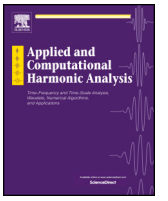
Applied and Computational Harmonic Analysis 74 (2025) 101719

Contents lists available at ScienceDirect

Applied and Computational Harmonic Analysis

journal homepage: www.elsevier.com/locate/acha



Full Length Article

Inverse problems are solvable on real number signal processing hardware

Holger Boche [a,b,c,d,*], Adalbert Fono [e], Gitta Kutyniok [e,f,g,h]

[a] Institute of Theoretical Information Technology, TUM School of Computation, Information and Technology, Technical University of Munich, Germany
[b] Munich Center for Quantum Science and Technology (MCQST), Munich, Germany
[c] Munich Quantum Valley (MQV), Munich, Germany
[d] BMBF Research Hub 6G-life, Dresden, Germany
[e] Department of Mathematics, Ludwig-Maximilians-Universität München, Germany
[f] Department of Physics and Technology, University of Tromsø, Norway
[g] Munich Center for Machine Learning (MCML), Munich, Germany
[h] DRL-German Aerospace Center, Institute of Robotics and Mechatronics, Germany

ARTICLE INFO

Communicated by Holger Rauhut

Keywords:
Inverse problems
Deep learning
Computing theory
BSS machines

ABSTRACT

Despite the success of Deep Learning (DL) serious reliability issues such as non-robustness persist. An interesting aspect is, whether these problems arise due to insufficient tools or fundamental limitations of DL. We study this question from the computability perspective by characterizing the limits the applied hardware imposes. For this, we focus on the class of inverse problems, which, in particular, encompasses any task to reconstruct data from measurements. On digital hardware, a conceptual barrier on the capabilities of DL for solving finite-dimensional inverse problems has in fact already been derived. This paper investigates the general computation framework of Blum-Shub-Smale (BSS) machines, describing the processing and storage of arbitrary real values. Although a corresponding real-world computing device does not exist, research and development towards real number computing hardware, usually referred to by "neuromorphic computing", has increased in recent years. In this work, we show that the framework of BSS machines does enable the algorithmic solvability of finite dimensional inverse problems. Our results emphasize the influence of the considered computing model in questions of accuracy and reliability.

1. Introduction

We study inverse problems in imaging sciences from the perspective of algorithmic constructability of solutions. Image reconstruction from measurements is crucial in scientific, industrial, and medical applications such as electron microscopy, seismic imaging, magnetic resonance imaging (MRI), and X-ray computed tomography (CT). Various methods to solve inverse problems have been introduced, ranging from sparse regularization techniques [42,45,94,110] including compressed sensing [33,34,36,46–48,65] to deep learning (DL) techniques [4,27,31,58,91,114].

DL systems learn how to perform reconstruction best by optimizing reconstruction quality based on previous data. They are the predominant approach to tackling inverse problems nowadays. Although DL has been applied with great success in various fields such

* Corresponding author.
E-mail address: boche@tum.de (H. Boche).

https://doi.org/10.1016/j.acha.2024.101719
Received 28 February 2023; Received in revised form 22 July 2024; Accepted 8 October 2024

Available online 24 October 2024

as image classification [59], playing board games [97], natural language processing [30], and protein folding prediction [95], there do exist certain drawbacks of this approach, most prominently, its lack of reliability [2]. In a larger context, lack of reliability refers to problems regarding safety, security, privacy, and responsibility of DL methods [28,60,71,77,108]. Instabilities towards adversarial examples, i.e., the fact that DL methods can be easily misled through minor input perturbations constructed by an adversary, are a typical example [37,101,104]. Despite methods to alleviate this instability phenomenon [74,82], a full understanding is still missing [63].

The disadvantages may be tolerable in static circumstances but in highly dynamic, safety-critical, and autonomous applications they pose a serious risk [70,80]. Integrating a human observer into the DL pipeline [111] severely limits the autonomy of DL methods and may not be feasible in certain applications such as autonomous driving. Therefore, incorporating certificates for a desired application is a step toward reliable DL, however, DL systems typically operate without 'hard' guarantees concerning the accuracy and correctness of their output [11,68,78,90,113]. Unfortunately, it is a priori not clear if implementing this process in a DL pipeline is possible. Thus it is critical to understand in which circumstances we can solve a given problem by a reliable algorithmic computation. Subsequently, the possibility of implementing certificates can be evaluated or the certificate-based approach to achieve reliable DL in this strict sense cannot be expected. In this way, we can distinguish between the inherent limitations of DL itself and the fundamental limitations of any algorithm tackling a given problem.

Specifically, we study whether the solution of inverse problems – a task often tackled by DL – can in principle be computed. For this purpose, we apply the notion of algorithmic solvability, which describes an abstract framework for accurate computations on a given computing device. We find that the algorithmic solvability of inverse problems depends on the utilized computing device; on digital hardware, algorithmic solvability cannot be achieved, whereas on analog hardware it is potentially feasible. Thus, obtaining reliable DL tackling inverse problems may eventually depend on the computing platform.

1.1. Analog and digital computing platforms

The most prominent and universally applied computing platform is digital hardware. Turing machines represent an abstract concept of digital machines [105], i.e., they provide a means to study whether it is in principle possible to compute the solution of a task algorithmically on a digital computer. Borel first introduced the concept of computability in 1912 [26]. On this basis, Turing refined the idea by linking it to an abstract computing device – the Turing machine – and established notions like computable numbers and computable functions [105]. Computable numbers constitute a countable, proper subset of the real numbers that can be effectively approximated by computable sequences of rational numbers – the natural domain of Turing machines. Many real-world problems are of a continuum nature, i.e., their input and output quantities are described by real-valued variables. Hence, these quantities can only be approximated but not represented exactly on Turing machines. This dichotomy lies at the core of the non-computability of many tasks. Problems, that are not computable by a Turing machine, cannot be solved or even algorithmically approximated in a controlled way by any current or future digital computer architecture.

In the framework of Turing machines the algorithmic solvability of finite-dimensional inverse problems was studied in [7,19,41]. It was found that any method that runs on digital hardware is subject to certain boundaries when approximating the solution maps of inverse problems. Hence, these results establish a fundamental barrier on digital computing devices that also affects DL implemented on digital hardware. An interesting question is whether this limitation is connected to the properties of Turing machines or inverse problems. In other words, is the result problem-specific or linked to digital hardware? Therefore, we study how powerful the signal processing unit must be to enable the algorithmic solution of inverse problems. In the following, we consider and analyze inverse problems under a more general (analog) computation model based on exact real number calculations, i.e., arbitrary real numbers can be processed and stored. Thus, such a model is not suitable for implementation on digital hardware. One might argue that treating computing models beyond standard digital hardware is purely of theoretical interest since such hardware platforms cannot be obtained with current (or possibly future) manufacturing capabilities. However, in recent years novel approaches inspired by biological neural networks such as neuromorphic computing and signal processing systems have been proposed [40].

Instead of solely relying on binary numbers, electronic neuromorphic systems combine digital and analog computations by incorporating real values through electrical values, like current and voltages. Regarding the practical development and implementation of neuromorphic hardware platforms, there has been made intriguing progress in industrial research, among others by IBM [62], Intel [64] and Samsung [57]. Furthermore, the expected energy savings from deploying artificial intelligence (AI) applications on neuromorphic hardware are promising [12,49,75,87,98]. Consequently, neuromorphic computing platforms are a forthcoming premium solution for implementing AI applications [83]. Lastly, neuromorphic computing offers advantages compared to digital platforms for emerging concepts like "in-memory computing" by design [29,67,84,92]. Another line of research focuses on biocomputing where living cells act as computing and signal processing platforms that allow analog computations of human-defined operations [56]. In recent years, progress has been made in understanding and implementing analog computing features in biological systems [85,106,109], however, still many challenges lie ahead. Finally, we also want to mention the progress in quantum simulations which provide yet another approach to analog computations; see [44,53] and the references therein.

In contrast to digital computers and Turing machines, no general mathematical concept to describe universal analog computations exists. The Blum-Shub-Smale (BSS) machine [15] is a suitable model to analyze the limits of analog computations; potentially, BSS machines yield a mathematical description of a universal analog computer based on exact real number computation. It was conjectured in [56] that the mathematical model of BSS machines provides an (abstract) description of biocomputing and neuromorphic systems. However, it is not clear whether and to what degree a realization is feasible in practice. Thus, the BSS model is not appropriate to evaluate the capabilities of current analog hardware. The BSS model instead enables us to investigate whether idealized analog, i.e.,

exact real number processing, hardware allows for the algorithmic solution of a given problem. A problem is deemed inherently hard and unlikely to be solved on any (future) analog hardware device if it is not algorithmically solvable on BSS machines. Therefore, an interesting question is whether the same limitations concerning inverse problems as in the Turing model also arise in the BSS model. Intriguingly, we show that the situation is different under the BSS model, namely, the solution maps of inverse problems can under certain conditions be computed without restrictions. Therefore, the algorithmic solvability of inverse problems is inherently connected to the computing platform.

1.2. Related work

We want to point out that the capabilities of BSS machines compared to Turing machines have already been studied for practically relevant tasks such as denial-of-service attacks [17,21,23] and remote state estimation [16]. To translate the theoretical advantages of BSS machines in detecting denial-of-service attacks to practice, the concept of a neuromorphic twin of the communication system was introduced in [22]. This concept extends the (previously digital) virtual twinning technology, i.e., a method for mapping complex communication systems onto suitable hardware platforms, to the neuromorphic domain based on BSS machines [25].

For inverse problems, [7,41] first showed non-computability in the Turing model, followed by [19] in a slightly different setup; we refer to [19] for a detailed discussion about the differences. One distinction between the findings in [19] and [7,41] is the employed Turing machine in the analysis. In [7,41], Oracle Turing machines were applied, where Turing machines gain access to arbitrary real numbers via rational Cauchy sequences provided by an oracle [69]. This is a clear distinction to Turing machines which operate on a proper subset of the real numbers – on the computable numbers. However, the results in [7,41] also transfer to the Turing model (without oracles).

Nevertheless, the assumption that the input representation is detached from the actual computing device via an oracle is significant. The idea of the oracle approach is that the input representation in a computing device will be necessarily inexact. Either the inputs are corrupted by noise, measurement error, rounding, etc. or the hardware inherently cannot represent the input exactly. Moreover, floating-point arithmetic, a common approach to represent numbers on today's digital hardware, stores even rational numbers such as $1/3$ only approximately. In this sense, the inexact input model aims for a general description of a given task, independent (to a certain degree) from the utilized computing device, and any notion of computability on specific hardware assumes the described input representation. Therefore, the capabilities of BSS machines for solving inverse problems were studied under inexact input representations via an oracle in [7,41] as well and it was found that the same non-computability results arise for BSS machines as for Turing machines.

1.3. Results and impact

Our goal is to assess the capabilities and limitations of BSS machines to solve finite-dimensional inverse problems. From our perspective, the core strength of BSS machines – to store and process real numbers precisely without error – is neglected when using the inexact input model. Indeed, the differences between Turing and BSS machines mainly affect irrational numbers excluded by the approximative input representations based on rational numbers. Thus, it is convincing that the same restrictions towards the computability of inverse problems hold for BSS machines as for Turing machines in the inexact input model, where only a partial power of BSS machines is considered. Our findings show that if BSS machines process real numbers exactly they have strictly greater capabilities than Turing machines for solving inverse problems.

For real-valued inverse problems, we establish a clear disparity between the capabilities of BSS and Turing machines. BSS machines allow for a solver that can be applied to any inverse problem of a given dimension. Thus, the solver is not connected to a specific inverse problem but can handle arbitrary inverse problems of a fixed dimension. In contrast, on Turing machines, the algorithmic solvability is limited such that the described solver cannot exist. Nevertheless, less powerful solvers adapted to a specific inverse problem may still be realizable on Turing machines. However, they lack the universality of the general solver on BSS machines since they apply only to a fixed, pre-defined task. The situation is more intricate in the broader case of complex-valued inverse problems. Here, the Turing and BSS models share limitations concerning the algorithmic solvability of finite-dimensional inverse problems. However, the limitations in the BSS model can be overcome to a certain degree. Modifications in the problem formulation suffice to guarantee BSS solvability while similar modifications do not rectify the negative result in the Turing model. Additionally, the algorithmic solvability of an approximate problem is shown in the BSS model. Thus, the Turing and BSS model behave differently and exhibit distinct computational barriers for complex-valued inverse problems.

Hence, approaches to solving inverse problems on upcoming analog hardware in the form of neuromorphic devices may potentially have strictly greater capacity than current digital solutions. The implications of our findings concerning DL are two-fold. DL implemented on digital hardware is bounded by the limitations of the hardware itself for solving inverse problems. Therefore, reliable DL may only be realized in restricted settings, where only a limited set of inverse problems is admissible. In contrast, idealized analog hardware in principle allows for reliable DL systems tackling any inverse problem of fixed dimension simultaneously. Thus, DL systems implemented on neuromorphic hardware theoretically enable optimal and trustworthy reconstructions in inverse problems in general circumstances. Albeit a real-world realization remains an open problem, developing neuromorphic twinning approaches as in [22,25] for DL systems to solve inverse problems could potentially be a first step.

1.4. Outline

A concise overview of real number computing theory is presented in Section 2, followed by an introduction to DL with a focus on inverse problems and the capabilities of DL on real number processing hardware in Section 3. Section 4 covers the formal statement of our main result about the algorithmic solvability of inverse problems in the real number computing model. Finally, the proofs of our findings and a discussion about the implications on today's solvers conclude the paper in Section 5 and Section 6, respectively.

2. Real number computing

In this section, we motivate the notion of algorithmic solvability and formally introduce a computing model on real numbers.

2.1. Algorithmic solvability

Given a mathematical formulation of a specific problem, a key question is whether a solution exists and, upon existence, how to obtain it. It is important to realize that the existence and construction of solutions are separate questions. Although the existence of a solution can be proved by explicitly constructing it, equally valid is an approach that deduces the existence logically without specifying the solution. In practice, typically a constructive solution is required. One aims to implement the solution strategy on a computer to perform the calculations autonomously. This task depends on the feasible algorithmic operations prescribed by the properties of the utilized computing device. For instance, we cannot expect exact solutions to real-valued problems on digital hardware, however, an approximate solution can be accepted if guarantees regarding accuracy, convergence, and worst-case error are satisfied.

More formally, we have the following dependencies: Given a task or problem expressed in some formal language defining the premises of the underlying system, an algorithm is a set of instructions that solves the posed problem under these premises. If an algorithm is intended to run on a hardware platform that can only perform specific operations, then the instructions of the algorithm and the feasible operations of the hardware need to coincide. In other words, the capabilities of the utilized computing device prescribe the admissible operations of the (sought) algorithm. Therefore, the intended hardware clearly defines the capabilities and limitations of any suitable algorithm for practical applications. Hence, it is crucial to be aware of the limitations of a given hardware platform concerning computing capabilities. In this manner, we can evaluate whether reliable DL is in principle attainable: Reliable DL is potentially achievable if the tackled problem is algorithmically solvable on a given hardware platform.

