

Construction of the Kolmogorov-Arnold representation using the Newton-Kaczmarz method

Michael Poluektov^{1,3} and Andrew Polar²

¹Department of Mathematical Sciences and Computational Physics, School of Science and Engineering,
University of Dundee, Dundee DD1 4HN, UK

²Independent software consultant, Duluth, GA, USA

³Corresponding author, email: mpoluektov001@dundee.ac.uk

DRAFT: June 18, 2024

Abstract

The Kolmogorov-Arnold representation of a continuous multivariate function is a decomposition of the function into a structure of inner and outer functions of a single variable. It can be a convenient tool for tasks where it is required to obtain a predictive model that maps some vector input of a black box system into a scalar output. However, the construction of such representation based on the recorded input-output data is a challenging task. In the present paper, it is suggested to decompose the underlying functions of the representation into continuous basis functions and parameters. It is then proposed to find the parameters using the Newton-Kaczmarz method for solving systems of non-linear equations. The paper demonstrates that such approach is also an excellent tool for data-driven solution of partial differential equations. Numerical examples show that the proposed approach is efficient and more robust with respect to the selection of the initial guess for the parameters than the straightforward application of the Gauss-Newton method for the task of data modelling.

Keywords: discrete Urysohn operator, generalised additive model, Kolmogorov-Arnold representation, ridge function, model identification, Kaczmarz method, Newton-Kaczmarz method.

1 Introduction

One of the typical data science tasks is constructing a predictive model that maps some vector input into a scalar output. For example, this can be a regression model that establishes a relationship between the atomic-scale structure of matter and its properties [1]. Such model can be viewed as a multivariate function. In 1950s, Andrei Kolmogorov and Vladimir Arnold showed that any continuous multivariate function can be represented by a composition of functions of a single variable [2,3]. Thus, the Kolmogorov-Arnold representation is convenient tool for such tasks, replacing the problem of construction of a multivariate function in its entirety by the task of building of a set of functions of a single variable. The representation has its use in practical applications, for example, in the field of image processing [4,5].

The original Kolmogorov's proof was not constructive and did not provide insights into choosing the underlying functions of the representation. There is a series of works that gives constructive proofs of the Kolmogorov's theorem and proposes convergent algorithms for determining the functions [6–11]. There are also papers, where various approximations of the underlying functions and algorithms for their determination have been suggested [12–14]. Recently, a lightweight iterative algorithm for construction of the underlying functions has been proposed by the authors of this paper [15]; however, the functions were

limited to the class of piecewise-linear functions and the algorithm was given without the convergence proof.

The aim of the present paper is three-fold: (a) to propose a more general decomposition of the underlying functions of the Kolmogorov-Arnold representation via continuous basis functions and parameters; (b) to show that having the input-output data, the parameters can be efficiently found using the Newton-Kaczmarz method for solving systems of non-linear equations; and (c) to show that the Kolmogorov-Arnold representation is an efficient tool for data-driven solution of non-linear partial differential equations (PDEs).

When an approximate form of the Kolmogorov-Arnold representation is considered, e.g. when functions are restricted to a space spanned by some basis functions, it is more correct to refer to it as the Kolmogorov-Arnold model. It is related to some simpler models and their training¹ techniques can have common elements. These connections, as well as some historical perspective, are summarised in sections 2.1 and 3.1. Readers who are only interested in the identification the Kolmogorov-Arnold model can skip these sections.

The approach is tested in section 5. It has already been shown in [15] that the Kolmogorov-Arnold model in application to data modelling tasks performs equivalently to neural networks in terms of accuracy, but can require less computational time and resources for model identification. Therefore, the numerical examples do not aim to compare it to other models, such as neural-network-based ones, but rather aim to demonstrate the advantages of the proposed identification technique.

2 Models

2.1 The Urysohn model

The Urysohn integral operator has originally been introduced within the theory of non-linear integral equations [16]. It remained largely a theoretical concept until 1970s, when it was picked up by researchers from the theory of automatic control and was reintroduced as a continuous-time integral operator that transforms function $x(t)$ to function $z(t)$ in the following way [17]:

$$z(t) = \int_0^T V(s, x(t-s)) ds, \quad (1)$$

where $x : [-T, +\infty) \rightarrow [x_{\min}, x_{\max}]$ and $z : [0, +\infty) \rightarrow \mathbb{R}$ are continuous almost everywhere functions, function $V : [0, T] \times [x_{\min}, x_{\max}] \rightarrow \mathbb{R}$ is integrable, $t \geq 0$, $T \geq 0$, and $x_{\min}, x_{\max} \in \mathbb{R}$.

