FORMAL ANALYSIS OF THE SCHULZ MATRIX INVERSION ALGORITHM: A PARADIGM TOWARDS COMPUTER AIDED VERIFICATION OF GENERAL MATRIX FLOW SOLVERS

Vassilios A. Tsachouridis*, Georgios Giantamidis**, Stylianos Basagiannis and Kostas Kouramas

United Technologies Research Center Ltd.
2nd Floor Penrose Wharf Business Centre, Penrose Quay
Cork, T29 XN53, Ireland

(Communicated by Yongzhong Song)

ABSTRACT. This paper pilots Schulz generalised matrix inverse algorithm as a paradigm in demonstrating how computer aided reachability analysis and theoretical numerical analysis can be combined effectively in developing verification methodologies and tools for matrix iterative solvers. It is illustrated how algorithmic convergence to computed solutions with required accuracy is mathematically quantified and used within computer aided reachability analysis tools to formally verify convergence over predefined sets of multiple problem data. In addition, some numerical analysis results are used to form computational reliability monitors to escort the algorithm on-line and monitor the numerical performance, accuracy and stability of the entire computational process. For making the paper self-contained, formal verification preliminaries and background on tools and approaches are reported together with the detailed numerical analysis in basic mathematical language. For demonstration purposes, a custom made reachability analysis program based on affine arithmetic is applied to numerical examples.

1. Introduction. Computational algorithms are fundamental components in many modern systems. In particular, iterative matrix numerical algorithms is an established area of research in multiple domains and research efforts have been invested by control and applied mathematicians [51], [52], as well as embedded systems engineers [37]. At present there is an increased research interest by systems, control [45], [44] and formal verification [14], [30] communities, regarding the rigorous verification of system and control algorithms. One motive for the above research is driven by the needs to provide operational assurance and safe algorithmic execution on software and hardware where matrix based system and control numerical algorithms are used.

2010 Mathematics Subject Classification. Primary: 65F20, 65F30; Secondary: 68Q45.
Key words and phrases. Matrix iterative algorithms, Schulz algorithm, Reachability analysis, Formal verification.

This work was supported by the Irish Development Agency (IDA) under the program “Network of Excellence in Aerospace Cyber-Physical Systems”, 2015-2019.
* Corresponding author: Vassilios A. Tsachouridis (tsachov@utrc.utc.com).
** Also with Aalto University, Finland.
Driven by the above motivation, this paper presents the development of a numerical analysis framework together with a tool aided reachability approach for verifying the Schulz generalized matrix inversion algorithm defined in (1), (2).

$$X_{k+1} = 2X_k - X_kAX_k$$  \hspace{1cm} (1)

$$X_0 = aA^T$$  \hspace{1cm} (2)

where $A$ is a non-zero ($A \neq 0$) $n \times p$ matrix with $n \leq p$, $k = 0, 1, 2, \ldots$ and $0 < a < \frac{2}{\|AA^T\|_1}$. It has to be noted that amongst various generalized matrix inverses, (1), (2) compute the Moore-Penrose pseudo-inverse. The case $n > p$ can be treated similarly by considering the transpose $A^T$ instead of $A$ in (1), (2), and hence computing the transpose of the generalized inverse. This is done to ensure computations with lowest possible matrix dimensions.

Generalized matrix inversion computational problems are the core routines in multiple numerical algorithms. Applications are in multiple domains ranging from numerical linear algebra and scientific computational problems to general systems design, optimization and control [40], [19]. Focusing on deterministic ([8], [72], [73]) rather then randomised (e.g. [76]) generalised matrix inversion algorithms, several methods similar to (1) exist. One example is the Neumann-type series approach which uses iterative hyperpower matrix series to compute generalised matrix inverses in complex domains [18]. Another example is the finite-time convergent Zhang neural network model approach [59] for computing the Drazin inverse of time varying matrices, where a defined ODE neural network can be computed through iterative ODE algorithms. Algorithm (1) is effectively used as a paradigm to illustrate how the proposed verification processes can be applied to deterministic matrix iterative algorithms in general. Using notions from numerical analysis, results on the numerical stability and convergence rate of Schulz algorithm (1) and similar matrix flow solvers in general can be derived. Such an analysis can establish rigorous upper bound estimates on the number of algorithmic iterations needed to compute solutions with a desired accuracy. At this point it is worth mentioning that existing analytical results for single data numerical problems have been already established in the literature (e.g. [8], [12]). Embedding algorithms similar to (1) on real time safe critical applications (particularly in aerospace) would require certification of the algorithms as they constitute lower level system components. Therefore, knowing a priori the required number of algorithmic iterations for specified sets of numerical problem data, under predefined numerical accuracy conditions, is crucial for verifying algorithm (1) and aid the certification process. Finally, the above knowledge can be proven very useful when sizing software and hardware resources accordingly during model based design and product development, as algorithmic iterations and the associated computational operations determine physical computational time and required power.

The convergence bounds formulae derived from mathematical analysis of algorithm (1) are analytical functions of quantities (e.g. eigenvalues of matrices) that are not necessarily well defined in the strict mathematical sense within generalised arithmetic operators (e.g. interval, affine arithmetic). For example, in affine or interval computer arithmetic used in several reachability analysis tools, norms of matrices could be evaluated eventually as negative floating point numbers that fundamentally prevent the computation of logarithms on these quantities. Such barriers with formal verification tools can be lifted, by analysing equivalent expressions of convergence bounds free of logarithms. Another option is to analyse the
dynamical evolution (flow) of the discrete system (1), (2) instead, or to explore the iterative evolution of defined solution errors and their convergence to desired accuracy levels. In general, as it will be shown with numerical examples later, there is no best approach solution for the formal verification of algorithmic convergence, as some approaches can fail where others succeed and vice versa. This inevitably would force the developer to include several alternative approaches, like the ones above, to resolve formal verification tool limitations used for the analysis of algorithms similar to (1). Regardless of the particular verification tool, the numerical analysis of the particular algorithm in study is always essential for deriving metrics for numerical stability, accuracy and performance (e.g. error thresholds to convergence, acceptable solution errors, etc.). This provides mathematical rigour and aids the customisation of verification solutions. In addition, by-products of algorithmic numerical analysis are computational reliability monitors, consisting of certain numerical error metrics and conditions, for the monitoring and computational assessment of the entire algorithmic process.

Arguably the background theory of matrix iterative algorithms in systems engineering, uses fundamental elements from functional analysis and dynamical systems theory (discrete and/or continuous time). For the representation and analysis of discrete and continuous functions and dynamical systems, a series of efforts in the past decade [64], [13], [22], [26], [33], [35] has been devoted using hybrid automata [2]. For example, in order to decide whether a set of unsafe system states is reachable, over approximations, symbolic representations [14], [27] and simulation based verification techniques [21], [5] are being used. Amongst these methods, there are trade-offs between the performance of the particular verification approach and accuracy of computed solutions. For instance, larger sets of initial conditions in system parameters could lead to infeasible computations due to state space explosion [16] of the inspected system. This is more likely to occur for non-linear dynamical systems. On the other hand, faster computations can be implemented under lower numerical precision settings but can generate spurious verification counterexamples [65], with obvious negative consequences.

Furthermore, theorem provers [28], while having demonstrated a wide success in proving automatically upper and lower bounds of real valued expressions up to a user specified precision [48], they do require advanced proof-engineering skills. Their adaptation to model based systems engineering is considered a 'hard-to-understand' technology transition with steep learning curves. Furthermore, their application still highly depends on the trade-offs between precision and efficiency, when developing generalized proof-producing strategies as have been shown in PVS [20]. The above fact strengthens the motivation for the approach presented in this paper, which combines algorithmic convergence mathematical results together with automatic exploration of reachable states using reachability analysis based on affine arithmetic.

A large number of system verification approaches and tools today are not capable of preserving the actual dependencies between the particular problems’ initial conditions and corresponding solutions, due to the over approximated nature of the reachable set representations. For the formal verification of dynamic numerical algorithms in particular this can cause failures to the solvability of the reachability problem, since the convergence and stability of the algorithm is sensitive to the above mentioned dependencies. The above limitation incites the use of grid techniques by implementing many algorithm runs for different fixed problem data, but with questionable coverage over the complete problem definition space. At this
point it should be noted that interval analysis has been used to compose interval based iterative algorithms \cite{47, 46} with conditional convergence of their interval iterative process to reachable sets that contain solutions. Interval algorithms of this kind were used for further improving already computed pseudo-inverse solutions by Schulz algorithm (1) \cite{54}, by representing floating point errors as small uncertainty intervals containing computed solutions. Unfortunately, such interval analysis iterative algorithms, although modified versions of their non-interval counterparts, are by structure mathematically different algorithms. By design they locate intervals containing solutions rather than compute coverage of transients of non-interval algorithms over a predefined range of numerical problem data. To the authors knowledge, there is no proof, that the iterated interval paths generated by interval analysis algorithms like \cite{46}, \cite{54} for given sets of problem initial conditions, will contain all respective transient paths generated by ordinary (non-interval) algorithms with respect to a single problem initial condition selected from the same given initial conditions set. Currently, there are diverse reachability analysis software, with respect to the type of background numerical routines and capabilities. For example, SpaceEx \cite{26} is designed only for linear hybrid systems analysis and uses variable step size numerical integration solvers during the reachability exploration in order to provide conclusiveness of the analysis. However, experience shows that it has limited applicability in cases where the problem initial value ranges exceed certain boundaries \cite{30}. At a similar direction, Flow\textsuperscript{\mbox{\star}} tool \cite{13}, which is based on time ordered Taylor approximations, follows an iterative forward reachability analysis technique combined with tight over approximations for analysing non-linear hybrid systems. In general, fast computations can be achieved in Flow\textsuperscript{\mbox{\star}} and the use of higher order Taylor models can potentially improve accuracy of solutions. Unfortunately, aim for fast computations in Flow\textsuperscript{\mbox{\star}} cannot always assure the preservation of quantity dependencies to a desired degree. In addition, high order Taylor models can cause failures due to memory exhaustion, while on the other hand low order models yield inconclusive results.

Apart from verification approaches, there are also falsification methods which can scale better on large dimensional problems in particular. However, falsification methods are inherently incomplete in the sense that finding always a counter example is not guaranteed. Nevertheless, falsification techniques can be used as last resort in cases where the verification approaches are practically infeasible due to problem size. One tool that features falsification is Breach \cite{21}, a Matlab toolbox that provides a variety of algorithms and heuristics to explore the available state space for detecting property violations. Moreover, Breach offers a simulation based approach to perform extensive testing which, when coupled with sensitivity analysis, can also be used for approximate reachability analysis.

