We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Chapter 2

The Eight Epochs of Math as Regards Past and Future Matrix Computations

Frank Uhlig

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.73329

Abstract

This survey paper gives a personal assessment of epoch-making advances in matrix computations, from antiquity and with an eye toward tomorrow. It traces the development of number systems and elementary algebra and the uses of Gaussian elimination methods from around 2000 BC on to current real-time neural network computations to solve time-varying matrix equations. The paper includes relevant advances from China from the third century AD on and from India and Persia in the ninth and later centuries. Then it discusses the conceptual genesis of vectors and matrices in Central Europe and in Japan in the fourteenth through seventeenth centuries AD, followed by the 150 year cul-de-sac of polynomial root finder research for matrix eigenvalues, as well as the superbly useful matrix iterative methods and Francis' matrix eigenvalue algorithm from the last century. Finally, we explain the recent use of initial value problem solvers and high-order 1-step ahead discretization formulas to master time-varying linear and nonlinear matrix equations via Zhang neural networks. This paper ends with a short outlook upon new hardware schemes with multilevel processors that go beyond the 0–1 base 2 framework which all of our past and current electronic computers have been using.

Math subject classifications: 01A15, 01A67, 65-03, 65F99, 65Q10

Keywords: math history, math computations, matrix, matrix computations, Zhang neural network, time-varying model, time-varying computations, 1-step ahead discretization formulas, time-varying equations, eigenvalues, computer hardware, numerical analysis

1. Introduction

In this paper we try to outline the epoch making achievements and transformations that have occurred over time for computations and more specifically for matrix computations. We will

IntechOpen

© 2018 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. CC BY

trace how our linear algebraic concepts and matrix computations have progressed from the beginning of recorded time until today and how they will likely progress into the future. We take this limited tack simply because in modern times, matrices have become the elemental and universal tools for most any computation.

This evolution of our matrix methods will be described in broad strokes. My main emphasis is to trace the mathematical genesis of matrices and their uses and to learn how the modern matrix concept has evolved in the past and how it is evolving. I am not interested in matrix theory by itself, but rather in matrix computations, i.e., how matrix concepts and algorithms have been developed from approximately 3000 BC to today, and even tomorrow.

This paper describes eight noticeably separate epochs that are distinguished from each other by the introduction of evolutionary new concepts and subsequent radically new computational methods. Following the historical trail through six historically established epochs, we will then look into the present and the near future.

What drives us to conceptualize and compute differently now, and what is leading us into the seventh and possibly eighth epoch? When and how will we likely compute in the future?

I am not a math historian, I have never taught a class in math history. Instead, throughout my academic career, I have worked with matrices: in matrix theory, in applications, and in numerical analysis. I like to construct efficient new algorithms that solve matrix equations. The idea for this paper is in part due to my listening by chance to a very short English broadcast from Egyptian radio on short wave some 40 years ago in the 1970s. It described an Egyptian papyrus from around 2000 BC that dealt with solving linear equations by row reduction and zeroing out coefficients in systems of linear equations, i.e., by what we now call "Gaussian elimination." When I heard this as a young PhD, I was fascinated and wrote the station for more information. They never answered, and when I was in Cairo many years later, the Egyptian Museum personnel could not help me either with locating the source.

Thus, I became aware that Carl Friedrich Gauß did not invent what we now call by his name, but who did?

For many decades, this snippet of math history just lingered in my mind until a year ago when I was sent a book on Zhang neural network (ZNN) methods for solving time-varying linear and nonlinear equations and was asked to review it. The ZNN methods were—to me and my understandings of numerics then—so other-worldly and brilliant that I began to think of the incredible leaps and "bounces" that math computations have gone through over the eons, from era to era. I eventually began to detect seven or eight computational sea changes, what I call "epochs," in our ability to compute with matrices, and that is my topic.

2. A short history of matrix computations

Nobody knows how numbers and number systems came about, just like nobody knows "who invented the wheel." I will start with a few historical facts about number systems and how they developed and were used across the globe in antiquity.

http://dx.doi.org/10.5772/intechopen.73329

2.1. Early number systems

Humankind's first developments of number systems were very diverse and geographically widely dispersed, yet rather slow. The first circle cipher for zero occurred in Babylonia around 2500 BC or 4500 years ago. A continent or two removed, the Mayans used the same concept and circle zero symbol from around 40 BC. In India, it was recognized during the seventh century. But zero only became recognized as a "number to compute with" like all the others in the 9th century in Central India. Our decimal system builds on the ten numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The decimal positional system came from China via the Indus valley, and it started to be used in Persia in the ninth century. It was combined with or derived from a Hindu number system of the same time period.

In fact Westerners call the current decimal number symbols wrongfully "Arabic," but most Westerners (and I) cannot read the license plates on cars in Egypt since the Arabic world does not use our Persian/Hindu numbers in writing but its own script using Arabic letters to designate numbers. Should we call our "western" numbers "Farsi" or "Hindu" instead?

Various bases have been used for numbering. There have been base 2, base 8, base 10, base 12, base 16, base 60, and base 200 number systems and possibly more at some time somewhere throughout human history. Counting and simple computations started with notched sticks for record keeping and with the invention of sand or wax tablets and then the abacus. These simple tools were developed a little bit differently and independently in many parts of the globe.

2.2. Antiquity: first epoch

Around 2200 to 1600 BC, Sumerian, Babylonian, and Egyptian land survey computations became mathematized in order to mark and allot land after the yearly Euphrates, Tigris, and Nile floods. That lead to linear equations in 2, 3, or 4 variables and subsequent methods to solve them that amounted to what we now call row reduction or Gaussian elimination.

Mathematical computations did not advance much during the Greek times as Greek mathematicians were mainly interested in mathematical theory and in establishing the concept of a formal proof, as well as elementary number theory of which the Euclidean algorithm is still used today.

Neither did the complicated Roman numerals lend themselves to easy computations, and no further computational advances happened there.

2.3. Early mathematical arts in China, India, and the Near East: second epoch

(Based in part on a lecture at Hong Kong University in 2017, given by Xiangyu Zhou, Chinese Academy of Sciences, for Chinese sources, and on Indian and Arabic sources from elsewhere).

In prehistoric and historic times (1600 BC–1400 AD), knot and rod calculus were prevalent in China. They were based on a decimal positional system, so-called rod numerals. These comprised the most advanced number system of the time, and it was used for several millennia

before being adopted and expanded in Persia and India in the ninth century AD and later on adopted in Central Europe.

The *Mathematical Classic of Sunzi* by Suanjing (from the third to the fifth century) gives a detailed description of the arithmetic rules for counting rods. In the Indus valley, clay tablets covered with sand were used for mathematical computations several millennia ago. Bhaskara (600–680 AD) in India was the first one to write numbers in the Hindu positional decimal system which used the circle for zero. In 629 AD he approximated the sine function by rational expressions while commenting on Aryabhatta's (476–550 AD) book *Aryabhatiyabhisya* from 499 AD. An Indian contemporary of Bhaskara, Brahmagupta (598–665 AD), was the first one to establish the rules that govern computing with zero. Brahmagupta texts were written in Sanskrit verse that used the Sanskrit word for "eyes" to denote 2, "senses" for the number 5, etc. This was common in Indian mathematics and science writings at the time. The earliest record of multiplication and division algorithms using the Hindu numerals 1 through 9 and 0 was in writings by Al Khwarizmi 780–850 AD, a Persian mathematician employed in Bagdad. His *The Book of Manipulation and Restoration* established the golden rule of Algebra that an equation remains true if one subtracts the same quantity from both sides. He also wrote down multiplication and division rules that are identical to those of Suanjing from the third to fifth century in China. To Suanjing we also owe the Chinese remainder theorem. Finally, the advanced Hindu-Arabic decimal number system was introduced into the west by Leonardo Fibonacci (1175–1250 AD) of Pisa in his *Liber Abaci* or *The Book of Calculations* (1202),

Applied and numerical computations were driving much of Chinese mathematics. Wang Xiaotang (580–640 AD), for example, tried to find the roots of cubic polynomials that appeared in civil engineering and water conservation problems. In the *Mathematical Treatise in Nine Sections* of 1247, Qin Jiushao (1202–1261) developed the "Qin Jiushao method" which is now commonly called the "Horner-Ruffini scheme" for computing with and finding roots of polynomials iteratively. William George Horner [1] and Paolo Ruffini (1804–1807–1813) reinvented the Qin Jiushao method unknowingly 600 years later.