Since digital equipment collects continuous signals in a form of numeric series, the discrete-time Urysohn operator has also been introduced [17]:

$$z_i = \sum_{j=1}^m g^j(x_{i-j+1}), \quad i \in \mathbb{N}, \quad (2)$$

where $x_i \in [x_{\min}, x_{\max}]$ is an element of the series of the input values, $z_i \in \mathbb{R}$ is an element of the series of the output values, m is the memory depth of the operator² and $g^j : [x_{\min}, x_{\max}] \rightarrow \mathbb{R}$ are continuous almost everywhere functions. It can be seen that equation (2) results from a numerical quadrature of equation (1), as outlined in [18]. In the present paper, superscripts are attached to functions and vectors, while quantities with subscripts are single scalars (e.g. elements of a vector).

An interesting property of the discrete-time form of the Urysohn operator is that it can be written for a general task of data modelling. Given set of data records (X^i, y_i) , $i \in \{1, \dots, N\}$, where $X^i \in \mathbb{R}^m$ is the input of the i -th record, $y_i \in \mathbb{R}$ is the output of the i -th record and N is the number of records, following [15], the Urysohn model³ can be introduced as

$$\hat{y}_i = \sum_{j=1}^m g^j(X_j^i), \quad (3)$$

¹In the present paper, terms “model training” and “model identification”, implying determination of the model’s parameters, are used interchangeably.

²In principle, m can be arbitrary; however, when such model is to be identified (constructed), sufficient data to determine all functions g^j is needed.

³Term ‘model’ rather than ‘operator’ will be used for the data modelling case to distinguish it from the case of mapping one series into another.

where X_j^i denote the j -th component of vector X^i , and \hat{y}_i is the calculated model output of the i -th record. It is important to emphasise that the distinction is made between actual (recorded) output y_i and model output \hat{y}_i . For consistency of the notation, it is assumed that $X_j^i \in [x_{\min}, x_{\max}]$.

It is useful to highlight that the Urysohn model given by equation (3) is also referred to as the generalised additive model (GAM) in literature [19]. The latter has been introduced and researched independently. One possible reason for this is that the discrete Urysohn operator has been introduced originally for dynamical systems, and most research has been done within the fields of automatic control and signal processing. In the original definition of the discrete Urysohn operator, the inputs are the discrete values of the same signal; therefore, all functions are defined on the same domain and the adjacent functions are expected to be similar. In the data modelling case, equation (3), the functions can be defined on different domains and each argument can be continuous or quantised.

2.2 The Kolmogorov-Arnold model

In [2,3], it has been shown that any continuous multivariate function can be represented by a composition of functions of a single variable. More precisely, continuous function $F : [0, 1]^m \rightarrow \mathbb{R}$ can be exactly represented as

$$F(X_1, X_2, \dots, X_m) = \sum_{k=1}^{2m+1} \Phi^k \left(\sum_{j=1}^m f^{kj}(X_j) \right), \quad (4)$$

where $f^{kj} : [0, 1] \rightarrow \mathbb{R}$ and $\Phi^k : \mathbb{R} \rightarrow \mathbb{R}$ are continuous functions. This decomposition can be used for a general task of data modelling, where the output is a continuous function of the inputs. Given set of data records (X^i, y_i) , intermediate variable $\theta^i \in \mathbb{R}^{2m+1}$ is introduced for each data record and, following [15], the Kolmogorov-Arnold model for the input-output relationship is then written as

$$\hat{y}_i = \sum_{k=1}^{2m+1} \Phi^k(\theta_k^i), \quad \theta_k^i = \sum_{j=1}^m f^{kj}(X_j^i), \quad k \in \{1, \dots, 2m+1\}, \quad (5)$$

where θ_k^i denotes the k -th component of vector θ^i . For consistency of the notation, the domains of functions f^{kj} have been extended to $[x_{\min}, x_{\max}]$ in equation (5), which can always be done by rescaling of the inputs. It should be mentioned that the Kolmogorov-Arnold model can also be constructed of discontinuous functions [20]; however, no practical identification algorithms for a model with discontinuous or unbounded functions have been proposed.

This model can also be interpreted as a particular tree of the discrete Urysohn operators (written for the case of data modelling), containing 1 ‘root’ operator with $2m+1$ functions and $2m+1$ ‘branch’ operators with m functions each. Intermediate variable θ^i can then be interpreted as a hidden layer of the model.

It is useful to note that the Kolmogorov-Arnold representation can also be viewed as a generalisation of the so-called ridge function [21]:

$$F(X_1, X_2, \dots, X_m) = \Phi \left(\sum_{j=1}^m c_j X_j \right), \quad (6)$$

where $c_j \in \mathbb{R}$ are some constants and $\Phi : \mathbb{R} \rightarrow \mathbb{R}$ is some function. The ridge function models are used in some areas of data modelling and are even suggested as an approximate replacement of the Kolmogorov-Arnold model and neural networks [21].

3 Discretisation and identification

3.1 Building the Urysohn model

To handle the models computationally, the underlying functions must be represented in some discretised form. For the Urysohn model, one option is to use the following decomposition of functions g^j :

$$g^j(x) = \sum_{p=1}^n U_{jp} \phi^p(x), \quad (7)$$

where U_{jp} are the model parameters and $\phi^p : [x_{\min}, x_{\max}] \rightarrow \mathbb{R}$ are the basis functions with compact support. From equations (3) and (7), it can be seen that the discrete Urysohn operator is defined by its parameters U_{jp} , which can also be understood as elements of m -by- n matrix U .