Regardless of the computer tool used for the verification of iterative numerical algorithms, the feasibility of state space exploration can be assured to a certain degree via appropriate set value initial parameters which are analytically proven to guarantee algorithmic convergence and reachability correctness. This encourages the mathematical analysis of the numerical algorithm, in order to derive rigorous conditions and computational criteria to accompany the reachability analysis process. The thesis of this paper is that computer science formal verification approaches should complement the mathematical and numerical analysis of algorithms and vice versa. Combined together they specify the fundamental ingredients for standard work definition, and the development of customised computational tools
towards the certification of deterministic numerical algorithms. It is worth noticing that, computer aided formal verification efforts focusing specifically to the internal dynamics of numerical matrix algorithms are limited in the literature. Exceptions are some isolated studies using theorem provers such as [50] as well as improving already obtained solutions [54]. Furthermore, the majority of research in computational algorithms is focused on the theoretical synthesis of matrix algorithms for computing/locating solutions, rather than the formal analysis and assessment of the internal algorithmic dynamics and computational behaviour over multiple problem data.

In view of the above, the present paper contributes with recent research results on applied reachability analysis for the verification of numerical algorithms and matrix iterative solvers in particular. The impact of contributions of the paper can be foreseen within the area of software certification standards DO178C [63], based on its formal methods supplement DO333 [62], targeting aerospace applications where embedded systems reliability is of primary importance. Furthermore, it is evident that adaptive system control and machine intelligence technologies [60] are emerging in modern cyber-physical system applications. Many of such developments are using advanced matrix iterative algorithms, comparable to Schulz algorithm (1), the operational assurance of which is significant for real time embedded systems applications. The current paper contributes along these areas by proposing ways of verification of convergence, transient response and algorithmic monitoring of matrix iterative solvers using algorithm (1) as example. The paper’s technical contributions in summary are: 1. Derivation of analytic formal convergence criteria of algorithm, as function of the minimum number of iterations required to compute reliable solutions with predefined accuracy. 2. Verification of convergence and computational algorithm transient through reachability analysis. 3. Computer aided evaluation of the proposed verification process with a custom made reachability analysis tool based on affine arithmetic. 4. Numerical analysis of algorithm’s computations at every step of the iteration process for constructing on-line computational reliability monitors for the entire computational process.

The remainder of the paper is organized mainly in two parts: a) the numerical analysis of (1) in sections 1-6 and b) the formal verification approach of (1) in sections 7, 8. Section 2 presents the numerical analysis of algorithm (1) for analytical formulae derivations of the condition number, forward and backward errors and floating point rounding errors for every step of the algorithm’s iterative process. Section 3 presents generalizations of Schulz algorithm 1 as discrete and continuous time ordinary differential equation matrix system, for which the respective numerical analysis is given in the appendix. Section 4 presents the numerical analysis of computed generalized inverses solutions regardless of the particular algorithm used for the obtaining such solutions. In contrast to sections 2, 3, the analysis in section 4 is not directed to the iterative computations at every step, instead it is focused only on the final obtained pseudo-inverse solutions. Such an analysis serves in deriving measures of stopping convergence conditions of any algorithm in general. Next, section 5 briefly outlines options for the synthesis of computational reliability monitors of algorithm (1), for the assessment of the entire iterative process. Section 6 presents an analytical convergence formula of Schulz algorithm (1) in accordance with the numerical criteria derived in previous sections. Section 7 outlines some relevant background and definitions on formal verification, hybrid
automata, and computer arithmetic. The proposed semi-automated formal verification process is described in section 8 and illustrated with numerical examples. Finally, some conclusions and future research directions are reported in section 9.

Throughout the paper standard notation is adopted. In particular, $\otimes$, $\text{vec} (\bullet)$ is a vector formed from the columns of a matrix and $\text{vec}^{-1} (\bullet)$ forms back the original matrix column wise. $\| \bullet \|$ denotes a norm of a matrix (2 or Frobenius). The superscript $\#$ denotes the generalized inverse (Moore-Penrose pseudo-inverse) of a matrix. $I$, 0 are identity, zero matrices of appropriate dimensions and $:\equiv$ means equal by definition.

2. Numerical analysis of the Schulz algorithm. In this section, the numerical analysis of Schulz algorithm is presented in basic mathematical language. As mentioned in the introduction, this analysis is essential in deriving appropriate measures to monitor the computations of the algorithm and derive numerical stability and convergence stopping criteria that will be used as metrics in the formal verification process. Based on the approach followed in [51], [36] equation (1) is perturbed with respect to problem data $\delta A$ and dynamic solution perturbations $\delta X_k$, $\delta X_{k+1}$ at every step of the algorithm’s iterative process, as shown in equation (3) below.

$$X_{k+1} + \delta X_{k+1} = 2 (X_k + \delta X_k) - (X_k + \delta X_k) (A + \delta A) (X_k + \delta X_k) \quad (3)$$

Let the residual

$$R_{k+1} := X_{k+1} - 2X_k + X_kAX_k \quad (4)$$

and assume for the moment that there are no rounding errors in the computations so that (4) is exact zero (i.e. $R_{k+1} = 0$), then it follows from (3), after dropping perturbation terms greater or equal to second order, that

$$\delta X_{k+1} = -X_k\delta AX_k + (2I - X_kA)\delta X_k - \delta X_kAX_k \quad (5)$$

Equation (5) is the first order perturbed equation of (1) and accommodates numerical uncertainty considered only in the form of perturbations in the data and on algorithm’s iterative evolution of solutions, under the assumption of zero rounding errors for all individual numerical operations so that $R_{k+1} = 0$. The individual sensitivities of (5) with respect to perturbations $\delta y \in \{\delta A, \delta X_k\}$ are defined as the respective Frechet derivatives of each of these perturbations [36]. Consequently, the norms of the above sensitivities define the individual absolute condition numbers $C_y^n (X_{k+1})$ of the Schulz algorithm (1) with respect to perturbations $\delta y \in \{\delta A, \delta X_k\}$. One way to calculate the above quantities, is to transform (5) into a vector equation and use 2-norms and F-norms. This approach will require the computations of F-norms, 2-norms of Kronecker products of matrix data in (5), namely $X_k^T \otimes X_k$ and $2I - (AX_k)^T \otimes I - I \otimes (X_kA)$. An alternative approach, not necessarily restrictive to 2 and F-norms, is to use general norm inequalities and define $C_A^n (X_{k+1})$ as upper bounds of norms of the matrix coefficients of $\delta y \in \{\delta A, \delta X_k\}$ in (5). Following the latter approach the individual absolute condition numbers for (1) are defined via formulae (6), (7) shown below.

$$C_A^n (X_{k+1}) := \|X_k\|^2 \quad (6)$$

$$C_{X_k}^n (X_{k+1}) := \|2I - X_kA\| + \|AX_k\| \quad (7)$$
Taking the norms on both sides in (5), in view of (6), (7) results in 
\( \|X_{k+1}\| \leq C_A \|X_k\| + C_X^a \|X_k\| \). Furthermore, assuming that every perturbation \( \delta y \in \{\delta A, \delta X_k\} \) for every \( k \) satisfies \( \|\delta y\| \leq \delta_c \|y\| \), for \( \delta_c > 0 \) sufficiently small, such that \( \|\delta X_{k+1}\| \leq \delta_c (C_A \|X_{k+1}\| |A| + C_X^a \|X_{k+1}\| |X_k|) \), yields the following inequality
\[
\delta_c \|X_{k+1}\| \leq C_A \|X_{k+1}\| + C_X^a \|X_{k+1}\| |X_k| \leq \frac{C_A \|X_{k+1}\| + C_X^a \|X_{k+1}\| |X_k|}{\|X_{k+1}\|}. \]
The relative condition number of the Schulz algorithm is defined as
\[
C^r(X_{k+1}) := \lim_{\delta_c \to 0} \left( \sup \left\{ \frac{\|\delta X_{k+1}\|}{\delta_c \|X_{k+1}\|} : \delta_c > 0, X_{k+1} = 2X_k - X_k AX_k, \|\delta A\| \leq \delta_c \|A\|, \|\delta X_k\| \leq \delta_c \|X_k\| \right\} \right).
\]
which in view of (8) outcomes the following relative condition number estimate
\[
C^r(X_{k+1}) := \frac{C_A \|X_{k+1}\| + C_X^a \|X_{k+1}\| |X_k|}{\|X_{k+1}\|}. \]

Formula (9) evaluates the conditioning of Schulz algorithm (1) dynamics at every discrete step of the iterative process.

The relative forward error of the algorithm (1) is defined as
\[
E^F_k(X_{k+1}) := \frac{\|X_{k+1} - A^\#\|}{\|X_{k+1}\|}, \]
which suggest the definition of a convergence error of (1) as \( \delta X_{k+1} := A^\# - X_{k+1} \). Since \( A^\# \) is not known, from (8) and (9) an upper bound of the relative forward error of the algorithm can be derived instead as it is shown in (10).
\[
E^F_k(X_{k+1}) \leq C^r(X_{k+1}) \delta_c. \]

Sufficiently small relative forward error indicates forward numerical stability of the algorithm computing the solution. In the case of well-conditioned problems, where \( C^r(X_{k+1}) \) is sufficiently close to unity, if \( E^F_k(X_{k+1}) \) is within the range of the machine’s zero precision as \( \delta_c \to 0 \) (i.e., (10) is satisfied), then this implies forward numerical stability of the algorithm [51].

