Analog Sorting

Theory and evaluation

Lusa Zhan
Columbia University
Department of Computer Science
Department of Mathematics
lz2371@columbia.edu

Yipeng Huang
Columbia University
Department of Computer Science
yipeng@cs.columbia.edu

ABSTRACT

This work analyzes the theory behind analog sorting using the Toda lattice system and compares its performance against that of other popular sorting algorithms. It discusses the roles of the variables in the Toda lattice system and how they drive the sorting mechanism. Testing and simulations validate the functionality of analog sorting and allow for a preliminary estimate of its performance.

Keywords

linear algebra; ordinary differential equations; algorithms

1. INTRODUCTION

The focus of this paper is on an analog sorting algorithm that is deeply connected to the Toda Lattice system.

In contrast to well-known sorting algorithms such as merge or quick sort, the analog algorithm discussed here sorts values in a continuous way. In fact, it is the continuous time equivalent of the QR algorithm.

The theory behind analog sorting relies on the Toda lattice, which has been thoroughly discussed by the mathematical community. Adding to the theory, this paper aims to show results of testing analog sorting on a chip and simulating the system using an ODE solver. Ultimately, we evaluate the complexity of analog sorting in comparison to other sorting algorithms.

2. BACKGROUND

2.1 Classical sorting algorithms

While digital sorting algorithms are efficient, no prior work has discussed what the time complexity of analog sorting. To get a sense of how analog sorting performs, we need a basis for comparison. Some of the classical sorting algorithms are merge sort or quick sort. Those generally have nonlinear time complexity, but quick sort has logarithmic space complexity. Table 1 lists some algorithms and their respective complexities.

Later, we will analyze how analog sorting compares to digital sorting algorithms in terms of complexity.

Table 1: Complexities of Sorting Algorithms

Sorting Algorithm	Time Complexity	Space Complexity
Merge Sort	$O(n \log n)$	$O(n)$
Quick Sort	$O(n \log n)$	$O(\log n)$
Insertion Sort	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(1)$

2.2 The QR algorithm

The analog sorting method we show is related to the classical QR algorithm. While the QR algorithm is a discrete algorithm operating step-by-step, the analog sorting algorithm does so in continuous time [5, 3, 4].

The QR algorithm finds the eigenvalues and eigenvectors of a square matrix. The eigenvalue problem is as follows:

$$A_0 x = \lambda x$$

The QR algorithm operates as follows: Given the QR decomposition, each step proceeds as:

$$A_k = Q_k R_k$$

$$A_{k+1} = R_k Q_k$$

A side effect of the QR algorithm is that the eigenvalues of the original matrix end up in sorted order along the diagonal. This is a useful property in eigenvalue problems, where users of the algorithm are interested in finding the largest few eigenvalues. But few researchers have pointed out that this may be useful in itself, for sorting. Other algorithms for finding the eigenvalues and eigenvectors include the Jacobi eigenvalue algorithm, and the divide-and-conquer algorithm. It seems that the QR algorithm is the algorithm among these that sorts the eigenvalues.

The close relationship between the analog sorting method and the QR algorithm implies that they should have similar properties as well.

2.3 Hamiltonian systems

The finite Toda lattice system of ODEs that our analog sorter relies on belongs to special class of ODEs called Hamiltonian systems. Hamiltonian systems are an important and efficient way to describe classical mechanics. Because of their importance, special ODE solvers called sym-

DOI: 10.1145/1235

plectic solvers have been developed specifically to solve Hamiltonian systems.

A Hamiltonian system is characterized by a total energy scalar $\hat{\mathcal{H}}$. The components of the system are described by vectors p for momenta and q for positions. The system obeys the laws of motion:

$$\frac{dp}{dt} = -\frac{dH}{dq} = f(q) \qquad \frac{dq}{dt} = \frac{dH}{dp} = g(p) \tag{1}$$

Conceptually, we encode the keys we like to sort as the momenta of particles. The momenta of the particles carries them to a final state that represents the sorted system. We can then use symplectic ODE solvers to implement our analog sorter.

