditions can only be met by a reservoir which is sufficiently chaotic. In these systems, the Lyapunov exponent is large enough to effectively decorrelate consecutive samples of trajectories in time. We employ an echo state network (ESN) in ReservoirPy as the reservoir, with $N = 600$ neurons, leakage rate $\alpha = 0.99$, spectral radius $\rho = 0.99$, and reservoir connectivity $p = 0.7$. We train the reservoir on a Mackey-Glass sequence prediction task, preprocessing the input and target sequences by learning the mapping $\mathbf{s}(t) \mapsto \mathbf{u}(t + \tau), \tau \in [1 \dots 5]$. For Mackey-Glass, the control parameters $r = 0.2, \gamma = 0.1, \tau = 17, n = 10, x_0 = 0.1$ are used.

V. DISCUSSION

The advent of parallel computing has shifted the focus towards developing efficient algorithms for large-scale, distributed matrix inversion. In this paper, we present several key results that highlight the advantages of utilizing memristive crossbar arrays for matrix inversion.

Our main algorithm emulates a differential equation to iteratively update the resistance values in a memristive crossbar array, effectively solving the system of linear equations. We demonstrate that our algorithm converges reliably to the correct matrix inverse, leveraging the inherent properties of memristive devices. Moreover, we have provided error estimates in the presence of noise, and shown that the algorithm is stable in the present of delays, relevant to the case of volatile devices.

We validated our theoretical results through extensive simulations, showcasing the practical applicability and robustness of our approach. The empirical data supports our claims, showing that the memristive array-based method not only performs accurately but also offers considerable advantages in terms of speed and energy efficiency. Our simulations indicate that the cumulative

power consumption converges exponentially to an upper bound, demonstrating the energy efficiency of our approach. The simulations were conducted using SPICE, ensuring realistic device behavior and providing a strong foundation for the practical implementation of the algorithm.

One of the key advantages of our method is its scalability. The number of operations required for convergence scales linearly with the matrix size, which is a significant improvement over traditional methods that often exhibit quadratic or higher-order complexity. This makes our approach particularly suitable for large-scale problems where conventional methods become computationally prohibitive.

We also highlight that our algorithm can handle both positive and negative entries in the matrix, which is a crucial capability for real-world applications. Theoretical results with noise show that our algorithm maintains stability and converges to the true inverse of the matrix under certain conditions. We provide upper bounds on the error, demonstrating that noise can be mitigated by adjusting the algorithm's pace (e.g. the applied voltage).

Furthermore, we provide analytical results that underpin the convergence and stability of our algorithm, reinforcing the reliability of our approach. The ability to handle observational and process noise with analytical guarantees adds to the robustness of our method.

Our analysis extends to non-invertible matrices, where we successfully apply our method to compute the Moore-Penrose inverse and the Drazin inverse. This flexibility underscores the robustness of our approach and its applicability to a wide range of linear algebra problems.

Moreover, in terms of implementation schemes, we demonstrate that our algorithm can be adapted to solve both for A^{-1} and for N parallel instances of the type $A^{-1}\vec{b}$ when the matrix is invertible, offering versatility in its application. This dual capability allows for broader use in various computational contexts.

The numerical simulations we provided corroborate

our theoretical findings, showing that our method performs well even with realistic device imperfections. The results indicate that memristive crossbar arrays can be effectively used for matrix inversion, providing a scalable, energy-efficient alternative to traditional methods.

Most importantly, the present manuscript shows that the use of crossbar arrays with memristive devices can be used both for online and offline matrix inversion. We have tested these schemes on two applications, i.e. parameter learning of multi-variate Gaussian distributions and reservoir computing. Specifically, we have shown that our method rapidly converges to the optimal solution.

In summary, our study demonstrates that memristive crossbar arrays offer a promising approach for efficient matrix inversion, with potential applications in various fields requiring large-scale linear algebra computations.

The results presented in this paper provide a strong foundation for further research and development in this area. In our future work, we will focus on the physical implementation of the algorithm, exploring its practical deployment in more realistic case studies, and on-chip.

ACKNOWLEDGMENTS

The authors acknowledge the support of NNSA for the U.S. DoE at LANL under Contract No. DE-AC52-06NA25396, and Laboratory Directed Research and Development (LDRD). FC and JL were financed via DOE LDRD grant 20240245ER, while FB via Director's Fellowship. FB and JL also gratefully acknowledge support from the Center for Nonlinear Studies at LANL. The authors are indebted to A. Jayakumar for some enlightening discussions on parameter learning.

-
- [1] G. H. Golub and C. F. van Loan, *Matrix Computations*, 4th ed. (JHU Press, 2013).
 - [2] Ø. Ryan, Linear algebra, signal processing, and wavelets - a unified approach, Springer Undergraduate Texts in Mathematics and Technology (2019).
 - [3] M. T. Heath, *Scientific Computing: An Introductory Survey, Revised Second Edition* (Society for Industrial and Applied Mathematics, 2018).
 - [4] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. (Cambridge University Press, 2007).
 - [5] C. C. Aggarwal, *Linear Algebra and Optimization for Machine Learning - A Textbook* (Springer, 2020) pp. 1–495.
 - [6] B. N. Datta, Linear and numerical linear algebra in control theory: some research problems, *Linear Algebra and its Applications* **197–198**, 755–790 (1994).
 - [7] S. C. Althoen and R. McLaughlin, Gauss-jordan reduction: A brief history, *The American Mathematical Monthly* **94**, 130–142 (1987).
 - [8] A. Krishnamoorthy and D. Menon, Matrix inversion using cholesky decomposition, in *2013 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)* (2013) pp. 70–72.
 - [9] Y. Wang, D. Dai, and Y. Wang, Characterization and modeling of the power consumption of gpus for server workloads, *IEEE Transactions on Computers* **70**, 1625 (2021).
 - [10] For instance, the NVIDIA GeForce RTX 3090 has an estimated power consumption of approximately 350 watts (W), while the NVIDIA GeForce RTX 3080 consumes around 320 W. Similarly, the AMD Radeon RX 6900 XT and AMD Radeon RX 6800 XT both have estimated power consumptions of approximately 300 W each.
 - [11] J. Yu, H. A. D. Nguyen, L. Xie, M. Taouil, and S. Hamdioui, Memristive devices for computation-in-memory, in *2018 Design, Automation and Test in Europe Conference and Exhibition (DATE)* (2018) pp. 1646–1651.
 - [12] F. Alibart, E. Zamanidoost, and D. B. Strukov, Pattern classification by memristive crossbar circuits using ex situ and in situ training, *Nature Communications* **4**, 10.1038/ncomms3072 (2013).
 - [13] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinali, J. J. Yang, W. Wu, X. Li, W. M. Tong, D. B. Strukov, G. S. Snider, G. Medeiros-Ribeiro, and R. S. Williams, Memristor-cmos hybrid integrated circuits for reconfigurable logic, *Nano Letters* **9**, 3640–3645 (2009).
 - [14] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, Nanoscale memristor device as synapse in neuromorphic systems, *Nano Letters* **10**, 1297–1301 (2010).
 - [15] S. Pi, C. Li, H. Jiang, W. Xia, H. Xin, J. J. Yang, and Q. Xia, Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension, *Nature Nanotechnology* **14**, 35–39 (2018).
 - [16] D. V. Christensen, R. Dittmann, B. Linares-Barranco, A. Sebastian, M. Le Gallo, A. Redaelli, S. Slesazeck, T. Mikolajick, S. Spiga, S. Menzel, I. Valov, G. Milano, C. Ricciardi, S.-J. Liang, F. Miao, M. Lanza, T. J. Quill, S. T. Keene, A. Salleo, J. Grollier, D. Marković, A. Mizrahi, P. Yao, J. J. Yang, G. Indiveri, J. P. Strachan, S. Datta, E. Vianello, A. Valentian, J. Feldmann, X. Li, W. H. P. Pernice, H. Bhaskaran, S. Furber, E. Neftci, F. Scherr, W. Maass, S. Ramaswamy, J. Tapson, P. Panda, Y. Kim, G. Tanaka, S. Thorpe, C. Bartolozzi, T. A. Cleland, C. Posch, S. Liu, G. Panuccio, M. Mahmud, A. N. Mazumder, M. Hosseini, T. Mohsenin, E. Donati, S. Tolu, R. Galeazzi, M. E. Christensen, S. Holm, D. Ielmini, and N. Pryds, 2022 roadmap on neuromorphic computing and engineering, *Neuromorphic Computing and Engineering* **2**, 022501 (2022).
 - [17] Q. Xia and J. J. Yang, Memristive crossbar arrays for brain-inspired computing, *Nature Materials* **18**, 309–323 (2019).
 - [18] W. G. Kim, H. M. Lee, B. Y. Kim, K. H. Jung, T. G. Seong, S. Kim, H. C. Jung, H. J. Kim, J. H. Yoo, H. D. Lee, S. G. Kim, S. Chung, K. J. Lee, J. H. Lee, H. S. Kim, S. H. Lee, J. Yang, Y. Jeon, and R. S. Williams,