Chinese rod calculus was the method of choice for computing in China until the abacus took over during the Ming dynasty (1388–1644). Cheng Dawei (1355–1606) is the author of the first "numerical analysis" book titled *The General Source of Computational Methods* published in 1592. It describes methods to add, subtract, multiply, and divide on an abacus. The abacus itself was invented in various incarnations at various times and in several locations of the globe. It essentially combines several decimal rods on one board with beads on strings.

Chinese mathematicians from the third century BC onward to the tenth century AD brought us the *The Nine Chapters on the Mathematical Arts* that uses the numbers 1 through 9. This book was later disseminated further to the west and to India and Persia as described above. In Chapter 7, determinants first appeared conceptually, while Chapter 8 abstracts the concept of linear equations to represent them by matrix-like coefficient tableaux. These "matrix equations" were solved in China, again by "Gaussian elimination," 1500 years before Gauß' birth and 1800 years after the middle-eastern seasonal flood prone countries had first used the Gaussian algorithm around 1800 BC. Gauß himself described the method as the "common

method of elimination" in his papers, and mathematicians then attached his name to it as an honor.

2.4. The genesis of vectors and matrices: third epoch

To advance matrix computations further, there was a need to conceptualize coordinates and vectors in space.

In the fourteenth century AD, Nicole Oresme developed a system of orthogonal coordinates for describing Euclidean space. This idea was taken up by René Descartes in the seventeenth century and is familiar to all of us now under the concept of Cartesian coordinates. Thereby, the world became ready for matrices and matrix computations in their own right.

In 1683 Gottfried Leibnitz in Germany and Seki Kowa in Japan both unbeknownst to each other reinvented the concept of a matrix as a rectangular array of coefficients for studying linear equations. Leibnitz also used and suggested row elimination to simplify and find their solutions. These efforts enabled Gauß to repeat what the Egyptians had done four millennia earlier: he was asked to survey the lands of his ruler, the Archduke George Augustus of Hanover, and measure the size of this kingdom inside Germany in the early 1800s. Beginning in the 1820s, Gauß, as Professor of Geodesy (and not of Mathematics) in Göttingen, would measure the angles and distances between many of the highest points there, such as the Brocken; the Inselsberg, 104 km apart; and the hills around Göttingen, and later he expanded the surveys all the way to the North Sea coast. He and his assistants did this multiple times, preferably when the weather was clear. Thereby, they set up systems of linear equations with generally more equations than unknown due to repeated measurements on different days.

To solve these overdetermined and naturally "unsolvable" systems $Ax = b$, Gauß devised the normal equation $A^T Ax = A^T b$ (1823) [2] and solved them approximately. But the normal equations method eventually turned out to be numerically unsound. It took over a century to find out why, the reason being that condition numbers multiply (see Olga Taussky [3]).

2.5. Eigenvalues and the characteristic polynomial: fourth epoch

As differential operators and matrices were beginning to be investigated and dealt with by the early 1800s, their connections and similarities were slowly recognized in the mathematics world.

The replication of certain functions $f \neq 0$ by a given differential operator $\mathscr{A}$ was noticed first and became the subject of studies. What were the functions f for which $\mathscr{A}f = \alpha f$ for some scalar α? How could they be found from $\mathscr{A}$, and what about α?

In 1829, Augustin Cauchy [4, p. 175] began to view the erstwhile "eigenvalue equation" $\mathscr{A}f = \alpha f$ as a "null space equation," namely, $\mathscr{A}f - \alpha f = 0$ or $(\mathscr{A} - \alpha\ id)\, f = 0$ for the identity operator with $id\ f = f$ for all f. Complete knowledge of the eigenvalues α and eigenfunctions f of a differential operator $\mathscr{A}$ allowed for a simple sum representations of the general solution of

the linear differential equation described by $\mathscr{A}$. Thus, Cauchy's "null space equation" became essential for determining the behavior of systems governed by linear differential equations.

Cauchy's knowledge of and interest in determinants (think of the Cauchy-Binet theorem) then led him to define the "characteristic polynomial" of a square matrix A in 1839 [5, p. 827] as $f_A(\alpha) = \det(A - \alpha I)$, and thereby he initiated renewed studies in polynomial root-finding algorithms in the hope of obtaining analogous diagonalization results for linear matrix times vector products. And the search for polynomial root finders was on. By modern-day hindsight, reducing the eigenvalue problem from an n^2 data problem of the entries of a matrix $A_{n,n}$ to one of the $n + 1$ coefficients of its characteristic polynomial is data compression, and therefore it was doomed to fail. But that remained unrealized by the mathematics community for more than 100 years.

James Sylvester finally gave the tableau concept of matrices its name "matrix" in 1848 or 1850. And after roughly two decades *1829 → 1839 → 1848/50*, the first century of matrix theory or theoretical linear algebra had begun.

2.6. Back to matrix computations

Cauchy's idea led mathematicians to try and compute the characteristic polynomials of matrices and find their roots in order to understand the eigen-behavior of matrices. We still teach many concepts and lessons today that are based on the "characteristic polynomial" $f_A(x)$ of a matrix A. Why, we should ask ourselves. Because unfortunately studying "characteristic polynomials" in place of matrices has turned out be a costly dead end for computational and applied mathematics: in the century and a half that followed Cauchy's work, more than 4000 papers on computing the roots of polynomials were published, together with 200 to 300 books on the subject, bringing us many algorithms, all of which failed more often than not. Many illustrious careers and schools of mathematics were founded based on this unfortunate and ever elusive goal.

During the same period, two-dimensional (2D) hand-cranked computing machines were invented and built to effect long number multiplications and divisions. First by Charles Babbage, then as commercial geared adding machines that were still being used in office work well into the 1960s. These worked as two-dimensional abaci of sorts. But eventually digital (at first punch card fed) computers became the tools of our computational trade in science, in engineering, in business, in GPS, in Google, in social media, in large data, in automation, etc.

But how could we or would we find matrix eigenvalues accurately? A turnaround, a new method, a new computational epoch was needed. From where, by whom, and how?

2.7. Iterative matrix algorithms: fifth epoch

To move us forward, it appears that matrix methods themselves might have to be developed that would solve the matrix-intrinsic eigenvalue problem by themselves. But before that was possible, there were further unfortunate "detours."

Carl Friedrich Gauß—in his doctoral thesis in 1799 [6]—had disproved all earlier attempts to establish the fundamental theorem of algebra, i.e., that all polynomials over the real numbers

http://dx.doi.org/10.5772/intechopen.73329

can be factored into as many real or complex conjugate factors as their degree says. His thesis then included the first complete and correct proof of the fundamental theorem of algebra.

In 1824 Niels Abel [7] showed that the roots of some fifth-degree polynomials cannot be found by using radical expressions of their coefficients; Gauß never opened or read the submitted paper and thus in fact rejected it knowingly on the grounds that God would not have complicated the World thus ... for us. Abel published his result privately, a broken man. Évariste Galois [8, 9] extended Abel's result in 1830 by giving group theoretic conditions for polynomials to be solvable by radicals; the extended paper (introducing Galois theory) was originally rejected and appeared only posthumously in 1846 [10].

Cautioned by these "rejected" inconvenient results, the polynomial approach to matrix eigenvalue computations could have been shunned by clearer heads early on, but the "dead end" determinants and characteristic polynomial roots road was taken instead for more than a century. Note that Cauchy's matrix result and most other fundamental matrix results from the nineteenth century were formulated in terms of determinants and only in the mid-twentieth century did the term "matrix" appear in matrix theoretical article and book titles.

A matrix-based approach to the eigenvalue problem nowadays starts from the simple fact that for any n by n matrix A and any n vector b, the sequence of vector iterates $b, Ab, A^2b, \ldots, A^kb, \ldots, A^nb$ contains $n+1$ vectors in n-space which makes these vectors linearly dependent. Their linear dependency then leads to an nth-degree polynomial $p_b(A)$ that sends b to zero. The vanishing polynomial for any b turns out to always be a factor of the characteristic polynomial of A and it can be found by Gaussian elimination rather than using determinants.

The same idea shows that vector iteration converges for every starting vector $b \neq 0$ and any given square matrix A and this has led engineers in the early twentieth century to construct iterative matrix algorithms that could solve linear equations and the matrix eigenvalue problem. Iterative matrix algorithms actually do go back further to the Jacobi method (1839, 1845) [11, 12], the so-called Gauß-Seidel method, invented by Seidel alone (1874) [13], and various SOR methods that are designed to solve linear systems iteratively. The latter generally use matrix splittings of A rather than vector iteration. For further thoughts on early iterative matrix methods, refer to Michele Benzi [14].