Consider now that the computed \( X_{k+1} \) satisfies the following first order perturbation of (1):
\[
X_{k+1} = 2X_k - X_k AX_k + 2\delta X_k - \delta X_k AX_k - X_k \delta A X_k - X_k A \delta X_k \]

Equation (11) follows after perturbing (1) with \( \delta y \in \{\delta A, \delta X_k\} \) and dropping second or grater order terms. Clearly in this case \( X_{k+1} - 2X_k + X_k AX_k \neq 0 \) and taking into account (4) results in
\[
vec(R_{k+1}) = K_k \eta_k \]
where \( R_{k+1} \) is determined by (4), \( \eta_k \) is
\[
vec(R_{k+1}) = \left[ \begin{array}{c} vec(\delta A) \\ vec(\delta X_k) \end{array} \right],
\]
and
\[
K_k := \left[ -X_k^T \otimes X_k \quad 2I - (AX_k)^T \otimes I - I \otimes (X_k A) \right] \]

Now, the relative backward error of the Schulz algorithm is defined as
\[
E^B_k(X_{k+1}) := \frac{1}{\|X_{k+1}\|} \min_{vec(R_{k+1}) = K_k \eta_k} \left( \|\eta_k\| \right).
\]
It is worth noting that \( \min_{vec(R_{k+1}) = K_k \eta_k} \left( \|\eta_k\| \right) \) defines the absolute backward error, which in view of (12) is the minimum (norm wise data and current step solution) perturbation \( \eta_k \) such that \( X_{k+1} \) is a solution of the perturbed equation \( vec(R_{k+1}) = K_k \eta_k \). If \( K_k^\# \) exists, then \( \eta_k = K_k^\# vec(R_{k+1}) \) is a solution to the above minimization problem (a least squares solution in general). Henceforth, \( E^B_k(X_{k+1}) \) and an upper bound \( E^B_k(X_{k+1}) \) can be determined as shown in (14).
\[ E_B'(X_{k+1}) = \frac{\left\| K_k^\# \text{vec}(R_{k+1}) \right\|_{X_{k+1}}} \leq \frac{\left\| K_k^\# \right\|_{\text{vec}(R_{k+1})}}{\|X_{k+1}\|} := \hat{E}_B'(X_{k+1}) \] (14)

Sufficiently small relative backward error indicates backward numerical stability of the algorithm computing the solution [51].

Based on the previous analysis, and in view of (10) and (14), numerical stability conditions of the Schulz algorithm (1) can be stated as the satisfaction of (15)-(17) below.

\[ E_B'(X_{k+1}) \leq u_r + \varepsilon \quad (15) \]
\[ E_F'(X_{k+1}) \leq (u_r + \varepsilon) C'(X_{k+1}) \quad (16) \]
\[ E_{\text{fl}}(X_{k+1}) := C'(X_{k+1}) \hat{E}_B'(X_{k+1}) \leq (u_r + \varepsilon) C'(X_{k+1}) \quad (17) \]

In (15)-(17), \( u_r \) is the round-off unit (i.e. the minimum number that can be added to 1 without any effect), \( \varepsilon \geq 0 \) is a sufficiently small error tolerance. Finally \( E_{\text{fl}}(X_{k+1}) \) in (17) defines an estimate of the relative forward error. It is worth remarking as well that in general, backward stability always implies forward stability but not vice versa [51].

To include in the above analysis computational rounding errors, the following standard floating point model for basic arithmetic operators is used [74]: \( fl(x \ast y) := (x \ast y) (1 + \delta), \delta \in \mathbb{R}, |\delta| \leq u_r, \ast = +, -, \times, \div \), where \( \ast \) denotes one of the basic four operators (addition, subtraction, multiplication, division) and \( u_r \) is the round-off unit as described in (15)-(17). \( fl(\cdot) \) is an arithmetic expression denoting the computed value of \( x \ast y \) where it is assumed to be as good as the rounded exact answer, in the sense that \( |\delta| \leq u_r \) with \( \delta := \frac{fl(x \ast y) - (x \ast y)}{(x \ast y)} \). Obviously, \( |\delta| \) above defines the relative rounding error and the ideal case when the computed answer is actually equal to the exact answer corresponds to \( \delta = 0 \). Generalizing the above scalar operators to matrices is straightforward, as matrix addition and multiplication consist of a number of elementary scalar operators. Now the effects of cumulative rounding errors in algorithm (1) can be quantified as an overall error estimate on the residual (4) of the Schulz algorithm, as Lemma 2.1 shows below.

**Lemma 2.1. (Rounding Error Bound):** For the numerical computations of residual (4), the following rounding error bound holds

\[ \|fl(R_{k+1}) - R_{k+1}\|_F \leq \sigma_n \|X_{k+1}\|_F + 2\sigma_n \|X_k\|_F + \sigma_n \|AX_k\|_F \|X_k\|_F + \sigma_p \|X_k\|_F^2 \|A\|_F + \sigma_n \|X_kA\|_F \|X_k\|_F := r_{k+1} \quad (18) \]

where \( \sigma_\nu := \frac{\nu u_r}{1 - \nu u_r}, 1 > \nu u_r \) for \( \nu = n, p \).

**Proof.** Let (4) perturbed as

\[ R_{k+1} (1 + \delta) = X_{k+1} (1 + \delta) - 2X_k (1 + \delta) + X_k (1 + \delta) A (1 + \delta) X_k (1 + \delta) \], where \( |\delta| \leq u_r \). The proof follows applying Lemmata 1 and 2 of [70] and using standard norm inequalities, since \( u_r < \sigma_\nu \) for \( \nu = n, p \).

In view of (18), the effects of rounding errors in computations can be housed into the relative backward error (14) as a function of residual (4). Therefore (14), in addition to numerical uncertainty, due to data and solution perturbations, can be
extended to accommodate rounding errors as shown in (19).

\[ E^*_B(X_{k+1}) := \frac{\|K^#\|}{\|X_{k+1}\|} (\|R_{k+1}\|_F + r_{k+1}) = E^*_B(X_{k+1}) + \frac{\|K^#\|}{\|X_{k+1}\|} r_{k+1} \]  

(19)

In this case (15) and (17) can be rewritten as (20) and (21) respectively.

\[ E^*_B(X_{k+1}) \leq u_r + \varepsilon \]  

(20)

\[ E^*_F(X_{k+1}) := C^*(X_{k+1}) E^*_B(X_{k+1}) \leq (u_r + \varepsilon) C^*(X_{k+1}) \]  

(21)

3. Extensions to more general discrete and continuous matrix flows. Consider now the more general than algorithm (1) iterative system

\[ X_{k+1} = (1 + t_s) X_k - t_s X_k A X_k \]  

(22)

where \(0 < t_s \leq 1\), \(t_s \in \mathbb{R}\), \(X_0 = t_s A^T\) and as before \(A\) is a \(n \times p\) matrix with \(n \leq p\). It follows from [53] that under \(X_0 = t_s A^T\) if \(\max_{1 \leq i \leq n} \left| 1 - t_s \lambda_i (AA^T) \right| < 1\) then (22) is convergent. One suitable choice that satisfies the last sufficient condition is \(t_s < \frac{1}{\|AA^T\|_1}\), and therefore suitable initial conditions for any \(t_s\) are given by

\[ X_0 = \frac{a}{\|AA^T\|_1} A^T \]  

(23)

It is worth noticing that if \(t_s = 1\), (22) collapses to the Schulz algorithm (1) for which convergence is guaranteed with \(X_0 = \frac{1}{\|AA^T\|_1} A^T\) [7]. One fundamental difference between (1) and (22) is that the convergence rate in Schulz algorithm (1) is quadratic and faster than the linear convergence rate of (22) [53]. Now, let \(X_t := X(t)\) be a continuous function and define the discrete function \(X_k := X_{kt_s}\) consisting of values of \(X_t\) sampled at times \(t = kt_s\). By definition it is \(X_t = \lim_{t_s \to 0} (X_{kt_s})\) and \(X_t := \frac{dX_t}{dt} = \lim_{t_s \to 0} \frac{1}{t_s} (X_{(k+1)t_s} - X_{kt_s}) = \lim_{t_s \to 0} \frac{1}{t_s} (X_{k+1} - X_k)\). Since now \(\frac{1}{t_s} (X_{k+1} - X_k) = X_k - X_k A X_k\) is equivalent to (22), the following initial value problem is derived

\[ \dot{X}_t = X_t - X_t A X_t \]  

(24)

where \(t \geq 0\) and \(X_0\) as in (23). Based on the above derivations, Lemma 3.1 is established for the continuous time initial value problem (24) subject to (23).

**Lemma 3.1.** Matrix differential equation (24) defines an asymptotically stable motion from the initial condition (23) to \(A^#\).

**Proof.** The solution \(X_t\) of (24) with \(X_0\) as in (23) can be represented by that of (22) as \(t_s \to 0\). Since (22) with \(0 < t_s < \frac{1}{\|AA^T\|_1}\) in (23), asymptotically converges with linear rate to \(A^#\), it will converge as well with such a rate as \(t_s \to 0\). This implies a linear rate asymptotic convergence for the evolution \(X_t\) of (24). \(\square\)

Equation (24) defines a non-linear (quadratic) continuous time dynamical system, the equilibrium, say \(X\), of which satisfies \(\dot{X}_t \bigg|_{X_t = X} = 0\) or equivalently \(X = X^* A X\), i.e. a Moore-Penrose pseudo-inverse condition [8]. Hence, (24) constitutes a continuous time matrix flow, on quadratic manifold \(X = X^* A X\), for computing the generalized inverse of a matrix, while (22) is a discrete matrix flow which can be viewed as first order discretization of (24). Conducting a numerical analysis for (22), similar to section 2, leads to the results shown in the appendix of this paper.
4. Numerical analysis of generalised inverse solutions irrespective of the used algorithm. The Schulz algorithm (1) and its generalizations in view of (22) and (24), compute the same steady state pseudo-inverse solution $A^\#$ as $X$ that satisfies the Moore-Penrose condition

$$X - XAX = 0$$

Applying to (25) a numerical analysis similar to section 2, the produced results are independent of any numerical algorithm used to compute the above pseudo-inverse solution. For completeness purposes, this analysis is described next, although state-of-the-art results for pseudo-inverses already exist in [67]. The first order perturbation of (25) with respect to the data and solution perturbations $\delta A, \delta X$ is

$$vec(\delta X) = \left( I - (AX)^T \otimes I - I \otimes (XA) \right)^\# (X^T \otimes X) vec(\delta A).$$

Assuming $\|\delta A\| \leq \delta_c \|A\|, \|\delta X\| \leq \delta_c \|X\|$, for $\delta_c > 0$ sufficiently small, and considering F and 2 norms it follows after some algebra that

$$\frac{\|\delta X\|_F}{\delta_c X\|_F} \leq \left\| \left( I - (AX)^T \otimes I - I \otimes (XA) \right)^\# \right\|_2 \|X\|_F \|A\|_F$$

The relative condition number of (25) is defined as

$$C^r(X) := \lim_{\delta_c \to 0} (\sup_{\|\delta X\| : \delta_c > 0, X = X - XAX},$$

$$\|\delta A\| \leq \delta_c \|A\|, \|\delta X\| \leq \delta_c \|X\|))$$

and in view of (26) it follows that

$$C^r(X) := \left\| \left( I - (AX)^T \otimes I - I \otimes (XA) \right)^\# \right\|_2 \|X\|_F \|A\|_F$$