2.4 Toda lattice - Double bracket

The finite Toda lattice is an ODE system that is related to the QR algorithm. If you plot the evolution of the Toda lattice ODE with respect to time, the values of the ODE at integer time steps is the intermediate states of the QR algorithm. The common property of the finite Toda lattice and the QR algorithm is that both preserve the eigenvalues of the matrix they operate on [1].

The finite Toda lattice is a Hamiltonian system with the form:

$$\hat{\mathcal{H}}(p,q) = \frac{1}{2}\sum_{1}^{n} p_k^2 + \sum_{1}^{n-1} exp(q_k - q_{k+1}) \tag{2}$$

This basic form can changed in two ways, using change of variables, into the equations for analog sorting described in [2].

The central idea of analog sorting as described in [1, 2] makes use of the Toda lattice. The system of Hamiltonian equations in (1) associated with the Hamiltonian in (2) gives us the following set of equations

$$\begin{aligned} \dot{p_k} &= \exp(q_{k-1} - q_k) - \exp(q_k - q_{k+1}) \\ \dot{q_k} &= p_k \end{aligned}$$

where $\exp(x_0 - x_1) = \exp(x_n - x_{n+1}) = 0$.

This system is also analogous to the double bracket notation

$$\dot{H} = [H, [H, N]] \tag{3}$$

where the brackets stand for the Lie bracket $[A, B] = AB - BA$. The connection between the Hamiltonian system and the double bracket notation of the Toda lattice will be discussed and outlined in the following sections.

2.4.1 Toda lattice - ODE form

The above system of ordinary differential equations can be transformed into the system described in [8] using a change of variables.

$$\begin{aligned} x_k &= -\frac{1}{2}p_k \\ y_k &= \frac{1}{2}\exp(\frac{q_k - q_{k+1}}{2}) \end{aligned} \tag{4}$$

In that case,

$$\begin{aligned} \dot{x_k} &= -\frac{1}{2}\dot{p_k} \\ &= -\frac{1}{2}(exp(q_{k-1} - q_k) - exp(q_k - q_{k+1})) \\ &= -\frac{1}{2}(4y_{k-1}^2 + 4y_k^2) \\ &= 2y_k^2 - 2y_{k-1}^2 \end{aligned}$$

and

$$\begin{aligned} \dot{y_k} &= \frac{1}{2}\exp(\frac{q_k - q_{k+1}}{2})\frac{\dot{q}_k - \dot{q}_{k+1}}{2} \\ &= y_k\frac{p_k - p_{k+1}}{2} \\ &= y_k(x_{k+1} - x_k) \end{aligned}$$

Taking into account the boundary conditions $y_0 = y_n = 0$, we get the desired system of ODEs with

$$\begin{aligned} \dot{x_k} &= 2y_k^2 - 2y_{k-1}^2 \\ \dot{y_k} &= y_k(x_{k+1} - x_k) \\ y_0 &= y_n = 0 \end{aligned} \tag{5}$$

This is the system of ODEs we will solve in the analog chip and in our simulations.

2.4.2 Toda lattice - Jacobi matrix

The connection between the Toda lattice and the double bracket notation $\dot{H} = [H, [H, N]]$ can be made through the Jacobi Matrix form of the Toda lattice. The Jacobi matrix for the Hamiltonian system after the change of variables (4) is given by

$$H = \begin{bmatrix} x_1 & y_1 & 0 & \dots & 0 \\ y_1 & x_2 & y_2 & \dots & 0 \\ & & \ddots & & \\ & & y_{n-2} & x_{n-1} & y_{n-1} \\ 0 & \dots & & y_{n-1} & x_n \end{bmatrix} \tag{6}$$

This is the form of H required for analog sorting as outlined by Brockett in [2].