Alexei Krylov [15] introduced and studied the vector iteration subspaces $\text{span}\left\{b, Ab, \ldots, A^kb\right\}$ in their own right. Following his ideas, large sparse matrix systems are nowadays treated iteratively in so-called Krylov-based methods, both to solve linear equations and to find matrix eigenvalues. Standard widely used Krylov-type iterative matrix algorithms carry the names of steepest descent and conjugate gradient by Hestenes and Stiefel [16], Arnoldi [17], Lanczos [18]. Others are called GMRES, BICGSTABLE, QMR, ADI, etc. Most Krylov-type methods are matrix and problem specific, and they are now mostly used for huge sparse and structured matrices where direct or semi-direct methods cannot be employed due to their high computational and storage costs. Krylov methods generally rely on preconditioner M for a linear system $Ax = b$ that shifts the spectrum of $M^{-1}A$ for faster convergence, and they thrive on incomplete matrix splittings, etc. Typically, they give only partial results. Who would need or

want to know all million eigenvalues of a million by million matrix model. Krylov methods can be tuned to give information where it is needed for the underlying system.

2.8. Francis algorithm and matrix eigenvalues: sixth epoch

The Second World War (WW2) and post-Second World War periods were filled with innovations.

The atomic era had begun, as well as rocket science; commercial air flight became popular; and digital computers were being developed, first as valve machines and later transistorized. Supersonic speeds were realized, computer science was developed, etc. But there were many crashes and disasters with the new technologies: commercial aircraft (Super-Constellation, Convair, etc.) and military ones (Starfighter, etc.) would crash weekly around the globe; and newly built suspension bridges would collapse in strong winds.

The crux of the matter was that while matrix models of the underlying mechanical systems could readily be made using the laws of physics and mechanics, no one could reliably compute their eigenvalues. Engineers could not test their designs for eigenmodes in the right half plane! And Krylov methods were unfortunately not sufficient for testing for eigenvalues in a half plane.

If a matrix model of a mechanical or electrical or other structure, circuit, et cetera has right half-plane eigenvalues λ, then—upon proper excitation—there would be an eigen-component of the ever-increasing form $e^{\lambda t} \to \infty$ as $t \to \infty$ that resonates and self-amplifies inside the structure itself. This then leads to ever-increasing destructive vibrations and ultimate failure. The aircraft "flutter problem" was discovered during the Second World War. In England during WW2, Gershgorin circles that contain all of a system's eigenvalues were drawn out in the complex plane by rather primitive valve computers and checked to ascertain system stability.

The general matrix eigenvalue problem was finally solved independently and similarly by John Francis in London and by Vera Kublanovskaya in Russia nearly simultaneously around 1960. Francis' (or the QR) algorithm [19, 20] is based on Alston Householder's idea to try and solve matrix problems by matrix factorizations. Francis' method is an orthogonal subspace projection method and it works differently than the Krylov-based methods which solve a given matrix eigenvalue problem by projecting onto a Krylov subspace that is derived from and suitable for A.

A "divide and conquer" matrix factorization strategy was first employed by Heinz Rutishauser (1955, 1958) [21, 22] in his LR matrix eigenvalue algorithm: if one can factor $A = LR$ into the product of a lower and an upper triangular matrix L and R as $A = LR$ and if L is invertible, then for the reverse order product $A_1 = RL$ we have $A_1 = L^{-1}AL$ since $R = L^{-1}A$. If A_1 again allows an LR factorization $A_1 = L_1R_1$ with L_1 nonsingular, then by reverse order multiplication we obtain

$$A_2 = L_1^{-1}A_1L_1 = L_1^{-1}L^{-1}ALL_1$$

and so for the sequence of likewise constructed matrices A_i for $i = 3, \ldots$ if the respective LR factorizations are possible at each stage i. In this case the iterates A_i clearly remain similar to the original matrix A, and thus the iterates all have the same eigenvalues as A. Note, however,

that if, for example, the (1,1) entry $A(1,1)$ is zero in A, then there exists no un-pivoted LR factorization for A, and the method breaks down since pivoting is a one-sided matrix process and not a similarity. Therefore, Rutishauser's method is only applicable to a limited set of matrices A for which every LR iterate A_i allows an un-pivoted LR factorization. Rutishauser had noted that if LR factorizations are possible for all iterates A_i, and the A_i becomes nearly upper triangular for large i with A's eigenvalues on the diagonal. (Very loosely said.)

John Francis was very interested in the flutter problem at the time when, by chance, someone dropped Rutishauser's 1958 LR paper [22] on his desk at the CRDC in London. *(In my interview with John Francis in 2009* [23], *he did not know who that might have been.)* Francis was aware through contacts with Jim Wilkinson of the backward stability of algorithms that involve orthogonal matrices Q. So rather than using Gaussian elimination matrices L, Francis experimented with orthogonal $A = QR$ factorizations. At roughly the same time, Vera Kublanovskaya worked on an LQ factorization of A as $A = LQ$ and subsequent reverse order multiplications [24] in Leningrad, Russia. Her LQ algorithm would also compute the eigenvalues of matrices reliably.

Rutishauser had observed convergence speedup for his LR method when replacing A by $A - \alpha I$, i.e., shifting. Hence Francis experimented with shifts for QR and then established the "implicit Q theorem" [20] in order to circumvent computing eigenvalues of real matrices over the complex numbers. Implicit shifts also avoid rounding errors that would be introduced by explicit shifts. Francis' second paper (1962) [20] also contains a fully computed flutter matrix problem of size 24 by 24. The eigenvalues of such "large" problems had never before been computed successfully.

Francis' implicit Q theorem then allowed Gene Golub and Velvel Kahan [25] to compute singular values of large matrices for the first time, and this application later spawned the original Google search engine and brought us—in a way—into the Internet age.

In 2002 the multishift QR algorithm was developed by Karen Braman et al. [26, 27]. It relies on subspace iteration, extends Francis' QR, and combines it with Krylov like methods. This extension allows us today to compute the complete eigenvalue and singular-value structure of dense matrices of sizes up to 10,000 by 10,000 economically on laptops.

What is being missed today computationally? What epoch(s) might come next? Why and how?

2.9. Two new epochs ahead: the seventh and eighth epochs (yet to come)

Two new epoch generating impulses have become visible on the matrix computational horizon of today:

A. One expands our computational abilities from static problem-solving algorithms to real-time methods for time-varying problems.

B. The other involves computer hardware advances.

2.9.1. Time-varying problems and real-time solvers: seventh epoch

Our current best numerical codes can solve static problem very well; that is what they are designed for.

As we begin to rely more and more on time-dependent sensoring and on robotic manufacture, we need to learn how to solve our erstwhile static equations but now in real time and with time-varying coefficients and preferably accurately as well. It seems quite alluring to try and solve a time-varying problem by using the static time-dependent inputs at each instance statically. But such a naive solution cannot suffice since at the next time step, whose "solution" has just been computed "statically", the problem parameters have already changed and thus our "static" solution solves a completely different problem, which—unfortunately—has little value. *If any at all*.

2.9.2. Computer hardware: eighth epoch

Since the earliest electronic computing devices of the 1940s, all our computers have worked as giant and embellished Turing machines with logic gates, switches, and memory that rely only on two numerical states: 0 and 1 or on or off. Hence, all our computer data is stored and manipulated as sequences of 0 and 1.

Lately our computing ability has come up against the limits of storing and working with data and processors that can only deal with zeros and ones. Our processing speeds have not advanced significantly over the last couple of years; we are still stuck with 3–4 GHz processors. To alleviate this bottleneck, chip makers have created multiprocessor chips, and software firms have introduced better and quicker software and operating systems, but the basic processor speed has not budged much.

At this time computer scientist and manufacturers are trying to overcome this 0–1 bottleneck by replacing our 0–1 processors, chips, memories, and transistors by improved transistors and chips that can store and process multistates, such as 0-1-2-3-4 or 0-1-2-3-4-5-6-7-8 or even higher-numbered data representations. This could lead us to another computing sea change bringing us into a new computational epoch via hardware. And, further out on the horizon lies the possibility of having infinitely many quantum states based computers.