With respect to 1, 2 and F norms the condition number of $A$ is defined as $C(A) := \|A^\#\| \|A\|$ [67]. Assuming that $A \neq 0$, the non-zero eigenvalues of $AA^\#$ and $A^\#A$ are equal to 1. Furthermore, since for any square matrices $K$ and $N$ of compatible dimensions, the eigenvalues of $K \otimes I + I \otimes N$ are equal to the eigenvalues of $K$ plus the eigenvalues of $N$, it follows that the non-zero eigenvalues of $I - (AA^\#)^T \otimes I - I \otimes (A^\#A)$ can be only 1 and/or -1 and the same hold for the inverse of that matrix (equal to generalized inverse of a full rank square matrix). Thus,

$$\left\| \left( I - (AA^\#)^T \otimes I - I \otimes (A^\#A) \right)^\# \right\|_2 = 1.$$ It is apparent that the algorithms (1), (22), and (24) can be used to compute the condition number of a matrix $A$ as $C(A) := \|X\| \|A\|$. If the computed solution is exact (i.e. $X = A^\#$), then based on the above, it follows that $C^r(X) = C(A)$ which conforms to the theoretical result $C(A) = C(A^\#)$ [67]. In view of (1), (22), and (24) it holds asymptotically that

$$\lim_{k \to \infty} C^r(X_k) = C(A).$$

The relative forward error of (25) is defined as $E^r_F(X) := \frac{\|X - A^\#\|}{\|X\|}$ with respect to which (10) holds alike. Consider now that a computed solution $X$ instead of (25) satisfies $X = XAX + X\delta AX$ with respect to data perturbation $\delta A$. In this case the residual is

$$R := X - XAX \neq 0$$

and can be equivalently rewritten as the vector equation

$$vec(R) = (X^T \otimes X) vec(\delta A).$$

Hence, the relative backward error is defined as
FORMAL ANALYSIS OF SCHULZ MATRIX ALGORITHM

\[ E_B'(X) := \frac{1}{\|X\|} \min_{\text{vec}(R) = (X^T \otimes X) \text{vec}(\delta A)} \left( \|\text{vec}(\delta A)\| \right) \] and can be determined similarly to (14) by (29)

\[ E_B'(X) := \frac{1}{\|X\|} \left\| \left( X^T \otimes X \right)^\# \text{vec}(R) \right\| \leq \frac{1}{\|X\|} \left\| \left( X^T \otimes X \right)^\# \|\text{vec}(R)\| : = \tilde{E}_B'(X) \] (29)

where \( \tilde{E}_B'(X) \), is an upper bound of \( E_B'(X) \). Numerical stability conditions focusing on the solution of (25), irrespective of the used algorithm, can be stated in the same way as formulae (15)-(17), after replacing \( X_{k+1} \) with \( X \) and considering (25), (27) and (29). Finally, Lemma 4.1 can be proven as follows.

**Lemma 4.1.** (Rounding Error Bound Irrespective of Algorithm); For the numerical computations of residual (28), the following rounding error bound holds

\[ \|fl(R) - R\|_F \leq \sigma_n \|X\|_F + \sigma_n \|AX\|_F + \sigma_p \|X\|^2\_F + \sigma_n \|AX\|_F \|X\|_F : = r \] (30)

where \( \sigma_v := \frac{u_v}{1-\nu u_r} \), for \( \nu = n, p \).

**Proof.** Let (28) be perturbed as \( R(1 + \delta) = X(1 + \delta) - X(1 + \delta)A(1 + \delta)X(1 + \delta) \), where \( |\delta| \leq u_r \). The proof follows by applying Lemmata 1 and 2 of [70] and using standard norm inequalities, since \( u_r < \sigma_v \) for \( \nu = n, p \).

Finally, the relative backward error (29) can presumably accommodate accumulative rounding error effects in computations (generated regardless of the used algorithm) as a function of residual (28). Hence, as in (19) in section 2, (31) is defined bellow.

\[ \tilde{E}_B'(X) := \left\| \left( X^T \otimes X \right)^\# \left\| R \right\|_F + r \right\| \frac{\|X\|}{\|X\|} = \tilde{E}_B'(X) + \left\| \left( X^T \otimes X \right)^\# \right\| \frac{r}{\|X\|} \] (31)

Moreover (20) and (21) can be imposed the same way after replacing \( X_{k+1} \) with \( X \) and considering (25), (27), and (31).

5. Computational reliability monitors. The main purpose of a computational reliability monitor is to monitor and assess the numerical stability and accuracy of computations at every step of the algorithm. For that cause, numerical metrics such as estimates of condition numbers, backward/forward errors and numerical stability conditions can be used to measure with some rigour the computational behaviour and convergence. In general, different options can be considered for the Schulz algorithm (1) and its variations (22), depending on the selection of the chosen numerical metrics in sections presented previously. As an example, a potential monitor for Schulz algorithm (1) could consist of: the relative condition number (9); the backward errors (14) and/or (19) (if rounding errors are considered); estimates of relative forward errors (17) and/or (21) (if rounding errors are considered); numerical stability conditions (15) and/or (20) (if rounding errors are considered); stopping convergence criteria for the algorithm that can be derived from a numerical analysis in section 4, irrespective of the particular used algorithm. For the case of algorithm (1), assuming that convergence is achieved at a certain step where \( X_k = X \) and \( X_{k+1} = A^\#, (16) \) can be considered with respect to condition number
$C^r(X) = C(A) = \|X\|\|A\|$. Therefore, a plausible stopping criterion of the iterative process can be written as $\|X_{k+1} - X_k\| < (u_r + \varepsilon) \|A\|\|X_k\|^2$ or equivalently

$$\log_{10} \left( \frac{\|X_{k+1} - X_k\|}{(u_r + \varepsilon) \|A\|\|X_k\|^2} \right) < 0 \quad (32)$$

where $\varepsilon > 0$ is a user defined tolerance offset to $u_r$. For the above case it is worth mentioning that (17), (21) require the expensive computation of $K^#_k$ in (13). However, in cases where the direct computation of $\|K^#\|_2$ needs to be avoided, estimates can be computed instead using state-of-the-art routines [39], [42].

6. Convergence bounds of the Schulz algorithm. Based on results proven in [7] the following theorem can be straightforwardly established on the convergence of Schulz algorithm.

**Theorem 6.1.** Let (1), (2) with $n \leq p$ and $0 < a < \frac{2}{\|AA^T\|_1}$. Then (1) converges to a computed solution $X_k = X$ satisfying (16), in no more than in $\text{Ceil}(\hat{k})$ steps, where

$$\hat{k} = \frac{1}{\ln(2)} \ln \left( \frac{1}{\ln(1 - a\lambda_{\min}^+(AA^T))} \ln \left( \frac{\lambda_{\min}^+(AA^T) (u_r + \varepsilon) C^r(X) \|X\|_F}{\sqrt{\lambda_{\max}(AA^T)}} \right) \right)$$

(33)

$\lambda_{\max}(AA^T)$ and $\lambda_{\min}^+(AA^T)$ denote the maximum and the minimum non-zero eigenvalues of $AA^T$ respectively, $C^r(X)$ is defined by (27), and $\text{Ceil}(\hat{k})$ denotes the minimum integer greater than or equal to $\hat{k}$.

**Proof.** Under the assumption $A \neq 0$, it follows from Theorem 4 of [7] that

$$\|X_k - A^#\|_2 \leq \frac{\sqrt{\lambda_{\max}(AA^T)(1 - a\lambda_{\min}^+(AA^T))}^{2k}}{\lambda_{\min}^+(AA^T)}$$

if $0 < a < \frac{2}{\|AA^T\|_1}$. Since $X_k = X$, it follows from forward error definition that $\|X_k - A^#\| = E^r_F(X) \|X\|$ and thus for (16) to hold it is sufficient that $\|X_k - A^#\|_2 \leq \frac{\sqrt{\lambda_{\max}(AA^T)(1 - a\lambda_{\min}^+(AA^T))}^{2k}}{\lambda_{\min}^+(AA^T)} \leq (u_r + \varepsilon) C^r(X) \|X\|_F$. Solving the last inequality with respect to $k$ yields (33). Finally, because by structure $\hat{k}$ is in general a real positive number (not necessarily integer), when it refers to number of steps $k$, it is rounded up to the next integer value $\text{Ceil}(\hat{k})$ ($k \leq \hat{k} \leq \text{Ceil}(\hat{k})$).

**Remark 1.** In practical applications $(u_r + \varepsilon) C^r(X) \|X\|_F$ in (33) is replaced either as a prior desired accuracy tolerance $\xi > 0$ or as $(u_r + \varepsilon) \|A\|_F \|X\|^2_F$ since $X$ is desired close to $A^#$ and therefore $C(A) = \|A\|_F \|X\|_F$. Hence a convergence expression satisfying bound (33) can be written as

$$\frac{\sqrt{\lambda_{\max}(AA^T)(1 - a\lambda_{\min}^+(AA^T))}^{2k}}{\lambda_{\min}^+(AA^T)} - (u_r + \varepsilon) \|A\|_F \|X\|^2_F \leq 0 \quad (34)$$

Moreover, when considering accuracy to machine precision $\varepsilon = 0$ and the computed solution is exact, i.e. $X = A^#$, it follows that $(u_r + \varepsilon) C^r(X) \|X\|_F = u_r \|A\|_F \|X\|^2_F = C(A) \|A^#\|_F$. Furthermore, because (33), (34) are functions of minimum and maximum eigenvalues of a matrix, it is not obvious how it can be further relaxed in such a way that the eigenvalue computations could be avoided, e.g. using non pessimistic upper and lower eigenvalue bounds similar to [75].
In order to find a fixed global initial condition for (1), (22) and (24) over a predefined compact interval of matrix data Lemma 6.2 is proven next.

**Lemma 6.2.** Let $A, A_{\min}, A_{\max}$ non-zero $n \times p$ matrices, where $n \leq p$, such that the elements of $A$ are greater than or equal to the corresponding elements of $A_{\min}$ and less than or equal to the corresponding elements of $A_{\max}$, i.e. $A \in [A_{\min}, A_{\max}] - \{0\}$. Then for any $A \in [A_{\min}, A_{\max}] - \{0\}$, Theorem 6.1 hold with $0 < a < \frac{2}{\|AA^T\|_1}$, where $\bar{A} := \max(|A_{\min}|, |A_{\max}|)$.

**Proof.** By the definition of $\bar{A}$, it follows that $\frac{2}{\|AA^T\|_1} \leq \frac{2}{\|AA^T\|_2}$ for any $A \in [A_{\min}, A_{\max}] - \{0\}$. Therefore, $0 < a < \frac{2}{\|AA^T\|_1}$ implies $0 < a < \frac{2}{\|\bar{A}^T\bar{A}\|_1}$ so that Theorem 6.1 hold.

