Computability of Matrix Functions and Compiling of Matrix Problems on Quantum Computers

1st Holger Boche
TUM School of Computation, Information and Technology
Technical University of Munich, Germany,
boche@tum.de

2nd Adalbert Fono
Department of Mathematics
Ludwig-Maximilians-Universität München, Germany
fono@math.lmu.de

3rd Gitta Kutyniok
Department of Mathematics
Ludwig-Maximilians-Universität München, Germany
kutyniok@math.lmu.de

***Abstract*—We consider the problem of quantum circuit synthesis formulated as quantum compiler functions, i.e., functions that map unitary matrices to corresponding gate-circuit approximations. The existence of quantum compiler functions that provide low-complexity gate-circuit approximations to arbitrary unitary matrices is well-known. However, being an entirely classical task, the implementation of quantum compiler functions must be considered in the context of digital computing. Digitally computing these functions has recently been shown to be infeasible in terms of Turing computability and computable analysis. Going one step further, we prove that there do not exist computable implementations of compiler functions restricted to certain, relevant subtasks. Specifically, we provide an infeasibility result for compiler functions based on currently known quantum algorithms for estimating features of the solutions of systems of linear equations. A key step in deriving the result is the computability analysis in the Turing framework of certain matrix functions, such as the pseudoinverse and the condition number of a matrix. Hereby, we fully characterize the computability breakdown of these functions and, consequently, also the infeasibility of computable compiler functions in this context.**

For a version of this work including all proofs, we refer to [1]

I. Introduction

Quantum computing promises to harness quantum mechanical phenomena to perform certain computations more efficiently than classical digital machines. Examples of such computational tasks include the factorization of large integers relevant to cryptography and the simulation of protein dynamics in computational chemistry [2]–[4]. Despite tremendous amounts of resources, building quantum computers has proven notoriously challenging.

The arguably best established model of quantum computing, both from theoretical and implementation perspective, is gate-based quantum computing [5]–[9]. In this model, information is processed by successively applying quantum gates, which in their specific order of application form a quantum gate-circuit. When neglecting the measurement on the quantum processor to obtain the final output, the implementation of this process decomposes into two main parts: In the quantum part, the computer successively applies a gate sequence to a state vector, where the sequence members belong to a finite and fixed gate family. The classical part, which is processed by classical digital hardware, controls the application of the gate sequence. This second task, which considers compiling quantum algorithms to gate-circuits, is referred to as circuit synthesis or the compiler problem [10]–[12].

Each available quantum gate corresponds to a unitary operator on some finite-dimensional Hilbert space. As only a countable number of unitary operators can be represented through finite gate circuits, it is necessary to implement general quantum algorithms, i.e., arbitrary unitary operators, in an approximate manner. In other words, gate-circuit quantum computing aims to approximate the action of unitary operators on the Hilbert space up to a desired accuracy of 2^{-n}. If there exists a suitable gate sequence for every unitary operator and every $n \in \mathbb{N}$, the gate family is universal [13]–[15].

Moreover, the tasks a gate-based quantum computer aims to solve need to be formulated so that they are accessible to (classical) digital hardware responsible for compiling the quantum circuits. Since classical computing acts on discrete mathematical objects, it cannot naturally account for the analytic principles of continuity and approximation. Accordingly, it is not immediately evident whether digital hardware is able to logically control the execution of operations such as quantum circuit emulation or quantum-circuit concatenation (on the Hilbert space), i.e., the compiler task, in a mathematically correct manner.

The existence of compiler functions is not the issue; they indeed exist as shown for instance by the Solovay-Kitaev theorem [16], [17]. However, the question of their computable (i.e., provably correct and with error control) implementation on digital hardware is not treated by these results and it turned out that there do not exist computable algorithmic emulations/approximations on digital hardware for universal compiler functions [10]–[12]. If not universally, computable implementations of compiler functions restricted to certain sub-tasks might still be feasible. We study this question for a sub-task of fundamental importance in many fields: solving linear systems of equations.

Quantum algorithms estimating features of the solutions of systems of linear equations have been proposed [18]–[20] and also serve as the basis of quantum machine learning algorithms [21], [22]. Crucially, the algorithms depend on the condition number of the matrix representing the linear system. Thus, a compiler function assuming a matrix as input to generate the sought quantum circuit that estimates the features of the

solution, needs to compute the condition number of the input matrix as a sub-process. In other words, the computation of the condition number must be included in the compiler function executed on (classical) digital hardware as a pre-processing step. However, we show in this work that a computable algorithmic implementation of this compiler function is not feasible on digital hardware. In particular, there does not exist a universal and computable algorithmic implementation of the pre-processing step to compute the condition number on digital hardware. More exactly, we show that there does not exist a computable algorithmic implementation for this specific compiler problem even when fixing the dimension of the input matrices. Therefore, exploiting the potential speed-up of computations via quantum algorithms in this case is not possible based on computable implementations. To potentially avoid this bottleneck, we need quantum algorithms that do not depend on the pre-computation of the condition number.