The identification problem for the Urysohn model is finding parameters U_{jp} given set of records (X^i, y_i) , such that the discrepancy between \hat{y}_i and y_i is minimised. This problem and its properties (e.g. the non-uniqueness of the solution) have been studied in detail in [18] for piecewise-linear and piecewise-constant basis functions.

It can be seen that \hat{y}_i is a linear function of the parameters (but not a linear function of the inputs). Therefore, it is possible to assemble a system of linear equations with respect to U_{jp} and use it to reformulate the identification problem. Thus, the model parameters are rearranged into vector Z , the values of the basis functions are rearranged into matrix M , rows of which are denoted as M^i , and the outputs are rearranged into vector Y :

$$\begin{aligned} Z &= [U_{11} \ \dots \ U_{1n} \ U_{21} \ \dots \ U_{2n} \ \dots \ U_{m1} \ \dots \ U_{mn}]^T, \\ M^i &= [\phi^1(X_1^i) \ \dots \ \phi^n(X_1^i) \ \phi^1(X_2^i) \ \dots \ \phi^n(X_2^i) \ \dots \ \phi^1(X_m^i) \ \dots \ \phi^n(X_m^i)], \\ Y &= [y_1 \ \dots \ y_N]^T. \end{aligned}$$

This transforms the identification problem into the following minimisation problem:

$$Z^* = \underset{Z}{\operatorname{argmin}} \|Y - MZ\|^2, \quad (8)$$

where Z^* is the sought solution and $\|\cdot\|$ denotes the standard l^2 -norm of a vector. In [18], it has been suggested to find an approximate solution of this problem iteratively row-by-row (or record-by-record) using the projection descend method [22, 23], also known as the Kaczmarz method, which represents the following sequence:

$$Z^{q+1} = Z^q + \mu \frac{y_i - M^i Z^q}{\|M^i\|^2} M^{iT}, \quad (9)$$

where $\mu \in (0, 1]$ is the regularisation parameter and Z^q is the approximation of the solution at iteration q . Index i changes each iteration. The method requires initial guess Z^0 and continues until some convergence criteria are reached. It should be emphasised that in [18], the method is presented in less general terms — it is written specifically for piecewise-linear and piecewise-constant basis functions.

A number of important remarks should be made briefly regarding the method above (the details are given in [18] and references therein). First, the method considers system of linear equations $MZ = Y$. For an underdetermined system, the method converges to a solution that is closest to the initial guess in terms of the l^2 -norm. For an overdetermined system, it produces a solution that oscillates in a region around the true solution of minimisation problem (8). The size of this region decreases with decreasing μ . Second, although an approximate solution is obtained, there is a significant advantage of the method — assembling of matrix M and keeping it in the memory is not required, as the method iterates row-by-row. Furthermore, in the case of the Urysohn model, since functions ϕ^p have compact support, matrix M is sparse¹; therefore, only a small subset of elements of Z^q is updated in equation (9), which is computationally cheap. Third, a consequence of the sparsity of M is that the projection descend method converges relatively fast, as the rows of M are either orthogonal or close to orthogonal (the detailed analysis of the convergence of the projection descend method can be found in e.g. [24] and references therein).

3.2 Building the Kolmogorov-Arnold model

As in the previous subsection, the underlying functions of the model are decomposed as

$$f^{kj}(x) = \sum_{p=1}^n H_{kjp} \phi^p(x), \quad (10)$$

$$\Phi^k(t) = \sum_{l=1}^s G_{kl} \psi^l(t), \quad (11)$$

¹For example, for piecewise-linear basis functions given by equation (19), it is easy to see that for any X_j^i , there will be at most two non-zero $\phi^p(X_j^i)$ out of n . Hence, M^i will contain at most $2m$ non-zero elements, and the fraction of the non-zero elements of matrix M is at most $2/n$.

where H_{kjp} and G_{kl} are the model parameters, $\phi^p : [x_{\min}, x_{\max}] \rightarrow \mathbb{R}$ and $\psi^l : [t_{\min}, t_{\max}] \rightarrow \mathbb{R}$ are the basis functions, and $t_{\min}, t_{\max} \in \mathbb{R}$. Functions ϕ^p and ψ^l must be continuous, and functions ψ^l must be differentiable everywhere except at a finite number of points. Integers n and s are the numbers of the basis functions used for the inner and the outer functions of the model, respectively.

The identification problem for the Kolmogorov-Arnold model is finding parameters H_{kjp} and G_{kl} given set of records (X^i, y_i) , such that the discrepancy between \hat{y}_i and y_i is minimised. As above, this problem can be written as minimisation problem

$$\{H_{kjp}^*, G_{kl}^*\} = \operatorname{argmin}_{H, G} \sum_{i=1}^N (\hat{y}_i - y_i)^2, \quad (12)$$

where H_{kjp}^* and G_{kl}^* are the sought parameters. In contrast to the Urysohn model, output \hat{y}_i is now a non-linear function of the model parameters.