3. On neural network methods: seventh epoch (already under way)

The last century brought us valuable tools to solve most static problems that involve matrices.

Our current numerical matrix tools can solve static linear equations and matrix equations such as Sylvester or Lyapunov equations, as well as eigenvalue problems, and generalizations of all of these, both of the dense or structured and of the solvable or unsolvable kinds. Likewise, we can solve static optimization problems of all sizes and for nearly all structured matrices and thereby solve most if not all static applications.

But what can we do with such problems when the entries are time-varying and the problem parameters change over time?

In numerical computations, there has always been a see-saw between models that resulted in derivative-inspired differential equations and in linear algebra based matrix equations. Their

http://dx.doi.org/10.5772/intechopen.73329

respective computational advantages differ from problem to problem. Neural networks (NN) are an amalgam of matrix methods and differential methods and use a mixture of both. NN methods are designed to solve time-varying dynamical systems. Numerical methods for time-varying dynamical systems first came about in the 1950s and subsequently have gained strength in the 1980s and 1990s and beyond (see the introduction in Getz and Marsden [28], for example). There are essentially three ways to go about solving the dynamical systems via differential equations: homotopy methods, gradient methods, and ZNN neural network methods introduced by Yunong Zhang et al. [29]. To solve a time-varying equation $f(X(t)) = G(t)$, the ZNN method starts from the error equation $E(t) = f(X(t)) - G(t)$ and stipulates exponential decay of the error function $E(t)$ to zero by trying to solve the differential equation:

$$\dot{E}(t) = -\lambda E(t) \tag{1}$$

for a positive decay constant λ. Ideally, the error differential equation (1) of a given problem can be discretized using high-order and convergent 1-step ahead difference formulas for the unknown. Their aim is to predict or simulate the solution at time t_{k+1} in real time from earlier event instants t_j with $j \leq k$ with a high degree of accuracy and low storage cost. The ZNN method will be explained below for three standard time-varying problems.

3.1. A neural network approach to solve time-varying linear equations: $\mathbf{A}(\mathbf{t})\mathbf{x}(\mathbf{t})=\mathbf{b}(\mathbf{t})$

Here $A(t)$ is a nonsingular time-varying n by n matrix and $b(t) \in \mathbb{R}^n$ is a time-varying vector, respectively. Clearly, the unknown solution $x(t)$ of the associated linear equation $A(t)x(t) = b(t)$ will be time-dependent as well.

The first paper on Zhang neural networks (ZNN) was written by Yunong Zhang et al. [29]. Today, there are well over 300 papers, mostly in engineering journals that deal with time-varying applications of the ZNN method, either in hardware chip design for specialized computational tasks as part of a plant or machine or for time-varying simulation problems in computer algorithms and codes. Unfortunately, the ZNN method and the ideas behind ZNN are hardly known today among numerical analysis experts. The method itself starts with using Suanjing's and Al Khwarizmi's ancient rule for reducing equations which first appeared 1 1/2 millennia ago. Recall that this simple rule was also employed by Cauchy [4] to transform the static matrix eigenvalue problem from $Ax = \lambda x$ to $Ax - \lambda x = 0$ and finally to $\det(A - \lambda I) = 0$. For the time-varying linear equation problem, Zhang's neural network method starts with

$$A(t)x(t) - b(t) = 0$$

and then works on the error function

$$E(t) = A(t)x(t) - b(t) : \mathbb{R} \to \mathbb{R}^n$$

and its time derivative $\dot{E}(t)$. Note that standard static methods would likely look at the error norm $\|E\|$ instead of the error function. Neural networks do not; they study the error function $E(t)$ instead. And they start with an implicit "ideal wish": *What could or should we wish for E*?

Time-varying computations would be near ideal if their error functions $E(t)$ were decaying exponentially fast as functions of t. This is impossible to achieve (or even ask for) with our best static equations and problem solvers of the twenty-first century. For static numerical matrix methods, backward stability is considered most desirable.

In Zhang NN methods stipulating that the error function $E(t)$ decreases exponentially fast over time to the zero function means that

$$\dot{E}(t) = -\lambda E(t) \text{ for some chosen number } \lambda \gg 0, \text{ the decay constant.}$$

Note that for the time-varying linear equation problem, we have

$$\dot{E}(t) = \dot{A}(t)x(t) + A(t)\dot{x}(t) - \dot{b}(t).$$

This leads to the following differential equation for the time-varying solution $x(t)$:

$$A(t)\dot{x}(t) = -\dot{A}(t)x(t) + \dot{b}(t) - \lambda(A(t)x(t) - b(t)). \tag{2}$$

And thus, we have transformed the time-varying linear equations problem into an initial value differential equation problem that needs to be solved for $t > 0$. This is where the different dynamical system methods split their ways. In Zhang neural networks, the continuous time differential equation (2) is then discretized for $0 < t_j < t_{end}$ and the ensuing derivatives are approximated by high-order difference quotients, with the one for the unknown $\dot{x}(t_j)$ being 1-step ahead and proven convergent, while the others such as for $\dot{A}(t_j)$ and $\dot{b}(t_j)$ in equation (2) above can be backward difference formulas. This process would then yield a way to generate $x(t_{j+1})$ from earlier known data such as $x(t_k)$, $A(t_k)$, $\dot{A}(t_k)$ and $b(t_k)$ and $\dot{b}(t_k)$ for indices $k \leq j$. How to proceed in this problem from equation (2) with solving $A(t)x(t) = b(t)$ via ZNN methods is still an open problem, especially for large-scale sparse or structured time-varying linear equations since the matrix $A(t_j)$ encumbers the unknown $\dot{x}(t_j)$ on the left-hand side of equation (2) and there is no known 1-step ahead differentiation formula that can be used here.

The general idea that underlies ZNN methods for time-varying problems is to replace repeated matrix computations by solving linear differential equations and associated initial value problems for discrete instances $0 < t_j < t_{end}$ instead.

3.2. A Zhang neural network approach to find time-varying generalized matrix inverses $\mathbf{Y(t)}$ for time-varying full rank matrices $\mathbf{B(t)}$ so that $\mathbf{B(t)_{m,n}Y(t)_{n,m} = I_m}$

This section is based on joint work with Jian Li et al. [30].

3.2.1. Continuous problem formulation

For an m by n real time-varying matrix $B(t)$ of full rank m with $m \leq n$, we form the matrix-valued error function:

http://dx.doi.org/10.5772/intechopen.73329

$$E(t) = B(t) - Y^{+}(t) \in \mathbb{R}^{m\times n} \tag{3}$$

where the upper + sign always means "generalized inverse." Then, we use the Zhang design formula:

$$\dot{E}(t) = -\lambda E(t) \tag{4}$$

with design parameter $\lambda > 0$. Based on [31, Lemma 3], we have

$$\dot{Y}^{+}(t) = -Y^{+}(t)\dot{Y}(t)Y^{+}(t). \tag{5}$$

And from equations (3) and (5), we obtain

$$\dot{E}(t) = \dot{B}(t) - \dot{Y}^{+}(t) = \dot{B}(t) + Y^{+}(t)\dot{Y}(t)Y^{+}(t). \tag{6}$$

Combining equations (4) and (6), we then get

$$\dot{B}(t) + Y^{+}(t)\dot{Y}(t)Y^{+}(t) = -\lambda\big(B(t) - Y^{+}(t)\big) \tag{7}$$

And, by right multiplying equation (7) with $Y(t)$, we have

$$\Big(\dot{B}(t) + Y^{+}(t)\dot{Y}(t)Y^{+}(t)\Big)Y(t) = -\lambda\big(B(t) - Y^{+}(t)\big)Y(t). \tag{8}$$

With $m \le n$, we have $Y^{+}_{m\times n}(t)Y_{n\times m}(t) = I_{m\times m}$, and thus

$$\dot{B}(t)Y(t) + Y^{+}(t)\dot{Y}(t) = -\lambda(B(t)Y(t) - I). \tag{9}$$

The solution of a generalized matrix inverse problem is not unique when $m < n$, and we only need to find a solution that satisfies equation (9). Consequently, the continuous model can be represented as

$$\dot{Y}(t) = -\lambda(Y(t)B(t)Y(t) - Y(t)) - Y(t)\dot{B}(t)Y(t) \tag{10}$$

which agrees completely with [28, formula (15), p. 317]. Substituting equation. (10) into equation. (9), we have

$$\dot{B}(t)Y(t) + Y^{+}(t)\big(-\lambda(Y(t)B(t)Y(t) - Y(t)) - Y(t)\dot{B}(t)Y(t)\big) = -\lambda(B(t)Y(t) - I), \tag{11}$$

which we rewrite as

$$\dot{B}(t)Y(t) + \big(-\lambda\big(Y^{+}(t)Y(t)B(t)Y(t) - Y^{+}(t)Y(t)\big) - Y^{+}(t)Y(t)\dot{B}(t)Y(t)\big) = -\lambda(B(t)Y(t) - I). \tag{12}$$

With $Y^{+}_{m\times n}(t)Y_{n\times m}(t) = I_{m\times m}$, we have

$$\dot{B}(t)Y(t) + \left(-\lambda(B(t)Y(t) - I) - \dot{B}(t)Y(t)\right) = -\lambda(B(t)Y(t) - I). \tag{13}$$

Thus model (10) satisfies model (9), and its solution solves the time-varying generalized matrix inverse problem.