A. Problem Statement

The condition number of a matrix is defined via its pseudoinverse [23], [24]. The pseudoinverse $A^\dagger \in \mathbb{C}^{n\times m}$ of a matrix $A \in \mathbb{C}^{m\times n}$ is characterized by

$$AA^\dagger A = A, \qquad A^\dagger AA^\dagger = A^\dagger,$$
$$(AA^\dagger)^H = AA^\dagger, \qquad (A^\dagger A)^H = A^\dagger A.$$

These equations uniquely define the pseudoinverse, i.e., exactly one matrix $A^\dagger$ satisfies all four conditions. Closed-form representations of the pseudoinverse are known for universal [25]–[27] as well as specific classes of matrices [28]. However, it is unclear if they can be implemented on digital hardware with error control, i.e., if the actual execution of the involved mathematical operations on digital hardware is feasible in the strict sense formalized in Section II. Simple closed-form expressions of $A^\dagger$ are only known in specific instances. For example, if A has full rank such that its columns are linearly independent, then it is easy to verify that $A^\dagger = (A^HA)^{-1}A^H$.

In practice, approaches to compute the pseudoinverse comprise strategies ranging from rank decomposition, applying the QR method, singular value decomposition [29], and based on Gaussian elimination [30], [31]. However, these approaches again typically lack either theoretically proven error control in the above sense or generality. Consequently, the algorithms provide seemingly correct outputs for any input instance and are indeed provably reliable in specific cases, such as well-conditioned inputs [32], [33], but intrinsic failure modes and their severity are unbeknownst to the user.

Hence, in the general case, an explicit description that allows for a straightforward effective computation is missing, i.e., an algorithm that takes any matrix (via arbitrarily accurate rational approximations) and any error parameter as input and computes an approximation of the associated pseudoinverse up to the given error. Broadening the scope, our core question is whether or under which conditions the pseudoinverse and fundamental problems associated with the pseudoinverse, such as the least squares problem and the condition number of a matrix, can be effectively approximated on digital computing hardware, which currently encompasses virtually any general-purpose computing device and high-performance computer. To that end, we consider the following mappings:

$$f^\dagger : \mathbb{C}^{m\times n} \to \mathbb{C}^{n\times m}, A \mapsto A^\dagger \tag{I}$$
$$f^{\|\dagger\|} : \mathbb{C}^{m\times n} \to \mathbb{R}, A \mapsto \|A^\dagger\|_F \tag{II}$$
$$f_{\text{lsq}} : \mathbb{C}^{m\times n} \times \mathbb{C}^m \to \mathbb{R}, (A,b) \mapsto \min_{x\in\mathbb{C}^n} \|Ax-b\|_2 \tag{III}$$
$$f_{\text{lsq-m}} : \mathbb{C}^{m\times n} \times \mathbb{C}^m \to \mathbb{C}^n, (A,b) \mapsto \operatorname*{arg\,min}_{x:\|Ax-b\|_2 = f_{\text{lsq}}(A,b)} \|x\|_2 \tag{IV}$$
$$f_{\text{lsq-n}} : \mathbb{C}^{m\times n} \times \mathbb{C}^m \to \mathbb{R}, (A,b) \mapsto \|f_{\text{lsq-m}}(A,b)\|_2 \tag{V}$$
$$\kappa : \mathbb{C}^{m\times n} \to \mathbb{R}, A \mapsto \|A\|_F \|A^\dagger\|_F \tag{VI}$$

B. Contributions

We analyze the existence of effective algorithms on Turing machines approximating the functions I.-VI. Since the pseudoinverse mapping plays a crucial role in the computation of all considered functions, we ask whether an effective algorithm exists to compute the pseudoinverse of every matrix $A \in \mathbb{C}^{m\times n}$? We can split this question into two parts and study them separately: First, given a fixed matrix $A \in \mathbb{C}^{m\times n}$, does a Turing machine TM_A computing the pseudoinverse $A^\dagger$ (with error control) exist? Second, does a Turing machine TM_U taking any matrix $A \in \mathbb{C}^{m\times n}$ as input and constructing TM_A as output exist? We provide, despite the existence of the pseudoinverse, a negative answer to the latter, yet a positive answer to the former question. By extending the negative finding to the condition number as defined in VI., we directly obtain the previously described non-possibility result for a computable algorithmic implementation of a compiler for current quantum algorithms treating systems of linear equations [18]. Next, we describe the computability breakdown in more detail.