An approximate solution of problem (12) can be found using the Newton-Kaczmarz method [25–27] for solving systems of non-linear equations. As outlined in appendix A, this method can be understood as an iterative sequence of alternating between equations' linearisation and application of one step of the Kaczmarz method.

The system of non-linear equations with respect to the model parameters is written as

$$\hat{y}_i - y_i = 0, \quad i \in \{1, \dots, N\}. \quad (13)$$

The solution is found iteratively. The approximations of the model parameters at iteration q are denoted as H_{kjp}^q and G_{kl}^q . It is assumed that initial guesses H_{kjp}^0 and G_{kl}^0 for the parameters are made. Writing the Newton-Kaczmarz method (summarised in appendix A) for system (13) leads to

$$H_{kjp}^{q+1} = H_{kjp}^q - \mu (\hat{y}_i - y_i) \zeta^{-1} \frac{\partial \hat{y}}{\partial H_{kjp}}, \quad (14)$$

$$G_{kl}^{q+1} = G_{kl}^q - \mu (\hat{y}_i - y_i) \zeta^{-1} \frac{\partial \hat{y}}{\partial G_{kl}}, \quad (15)$$

where $\mu \in (0, 1]$ is the regularisation parameter. Variable ζ is the squared norm of the gradient of \hat{y} with respect to the parameters:

$$\zeta = \sum_{k=1}^{2m+1} \sum_{j=1}^m \sum_{p=1}^n \left(\frac{\partial \hat{y}}{\partial H_{kjp}} \right)^2 + \sum_{k=1}^{2m+1} \sum_{l=1}^s \left(\frac{\partial \hat{y}}{\partial G_{kl}} \right)^2. \quad (16)$$

The derivatives of \hat{y} are summarised in appendix B. It is implied that the values of the derivatives are calculated at X^i and using the parameters at iteration q . As previously, index i changes each iteration. The iterations are performed until some convergence criteria are reached, e.g. $|\hat{y}_i - y_i|$ is sufficiently small for sufficiently large number of iterations (the exact thresholds are decided by the user).

As for the case of the Urysohn model, system (13) can be underdetermined or overdetermined. An example of the latter is a system with noise-affected recorded output y_i . As in section 3.1, the method then gives an approximate solution, which oscillates in a region around the true solution of minimisation problem (12) with the size of this region depending on μ .

It is important to note that the original Newton-Kaczmarz method is formulated for systems of non-linear equations where functions are continuously differentiable, as outlined in appendix A. Thus, if functions ψ^l are continuously differentiable, then the algorithm is locally convergent, as it is the direct application of the Newton-Kaczmarz method to system (13), which makes it distinct from the algorithm proposed previously in [15] that was given without the convergence proof.

If the derivatives of functions ψ^l are discontinuous only at points belonging to a finite set, iterations at which θ_k^i falls exactly at a point of discontinuity can be simply skipped. In practice, this does not impact the convergence of the method. The relaxation of the continuity requirements allows using piecewise-linear basis functions, which is convenient from a practical point of view.

3.2.1 Piecewise-linear basis functions

In practice, it is convenient to use the piecewise-linear basis functions because of simplicity. To define the functions, equally-spaced nodes are introduced as

$$x_p = x_{\min} + (p-1) \Delta x, \quad \Delta x = \frac{x_{\max} - x_{\min}}{n-1}, \quad p \in \{1, \dots, n\}. \quad (17)$$

The same is done for the intermediate variable:

$$t_l = t_{\min} + (l - 1) \Delta t, \quad \Delta t = \frac{t_{\max} - t_{\min}}{s - 1}, \quad l \in \{1, \dots, s\}. \quad (18)$$

As introduced previously, n and s are the numbers of the inner and the outer basis functions, respectively. It should be noted that in many problems, the optimal choice of nodes is not equally-spaced, and redistributing the nodes adaptively can improve the accuracy of the model, however, leading to a more complex code/implementation. For convenience, the nodes are combined in sets $P = \{x_p\}$ and $Q = \{t_l\}$. The basis functions are continuous, are linear between the nodes, are exactly 1 at a given node and are exactly 0 at all other nodes:

$$\phi^p(x) = \begin{cases} 0, & \text{if } x \in P \text{ and } x \neq x_p, \\ 1, & \text{if } x = x_p, \\ \text{linear,} & \text{otherwise,} \end{cases} \quad \psi^l(t) = \begin{cases} 0, & \text{if } t \in Q \text{ and } t \neq t_l, \\ 1, & \text{if } t = t_l, \\ \text{linear,} & \text{otherwise.} \end{cases} \quad (19)$$

3.2.2 Gaussian basis functions

A simple choice of smooth basis functions is a set of Gaussian functions. Using equally-spaced nodes introduced above, these functions can be formally written as

$$\phi^p(x) = \exp\left(-\frac{\gamma}{\Delta x^2}(x - x_p)^2\right), \quad (20)$$

$$\psi^l(t) = \exp\left(-\frac{\gamma}{\Delta t^2}(t - t_l)^2\right), \quad (21)$$

where γ is the decay-rate parameter.