- Nbo-based low power and cost effective 1s1r switching for high density cross point reram application, in *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers* (IEEE, 2014).
- [19] K. Ohba, S. Yasuda, T. Mizuguchi, H. Sei, T. Tsushima, M. Shimuta, T. Shiimoto, T. Yamamoto, T. Sone, S. Nonoguchi, A. Kouchiyama, W. Otsuka, K. Aratani, and K. Tsutsui, Cross point cu-reram with bc-doped selector, in *2018 IEEE International Memory Workshop (IMW)* (IEEE, 2018).
- [20] J. Woo, X. Peng, and S. Yu, Design considerations of selector device in cross-point rram array for neuromorphic computing, in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (IEEE, 2018).
- [21] Z. Wang, S. Joshi, S. Savel'ev, W. Song, R. Midya, Y. Li, M. Rao, P. Yan, S. Asapu, Y. Zhuo, H. Jiang, P. Lin, C. Li, J. H. Yoon, N. K. Upadhyay, J. Zhang, M. Hu, J. P. Strachan, M. Barnell, Q. Wu, H. Wu, R. S. Williams, Q. Xia, and J. J. Yang, Fully memristive neural networks for pattern classification with unsupervised learning, *Nature Electronics* **1**, 137–145 (2018).
- [22] W. Song, M. Rao, Y. Li, C. Li, Y. Zhuo, F. Cai, M. Wu, W. Yin, Z. Li, Q. Wei, S. Lee, H. Zhu, L. Gong, M. Barnell, Q. Wu, P. A. Beerel, M. S.-W. Chen, N. Ge, M. Hu, Q. Xia, and J. J. Yang, Programming memristor arrays with arbitrarily high precision for analog computing, *Science* **383**, 903–910 (2024).
- [23] A. Mehonic and D. Jokšas, Emerging nonvolatile memories for machine learning (2023).
- [24] Z. Sun, G. Pedretti, E. Ambrosi, A. Bricalli, W. Wang, and D. Ielmini, Solving matrix equations in one step with cross-point resistive arrays, *Proceedings of the National Academy of Sciences* **116**, 4123–4128 (2019).
- [25] V. Strassen, Gaussian elimination is not optimal, *Numerische Mathematik* **13**, 354–356 (1969).
- [26] V. Y. Pan, Strassen's algorithm is not optimal trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations, in *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)* (1978) pp. 166–176.
- [27] D. Coppersmith and S. Winograd, On the asymptotic complexity of matrix multiplication, *SIAM Journal on Computing* **11**, 472–492 (1982).
- [28] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, in *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of Computing* (ACM, 1987).
- [29] H. Cohn and C. Umans, A group-theoretic approach to fast matrix multiplication, in *Proceedings of the 44th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society, Cambridge, MA, 2003) pp. 438–449.
- [30] N. J. Higham, *Accuracy and Stability of Numerical Algorithms* (Society for Industrial and Applied Mathematics, 2002).
- [31] A. W. Harrow, A. Hassidim, and S. Lloyd, Quantum algorithm for linear systems of equations, *Physical Review Letters* **103**, 10.1103/physrevlett.103.150502 (2009).
- [32] Y. Subaşı, R. D. Somma, and D. Orsucci, Quantum algorithms for systems of linear equations inspired by adiabatic quantum computing, *Physical Review Letters* **122**, 10.1103/physrevlett.122.060504 (2019).
- [33] G. E. Forsythe and S. Leibler, Matrix inversion by a monte carlo method, *Mathematical Tables and Other Aids to Computation* **13**, 166 (1958).
- [34] Y. S. Wu and N. Halko, *Randomized Algorithms for Scientific Computing* (SIAM, 2021).
- [35] K. K. Sabelfeld, *Monte Carlo Methods in Boundary Value Problems* (Springer, 2010).
- [36] S. Bartolucci, F. Caccioli, F. Caravelli, and P. Vivo, “spectrally gapped” random walks on networks: a mean first passage time formula, *SciPost Physics* **11**, 10.21468/scipostphys.11.5.088 (2021).
- [37] S. Bartolucci, F. Caccioli, F. Caravelli, and P. Vivo, Ranking influential nodes in networks from aggregate local information, *Physical Review Research* **5**, 10.1103/physrevresearch.5.033123 (2023).
- [38] S. Bartolucci, F. Caravelli, F. Caccioli, and P. Vivo, Distribution of centrality measures on undirected random networks via cavity method (2024).
- [39] M. Aifer, K. Donatella, M. H. Gordon, S. Duffield, T. Ahle, D. Simpson, G. E. Crooks, and P. J. Coles, Thermodynamic linear algebra (2023).
- [40] Y. Saad, *Iterative Methods for Sparse Linear Systems* (Society for Industrial and Applied Mathematics, 2003).
- [41] W. Wiggers, V. Bakker, A. Kokkeler, and G. Smit, Implementing the conjugate gradient algorithm on multi-core systems, in *2007 International Symposium on System-on-Chip* (IEEE, 2007).
- [42] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, Sparse matrix solvers on the gpu: conjugate gradients and multigrid, in *ACM SIGGRAPH 2005 Courses on - SIGGRAPH '05*, SIGGRAPH '05 (ACM Press, 2005).
- [43] This fact is often overlooked in quantum computing analysis.
- [44] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, The missing memristor found, *Nature* **453**, 80–83 (2008).
- [45] M. Stern, D. Hexner, J. W. Rocks, and A. J. Liu, Supervised learning in physical networks: From machine learning to learning machines, *Physical Review X* **11**, 10.1103/physrevx.11.021045 (2021).
- [46] Y. N. Joglekar and S. J. Wolf, The elusive memristor: properties of basic electrical circuits, *European Journal of Physics* **30**, 661–675 (2009).
- [47] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms* (Copyright Cambridge University Press, 2003).
- [48] C. W. Fox and S. J. Roberts, A tutorial on variational bayesian inference, *Artificial Intelligence Review* **38**, 85–95 (2011).
- [49] M. Lukoševičius and H. Jaeger, Reservoir computing approaches to recurrent neural network training, *Computer Science Review* **3**, 127–149 (2009).
- [50] M. Cucchi, S. Abreu, G. Ciccone, D. Brunner, and H. Kleemann, Hands-on reservoir computing: a tutorial for practical implementation, *Neuromorphic Computing and Engineering* **2**, 032002 (2022).
- [51] D. Showalter, Representation and computation of the pseudoinverse, *Proceedings of the American Mathematical Society* **18**, 584–586 (1967).

Appendix A: Basic derivations

Each memristor follows an equation of the form

$$\frac{dR_{ij}}{dt} = -\alpha R_{ij} + C_{ij} + V_{ij}(t) \quad (\text{A1})$$

and we ask for solutions such that $R_{ij}^* = (\mathbf{A}^{-1})_{ij}$. Let us assume that $0 \leq R_{ij} \leq \infty$. The values of C_{ij} are the natural resistance values at equilibrium.

Let us choose $\mathbf{V} = \alpha(-\mathbf{A} + \mathbf{B})\mathbf{R} - \mathbf{C} + \mathbf{M}$, and let \mathbf{A} be a monotone matrix, e.g. \mathbf{A}^{-1} be positive element-wise. Choosing $\mathbf{M} = \alpha^* \mathbf{I}$ provides the inverse of \mathbf{A} , while choosing $\mathbf{M} = \text{diag}(\vec{b})$ provides the solution of $\mathbf{A}\vec{x} = \vec{b}$ if \mathbf{A} is invertible.