- We show in Theorem 6 that no algorithm, i.e., the Turing machine TM_U, computing the pseudoinverse mapping with finite error exists: Any algorithm will make arbitrarily large errors for certain input instances. This is contrasted with the observation in Proposition 9 that Turing machines computing the associated pseudoinverse of a fixed matrix exist. Therefore, the computability breakdown in the universal case arises at the level of finding said Turing machine for a given input matrix. The construction of the sought Turing machine requires specific information about the input matrix, which is not provided in this general setting. Exemplarily, we analyze and identify this breakdown in the special case of iterative algorithms with convergence guarantees in Corollary 8.
- The previous findings indicate that by diminishing the generality of the problem, i.e., restricting the input domain of potential algorithms, effective algorithmic approaches to compute the pseudoinverse may exist since appropriately reducing the input domain corresponds to increasing the available information about the remaining input instances. We demonstrate for two classes of matrices—matrices of full rank and matrices with elements bounded away from zero—that the observation is valid by establishing the existence of effective algorithms in both cases. Hence, in

practical applications, it is important to assess the given problem and derive information about the feasible input instances for potentially realizing effective solvability.
- Via Theorem 11 and its implication, we fully characterize the computability breakdown and describe a tool to check whether a problem with a specific input domain may be amenable to effective approaches. This also closely aligns with the known discontinuities of the pseudoinverse mapping. However, we wish to highlight that no algorithm effectively decides whether an arbitrary matrix satisfies the desired input conditions generally. In this sense, a verifiably correct 'red flag' functionality does not exist for the above benign input classes.

The described computational barrier does not arise due to numerical errors in real-world implementations but is an inherent feature of Turing machines. Indeed, the condition number for invertible matrices provides a measure of numerical errors for implementations of the inversion operation on actual digital hardware with floating point arithmetic. However, computing the inverse of a fixed matrix can be performed effectively, even on instances with ill-conditioned inputs. Thus, a non-computable/non-approximable property establishes an impossibility result for any realization on a digital machine. In contrast, the condition number only provides a heuristic to decide whether a specific input is benign in certain circumstances.

II. Digital Computations

The predominant computing devices today are digital computers. Digital hardware, in principle, allows for the exact computation of specific problems, such as discrete ones. For continuous problems, algorithms on digital hardware only provide an approximation of the solution. Thus, a well-defined notion of an approximate solution is required to evaluate the computed approximation and guarantee its correctness. An approach is to ask for *effective* algorithms, where algorithms compute an approximate solution and quantify the error to the (generally unknown) exact solution. In particular, algorithms must be able to provide an approximation within any prescribed error bound. A theoretical model for digital computers, which captures the logic of digital computations but neglects real-world limitations such as memory constraints, energy consumption, etc, is given by Turing machines [34]. Any algorithm that can be executed by a real-world (digital) computer can, in theory, be simulated by a Turing machine and vice versa. Therefore, the existence of effective algorithms for digital computers is studied via Turing machines [35]–[41].

A. Computable Analysis

This section presents the necessary definitions from computability theory on continuous domains based on Turing machines. For a comprehensive overview of real-valued computability, we refer to [42]–[45], whereas elementary topics of computability theory, such as recursive functions on discrete domains and the formal definition of Turing machines, can be found in any textbook [46].

The general paradigm describing digital computation on real numbers introduced by Turing himself [34] is based on the transformation of sequences approximating real numbers with arbitrary precision.

Definition 1.
- *A sequence $(r_k)_{k\in\mathbb{N}}$ in $\mathbb{Q}$ is computable, if there exist recursive functions $a, b, s : \mathbb{N} \to \mathbb{N}$ such that*

$$b(k) \neq 0 \text{ and } r_k = (-1)^{s(k)} \cdot \frac{a(k)}{b(k)} \qquad \forall k \in \mathbb{N}.$$

- *A sequence $(x_k)_{k\in\mathbb{N}}$ in $\mathbb{R}$ converges effectively to $x \in \mathbb{R}$, if there exists a recursive function $e : \mathbb{N} \to \mathbb{N}$ such that*

$$|r_k - x| \leq 2^{-N} \quad \forall N \in \mathbb{N} \text{ and all } k \geq e(N).$$

- *A real number $x \in \mathbb{R}$ is computable, if there exists a computable rational sequence $(r_k)_{k\in\mathbb{N}}$ converging effectively to x. We call $(r_k)_{k\in\mathbb{N}}$ a representation of x.*
- *A sequence $(x_n)_{n\in\mathbb{N}}$ in $\mathbb{R}$ is* computable*, if there exist a computable rational sequence $(r_{n,k})_{n,k\in\mathbb{N}}$ and a recursive function $e : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that*

$$|x_n - r_{n,k}| \leq 2^{-N} \quad \forall n, N \in \mathbb{N} \text{ and all } k \geq e(n, N).$$