3.2.3 Spline basis functions

Another convenient choice of basis functions is a set of spline functions. This choice is common in numerical approximation and forms the foundation of techniques such as isogeometric analysis [28]. Splines have been previously used to approximate functions of the Kolmogorov-Arnold model [29] and of multi-layer perceptrons [30].

The classical choice for 1D interpolation problems is cubic splines. Thus, the basis functions can be formally written as twice continuously differentiable functions such that

$$\phi^p(x) = \begin{cases} 0, & \text{if } x \in P \text{ and } x \neq x_p, \\ 1, & \text{if } x = x_p, \\ \text{cubic,} & \text{otherwise,} \end{cases} \quad \psi^l(t) = \begin{cases} 0, & \text{if } t \in Q \text{ and } t \neq t_l, \\ 1, & \text{if } t = t_l, \\ \text{cubic,} & \text{otherwise.} \end{cases} \quad (22)$$

The latter must be supplemented with the end conditions, such as *not-a-knot* or *natural*.

3.2.4 Initial guess for the parameters

In general, initial approximations H_{kjp}^0 and G_{kl}^0 can be arbitrary. However, the outputs of the inner functions constitute the intermediate variable, the components of which are discretised according to equation (18). With ‘bad’ initial approximations of the parameters, the intermediate variable might significantly drift during the identification process. In the previous method [15], it was suggested to update limits t_{\min} and t_{\max} of the intermediate variable and its nodal positions t_l . A better approach is to initialise the parameters in such a way that this updating procedure is not necessary.

One such way is to take $t_{\min} = y_{\min}$ and $t_{\max} = y_{\max}$, where y_{\min} and y_{\max} are the minimum and the maximum values of output y_i , respectively. The model parameters are then initialised as uniformly-distributed random numbers:

$$H_{kjp}^0 \sim \text{unif}\left(\frac{y_{\min}}{m}, \frac{y_{\max}}{m}\right), \quad G_{kl}^0 \sim \text{unif}\left(\frac{y_{\min}}{2m+1}, \frac{y_{\max}}{2m+1}\right), \quad (23)$$

where $2m+1$ in the denominator corresponds to the number of outer functions Φ^k in the model¹.

¹If the number of addends is decreased, then the denominator in equation (23) should be modified accordingly.

3.2.5 Prevention of overfitting

The standard way of the overfitting detection is the usage of the training and the validation datasets, where the model is fit to the former and the model's performance is verified using the latter. Using this process, the complexity of the model is adjusted. The complexity of the Kolmogorov-Arnold model is determined by the numbers of the basis functions, n and s .

Furthermore, as shown in [15], sometimes, it is not necessary to use the 'full' Kolmogorov-Arnold model, which consists of $2m + 1$ addends (corresponding to functions Φ^k). Depending on the complexity of the modelled system and the desired accuracy, the number of addends can be decreased, which gives the third parameter responsible for the complexity of the model that should be fine-tuned using the training and the validation datasets.

4 Data-driven solution of non-linear PDEs

Most multi-physics problems are described by partial differential equations (PDEs). Solution of PDEs provides physical quantities within predefined spatial and temporal domains, but it requires exact boundary data, which is not often readily available due to experimental limitations. Noisy and partially-available data can be utilised in combination with traditional solution methods after post-processing, but such techniques typically have an *ad hoc* nature.

An alternative approach to predicting evolution of physical quantities under scarcity and/or uncertainty of boundary data is physics-informed machine learning [31]. It is an emerging field focusing on embedding physical constraints (e.g. in form of PDEs) into machine-learning models by training the latter not only on the experimentally-obtained data, but also on the additional data resulting from the enforcement of the constraints. Neural networks are predominantly utilised as the underlying model, and physics-informed neural networks (PINNs) have been successfully used to solve, for example, non-linear PDEs with shock waves [32] and the Navier-Stokes equations [33].

Conceptually, the approach consists in assuming that the unknown function of a PDE is the output of a model, the inputs of which are the spatial and the temporal coordinates. The model parameters are then identified using points internal to the computational domain, at which the residuals are calculated using the PDE, and points at the boundary of the computational domain, at which the residuals are obtained from the data.

4.1 Kolmogorov-Arnold model as a PDE solver

The Kolmogorov-Arnold model in combination with the Newton-Kaczmarz identification method can be used not only for data modelling, as described above, but also for data-driven solution of non-linear PDEs. Since the present paper focuses on a model with a single output, the present section is limited to PDEs with scalar unknown functions, but the generalisation does not seem to present a major technical challenge.