3.2.2. Zhang neural network discretization

Given a sequence of rectangular matrices B_j at time instances $t_j \leq t_k$, we want to find the discrete time-varying generalized matrix inverse Y_{k+1} of B_{k+1} on each computational time interval $[k\tau, (k+1)\tau) \subseteq [0, t_f]$ so that

$$B_{k+1} - Y_{k+1}^{+} = 0. \tag{14}$$

Here $B_{k+1} = B(t_{k+1}) = B((k+1)\tau) \in \mathbb{R}^{m\times n}$ is a time-varying full rank equidistant matrix sequence, $m \leq n$, and $Y_{k+1} \in \mathbb{R}^{n\times m}$ is unknown. Y_{k+1} needs to be computed in real time for each time interval $[k\tau, (k+1)\tau) \subseteq [0, t_{end}]$. Here the matrix operator $^{+}$ denotes the generalized inverse of a matrix and $0 \in \mathbb{R}^{m\times n}$ is the zero matrix. Besides, $k = 0, 1, \cdots$ denotes the updating index, t_{end} denotes the task duration, and τ denotes the constant sampling gap of the time-varying matrix sequence B_{k+1}. For $m > n$, the procedure is similar.

Note that we must obtain each Y_{k+1} at or before time t_{k+1} for real-time calculations, while the actual value of B_{k+1} is unknown before t_{k+1}. Thus we cannot obtain the solution by calculating $Y_{k+1} = B_{k+1}^{+}$. To obtain Y_{k+1} in real time, we must develop a model based on the available information before t_{k+1} such as that in B_j, Y_j, and Y_{j-1} for $j \leq k$ instead of unknown information such as B_{k+1}.

To obtain a discrete time model that solves the original discrete time-varying generalized matrix inverse problem (14), we need to discretize the continuous model (10). First, we use the conventional 1-step forward Euler formula:

$$\dot{x}(t_k) \approx \frac{x(t_{k+1}) - x(t_k)}{\tau} \tag{15}$$

with truncation error of order $O(\tau)$. Based on formula (15), we approximate

$$\dot{Y}(t_k) = \frac{Y(t_{k+1}) - Y(t_k)}{\tau} \tag{16}$$

and use this equation to discretize the continuous model (10) as follows:

$$Y_{k+1} = -h(Y_k B_k Y_k - Y_k) - \tau Y_k \dot{B}_k Y_k + Y_k. \tag{17}$$

Here $h = \tau\lambda$. In most real-world applications, information of the first-order time derivatives, i.e., the value of $\dot{B}_k$, may not be explicitly known for the discrete time-varying generalized matrix inverse problem (14). If this is so, the value of $\dot{B}_k$ can be approximated by a backward finite difference formula. To assure the accuracy and simplicity of the discretized model, the

http://dx.doi.org/10.5772/intechopen.73329

truncation error of the backward finite difference formula for $\dot{B}_k$ should be near equal to that of the 1-step ahead finite difference formula that approximates $\dot{Y}_k$. Thus, $\dot{B}_k$ in equation (17) should best be approximated by Euler's backward finite difference formula:

$$\dot{b}_k \approx \frac{b_k - b_{k-1}}{\tau}, \tag{18}$$

because the truncation error order $O(\tau)$ of formula (18) equals that of formula (15). Thus, we approximately have

$$\dot{B}_k = \frac{B_k - B_{k-1}}{\tau}. \tag{19}$$

Then we combine equation (17) with equation (19) and the Euler discrete model becomes

$$Y_{k+1} = -h(Y_k B_k Y_k - Y_k) - Y_k(B_k - B_{k-1})Y_k + Y_k. \tag{20}$$

Note that the truncation error of the discrete model (20) is of order $\mathbf{O}(\tau^2)$ where the symbol $\mathbf{O}(\tau^2)$ denotes a matrix in which each entry is of order $O(\tau^2)$. This model uses only present or past information of B_k, B_{k-1}, and Y_k and solves for Y_{k+1}. Thus Y_{k+1} can be calculated during the time interval $[t_k, t_{k+1})$ and if Y_{k+1} can be computed quickly enough in real time it will be ready when time instant t_{k+1} arrives.

Higher-accuracy 1-step ahead formulas exist for discrete models, namely,

$$\dot{x}(t_k) \approx \frac{2x(t_{k+1}) - 3x(t_k) + 2x(t_{k-1}) - x(t_{k-2})}{2\tau} \tag{21}$$

and

$$\dot{x}(t_k) \approx \frac{6x(t_{k+1}) - 3x(t_k) - 2x(t_{k-1}) - x(t_{k-2})}{10\tau}. \tag{22}$$

Both have truncation errors of order $O(\tau^2)$. For simplicity we only consider formula (21) and call it the 4-IFD formula because four instances in time are used to approximate the first-order derivative of $x(t_k)$. When we employ the 4-IFD formula (21) inside our continuous model (10), we obtain

$$Y_{k+1} = -h(Y_k B_k Y_k - Y_k) - \tau Y_k \dot{B}_k Y_k + \frac{3}{2}Y_k - Y_{k-1} + \frac{1}{2}Y_{k-2}. \tag{23}$$

Next, we use the three-instant backward finite difference formula:

$$\dot{b}_k \approx \frac{3b_k - 4b_{k-1} + b_{k-2}}{2\tau} \tag{24}$$

with error order $O(\tau^2)$ to approximate the value of $\dot{B}_k$ in equation (23). Then the 4-IFD-type discretized model becomes

$$Y_{k+1} = -h(Y_k B_k Y_k - Y_k) - Y_k\left(\frac{3}{2}B_k - 2B_{k-1} + \frac{1}{2}B_{k-2}\right)Y_k + \frac{3}{2}Y_k - Y_{k-1} + \frac{1}{2}Y_{k-2}. \tag{25}$$

Its truncation error is of order $\mathbf{O}(\tau^3)$. Similar to the Euler based discrete model (20), the 4-IFD-type discrete model uses only present and past information such as B_k, B_{k-1}, B_{k-2}, Y_k, Y_{k-1}, and Y_{k-2} to solve for Y_{k+1}. Thus it also satisfies the requirements for real-time computation.

3.2.3. A five-instant finite difference formula

Any usable finite difference formula for discretizing the continuous model (10) must satisfy several restrictions. It must be one step ahead for $\dot{x}$, i.e., approximate $\dot{x}(t_k)$ by using only $x(t_{k+1})$, $x(t_k)$, $x(t_{k-1})$ and possibly earlier x data, and it must be 0-stable and convergent. However, 1-step ahead finite difference formulas do not necessarily generate stable and convergent discrete models (see, e.g., [32, 33]).

Here is a new 1-step ahead finite difference formula with higher accuracy than the Euler and 4-IFD formulas. It will be used to generate a stable and convergent discrete model that finds time-varying generalized matrix inverses more accurately in real time.

Theorem 1 *The 5-IFD formula*

$$\dot{x}(t_k) \approx \frac{8x(t_{k+1}) + x(t_k) - 6x(t_{k-1}) - 5x(t_{k-2}) + 2x(t_{k-3})}{18\tau} \tag{26}$$

has truncation error order $O(\tau^3)$.

The proof relies on four Taylor expansions that use $x(t_{k+1})$ and $x(t_{k-1})$ through $x(t_{k-3})$ around $x(t_k)$ and clever linear combinations thereof.