Remark 2. *We denote the set of computable real numbers by $\mathbb{R}_c$. Moreover, the previous definitions can be extended to $\mathbb{R}^d$ and $\mathbb{C}^d$—considered as a $2d$-dimensional real vector space—by requiring that each one-dimensional component(-wise sequence) is computable. We denote the set of computable complex numbers by $\mathbb{C}_c := \{x + iy \in \mathbb{C} : x, y \in \mathbb{R}_c\}$.*

The computability of functions is a well-studied property, and various computability notions exist [45]; we will employ the following ones to establish positive and negative results regarding effective computations on digital hardware.

Definition 3. *For $D \subset \mathbb{C}^d$, a function $f : D \to \mathbb{C}_c^m$ is*
- *Borel-Turing computable, if there exists a Turing machine that transforms each representation of $x \in D \cap \mathbb{C}_C^d$ into a representation for $f(x)$.*
- *Banach-Mazur computable, if f maps any computable sequences $(x_n)_{n\in\mathbb{N}}$ in $D \cap \mathbb{C}_c^d$ onto computable sequences $(f(x_n))_{n\in\mathbb{N}}$.*

Remark 4. *Borel-Turing computability formalizes an intuitive notion of computation, where an algorithm approximates a given function to any accuracy with error control. Thus, effective algorithms can only exist if the associated problem is Borel-Turing computable. A weaker form of computability is given by Banach-Mazur computable functions, which preserve computability of sequences. It is well known that Borel-Turing computability implies Banach-Mazur computability and that computable functions in both sense are continuous [45].*

Remark 5. *Basic arithmetic operations, as well as elementary operations such as the absolute value function, are computable [44]. This also immediately implies the computability of the Euclidean and Frobenius norm.*

Another slightly different notion of computation is established via oracles providing representations for any (and not

just computable) objects [47]. However, for our purposes of analyzing digital computations, the natural domain is indeed characterized by computable objects, and we apply the introduced computability framework in the remainder.

III. Main Results

Next, we present our results regarding the effective computation of the functions I.-VI. The crucial step therein is the computation of the pseudoinverse mapping, and we first show that an effective approach to approximate this mapping does not exist. Subsequently, we analyze the breakdown of existing algorithmic strategies on the basis of iterative processes and establish restrictions on the input domain to enable the introduction of effective algorithms.

A. Banach-Mazur Non-Computability of the Pseudoinverse

Assuming Borel-Turing computability of the mapping of a matrix on its pseudoinverse immediately implies the effective computation of the functions I.-VI. on digital hardware. Can we find a constructive algorithmic approach that enables us to verify the computability of said mapping? For instance, the singular value decomposition (SVD) is a standard method to compute the pseudoinverse. A crucial step therein consists of identifying the non-zero singular values of the input matrix. However, comparisons to zero can not be performed effectively on digital hardware [44]. Hence, SVD does not suffice to establish an effective approach to compute the pseudoinverse. The next theorem shows that this is not due to a specific property of SVD, but an effective approach to approximate the pseudoinverse does not exist in full generality.

Theorem 6. *For a function f, denote a function with the same domain and codomain by $\hat{f}$. Then, for $m, n \geq 2$, the functions I.-VI. are not algorithmically approximable, i.e., there exists no Banach-Mazur computable function $\hat{f}$ such that*

(i) $\sup\limits_{\substack{A\in\mathbb{C}^{m\times n}:\\ \|A\|_F\leq\sqrt{2}}} \|f(A)-\hat{f}(A)\|_F < \infty$ *for* $f \in \{f^\dagger, f^{\|\dagger\|}, \kappa\}$;

(ii) $\sup\limits_{\substack{(A,b)\in\mathbb{C}^{m\times n}\times\mathbb{C}^m:\\ \|A\|_F\leq\sqrt{2},\|b\|_2\leq\sqrt{2}}} |f(A,b)-\hat{f}(A,b)| < \frac{1}{4}$ *for* $f = f_{lsq}$;

(iii) $\sup\limits_{\substack{(A,b)\in\mathbb{C}^{m\times n}\times\mathbb{C}^m:\\ \|A\|_F\leq\sqrt{2},\|b\|_2\leq\sqrt{2}}} \|f(A,b)-\hat{f}(A,b)\|_2 < \infty$ *for* $f \in \{f_{lsq\text{-}m}, f_{lsq\text{-}n}\}$.