Lemma 6.2 could be proven useful when computer aided reachability software is used for the verification of (1), (22). For every different $A \in [A_{\min}, A_{\max}] - \{0\}$, $\alpha$ has to be calculated appropriately to guarantee convergence, which requires the computation of 1-norm. It is possible that the reachability analysis software/tool does not support computations of 1-norms, or such computations are not well defined in the strict mathematical sense, as negative reals can be assigned to norm intervals due to approximations. In such cases and in order to avoid erratic computations that could lead to numerical failures and non-conclusive reachability analysis, $\alpha$ can be determined off-line and hence fixed as global constant satisfying $0 < a < \frac{2}{\|\bar{A}^T\bar{A}\|_1}$ for the entire range of matrices $A \in [A_{\min}, A_{\max}] - \{0\}$.

7. Formal verification preliminaries. Formal verification is a method of proving system correctness with respect to a set of requirements. Contrary to testing and simulation, which can only discover the presence of bugs, formal verification can guarantee their absence. There are several ways to conduct formal verification, which vary in the classes of systems they can handle and the amount of manual work required. For example, model checking [17] is a completely automatic method, in which a model of a system as well as formalized requirements are provided as inputs to specialized software called a model checker. The model checker in turn either confirms that the system satisfies the requirements or provides a counterexample behaviour illustrating it doesn’t. Other examples are the so called deductive verification approaches, such as theorem proving [56], which typically require a significant amount of manual effort but are able to prove properties about infinite state systems that model checking approaches cannot handle.

7.1. Hybrid Automata. Formal verification of hybrid systems, in particular, is an active research field where some important results have already been established and approaches are continuously being proposed. A popular formalism for modelling a hybrid system is the so called hybrid automaton [32]. A hybrid automaton is basically a finite state machine equipped with continuous variables. The states of the machine, called locations, represent different modes of the system, and transitions from one location to the other are represented by edges annotated with transition guards (Boolean expressions that must hold for the transition to be taken) and variable updates. Each location is associated with a set of differential equations that describe the behaviour of the system in that mode, as well as an invariant (Boolean expression that must hold while the system is in the respective mode).
Formally, a hybrid automaton is defined as a tuple \((\text{Loc}, \text{Edge}, \text{Event}, X, \text{Init}, \text{Inv}, \text{Flow}, \text{Jump})\) as follows.

\(\text{Loc}\) is the set of locations (modes) of the automaton, \(\text{Event}\) is a set of input events and \(\text{Edge} \subseteq \text{Loc} \times \text{Event} \times \text{Loc}\) is a set of annotated edges that represent mode transitions of the system.

\(X\) is the set of continuous, real valued variables of the system. We use \(\dot{X}\) to denote the set of first order derivatives of the variables in \(X\). In the context of a mode change, we use \(X'\) to denote the updated variables in the target location.

\(\text{Init}\), \(\text{Inv}\) and \(\text{Flow}\) are functions that assign predicates to each mode (location).

\(\text{Init}(\text{loc})\) is a predicate over variables in \(X\), which states the initial value range for these variables when the initial mode of the hybrid system is \(\text{loc}\).

\(\text{Inv}(\text{loc})\) is also a predicate over variables in \(X\), which constrains the values of these variables while the system is in mode \(\text{loc}\).

\(\text{Flow}(\text{loc})\) is a predicate over variables in \(X \cup \dot{X}\), and essentially describes the set of differential equations that govern the behaviour of the system while in mode \(\text{loc}\).

Finally, \(\text{Jump}\) is a function that maps an edge in \(\text{Edge}\) to a guard-and-update predicate over variables in \(X \cup X'\), i.e. a predicate that describes both the conditions under which a transition is allowed as well as the effects the transition has on the values of state variables.

Consider for example the hybrid automaton in Figure 1, which models the Schulz matrix inversion algorithm (1), (2) for a problem data matrix range of dimensions \(1 \times 2\). There are \(N + 1\) locations (modes) in the automaton, the last of which represents the main computation loop and the previous ones initialization of the resulting pseudo-inverse. The elements \(A_{11}, A_{12}\) of the original matrix and the elements \(X_{11}, X_{21}\) of its pseudo-inverse \(X = A^\#\), are modelled as state variables of the automaton. Two additional state variables \(c\) and \(v\) are shown. State variable \(c\) is used as a counter to provide the desired ordering in the mode transitions.

**Figure 1.** Hybrid automaton for a \(1 \times 2\) matrix \(A\).
Specifically, the value of \( c \) is initially 0 and is always increasing. Whenever it reaches a value of 1, the enabled mode transition happens and the value of \( c \) is reset to 0. State variable \( v \) is similarly used in the main computation loop in order to keep track of the number of iterations needed for convergence.

7.2. Verification and falsification of hybrid automata. A fundamental problem in verification, in the sense that many other problems (like verification of safety properties) can be reduced to it, is the so called reachability problem. In the context of hybrid systems it can be formulated as follows:

“Given a hybrid automaton, a set of initial states and a target state \( S \), decide whether it is possible to reach state \( S \) starting in any of the initial states”.

For the general class of hybrid automata, the reachability problem is undecidable \[31\]. However, over the years, a rich theory around hybrid systems has been built; Several classes of hybrid systems where the problem is tractable have been identified \[2\], \[32\], \[31\], \[38\], \[41\], \[57\], approaches that compute simplified abstractions that are analyzed in place of the original complex system have been developed \[4\], \[3\], \[15\], \[29\], \[61\], and various ways for reachable set representation have been proposed \[2\], \[6\], \[10\], \[66\], \[68\], \[69\]. A problem closely related to the reachability problem is the so called falsification problem. In the context of hybrid systems this can be formulated as follows:

“Given a hybrid automaton, a set of initial states and an unsafe state \( S \), produce a witness trajectory from an initial state to state \( S \), if any such trajectory exists.”

Regardless of the progress on the reachability problem today, and due to the intrinsic difficulty of the former, the problem of falsification has received a great amount of attention over the years as well \[55\], \[9\], \[11\], \[23\], \[34\], \[24\], \[49\]. The idea behind this is to sacrifice completeness for the ability to handle more complex systems in an efficient manner.

7.3. Interval Arithmetic, Affine Arithmetic and Taylor Models. Interval arithmetic \[47\] is a well-known method that can be used to derive error bounds for numerical computations. The idea behind it is quite simple; just replace all numbers involved in operations by intervals, and modify the elementary operations appropriately. For example, if \( x \) belongs in \([1, 2]\) and \( y \) in \([3, 4]\), then \( x + y \) should belong in \([4, 6]\). While straightforward to implement and very efficient, interval arithmetic suffers from the so called dependency problem. To illustrate this, imagine that \( x \) belongs in \([1, 2]\) and we want to evaluate an interval for the expression \( x - x \). It is easy to see that no matter where \( x \) belongs, the answer should be \([0, 0]\), however, interval arithmetic naively returns \([-1, 1]\) instead, as the two instances of \( x \) in the expression are allowed to vary independently.

Affine arithmetic \[25\] is a technique similar to interval arithmetic, but more refined, in the sense that the dependency problem is much less pronounced, as first order correlations between inputs and computed quantities are automatically kept track of. As in interval arithmetic, elementary operations are defined for affine quantities, which allows application of the method to arbitrarily complex expressions built out of these basic operators.

While affine arithmetic is only able to keep track of first order dependencies between inputs and results, Taylor models \[46\], introduced by Berz and Makino, offer a generalization to arbitrary order. Specifically, an n-order Taylor model is a representation of a function, consisting of its n-order Taylor expansion and a remainder interval. In the above case there is a trade-off involved: As the order...
of the Taylor models increases, the dependency problem becomes less of an issue, however, at the same time, computation times increase as well.

8. Computer aided certification of Schulz algorithm. For a computer aided certification of the Schulz algorithm the authors have evaluated several approaches from both a verification (reachability) and a falsification point of view. Regarding verification, the approaches evaluated were interval arithmetic, affine arithmetic, Taylor models and, finally, affine arithmetic coupled with adaptive domain subdivision. For falsification, the approaches bundled with the Breach Matlab toolbox have been evaluated. The goal in all cases was to identify a bound on the number of steps required for algorithm convergence on a given matrix range (i.e. a matrix the elements of which are ranges).

8.1. Verification. This section presents the various approaches tested for computer aided verification of the Schulz algorithm (1), starting from the simpler ones and leading to affine arithmetic with adaptive domain subdivision, which was the one that provided by far the best results. The first approach tested was using interval arithmetic. Specifically, the algorithm (1) was implemented in the programming language Python using the interval arithmetic library PyInterval [58]. Unfortunately, this approach didn’t yield particularly encouraging results. Due to the dependency problem of interval arithmetic mentioned in section 7.3, the bounds derived by interval arithmetic can be overly conservative in general. In our case, this triggers another problem: Schulz’s algorithm (1) is quite sensitive to the choice of initial condition, and while there is a way to obtain an appropriate initial condition $X_0$ according to (2) for any given matrix $A$, when a range of matrices (i.e. a matrix with ranges as elements) is given, a range of initial conditions is generated and the 1-1 correspondence between specific matrices and their respective initial conditions is lost, which causes the algorithm to diverge, even for very small initial matrix ranges. We will refer to this problem as 1-1 correspondence loss, hereafter.

The second approach tested was using affine arithmetic. Here, Schulz’s algorithm (1) was implemented in the programming language C++ and the aaflib [1] affine arithmetic library was employed. As expected, the obtained results were better than the ones obtained using interval arithmetic, since the dependency problem was less pronounced in the second approach. Convergence was able to be verified within specific number of steps for several $2 \times 2$ matrix ranges. However, the dependency problem was not completely eliminated, and when the range of the initial matrix is increased, the same problem as in interval arithmetic was manifested.

The observation that alleviating the dependency problem leads to better results motivated the subsequent Taylor models approach. In particular, Flow* was used [14], as it is a tool for non-linear hybrid system reachability analysis, that uses Taylor models as representations for over approximations of ODE solutions. In order to be able to use Flow* for the purposes of analysis in this paper, it was necessary to first model the Schulz algorithm itself as a hybrid automaton. Since this better fits the nature of the algorithm, a solution with trivial dynamics was opted for where all computations happen during mode changes. Specifically, each matrix element was modelled as a state variable, the main loop of the algorithm was modelled as a system mode with a transition to itself, and the computation of the initial condition was modelled using a sequence of modes linked with transitions eventually leading to the main computation mode, as explained in section 7.1 and schematically outlined in Figure 1. One of the challenges faced during this encoding was the absence
of support for inversion of a state variable as part of an update during a mode change. This was necessary in the computation of the initial condition $X_0$ from the given input matrix $A$. To overcome this limitation, the above inversion was implemented using Schulz algorithm (1) for a scalar (i.e. $1 \times 1$ matrix), the initial condition of which was computed by a simple iterative approach. This led to an additional set of modes and transitions added to the initialization phase before reaching the main computation mode. Compared to the affine arithmetic approach, Flow* generally required more time to yield results, and the use of high order Taylor models in the computations did not appear to make much difference with respect to the dependency problem. These observations, coupled with the fact that modelling the Schulz algorithm as a hybrid automaton in the language of Flow* was quite challenging, directed towards trying an alternative approach described next.