Let us consider

$$0 = -\alpha\mathbf{R} + \mathbf{C} + \alpha(-\mathbf{A} + \mathbf{B})\mathbf{R} - \mathbf{C} + \mathbf{M} \quad (\text{A2})$$

It follows that

$$\alpha(\mathbf{I} + \mathbf{A} - \mathbf{B})\mathbf{R} = \mathbf{M} \quad (\text{A3})$$

and thus

$$\mathbf{R} = -\frac{1}{\alpha}(\mathbf{I} + \mathbf{A} - \mathbf{B})^{-1}\mathbf{M}. \quad (\text{A4})$$

If we choose $\mathbf{B} = \mathbf{I}$, then

$$\mathbf{R} = \frac{1}{\alpha}\mathbf{A}^{-1}\mathbf{M}. \quad (\text{A5})$$

For $M = \alpha I$, then the solution is directly the matrix inverse. Let us choose $M_{ij} = \alpha * b_i \delta_{j1}$. Then

$$R_{rt} = \sum_k (A^{-1})_{rk} b_k \delta_{t1} \quad (\text{A6})$$

Thus, the memristive devices $R_{i1} = x_i$.

Let us consider the vectorized version of the differential equation. To avoid doubts, want to show that we can write the matrix solution as we would write the vector solution of the differential equation. We define $\text{vec}(M)$ to be the vector with the columns of M stacked. Let us call $\vec{r} = \text{vec}(R)$ and $\vec{e} = \text{vec}(I_N)$. Then, using the fact for the matrix multiplication we have

$$\text{vec}(AR) = (I_N \otimes A)\vec{r} \equiv \mathcal{A}\vec{r}. \quad (\text{A7})$$

and the differential equation can be written in the form

$$\frac{d}{dt}\vec{r} = -\alpha\mathcal{A}\vec{r} + \alpha\vec{e}. \quad (\text{A8})$$

Let us briefly discuss on the stability. We consider the following system of gradient descent dynamical equations

$$\frac{dx_{ij}}{dt} = -\frac{\partial}{\partial x_{ij}}\mathcal{L}(A, x) \quad (\text{A9})$$

where

$$\mathcal{L}(A, x) = \frac{1}{2} \sum_{ijlm} \left(\sum_k A_{ik} x_{kj} - \delta_{ij} \right) A_{jl}^{-1} \left(\sum_r A_{lr} x_{rm} - \delta_{lm} \right).$$

from which we obtain, for \mathbf{A} symmetric, invertible and positive, that

$$\frac{\delta \mathcal{L}}{\delta x_{ij}} = - \sum_k A_{ik} x_{kj} + \delta_{ij}. \quad (\text{A10})$$

As a result, the continuous-time algorithm is stable.

1. Drazin inverse

In the case when we choose $\mathbf{M} = \alpha\mathbf{I}$, we also see that the matrix \mathbf{R} converges to \mathbf{A}^{-1} by assuming a diagonalizable \mathbf{A} and solving for the closed form solution of the matrix differential equation $\dot{\mathbf{R}} = -\alpha\mathbf{A}\mathbf{R} + \alpha\mathbf{I}$, where we discard the \mathbf{B} and \mathbf{C} terms by similar steps as above and substitute \mathbf{M} . The solution is then

$$\mathbf{R}(t) = e^{-\alpha t \mathbf{A}} \mathbf{R}(0) + \int_0^t e^{(t-\tau)(-\alpha \mathbf{A})} \alpha \mathbf{I} d\tau \quad (\text{A11})$$

$$= e^{-\alpha t \mathbf{A}} \mathbf{R}(0) + \alpha \int_0^t e^{-\alpha(t-\tau)\mathbf{A}} d\tau \quad (\text{A12})$$

For positive semi-definite \mathbf{A} (note $-\alpha$) the constant term vanishes and the limiting value \mathbf{R}_∞ becomes

$$\lim_{t \rightarrow \infty} \mathbf{R}(t) = \mathbf{R}_\infty = \alpha \int_0^\infty e^{-\alpha t \mathbf{A}} dt \quad (\text{A13})$$

Now we assume that \mathbf{A} is symmetric. We can then diagonalize it via $\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^{-1}$, yielding

$$\mathbf{R}_\infty = \alpha \int_0^\infty \mathbf{P} e^{-\alpha t \mathbf{D}} \mathbf{P}^{-1} dt \quad (\text{A14})$$

$$= \alpha \mathbf{P} \left(\int_0^\infty e^{-\alpha t \mathbf{D}} dt \right) \mathbf{P}^{-1} \quad (\text{A15})$$

$$= \alpha \mathbf{P} \left(-\frac{1}{\alpha} \mathbf{D}^{-1} e^{-\alpha t \mathbf{D}} \Big|_0^\infty \right) \mathbf{P}^{-1} \quad (\text{A16})$$

$$= \alpha \mathbf{P} \left(-\frac{1}{\alpha} \mathbf{D}^{-1} (-I) \right) \mathbf{P}^{-1} \quad (\text{A17})$$

$$= \mathbf{P} \mathbf{D}^{-1} \mathbf{P}^{-1} = \mathbf{A}^{-1} \quad (\text{A18})$$

where the improper integral converges because \mathbf{A} is positive semi-definite. We see then that this flow algorithm converges to the inverse because dynamically, it inverts the eigenvalues of \mathbf{A} . This also allows us to observe that the convergence time of the algorithm is dependent on the decay parameter α and the minimum eigenvalue λ_{\min} of \mathbf{A} , in addition to the time required to relax from the initial transient state.

2. Moore-Penrose pseudo-inverse

It is known (see Showalter [51]) that the integral representation of the Moore-Penrose pseudoinverse is

$$\mathbf{A}^\dagger = \int_0^\infty e^{(-\mathbf{A}^* \mathbf{A})t} \mathbf{A}^* dt \quad (\text{A19})$$

Then by knowledge of the exact solution in (10) and using identical steps, we can see that to compute the pseudoinverse, the flow equation for the pseudoinverse should take the form

$$\dot{\mathbf{R}} = -\alpha(\mathbf{A}^* \mathbf{A})\mathbf{R} + \alpha\mathbf{A}^* \quad (\text{A20})$$

after discarding the \mathbf{B} and \mathbf{C} terms, meaning that the initial form of the forcing \mathbf{V} should be

$$\mathbf{V} = \alpha(-\mathbf{A}^* \mathbf{A} + B)\mathbf{R} - \mathbf{C} + \alpha\mathbf{A}^* \quad (\text{A21})$$

Solving for eqn. (A1) with this forcing, we find the steady-state solution is

$$\mathbf{R}(\infty) = \alpha \int_0^\infty e^{-\alpha(\mathbf{A}^* \mathbf{A})t} \mathbf{A}^* dt = \mathbf{A}^\dagger \quad (\text{A22})$$

fitting Showalter's representation for \mathbf{A}^\dagger [51]. Note that the coefficient α is necessary before the $\mathbf{A}^* \mathbf{A}$ term in the forcing in order to reduce the equation to the correct one.

Appendix B: Stochastic analysis

1. Perturbations to the Drazin inverse

We start with the vectorized equation

$$\frac{d\vec{r}}{dt} = -\alpha \mathcal{A}\vec{r} + \alpha \vec{e} + \vec{\xi}(t) \quad (\text{B1})$$

where

$$\langle \xi_{ij}(t) \rangle = 0 \quad (\text{B2})$$

$$\langle \xi_{ij}(t)\xi_{kl}(t') \rangle = 2\sigma^2 \delta_{ik}\delta_{jl}\delta(t-t') \quad (\text{B3})$$

We have

$$\vec{r}(t) = e^{-\alpha t \mathcal{A}}\vec{r}(0) + \int_0^t e^{(t-\tau)(-\alpha \mathcal{A})}(\alpha \vec{e} + \vec{\xi}(\tau)) d\tau \quad (\text{B4})$$

Let us make a change of variables inside the integral, and write $\tau = t - s$, $ds = -d\tau$. Then

$$\int_0^t e^{\alpha A(\tau-t)} d\tau = - \int_t^0 e^{-s\alpha A} ds = \int_0^t e^{-s\alpha A} ds. \quad (\text{B5})$$

$$\vec{r}(t) = e^{-\alpha t \mathcal{A}}\vec{r}(0) + \int_0^t e^{-s\alpha \mathcal{A}}(\alpha \vec{e} + \vec{\xi}(t-s)) ds \quad (\text{B6})$$