Spatio-temporal domain $\Omega \in \mathbb{R}^m$ is considered, and $u : \Omega \rightarrow \mathbb{R}$ is the unknown function of partial differential equation

$$L_I(X, u, \nabla u, \nabla \nabla u, \dots) = 0, \quad u = u(X), \quad X \in \Omega, \quad (24)$$

where L_I is some function, ∇u denotes the vector of partial derivatives $\partial u / \partial X_j$, each component X_j of vector X may stand for spatial or temporal coordinate, $\nabla \nabla u$ denotes the matrix of the second partial derivatives, etc. The boundary conditions are written in a similar form:

$$L_B(X, u, \nabla u, \nabla \nabla u, \dots) = 0, \quad X \in \Gamma, \quad \Gamma \in \partial \Omega. \quad (25)$$

It is assumed that functions L_I and L_B are such that the problem is well-posed.

A set of randomly-generated domain-internal points $X^i \in \Omega / \Gamma$ for $i \in \{1, \dots, N_I\}$ is created. In the case of experimentally-measured boundary data, the values of L_B are available only at a finite number of points, which can be written as $X^i \in \Gamma$ for $i \in \{N_I + 1, \dots, N_I + N_B\}$. The key idea of the approach is to assume that u can be adequately approximated by the Kolmogorov-Arnold model:

$$u(X^i) = \hat{y}_i, \quad (26)$$

where \hat{y}_i is given by equations (5), (10) and (11). It is implied that $N = N_I + N_B$.

The problem can then be formulated as finding parameters H_{kjp} and G_{kl} , such that the absolute values of functions L_I and L_B evaluated at points X^i are minimised. As above, this problem can be written as minimisation problem

$$\{H_{kjp}^*, G_{kl}^*\} = \operatorname{argmin}_{H, G} \left(\frac{\nu}{N_I} \sum_{i=1}^{N_I} L_I^2(X^i) + \frac{1-\nu}{N_B} \sum_{i=N_I+1}^{N_I+N_B} L_B^2(X^i) \right), \quad (27)$$

where H_{kjp}^* and G_{kl}^* are the sought parameters, ν is a weight factor associated with the internal data.

An approximate solution of problem (27) can be found using the Newton-Kaczmarz method. Equations (24) and (25) evaluated at points X^i are rewritten with respect to the model parameters:

$$L_I(X^i) = L_I(X^i, H_{kjp}, G_{kl}) = 0, \quad i \in \{1, \dots, N_I\}, \quad (28)$$

$$L_B(X^i) = L_B(X^i, H_{kjp}, G_{kl}) = 0, \quad i \in \{N_I+1, \dots, N_I+N_B\}. \quad (29)$$

This is a system of non-linear equations. As above, the solution process starts with random initial values H_{kjp}^0 and G_{kl}^0 and updates the parameters iteratively point-by-point in the opposite direction to the gradient of L with respect to the parameters. The subscripts of L are omitted for brevity — it is implied that $L = L_I(X^i)$ when $X^i \in \Omega/\Gamma$ and $L = L_B(X^i)$ when $X^i \in \Gamma$. This gives the following iterative procedure:

$$H_{kjp}^{q+1} = H_{kjp}^q - \mu L \zeta^{-1} \frac{\partial L}{\partial H_{kjp}}, \quad G_{kl}^{q+1} = G_{kl}^q - \mu L \zeta^{-1} \frac{\partial L}{\partial G_{kl}}, \quad (30)$$

where $\mu \in (0, 1]$ is the regularisation parameter. Superscript q denotes the approximation of the parameters at iteration q . The values of L and its derivatives are calculated using the parameters at iteration q . Variable ζ is the squared norm of the gradient of L with respect to the parameters:

$$\zeta = \sum_{k=1}^{2m+1} \sum_{j=1}^m \sum_{p=1}^n \left(\frac{\partial L}{\partial H_{kjp}} \right)^2 + \sum_{k=1}^{2m+1} \sum_{l=1}^s \left(\frac{\partial L}{\partial G_{kl}} \right)^2. \quad (31)$$

As previously, index i changes each iteration.

In the case of data-driven solution of PDEs, the set of equations for parameter identification consists of two distinct groups, corresponding to the internal and to the boundary points; therefore, the choice of how index i changes will influence the result, i.e. the user might decide to pick the internal or the boundary points more often. Weight factor ν from problem (27) corresponds to the frequency of selection of a record with domain-internal input X^i . To perform the model training systematically, consecutive iterations of the model update can be combined into batches. One such batch update consists of using N_b boundary points and N_i internal points. In this case, $\nu = N_i / (N_i + N_b)$.