Remark 7. *Since the results are also valid in Borel-Turing sense, effective algorithmic computations on digital hardware of the problems posed by I.-VI. in the sense of Remark 4 are not feasible for accuracies below the introduced lower bounds. In particular, except for f_{lsq}, there does not exist a finite uniform bound: any algorithm will potentially make arbitrarily large errors on any sufficiently large compact input domain.*

How does the finding relate to our intuition and empirical evidence? Can we precisely identify input instances or input domains that lead to algorithmic failure? Regarding the former, we demonstrate, based on iterative processes with convergence guarantees, at what level the algorithmic breakdown arises, which leads to concrete observations about the latter question.

B. Non-Effectiveness of Iterative Processes computing the Pseudoinverse

It is important to stress that Theorem 6 does not contradict the existence of procedures, which compute the pseudoinverse with provable convergence guarantees. For a non-zero matrix $A \in \mathbb{C}^{m\times n}$ with rank p and some $0 < \alpha < 2\,\|A\|_2^{-2}$, the iterative process $(A_k)_{k\in\mathbb{N}}$ based on

$$A_k := \begin{cases} \alpha A^H & : \text{ if } k = 0, \\ 2A_{k-1} - A_{k-1}AA_{k-1} & : \text{ if } k \in \mathbb{N}, \end{cases} \tag{1}$$

converges to the pseudoinverse $A^\dagger$ for $k \in \mathbb{N}\cup\{0\}$ via

$$\|A^\dagger - A_k\|_2 \leq \frac{\|A\|_2}{\lambda_p(A^HA)}(1-\alpha\lambda_p(A^HA))^{2^k}, \tag{2}$$

where $\|\cdot\|_2$ denotes the spectral norm and $\lambda_p(A^HA) > 0$ the p-th largest eigenvalue of A^HA [26]. In particular, the convergence rate obeys

$$\|A^\dagger - A_k\|_2 \leq \|A\|_2\,\|A^\dagger - A_{k-1}\|_2^2 \quad \text{for } k \in \mathbb{N}. \tag{3}$$

This iterative process seemingly yields an effective algorithm to compute the pseudoinverse to any desired precision via the convergence of the sequence $(A_k)_{k\in\mathbb{N}}$. However, convergence is insufficient to guarantee an effective algorithm's existence. Indeed, the iteration step in (1) is expressed via the Borel-Turing computable function $f : \mathbb{C}^{n\times m}\times\mathbb{C}^{m\times n} \to \mathbb{C}^{n\times m}$ given by $f_B(X) := f(X,B) = 2X - XBX$ as

$$A_k = f_A(A_{k-1}) = f_A(f_A(\cdots f_A(A_0))) \quad \text{for } k \in \mathbb{N}. \tag{4}$$

Hence, the sequence $(A_k)_{k\in\mathbb{N}}$ is a computable sequence provided that the initialization A_0 is a computable matrix since the composition of Borel-Turing computable functions is computable. Crucially, implementing the iterative process requires a stopping criterion that aborts the computation once the approximation is sufficiently close to $A^\dagger$. Ideally, the algorithm would take an error parameter $N \in \mathbb{N}$ as additional input and halt if, for some $m \in \mathbb{N}$, the approximation error of A_m is smaller than 2^{-N}. Therefore, an effective implementation takes a representation of any (non-zero) matrix $A \in \mathbb{C}^{m\times n}$ and an error parameter N as input and outputs (an approximation of) $A^\dagger$ up to an error of 2^{-N}. The key point is that the initialization and the termination of the iteration are integral parts of the effective implementation. Otherwise, the iterative process could not be performed independently and effectively on digital hardware.

In the context of Turing machines the question of the effective implementation can be formalized in the following way: Does there exist a Borel-Turing computable function $G : \mathbb{C}^{m\times n} \to \mathbb{C}^{n\times m}$ such that $(A_k)_{k\in\mathbb{N}}$ computed according to (4) with $A_0 = G(A)$ converges effectively to the pseudoinverse $A^\dagger$, i.e., does there exist a recursive function $e_G : \mathbb{N} \to \mathbb{N}$ such that, for any $A \in \mathbb{C}^{m\times n}$ with $G(A) = A_0$,

$$\|A^\dagger - A_k\|_2 \leq \frac{1}{2^N} \text{ holds true } \forall N \in \mathbb{N} \text{ and } k \geq e_G(N)? \tag{5}$$

Assuming that G and e_G exist entails that the representation of any non-zero matrix $A \in \mathbb{C}^{m\times n}$ can be effectively transformed

into a representation of its pseudoinverse via (5), i.e., the function $f^\dagger$ is Borel-Turing computable on $\mathbb{C}^{m\times n}\setminus\{0\}$. This contradicts Theorem 6 stating that (5) can not even be realized for $N=1$ on a compact input set $\|A\|_F \leq \sqrt{2}$—the proof verifies that excluding the zero matrix from the input domain does not impact the result.