Now, we have

$$\langle \vec{r}(t) \rangle = e^{-\alpha t \mathcal{A}}\vec{r}(0) + \int_0^t e^{-s\alpha \mathcal{A}}(\alpha \vec{e}) ds \quad (\text{B7})$$

and

$$\langle r(t) \rangle = e^{-\alpha t \mathcal{A}}\vec{r}(0) + \alpha \int_0^t e^{-s\alpha \mathcal{A}}\vec{e} ds \quad (\text{B8})$$

Now we use the fact that $\mathcal{A} = I_N \otimes A$. We have $e^{-\alpha t \mathcal{A}} = I_N \otimes e^{-t\alpha A}$. As a result, $\text{mat}(e^{-\alpha t \mathcal{A}}\vec{r}) = e^{-t\alpha A}R$. Then

$$\mathbf{R}(t) = e^{-\alpha t \mathbf{A}}\mathbf{R}(0) + \int_0^t e^{-s\alpha \mathbf{A}}(\alpha \mathbf{I} + \vec{\xi}(t-s)) ds \quad (\text{B9})$$

The solution is written then as

$$\langle \mathbf{R}(t) \rangle = e^{-\alpha t \mathbf{A}}\mathbf{R}(0) + \alpha \int_0^t e^{-\alpha \mathbf{A}s} ds \quad (\text{B10})$$

Now, let us add and subtract

$$\mathbf{R}(\infty) = \alpha \int_0^\infty e^{-\alpha \mathbf{A}s} ds = \int_0^\infty e^{-\mathbf{A}s} ds = \mathbf{A}^D \quad (\text{B11})$$

is the definition of the Drazin inverse of the matrix \mathbf{A} . If the matrix \mathbf{A} is invertible, then $\mathbf{A}^D = \mathbf{A}^{-1}$. We have

$$\mathbf{R}(\infty) - \langle \mathbf{R}(t) \rangle = \alpha \int_t^\infty e^{-\alpha \mathbf{A}s} ds - e^{-\alpha t \mathbf{A}}\vec{R}(0) \quad (\text{B12})$$

Let us now look at the fluctuations. We introduce the anticommutator $\{\mathbf{A}, \mathbf{B}\} = \mathbf{AB} + \mathbf{BA}$.

We have, ignoring terms that are linear in ξ ,

$$\langle \mathbf{R}(t)\mathbf{R}(t') \rangle = \{e^{-\alpha t \mathbf{A}}\mathbf{R}(0), e^{-\alpha t' \mathbf{A}}\mathbf{R}(0)\} \quad (\text{B13})$$

$$\begin{aligned} &+ \alpha \left(e^{-\alpha t \mathbf{A}}\mathbf{R}(0) \int_0^{t'} e^{-s' \mathbf{A}} ds' \right. \\ &\left. + e^{-\alpha t' \mathbf{A}}\mathbf{R}(0) \int_0^t e^{-s \mathbf{A}} ds \right) \end{aligned} \quad (\text{B14})$$

$$+ \int_0^t \int_0^{t'} ds ds' \langle \{e^{-s \alpha \mathbf{A}}(\alpha I + \xi(t-s)), e^{-s' \alpha \mathbf{A}}(\alpha \mathbf{I} + \xi(t'-s'))\} \rangle$$

Now note that

$$\langle \{e^{-s \alpha \mathbf{A}}(\alpha \mathbf{I} + \xi(t-s)), e^{-s' \alpha \mathbf{A}}(\alpha \mathbf{I} + \xi(t'-s'))\} \rangle = \alpha^2 e^{-(s+s')\alpha \mathbf{A}} \quad (\text{B15})$$

$$+ \langle \{e^{-s \alpha \mathbf{A}}\xi(t-s), e^{-s' \alpha \mathbf{A}}\xi(t'-s')\} \rangle \quad (\text{B16})$$

Since the integral is symmetric, we can simply focus on

$$\mathbf{M} = 2 \langle e^{-s \alpha \mathbf{A}}\xi(t-s)e^{-s' \alpha \mathbf{A}}\xi(t'-s') \rangle \quad (\text{B17})$$

Let us make the indices explicit (we call the element ij of the matrix e^M , e_{ij}^M below):

$$\langle M_{ij} \rangle = 2 \langle \sum_{klm} e_{ik}^{-s \alpha A} \xi_{kl}(t-s) e_{lm}^{-s' \alpha A} \xi_{mj}(t'-s') \rangle \quad (\text{B18})$$

$$= 2 \sum_{klm} e_{ik}^{-s \alpha A} e_{lm}^{-s' \alpha A} \langle \xi_{kl}(t-s) \xi_{mj}(t'-s') \rangle \quad (\text{B19})$$

$$= 4\sigma^2 \sum_{klm} e_{ik}^{-s \alpha A} e_{lm}^{-s' \alpha A} \delta_{km} \delta_{lj} \delta(t-t'+s-s') \quad (\text{B20})$$

$$= 4\sigma^2 \sum_k e_{ik}^{-s \alpha A} e_{kj}^{-s' \alpha A^t} \delta(t-t'+s-s') \quad (\text{B21})$$

Thus

$$\langle \mathbf{M} \rangle = 4\sigma^2 e^{-s \alpha \mathbf{A}} e^{-s' \alpha \mathbf{A}^t} \delta(t-t'+s-s') \quad (\text{B22})$$

where A^t is the transpose. Thus

$$\begin{aligned} \langle \mathbf{R}(t)\mathbf{R}(t') \rangle &= \{e^{-\alpha t \mathbf{A}}\mathbf{R}(0), e^{-\alpha t' \mathbf{A}}\mathbf{R}(0)\} \\ &+ \alpha \left(e^{-\alpha t \mathbf{A}}\mathbf{R}(0) \int_0^{t'} e^{-s' \mathbf{A}} ds' + e^{-\alpha t' \mathbf{A}}\mathbf{R}(0) \int_0^t e^{-s \mathbf{A}} ds \right) \\ &+ \alpha^2 \int_0^t \int_0^t ds ds' e^{-(s+s')\mathbf{A}} \end{aligned} \quad (\text{B23})$$

$$+ 4\sigma^2 \int_0^t ds \int_0^{t'} ds' e^{-s \alpha \mathbf{A}} e^{-s' \alpha \mathbf{A}^t} \delta(t-t'+s-s') \quad (\text{B24})$$

Now, we note that the first tree lines arise from $\langle \mathbf{R}(t) \rangle \langle \mathbf{R}(t') \rangle$. Thus

$$\langle \mathbf{R}(t)\mathbf{R}(t') \rangle - \langle \mathbf{R}(t) \rangle \langle \mathbf{R}(t') \rangle = 4\sigma^2 \int_0^t ds \int_0^{t'} ds' e^{-s \alpha \mathbf{A}} e^{-s' \alpha \mathbf{A}^t} \delta(t-t'+s-s') \quad (\text{B25})$$

In the case $t=t'$ we have

$$\Delta \mathbf{R}(t)^2 = 4\sigma^2 \int_0^t ds e^{-s \alpha \mathbf{A}} e^{-s \alpha \mathbf{A}^t} \quad (\text{B26})$$

If the matrix \mathbf{A} is symmetric, it reduces to

$$\Delta \mathbf{R}(t)^2 = 4\sigma^2 \int_0^t ds e^{-2s \alpha \mathbf{A}} \quad (\text{B27})$$

Thus, the fluctuation over the mean for uncorrelated white noise at large times is

$$\Delta \mathbf{R}(\infty)^2 = \frac{2\sigma^2}{\alpha} \langle \mathbf{R}(\infty) \rangle \quad (\text{B28})$$

2. Case of Moore-Penrose inverse

We now consider the perturbation of the dynamics leading to the Moore-Penrose equation, e.g.

$$\frac{d\mathbf{R}}{dt} = -\alpha \mathbf{A}^t \mathbf{A} \mathbf{R} + \alpha \mathbf{A}^t + \xi(t). \quad (\text{B29})$$

The solution is given by a simple generalization of eqn. (B9):

$$\mathbf{R}(t) = e^{-\alpha t \mathbf{A}^t \mathbf{A}} \mathbf{R}(0) + \int_0^t e^{-s \alpha \mathbf{A}^t \mathbf{A}} (\alpha \mathbf{A}^t + \xi(t-s)) ds \quad (\text{B30})$$