5 Numerical examples

In this section, three numerical examples are presented to show the efficiency of the proposed approach. After having represented unknown functions f^{kj} and Φ^k via basis functions ϕ^p and ψ^l and parameters H_{kjp} and G_{kl} , various methods can be used to determine the parameters. The straightforward choice is the Gauss-Newton (GN) method. Therefore, it is necessary to investigate how it performs compared to the Newton-Kaczmarz (NK) method for the considered identification problem, which is done in the first example. Furthermore, as the NK method is related to the stochastic gradient descent (SGD) method (as summarised in appendix A.2), it is useful to compare the NK method to other variants of the SGD method. For such comparison, the relatively-popular ‘Adam’ SGD method is chosen [34].

5.1 Example 1: Identification of a ridge function

To compare the identification methods, the ridge function model, equation (6), is taken. It has similar structure to the Kolmogorov-Arnold model, but with one addend and with the inner functions being linear. Reasons for this choice will be clarified later.

Since Newton-type methods require an initial guess, it is important that the identification technique is robust with respect to the choice of the guess and does not get stuck at local minima. Therefore, this example aims to compare the dependence of the methods’ convergence on the initial guess.

The example consists in assuming a certain model, generating the data (i.e. generating the inputs and calculating the corresponding outputs using the assumed model), and identifying the model parameters using the data. The advantage of such approach is the availability of the exact solution of the identification problem. The code of this example is available at the GitHub repository¹.

As before, outer function Φ is represented via the basis functions and the parameters — equation (11) with omitted index k . The Gaussian basis functions, equation (21), with $\gamma = 2$, $t_{\min} = 0.5$, and $t_{\max} = 2.5$, are taken. To generate the data, a model with $m = 5$ inputs and $s = 3$ basis functions is taken. The following parameters are assumed:

$$c = [-0.7 \quad 2.5 \quad -1.2 \quad 0.8 \quad 1.6], \quad G = [2.1 \quad -0.9 \quad 0.7]. \quad (32)$$

To create the datasets of $N = 400$ records, uniformly-distributed random inputs from $x_{\min} = 0$ to $x_{\max} = 1$ are generated, $X_j^i \sim \text{unif}(0, 1)$, and outputs y_i are calculated.

Having the datasets, the identification problem is solved using the NK, the GN, and the ‘Adam’ SGD methods. The GN method is summarised in appendix C. The ‘Adam’ SGD method is summarised in appendix D. In the NK method, regularisation parameter μ is varied (the values are given in the corresponding tables) and 10^4 iterations are performed. The NK method to identify the ridge function model can be trivially obtained from the formulation for the Kolmogorov-Arnold model, equation (15), by setting functions ϕ^p to be linear, setting the number of such functions to 1, and setting the number of addends in the model to 1. In the GN method, tolerance $\delta = 10^{-12}$ is set and the maximum number of iterations is taken to be 100. In the ‘Adam’ SGD method, 10^4 iterations are performed (the same as in the NK method to establish the proper comparison), parameter $\varepsilon = 10^{-8}$ is set (the default value proposed in [34]), and parameters β_1, β_2, μ are varied (the values are given in the corresponding tables).

To solve the identification problem, the initial guess for the parameters is taken as the exact solution plus a random perturbation, i.e.

$$c_j^0 = c_j + \alpha e_j^1, \quad G_l^0 = G_l + \alpha e_l^2, \quad (33)$$

where $e_j^1, e_l^2 \sim \text{unif}(-0.5, 0.5)$ are uniformly-distributed random numbers. For small α , the initial guess is close to the true solution and the methods are expected to converge, while for large α , the methods might not converge.

The normalised root-mean-square error (RMSE),

$$E_{\text{RMSE}} = \frac{1}{y_{\max} - y_{\min}} \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}, \quad (34)$$

is used as an accuracy metric. After the methods solve the identification problem and produce the model parameters for given α , error E_{RMSE} is calculated. Since the generation of the datasets and the initial guess for the parameters involve randomisation, error E_{RMSE} can significantly differ from one run² to another; therefore, its average behaviour should be considered. In particular, an illustrative characteristic of the performance of the methods is the average proportion of runs, in which E_{RMSE} is below some threshold. Such characteristic is more useful than a direct comparison of average E_{RMSE} produced by the GN and the NK methods because it is not sensitive to the outliers, where term ‘outlier’ is used loosely to refer to the results way above the threshold.

The average percentages of runs, for which the RMSE is below 5%, 10%, and 20% for different amplitude of the perturbations of the initial guess, parameter α , are shown in table 1 for all three methods. As expected, as α increases, the percentage of the accurate runs drops. More importantly, it can be seen that the GN method consistently results in fewer runs with the error below a given threshold than the NK method. Furthermore, the accuracy of the GN method drops more rapidly than of the NK method. For example, for $\alpha = 1.2$, more than 90% of the runs of the NK method produce errors less than 20%, while only about 50% of the runs of the GN method have such accuracy; for $\alpha = 2.8$, which is close to a completely random initial guess, since such value of α is greater than any parameter value in the assumed model, about 20% of the runs of the NK method produce errors less than 10%, while only about 1% of the runs of the GN method have such accuracy.