Corollary 8. *For $n,m \geq 2$, there does not exist a Borel-Turing computable function $G:\mathbb{C}^{m\times n}\to\mathbb{C}^{n\times m}$ to compute an initialization $A_0=G(A)$ such that the sequence $(A_k)_{k\in\mathbb{N}}$,*

$$A_k = 2A_{k-1} - A_{k-1}AA_{k-1} \quad \text{for } k\in\mathbb{N},$$

converges effectively to the pseudoinverse $A^\dagger$.

Since the spectral norm is bounded by the (computable) Frobenius norm, fixing a computable function G, which provides a suitable initialization via (1), is straightforward. However, the associated recursive function e_G measuring the convergence rate via (2) does not exist because determining the rank of a matrix $A\in\mathbb{C}^{m\times n}$ and finding a (non-zero) lower bound on the $\operatorname{rank}(A)$-th largest eigenvalue of A^HA can not be effectively computed without further information—hereby, determining the rank effectively is the prohibitive step. On the other hand, providing additional information about the input matrices by restricting the input domain, e.g., in the edge case to contain exactly one computable element, the iterative process yields an effective approach, as the next result shows.

Proposition 9. *For any function I.-VI., i.e., $f\in\{f^\dagger, f^{\|\dagger\|}, \kappa, f_{lsq}, f_{lsq\text{-}m}, f_{lsq\text{-}n}\}$, and computable input $x\in dom(f)$, $f(x)$ is also computable.*

Remark 10. *Corollary 8 and Proposition9 illustrate the contrast between effective initialization and effective convergence of the iterative process to obtain the pseudoinverse. If an effective initialization is implemented, then effective convergence is not feasible (Corollary 8). Likewise, if effective convergence is guaranteed, then the corresponding initialization can not be performed effectively but needs to be manually crafted (Proposition 9).*

The previous discussion shows that any iterative process needs to rely on heuristics for initialization and stopping criteria. The heuristics may work reasonably well for specific classes of matrices, but they can not guarantee correctness (in the effective sense) in the general case. In particular, the proof of Proposition 9 is not constructive and can therefore not be generalized to an algorithm taking arbitrary computable inputs since specific properties of the given instance need to be provided to the associated algorithm. However, is it possible to at least establish effective approaches (and not only heuristics) to tackle the problems I.-VI. for specific classes of matrices? This is indeed feasible as we establish next.

C. Full description of computability breakdown

Can we identify the properties of matrices that lead to the computability breakdown and thereby identify domains that allow for effective treatment? The proof of Theorem 6 and the short analysis of the SVD approach in Section III-A indicate that non-computability indeed arises in specific circumstances. The following result and its implication show that non-computability can be prevented in certain instances, although it can not be circumvented for sufficiently general domains.

Theorem 11. *For any function I.-VI., i.e., $f\in\{f^\dagger, f^{\|\dagger\|}, \kappa, f_{lsq}, f_{lsq\text{-}m}, f_{lsq\text{-}n}\}$, there exists a Turing machine that transforms a representation of any computable input $x\in dom(f)$ into a representation for $f(x)$ provided that each member of the representation of x has the same rank as x itself.*

Remark 12. *Note that Theorem 11 does not imply Borel-Turing computability. Indeed, the non-compliance with the rank property is exactly the condition exploited in the proof of the non-approximability result in Theorem 6.*

Theorem 6 together with Theorem 11 fully characterize the algorithmic approximability of the pseudoinverse and the related functions II.-VI.:

- If the rank property is satisfied, then the rank of any input matrix can be directly computed via its given representation. Subsequently, the pseudoinverse can be computed to any desired accuracy by the constructive algorithm described in the proof of Theorem 11.
- In certain settings, the rank property can be effectively guaranteed, i.e., there exists an effective algorithm that transforms any given representation into a representation meeting the rank property, e.g., the set of full rank matrices in $\mathbb{C}^{m\times n}$ or the set $\mathbb{C}^{m\times n}_\varepsilon := \{A\in\mathbb{C}^{m\times n} : |A_{ij}| > \varepsilon \text{ or } A_{ij}=0\ \forall i,j\}$ for any $\varepsilon>0$. Therefore, the pseudoinverse mapping is Borel-Turing computable on the respective domains. However, in both cases, the membership to the given set can not be effectively decided, i.e., for general input domains the best one can hope for is a pre-processing step that verifies if a given input belongs to a 'benign' class of matrices (but does not effectively distinguish between benign and non-benign classes), i.e., only semi-decides the membership property.
- The algorithmic non-approximability arises for input domains being arbitrarily close to or containing matrices with distinct rank (if no further conditions are imposed on the input instances). This is, in some sense, an expected behaviour because it coincides with the discontinuities of the pseudoinverse mapping [48]. In these cases, a non-computable problem can be encoded in the considered input domain, leading to a non-approximability result. Hence, the key feature of the set of full rank matrices and $\mathbb{C}^{m\times n}_\varepsilon$ is preventing a (smooth) rank transition by design—the set of full rank matrices is open, whereas $\mathbb{C}^{m\times n}_\varepsilon$ artificially imposes via ε a testable condition on rank transitioning. In contrast, the set of matrices with fixed rank smaller than $\min(m,n)$ also contains only matrices with equal rank, however, the set is closed so that for instances on the boundary, there exist representations solely consisting of matrices with a distinct rank.

REFERENCES

[1] H. Boche, A. Fono, and G. Kutyniok, "Turing meets Moore-Penrose: Computing the pseudoinverse on Turing machines," *arXiv:2212.02940*, 2025.

[2] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proc. Annu. Symp. Found. Comput. Sci.*, pp. 124–134, 1994.

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[4] A. Montanaro, "Quantum algorithms: an overview," *npj Quantum Inf.*, vol. 2, p. 15023, 2016.

[5] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[6] A. Kitaev, A. Shen, and M. Vyalyi, *Classical and Quantum Computation*. Graduate studies in mathematics, American Mathematical Soc., 2002.

[7] R. P. Feynman, "Quantum mechanical computers," *Optics News*, vol. 11, no. 2, pp. 11–20, 1985.

[8] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[9] J. M. Gambetta, J. M. Chow, and M. Steffen, "Building logical qubits in a superconducting quantum computing system," *npj Quantum Inf.*, vol. 3, no. 1, p. 2, 2017.

[10] H. Boche, Y. N. Böck, Z. G. del Toro, and F. H. P. Fitzek, "Feynman meets Turing: The uncomputability of quantum gate-circuit emulation and concatenation," *IEEE Trans. Comput.*, vol. 74, no. 3, 2025.

[11] Y. N. Böck, H. Boche, Z. G. del Toro, and F. H. P. Fitzek, "Feynman meets Turing: Computability aspects of quantum compiling revisited," *IEEE Trans. Comput.*, vol. 75, no. 2, pp. 516–526, 2026.

[12] Y. N. Böck, H. Boche, and F. H. P. Fitzek, "Feynman meets Turing: Computability aspects of exact circuit synthesis, gate efficiency, and the spectral gap conjecture," *IEEE Trans. Quantum Eng.*, vol. 7, pp. 1–31, 2026.

[13] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Phys. Rev. A*, vol. 52, pp. 3457–3467, 1995.

[14] H. F. Chau and F. Wilczek, "Simple realization of the Fredkin gate using a series of two-body operators," *Phys. Rev. Lett.*, vol. 75, pp. 748–750, 1995.

[15] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, pp. 4091–4094, 1995.

[16] A. Y. Kitaev, "Quantum computations: algorithms and error correction," *Russian Mathematical Surveys*, vol. 52, no. 6, pp. 1191–1249, 1997.

[17] C. M. Dawson and M. A. Nielsen, "The Solovay-Kitaev algorithm," *Quant. Inf. Comput.*, vol. 6, pp. 081–095, 2006.

[18] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Phys. Rev. Lett.*, vol. 103, 2009.

[19] S. Barz, I. Kassal, M. Ringbauer, Y. O. Lipp, B. Dakić, A. Aspuru-Guzik, and P. Walther, "A two-qubit photonic quantum processor and its application to solving systems of linear equations," *Scientific Reports*, vol. 4, 2014.

[20] X.-D. Cai, C. Weedbrook, Z.-E. Su, M.-C. Chen, M. Gu, M.-J. Zhu, L. Li, N.-L. Liu, C.-Y. Lu, and J.-W. Pan, "Experimental quantum computing to solve systems of linear equations," *Phys. Rev. Lett.*, vol. 110, 2013.

[21] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, pp. 195–202, 2017.

[22] J. Liu, M. Liu, J.-P. Liu, Z. Ye, Y. Wang, Y. Alexeev, J. Eisert, and L. Jiang, "Towards provably efficient quantum algorithms for large-scale machine-learning models," *Nat. Commun.*, vol. 15, p. 434, 2024.

[23] E. H. Moore, "On the reciprocal of the general algebraic matrix," *Bull. Am. Math. Soc.*, vol. 26, pp. 394–395, 1920.