The new 1-step ahead discretization formula (26) then leads to the five-instant discrete model:

$$\begin{aligned} Y_{k+1} = \quad & -\frac{9}{4}h(Y_k B_k Y_k - Y_k) - \frac{9}{4}Y_k\left(\frac{11}{6}B_k - 3B_{k-1} + \frac{3}{2}B_{k-2} - \frac{1}{3}B_{k-3}\right)Y_k \\ & - \frac{1}{8}Y_k + \frac{3}{4}Y_{k-1} + \frac{5}{8}Y_{k-2} - \frac{1}{4}Y_{k-3}. \end{aligned} \tag{27}$$

which has a truncation error of order $\mathbf{O}(\tau^4)$.

Theorem 2 *The five-instant discrete model (27) is 0-stable.*

The multistep formula of the five-instant discrete model time-varying generalized matrix inverses has the characteristic polynomial:

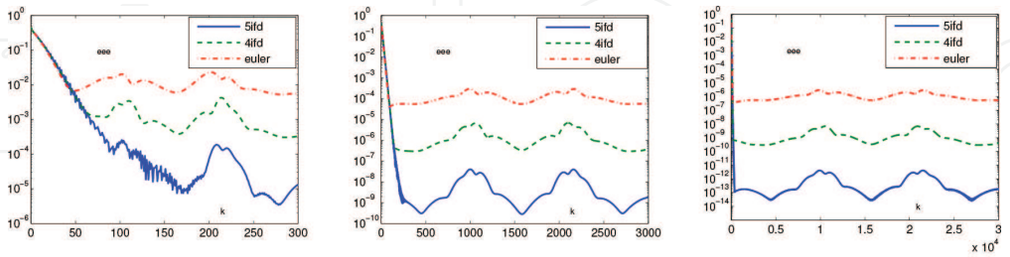
$$P_4(\theta) = \theta^4 + \frac{1}{8}\theta^3 - \frac{3}{4}\theta^2 - \frac{5}{8}\theta + \frac{1}{4} \tag{28}$$

with four distinct roots $\theta_{1,2} = -0.7160 \pm 0.5495i$, $\theta_3 = 0.3069$, and $\theta_4 = 1$ inside the complex unit circle, making this model 0-stable (**Figures 1** and **2**).

3.2.4. Numerical examples

Example 1 Consider the discrete time-varying generalized matrix inverse problem:

$$B_{k+1} - Y_{k+1}^{+} = 0, \text{ with } B_k = \begin{bmatrix} \sin(0.5t_k) & \cos(0.1t_k) & -\sin(0.1t_k) \\ -\cos(0.1t_k) & \sin(0.1t_k) & \cos(0.1t_k) \end{bmatrix}. \tag{29}$$

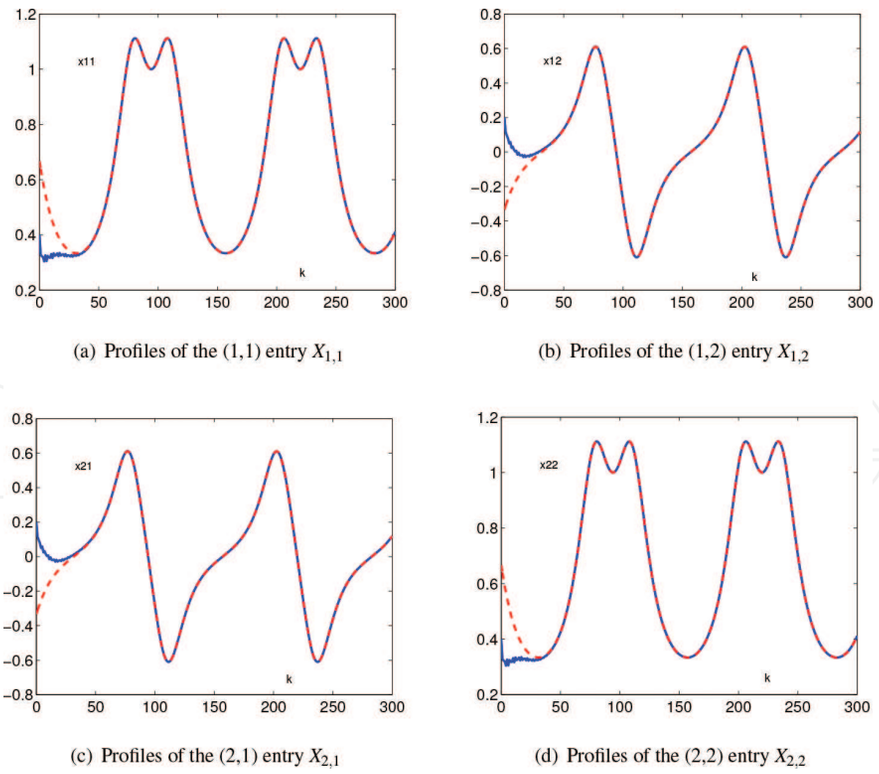


(a) With $\tau = 0.1$ s (b) With $\tau = 0.01$ s (c) With $\tau = 0.001$ s

Figure 1. Typical residual errors generated by the five-instant, the four-instant, and the Euler formulas with different sampling gaps τ when solving the discrete time-varying generalized matrix inverse problem (29) for $t_{end} = 30$ s and $h = 0.1$.

Example 2 Here we consider the discrete time-varying matrix inversion problem:

$$A_{k+1}X_{k+1} = I \quad \text{with } A_k = \begin{bmatrix} \sin(0.5t_k) + 2 & \cos(0.5t_k) \\ \cos(0.5t_k) & \sin(0.5t_k) + 2 \end{bmatrix}. \tag{30}$$



(a) Profiles of the (1,1) entry $X_{1,1}$ (b) Profiles of the (1,2) entry $X_{1,2}$

(c) Profiles of the (2,1) entry $X_{2,1}$ (d) Profiles of the (2,2) entry $X_{2,2}$

Figure 2. Profiles of the four entries of the solution X when solving the discrete time-varying matrix inverse problem (30) with $\tau = 0.1$ s. Here the solid curves show the solution entries generated by the five-instant discrete model obtained from random starting values, and the dash-dotted curves depict the theoretical solutions.

3.3. A Zhang neural network approach for solving nonlinear convex optimization problems under time-varying linear constraints

This section is based on joint work with Jian Li et al. [34].

Problem formulation:

$$\begin{array}{ll} \text{find} & \min\ f(x(t),t) \\ \text{such that} & A(t)x(t) = b(t) \quad \text{with } x(t)\in\mathbb{R}^n,\ \ b(t)\in\mathbb{R}^m \text{ and } A(t)\in\mathbb{R}^{m,n}. \end{array}$$

Building a continuous time model for the problem:

The Zhang neural network approach can be built on the Lagrange function:

$$L(x(t),l(t),t) = f(x(t),t) + l^T(t)(A(t)x(t) - b(t)),$$

where $l(t) = [l_1(t),\cdots,l_m(t)]^T \in \mathbb{R}^m$ is the Lagrange multiplier vector and $..^T$ denotes the transpose. Note that there will be no need to solve for the Lagrange functions $l(t)$ here. Set

$$y(t) = \left[x^T(t), l^T(t)\right]^T = \left[y_1(t),\cdots,y_n(t),y_{n+1}(t),\cdots,y_{n+m}(t)\right]^T \in \mathbb{R}^{n+m}$$

and

$$h(y(t),t) = \begin{bmatrix} \dfrac{\partial f(x(t),t)}{\partial x} + A^T(t)l(t) \\ A(t)x(t) - b(t) \end{bmatrix} = \begin{bmatrix} h_1(y(t),t) \\ \vdots \\ h_n(y(t),t) \\ h_{n+1}(y(t),t) \\ \vdots \\ h_{n+m}(y(t),t) \end{bmatrix} \in \mathbb{R}^{n+m}.$$

We transform the multiplier problem into an initial value DE problem instead. By stipulating exponential decay for $h(t)$, we obtain the model equation

$$\dot{y}(t) = -H^{-1}(y(t),t)(\lambda h y(t),t) + \dot{h}_t(y(t),t))$$

for the Jacobian matrix

$$H(y(t),t) = \begin{bmatrix} \dfrac{\partial f^2(x(t),t)}{\partial x\ \partial^T x} & A^T(t) \\ A(t) & 0 \end{bmatrix} \text{and } \dot{h}_t(y(t),t) = \frac{\partial h(y(t),t)}{\partial t}\ .$$