We are interested in the average

$$\langle \mathbf{R}(t) \mathbf{R}(t') \rangle = \langle (e^{-\alpha t \mathbf{A}^t \mathbf{A}} \mathbf{R}(0) + \int_0^t e^{-s \alpha \mathbf{A}^t \mathbf{A}} (\alpha \mathbf{A}^t + \xi(t-s)) ds) \cdot (e^{-\alpha t' \mathbf{A}^t \mathbf{A}} \mathbf{R}(0) + \int_0^{t'} e^{-s \alpha \mathbf{A}^t \mathbf{A}} (\alpha \mathbf{A}^t + \xi(t'-s)) ds) \rangle \quad (\text{B31})$$

$$\cdot (e^{-\alpha t' \mathbf{A}^t \mathbf{A}} \mathbf{R}(0) + \int_0^{t'} e^{-s \alpha \mathbf{A}^t \mathbf{A}} (\alpha \mathbf{A}^t + \xi(t'-s)) ds) \rangle \quad (\text{B32})$$

Note that we have

$$\langle \mathbf{R}(t) \rangle \langle \mathbf{R}(t') \rangle = (e^{-\alpha t \mathbf{A}^t \mathbf{A}} \mathbf{R}(0) + \alpha \int_0^t e^{-s \alpha \mathbf{A}^t \mathbf{A}} \mathbf{A}^t ds) \quad (\text{B33})$$

$$\cdot (e^{-\alpha t' \mathbf{A}^t \mathbf{A}} \mathbf{R}(0) + \alpha \int_0^{t'} e^{-s \alpha \mathbf{A}^t \mathbf{A}} \mathbf{A}^t ds) \quad (\text{B34})$$

The result here is a little bit simpler as $\mathbf{A}^t \mathbf{A}$ is symmetric, and thus

$$\langle \mathbf{R}(t) \mathbf{R}(t') \rangle - \langle \mathbf{R}(t) \rangle \langle \mathbf{R}(t') \rangle = 4\sigma^2 \int_0^t \int_0^{t'} ds ds' e^{-\alpha(s+s') \mathbf{A}^t \mathbf{A}} \delta(t-t'+s-s')$$

For $t = t'$ we have

$$\Delta \mathbf{R}(t)^2 = 4\sigma^2 \int_0^t e^{-2s \alpha \mathbf{A}^t \mathbf{A}} ds = \frac{2\sigma^2}{\alpha} \int_0^t e^{-s \mathbf{A}^t \mathbf{A}} ds \quad (\text{B35})$$

which is the formula reported in the main text. Thus, at long times the fluctuations are proportional to the Drazin inverse of $\mathbf{A}^t \mathbf{A}$. Since $\mathbf{A}^t \mathbf{A}$ is invertible, then this is just the inverse of $\mathbf{A}^t \mathbf{A}$.

Appendix C: Stability analysis

We now discuss the stability of the algorithm if volatile devices are present.

In the analysis of dynamical systems, the poles of the transfer function play a critical role in determining system stability. The transfer function, $H(s)$, obtained through the Laplace transform of the system's differential equations, is expressed as a ratio of polynomials in the complex frequency variable s . The poles are the roots of the denominator polynomial, representing the values of s for which $H(s)$ becomes unbounded. Mathematically, if $H(s) = \frac{N(s)}{D(s)}$, the poles are the solutions to $D(s) = 0$. The location of these poles in the complex plane directly influences the stability: if all poles lie in the left half of the complex plane (i.e., have negative real parts), the system is stable as perturbations decay over time. Conversely, poles with positive real parts indicate instability, causing perturbations to grow. Poles on the imaginary axis suggest marginal stability, where perturbations neither decay nor grow but persist indefinitely. Thus, the analysis of the poles provides crucial insights into the dynamic behavior and stability characteristics of the system.

1. Realistic function on the right-hand side

Instead of $x(t-\tau)$, the function that should be applied is of the form, for a certain τ

$$f_\tau(t-\tau) = \sum_{k=0}^{\infty} f((k-1)\tau) (\theta(t-(k-1)\tau) - \theta(t-k\tau)) \quad (\text{C1})$$

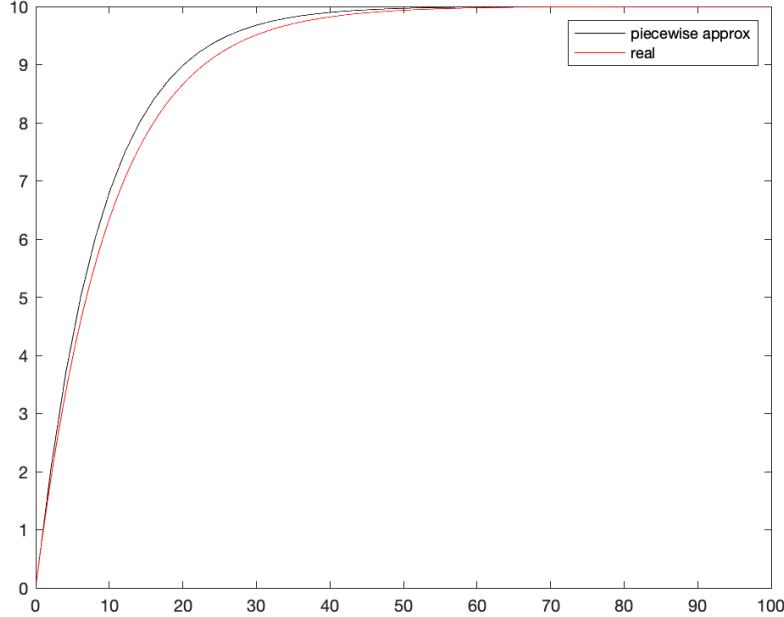


FIG. 8: Evolution for a delayed controller with a piecewise approximate versus instantaneous.

In the limit $\tau \rightarrow 0$, $\lim_{\tau \rightarrow 0} f_\tau(t - \tau) = f(t)$ pointwise. The Laplace transform is

$$\begin{aligned} f_\tau(s) &= \mathcal{L}(f_\tau) = \sum_{k=0}^{\infty} f((k-1)\tau) \frac{(e^{-(k-1)\tau s} - e^{-k\tau s})}{s} \\ &= \sum_{k=0}^{\infty} f((k-1)\tau) e^{-k\tau s} \frac{(e^{\tau s} - 1)}{s} \end{aligned} \quad (\text{C2})$$

If the delay τ is sufficiently small, we can identify $k\tau = t$ and $\tau = dt$, and it is easy to see that

$$f_\tau(s) = \mathcal{L}(f_\tau) = \int_0^\infty dt f(t - \tau) e^{-ts} + O(\tau) = e^{-\tau s} \mathcal{L}(f) + O(\tau). \quad (\text{C3})$$

We simulated eqn. (C4) with a small delay and a piecewise approximation for $x(t - \tau)$ as a controller would do. We see in Fig. 8 that the asymptotic point is the same.

2. Laplace transform for scalar linear delayed ODE

Let us look at the delayed scalar differential equation:

$$\frac{dx}{dt} = -\alpha a x(t - \tau) + b. \quad (\text{C4})$$

Using the Laplace transform we have

$$sx(s) - x_0 = -\alpha a e^{-\tau s} x(s) + \frac{b}{s} \quad (\text{C5})$$

from which we get the transfer function

$$x(s) = \frac{s x_0 + b}{s(s + \alpha a e^{-\tau s})} \quad (\text{C6})$$

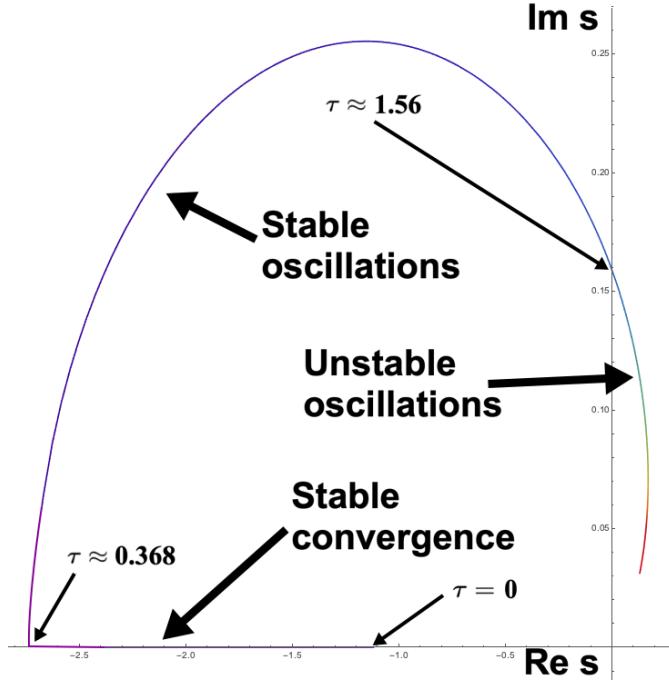


FIG. 9: Phase oscillations and instability for the simple equation of eq. (C4).