Another related topic is the complexity of an algorithm. Here, the efficiency of algorithms measured in the number of computation steps is studied, not only their mere existence. Then, one can classify algorithms according to their complexity and decide which suffices practical demands concerning computation time, memory requirements, etc. However, questions of complexity and practical applicability are posed only after the algorithmic solvability is established.

2.2. Blum-Shub-Smale machines

In 1989, Blum, Shub, and Smale proposed in [15] a general computing model over an arbitrary ring or field R: the Blum-Shub-Smale (BSS) machine, which is the basis of algebraic complexity theory. The BSS model allows us to carry over important concepts from classical complexity theory in the Turing machine model to a larger variety of structures, e.g., infinite fields such as $\mathbb{R}$. Here, a BSS machine can store arbitrary real numbers, can compute all field operations on "$\mathbb{R}$", i.e., "$+$" and "$\cdot$", and can compare real numbers according to the relations "$<$", "$>$", and "$=$". BSS machines therefore provide the mathematical basis for exact real number signal processing. Thus, in principle, the BSS model is suitable to investigate the power of analog information processing hardware like neuromorphic computing processors.

In essence, BSS machines can be considered a generalization of Turing machines. If R is chosen to be $\mathbb{Z}_2 = \{\{0,1\},+,\cdot\}$, then BSS machines recover the theory of Turing machines. Similar to Turing machines, BSS machines operate on infinite strips of tape according to a so-called program. The program is a finite, directed graph with five types of nodes – input, computation, branch, shift, and output – associated with different operations. For every admissible input, the output of a BSS machine is calculated according to the program in a finite number of steps, i.e., the BSS machine executes its program in finite time and stops. This immediately gives rise to the following two definitions.

Definition 2.1. *BSS-computable functions* are input-output maps Φ of the BSS machine $\mathcal{B}$. The output $\Phi_{\mathcal{B}}(x)$ is defined if the BSS machine $\mathcal{B}$ terminates on input x and the output is generated by the program of the BSS machine $\mathcal{B}$.

Definition 2.2. A set $A \subset \mathbb{R}^N$ is *BSS-decidable* if there exists a BSS machine $\mathcal{B}_A$ such that, for all $x \in \mathbb{R}^N$, we have $\mathcal{B}_A(x) = \chi_A(x)$, i.e., the characteristic function χ_A of the set A is BSS-computable.
A set $A \subset \mathbb{R}^n$ is *BSS-semidecidable* if there exists a BSS machine $\mathcal{B}_A$ such that $\mathcal{B}_A(x) = 1$ for all $x \in A$ and does not halt otherwise.

For a detailed description of BSS machines and programs running on BSS machines, we refer the reader to [13,14] and references therein. In this paper, we study whether BSS machines can compute the reconstruction maps of inverse problems. We prove that these reconstruction maps are indeed computable by BSS machines showing that real number signal processing enables algorithmic solvability for this class of problems.

3. Deep learning for inverse problems

Next, we give a short introduction to DL with a particular focus on solving inverse problems [1,66,79,81,91]. For a comprehensive depiction of DL theory, we refer to [55] and [10].

3.1. Inverse problems

We consider the following finite-dimensional, underdetermined linear inverse problem:

$$\text{Given noisy measurements } y = Ax + e \in \mathbb{C}^m \text{ of } x \in \mathbb{C}^N, \text{ recover } x, \tag{3.1}$$

where $A \in \mathbb{C}^{m\times N}, m < N$, is the *sampling operator* (or measurement matrix), $e \in \mathbb{C}^m$ is a noise vector, $y \in \mathbb{C}^m$ is the *vector of measurements*, and $x \in \mathbb{C}^N$ is the object to recover (typically a vectorized discrete image). Classical examples from medical imaging are magnetic resonance imaging (MRI), where A encodes the Fourier transform, and computed tomography (CT), where A encodes the Radon transform. In addition, the underdetermined setting $m < N$ is common in many practical applications since the number of measurements is severely limited due to time, cost, power, or other constraints.

Different approaches for solving inverse problems exist, a particularly successful is given by deep learning (DL). DL methods to a certain extent swept the area of inverse problems in imaging sciences in recent years. A unified framework for image reconstruction by manifold approximation as a data-driven supervised learning task is proposed in [114]. In [4], the authors survey methods to solve ill-posed inverse problems by combining data-driven models, particularly those based on DL, with domain-specific knowledge exploited in physical–analytical models. Approaches to tackle specific inverse problems have been presented for instance in [31] for limited angle CT, in [58,112] for MRI, in [38] for low-light photography, in [88] for computational microscopy, and in [3] for geophysical imaging. In the remainder, we focus on an end-to-end approach, where the goal is to directly learn a mapping from measurements y to reconstructed data x; see [81] for further approaches that employ DL at different steps in the processing pipeline, e.g., to learn a regularizer. The end-to-end approach represents the most fundamental method since it requires no further problem-specific knowledge or assumptions.

3.2. Basics of deep learning

The inspiration for DL originates from biology, as it utilizes an architecture called (artificial) *neural network* mimicking the human brain. A neural network consists of a collection of connected units or nodes subdivided into several layers allowing an artificial neural network to learn several abstraction levels of the input signal. In its simplest form an *L-layer feedforward neural network* is a mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^m$ of the form

$$\Phi(x) = T_L \rho(T_{L-1}\rho(\dots \rho(T_1 x))), \quad x \in \mathbb{R}^d, \tag{3.2}$$

where $T_\ell : \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$, $\ell = 1, \dots, L$, are affine-linear maps

$$T_\ell x = W_\ell x + b_\ell, \quad W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}, b_\ell \in \mathbb{R}^{n_\ell},$$

$\rho : \mathbb{R} \to \mathbb{R}$ is a non-linear function acting component-wise on a vector, and $n_0 = d, n_L = m$. The matrices W_ℓ are called *weights*, the vectors b_ℓ *biases*, and the function ρ *activation function*. A neural network can easily be adapted to work with complex inputs by representing the inputs as real vectors consisting of the real and imaginary parts.

Thus, a neural network implements a non-linear mapping parameterized by its weights and biases. The primary goal is to approximate an unknown function based on a given set (of samples) of input-output value pairs. This is typically accomplished by adjusting the network's parameters, i.e., its weights and biases, according to an optimization process; the standard approach so far is stochastic gradient descent (via backpropagation [89]). This process is usually referred to as the training of a neural network.

Triggered by the drastic improvements in the computing power of digital computers and the availability of vast amounts of training data, the area of DL has seen overwhelming practical progress in the last fifteen years. Deep neural networks (which in fact inspired the name "deep learning"), i.e., networks with large numbers of layers, lead to several breakthroughs in many applications [30,59,95,97]. Moreover, the current trend toward neuromorphic hardware promises further fundamental developments in this area. Therefore, the necessity for a thorough analysis of the mathematical foundations of DL is immanent.

3.3. Deep learning on BSS machines

An obvious question is whether a neural network can be implemented on a BSS machine, in particular, can the network be evaluated on a given input? If this is not the case, then BSS machines certainly do not provide the necessary tools to improve DL compared to implementations on Turing machines. Therefore, the input-output relation of the network has to be computable, i.e., the network has to be a BSS-computable function. Next, we show this is indeed the case under some mild assumptions.

Theorem 3.1. *A neural network Φ as defined in (3.2) is a BSS-computable function given that the activation function $\rho : \mathbb{R} \to \mathbb{R}$ is BSS-computable.*

Proof. The network Φ composes affine linear maps represented by matrix operations (matrix multiplications and vector additions) and non-linearities represented by the activation function. Affine linear maps are BSS-computable functions since they are formed exclusively through basic field operations (addition and multiplication) that can be executed on the computation nodes of a BSS machine. Since the activation function ρ is assumed to be BSS-computable, there does exist a program of a BSS machine that generates the input-output relation of Φ which is equivalent to saying that Φ is BSS-computable. □

Remark 3.2. The most common and universally applied activation function is the ReLU activation $\mathrm{ReLU}(x) = \max\{x, 0\}$, which is merely a branch condition depending on a comparison, i.e., it can be performed on the branching nodes of a BSS machine. Hence, the ReLU activation is a BSS-computable function.

Having established that evaluating a given neural network is feasible on a BSS machine, the next question is whether a specific neural network can be trained on a BSS machine to solve inverse problems. Subsequently, we will formalize this problem precisely.

4. Computability of the reconstruction map

The goal of the training process in DL is to obtain a neural network Φ, which approximates the mapping from measurements to the original data. An important question is whether Φ exists, and upon existence, can the network be constructed by an algorithm? In other words, does a training algorithm exist that computes suitable weights and biases such that Φ performs the intended reconstruction reliably? Such a training algorithm can only exist if the underlying inverse problem is algorithmically solvable on the applied hardware platform. Note that mere algorithmic solvability does not imply the existence of a practically applicable and efficient algorithm to solve inverse problems. Therefore, applying DL to algorithmically solvable problems is still beneficial since it ideally provides a generic and straightforward practical approach. On the other hand, algorithmic non-solvability indicates limitations that even DL cannot circumvent, e.g., a trade-off between generality and reliability cannot be avoided. Note that algorithms might exist that can solve a problem to a certain degree in practice although the problem is not computable. Depending on the hardware platform, the algorithm might only work properly for a restricted class of inputs, for a certain accuracy, or without correctness guarantees. Hence, understanding the limitations of a given computing device is a necessity if one aims to evaluate the solvability of a certain problem.

4.1. Problem setting

A general solution strategy for inverse problems is to rewrite the model (3.1) in a mathematically more tractable form since (3.1) is in general ill-posed. Aiming to account for uncertainties of the measurements, a relaxed formulation is considered, which has a considerably simpler solution map than the original description (3.1). Typically, the goal is to express (3.1) as an optimization problem given a sampling operator $A \in \mathbb{C}^{m\times N}$ and a vector of measurements $y \in \mathbb{C}^m$. There exist various formulations of this optimization problem, a straightforward one is given by the least-squares problem

$$\arg\min_{x\in\mathbb{C}^N} \|Ax - y\|_{\ell_2}\,. \tag{ls}$$

A minimizer of the least-squares problem can be straightforwardly obtained via the pseudoinverse of A. Since there exists an algorithm in the BSS model that computes the pseudoinverse of an arbitrary matrix (see Appendix B, in particular proof of Theorem 4.11), we can conclude that a minimizer of (ls) can be algorithmically computed on BSS machines. Conversely, such an algorithm cannot exist on Turing machines [18]. However, the solution of (ls) is generally not unique and the individual solutions tend to display different qualitative properties, i.e., not all minimizers of (ls) are of the same value when reconsidering the original problem (3.1). Therefore, additional regularization terms, which impose desired characteristics on the solution set, are added to the optimization problem. This results in minimization problems such as (quadratically constrained) *basis pursuit* [35,39]

$$\arg\min_{x\in\mathbb{C}^N} \|x\|_{\ell_1} \text{ such that } \|Ax - y\|_{\ell_2} \le \varepsilon \tag{bp}$$

and unconstrained *lasso* [9,73,103]

$$\arg\min_{x\in\mathbb{C}^N} \lambda\|x\|_{\ell_1} + \|Ax - y\|_{\ell_2}^2\,, \tag{la}$$

where the magnitude of $\varepsilon > 0$ and $\lambda > 0$ control the relaxation. The underlying idea is to exploit sparsity in the recovery without explicitly forcing sparse solutions via the ℓ_0 norm, which is typically intractable in many applications.

To reconstruct data from any given measurement we need to solve the optimization task described in (bp) or (la). The existence of an algorithmic solution presupposes the computability of the reconstruction map. We introduce the following multi-valued functions to study the computability of inverse problems: For fixed sampling operator $A \in \mathbb{C}^{m\times N}$ and some fixed optimization parameter $\mu > 0$ denote by

$$\begin{aligned} \Xi_{P,A,\mu} &: \mathbb{C}^m \rightrightarrows \mathbb{C}^N \\ &\quad y \mapsto P(A, y, \mu) \end{aligned} \tag{4.1}$$

the (multi-valued) reconstruction map so that $\Xi_{P,A,\mu}(y)$ represents the set of minimizers for the optimization problem $P(A, y, \mu)$ given a measurement $y \in \mathbb{C}^m$. For instance, for basis pursuit $P(A, y, \mu)$ is described in (bp) with $\mu = \varepsilon$.

Typically, we are not only interested in approximating the reconstruction map of a specific inverse problem associated with fixed sampling operator A and optimization parameter μ. Instead, we want to approximate the reconstruction map of any inverse problem of fixed dimension, i.e., we aim for an algorithm that may be applied to an arbitrary inverse problem described by the optimization P, without adjusting it to specific properties of an individual case. The reconstruction map of these general problems with fixed dimension $m \times N$ is given by

$$\begin{aligned} \Xi_{P,m,N} : \mathbb{C}^{m\times N} \times \mathbb{C}^m \times \mathbb{R}_{>0} &\rightrightarrows \mathbb{C}^N \\ (A, y, \mu) &\mapsto P(A, y, \mu) \end{aligned} \tag{4.2}$$

with $\Xi_{P,m,N}(A, y, \mu)$ representing the set of minimizers of the optimization problem $P(A, y, \mu)$ given an optimization parameter $\mu > 0$, a sampling operator $A \in \mathbb{C}^{m\times N}$, and an associated measurement $y \in \mathbb{C}^m$. Observe that the mappings (4.1) and (4.2) are in general set-valued, since the solution of the considered optimization problems, e.g., (bp) and (la), does not need to be unique. However, the computability of multi-valued maps is a stronger result than required for algorithmic solvability in practical applications. Indeed, computing all feasible solutions is much harder than computing a single one. In most practical circumstances, the user is not interested in the whole solution set, but it suffices to obtain exactly one feasible solution. Thus, a hypothetical algorithm should take a sampling operator $A \in \mathbb{C}^{m\times N}$, a measurement $y \in \mathbb{C}^m$, and an optimization parameter $\mu > 0$ as input and yield exactly one corresponding reconstruction $x \in \Xi_{P,m,N}(A, y, \mu)$.

To formalize this concept, note that for a multi-valued function $f : \mathcal{X} \rightrightarrows \mathcal{Y}$ there exists for each input $x \in \mathrm{dom}(f)$ at least one output $y_x^* \in f(x) \subset \mathcal{P}(\mathcal{Y})$. A single-valued restriction of f can then be defined as the function

$$\begin{aligned} f^s : \mathcal{X} &\to \mathcal{Y} \\ x &\mapsto y_x^*. \end{aligned}$$

We denote by $\mathcal{M}_f$ the set of all the single-valued functions associated with the multi-valued function f. Hence, the set $\mathcal{M}_f$ encompasses all single-valued functions f^s formed by restricting the output of a multi-valued map f to a single value for each input. If at least one function in $\mathcal{M}_f$ is computable, then we can algorithmically solve the problem proposed by f.