A way to alleviate the dependency problem, regardless of the underlying representation of the computed quantities (intervals, affine forms, Taylor models), is domain subdivision. To illustrate this, consider the following example in the context of interval arithmetic:

$x \in [-1, 1] \rightarrow (x - x) \in [-2, 2]$

When $x$ belongs in $[-1, 1]$, in interval arithmetic $x - x$ is computed as $[-2, 2]$. However, if the domain of is split in two, the computation is applied in each resulted sub-domain and then the union of the results is taken, a better approximation is obtained for the final result shown next.

$x_1 \in [-1, 0] \rightarrow (x_1 - x_1) \in [-1, 1]$

$x_2 \in [0, 1] \rightarrow (x_2 - x_2) \in [-1, 1]$

$(x_1 - x_1) \cup (x_2 - x_2) = [-1, 1]$

Further subdivisions yield even better results. Even though the example above was specific to interval arithmetic, the same principle applies in affine arithmetic too, which led to the last approach tested for computer aided verification of the Schulz algorithm (1), namely: affine arithmetic coupled with adaptive domain subdivision. A high level description of this downselected approach has as follows.

ADAPTIVE DOMAIN SUBDIVISION

1. Get a matrix range as input
2. Run the Schulz algorithm
3. If the algorithm doesn’t converge
   (a) Choose one of the matrix elements and split it in two.
   (b) Create two matrices based on the element that was split.
   (c) Apply the same process recursively on the two matrices starting at step 1.
4. If the algorithm converges
   (a) Store results for that particular matrix range.
   (b) If there are more matrix ranges left to compute, continue from step 1.
   (c) Otherwise, gather results and terminate.

At this point it should be clarified what we mean by convergence and divergence of the algorithm (1) in the affine arithmetic computation context described above, as there are differences with respect to the ordinary floating point case. In the latter, the algorithm is considered to converge if after a number of iterations the value of an expression, which we refer to as the convergence expression, drops below a predefined threshold. Moreover, in the floating point case, once this happens, the value of the
convergence expression never rises above that threshold if iterations continue. Also, given that the initial condition is chosen appropriately, the computation never diverges in the floating point case. In the affine arithmetic context on the other hand, to signal a convergence, the entire interval representing the convergence expression is required to be located below the predefined threshold. If this does not happen within a predefined limit of iterations, say IMAX, divergence is conservatively signalled and then the process proceeds with domain subdivision. Divergence is also triggered if any values involved in the computation become non finite (i.e., INF or NAN), which is possible in the affine arithmetic computation context due to the dependency problem described previously. Contrary to the floating point case, cases have been noticed where the value of the convergence expression drops below the threshold but then slowly rises back up. This behaviour is attributed to rounding errors accumulation and it is not treated as divergence, as long as the entire interval representing the convergence expression drops below the threshold at least once. In the evaluated experiments (32) is used as the convergence expression with zero threshold.

The downselected affine arithmetic approach is demonstrated via numerical examples next. For algorithm (1), (2) and convergence expression (32), the approach is coupled with domain subdivision. For the bound convergence expression (34) domain subdivision is not considered. All computations have taken place on a 2 GHz Intel i5 CPU using double precision \( u_r = 2^{-53} \) and setting an error tolerance \( u_r + \varepsilon = 10^{-10} \) to the formulae.

Example 1; The \( 2 \times 2 \) interval matrix

\[
\begin{bmatrix}
[0.975, 1.025] & [9.75 \times 10^6, 10.25 \times 10^6] \\
[0.975, 1.025] & [0.975, 1.025]
\end{bmatrix}
\]

The results generated are illustrated in Figure 2.

The first four graphs shown in Figure 2 show the resulting element ranges for the pseudoinverse solution evolution according to (1), (2). The last graph of Figure 2 shows the flow of value of the convergence expression (32). When this value drops below zero, convergence is signalled, which is denoted in all figures by a dashed vertical line. In this case, convergence happens after 45 iterations. The computation time taken for the above analysis was 5.34 seconds.

Example 2; The \( 2 \times 3 \) interval matrix

\[
\begin{bmatrix}
[0.975, 1.025] & [9.75 \times 10^6, 10.25 \times 10^6] & 0 \\
0 & [0.975, 1.025] & [-1.025, 0.975]
\end{bmatrix}
\]

The results generated are illustrated in Figure 3.

The first six graphs of Figure 3 show the resulting element ranges for the pseudoinverse solution evolution according to (1), (2). The last graph of Figure 3 shows the flow of value of the convergence expression (32) with zero level threshold, as in Example 1. Here, again, convergence happens after 45 iterations. The required computation time for the analysis took 37.25 seconds, longer than Example 1, as the matrix dimension of the problem is now larger.

Apart from the Schulz algorithm (1), (2) itself, the adopted affine arithmetic approach was used to analyse the upper convergence bound according to Theorem 6.1 by testing (34) as function of iterations. This is demonstrated in Example 3 next.
Example 3; $2 \times 2$ interval matrix

\[
\begin{bmatrix}
0.985 & 1.015 \\
0.985 & 1.015 \\
\end{bmatrix}
\begin{bmatrix}
0.985 & 1.015 \\
0.985 & 1.015 \\
\end{bmatrix}
\]

The results generated are illustrated in Figure 4.

From Figure 4, it is apparent that the analysis of (34) in view of Theorem 6.1, yielded an upper bound of 29 iterations and required 9.27 seconds to compute. It is worth noting that a respective analysis of (34) took more than 60 minutes to run for Example 1 and the process was programmed to stop when exceed such a time limit. Conversely the analysis of (1), (2) and (32) for Example 4 took more than 60 minutes to run and thus the programme run was forced to terminate. This why, these results are not presented herein.
Figure 3. Hybrid automaton for a $2 \times 3$ matrix $A$. 
Example 4: 2 × 3 interval matrix

\[
\begin{bmatrix}
[0.985, 1.015] & [985, 1015] & 0 \\
0 & [0.985, 1.015] & [−1.015, −0.985]
\end{bmatrix}
\]

The results generated are illustrated in Figure 5.

From Figure 5, it is apparent that the analysis of (34) in view of Theorem 6.1, yielded an upper bound of 29 iterations and required 10.47 seconds to compute. The analysis of (34) took more than 60 minutes to run for Example 2 and the process was forced to stop. Conversely the analysis of (1), (2) and (32) for Example 5 took more than 60 minutes to run and thus the programme run was forced to terminate. These results are not presented herein.

As becomes apparent from Examples 1-4, analysing both the Schulz algorithm (1), (2) and the theoretical bound (33) in view of (34), are useful in practice for verifying convergence, as there are cases where the former yields results faster than the latter and vice versa.

One limitation of the proposed approach is that it can become quite memory intensive for large matrices (> 1000 elements), due to the internal book keeping
required by affine arithmetic operations. Another limitation is that, while the approach is sound, it is not complete. Specifically, there are edge cases where recursive domain subdivision does not terminate, as the error terms introduced by affine arithmetic to guarantee soundness of the results are enough, in these particular cases, to cause 1-1 correspondence loss, which triggers divergence, which triggers further subdivision and so on. During experimentation, a correlation between occurrence of the above problem and matrices of very high condition number has been observed. Example 5 below illustrates the above observation.

Example 5: $2 \times 2$ interval matrix $A = \begin{bmatrix} A_{11} & -201 \\ -800 & 401 \end{bmatrix}$, $400 \leq A_{11} \leq 401$.

Figure 6 shows the relationship between the value of $A_{11}$ and the condition number of $A$ in logarithmic scale. When $A_{11}$ takes the value $400.99750623441395$, the determinant of $A$ is very close to zero and its condition number around $2.9e+16$. When, during analysis of either the Schulz algorithm (1), (2) or the theoretical bound (33), (34) for this matrix range, the value of $A_{11}$ approaches $400.99750623441395$, domain subdivision becomes virtually never ending due to the reasons mentioned above.

\[ \text{Figure 6. Variation of condition number of } A \text{ as function of } A_{11}. \]

8.2. **Falsification.** The limitations of the verification approaches mentioned in the previous section motivated to approach the problem from a falsification view point. To this end, Breach [21] was chosen as it supports falsification, providing a variety of algorithms and heuristics for exploring the available state space for property violations detection. The latter functionality is what was used in this paper. In order to be able to do that, Schulz algorithm was modelled in Simulink and the assertion that the algorithm converges in a given number of steps, was encoded in metric interval temporal logic (the formal language Breach uses for property specification). Having modelled the algorithm in Simulink also enables to perform gridded simulation using Breach, which provides an approximated view of the possible element trajectories of (1), (2) for a given initial matrix range. The falsification approach above is illustrated with the following example.

Example 6: $43 \times 68$ matrix, taken from the University of Florida sparse matrix
collection [71] (it is listed in the collection under the id \textit{lpkb2}). It contains 313 nonzero elements, a visualization of which is portrayed in Figure 7.

![43 x 68 Matrix: 313 non zero elements](image)

**Figure 7.** 43 × 68 matrix \textit{lpkb2} University of Florida sparse matrix collection.

In the numerical experiments matrix elements (8,59) and (9,60) were varied within the interval \([0, 2]\) and elements (27,13) and (28, 14) within the interval \([-2, 0]\). The above element variation corresponds to a ±100\% range of their original values. These particular elements were chosen because changing them was found to trigger the biggest changes in the condition number of the matrix itself.

Initially, a coarse gridding of the input matrix range was induced by allowing each element to vary uniformly in its allowed range. Then in order to obtain a basic estimate on the maximum number of iterations needed for convergence, the Schulz algorithm (1), (2) was executed for each resulting matrix above in order to obtain a basic estimate. The above results are presented in Figure 8 where, out of the 636 elements of the pseudo-inverse solution, the evolution of elements (59,8), (60,9), (13,27) and (14,28) is indicatively shown in the first four graphs of the figure. In addition, the last graph Figure 8 shows the evolution of the \textit{convergence expression} (32).