The two poles of this transfer function are in $s_1 = 0$ and the solution of

$$s + \alpha a e^{-\tau s} = 0 \quad (\text{C7})$$

whose solution is given by

$$s_2 = \frac{W(-a\alpha\tau)}{\tau} \quad (\text{C8})$$

where W is the product-log function. In the limit $\tau \rightarrow 0$, $s_2 \rightarrow -a\alpha$ which is real. However, as τ increases the function can develop into the complex plane. If a pole arises in the complex plane in the transfer function, then oscillations occur. Using this method, we can calculate the phase diagram in terms of the delay. Let us set $a\alpha = 1$, and measure τ in units of $a\alpha$. This can be seen in Fig. 9.

3. Poles of the matricial transfer function

Let us apply the Laplace transfer method to the matricial equation with a small delay as an approximate method. We have

$$s\mathbf{R}(s) - \mathbf{R}(0) = -\alpha\mathbf{A}\mathbf{R}(s)e^{-\tau s} + \frac{\alpha}{s}\mathbf{I} \quad (\text{C9})$$

and thus

$$\mathbf{R}(s) = (s\mathbf{I} + \alpha\mathbf{A}e^{-\tau s})^{-1}(\mathbf{R}(0) + \frac{\alpha}{s}\mathbf{I}) \quad (\text{C10})$$

$$= \frac{1}{s}(s\mathbf{I} + \alpha\mathbf{A}e^{-\tau s})^{-1}(s\mathbf{R}(0) + \alpha\mathbf{I}), \quad (\text{C11})$$

where the transfer function is now a matrix. We then need to look at the poles of the inverse matrix. These can be found via

$$\det(s\mathbf{I} + \alpha\mathbf{A}e^{-\tau s}) = 0. \quad (\text{C12})$$

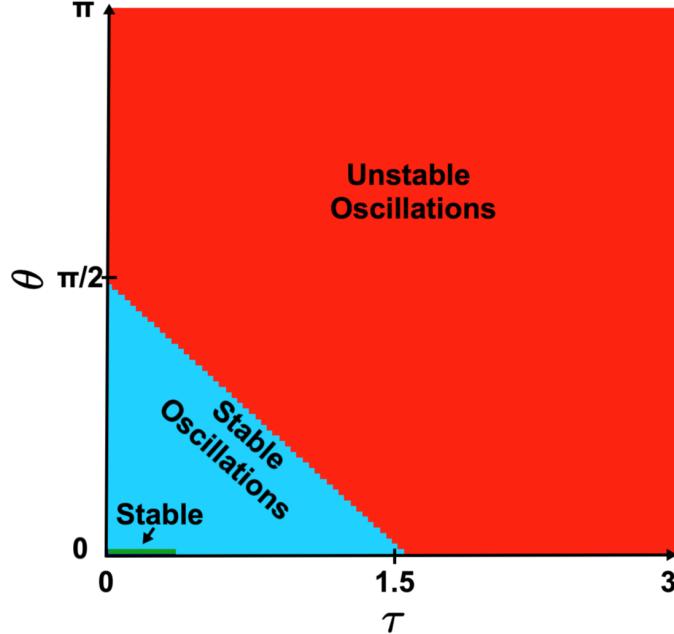


FIG. 10: Phase diagram of the stability of the delayed ODE of eqn. (10) with $\xi_P = \xi_O = 0$. Specifically, we analyze the angular dependence of the pole in the complex plane and identify the stability. The plot is symmetric on the $[0, 2\pi]$ interval. As we can see, for short delays the system is stable, while at slightly longer delays the system is marginally stable. At much longer delays there is an onset of instability.

This is the matricial equivalent of eqn. (C7). We recall that the eigenvalues of a matrix A are the solution of $\det(\lambda I - A) = 0$. Let us call $\lambda = -\frac{se^{\tau s}}{\alpha}$ for $s \neq 0$. We can find the poles by looking at the eigenvalues of the matrix A , and identifying

$$-\alpha\lambda_i = se^{\tau s}. \quad (\text{C13})$$

We thus see that the problem of the stability of the system is equivalent to the scalar problem, e.g.

$$s_i = \frac{W(-\alpha\lambda_i\tau)}{\tau}, \quad (\text{C14})$$

with the caveat that λ_i now can be complex. If the matrix A is normal and eigenvalues are real, then the analysis of the previous section is valid. If instead, the eigenvalues are complex the analysis becomes more complicated. Let us choose $|\lambda_i| = \rho_i$. Fixing $|\alpha\rho_i| = 1$, the graph of the poles as a function of τ and θ is shown in Fig. 10 where we wrote $\lambda_i = \rho_i e^{i\theta_i}$.

Appendix D: Crossbar implementation

1. Calibration

The computed forcing requires precise knowledge of the decay and steady-state parameters α and C . It is therefore desirable to be able to quickly measure these properties, preferably using a protocol that is simple and requires minimal computational complexity such that it may be repeated as necessary. The following protocol requires only two memristor reads, basic arithmetic, a simple constant forcing, and takes time on the order of $\Theta(2\alpha)$.

Suppose we allow an unforced memristor to relax fully to its steady-state resistance $R(\infty) = \frac{C}{\alpha}$. Let us call this value R_1 .

Now suppose we force the memristor with a voltage $V = R_1$. It is easy to see from (A1) that the steady-state value R_2 of the memristor under this forcing is

$$R_2 = \frac{C + \frac{C}{\alpha}}{\alpha} = \frac{C(1 + \alpha)}{\alpha^2} \quad (\text{D1})$$

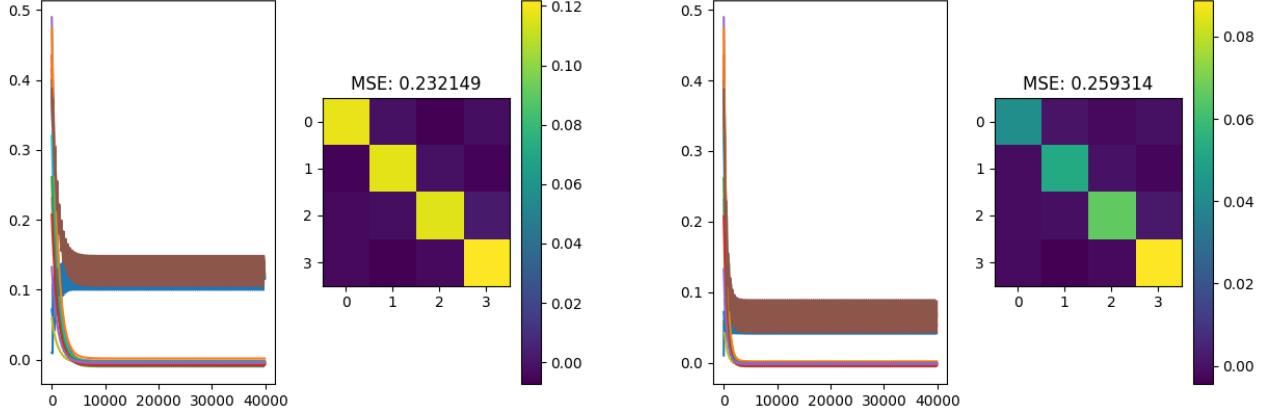


FIG. 11: Simulation of matrix inversion done naïvely on a crossbar containing isolated volatile crossbar elements (versus the nonvolatile crossbars considered in the main text). Because each subset of elements has zero bias while a different subset is being written, it decays freely and the matrix inverse fails to converge. **Left.** Inversion results when entire diagonals of elements are written simultaneously (i.e. $O(N)$). **Right.** Inversion results when each element is written sequentially in $O(N^2)$.