Definition 4.1. A problem with an input-output relation described by a multi-valued function $f : \mathcal{X} \rightrightarrows \mathcal{Y}$ is *algorithmically solvable on a BSS machine or Turing machine* if there exists a function $f^s \in \mathcal{M}_f$ that is computable on a BSS or Turing machine, respectively.

Remark 4.2. Different notions for evaluating successful algorithmic computations exist, e.g., the computability of multi-valued mappings can also be assessed via the distance to the solution set [7,41]. In the remainder, we base our analysis on Definition 4.1 and compare our findings in the BSS model to the results in the Turing model in [19], where the same computability notion for multi-valued mappings is applied.

It is irrelevant which among the possibly infinitely many functions in $\mathcal{M}_f$ is computable since any of those is an appropriate solution. Even more, it may be the case that most functions in $\mathcal{M}_f$ are non-computable; but as long as we succeed in showing computability for just one of them, we consider the task algorithmically solvable. For our needs, the relevant set of functions is $\mathcal{M}_{\Xi_{P,m,N}}$ corresponding to the multi-valued function $\Xi_{P,m,N}$ introduced in (4.2). Fig. 1 now illustrates a comparison and summary of the described settings.

Our goal is to study the BSS-computability of the reconstruction map $\Xi_{P,m,N}$ for basis pursuit (bp) and lasso optimization (la). We show that in contrast to the Turing model a general non-computability result is no longer valid. Under certain circumstances, we can even guarantee the computability of at least a function in $\mathcal{M}_{\Xi_{\mathrm{bp},m,N}}$ and $\mathcal{M}_{\Xi_{\mathrm{la},m,N}}$. To that end, we consider real and complex domains separately, since the BSS model exhibits different behavior depending on the underlying structure. If not stated otherwise, the proofs of the subsequent theorems are presented in Section 5.

4.2. Real inverse problems

First, we consider basis pursuit optimization (bp). In the Turing model, for fixed dimension $m, N \in \mathbb{N}$, $m < N$, and any relaxation parameter $\varepsilon \in (0, 1/4)$ algorithmic non-solvability of $\Xi_{\mathrm{bp},m,N}(\cdot,\cdot,\varepsilon)$ has already been established [19]. Moreover, the proof of the statement implies that non-computability remains valid in the strictly real case, i.e., the problem posed by $\Xi^{\mathbb{R}}_{\mathrm{bp},m,N}(\cdot,\cdot,\varepsilon)$ is not algorithmically solvable, whereby for a multi-valued mapping $f : \mathcal{X} \rightrightarrows \mathcal{Y}$, $f^{\mathbb{R}}$ denotes its restriction to real inputs and outputs. Unlike the Turing model, we can derive BSS-computability in the more general setting of $\Xi^{\mathbb{R}}_{\mathrm{bp},m,N}$ with arbitrary regularization parameter.

Theorem 4.3. *There exists a BSS-computable function* $\Xi^s \in \mathcal{M}_{\Xi^{\mathbb{R}}_{bp,m,N}}$.

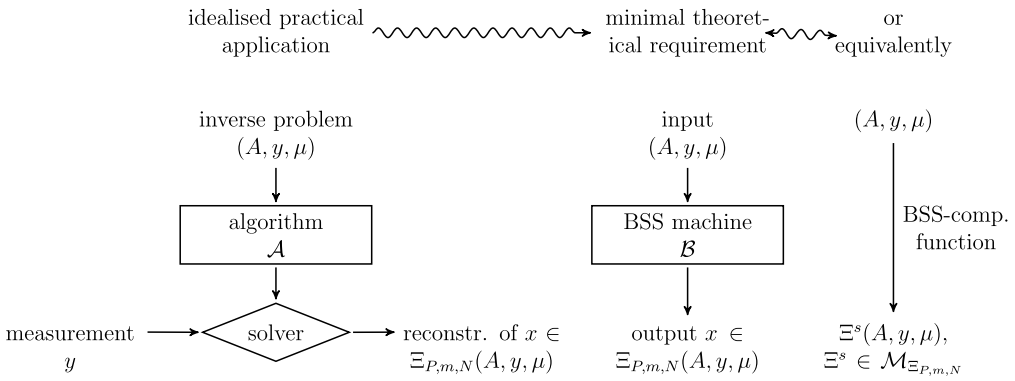


Fig. 1. Our goal is to study the existence of an algorithm $\mathcal{A}$ on an analog computing device which tackles inverse problem described by an optimization process P. In particular, $\mathcal{A}$ takes the sampling operator $A \in \mathbb{C}^{m\times N}$, measurement $y \in \mathbb{C}^m$, and optimization parameter $\mu > 0$ and generates a solver which reconstructs data $x \in \Xi_{P,m,N}(A, y, \mu)$ given measurement y. In an abstract mathematical description, the setting boils down to the existence of a BSS machine $\mathcal{B}$ taking inputs (A, y, μ) and outputting x. The existence of this BSS machine requires that there exists at least one BSS-computable function $\Xi^s \in \mathcal{M}_{\Xi_{P,m,N}}$.

Remark 4.4. We can conclude that inverse problems described by $\Xi^{\mathbb{R}}_{\mathrm{bp},m,N}$ are algorithmically solvable in the BSS model. The special case for fixed $\varepsilon > 0$ is included in Theorem 4.3. Hence, inverse problems via basis pursuit minimization (bp) are algorithmically solvable on BSS machines but not on Turing machines.

Theorem 4.3 implies the existence of an algorithm (in the BSS sense), which for every sampling operator A and measurement y yields a feasible reconstruction x. Is this algorithm connected to specific properties of basis pursuit minimization or can it be adapted to inverse problems with different regularization schemes as well? It turns out that algorithmic solvability is also achievable in the lasso formulation (la).

Theorem 4.5. *There exists a BSS-computable function* $\Xi^s \in \mathcal{M}_{\Xi^{\mathbb{R}}_{\mathrm{la},m,N}}$.

Remark 4.6. We can infer that inverse problems described by $\Xi^{\mathbb{R}}_{\mathrm{la},m,N}$ are algorithmically solvable in the BSS model, whereas it was shown in [19] that inverse problems via square root lasso optimization are not algorithmically solvable on Turing machines for a certain range of fixed optimization parameter λ. Square root lasso optimization is defined as

$$\arg\min_{x\in\mathbb{C}^N} \lambda \|x\|_{\ell_1} + \|Ax - y\|_{\ell_2}, \tag{sla}$$

i.e., the only difference to lasso (la) is the power of the ℓ_2 norm. In particular, the analogous proof technique can be applied to derive algorithmic non-solvability of lasso minimization. Thus, we also obtain different capabilities of Turing and BSS machines for solving inverse problems via lasso minimization. However, square root lasso optimization cannot be directly tackled on BSS machines due to the non-computability of the square root function connected to the ℓ_2 norm. Hence, similar modifications as described in Section 4.3 for complex inverse problems must be applied to circumvent this problem.

To summarize, a clear distinction exists between the algorithmic solvability of inverse problems on Turing and BSS machines restricted to the real domain. In the BSS model, we can prove algorithmic solvability for various underlying optimization formulations such as basis pursuit and lasso, whereas Turing machines do not allow for algorithmic solvability of inverse problems via the same optimization problems [19]. Hence, the limitations arising in the Turing model, do not generally occur in the BSS model for solving inverse problems algorithmically.

4.3. Complex inverse problems

The study of computability is more evolved in the complex domain. The reason is that BSS machines over complex fields have different properties than BSS machines over real numbers. Complex BSS machines treat complex numbers as entities whereas real BSS machines rely on real numbers as basic entities. One intrinsic difference is that complex numbers do not form an ordered field. Thus, complex BSS machines cannot compare arbitrary complex numbers but only check the equality to zero at their branch nodes. Moreover, even basic functions connected to complex numbers such as $z \mapsto \Re(z)$, $z \mapsto \Im(z)$, $z \mapsto \bar{z}$, and $z \mapsto |z|$ are not BSS-computable which follows from the fact that $\mathbb{R}$ is not a BSS-decidable set in $\mathbb{C}$ [14]. Therefore, ℓ_p norms are not BSS-computable on complex inputs as they require to compute the absolute value of the inputs. A possible workaround is representing $\mathbb{C}$ by $\mathbb{R}^2$ and applying operations on real numbers. That is, we consider the real model and employ real BSS machines that take complex inputs x in the form of $(\Re(x), \Im(x))$. Then, the functions $z \mapsto \Re(z)$ and $z \mapsto \Im(z)$ are computable on a real BSS machine since it only requires the machine

to process the respective part of the representation of z. Hence, the representation of the complex field influences the capabilities of the corresponding BSS machine. Unfortunately, this is still not sufficient to ensure the computability of the ℓ_1 norm

$$\|x\|_{\ell_1} = \sum_i |x_i| = \sum_i \sqrt{\mathfrak{R}(x_i)^2 + \mathfrak{I}(x_i)^2}$$

on real BSS machines, since the square root function $\sqrt{\cdot}$ is not BSS-computable on $\mathbb{R}$. We wish to remark that the non-computability of the square root function on BSS machines assumes exact computations, which we consider throughout this work. By relaxing to approximate computations the non-computability on BSS machines may be avoided. Similarly, on Turing machines the square root function can also not be computed exactly, even on a discrete set such as $\mathbb{N}$ since the irrational number $\sqrt{2}$ cannot be represented exactly on a Turing machine. However, the square root function can be approximated to an arbitrary degree on Turing machines, i.e., it is a Turing-computable function on $\mathbb{R}$.

Nevertheless, we can observe that slight modifications suffice to obtain a real BSS-computable function. Replacing the ℓ_1 by the squared ℓ_2 norm yields a BSS-computable function, since

$$\|x\|_{\ell_2}^2 = \sum_i |x_i|^2 = \sum_i \mathfrak{R}(x_i)^2 + \mathfrak{I}(x_i)^2$$

can be expressed by elementary field operations and the real BSS-computable functions $\mathfrak{R}$ and $\mathfrak{I}$. This is no exception, similar arguments can be made for any ℓ_p norm when p is even. The basis pursuit formulation in (bp) typically promotes sparse solutions due to the ℓ_1 norm in the objective function. This property is neglected when replacing the ℓ_1 with the squared ℓ_2 norm in the objective. Hence, a potentially BSS-computable substitute in the objective ideally remains close to the original objective function so desired properties are maintained. Therefore, we introduce a norm $\|\cdot\|_*$ which satisfies these requirements and also emphasizes the difference between the capabilities of BSS and Turing machines. We define $\|\cdot\|_*$ on $\mathbb{C}^N$ by

$$\|x\|_* := \sum_{i=1}^{N} |\mathfrak{R}(x_i)| + |\mathfrak{I}(x_i)|$$

Next, we show that by changing the objective in basis pursuit (bp) from ℓ_1 norm to $\|\cdot\|_*$ we obtain an algorithmically solvable optimization problem.

Theorem 4.7. *For $A \in \mathbb{C}^{m\times n}$, $y \in \mathbb{C}^m$ and $\varepsilon > 0$ consider the optimization problem*

$$\arg\min_{x\in\mathbb{C}^N} \|x\|_* \text{ such that } \|Ax - y\|_{\ell_2} \le \varepsilon. \tag{bp*}$$

Then, there exists a BSS-computable function $\Xi^s \in \mathcal{M}_{\Xi_{bp^*,m,N}}$.

Remark 4.8. For representations of complex numbers not based on real and imaginary parts, the result of the theorem does not necessarily translate to the corresponding BSS machines.

With the same adjustment, i.e., replacing the $\|\cdot\|_{\ell_1}$ by $\|\cdot\|_*$, and the analogous proof technique we obtain a similar result for lasso optimization (la).

Theorem 4.9. *For $A \in \mathbb{C}^{m\times n}$, $y \in \mathbb{C}^m$ and $\lambda > 0$ consider the optimization problem*

$$\arg\min_{x\in\mathbb{C}^N} \lambda \|x\|_* + \|Ax - y\|_{\ell_2}^2. \tag{la*}$$

Then, there exists a BSS-computable function $\Xi^s \in \mathcal{M}_{\Xi_{la^*,m,N}}$.

Remark 4.10. We can conclude that inverse problems expressed through the optimization problems (bp*) and (la*) with the modified objective $\|\cdot\|_*$ are algorithmically solvable in the BSS model. Hence, there exist algorithms in the BSS sense that for every sampling operator A and measurement y yield a feasible reconstruction $x \in \Xi_{\mathrm{bp}^*,m,N}$ and $x \in \Xi_{\mathrm{la}^*,m,N}$, respectively.

Several questions concerning our results arise. In particular, we aim to relate our findings to the algorithmic solvability of inverse problems on Turing machines.

- First, how do the modifications of the optimization problems influence the outcomes in the Turing model? Applying the same proof technique as in [19] also yields algorithmic non-solvability in the Turing model for the adjusted description in (bp*) and (la*). We demonstrate the proof technique in Appendix A for (bp*).
- Second, can we find similar modifications of the optimization problems that allow us to solve inverse problems successfully on a Turing machine? Here, the answer appears to be negative. The reasoning for algorithmic non-solvability of basis pursuit and lasso in the Turing model was not connected to the utilized objective function. Rather, non-computability conditions were established

that are fairly independent of the objective or, more specifically, can be easily adapted to certain changes in the objective function. The requirements to apply the non-computability conditions are mainly connected to the properties of the solution set of a given problem. If the solution set satisfies certain properties, the non-computability statement can also be extended to the corresponding problem formulation. Hence, as long as the modified objective function does not promote substantial changes in the solution set the non-computability statement remains valid. Moreover, changing the objective entirely may result in a quality loss in the computed solutions since the desired properties such as sparsity may be no longer encouraged by the new objective.

- Our final question directly relates to the previous discussion: Can we justify the changes in the objective function resulting in the optimization problems (bp*) and (la*)? In particular, how do (bp*) and (la*) relate to the original problems in (bp) and (la), respectively? Although these adjustments would typically not be applied to solve inverse problems in practice, the approach tries to maintain the underlying properties of the original formulations. Hence, it is striking that fairly similar inverse problem descriptions exist, which can be successfully solved in the BSS model.

Nevertheless, rigorously establishing modifications of the optimization problems is desirable. Therefore, we aim to approximate instead of replacing the ℓ_1 norm. Next, this strategy is exemplarily demonstrated for the basis pursuit problem. The proof is provided in Appendix B.