Based on the simulation results above, 50 appears to be a conservative upper bound for steps to convergence. Indeed, when asking \textit{Breach} to check this assumption it does not respond with a counterexample. In fact, it was possible to lower the above bound to 39 without \textit{Breach} complaining. For a value of 38 steps \textit{Breach} returns a counterexample. As in the previous affine arithmetic analysis, a procedure that returns an upper bound (33) without running the Schulz algorithm (1), (2) itself was implemented for Example 6 too. In this case, it was possible to lower the bound to 44 steps without \textit{Breach} responding with a counterexample, which is consistent with the previous result. It has to be reminded, that the analysis results for Example 6 are not sound as it is not proved or disproved that there exist matrices in the given problem data range that will require more than 44 steps for (1),
Nevertheless, since a sound analysis in cases of large matrices is practically infeasible from a computational point of view, using falsification can still be useful to reveal converging behaviour in more systematic manner than endless simulation runs.

8.3. Online computational monitor. To illustrate a computational reliability monitor according to section 5, the original matrix from Example 6 was used. The algorithm (1), (2) was executed for this case in Matlab and the following information was monitored at every step of the computational process: relative condition number (9); backward errors (14) and (19) in view of (13); relative forward error estimates (17) and (21); numerical stability conditions (15) and (20); convergence stopping criterion (32) with threshold $u_r + \varepsilon = 10^{-10}$. In this example, in all formulae, the Frobenius norm is considered apart from the norm of $K^\#$ where the 2-norm is used. Regarding now the estimation of $\|K^\#\|_2$, the normest function of Matlab is used. The above metrics are shown in Figure 9 in logarithmic scale.

From Figure 9 in view of (14) and (17) it is apparent that, numerical stability and required accuracy is maintained for the entire algorithmic process, with respect to user defined convergence threshold $10^{-10}$. In addition, when rounding errors are to be considered in view of (19) and (20), the results indicate violation of numerical stability condition after step 28 of the process. For the last case questions could be raised about the reliability of computations, unless $u_r + \varepsilon$ is relaxed to larger value, e.g. $10^{-8}$. However, one should also consider possible pessimism in the rounding error bound (18) for this particular example, and should confirm this by further analysis, before reaching definite decisions on numerical stability. In general, including rounding error bounds in the analysis always increases conservatism from
the point of view that such bounds are additive to the backward errors instead of being subtracted in an optimistic scenario.

9. Conclusions. This paper has presented an approach of computer aided reachability analysis for formally verifying matrix iterative numerical algorithms. The Schulz general matrix inversion algorithm (1), (2) was used as a representative example to illustrate the proposed method towards building a certification framework of general deterministic matrix flow based solvers. The thesis supported in this paper is that, computer science formal verification approaches should complement the mathematical and numerical analysis of the algorithms in study and vice versa. As a first step for supporting the above proposition, the paper has illustrated how customised numerical analysis and formal verification analysis are combined, to construct a formal verification process for convergence and computational reliability of the algorithm in study.

Taking into account the achieved preliminary results, future work will include the exploration of the following three directions, regarding the affine arithmetic verification framework:

a) Change of the underlying computation technique, from first order affine arithmetic to a higher order approach, such as second order quadratic form arithmetic [43]. While this will generally slightly increase the computational burden per operation in the proposed framework, it is expected to greatly reduce the total number of operations, as the dependency problem would be less pronounced and
the number of domain subdivisions would drop as a consequence. Therefore, a significant net gain in computation time is anticipated.
b) Investigation of various domain subdivision heuristics that decide the order in which the different matrix element intervals should be split when the need arises, as well as where the split point should occur (currently the only option is the interval midpoint). Preliminary experimentation in the current prototype implementation has shown that this can have a tremendous effect in running time (e.g., up to 15 times faster computation in some cases).
c) Since the computational load at the present is highly parallelizable, parallel and distributed versions of the framework should be able to take advantage of multiple processing cores.

Now, a second order perturbation analysis is possible to provide better error estimates as shown in [36]. This kind of analysis constitutes future research worth implementing for algorithms (1) and (22). Finally, regarding only the verification of convergence of (1), more relaxed (pessimistic) bounds than (33) can be provided computationally by analyzing the continuous flow (24). This is apparent from section 3 since (24) in view of its approximation in (22) converges with linear rate slower to (1). Although the evolution of (24) is different than that of (1), analysing (24) in combination with Lemma 6.2, could form a compatible continuous time problem for Flow* in order to verify worst case upper convergence bounds. The above is left for future work.

Acknowledgements. The authors acknowledge the financial support by the Irish Development Agency (IDA) under the program “Network of Excellence in Aerospace Cyber-Physical Systems”, 2015-2019. Finally, the authors would like to thank the editor and the anonymous reviewers for their constructive feedback.

Appendix.

Numerical analysis of algorithm (22). Applying the approach followed in (1) for (22) results in the same formulae with the following modifications.

\[ X_{k+1} + \delta X_{k+1} = (1 + t_s + \delta t_s) (X_k + \delta X_k) \]
\[ - (t_s + \delta t_s) (X_k + \delta X_k) (A + \delta A) (X_k + \delta X_k) \]
\[ R_{k+1} := X_{k+1} - (1 + t_s) X_k + t_s X_k A X_k \]  

(35)

Assuming that (36) is exact zero and neglecting second order perturbation terms, the first order perturbation equation of (22) follows

\[ \delta X_{k+1} = ((1 + t_s) I - t_s X_k A) \delta X_k - t_s \delta X_k A X_k - t_s X_k \delta A X_k + \delta t_s (X_k - X_k A X_k) \]  

(37)

The individual absolute \( C_a (X_{k+1}) \) and relative condition numbers \( C_r (X_{k+1}) \) of (22) with respect to perturbations \( \delta y \in \{ \delta A, \delta X, \delta t_s \} \) are defined next.

\[ C_a (X_{k+1}) := t_s \| X_k \|^2 \]  

(38)

\[ C_{X_k} (X_{k+1}) := \| (1 + t_s) I - t_s X_k A \| + \| t_s A X_k \| \]  

(39)

\[ C_{t_s} (X_{k+1}) := \| X_k - X_k A X_k \| \]  

(40)

Now, \( \| \delta X_{k+1} \| \leq C_a (X_{k+1}) \| \delta A \| + C_{X_k} (X_{k+1}) \| \delta X_k \| + C_{t_s} (X_{k+1}) \| \delta t_s I \| \) and assuming that every \( \delta y \in \{ \delta A, \delta X, \delta t_s \} \) for every \( k \) satisfies \( \| \delta y \| \leq \delta_c \| y \| \), for \( \delta_c > 0 \) sufficiently small, such that
The relative condition number of (22) is defined as the residual (36), the following rounding error bound holds.

\[ \frac{\| \delta X_{k+1} \|}{\delta_c \| X_{k+1} \|} \leq C_{R}^{\alpha} (X_{k+1}) \| A \| + C_{X_k}^{\alpha} (X_{k+1}) \| X_k \| + C_{\delta t_s I}^{\alpha} (X_{k+1}) \| \delta t_s I \| \]

it follows that

\[ \frac{\| \delta X_{k+1} \|}{\delta_c \| X_{k+1} \|} \leq \frac{C_{R}^{\alpha} (X_{k+1}) \| A \| + C_{X_k}^{\alpha} (X_{k+1}) \| X_k \| + C_{\delta t_s I}^{\alpha} (X_{k+1}) \| t_s I \|}{\| X_{k+1} \|} \] (41)

The relative condition number of (22) is defined as

\[ C^R (X_{k+1}) := \lim_{\delta_c \to 0} \left( \sup \left\{ \frac{\| \delta X_{k+1} \|}{\delta_c \| X_{k+1} \|} : \delta_c > 0, X_{k+1} = (1 + t_s) X_k - t_s X_k A X_k, \right. \right. \]
\[ \left. \left. \| \delta A \| \leq \delta_c \| A \|, \| \delta X_k \| \leq \delta_c \| X_k \|, \| \delta t_s I \| \leq \delta_c \| t_s I \| \right\} \right) \]

and in view of (42) it follows for that

\[ C^R (X_{k+1}) := \frac{C_{R}^{\alpha} (X_{k+1}) \| A \| + C_{X_k}^{\alpha} (X_{k+1}) \| X_k \| + C_{\delta t_s I}^{\alpha} (X_{k+1})}{\| X_{k+1} \|} \] (42)

Perturbing (22) with \( \delta y \in \{ \delta A, \delta X_k, \delta t_s \} \), dropping second and higher order perturbation terms and assuming that \( X_{k+1} = (1 + t_s) X_k + t_s X_k A X_k \neq 0 \), results in (12) with respect to (36), \( \eta_k := \begin{bmatrix} \text{vec}(\delta t_s I) \\ \text{vec}(\delta A) \\ \text{vec}(\delta X_k) \end{bmatrix} \) and

\[ K_k := \begin{bmatrix} (X_k - X_k A X_k)^T \otimes I & -t_s (X_k^T \otimes X_k) \\ (1 + t_s) I - (t_s X_k A)^T \otimes I - I \otimes (t_s X_k A) \end{bmatrix} \] (43)

Similarly to Lemma 2.1, Lemma 9.1 is stated below to account for rounding errors.

**Lemma 9.1.** (Rounding Error Bound of (36)); For the numerical computations of residual (36), the following rounding error bound holds

\[ \| f (R_{k+1}) - R_{k+1} \|_F \leq \sigma_n \| X_{k+1} \|_F + \sigma_n \| (1 + t_s) X_k \|_F + \sigma_n \| t_s X_k A \|_F \| X_k \|_F \]
\[ + \sigma_p \| t_s X_k \|^2 \| A \|_F + \sigma_n \| t_s X_k A \|_F \| X_k \|_F + \sigma_p \| X_k - X_k A X_k \|_F := r_{k+1} \] (44)

**Proof.** Let \( |\delta| \leq u_\nu \) and (36) be perturbed as

\[ R_{k+1} (1 + \delta) = X_{k+1} (1 + \delta) - (1 + t_s) (1 + \delta) X_k (1 + \delta) \]
\[ + t_s (1 + \delta) X_k (1 + \delta) A (1 + \delta) X_k (1 + \delta) \]

The proof follows applying Lemmata 1 and 2 of [70] and using standard norm inequalities, since \( u_\nu < \sigma_\nu \) for \( \nu = n, p \). \( \square \)

**REFERENCES**

[1] **AAFLIB - An Affine Arithmetic C++**, 2019. Available from: [http://aaflib.sourceforge.net](http://aaflib.sourceforge.net).

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis and S. Yovine, *The algorithmic analysis of hybrid systems*, *Theoretical Computer Science*, **138** (1995), 3–34.