In order to get the double bracket form, we need a diagonal matrix $N = \text{diag}(n, n-1, \dots, 1)$ whose role will be discussed in 2.5.1.

$$N = \begin{bmatrix} n & 0 & \dots & 0 \\ 0 & n-1 & & \\ \vdots & & \ddots & \vdots \\ 0 & \dots & & 1 \end{bmatrix}$$

From this, we can calculate $[H[H, N]] = H[H, N] - [H, N]H$ step by step:

$$HN = \begin{bmatrix} nx_1 & (n-1)y_1 & \dots & 0 \\ ny_1 & (n-1)x_2 & \dots & 0 \\ \vdots & & \ddots & 0 \\ & & 2x_{n-1} & y_{n-1} \\ 0 & \dots & 2y_{n-1} & x_n \end{bmatrix}$$

$$NH = \begin{bmatrix} nx_1 & ny_1 & \dots & 0 \\ (n-1)y_1 & (n-1)x_2 & \dots & 0 \\ \vdots & & \ddots & 0 \\ & & 2x_{n-1} & 2y_{n-1} \\ 0 & \dots & y_{n-1} & x_n \end{bmatrix}$$

$$HN - NH = \begin{bmatrix} 0 & -y1 & \dots & & 0 \\ y_1 & 0 & -y_2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ & & y_{n-2} & 0 & -y_{n-1} \\ 0 & & \dots & y_{n-1} & 0 \end{bmatrix}$$

$$H[H,N] = \begin{bmatrix} y_1^2 & -x_1y_1 & \dots & 0 \\ x_2y_1 & -y_1^2 + y_2^2 & \dots & 0 \\ y_1y_2 & \ddots & & 0 \\ \vdots & & y_{n-1}^2 - y_{n-2}^2 & -x_{n-1}y_{n-1} \\ 0 & \dots & x_ny_{n-1} & -y_{n-1}^2 \end{bmatrix}$$

$$[H,N]H = \begin{bmatrix} y_1^2 & -x_2y_1 & \dots & 0 \\ x_1y_1 & y_1^2 - y_2^2 & \dots & 0 \\ y_1y_2 & \ddots & & 0 \\ \vdots & & -y_{n-1}^2 + y_{n-2}^2 & -x_ny_{n-1} \\ 0 & \dots & x_{n-1}y_{n-1} & y_{n-1}^2 \end{bmatrix}$$

Therefore, we get

$$[H,[H,N]] = \begin{bmatrix} 2y_1^2 & y_1(x_2 - x_1) & 0 \\ y_1(x_2 - x_1) & -2(y_1^2 - y_2^2) & 0 \\ 0 & \ddots & 0 \\ \vdots & & y_{n-1}(x_n - x_{n-1}) \\ 0 & \dots & -2y_{n-1}^2 \end{bmatrix}$$

This is equivalent to the result we get from combining the matrix form of H with the values for $\dot{x}$ and $\dot{y}$

$$\dot{H} = \begin{bmatrix} \dot{x}_1 & \dot{y}_1 & 0 & \dots & 0 \\ \dot{y}_1 & \dot{x}_2 & \dot{y}_2 & \dots & 0 \\ & & \ddots & & \\ & & \dot{y}_{n-2} & \dot{x}_{n-1} & \dot{y}_{n-1} \\ 0 & \dots & & \dot{y}_{n-1} & \dot{x}_n \end{bmatrix}$$

In total, we conclude that the double bracket notation of the Toda lattice used in other papers is analogous to the system of ODEs. We use the equivalent system of ODEs to construct the analog sorter.

2.4.3 Role of N

The choice of $N = \text{diag}(n, n-1, \dots, 1)$ for analog sorting is clarified by Theorem 1.5 in [6]. To summarize, the theorem states the following:

For each N, $\dot{H}(t) = [H,[H,N]]$ converges to an equilibrium H_∞ as $t \to \infty$. If $N = \text{diag}(\mu_1, \dots, \mu_n)$ where $\mu_1 > \dots > \mu_n$, then the Hessian of $f_N(H) = \frac{1}{2}\|N - H\|^2$ is nonsingular and negative definite.

The linearization of the double bracket flow at an equilibrium point $H_\infty = \text{diag}(\lambda_{\pi(1)}, \dots, \lambda_{\pi(n)})$ is

$$\dot{\xi}_{ij} = -(\lambda_{\pi(i)} - \lambda_{\pi(j)})(\mu_i - \mu_j)\xi_{ij}$$

where $\xi = [H_\infty, N]$. Since the Hessian is negative definite, ξ must be at its maximum. If $N = \text{diag}(n, n-1, \dots, 1)$, then ξ will only reach its maximum if the diagonal entries of H_∞ are sorted in descending order as well.