Theorem 4.11. *Let $\beta, \gamma > 0$. For $A \in \mathbb{C}^{m\times N}$, $y \in \mathbb{C}^m$ and $\varepsilon > 0$ consider the optimization problem*

$$\arg\min_{x\in\mathbb{C}^N} p_{\beta,\gamma}(x) \text{ such that } \|Ax - y\|_{\ell_2} \leq \varepsilon \ \wedge \ \|x\|_{\ell_2} < \sqrt{N}\beta, \tag{p(β,γ)}$$

where $p_{\beta,\gamma}$ is a polynomial satisfying

$$\sup_{x\in\mathbb{C}^N : \|x\|_{\ell_2} < \sqrt{N}\beta} \left| \|x\|_{\ell_1} - p_{\beta,\gamma}(x) \right| \leq \gamma. \tag{4.3}$$

Then, there exists a BSS-computable function $\Xi^s \in \mathcal{M}_{\Xi_{\mathrm{p}(\beta,\gamma),m,N}}$. Moreover, there also exists a BSS-computable function $g : \mathbb{C}^{m\times N} \times \mathbb{C}^m \times \mathbb{R}_{>0} \times \mathbb{R}_{>0} \to \{0,1\}$ so that $g(A,y,\varepsilon,\beta) = 1$ is equivalent to the following statement: For (A,y,ε) there exists at least one solution of basis pursuit (bp) and the solution(s) are contained in $I_\beta := \{x \in \mathbb{C}^N : \|x\|_{\ell_2} < \sqrt{N}\beta\}$.

Remark 4.12. We can conclude that inverse problems described by $\Xi_{\mathrm{p}(\beta,\gamma),m,N}$ are algorithmically solvable in the BSS model. Note that the objective $p_{\beta,\gamma}$ in (p(β,γ)) approximates up to an error of γ the ℓ_1 norm, i.e., the objective of the original basis pursuit optimization, on the set I_β. Therefore, (p(β,γ)) represents an approximation of basis pursuit if its minimizers are contained in I_β. This property can be checked via the function g in Theorem 4.11, i.e., before evaluating $\Xi^s \in \mathcal{M}_{\Xi_{\mathrm{p}(\beta,\gamma),m,N}}$ for given (A,y,ε) we can algorithmically verify if $\Xi_{\mathrm{bp},m,N}(A,y,\varepsilon) \subset I_\beta$. However, the minimizers of (p(β,γ)) and basis pursuit must not agree and we do not obtain worst-case bounds on their distance.

Remark 4.13. The algorithmic solvability of $\Xi_{\mathrm{p}(\beta,\gamma),m,N}$ and the existence of g imply that there exists a BSS machine $\mathcal{B}_{\beta,\gamma}$ that computes a solution of (p(β,γ)) given $A \in \mathbb{C}^{m\times n}$, $y \in \mathbb{C}^m$ and $\varepsilon > 0$ as input, provided that (A,y,ε) satisfy certain properties. In particular, $\mathcal{B}_{\beta,\gamma}$ first checks if the solutions of basis pursuit for (A,y,ε) are contained in I_β. In this way, we can ensure that the approximation of the ℓ_1 norm by $p_{\beta,\gamma}$ is valid on the relevant domain. Otherwise, the computation is aborted. If not, a solution of (p(β,γ)) for (A,y,ε) is computed. Note that $\mathcal{B}_{\beta,\gamma}$ needs to be constructed for fixed approximation accuracy γ and fixed acceptance domain depending on I_β, i.e., β and γ are not part of the input to the machine. In particular, we need to encode a finite set of constants, which depend on β and γ but cannot be computed by a BSS machine, that enable $\mathcal{B}_{\beta,\gamma}$ to compute an appropriate polynomial $p_{\beta,\gamma}$ and subsequently a minimizer of (p(β,γ)). Therefore, changes in the acceptance domain and approximation accuracy entail a new construction of the associated BSS machines $\mathcal{B}_{\beta,\gamma}$.

Remark 4.14. In the Turing model, the outlined approach to approximate basis pursuit is not feasible; we refer to Appendix B for details. Nevertheless, different approximation schemes may exist for Turing machines. However, we have lower bounds on the accuracy an algorithm executed on a Turing machine can approximate the minimizers of basis pursuit [19].

5. Proof section

We base our proofs on a connection between BSS machines and semialgebraic sets. After exploring the link between the two concepts in more detail, we use it to show the statements in the previous section.

5.1. Semialgebraic sets

The concept of semialgebraic sets, i.e., sets defined by polynomial equations and inequalities, is key for BSS-decidability. Every BSS-semidecidable set in $\mathbb{R}^n$ can be expressed as a countable union of semialgebraic sets [14]. This follows from the algebraic structure of the available basic operations in BSS-algorithms. For an introduction and discussion of semialgebraic sets, we refer to [24]. We follow mainly [8] when introducing this topic.

Definition 5.1. The *class of semialgebraic sets* in $\mathbb{R}^n$ is the smallest class of subsets of $\mathbb{R}^n$ that contains all sets $\{x \in \mathbb{R}^n : p(x) > 0\}$ with real polynomials $p : \mathbb{R}^n \to \mathbb{R}$ and is in addition closed under finite intersections and unions as well as complements.

Remark 5.2. Any semialgebraic set in $\mathbb{R}^n$ can be expressed as the finite union of sets of the form $\{x \in \mathbb{R}^n \mid p(x) = 0 \wedge \bigwedge_{q \in Q} q(x) > 0\}$ with a finite set $p \cup Q$ of real polynomials $p, q : \mathbb{R}^n \to \mathbb{R}$.

The Tarski-Seidenberg theorem [93,102] states that any projection map on $\mathbb{R}^n \to \mathbb{R}^m$, where $n \geq m$, projects semialgebraic sets in $\mathbb{R}^n$ onto semialgebraic sets in $\mathbb{R}^m$. Consequently, quantifier elimination is possible over $\{\mathbb{R}, +, \cdot, 0, 1, >\}$. This means that every first-order formula built over $\{\mathbb{R}, +, \cdot, 0, 1, >\}$, i.e., a formula involving polynomials with logical operations "$\wedge$", "$\vee$", and "$\neg$" and quantifiers "$\forall$", "$\exists$", can be algorithmically transformed into an equivalent quantifier-free formula. Thus, an algorithm exists that, given an arbitrary formula (with quantifiers) as input, computes an equivalent formula without quantifiers. This allows us to eliminate all quantifiers in semialgebraic sets algorithmically.

For instance, the projection theorem implies that a semialgebraic set defined by formulas of the form

$$\{(x_1, \dots, x_n) \in \mathbb{R}^n : \exists x_{n+1} \text{ such that } p(x_1, \dots, x_n, x_{n+1}) \,\Delta\, 0\}$$

can be rewritten as a semialgebraic set defined by formulas of the form

$$\{(x_1, \dots, x_n) \in \mathbb{R}^n : q(x_1, \dots, x_n) \,\Delta\, 0\},$$

where $\Delta \in \{<, >, \leq, \geq, =\}$ and p, q are real polynomials.

Furthermore, Tarski found an algorithm to decide the truth of sentences, i.e., formulas that have no free variables, in the first-order language built from $\{\mathbb{R}, +, \cdot, 0, 1, >\}$ (cf. [43]). This algorithm directly results from the described transformation of semialgebraic sets with quantifiers into semialgebraic sets without quantifiers.

The complexity of the elimination process has been reduced significantly since the original algorithm proposed by Tarski, see [43] and references therein. However, for many applications in information theory, the computational complexity of quantifier elimination remains too high. In the case of computability results the complexity of the algorithm is not relevant (just its existence), hence quantifier elimination provides a useful theoretical tool. Tarski's algorithm can be computed on a BSS machine, whereas, the elimination of quantifiers is generally impossible on Turing machines.

Many problems in real geometry can be formulated as first-order formulas. Tarski-Seidenberg Elimination Theory can then be used to solve these problems. For our needs, the crucial observation is that, based on quantifier-elimination, it is possible to develop an algorithm for finding a minimizer of a polynomial on a semialgebraic set if there exists one. In this setting, an algorithm is a computational procedure that takes an input and produces an output after performing a finite number of admissible operations. The feasible operations are the ring or field operations of the considered structure and comparisons between elements provided the structure is ordered. By construction, such an algorithm can be executed on a BSS machine.

For further details such as the procedure of the algorithm, we refer to [8]. We only state the input and output relation of the algorithm for real polynomials for the convenience of the reader:

Algorithm 1 Global optimization.

Input: a finite set $\mathcal{P}$ of real polynomials $p : \mathbb{R}^n \to \mathbb{R}$ describing a (non-empty) semialgebraic set S by a quantifier-free formula Φ and a polynomial $f : \mathbb{R}^n \to \mathbb{R}$.
Output: the infimum w of f on S, and a minimizer, i.e. a point $x^* \in S$ such that $f(x^*) = w$ if such a point exists.

A special case of quantifier elimination implies that the non-emptiness of a semialgebraic set can be algorithmically decided in the BSS model. Hence, we can check whether the optimization domain is empty before calling the global optimization algorithm. In terms of complexity, the algorithm requires $|\mathcal{P}|^{2n+1} d^{\mathcal{O}(n)}$ operations to compute the minimizer, where d is a bound on the degree of the involved polynomials. Thus, informally speaking it does not qualify as a polynomial time algorithm in the input dimension n. For a more formal treatment of complexity on BSS machines, including the meaning of polynomial time in this context, we refer to [14], albeit we note that structural properties of the associated complexity class FP of functional problems solved in polynomial time on BSS machines were hardly studied and remain an open problem [6,14,32]. Finally, let us mention that these capabilities of BSS machines also highlight the differences from Turing machines. The analogous discrete problem, commonly known as Hilbert's tenth problem [61], of deciding whether a polynomial equation with integer coefficients and a finite number of unknowns possesses integer-valued roots is not solvable on Turing machines [76].

5.2. Proofs

The common proof technique for showing Theorem 4.3, Theorem 4.5, and Theorem 4.7 relies on rewriting the considered problems to apply Algorithm 1. This constructive solution strategy implies that a single-valued restriction of the multi-valued problem is BSS-computable. Hence, algorithmic solvability as claimed in the theorems is immediate once this constructive approach is verified.

5.2.1. Proof of Theorem 4.3

Lemma 5.3. *For $A \in \mathbb{R}^{m\times N}$, $y \in \mathbb{R}^m$, and $\varepsilon > 0$ set $S_{(A,y,\varepsilon)} := \{x \in \mathbb{R}^N \mid \|Ax - y\|_{\ell_2} \le \varepsilon\}$ and consider the multi-valued map*

$$\Psi : \mathbb{R}^{m\times N} \times \mathbb{R}^m \times \mathbb{R}_{\ge 0} \rightrightarrows \mathbb{R}^N$$
$$(A, y, \varepsilon) \mapsto \{x^* \in \mathbb{R}^N \mid \|x^*\|_{\ell_1} = \min_{x \in S_{(A,y,\varepsilon)}} \|x\|_{\ell_1} \text{ and } \|Ax^* - y\|_{\ell_2} \le \varepsilon\}.$$

Then, there exists a (single-valued) function $\Psi^s \in \mathcal{M}_\Psi$ that is BSS-computable.

Proof. We need to show that there exists a program for a BSS machine $\mathcal{B}$ so that its input-output map $\Phi_\mathcal{B}$ is equivalent to a function $\Psi^s \in \mathcal{M}_\Psi$. Given $A \in \mathbb{R}^{m\times N}$, $y \in \mathbb{R}^m$, and $\varepsilon > 0$ we therefore have to compute x^* with

$$x^* \in \operatorname*{arg\,min}_{x\in\mathbb{R}^N} \|x\|_{\ell_1} \text{ such that } \|Ax - y\|_{\ell_2} \le \varepsilon. \tag{5.1}$$

For this, let $A = (a_{i,j})$ and note that

$$\|Ax - y\|_{\ell_2} \le \varepsilon \quad \Leftrightarrow \quad \|Ax - y\|_{\ell_2}^2 \le \varepsilon^2 \quad \Leftrightarrow \quad \sum_{i=1}^m \Big(\sum_{j=1}^N a_{i,j}x_j - y_i\Big)^2 - \varepsilon^2 \le 0.$$

This implies

$$S_{(A,y,\varepsilon)} = \{x \in \mathbb{R}^N \mid \|Ax - y\|_{\ell_2} \le \varepsilon\} = \{x \in \mathbb{R}^N \mid q(x) \le 0\}$$

where $q : \mathbb{R}^N \to \mathbb{R}$ is a polynomial, i.e., $S_{(A,y,\varepsilon)}$ is a semialgebraic set with a quantifier-free description.

Now, our goal is to apply the optimization algorithm, Algorithm 1, to find a minimizer of a polynomial on $S_{(A,y,\varepsilon)}$. In its current form, $\|x\|_{\ell_1} = \sum_i |x_i|$ is not a polynomial, however, we can rewrite it as

$$\|x\|_{\ell_1} = \sum_i |x_i| = \sum_i x_i^+ + x_i^-$$

where $x_i^+ = \max\{0, x_i\}$ and $x_i^- = -\min\{0, x_i\}$. Observe that $p : \mathbb{R}^{2N} \to \mathbb{R}$, $(x^+, x^-) \mapsto \sum_i x_i^+ + x_i^-$ is indeed a polynomial.

Hence, we can conclude that (5.1) is equivalent to the problem

$$(w^*, z^*) \in \operatorname*{arg\,min}_{\substack{w,z\in\mathbb{R}^N,\\ w,z\ge 0}} \sum_{i=1}^N w_i + z_i \text{ such that } \|A(w - z) - y\|_{\ell_2} \le \varepsilon.$$

Now it is easy to deduce that the task is to minimize the polynomial $f : \mathbb{R}^{2N} \to \mathbb{R}$, $f(w, z) = \sum_{i=1}^N w_i + z_i$ on

$$\begin{aligned}&\{(w, z) \in \mathbb{R}^{2N} \mid \|A(w - z) - y\|_{\ell_2} \le \varepsilon, w \ge 0, z \ge 0\}\\ &= \{(w, z) \in \mathbb{R}^{2N} \mid q(w, z) \le 0 \wedge \bigwedge_{i=1}^N r_i(w, z) \ge 0 \wedge \bigwedge_{i=1}^N s_i(w, z) \ge 0, \}\end{aligned} \tag{5.2}$$

where $q, r_i, s_i : \mathbb{R}^{2N} \to \mathbb{R}$ are polynomials – q describes the ℓ_2 norm term whereas $r_i(w, z) = w_i, s_i(w, z) = z_i$ describe the non-negativity conditions. Consequently,

$$\{(w, z) \in \mathbb{R}^{2N} \mid \|A(w - z) - y\|_{\ell_2} \le \varepsilon, w \ge 0, z \ge 0\}$$

is a semialgebraic set with a quantifier-free description. Thus, we can apply the optimization algorithm, Algorithm 1, to minimize the polynomial f on a semialgebraic set and obtain a corresponding minimizer $(w^*, z^*) \in \mathbb{R}^{2N}$. By construction, the difference $x^* = w^* - z^*$ minimizes the original problem (5.1). Thus, we can define a program for $\mathcal{B}$ which, on input $A \in \mathbb{R}^{m\times N}$, $y \in \mathbb{R}^m$, and $\varepsilon > 0$, yields the polynomials q, r_i, s_i specified in (5.2), uses Algorithm 1 to obtain a minimizer (w^*, z^*) of f, and finally outputs the solution $x^* = w^* - z^*$. This completes the proof. □

5.2.2. Proof of Theorem 4.5

Lemma 5.4. *For $A \in \mathbb{R}^{m\times N}$, $y \in \mathbb{R}^m$, and $\lambda > 0$ consider the multi-valued map*

$$\Psi : \mathbb{R}^{m\times N} \times \mathbb{R}^m \times \mathbb{R}_{\ge 0} \rightrightarrows \mathbb{R}^N$$
$$(A, y, \varepsilon) \mapsto \{x^* \in \mathbb{R}^N \mid \|x^*\|_{\ell_1} = \min_{x\in\mathbb{R}^N} \lambda\|x\|_{\ell_1} + \|Ax - y\|_{\ell_2}^2\}.$$

Then, there exists a (single-valued) function $\Psi^s \in \mathcal{M}_\Psi$ that is BSS-computable.