3.3.1. Discretizing the model and choosing suitable high-order finite difference formulas

To discretize the continuous model

$$\dot{y}(t) = -H^{-1}(y(t),t)\Big(\lambda h y(t),t) + \dot{h}_t(y(t),t)\Big)$$

we can use the forward Euler difference formula with truncation error order $O(\tau)$:

$$\dot{x}(t_k) = \frac{x(t_{k+1}) - x(t_k)}{\tau}$$

or the four-instant forward difference formula (4-IFD):

$$\dot{x}(t_k) = \frac{5x(t_{k+1}) - 3x(t_k) - x(t_{k-1}) - x(t_{k-2})}{8\tau}$$

with truncation error order $O(\tau^2)$. The Euler formula yields the discretized model:

$$y_{k+1} = -H^{-1}(y_k, t_k)\Big(\kappa h(y_k, t_k) + \tau \dot{h}_t(y_k, t_k)\Big) + y_k \text{ with } \kappa = \tau\lambda$$

while the 4-IFD formula results in

$$y_{k+1} = -\frac{8}{5}H^{-1}(y_k, t_k)\Big(\kappa h(y_k, t_k) + \tau \dot{h}_t(y_k, t_k)\Big) + \frac{3}{5}y_k + \frac{1}{5}y_{k-1} + \frac{1}{5}y_{k-2} + O(\tau^3).$$

Both discretization formulas are consistent and convergent. This can be proven via the roots of the associated characteristic polynomial. Its roots must lie in the complex unit circle and cannot be repeated on its boundary.

Since the value of $\dot{h}_t(y_k, t_k)$ may not be known explicitly, we may replace it by

$$\dot{h}_t(y_k, t_k) = \frac{3h(y_k, t_k) - 4h(y_k, t_{k-1}) + h(y_k, t_{k-2})}{2\tau}$$

which uses the three-point backward finite difference formula:

$$\dot{x}(t_k) = \frac{3x(t_k) - 4x(t_{k-1}) + x(t_{k-2})}{2\tau}$$

of order $O(\tau^2)$.

Then the discretized 4-IFD formula becomes more complicated but easier to implement:

$$y_{k+1} = -\frac{8}{5}H^{-1}(y_k, t_k)\left(\left(\kappa + \frac{3}{2}\right)h(y_k, t_k) - 2h(y_k, t_{k-1}) + \frac{1}{2}h(y_k, t_{k-2})\right) + \frac{3}{5}y_k + \frac{1}{5}y_{k-1} + \frac{1}{5}y_{k-2} \quad \text{of order} \quad O(\tau^3).$$

To implement this formula, the inverse of the Jacobian matrix H can be computed at each time t_k in a fraction of the available real-time interval $[t_k, t_{k+1})$ by using the real-time inverse finding ZNN method from the previous subsection (Section 3.2).

3.3.2. Numerical example and results:

As an example we solve the following convex nonlinear optimization problem with known theoretical solution numerically by using our ZNN method; for further details and applications see [34]:

$$\begin{aligned} \text{Find} \quad & \min \quad (\cos(0.1t_{k+1})+2)x_1^2+(\cos(0.1t_{k+1})+2)x_2^2+2\sin(t_{k+1})x_1x_2+\sin(t_{k+1})x_1+\cos(t_{k+1})x_2 \\ & \text{so that} \quad \sin(0.2t_{k+1})x_1+\cos(0.2t_{k+1})x_2=\cos(t_{k+1}). \end{aligned}$$

The 4-IFD formula is a four-instant formula, while the Euler formula needs only two. Both discretization models work in real time, and both typically create the optimal solution in a fraction of a second with differing degrees of accuracy according to their orders.

The example below runs for 10 sec. The time-varying values for $f(x(t),t)$, $A(t)$, and $b(t)$ are given as functions and evaluated from their function formulations. In real-world applications, these values might be supplied by sensors during each time interval $t_i \le t_{i+1}$, and the empirical values would be inserted into the difference formulas as they are evaluated by sensors in real time (**Figure 3**).

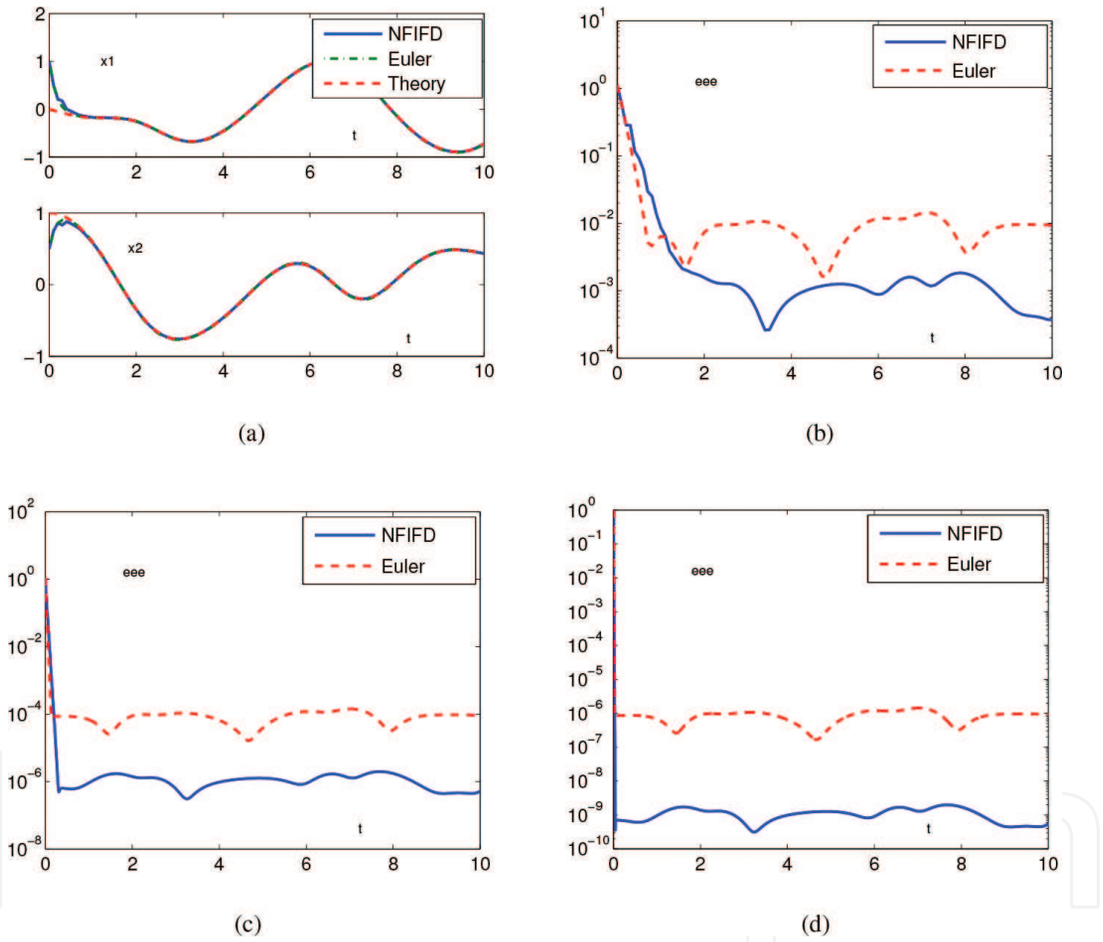


Figure 3. Solution states with $\tau = 0.1$ sec and solution errors generated by the 4-IFD based discretization model (of order $O(\tau^3)$) and the Euler based model (of order $O(\tau^2)$) with $\tau = 0.1$, 0.01, and 0.001 sec and $\lambda = 10$: (a) solution states with $\tau = 0.1$ sec, (b) solution errors with $\tau = 0.1$ sec, (c) solution errors with $\tau = 0.01$ sec, (d) solution errors with $\tau = 0.001$ sec.

4. On quantum and multistate computing: eight epoch (yet to start and come)

Quantum computing and multistate memory and computers with multistate processors will change the way we compute once they become available. They will require new operating

http://dx.doi.org/10.5772/intechopen.73329

systems and new software with new and yet-to-be-discovered algorithms. What will this new era entail? Nobody knows or can reliably predict.

I asked an "expert" on quantum computing 3 years ago as to when he expected to have a quantum computer at his disposal or on his desk. The answer was "Not in my lifetime, not in 20 years."