[24] R. Penrose, "A generalized inverse for matrices," *Math. Proc. Camb. Philos. Soc.*, vol. 51, no. 3, p. 406–413, 1955.

[25] J. C. A. Barata and M. S. Hussein, "The Moore–Penrose pseudoinverse: A tutorial review of the theory," *Brazilian Journal of Physics*, vol. 42, no. 1, pp. 146–165, 2012.

[26] A. Ben-Israel and D. Cohen, "On iterative computation of generalized inverses and associated projections," *SIAM J. Numer. Anal.*, vol. 3, no. 3, pp. 410–419, 1966.

[27] S. L. Campbell and C. D. Meyer, *Generalized inverses of linear transformations*. SIAM, 2009.

[28] I. Bajo, "Computing Moore–Penrose inverses with polynomials in matrices," *Am. Math. Mon.*, vol. 128, no. 5, pp. 446–456, 2021.

[29] A. Ben-Israel and T. N. E. Greville, *Generalized Inverses: Theory and Applications*. Springer New York, NY, 2003.

[30] X. Sheng and G. Chen, "A note of computation for M-P inverse $A^{\dagger}$," *Int. J. Comput. Math.*, vol. 87, no. 10, pp. 2235–2241, 2010.

[31] J. Ji, "Gauss–Jordan elimination methods for the Moore–Penrose inverse of a matrix," *Linear Algebra Its Appl.*, vol. 437, no. 7, pp. 1835–1844, 2012.

[32] V. Pan and R. Schreiber, "An improved Newton iteration for the generalized inverse of a matrix, with applications," *SIAM J. Sci. Stat. Comput.*, vol. 12, no. 5, pp. 1109–1130, 1991.

[33] T. Söderström and G. W. Stewart, "On the numerical properties of an iterative method for computing the Moore–Penrose generalized inverse," *SIAM J. Numer. Anal.*, vol. 11, no. 1, pp. 61–74, 1974.

[34] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proc. London Math. Soc.*, vol. s2-42, no. 1, pp. 230–265, 1936.

[35] J. Myhill, "A recursive function, defined on a compact interval and having a continuous derivative that is not recursive.," *Michigan Mathematical Journal*, vol. 18, no. 2, pp. 97 – 98, 1971.

[36] M. B. Pour-El and N. Zhong, "The wave equation with computable initial data whose unique solution is nowhere computable," *Mathematical Logic Quarterly*, vol. 43, no. 4, pp. 499–509, 1997.

[37] H. Boche and V. Pohl, "Turing meets circuit theory: Not every continuous-time LTI system can be simulated on a digital computer," *IEEE Trans. Circuits Syst. I: Regul. Pap.*, vol. 67, no. 12, pp. 5051–5064, 2020.

[38] D. Elkouss and D. Pérez-García, "Memory effects can make the transmission capability of a communication channel uncomputable," *Nat. Commun.*, vol. 9, 03 2018.

[39] R. F. Schaefer, H. Boche, and H. V. Poor, "Turing meets Shannon: On the algorithmic computability of the capacities of secure communication systems (invited paper)," in *2019 IEEE Int. Workshop Signal Process. Adv. Wirel. Commun. (SPAWC)*, pp. 1–5, 2019.

[40] H. Boche and V. Pohl, "On the algorithmic solvability of spectral factorization and applications," *IEEE Trans. Inf. Theory*, vol. 66, no. 7, pp. 4574–4592, 2020.

[41] H. Boche and U. J. Mönich, "Turing computability of Fourier transforms of bandlimited and discrete signals," *IEEE Trans. Signal Process.*, vol. 68, pp. 532–547, 2020.

[42] R. I. Soare, "Recursively enumerable sets and degrees," *Bull. Am. Math. Soc.*, vol. 84, pp. 1149–1181, 1987.

[43] K. Weihrauch, *Computable Analysis: An Introduction*. Berlin, Heidelberg: Springer-Verlag, 2000.

[44] M. B. Pour-El and J. I. Richards, *Computability in Analysis and Physics*. Perspectives in Logic, Cambridge University Press, 2017.

[45] R. Downey, ed., *Turing's legacy: developments from Turing's ideas in logic*. Lecture Notes in Logic, Cambridge University Press, 2014.

[46] S. Cooper, *Computability Theory*. Chapman Hall/CRC Mathematics Series, CRC Press, 2017.

[47] K.-I. Ko, *Complexity Theory of Real Functions*. USA: Birkhauser Boston Inc., 1991.

[48] V. Rakočević, "On continuity of the Moore-Penrose and Drazin inverses," *Matematichki Vesnik*, vol. 49, no. 3-4, pp. 163–172, 1997.