Proof. We need to show that there exists a program for a BSS machine $\mathcal{B}$ so that its input-output map $\Phi_{\mathcal{B}}$ is equivalent to a function $\Psi^s \in \mathcal{M}_\Psi$. Given $A \in \mathbb{R}^{m\times N}$, $y \in \mathbb{R}^m$, and $\varepsilon > 0$ we therefore have to compute x^* with

$$x^* \in \operatorname*{arg\,min}_{x\in\mathbb{R}^N} \lambda \|x\|_{\ell_1} + \|Ax - y\|_{\ell_2}^2 . \tag{5.3}$$

For this, let $A = (a_{i,j})$ and note that

$$\begin{aligned}\lambda \|x\|_{\ell_1} + \|Ax - y\|_{\ell_2}^2 &= \lambda \sum_i |x_i| + \sum_{i=1}^m \Big(\sum_{j=1}^N a_{i,j} x_j - y_i \Big)^2 \\ &= \lambda \sum_i x_i^+ + x_i^- + \sum_{i=1}^m \Big(\sum_{j=1}^N a_{i,j} \big(\sum_i x_i^+ - x_i^-\big) - y_i \Big)^2\end{aligned}$$

where $x_i^+ = \max\{0, x_i\}$ and $x_i^- = -\min\{0, x_i\}$. This implies that a solution of (5.3) can be obtained via

$$(w^*, z^*) \in \operatorname*{arg\,min}_{\substack{w,z\in\mathbb{R}^N,\\ w,z\geq 0}} \lambda \sum_i w_i^* + z_i^* + \sum_{i=1}^m \Big(\sum_{j=1}^N a_{i,j} \big(\sum_i w_i^* - z_i^*\big) - y_i \Big)^2. \tag{5.4}$$

as $x^* = w^* - z^*$. Now, consider the polynomial $f : \mathbb{R}^{2N} \to \mathbb{R}$ given by

$$f(w, z) = \lambda \sum_i w_i + z_i + \sum_{i=1}^m \Big(\sum_{j=1}^N a_{i,j} \big(\sum_i w_i - z_i\big) - y_i \Big)^2$$

and the semialgebraic set with a quantifier-free description

$$\begin{aligned}&\{(w, z) \in \mathbb{R}^{2N} \mid w \geq 0, z \geq 0\} \\ &= \{(w, z) \in \mathbb{R}^{2N} \bigwedge_{i=1}^N r_i(w, z) \geq 0 \wedge \bigwedge_{i=1}^N s_i(w, z) \geq 0\},\end{aligned}$$

where $r_i, s_i : \mathbb{R}^{2N} \to \mathbb{R}$, $r_i(w, z) = w_i, s_i(w, z) = z_i$ are polynomials describing the non-negativity conditions. We can apply the optimization algorithm, Algorithm 1, to find a minimizer $(w^*, z^*) \in \mathbb{R}^{2N}$ of f on the semialgebraic set specified by the polynomials r_i, s_i. By construction (w^*, z^*) is a solution of (5.4) and the difference $x^* = w^* - z^*$ is a sought solution of (5.3). Consequently, there exists a program for $\mathcal{B}$ which, on input $A \in \mathbb{R}^{m\times N}$, $y \in \mathbb{R}^m$, and $\lambda > 0$, yields the polynomials r_i, s_i and uses Algorithm 1 to obtain a minimizer (w^*, z^*) of f, and finally outputs the solution x^*. This completes the proof. □

5.2.3. *Proof of Theorem 4.7*

Lemma 5.5. *For $A \in \mathbb{C}^{m\times N}$, $y \in \mathbb{C}^m$, and $\varepsilon > 0$ set $S_{(A,y,\varepsilon)} := \{x \in \mathbb{C}^N \mid \|Ax - y\|_{\ell_2} \leq \varepsilon\}$ and consider the multi-valued map*

$$\begin{aligned}\Psi : \mathbb{C}^{m\times N} \times \mathbb{C}^m \times \mathbb{R}_{\geq 0} &\rightrightarrows \mathbb{C}^N \\ (A, y, \varepsilon) &\mapsto \{x^* \in \mathbb{C}^N \mid \|x^*\|_* = \min_{x\in S_{(A,y,\varepsilon)}} \|x\|_* \text{ and } \|Ax^* - y\|_{\ell_2} \leq \varepsilon\}.\end{aligned}$$

Then, there exists a (single-valued) function $\Psi^s \in \mathcal{M}_\Psi$ that is BSS-computable.

Proof. We need to show that there exists a program for a real BSS machine $\mathcal{B}$ so that its input-output map $\Phi_{\mathcal{B}}$ is equivalent to a function $\Psi^s \in \mathcal{M}_\Psi$. Given $A \in \mathbb{C}^{m\times N}$, $y \in \mathbb{C}^m$, and $\varepsilon > 0$, whereby each complex element is represented by its real and imaginary part, we have to compute x^* with

$$x^* \in \operatorname*{arg\,min}_{x\in\mathbb{C}^N} \|x\|_* \text{ such that } \|Ax - y\|_{\ell_2} \leq \varepsilon. \tag{5.5}$$

Let $A = (a_{i,j})$ and note that

$$\|Ax - y\|_{\ell_2} \leq \varepsilon \quad \Leftrightarrow \quad \|Ax - y\|_{\ell_2}^2 \leq \varepsilon^2 \quad \Leftrightarrow \quad \sum_{i=1}^m \left(\left| \sum_{j=1}^N a_{i,j} x_j - y_i \right| \right)^2 - \varepsilon^2 \leq 0 \tag{5.6}$$

and for fixed $i \in \{1, \dots, m\}$

$$\left(\left| \sum_{j=1}^N a_{i,j} x_j - y_i \right| \right)^2 = \left(\Re \left(\sum_{j=1}^N a_{i,j} x_j - y_i \right) \right)^2 + \left(\Im \left(\sum_{j=1}^N a_{i,j} x_j - y_i \right) \right)^2$$

$$= \left(\sum_{j=1}^{N} \mathfrak{R}(a_{i,j})\mathfrak{R}(x_j) - \mathfrak{I}(a_{i,j})\mathfrak{I}(x_j) - \mathfrak{R}(y_i) \right)^2 + \left(\sum_{j=1}^{N} \mathfrak{R}(a_{i,j})\mathfrak{I}(x_j) + \mathfrak{I}(a_{i,j})\mathfrak{R}(x_j) - \mathfrak{I}(y_i) \right)^2 . \tag{5.7}$$

Combining (5.6) and (5.7) yields

$$\|Ax - y\|_{\ell_2} \leq \varepsilon \quad \Leftrightarrow \quad \sum_{i=1}^{m} \left(\sum_{j=1}^{N} \mathfrak{R}(a_{i,j})\mathfrak{R}(x_j) - \mathfrak{I}(a_{i,j})\mathfrak{I}(x_j) - \mathfrak{R}(y_i) \right)^2 + \left(\sum_{j=1}^{N} \mathfrak{R}(a_{i,j})\mathfrak{I}(x_j) + \mathfrak{I}(a_{i,j})\mathfrak{R}(x_j) - \mathfrak{I}(y_i) \right)^2 - \varepsilon^2 \leq 0,$$

i.e., we can identify the set $\{x \in \mathbb{C}^N \,|\, \|Ax - y\|_{\ell_2} \leq \varepsilon\}$ with $\{x \in \mathbb{R}^{2N} | q(x) \leq 0\}$ where $q : \mathbb{R}^{2N} \to \mathbb{R}$ is a polynomial. Hence, $\{x \in \mathbb{C}^N \,|\, \|Ax - y\|_{\ell_2} \leq \varepsilon\}$ can be represented by a (real) semialgebraic set with a quantifier-free description. To use the optimization algorithm, Algorithm 1, it remains to show that the objective function in (5.5) is a real polynomial $f : \mathbb{R}^{2N} \to \mathbb{R}$.

However, the objective function $\|\cdot\|_*$ is not a polynomial (due to the absolute values in its definition). By applying the same workaround as in the proof of Lemma 5.3 – rewriting the absolute value as the sum of two variables – we obtain that

$$\|x\|_* = \sum_i \left|\mathfrak{R}(x_i)\right| + \left|\mathfrak{I}(x_i)\right| = \sum_i \mathfrak{R}(x_i)^+ + \mathfrak{R}(x_i)^- + \mathfrak{I}(x_i)^+ + \mathfrak{I}(x_i)^-,$$

where $x_i^+ = \max\{0, x_i\}$ and $x_i^- = -\min\{0, x_i\}$. Thus, $\|\cdot\|_*$ can be represented by a polynomial $p : \mathbb{R}^{4N} \to \mathbb{R}$,

$$p(\mathfrak{R}(x_i)^+, \mathfrak{R}(x_i)^-, \mathfrak{I}(x_i)^+, \mathfrak{I}(x_i)^-) = \sum_i \mathfrak{R}(x_i)^+ + \mathfrak{R}(x_i)^- + \mathfrak{I}(x_i)^+ + \mathfrak{I}(x_i)^-.$$

Therefore, the same reasoning as in the real case shows that a minimizer $\hat{x} \in \mathbb{R}^{2N}$ of $\|\cdot\|_*$ on $\{x \in \mathbb{R}^{2N} | q(x) \leq 0\}$ can be algorithmically computed. The sought minimizer $x^* \in \mathbb{C}^N$ of (5.5) is then simply $\hat{x}$, whereby the elements in $\hat{x}$ are considered as the real and imaginary parts of x^*. □

6. Discussion

Our findings have noteworthy implications concerning the solvers of finite-dimensional inverse problems. In particular, they allow us to characterize the boundaries of any general algorithm applied to solve this class of inverse problems. More precisely, we examined the limits of any algorithmic computation implemented on a specific computing device imposed by the hardware. Thus, any solver implemented on the given computing device is subject to these restrictions. On digital hardware, which is the currently predominant hardware platform, the algorithmic solvability via basis pursuit (bp) and lasso (la) is limited [7,19,41]. Our findings indicate less restrictive characteristics on analog hardware described by the mathematical model of BSS machines. First, algorithmic solvability on BSS machines of inverse problems can be established when restricting to the real domain. Furthermore, we can find modifications (bp*), (la*), and (p(β, γ)) that allow for algorithmic solvability of complex inverse problems on BSS machines while staying 'close' to the original basis pursuit and lasso problems. At the same time, the algorithmic non-solvability on Turing machines, i.e., digital computers, persists. Hence, the utilized computing device strongly impacts the existence and capabilities of solvers applied to inverse problems. What are the effects of these results on current solution techniques?

Today's solvers are in general implemented on digital hardware. Thus, they are subject to the limitations of digital hardware and even improving the solvers will not suffice to circumvent the imposed boundaries. Therefore, DL on digital hardware as the premium approach to solving inverse problems is subject to these limitations. This imposes restrictions on the reliability and trustworthiness of DL on digital hardware for solving inverse problems. In particular, a general and reliable DL solver for inverse problems cannot exist if formal proof of correctness or a certificate for the output is expected. Future developments in hardware technology (as described in Section 1.1) to analog hardware may enable more powerful solvers. Here, the question is whether the BSS model is an appropriate model to describe the capabilities of the upcoming analog hardware solution. Although a BSS machine is certainly an idealized model, due to exact real number processing, an important point is whether and to what degree these models can be implemented and approximated. In particular, the effects of noisy computations and the realization of approximative computing must be evaluated. For now, in an idealized theoretical setting modeled by BSS machines, analog hardware offers a platform for potentially reliable DL.

This is even more relevant since recently policymakers proposed guidelines and regulations for AI models including DL to counter their lack of trustworthiness and the resulting potential for harmful outcomes. Among the most influential ones are the European AI Act and the G7 Hiroshima Leaders Communiqué with demands such as algorithmic transparency, i.e., comprehensible and transparent algorithmic computations in AI [50,54]. For inverse problems, we can conclude that strict conformance with algorithmic transparency may ultimately hinge on the hardware platform [20]. In the bigger picture, the need for trustworthiness and reliability goes beyond AI applications – any technology interfering with the critical infrastructure of our modern society is affected [51,52]. Hence, understanding the capabilities and limits of computing platforms is crucially importance.

Acknowledgments

This work has received funding by the Bavarian State Ministry of Science and the Arts as part of the project GAIn "Next Generation AI Computing".

H. Boche was supported in part by the German Federal Ministry of Education and Research (BMBF) in the project Hardware Platforms and Computing Models for Neuromorphic Computing (NeuroCM) under Grant 16ME0442 and within the national initiative on 6G Communication Systems through the research hub 6G-life under Grant 16KISK002.

G. Kutyniok acknowledges support from LMUexcellent, funded by the Federal Ministry of Education and Research (BMBF) and the Free State of Bavaria under the Excellence Strategy of the Federal Government and the Länder as well as by the Hightech Agenda Bavaria. Further, G. Kutyniok was supported in part by the DAAD programme Konrad Zuse Schools of Excellence in Artificial Intelligence, sponsored by the Federal Ministry of Education and Research. G. Kutyniok also acknowledges support from the Munich Center for Machine Learning (MCML) as well as the German Research Foundation under Grants DFG-SPP-2298, KU 1446/31-1 and KU 1446/32-1 and under Grant DFG-SFB/TR 109, Project C09 and the German Federal Ministry of Education and Research (BMBF) under Grant MaGriDo.

Appendix A. Algorithmic non-solvability in the Turing model

In this section, we show that inverse problems expressed through the optimization problem

$$\arg\min_{x\in\mathbb{C}^N} \|x\|_* \text{ such that } \|Ax-y\|_{\ell_2} \le \varepsilon, \tag{bp*}$$

are not algorithmically solvable in the Turing model, where the norm $\|\cdot\|_*$ is given by

$$\|x\|_* := \sum_{i}^{N} \left|\mathfrak{R}(x_i)\right| + \left|\mathfrak{I}(x_i)\right|.$$

Theorem A.1. *For $\varepsilon \in (0,1)$, the problem described by $\Xi_{bp^*,m,N,\varepsilon}$ is not algorithmically solvable on a Turing machine.*

First, we introduce the necessary notions describing Turing-computable functions.

A.1. Preliminaries from computation theory

We refer to [5,86,99,107] for a detailed introduction to the topic. Here, we only provide a concise overview, starting with the computability of (real) numbers on Turing machines.