We now note that

$$\frac{R_1}{R_2} = \frac{C}{\alpha} \cdot \frac{\alpha^2}{C(1+\alpha)} \quad (\text{D2})$$

$$= \frac{\alpha}{1+\alpha} \quad (\text{D3})$$

and thus clearly $\alpha = \left(\frac{R_2}{R_1} - 1\right)^{-1} \rightarrow C = \alpha R_1$. We see that the speed of this scheme is primarily determined by the time it takes the memristors to relax to steady state.

On-chip, arithmetic operations can be implemented by ADCs and common integrated circuits. The scheme can also be performed elementwise for maximum accuracy.

2. Wake up signal for volatile devices

We now consider the implementation of this algorithm in a physical crossbar array, in which volatile memristors share input buses. It is clear that only a subset of memristors may be forced simultaneously to achieve high-rank updates to the state; moreover, a finite amount of time must be spent on each subset to write their memory parameters adequately. There is then a clear obstacle to convergence in the event that the system comprises memristors with high decay parameter α and relatively high subset writing time $\tau + \theta$, where τ represents the switching time of a transistor gating individual memristors and θ represents the width of a square wave pulse applied to the memristor. In this situation convergence is significantly hampered by the fact that the memory values of one subset will decay while another is being driven. Indeed, we observe this to be the case in simulation curves shown in Figure 11; the decay of inactive subsets significantly outpaces the contributions from the forcing in (A1).

It is thus necessary to further modify the forcing to account for the decays during inactivity; the general idea is that each subset of memristors must be driven higher than in (A1) to compensate for decays while the other subsets are being written. We refer to this modification as a wake up signal.

We begin by finding the state of an unforced memristor relaxing from state $R(0)$ at some time τ according to (A12),

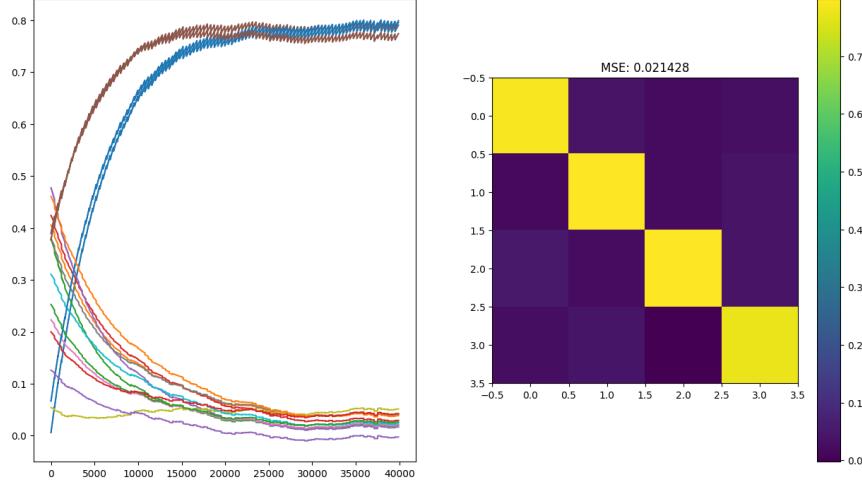


FIG. 12: When each volatile element is forced with a higher voltage to compensate for decays during period of zero bias, the crossbar state again converges to the desired inverse \mathbf{A}^{-1} .

setting $V = 0$ to find

$$R(\tau) = e^{-\alpha\tau} R(0) + C \int_0^\tau e^{-\alpha(\tau-t)} dt \quad (\text{D4})$$

$$= e^{-\alpha\tau} R(0) + C e^{-\alpha\tau} \left(\frac{1}{\alpha} e^{\alpha t} \Big|_0^\tau \right) \quad (\text{D5})$$

$$= e^{-\alpha\tau} R(0) + \frac{C e^{-\alpha\tau}}{\alpha} (e^{\alpha\tau} - 1) \quad (\text{D6})$$

$$= e^{-\alpha\tau} \left[\frac{\alpha R(0) + C(e^{\alpha\tau} - 1)}{\alpha} \right] \quad (\text{D7})$$

At the end of constant forcing V over the interval $[T_1, T_2]$, the final resistance is thus

$$R_{\text{force}}(T_2) = e^{-\alpha\tau} \left[\frac{\alpha R(T_1) + (C + V)(e^{\alpha\tau} - 1)}{\alpha} \right] \quad (\text{D8})$$

and it is easy to see that given an initial resistance $R(T_1)$, the constant voltage required to drive the memristor to a target value $R_{\text{force}}(T_2)$ is

$$V(R_{\text{force}}(T_2)) = \left(\frac{\alpha R_{\text{force}}(T_2)}{e^{-\alpha T}} - \alpha R(T_1) \right) \frac{1}{e^{\alpha T} - 1} - C \quad (\text{D9})$$

where $T = T_2 - T_1$. Finally we observe from (D4) that for (D8)

$$\lim_{\tau \rightarrow \infty} R_{\text{force}}(\tau) = (C + V) \int_0^\infty e^{-\alpha t} dt = \frac{C + V}{\alpha} \quad (\text{D10})$$

and we may progress to modifying the forcing.

We now consider a case in which the restricted writing subsets of a crossbar matrix R are its columns, such that a complete write of the crossbar takes time $N(\tau + \theta)$. Each column thus experiences a delay of duration $(N - 1)(\tau + \theta)$ before it gets written again.

Suppose that at a single write step, we have already quickly read the crossbar and computed the desired forcing, e.g. according to (A1). Let this forcing be $V^{(\text{ideal})} \in \mathbb{R}^{N \times N}$.

We now ask what resistances the array R would be driven to if $V^{(\text{ideal})}$ were applied infinitely. Using (68), we find that under this forcing $R_{ij}^{(\text{ideal})}(\infty) = \frac{C+V_{ij}}{\alpha}$.

We wish to find $R^{(\text{pre})}(0)$ such that $R((N-1)(\tau+\theta)) = R^{(\text{ideal})}(\infty)$ before the next write. We note that this value is just (D8) with a negative time input, i.e. evaluating $R(-(N-1)(\tau+\theta))$ with $R(0) = R^{(\text{ideal})}(\infty)$. We therefore drive each element R_{ij} of R to be

$$R_{ij}^{(\text{pre})} = e^{\alpha(N-1)(\tau+\theta)} \left[\frac{\alpha R_{ij}^{(\text{ideal})}(\infty) + C(e^{-\alpha(N-1)(\tau+\theta)} - 1)}{\alpha} \right] \quad (\text{D11})$$

$$= e^{\alpha(N-1)(\tau+\theta)} \left[\frac{\alpha \left(\frac{C+V_{ij}^{(\text{ideal})}}{\alpha} \right) + C(e^{-\alpha(N-1)(\tau+\theta)} - 1)}{\alpha} \right] \quad (\text{D12})$$

$$= e^{\alpha(N-1)(\tau+\theta)} \left[\frac{C + V_{ij}^{(\text{ideal})} + C(e^{-\alpha(N-1)(\tau+\theta)} - 1)}{\alpha} \right] \quad (\text{D13})$$

which yields our modified forcing

$$V_{ij}^{(\text{pre})}(R_{ij}^{(\text{pre})}) = \left(\frac{aR_{ij}^{(\text{pre})}}{e^{-\alpha T}} - \alpha R_{ij} \right) \frac{1}{e^{\alpha T} - 1} - C \quad (\text{D14})$$

which converges to the inverse even in the presence of subset decay (Figure ??).

3. Gating requirements for analog switches

The algorithm assumes that memristors are perfectly isolated from each other during reading and writing. In reality, the analog switches used to achieve 1T1R gating have a finitely high “off” resistance. This introduces an error with weak dependence on system size N , as in fact nonzero current flows through the entire system during all operations. While this current is extremely small, it becomes relevant in the context of highly nonlinear operations like matrix inversion. This problem is also commonly referred to as the “sneak path” problem in crossbars.

We can investigate the required value of R_{off} for the gating switches by varying between $1 \times 10^6 \Omega$ and $1 \times 10^9 \Omega$ and observing the final dependence on system size.