Currently, about a dozen or more research centers in Europe and South-East Asia are trying to build quantum computers based on the quantum superposition principle and quantum entanglement of elementary particles. They do so in a multitude of different ways. The envisioned benefit of these efforts would be to be able to compute superfast in parallel and in simulations to solve huge data problems quicker than ever before and to solve problems that are unassailable now with our current best supercomputer networks. All of the proposed quantum science techniques make use of superconducting circuits and particles. The aim is to build quantum computers in one or two decades with around 100 entangled quantum bits. Such a quantum computer would be bulky; it would need much supplementary equipment for cooling and so forth and could easily take up a whole floor of a building, just as the first German and British valve computers did in the 1940s. But it would surpass the computing capacity of all current supercomputers and desk and laptops on Earth combined. Currently, the largest working entangled quantum array contains fewer than 10 quantum bits. Access of a 100 bit quantum computer would probably be via the cloud and there would be no quantum computer laptops. Quantum computers may take another 10, 20, or 30 years to materialize.

How will they come about? Which yet unknown algorithms will they use? Who will invent them? Who code them?

If history can be a guide, John Francis and Vera Kublanovskaya were both working independently on circuit diagrams and logic gate designs for valve computers in England and in Russia at the time when they discovered QR (or LQ) in the late 1950s.

So, we possibly are looking for quantum computer hardware and software designers who know numerical analysis and algorithm development in or about the year 2040. In a similar fashion, Leibnitz and Seki formalized our now ubiquitous matrix concept independently but simultaneously in 1683, in Germany and in Japan.

Maybe it will take two again?

The references given below only go back to the year 1799.

Author details

Frank Uhlig

Address all correspondence to: uhligfd@auburn.edu

Department of Mathematics and Statistics, Auburn University, Alabama, United States

References

[1] Horner W. G. A new method of solving numerical equations of all orders, by continuous approximation. Philosophical Transactions. Royal Society of London. July 1819:308-335

[2] Gauß C. F. Private letter to Gerling. December 26, 1823. Available at: http://gdz.sub.uni-goettingen.de/en/dms/loader/img/?PID=PPN23601515X|LOG_0112physid=PHYS_0286. pp. 278-281

[3] Taussky O. Note on the condition of matrices. Mathematical Tables and Other Aids to Computation. 1950;**4**:111-112

[4] Cauchy A. L. Sur l'équation à l'aide de laquelle on determine les inégalités séculaires des mouvements des planètes. Exerc. de Math. 1829;**4**. also Oeuvres (2) 9, pp. 174-195

[5] Cauchy A. L. Mémoire sur l'intégration des équations linéaires. Comptes Rendus. 1839;**8**: 827-830, 845-865, 889-907, 931-937

[6] Gauß C. F. Demonstratio nova theorematis omnem functionem algebraicam rationalem integram unius variabilis in factores reales primi vel secundi gradus resolvi posse. PhD thesis: Universität Helmstedt; 1799, Werke III. pp. 1-30

[7] Abel N. H. Mémoire sur les équations algébriques où on démontre l'impossibilité de la résolution de l'équation générale du cinquième degré. Christiana (Kopenhagen): Groendahl; 1824. 7 p

[8] Galois É. Analyse d'un mémoire sur la résolution algébraique des équations. Bulletin des Sciences mathématiques, physiques et chimiques. 1830;**XIII**:271-272

[9] Galois É. Note sur la résolution des équations numériques. Bulletin des Sciences mathématiques, physiques et chimiques. 1830;**XIII**:413-414

[10] Galois É. Mémoire sur les conditions de résolubilité des équations par radicaux. Journal de mathématiques pures et appliquées, (published by Joseph Liouville). 1846;**XI**:417-433

[11] Jacobi C. G. Canon Arithmeticus. Berlin: Typis Academicis Berolini; 1839

[12] Jacobi C. G. Über eine neue Auflösungsart der bei der Methode der kleinsten Quadrate vorkommenden linearen Gleichungen. Astronomische Nachrichten. 1845;**22**:297-306. Reprinted in Gesammelte Werke, vol. III, pp. 469-478

[13] Ludwig von Seidel, Über ein Verfahren, die Gleichungen, auf welche die Methode der kleinsten Quadrate führt, sowie lineäre Gleichungen überhaupt, durch successive Annäherung aufzulösen, Lecture at the Bavarian Academy on February 7, 1874, Abhandlungen der Bayerischen Akademie der Wissenschaften II. Mathematisch-Physikalische Klasse. Cl. XI, Bd. III Abth, München. 1874. 28 pp

[14] Benzi M. The early history of matrix iterations: With focus on the Italian contribution. SIAM Conference lecture. October 2009. 36 pp. https://www.siam.org/meetings/la09/talks/benzi.pdf

[15] Krylov A. N. On the Numerical Solution of Equation by Which are Determined in Technical Problems the Frequencies of Small Vibrations of Material Systems. News of Academy of Sciences of the USSR), Otdel. mat. i estest. nauk. 1931;**VII**(4):491-539 (in Russian)

[16] Hestenes M. R., Stiefel E. Methods of conjugate gradients for solving linear systems. Journal of Research of the National Bureau of Standards. 1952;**49**(6):409-436

[17] Arnoldi W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. Quarterly of Applied Mathematics. 1951;**9**:17-29

[18] Lanczos C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. Journal of Research of the National Bureau of Standards. 1950;**45**:255-282

[19] Francis J. G. F. The QR transformation, I. The Computer Journal. 1961;**4**:265-271

[20] Francis J. G. F. The QR transformation, II. The Computer Journal. 1962;**4**:332-345

[21] Rutishauser H. Une methode pour la determination des valeurs propres d'une matrice. Comptes Rendus de l'Académie des Sciences. 1955;**240**:34-36

[22] Rutishauser H. Solution of eigenvalue problems with the LR-transformation. National Bureau of Standard: Applied Mathematics Series. 1958;**49**:47-81

[23] Golub G, Uhlig F. The QR algorithm: 50 years later; its genesis by John Francis and Vera Kublanovskaya, and subsequent developments. IMA Journal of Numerical Analysis. 2009;**29**:467-485

[24] Kublanovskaya V. N. On some algorithms for the solution of the complete eigenvalue problem. USSR Computational Mathematics and Mathematical Physics. 1963;**1**:pp. 637-657 (1963, received Feb 1961). Also published in: Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki [Journal of Computational Mathematics and Mathematical Physics]. 1 (1961). pp. 555-570

[25] Gene Golub W. Kahan, calculating the singular values and pseudo-inverse of a matrix. Journal of SIAM Numerical Analysis Series B. 1965;**2**:205-224

[26] Braman K, Byers R, Mathias R. The multishift QR algorithm. Part I: Maintaining well-focused shifts and level 3 performance. SIAM Journal on Matrix Analysis & Applications. 2006;**23**:929-947

[27] Braman K, Byers R, Mathias R. The multishift QR algorithm. Part II: Aggressive early deflation, SIAM Journal on Matrix Analysi & Application. 2006;**23**:948-973

[28] Getz N, Marsden J. E. Dynamical methods for polar decomposition and inversion of matrices. Linear Algebra and its Applications. 1997;**258**:311-343

[29] Zhang Y, Jiang D, Wang J. A recurrent neural network for solving Sylvester equation with time-varying coefficients. IEEE Transactions on Neural Networks. 2002;**13**:1053-1063

[30] Jian L, Mingzhi M, Frank U, Yunong Z. A 5-Instant finite difference formula to find discrete time-varying generalized matrix inverses, matrix inverses and scalar reciprocals, 17 pp, submitted

[31] Liao B, Zhang Y. From different ZFs to different ZNN models accelerated via li activation functions to finite-time convergence for time-varying matrix pseudoinversion. Neurocomputing. 2014;**133**:512-522

[32] Zhang Y, Jin L, Guo D, Yin Y, Chou Y. Taylor-type 1-step-ahead numerical differentiation rule for first-order derivative approximation and ZNN discretization. Journal of Computational and Applied Mathematics. 2014;**273**:29-40

[33] Zhang Y, Chou Y, Chen J, Zhang Z, Xiao L. Presentation, error analysis and numerical experiments on a group of 1-step-ahead numerical differentiation formulas. Journal of Computational and Applied Mathematics. 2013;**239**:406-414

[34] Li J, Mao M, Uhlig F, Zhang Y. Z-type neural-dynamics for time-varying nonlinear optimization under a linear equality constraint with robot application. Journal of Computational and Applied Mathematics. 2018;**327**:155-166

IntechOpen