Definition A.2. A sequence $(r_k)_{k\in\mathbb{N}}$ of *rational numbers* is *computable*, if there exist three recursive functions $a, b, s : \mathbb{N} \to \mathbb{N}$ such that $b(k) \neq 0$ for all $k \in \mathbb{N}$ and

$$r_k = (-1)^{s(k)} \frac{a(k)}{b(k)} \qquad \text{for all } k \in \mathbb{N}.$$

Definition A.3.

(1) A number $x \in \mathbb{R}$ is computable if there exists a computable sequence of rational numbers $(r_k)_{k\in\mathbb{N}}$ such that

$$\left|r_k - x\right| \le 2^{-k} \qquad \text{for all } k \in \mathbb{N}.$$

(2) A *vector* $v \in \mathbb{R}^n$ is *computable* if each of its components is computable.
(3) A sequence $(x_n)_{n\in\mathbb{N}} \subset \mathbb{R}$ is *computable* if there exists a computable double-indexed sequence $\{r_{n,k}\}_{n,k\in\mathbb{N}} \subset \mathbb{Q}$ such that

$$\left|r_{n,k} - x_n\right| \le 2^{-k} \qquad \text{for all } k, n \in \mathbb{N}.$$

Next, we will define the computability of a function via Banach-Mazur computability. This is in fact the weakest form of computability on a Turing machine in our setting, i.e., if a function is not Banach–Mazur computable, it is not computable with respect to any other reasonable notion of computability on a Turing machine.

Definition A.4 *(Banach-Mazur computability).* A function $f : I \to \mathbb{R}_c^n$, $I \subset \mathbb{R}_c^m$, is said to be *Banach-Mazur computable*, if f maps computable sequences $(t_n)_{n\in\mathbb{N}} \subset I$ onto computable sequences $(f(t_n))_{n\in\mathbb{N}} \subset \mathbb{R}_c^n$.

A.1.1. Proof sketch

To prove algorithmic non-solvability in Theorem A.1, we will apply non-approximability conditions for optimization problems that were introduced in [19] following a proof technique established in [41].

Lemma A.5. *Consider for the optimization problem P with optimization parameter $\mu > 0$ the multi-valued mapping $\Xi_{P,m,N,\mu}$ defined by*

$$\Xi_{P,m,N,\mu} : \mathbb{C}^{m\times N} \times \mathbb{C}^m \rightrightarrows \mathbb{C}^N$$
$$(A, y) \mapsto P(A, y, \mu).$$

Choose an arbitrary single-valued restriction $\Psi^s \in \mathcal{M}_{\Xi_{P,m,N,\mu}}$ and $\Omega \subseteq \mathrm{dom}(\Psi^s) = \{\omega = (A, y) \in \mathbb{C}^{m\times N} \times \mathbb{C}^m \mid P(\omega, \mu) \neq \emptyset\}$. Further, suppose that there are two computable sequences $(\omega_n^1)_{n\in\mathbb{N}}, (\omega_n^2)_{n\in\mathbb{N}} \subset \Omega$ satisfying the following conditions:

(a) There are sets $S^1, S^2 \subset \mathbb{C}^N$ and $\kappa > 0, \kappa \in \mathbb{Q}$ such that $\inf_{x_1\in S^1, x_2\in S^2} \|x_1 - x_2\|_{\ell_2} > \kappa$ and $\Psi^s(\omega_n^j) \in S^j$ for $j = 1, 2$.

(b) There exists $\omega^* \in \Omega$ such that $\left\|\omega_n^j - \omega^*\right\|_{\ell_2} \leq 2^{-n}$ for all $n \in \mathbb{N}$, $j = 1, 2$.

In addition, let $\Psi : \Omega_\Psi \to \mathbb{C}^N$, $\Omega \subset \Omega_\Psi \subset \mathbb{C}^{m\times N} \times \mathbb{C}^m$, be an arbitrary function with

$$\sup_{\omega\in\Omega} \|\Psi^s(\omega) - \Psi(\omega)\|_{\ell_2} < \frac{\kappa}{8}.$$

Then Ψ is not Banach-Mazur computable.

Verifying the assumptions of Lemma A.5 for $\varepsilon \in (0, 1)$ is sufficient to obtain Theorem A.1. We will only sketch the approach to demonstrate algorithmic non-solvability in the Turing model, however, a more thorough analysis also yields algorithmic non-approximability for a certain range of optimization parameters. The construction of the sequences satisfying conditions (a) and (b) in Lemma A.5 relies on a characterization of the solution set. The same properties of the solution set as in the original basis pursuit problem can be derived by adapting the proof technique in [41, Appendix]) to the (bp*) formulation.

Lemma A.6. *Let $N \geq 2$ and consider (bp*) for*

$$A = \begin{pmatrix} a_1 & a_2 & \dots & a_N \end{pmatrix} \in \mathbb{C}^{1\times N}, \quad y = 1, \quad \varepsilon \in [0, 1),$$

where $a_j > 0$ for $j = 1, \dots, N$. Then the set of solutions of bp* is given by

$$\sum_{j=1}^{N} \left(t_j(1-\varepsilon)a_j^{-1}\right) e_j, \quad s.t. \quad t_j \in [0, 1], \quad \sum_{j=1}^{N} t_j = 1,$$
$$\text{and } t_j = 0 \text{ if } a_j < \max_k a_k,$$

where $\{e_j\}_{j=1}^N$ is the canonical basis of $\mathbb{C}^N$.

Finally, using Lemma A.6 the sequences establishing Banach-Mazur non-computability of any map $\Psi^s \in \Xi_{\mathrm{bp}^*,m,N,\varepsilon}$ for $\varepsilon \in (0, 1)$ can be constructed exactly as for (bp) in [19, Proof of main result] so that Theorem A.1 follows.

Appendix B. Proof of Theorem 4.11

In this section, we prove Theorem 4.11. First, we introduce some preliminary results starting with the Weierstrass approximation theorem.

Theorem B.1. *Let $f : [a, b] \to \mathbb{R}$ be a continuous function and $[a, b] \subset \mathbb{R}$ an interval. For every $\gamma > 0$, there exists a polynomial p_γ such that*

$$\sup_{x\in[a,b]} \left|f(x) - p_\gamma(x)\right| < \gamma.$$

Remark B.2. A constructive proof, see e.g. [72], of the Weierstrass approximation theorem can be established via Bernstein polynomials

$$B_n(x, f) := \sum_{k=0}^{n} f\left(\frac{k}{n}\right)\binom{n}{k} x^k(1-x)^{n-k} \text{ of degree } n \in \mathbb{N}$$

associated to a real-valued function $f \in C([0, 1])$. In particular, for $\gamma > 0$ we have for any $x \in [0, 1]$ that

$$\left|B_n(x,f)-f(x)\right| \le \gamma \quad \text{if } n \ge \frac{\|f\|_\infty}{\delta^2\gamma}, \tag{B.1}$$

where $\delta > 0$ satisfies for all $x, y \in [0,1]$:

$$|x-y| \le \delta \quad \Longrightarrow \quad |f(x)-f(y)| \le \frac{\gamma}{2}.$$

Note that for any $\gamma > 0$, a corresponding $\delta > 0$ exists due to the uniform continuity of f on $[0,1]$. By considering the function $\Phi : [0,1] \to [a,b]$ given by

$$\phi(x) = (b-a)x + a,$$

we can extend (B.1) to real-valued functions $g \in C([a,b])$ on arbitrary intervals $[a,b]$, i.e., for $\gamma > 0$ we have for any $x \in [a,b]$ that

$$\left|B_n(\phi^{-1}(x), g\circ\phi) - g(x)\right| \le \gamma \quad \text{if } n \ge \frac{\|g\circ\phi\|_\infty}{\delta^2\gamma},$$

where $\delta > 0$ satisfies for all $x, y \in [a,b]$:

$$|x-y| \le \delta \quad \Longrightarrow \quad |(g\circ\phi)(x) - (g\circ\phi)(y)| \le \frac{\gamma}{2}.$$

Our goal is to approximate the objective of basis pursuit optimization (bp), i.e., the ℓ_1 norm, by a polynomial on a suitable domain $I \subset \mathbb{C}^N$. In particular, we want I to contain the solutions of (bp) so that the optimization on the approximate objective ideally leads to a solution of the original problem, albeit this cannot be guaranteed. After identifying I, we can invoke Theorem B.1 and Remark B.2 to construct the sought polynomial. Hence, the first step is to find a domain satisfying our conditions. Via the pseudoinverse, we can construct I explicitly. In particular, we will use the following fact [100]: For $A \in \mathbb{C}^{m\times N}$ and $y \in \mathbb{C}^m$ the minimal ℓ_2 norm solution of

$$\arg\min_{x\in\mathbb{C}^N} \|Ax-y\|_{\ell_2} \tag{B.2}$$

is given by $A^\dagger y$, where $A^\dagger \in \mathbb{C}^{N\times m}$ denotes the pseudoinverse of A.

Lemma B.3. *Let $A \in \mathbb{C}^{m\times N}$, $y \in \mathbb{C}^m$ and $\varepsilon > 0$ and consider*

$$\arg\min_{x\in\mathbb{C}^N} \|x\|_{\ell_1} \text{ such that } \|Ax-y\|_{\ell_2} \le \varepsilon. \tag{B.3}$$

If the solution set is not empty, then the solution set is contained in $I = \{x \in \mathbb{C}^N : \|x\|_{\ell_2} < \sqrt{N}\left\|A^\dagger y\right\|_{\ell_2}\}$.

Proof. Set $\hat{x} = A^\dagger y$. If $\|A\hat{x}-y\|_{\ell_2} > \varepsilon$, then (B.2) implies that there does not exist a minimizer of (B.3) for the given set of parameters. Hence, we assume $\|A\hat{x}-y\|_{\ell_2} \le \varepsilon$ so that $\hat{x}$ satisfies the constraint of (B.3). Note that $\|x\|_{\ell_1} \ge \|x\|_{\ell_2}$ and $\|x\|_{\ell_1} \le \sqrt{n}\,\|x\|_{\ell_2}$ for all $x \in \mathbb{C}^n$. Thus, $\|z\|_{\ell_2} > \sqrt{N}\,\|\hat{x}\|_{\ell_2}$ entails that

$$\|z\|_{\ell_1} \ge \|z\|_{\ell_2} > \sqrt{N}\,\|\hat{x}\|_{\ell_2} \ge \|\hat{x}\|_{\ell_1}$$

for $z \in \mathbb{C}^N$, i.e., z is not a minimizer of (B.3). Therefore, the optimization problem

$$\arg\min_{x\in\mathbb{C}^N} \|x\|_{\ell_1} \text{ such that } \|Ax-y\|_{\ell_2} \le \varepsilon \ \wedge\ \|x\|_{\ell_2} < \sqrt{N}\,\|\hat{x}\|_{\ell_2}$$

is equivalent to (B.3) and its minimizers are contained in I. □

Now, we turn to the approximation of the ℓ_1 norm via polynomials.

Lemma B.4. *Let $\beta > 0$. For every $\gamma > 0$, there exists a polynomial $p_{\beta,\gamma}$ such that*

$$\sup_{x\in\mathbb{C}^n : \|x\|_{\ell_2} < \sqrt{n}\beta} \left|\|x\|_{\ell_1} - p_{\beta,\gamma}(\Re(x), \Im(x))\right| < \gamma.$$

The polynomial $p_{\beta,\gamma}$ can be explicitly expressed as

$$p_{\beta,\gamma}(\Re(x),\Im(x)) = \frac{1}{(n\beta)^{2k-1}} \sum_{j=0}^{k} \sqrt{\frac{j}{k}}\binom{k}{j} \sum_{i=1}^{n} \left(\Re(x_i)^2 + \Im(x_i)^2\right)^j \left(n^2\beta^2 - \Re(x_i)^2 - \Im(x_i)^2\right)^{k-j}, \tag{B.4}$$

where $k \ge \frac{4n^8\beta^5}{\gamma^5}$ is required.

Proof. Let $x \in \mathbb{C}^n$ such that $\|x\|_{\ell_2} < \sqrt{n}\beta$ holds. Then,

$$\|x\|_{\ell_1} \leq \sqrt{n}\,\|x\|_{\ell_2} < n\beta$$

so that

$$n\beta > \|x\|_{\ell_1} = \sum_{i=1}^{n} |x_i| = \sum_{i=1}^{n} \sqrt{\Re(x_i)^2 + \Im(x_i)^2}$$

and in particular

$$n\beta > \sqrt{\Re(x_i)^2 + \Im(x_i)^2} \quad \text{for all } i = 1, \dots, n. \tag{B.5}$$

Next, we want to invoke Theorem B.1 and Remark B.2 to approximate $\sqrt{\cdot}$ on the interval $[0, n^2\beta^2]$. To that end, let $g(x) = \sqrt{x}$ on $[0, n^2\beta^2]$ and $\phi(x) = n^2\beta^2 x$ on $[0,1]$. Observe that $(g\circ\phi)(x) = n\beta\sqrt{x}$ so that $\|g\circ\phi\|_\infty = n\beta$ and

$$\left|n\beta\sqrt{x} - n\beta\sqrt{y}\right|^2 \leq n^2\beta^2 \left|\sqrt{x} - \sqrt{y}\right| \left|\sqrt{x} + \sqrt{y}\right| = n^2\beta^2\,|x - y| \quad \text{for all } x, y \in [0,1],$$

i.e., we obtain

$$|x - y| \leq \frac{1}{n^2\beta^2}\left(\frac{\gamma}{2n}\right)^2 =: \delta \implies |(g\circ\phi)(x) - (g\circ\phi)(y)| \leq \frac{\gamma}{2n}.$$

Hence, applying Remark B.2 yields that for all $x, y \in [0, n^2\beta^2]$

$$\left|B_k(\phi^{-1}(x), g\circ\phi) - g(x)\right| \leq \frac{\gamma}{n} \quad \text{if } k \geq \frac{n\beta}{\delta^2 \frac{\gamma}{n}} = \frac{4n^8\beta^5}{\gamma^5}.$$