4. Atomic operations

We define an atomic operation as an update of the values output by the N voltage sources in the crossbar, and an update of the biases on the N^2 switches gating each element. A matrix-vector product Gv is thus $O(1)$ because it requires a single update of the N sources (to be the elements of v) and a single update of the N^2 transistors (biasing all gates). Likewise, read and write operations are $O(N)$ because they require $2N - 1$ distinct updates of the transistor states (and of the voltage sources in the latter case).

In general, a single “iteration” comprises a read of the matrix state R , a computation of the matrix product AR , and an update employing this product (e.g. $-\alpha AR + \alpha I$). All these operations are in $O(N)$ by the definition above, so a single iteration is in $O(N)$.

5. Mixed-Sign Computation

In general, A will have both positive and negative terms, while a given crossbar can only store terms of positive sign. It is therefore necessary to employ two crossbars for general computation, in which the matrix A is decomposed into positive and negative-only matrices such that $A = A^+ + A^-$, as suggested in [24]. Suppose we can only store positive-valued matrix entries. Then to compute a matrix-vector product we have $Av = (A^+ + A^-)v = A^+v - |A^-|v$, where the final expression follows because $A_{ij}^- \leq 0$. This practice is common for any form of single-sign storage medium.

What is more interesting is that this approach can be applied immediately to this iterative algorithm, provided that both R^+ and R^- are initialized to $\mathbf{0}$. When initialized in this way, the algorithm evolves such that R_{ij} either increase

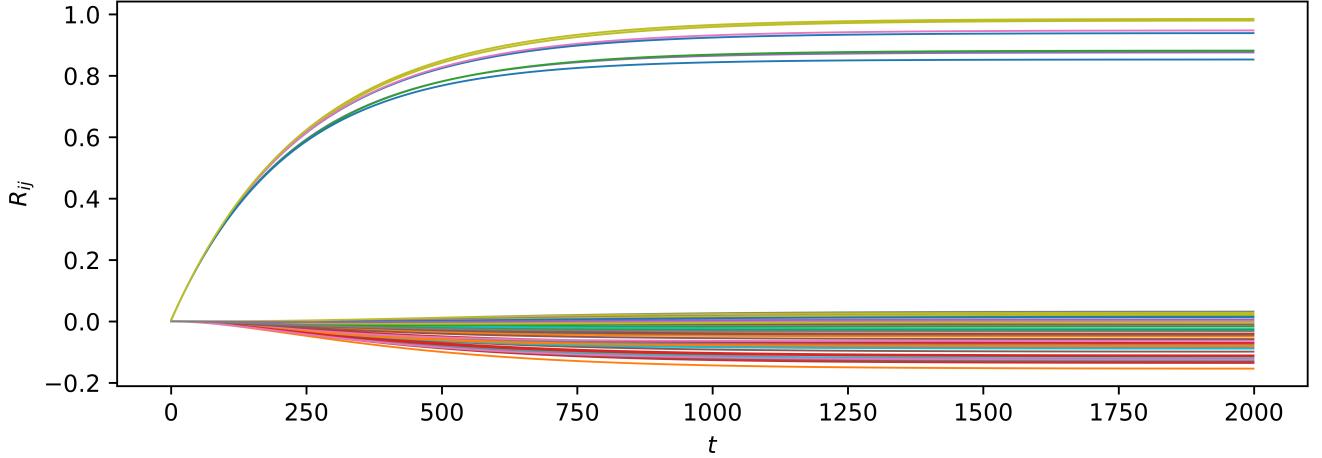


FIG. 13: When terms of A are of mixed sign, the terms R_{ij} converge monotonically to their respective destinations from 0.

or decrease monotonically (Figure 13), such that positive terms only experience positive gradients and negative terms likewise. Then if R^+ and R^- are crossbars containing memristors of opposite polarity, they will naturally separate to the correct values.

6. Parameter learning

We will use variational inference as a method to learn a Gaussian distribution. We assume that we are given a set of continuous data $\vec{x}_i \in \mathbb{R}^N$. This means that there is a method (experiments, a sampler, etc) from which, given a distribution $p(\vec{x})$, we obtain a set of points \vec{x}_i sampled from $p(\vec{x})$. Our assumption in this section in this is that $p(\vec{x})$ is Gaussian, e.g.

$$p_{\mathbf{A}}(\vec{x}) = \frac{1}{Z(A)} e^{-\frac{1}{2}\vec{x}^t \mathbf{A} \vec{x}} \quad (\text{D15})$$

$$Z(A) = \int d\vec{x}^N e^{-\frac{1}{2}\vec{x}^t \mathbf{A} \vec{x}}. \quad (\text{D16})$$

The question is how can we learn the parameters of this model, e.g. A_{ij} , which is called parameter estimation.

The method we will discuss here is variational inference. Let us look at the KL divergence, which will be our function to minimize to obtain the parameters:

$$\mathcal{L}(\mathbf{A}) = D_{KL}(p \| p_{\mathbf{A}}) = \int d\vec{x}^N p(\vec{x}) \log \frac{p(\vec{x})}{p_{\mathbf{A}}(\vec{x})}. \quad (\text{D17})$$

We know that $\mathcal{L}(\mathbf{A}) = 0$ if and only if $p(x) = p_{\mathbf{A}}(x)$. However, we already see a few problems here. First, we do not know $p(x)$. However, we can estimate the average from our samples. First, note that the term

$$p(x) \log p(x) \quad (\text{D18})$$

does not depend on \mathbf{A} , and thus from the point of view of minimization of \mathcal{L} it does not contribute, as it is just a constant. Thus, we are left with

$$\mathcal{L}(\mathbf{A}) = D_{KL}(p \| p_{\mathbf{A}}) = -\langle \log p_{\mathbf{A}}(x) \rangle_p + \text{constant} \quad (\text{D19})$$

where the constant is intended as constant for A . Second, we note that

$$\langle O(x) \rangle_{p(x)} \approx \frac{1}{M} \sum_{i=1}^M O(\vec{x}_i) \quad (\text{D20})$$

where M is the number of samples. From Chebyshev's inequality, we know that

$$P\left(\left|\frac{1}{N} \sum_i x_i - \mu\right| < \epsilon\right) \leq \frac{\sigma^2}{n\epsilon^2} \quad (\text{D21})$$

Thus, we can replace the analytical mean with an empirical one, given the assumption that the number of samples is large enough and the variance is not scaling with M .

We rewrite then

$$\mathcal{L}_{emp}(\mathbf{A}) = -\frac{1}{M} \sum_{i=1}^M \log e^{-\frac{1}{2} \vec{x}_i^t \mathbf{A} \vec{x}_i} + \log Z(\mathbf{A}) \quad (\text{D22})$$

$$= -\frac{1}{M} \sum_{ab} \sum_{i=1}^M \frac{1}{2} (x_i)_a A_{ab} (x_i)_b + \log Z(\mathbf{A}). \quad (\text{D23})$$

Since our distribution is Gaussian, we have

$$Z(\mathbf{A}) = \frac{(2\pi)^N}{\sqrt{\det(\mathbf{A})}} \quad (\text{D24})$$

$$\log Z(\mathbf{A}) = \log \sqrt{2\pi} - \frac{1}{2} \log \det(\mathbf{A}) \quad (\text{D25})$$

The question then arises of how to minimize this functional. We will use a variational method. We minimize step by step the empirical KL divergence by following the gradient, e.g.

$$\frac{\partial \mathcal{L}_{emp}(\mathbf{A})}{\partial A_{ab}} = -\frac{1}{M} \sum_{i=1}^M (x_i)_a (x_i)_b + \partial_{A_{ab}} \log Z(\mathbf{A}) \quad (\text{D26})$$

Let us write $1/M \sum_{i=1}^M (x_i)_a (x_i)_b = \langle x_a x_b \rangle_{emp}$. We note immediately that this is the empirical correlator. We now recall then that

$$\partial_{A_{ab}} \log Z(\mathbf{A}) = (A)_{ab}^{-1}. \quad (\text{D27})$$

As a result, it is possible to write the following gradient algorithm

$$\frac{d(A_t)_{ab}}{dt} = -\xi \frac{\partial \mathcal{L}_{emp}(\mathbf{A})}{\partial A_{ab}} = \xi (\langle x_a x_b \rangle_{emp} - (A_t^{-1})_{ab}) \quad (\text{D28})$$

where ξ is a parameter sometimes called *learning rate*. This method is called variational inference variational inference and we have seen here in action for the simplest example. Of course, this method works if $\langle x \rangle_{emp} = 0$.