[3] R. Alur, T. A. Henzinger, G. Lafferriere and G. Pappas, *Discrete abstractions of hybrid systems*, *Proceedings of the IEEE 88*, **7** (2000), 971–984.

[4] R. Alur, T. Dang and F. Ivancic, *Counter example-guided predicate abstraction of hybrid systems*, *Theoretical Computer Science*, **354** (2006), 250–271.

[5] Y. Annappready, C. Liu, G. Faineos and S. Sankaranarayanan, S-TaLiRo: A tool for temporal logic falsification for hybrid systems, *Proc. of Tools and Algorithms for the Construction and Analysis of Systems*, (2011), 254–257.

[6] E. Asarin, T. Dang and O. Maler, The \( \frac{d}{dt} \) tool for verification of hybrid systems, *Int. Conf. on Computer Aided Verification, LNCS*, Springer-Verlag, (2002), 365–350.

[7] A. Ben-Israel and D. Cohen, *On iterative computation of generalized inverses and associated projections*, *SIAM J. Numer. Anal.*, **3** (1966), 410–419.
[8] A. Ben-Israel and T. N. E. Greville, *Generalized Inverses Theory and Applications*, Springer, 2003, ISBN 978-0-387-00293-4.

[9] A. Bhatia and E. Frazzoli, Incremental search methods for reachability analysis of continuous and hybrid systems, *Hybrid Systems: Computation and Control*, LNCS, Springer-Verlag, 2004, 142–156.

[10] O. Botchkarev and S. Tripakis, Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations, *Hybrid Systems: Computation and Control*, LNCS, Springer-Verlag, 2000, 73–78.

[11] S. Branicky, M. M. Curtiss, J. Levine and S. Morgan, Sampling-based planning, control, and verification of hybrid systems, *Control Theory and Applications*, 2006, 575–590.

[12] X. Chen, E. Abraham and S. Sankaranarayanan, Flow* An analyzer for non-linear hybrid systems, *Proc. of CAV13*, LNCS, Springer, 2013, 258–263.

[13] M. Esposito, J. Kim and V. Kumar, Adaptive RRTs for validating hybrid robotic control systems, *Workshop on Algorithmic Foundations of Robotics*, Zeist, Netherlands, 2004, 107–132.

[14] G. Frehse, C. L. Guernic, A. Donzé, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang and O. Maler, SpaceEx Scalable verification of hybrid systems, *Proc. of CAV11*, LNCS, Springer, 2011, 379–395.

[15] F. Immler, Tool presentation Isabelle/hol for reachability analysis of continuous systems, in *ARCH14-15, 1st and 2nd International Workshop on Applied Verification for Continuous and Hybrid Systems* (eds. M. Frehse and M. Althoff), Academic Press, 2011, 35–75.
[34] A. A. A. Julius, G. E. Fainekos, M. Anand, I. Lee and G. J. Pappas, IRobust test generation and coverage for hybrid systems, Hybrid Systems: Computation and Control, LNCS, Springer-Verlag, 4416 (2007), 329–342.

[35] S. Kong, S. Gao and W. Chen, Reachability analysis for hybrid systems, Proc. of TACAS15, LNCS, Springer, 9035 (2015), 200–205.

[36] M. Konstantinov, D. Gu, V. Mehrmann and P. Petkov, Perturbation Theory for Matrix Equations, 2nd edition, Elsevier, Amsterdam, 2003, ISBN-9780444513151.

[37] G. A. Kumar, T. V. Subbareddy, B. M. Redd, N. Raju and V. Elamaran, An approach to design a matrix inversion hardware module using FPGA, Int. Conf. on Control, Instrumentation, Comm. and Comput. Technologies, 230 (2014), 87–90.

[38] G. LaFerriere, G. Pappas and S. Yovine, A new class of decidable hybrid systems, Hybrid Systems: Computation and Control, LNCS, 1569 (1999), 137–151.

[39] LAPACK-Linear Algebra PACKage, 2019. Available from: http://www.netlib.org/lapack/.

[40] W. Levine, The Control Handbook, IEEE Press, 1996.

[41] C. Livadas and N. Lynch, A new class of decidable hybrid systems, Hybrid Systems: Computation and Control, LNCS, 1386 (1998), 253–272.

[42] Matlab-Mathworks, 2019. Available from: https://www.mathworks.com

[43] F. Messine and A. Touhami, A general reliable quadratic form: an extension of affine arithmetic, Reliable Computing, 12 (2006), 171–192.

[44] D. Monniaux, Toward verifiably correct control implementations, The impact of control technology Part 2: Challenges for control research, Report of IEEE Control Systems Society, 2nd Ed, 2011. Available from: http://ieeecas.org/general/IoCT2-report.

[45] D. Monniaux and A. Mine, Verification of control system software, the impact of control technology, Part 1: Success stories for control, Report of IEEE Control Systems Society, 2nd Ed, 2011. Available from: http://ieeecas.org/general/IoCT2-report.

[46] R. E. Moore, Methods and Applications of Interval Analysis, SIAM Studies in Applied and Numerical Mathematics, Philadelphia, 1987, ISBN-10: 0898711614.

[47] R. E. Moore, R. B. Kearfott and M. J. Cloud, Introduction to Interval Analysis, SIAM, 2009.

[48] C. Munoz and D. Lester, Real number calculations and theorem proving, 18th Int. Conf. on Theorem Proving in Higher Order Logics, England, (2005), 239–254.

[49] T. Nahhal and T. Dang, Test coverage for continuous and hybrid systems, Int. Conf. on Computer Aided Verification, LNCS, 4590 (2007), 449–462.

[50] I. Pasca, Formal Verification for Numerical Methods, PhD Thesis in Universit Nice Sophia Antipolis, France, 2010.

[51] P. Petkov, M. Konstantinov and N. Christov, Computational Methods for Linear Control Systems, 2nd edition, Prentice-Hall, Hemel Hempstead, 1991, ISBN-9780444513151.

[52] M. D. Petkovic and P. S. Stanimirovic, Generalized matrix inversion is not harder than matrix multiplication, J. Comput. and Applied Math., 230 (2009), 270–282.

[53] M. D. Petkovic and P. S. Stanimirovic, Iterative method for computing Moore-Penrose inverse based on Penrose equations, J. Comput. and Applied Math., 235 (2011), 1604–1613.

[54] M. S. Petkovic, Iterative methods for bounding the inverse of a matrix (a survey), FILOMAT Nis, Algebra Logic & Discrete Mathematics, 9 (1995), 543–577.

[55] E. Plaku, L. Kavraki and M. Vardi, Hybrid systems: from verification to falsification by combining motion planning and discrete search, Formal Methods in System Design, 34 (2009), 157–182.

[56] A . Platzer and J-D Quesel, KeYmaera a hybrid theorem prover for hybrid systems (system description), Automated Reasoning, IJCAR 2008, Lecture Notes in Computer Science (eds. M. Frehse and M. AlthoffArmando A., Baumgartner P., Dowek G.), Springer Berlin Heidelberg, 5195 (2008), 163–183.

[57] A. Puri, Theory of Hybrid Systems and Discrete Event Systems, PhD Thesis in University of California, Berkeley, 1995.

[58] PyInterval - Interval arithmetic in Python, 2019. Available from: https://github.com/taschini/pyinterval

[59] S. Qiao, X. Wang and Y. Wei, Two finite-time convergent Zhang neural network models for time-varying complex matrix Drazin inverse, Linear Algebra and Its Applications, 542 (2018), 101–117.

[60] Y. Rahman, A. Xie and S. Bernstein, Retrospective cost adaptive control: pole placement, frequency response, and connections with LQG control, IEEE Control System Magazine, (2017), 28–69.
[61] S. Ratschan and Z. She, Safety verification of hybrid systems by constraint propagation based abstraction refinement, ACM Transactions on Embedded Computing Systems, 6 (2007), Article No. 8.
[62] RTCA Formal Methods Supplement to DO-178C and DO-278A, RTCA DO-333, December 13, 2011.
[63] RTCA, Software Considerations in Airborne Systems and Equipment Certification, RTCA DO-178C, December 13, 2011.
[64] S. Schupp, E. Abraham, X. Chen, I. B. Makhlouf, G. Frehse, S. Sankaranarayanan and S. Kowalewski, Current challenges in the verification of hybrid systems, Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems, (2015), 8–24.
[65] S. Schupp and E. Abraham, Efficient dynamic error reduction for hybrid systems reachability analysis, Proc. of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS18, LNCS, Springer, 2018.
[66] B. I. Silva and B. H. Krogh, Formal verification of hybrid systems using CheckMate: A case study, American Control Conference, (2000), 1679–1683.
[67] G. Stewart and J. Sun, Matrix Perturbation Theory, Academic Press, New York, 1990.
[68] O. Stursberg and B. H. Krogh, Efficient representation and computation of reachable sets for hybrid systems, Hybrid Systems: Computation and Control, LNCS, 2623 (2003), 482–497.
[69] C. J. Tomlin, I. Mitchell, A. Bayen and M. Oishi, Computational techniques for the verification and control of hybrid systems, Proceedings of the IEEE 91, Springer-Verlag, 7 (2003), 986–1001.
[70] V. A. Tsachouridis, Numerical analysis of $H_{\infty}$ filter for system parameter identification, Int. J. Modelling, Identification and Control, 30 (2018), 163–183.
[71] University of Florida Sparse Matrix Collection, 2019. Available from: https://www.cise.ufl.edu/research/sparse/matrices/list_by_id.html
[72] G. Wang, Y. Wei and S. Qiao, Generalized Inverses: Theory and Computations, Springer, Singapore: Science Press Beijing, Beijing, 2018.
[73] Y. Wei, P. Stanimirovic and M. Petkovic, Numerical and Symbolic Computations of Generalized Inverses, World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2018.
[74] J. Wilkinson, Rounding Errors in Algebraic Processes, Prentice Hall, Englewood Cliff, NJ, 1963.
[75] H. Wolkovicz and G. P. H. Stayan, Bounds for eigenvalues using traces, Linear Algebra and Its Applications, 29 (1980), 471–506.
[76] P. Xie, H. Xiang and Y. Wei, Randomized algorithms for total least squares problems, Numerical Linear Algebra with Applications, 26 (2019), e2219, Available from: https://doi.org/10.1002/nla.2219

Received January 2019; 1st revision May 2019; Final revision July 2019.

E-mail address: TsachoV@utrc.utc.com
E-mail address: GiantaGE@utrc.utc.com
E-mail address: BasagiS@utrc.utc.com
E-mail address: KouramK@utrc.utc.com