Thus, due to (B.5)

$$\begin{aligned}
&\sup_{x\in\mathbb{C}^n : \|x\|_{\ell_2} < \sqrt{n}\beta} \left| \|x\|_{\ell_1} - \sum_{i=1}^{n} B_k(\phi^{-1}(\Re(x_i)^2 + \Im(x_i)^2), g\circ\phi) \right| \\
&\qquad = \left| \sum_{i=1}^{n} \sqrt{\Re(x_i)^2 + \Im(x_i)^2} - B_k(\phi^{-1}(\Re(x_i)^2 + \Im(x_i)^2), g\circ\phi) \right| \\
&\qquad \leq \sum_{i=1}^{n} \left| g(\Re(x_i)^2 + \Im(x_i)^2) - B_k(\phi^{-1}(\Re(x_i)^2 + \Im(x_i)^2), g\circ\phi) \right| \\
&\qquad \leq n \cdot \frac{\gamma}{n} = \gamma \quad \text{if } k \geq \frac{4n^8\beta^5}{\gamma^5}
\end{aligned}$$

and the sought polynomial $p_{\beta,\gamma}$ indeed exists via

$$\begin{aligned}
p_{\beta,\gamma}(\Re(x), \Im(x)) &= \sum_{i=1}^{n} B_k(\phi^{-1}(\Re(x_i)^2 + \Im(x_i)^2), g\circ\phi) \\
&= \sum_{i=1}^{n} \sum_{j=0}^{k} (g\circ\phi)\left(\frac{j}{k}\right) \binom{k}{j} \left(\phi^{-1}\left(\Re(x_i)^2 + \Im(x_i)^2\right)\right)^j \left(1 - \phi^{-1}\left(\Re(x_i)^2 + \Im(x_i)^2\right)\right)^{k-j} \\
&= \sum_{i=1}^{n} \sum_{j=0}^{k} n\beta\sqrt{\frac{j}{k}} \binom{k}{j} \left(\frac{1}{n^2\beta^2}(\Re(x_i)^2 + \Im(x_i)^2)\right)^j \left(1 - \frac{1}{n^2\beta^2}(\Re(x_i)^2 + \Im(x_i)^2)\right)^{k-j} \\
&= \frac{1}{(n\beta)^{2k-1}} \sum_{j=0}^{k} \sqrt{\frac{j}{k}} \binom{k}{j} \sum_{i=1}^{n} \left(\Re(x_i)^2 + \Im(x_i)^2\right)^j \left(n^2\beta^2 - \Re(x_i)^2 - \Im(x_i)^2\right)^{k-j}. \quad \square
\end{aligned}$$

So far, we have established the existence of an optimization problem approximating basis pursuit in the following sense: For given $A \in \mathbb{C}^N$, $y \in \mathbb{C}^m$ and $\varepsilon > 0$

- Lemma B.3 describes a domain

$$I_{A,y} := \{x \in \mathbb{C}^N : \|x\|_{\ell_2} < \sqrt{N}\left\|A^\dagger y\right\|_{\ell_2}\} \tag{B.6}$$

that contains the solutions of basis pursuit for (A, y, ε),

- and Lemma B.4 describes a polynomial $p_{\beta,\gamma}$ depending on parameters $\beta, \gamma > 0$ that approximates the ℓ_1 norm on a set

$$I_\beta := \{x \in \mathbb{C}^N : \|x\|_{\ell_2} < \sqrt{N}\beta\} \tag{B.7}$$

up to an error of γ.

Hence, an appropriate choice of $\beta = \left\|A^\dagger y\right\|_{\ell_2}$ enables to approximate the objective of basis pursuit up to an error of γ on $I_{A,y}$. Note that the convergence of $p_{\beta,\gamma}$ to the ℓ_1 norm for $\gamma \to 0$ does not imply that the minimizer(s) of the respective optimization problems also converge towards the original minimizer(s) of basis pursuit. Unfortunately, obtaining a guarantee of this form does not appear feasible in the general case. Thus, we have to be content with the convergence of the objectives.

It remains to analyze whether the adjusted optimization problem with additional parameters β and γ can be tackled via BSS machines. We immediately observe that the function $p_{\beta,\gamma}$ given in (B.4) cannot be computed on a BSS machine. Indeed, $p_{\beta,\gamma}$ depends on coefficients encompassing values $(j/k)^{1/2}$, where $j = 0, \dots, k$ and $k \in \mathbb{N}$ needs to be chosen based on β and γ. Thus, the non-computability of the square-root function prohibits the computation of the coefficients on a BSS machine. Therefore, the corresponding optimization problem cannot be implemented on a BSS machine for adjustable parameters β and γ. On the other hand, we will show next that in the restricted case with fixed β and γ the polynomial $p_{\beta,\gamma}$ can be implemented on a BSS machine.

Lemma B.5. *Fix $\beta, \gamma > 0$ and $k \geq \frac{4n^8\beta^5}{\gamma^5}$. Then $p_{\beta,\gamma}$ as defined in (B.4) is BSS-computable.*

Proof. Based on k the finite set $\{\sqrt{1}, \dots, \sqrt{k}\} \cup \{\beta, \gamma\}$ can be encoded as constants in the memory of a BSS machine. Therefore, the coefficients of the polynomial $p_{\beta,\gamma}$ can be computed on a BSS machine so that $p_{\beta,\gamma}$ is a BSS-computable function as polynomials can be expressed through the concatenation of arithmetic operations. □

Finally, we prove that the optimization problem with objective $p_{\beta,\gamma}$ on I_β is solvable on a BSS machine. This concludes the proof of Theorem 4.11.

Proof of Theorem 4.11. We first show that there exists a BSS-computable function $\Xi^s \in \mathcal{M}_{\Xi_{\mathrm{p}(\beta,\gamma),m,N}}$ for fixed $\beta, \gamma > 0$. We can manually compute $k \in \mathbb{N}$ such that $k \geq \frac{4N^8\beta^5}{\gamma^5}$. Then, Lemma B.5 implies that the polynomial $p_{\beta,\gamma}$ given in (B.4) can be implemented on a BSS machine and Lemma B.4 entails that (4.3) is satisfied. Additionally, observe that

$$\|x\|_{\ell_2} < \sqrt{N}\beta \Leftrightarrow \|x\|_{\ell_2}^2 < N\beta^2 \Leftrightarrow \sum_{i=1}^n |x_i|^2 - N\beta^2 < 0 \Leftrightarrow \sum_{i=1}^n \mathfrak{R}(x_i)^2 + \mathfrak{F}(x_i)^2 - N\beta^2 < 0$$

is a semialgebraic set defined by a polynomial in $(\mathfrak{R}(x), \mathfrak{F}(x))$. Therefore $(\mathrm{p}(\beta,\gamma))$ describes the optimization of the polynomial $p_{\beta,\gamma}$ on a semialgebraic set. Thus, the existence of Ξ^s follows analogously to the proofs of Theorem 4.3 and Theorem 4.7 via the optimization Algorithm 1.

It is left to show that there exists a BSS-computable function $g : \mathbb{C}^{m\times N} \times \mathbb{C}^m \times \mathbb{R}_{>0} \times \mathbb{R}_{>0} \to \{0, 1\}$ so that $g(A, y, \varepsilon, \beta) = 1$ implies that the solutions of basis pursuit (bp) are contained in $I_\beta = \{x \in \mathbb{C}^N : \|x\|_{\ell_2} < \sqrt{N}\beta\}$. We will explicitly construct g. First, note that the function $g_{\mathrm{pi}} : \mathbb{C}^{m\times N} \to \mathbb{C}^{N\times m}$ mapping A on its pseudoinverse $g_{\mathrm{pi}}(A) = A^\dagger$ is a BSS-computable function. This follows immediately from the fact that the pseudoinverse can be computed via multiple applications of the Gaussian elimination algorithm and exchange of rows of matrices based on identifying zero-rows [96]. These operations can all be implemented on BSS machines since they depend only on basic arithmetic operations and comparisons to zero. Hence, the computability of g_{pi} follows.

Next, define the set $S := \{(A, y, \varepsilon) \in \mathbb{C}^{m\times N} \times \mathbb{C}^m \times \mathbb{R}_{>0} : g_{\mathrm{sol}}(A, y, \varepsilon) < 0\}$, where $g_{\mathrm{sol}} : \mathbb{C}^{m\times N} \times \mathbb{C}^m \times \mathbb{R}_{>0}$ is given by

$$g_{\mathrm{sol}}(A, y, \varepsilon) = \left\|Ag_{\mathrm{pi}}(A)y - y\right\|_{\ell_2}^2 - \varepsilon^2.$$

We can immediately conclude that g_{sol} is BSS-computable since g_{pi} is BSS-computable and g_{sol} is a polynomial in (A, y, ε). Hence, S is decidable on a BSS machine. Recall that $\left\|Ag_{\mathrm{pi}}(A)y - y\right\|_{\ell_2}^2 - \varepsilon^2 > 0$ implies that the associated basis pursuit optimization does not have any solution; compare proof of Lemma B.3. Consequently, the existence of minimizers of basis pursuit can be decided on BSS machines.

Finally, given that minimizer(s) for basis pursuit exist, we need to check whether they are contained in I_β. Due to (B.6) and (B.7) it suffices to decide the set $V := \{(A, y, \beta) \in \mathbb{C}^{m\times N} \times \mathbb{C}^m \times \mathbb{R}_{>0} : g_{\mathrm{dec}}(A, y, \beta) \leq 0\}$, where $g_{\mathrm{dec}} : \mathbb{C}^{m\times N} \times \mathbb{C}^m \times \mathbb{R}_{>0}$ is given by

$$g_{\mathrm{dec}}(A, y, \beta) = \left\|g_{\mathrm{pi}}(A)y\right\|_{\ell_2}^2 - \beta^2.$$

Again the BSS-computability of g_{dec} follows directly from the BSS-computability of g_{pi} so that V is decidable on a BSS machine. Therefore, setting

$$g(A, y, \varepsilon, \beta) = \chi_S(A, y, \varepsilon)\chi_V(A, y, \beta)$$

gives that g is BSS-computable as the product of two BSS-computable functions. Moreover, (B.6) shows that $g(A, y, \varepsilon, \beta) = 1$ if and only if the solutions of basis pursuit for (A, y, ε) are contained in I_β. Finally, the BSS-computability of g remains valid when considering real BSS machines representing $\mathbb{C}$ via the real and imaginary parts. □

Remark B.6. In the Turing model, the outlined approach to approximate basis pursuit is not feasible. In particular, the step involving the pseudoinverse of the input matrix is not transferable since the mapping of matrices on their pseudoinverses is not computable on a Turing machine [18].

Data availability

No data was used for the research described in the article.

References

[1] J. Adler, O. Öktem, Solving ill-posed inverse problems using iterative deep neural networks, Inverse Probl. 33 (12) (2017), p. 124 007.
[2] V. Antun, F. Renna, C. Poon, B. Adcock, A.C. Hansen, On instabilities of deep learning in image reconstruction and the potential costs of AI, Proc. Natl. Acad. Sci. 117 (48) (2020) 088.
[3] M. Araya-Polo, J. Jennings, A. Adler, T. Dahlke, Deep-learning tomography, Lead. Edge 37 (1) (2018) 58–66.
[4] S.R. Arridge, P. Maass, O. Öktem, C.-B. Schönlieb, Solving inverse problems using data-driven models, Acta Numer. 28 (2019) 1–174.
[5] J. Avigad, V. Brattka, Computability and analysis: the legacy of Alan Turing, in: R. Downey (Ed.), Turing's Legacy: Developments from Turing's Ideas in Logic, in: Lecture Notes in Logic., Cambridge University Press, 2014, pp. 1–47.
[6] P. Baillot, M. Pedicini, An embedding of the bss model of computation in light affine lambda-calculus, arXiv:cs/0608040v1, 2006.
[7] A. Bastounis, A.C. Hansen, V. Vlačić, The extended Smale's 9th problem – on computational barriers and paradoxes in estimation, regularisation, computer-assisted proofs and learning, arXiv:2110.15734, 2021.
[8] S. Basu, R. Pollack, M.-F. Roy, Algorithms in Real Algebraic Geometry, 2nd ed., Springer Verlag, Berlin, Heidelberg, 2006.
[9] A. Belloni, V. Chernozhukov, L. Wang, Square-root lasso: pivotal recovery of sparse signals via conic programming, Biometrika 98 (4) (2011) 791–806.
[10] J. Berner, P. Grohs, G. Kutyniok, P. Petersen, The modern mathematics of deep learning, in: Mathematical Aspects of Deep Learning, Cambridge University Press, 2022.
[11] A. Biondi, F. Nesti, G. Cicero, D. Casini, G. Buttazzo, A safe, secure, and predictable software architecture for deep learning in safety-critical systems, IEEE Embed. Syst. Lett. 12 (3) (2020) 78–82.
[12] P. Blouw, C. Eliasmith, Event-driven signal processing with neuromorphic computing systems, in: ICASSP, IEEE, 2020, pp. 8534–8538.
[13] L. Blum, Computing over the reals: where Turing meets Newton, Not. Am. Math. Soc. 51 (9) (2004) 1024–1034.
[14] L. Blum, F. Cucker, M. Shub, S. Smale, Complexity and Real Computation, Springer Verlag, New York, 1998.
[15] L. Blum, M. Shub, S. Smale, On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, Bull., New Ser., Am. Math. Soc. 21 (1) (1989) 1–46.
[16] H. Boche, Y. Böck, C. Deppe, Deciding the problem of remote state estimation via noisy communication channels on real number signal processing hardware, in: ICC 2022, IEEE, 2022, pp. 4510–4515.
[17] H. Boche, M. Cai, H.V. Poor, R.F. Schaefer, Detectability of denial-of-service attacks on arbitrarily varying classical-quantum channels, in: ISIT 2021, IEEE, 2021, pp. 912–917.
[18] H. Boche, A. Fono, G. Kutyniok, Non-computability of the pseudoinverse on digital computers, arXiv:2212.02940, 2022.
[19] H. Boche, A. Fono, G. Kutyniok, Limitations of deep learning for inverse problems on digital hardware, IEEE Trans. Inf. Theory 69 (12) (2023) 7887–7908.
[20] H. Boche, A. Fono, G. Kutyniok, Mathematical algorithm design for deep learning under societal and judicial constraints: the algorithmic transparency requirement, arXiv:2401.10310, 2024.
[21] H. Boche, R.F. Schaefer, H.V. Poor, Denial-of-service attacks on communication systems: detectability and jammer knowledge, IEEE Trans. Signal Process. 68 (2020) 3754–3768.
[22] H. Boche, R.F. Schaefer, H.V. Poor, F.H.P. Fitzek, On the need of neuromorphic twins to detect denial-of-service attacks on communication networks, IEEE/ACM Trans. Netw. (ISSN 1063-6692) 32 (4) (2024) 2875–2887, https://doi.org/10.1109/TNET.2024.3369018 [Online]. Available.
[23] H. Boche, R.F. Schaefer, H. Vincent Poor, Real number signal processing can detect denial-of-service attacks, in: ICASSP 2021, IEEE, 2021, pp. 4765–4769.
[24] J. Bochnak, M. Coste, M.-F. Roy, Real Algebraic Geometry, Springer Verlag, Berlin, Heidelberg, 1998.
[25] Y.N. Böck, H. Boche, R.F. Schaefer, F.H. Fitzek, H.V. Poor, Virtual-twin technologies in networking, IEEE Commun. Mag. 61 (11) (2023) 136–141, https://doi.org/10.1109/MCOM.001.2200861.
[26] E. Borel, Le calcul des intégrales définies, J. Math. Pures Appl. 8 (1912) 159–210.
[27] M. Borgerding, P. Schniter, S. Rangan, AMP-inspired deep networks for sparse linear inverse problems, IEEE Trans. Signal Process. 65 (16) (2017) 4293–4308.
[28] A. Boulemtafes, A. Derhab, Y. Challal, A review of privacy-preserving techniques for deep learning, Neurocomputing 384 (2020) 21–45.
[29] I. Boybat, B. Kersting, S.G. Sarwat, et al., Temperature sensitivity of analog in-memory computing using phase-change memory, in: IEDM 2021, IEEE, 2021.
[30] T. Brown, B. Mann, N. Ryder, et al., Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), NeurIPS 2020, vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.
[31] T.A. Bubba, G. Kutyniok, M. Lassas, et al., Learning the invisible: a hybrid deep learning-shearlet framework for limited angle computed tomography, Inverse Probl. 35 (6) (2019).
[32] P. Bürgisser, F. Cucker, Counting complexity classes over the reals I: The additive case, in: T. Ibaraki, N. Katoh, H. Ono (Eds.), Algorithms and Computation, Berlin, Heidelberg, Springer, Berlin, Heidelberg, 2003, pp. 625–634.
[33] E. Candes, J. Romberg, T. Tao, Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information, IEEE Trans. Inf. Theory 52 (2) (2006) 489–509.
[34] E. Candes, T. Tao, Decoding by linear programming, IEEE Trans. Inf. Theory 51 (12) (2005) 4203–4215.
[35] E.J. Candes, J.K. Romberg, T. Tao, Stable signal recovery from incomplete and inaccurate measurements, Commun. Pure Appl. Math. 59 (8) (2006) 1207–1223.
[36] E.J. Candes, T. Tao, Near-optimal signal recovery from random projections: universal encoding strategies?, IEEE Trans. Inf. Theory 52 (12) (2006) 5406–5425.
[37] N. Carlini, D. Wagner, Audio adversarial examples: Targeted attacks on speech-to-text, in: SPW 2018, IEEE, 2018, pp. 1–7.
[38] C. Chen, Q. Chen, J. Xu, V. Koltun, Learning to see in the dark, in: CVPR 2018, IEEE, 2018.
[39] S.S. Chen, D.L. Donoho, M.A. Saunders, Atomic decomposition by basis pursuit, SIAM J. Sci. Comput. 20 (1) (1998) 33–61.
[40] D.V. Christensen, R. Dittmann, B. Linares-Barranco, et al., 2022 Roadmap on neuromorphic computing and engineering, Neuromorph. Comput. Eng. 2 (2) (2022).