Thus the diagonal matrix N determines the order of the diagonal entries in H_∞. Letting $N = \text{diag}(n, , n-1, \dots, 1)$, we will result in $x_1 \geq x_2 \geq \dots \geq x_n$ in H_∞.

2.4.4 Role of off-diagonals y_k

The choice of initial values for the off-diagonal y_k values affects the results of analog sorting as well.

First of all, as mentioned previously, $y_0 = y_n = 0$. To analyze the initial values of y_k, consider the relation to $\dot{x}_k$

$$\dot{x_k} = 2y_k^2 - 2y_{k-1}^2$$
$$\dot{y_k} = y_k(x_{k+1} - x_k)$$

If $y_k = 0$, then $\dot{x}_k = 0$, meaning $x_k(t) = x_k(0)$ for all t. This means that the values along the diagonal of H will remain constant and never get sorted.

In addition, the y_k values should be small (but still positive). Recall that the eigenvalues will be the values along the diagonal of H as $t \to \infty$. Consider the determinant $f_n = \det(H - \lambda I)$, where n is the size of the matrix. Since H is a tridiagonal matrix, this determinant can be formulated using a recurrence relation on the size of the matrix n.

$$f_n = (x_n - \lambda)f_{n-1} - y_{n-1}^2 f_{n-2}$$
$$f_1 = x_1 - \lambda$$

We can use this to calculate the first few f_n and set them to 0 (what we would do to calculate the eigenvalues):

$$f_1 = x_1 - \lambda = 0$$
$$f_2 = (x_2 - \lambda)(x_1 - \lambda) - y_1^2 = 0$$
$$f_3 = (x_3 - \lambda)(x_2 - \lambda)(x_1 - \lambda) - (x_3 - \lambda)y_1^2 - (x_1 - \lambda)y_2^2 = 0$$

Intuitively, the eigenvalues will be closest to x_k if the y_k values are smaller. When setting up the analog sorter, we want the off-diagonals to be nonzero, but small enough to not significantly change the values to be sorted.

3. METHODOLOGY

3.1 Realizing the analog sorter

We validate the functionality of analog sorting using a prototype analog computer chip. After reviewing the mathematics of analog sorting, we now discuss how the analog sorter is put in practice.

The analog sorter works as follows: We construct matrix $H(t = 0)$ with the real numbers we want to sort on the diagonal and some small values on the off-diagonals. Naturally $H(t = 0)$ has eigenvalues consisting of the same real numbers. We set up a special ODE that involves the vector x, and another vector consisting of the natural numbers. This vector of the natural numbers provides the "discreteness"

for the algorithm. Specifically, this ODE is the finite Toda lattice ODE, which preserves the eigenvalues of $H(t)$, while reordering the elements on the diagonal to the sorted sequence. This solves this ODE on an analog computer. The final steady state of the analog output would have the original elements of the vector x, but now in sorted order. For example, the first integrator would have the lowest magnitude element of x.

There are some variations to this idea. One is we can tackle the finite Toda lattice system directly as a Hamiltonian system. Another is we can reverse the roles of the sort keys and the indices. This is described by Brockett briefly in page 802 of [2].

3.2 Running the simulation

In order to explore how analog sorting works in larger systems, we use an ODE solver built on the odeint solver in the Python SciPy library [7]. We first construct an array of variables, an array of the ODEs from (5), and an array of initial values, which we supply to SciPy's odeint. The initial values for x are randomly chosen from a dynamic range using randint from Python's random module. The initial y values are set at 0.9. SciPy's odeint returns the values of x and y at each specified time interval of 0.04 seconds.

The data is used to plot graphs, but also to determine when the x values have settled in at their final value.

4. EVALUATION

After testing analog sorting on a prototype analog computer chip and simulating the process using an ODE solver, we can evaluate its time and space complexity.

Compared to the other sorting algorithms mentioned at the beginning of the paper, analog sorting is competitive in terms of time complexity. However, due to the nature of circuits, it is not as space efficient as most other popular sorting algorithms.