[41] M.J. Colbrook, V. Antun, A.C. Hansen, The difficulty of computing stable and accurate neural networks: on the barriers of deep learning and Smale's 18th problem, Proc. Natl. Acad. Sci. 119 (12) (2022).
[42] S. Cotter, B. Rao, K. Engan, K. Kreutz-Delgado, Sparse solutions to linear inverse problems with multiple measurement vectors, IEEE Trans. Signal Process. 53 (7) (2005) 2477–2488.
[43] F. Cucker, Recent advances in the computation of the homology of semialgebraic sets, in: F. Manea, B. Martin, D. Paulusma, G. Primiero (Eds.), Computing with Foresight and Industry, Springer-Verlag, Berlin, Heidelberg, 2019, pp. 1–12.
[44] A.J. Daley, I. Bloch, C. Kokail, et al., Practical quantum advantage in quantum simulation, Nature 607 (2022) 667–676.
[45] I. Daubechies, M. Defrise, C. De Mol, An iterative thresholding algorithm for linear inverse problems with a sparsity constraint, Commun. Pure Appl. Math. 57 (11) (2004) 1413–1457.
[46] D. Donoho, Compressed sensing, IEEE Trans. Inf. Theory 52 (4) (2006) 1289–1306.
[47] M.F. Duarte, Y.C. Eldar, Structured compressed sensing: from theory to applications, IEEE Trans. Signal Process. 59 (9) (2011) 4053–4085.
[48] M. Elad, Optimized projections for compressed sensing, IEEE Trans. Signal Process. 55 (12) (2007) 5695–5702.
[49] S.K. Esser, R. Appuswamy, P. Merolla, J.V. Arthur, D.S. Modha, Backpropagation for energy-efficient neuromorphic computing, in: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett (Eds.), NIPS 2015, vol. 28, Curran Associates, Inc., 2015.
[50] European Parliament, Artificial intelligence act, https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/698792/EPRS_BRI(2021)698792_EN.pdf, 2023.
[51] G. Fettweis, H. Boche, 6G: the personal tactile Internet—and open questions for information theory, IEEE BITS Inf. Theory Mag. 1 (1) (2021) 71–82.
[52] G. Fettweis, H. Boche, On 6G and trustworthiness, Commun. ACM 65 (4) (2022) 48–49.
[53] S. Flannigan, N. Pearson, G.H. Low, et al., Propagation of errors and quantitative quantum simulation with quantum advantage, Quantum Sci. Technol. 7 (4) (2022).
[54] G7 Hiroshima Summit 2023, G7 Hiroshima leaders' communiqué, https://www.g7hiroshima.go.jp/documents/pdf/Leaders_Communique_01_en.pdf, 2023.
[55] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
[56] L. Grozinger, M. Amos, T.E. Gorochowski, et al., Pathways to cellular supremacy in biocomputing, Nat. Commun. 10 (2019).
[57] D. Ham, H. Park, S. Hwang, K. Kim, Neuromorphic electronics based on copying and pasting the brain, Nat. Electron. 4 (2021) 635–644.
[58] K. Hammernik, T. Klatzer, E. Kobler, et al., Learning a variational network for reconstruction of accelerated MRI data, Magn. Reson. Med. 79 (6) (2018) 3055–3071.
[59] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, in: ICCV 2015, IEEE, 2015, pp. 1026–1034.
[60] Y. He, G. Meng, K. Chen, X. Hu, J. He, Towards security threats of deep learning systems: a survey, IEEE Trans. Softw. Eng. 48 (5) (2022) 1743–1770.
[61] D. Hilbert, Mathematical problems, Bull. Am. Math. Soc. 8 (10) (1902) 437–479.
[62] IBM Research Zurich, Neuromorphic Devices & Systems, https://www.zurich.ibm.com/st/neuromorphic/. (Accessed 5 July 2024).
[63] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, A. Madry, Adversarial examples are not bugs, they are features, in: NeurIPS 2019, Curran Associates Inc., Red Hook, NY, USA, 2019.
[64] Intel, Neuromorphic computing - next generation of AI, https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html. (Accessed 5 July 2024).
[65] S. Ji, Y. Xue, L. Carin, Bayesian compressive sensing, IEEE Trans. Signal Process. 56 (6) (2008) 2346–2356.
[66] K.H. Jin, M.T. McCann, E. Froustey, M. Unser, Deep convolutional neural network for inverse problems in imaging, IEEE Trans. Image Process. 26 (9) (2017) 4509–4522.
[67] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, A. Sebastian, In-memory hyperdimensional computing, Nat. Electron. 3 (6) (2020) 327–337.
[68] G. Katz, C. Barrett, D.L. Dill, K. Julian, M.J. Kochenderfer, Reluplex: an efficient smt solver for verifying deep neural networks, in: R. Majumdar, V. Kunčak (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2017, pp. 97–117.
[69] K.-I. Ko, Complexity Theory of Real Functions, Birkhauser Boston Inc., USA, 1991.
[70] L. Liu, S. Lu, R. Zhong, et al., Computing systems for autonomous driving: state of the art and challenges, IEEE Int. Things J. 8 (8) (2021) 6469–6486.
[71] X. Liu, L. Xie, Y. Wang, et al., Privacy and security issues in deep learning: a survey, IEEE Access 9 (2021) 4566–4593.
[72] G. Lorentz, Bernstein Polynomials, (AMS Chelsea Publishing). American Mathematical Society, 2013.
[73] X. Lv, G. Bi, C. Wan, The group lasso for stable recovery of block-sparse signal representations, IEEE Trans. Signal Process. 59 (4) (2011) 1371–1382.
[74] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, in: ICLR 2018, 2018.
[75] D. Marković, A. Mizrahi, D. Querlioz, J. Grollier, Physics for neuromorphic computing, Nat. Rev. Phys. 2 (9) (2020) 499–510.
[76] Y.V. Matiyasevich, Enumerable sets are Diophantine, Sov. Math. 11 (2) (1970) 354–357.
[77] F. Mireshghallah, M. Taram, P. Vepakomma, A. Singh, R. Raskar, H. Esmaeilzadeh, Privacy in deep learning: a survey, arXiv:2004.12254, 2020.
[78] M. Mirman, A. Hägele, P. Bielik, T. Gehr, M. Vechev, Robustness certification with generative models, in: SIGPLAN PLDI 2021, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1141–1154.
[79] A. Mousavi, A.B. Patel, R.G. Baraniuk, A deep learning approach to structured signal recovery, in: Allerton Conference 2015, 2015, pp. 1336–1343.
[80] K. Muhammad, A. Ullah, J. Lloret, J.D. Ser, V.H.C. de Albuquerque, Deep learning for safe autonomous driving: current challenges and future directions, IEEE Trans. Intell. Transp. Syst. 22 (7) (2021) 4316–4336.
[81] G. Ongie, A. Jalal, C.A. Metzler, R.G. Baraniuk, A.G. Dimakis, R. Willett, Deep learning techniques for inverse problems in imaging, IEEE J. Sel. Areas Inf. Theory 1 (1) (2020) 39–56.
[82] N. Papernot, P. McDaniel, X. Wu, S. Jha, A. Swami, Distillation as a defense to adversarial perturbations against deep neural networks, in: 2016 IEEE Symposium on Security and Privacy, IEEE Computer Society, 2016, pp. 582–597.
[83] Á. Papp, W. Porod, G. Csaba, Nanoscale neural network using non-linear spin-wave interference, Nat. Commun. 12 (2021).
[84] M. Payvand, M.V. Nair, L.K. Müller, G. Indiveri, A neuromorphic systems approach to in-memory computing with non-ideal memristive devices: from mitigation to exploitation, Faraday Discuss. 213 (2019) 487–510.
[85] P. Poirazi, A. Papoutsi, Illuminating dendritic function with computational models, Nat. Rev. Neurosci. 21 (2020) 303–321.
[86] M.B. Pour-El, J.I. Richards, Computability in Analysis and Physics (Perspectives in Logic), Cambridge University Press, 2017.
[87] A. Rao, P. Plank, A. Wild, W. Maass, A long short-term memory for AI applications in spike-based neuromorphic hardware, Nat. Mach. Intell. 4 (5) (2022) 467–479.
[88] Y. Rivenson, Z. Göröcs, H. Günaydin, Y. Zhang, H. Wang, A. Ozcan, Deep learning microscopy, Optica 4 (11) (2017) 1437–1443.
[89] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[90] H. Salman, J. Li, I. Razenshteyn, et al., Provably robust deep learning via adversarially trained smoothed classifiers, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), NeurIPS 2019, vol. 32, Curran Associates, Inc., 2019.
[91] J. Schlemper, J. Caballero, J.V. Hajnal, A.N. Price, D. Rueckert, A deep cascade of convolutional neural networks for dynamic MR image reconstruction, IEEE Trans. Med. Imaging 37 (2) (2018) 491–503.
[92] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, E. Eleftheriou, Memory devices and applications for in-memory computing, Nat. Nanotechnol. 15 (7) (2020) 529–544.
[93] A. Seidenberg, A new decision method for elementary algebra, Ann. Math. 60 (2) (1954) 365–374.

[94] I. Selesnick, Sparse regularization via convex analysis, IEEE Trans. Signal Process. 65 (17) (2017) 4481–4494.
[95] A.W. Senior, R. Evans, J. Jumper, et al., Improved protein structure prediction using potentials from deep learning, Nature 577 (2020) 706–710.
[96] X. Sheng, G. Chen, A note of computation for M-P inverse $A^{\dagger}$, Int. J. Comput. Math. 87 (10) (2010) 2235–2241.
[97] D. Silver, A. Huang, C.J. Maddison, et al., Mastering the game of Go with deep neural networks and tree search, Nature 529 (2016) 484–503.
[98] J.D. Smith, A.J. Hill, L.E. Reeder, et al., Neuromorphic scaling advantages for energy-efficient random walk computations, Nat. Electron. 5 (2) (2022) 102–112.
[99] R.I. Soare, Recursively enumerable sets and degrees, Bull. Am. Math. Soc. 84 (1987) 1149–1181.
[100] G.W. Stewart, On the perturbation of pseudo-inverses, projections and linear least squares problems, SIAM Rev. 19 (4) (1977) 634–662.
[101] C. Szegedy, W. Zaremba, I. Sutskever, et al., Intriguing properties of neural networks, in: Y. Bengio, Y. LeCun (Eds.), ICLR 2014, 2014.
[102] A. Tarski, A Decision Method for Elementary Algebra and Geometry, RAND Corporation, 1951.
[103] J. Tropp, Just relax: convex programming methods for identifying sparse signals in noise, IEEE Trans. Inf. Theory 52 (3) (2006) 1030–1051.
[104] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, A. Madry, Robustness may be at odds with accuracy, in: ICLR 2019, 2019.
[105] A.M. Turing, On computable numbers, with an application to the Entscheidungs-problem, Proc. Lond. Math. Soc. s2–42 (1) (1936) 230–265.
[106] K.F. Wagenbauer, C. Sigl, H. Dietz, Gigadalton-scale shape-programmable DNA assemblies, Nature 552 (2017) 78–83.
[107] K. Weihrauch, Computable Analysis: An Introduction, Springer-Verlag, Berlin, Heidelberg, 2000.
[108] O. Willers, S. Sudholt, S. Raafatnia, S. Abrecht, Safety concerns and mitigation approaches regarding the use of deep learning in safety-critical perception tasks, in: A. Casimiro, F. Ortmeier, E. Schoitsch, F. Bitsch, P. Ferreira (Eds.), SAFECOMP 2020 Workshops, Springer International Publishing, Cham, 2020, pp. 336–350.
[109] L.G. Wright, T. Onodera, M.M. Stein, et al., Deep physical neural networks trained with backpropagation, Nature 601 (2022) 549–555.
[110] S.J. Wright, R.D. Nowak, M.A.T. Figueiredo, Sparse reconstruction by separable approximation, IEEE Trans. Signal Process. 57 (7) (2009) 2479–2493.
[111] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, L. He, A survey of human-in-the-loop for machine learning, Future Gener. Comput. Syst. 135 (2022) 364–381.
[112] Y. Yang, J. Sun, H. Li, Z. Xu, Deep ADMM-net for compressive sensing MRI, in: D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, R. Garnett (Eds.), NIPS 2016, vol. 29, Curran Associates, Inc., 2016.
[113] H. Zhang, H. Chen, C. Xiao, et al., Towards stable and efficient training of verifiably robust neural networks, in: ICLR 2020, 2020.
[114] B. Zhu, J.Z. Liu, S.F. Cauley, B.R. Rosen, M.S. Rosen, Image reconstruction by domain-transform manifold learning, Nature 555 (2018) 487–492.