4.1 Functionality validation of analog sorting

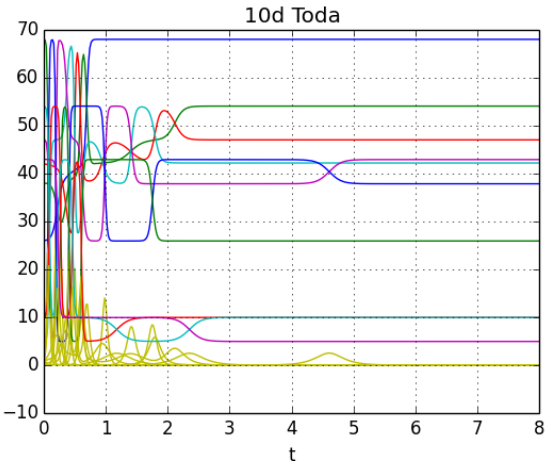

Figure 1: Simulated sorting a 10 element vector.

We see in Figure 1, generated from the ODE solver, that the sorting system has periods of swapping, interspersed in quiet periods of little change. The ODE solver was used on a 10 element vector.

The output of the analog chip, run on two variables and over multiple loops, is shown in Figure 2.

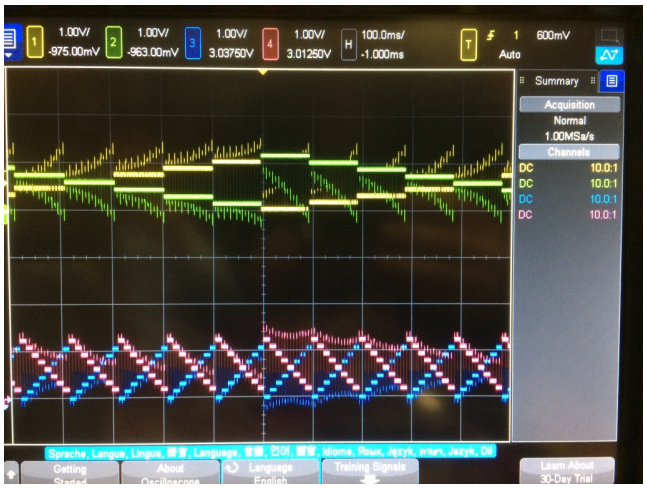

Figure 2: Sorted 2 elements using analog computer chip.

This confirms that the values on the diagonals will converge towards a final value.

4.2 Time cost of the discrete QR algorithm

Since the analog sorting method is the continuous version of the QR algorithm, there might be a correlation between their time complexities as well.

In essence, our analog sorter takes a tridiagonal matrix H and sorts the eigenvalues along the diagonal in descending order. Deift explains that the QR algorithm takes $O(N)$ time for a tridiagonal $N \times N$ matrix.

The assumption is that the time complexity of analog sorting will also be in that range.

4.3 Time cost of analog sorting

In terms of time, the sorter takes at least $O(N)$ time because of the time it takes just for signals to propagate across the circuit. Another issue is the time it takes for the ODE to settle to its final value. This shows preliminary data showing the time to convergence of analog sorting, plotted against problem size.

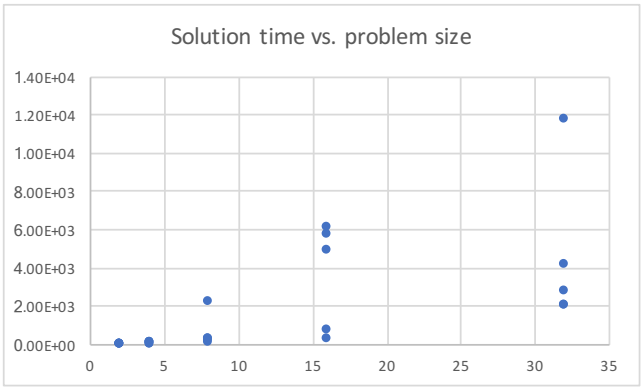

Figure 3: Preliminary time to convergence vs. problem size.

The axes are the time to solution, plotted against N, the number of real numbers we are sorting. The data points come from multiple random trials at each N. The set of real numbers is randomly generated from a chosen dynamic range. It appears that as the N size increases, the average time grows linearly with respect to N. Furthermore, the variance of the solution time, measured as the standard deviation, also grows.

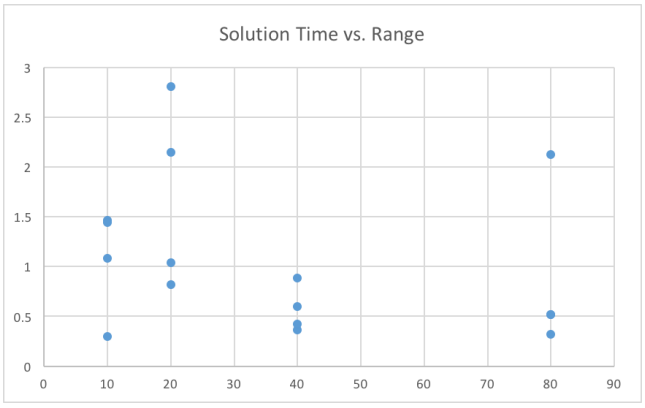

Figure 4: Preliminary time to convergence vs. dynamic range.

Another possible factor for solution time to consider is the dynamic range of the real numbers to sort. However, running the simulation on a constant number of elements with different dynamic ranges did not show any significant trends.

4.4 Hardware cost of analog sorting

Analog sorting on the chip relies on the hardware components needed to build the circuit. We need a constant amount of components for each of the N ODEs. Thus, the analog sorter takes up $O(N)$ amount of circuit components to sort N elements.

4.5 Limitations

As mentioned before, the choice of initial y values affects the sorting mechanism. Changing the initial values of the off-diagonals impacts whether or not the algorithm ever reaches a correctly sorted state of the diagonal entries.

Setting the initial values for the off-diagonals to 0 will prevent the algorithm from starting in the first place. Initial y values that are too small may stop the algorithm mid-way and result in an approximate sorting with two elements being out of place. After the diagonal entries converge to their final state, the y values are nonzero, but in the magnitude of less than 10^{30}. How big the initial off-diagonal entries have to be might depend on the ratio of the numbers to sort.

Brockett mentions briefly that H_∞ will be sorted in almost all cases, but does not elaborate on the exceptions [2].

5. APPLICATIONS & CONCLUSIONS

In this work, we validate that analog sorting works in practice. While this idea has been presented several times in mathematical literature, this is the first where it is tested in practice, using analog computer hardware. Using real measurements and simulated experiments, we evaluate the performance and efficiency of analog sorting. We compare this technique against classical sorting algorithms. In addition, we find limitations to analog sorting through experiments. Cases in which analog sorting could fail were not discussed in previous literature.

In terms of concrete impact of our findings, analog sorting can be a new way to sort in future computer architectures. Sorting is the underpinning of big data. Increasingly, approximate sorts are useful.

In terms of the impact of this work on the future of analog computer techniques, we show in this paper two key ideas. One, using the soliton behavior of the finite Toda lattice, we are able to swap two values in an analog computer without having to use a third capacitor as a buffer. Two, it is possible to perform a typically "discrete" task using continuous-time hardware. From this discrete primitive operation, we can build other discrete algorithms.

6. REFERENCES

[1] A. M. Bloch and A. G. Rojo. *Sorting: The Gauss Thermostat, the Toda Lattice and Double Bracket Equations*, pages 35–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[2] R. W. Brockett. Dynamical systems that sort lists, diagonalize matrices and solve linear programming problems. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 799–803 vol.1, Dec 1988.

[3] M. T. Chu. On the continuous realization of iterative processes. *SIAM Review*, 30(3):375–387, 1988.

[4] M. T. Chu. A list of matrix flows with applications. In *in Hamiltonian and Gradients Flows, Algorithms and Control*, pages 87–97, 1994.

[5] P. Deift, T. Nanda, and C. Tomei. Ordinary differential equations and the symmetric eigenvalue problem. *SIAM Journal on Numerical Analysis*, 20(1):1–22, 1983.

[6] U. Helmke and J. B. Moore. *Double Bracket Isospectral Flows*, pages 43–80. Springer London, London, UK, 1994.

[7] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2016-12-20].

[8] H. R. Laboratory. Analog computation, 2003–2004. [Online URL hrl.harvard.edu/analog/; accessed 2016-12-20].