The percentages of the accurate runs of the methods might seem to be relatively small, but this example is specifically chosen to be challenging for the methods. In particular, the exact values of G_l given

¹<https://github.com/andrewpolar/RidgeIdentM>

²One execution of the code is implied by the ‘run’.

α	0.4	0.8	1.2	1.6	2.0	2.4	2.8
Gauss-Newton (GN) method							
err. < 5%	93.6 \pm 1.3	59.0 \pm 5.2	31.4 \pm 1.7	12.4 \pm 2.1	5.0 \pm 2.1	2.0 \pm 1.6	0.4 \pm 0.5
err. < 10%	93.8 \pm 1.1	59.0 \pm 5.2	32.6 \pm 2.3	13.6 \pm 2.3	5.6 \pm 1.9	2.2 \pm 1.9	1.2 \pm 0.8
err. < 20%	99.2 \pm 0.4	80.0 \pm 4.8	55.0 \pm 6.4	33.8 \pm 2.9	21.8 \pm 4.8	11.0 \pm 3.1	7.4 \pm 2.9
% of conv.	99.4 \pm 0.5	83.2 \pm 5.5	62.4 \pm 7.2	40.8 \pm 3.6	29.8 \pm 7.2	22.2 \pm 2.9	19.6 \pm 5.6
RMSE in %	0.8 \pm 0.2	4.3 \pm 0.1	8.3 \pm 1.1	11.9 \pm 1.0	15.1 \pm 2.8	22.1 \pm 2.5	24.7 \pm 3.1
Newton-Kaczmarz (NK) method, $\mu = 0.1$							
err. < 5%	98.4 \pm 0.9	78.2 \pm 3.0	49.4 \pm 4.7	35.8 \pm 4.1	24.8 \pm 2.6	16.0 \pm 5.1	13.0 \pm 4.3
err. < 10%	99.8 \pm 0.4	95.0 \pm 2.0	78.0 \pm 7.2	56.2 \pm 3.6	42.2 \pm 4.3	30.2 \pm 5.8	20.4 \pm 4.5
err. < 20%	100 \pm 0.0	98.8 \pm 1.1	90.8 \pm 3.8	75.4 \pm 1.7	64.4 \pm 3.2	48.0 \pm 5.3	34.8 \pm 7.2
Newton-Kaczmarz (NK) method, $\mu = 1$							
err. < 5%	96.6 \pm 2.3	82.0 \pm 6.0	67.6 \pm 2.7	50.6 \pm 8.4	37.0 \pm 3.9	23.8 \pm 4.9	17.6 \pm 3.0
‘Adam’ SGD method, $\mu = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ (default parameters)							
err. < 5%	98.6 \pm 1.3	79.6 \pm 5.3	51.6 \pm 8.1	28.0 \pm 3.1	15.6 \pm 3.2	6.2 \pm 0.8	1.8 \pm 1.3
‘Adam’ SGD method, $\mu = 10^{-2.5}$, $\beta_1 = 0.9$, $\beta_2 = 0.99$ (fine-tuned parameters)							
err. < 5%	96.2 \pm 1.3	85.6 \pm 4.6	72.8 \pm 5.8	49.4 \pm 3.5	37.8 \pm 2.9	26.6 \pm 3.5	14.0 \pm 1.9

Table 1: The average number of runs (executions of the code) out of 100, where the normalised root-mean-square error is below 5%, 10% and 20% for different amplitude of the perturbations of the initial guess (parameter α), for the GN method (lines 3-5), for the NK method with $\mu = 0.1$ (lines 9-11), for the NK method with $\mu = 1$ (line 13), for the ‘Adam’ SGD method with default parameters (line 15) and for the ‘Adam’ SGD method with fine-tuned parameters (line 17). The average number of runs out of 100, where the convergence criteria of the GN method is fulfilled for different α (line 6). The average normalised root-mean-square error in %, calculated only for the converged runs of the GN method, for different α (line 7). In each cell, the mean and the standard deviation across 5 ensembles, each consisting of 100 runs, are given.

in equation (32) are such that outer function Φ is highly non-linear and non-monotonous. Furthermore, since the model is the ridge function, there is no redundancy — it consists of only one addend that must be identified precisely, otherwise the error is large. This means that the convergence of the methods to local minima (with the identified parameters that are different from the exact model parameters) is insufficient in terms of accuracy.

In the comparison above, all runs of the GN method are used, even if the convergence criteria is not fulfilled and the iterations are stopped after the maximum number of iterations has been reached. An alternative approach is disregarding the runs where the convergence criteria is not fulfilled. However, this implies that if the convergence criteria is fulfilled, then the result should be considered to be accurate. In this case, first, the percentage of runs with the ‘converged’ results drops with α , as shown in line 6 of table 1. Second, the average RMSE, calculated only for the ‘converged’ runs, still increases with α and reaches almost 25% for $\alpha = 2.8$, as shown in line 7 of table 1. This means that the runs converge to local minima, providing insufficiently accurate result.

Table 1 also provides a comparison between the NK and the ‘Adam’ SGD methods. The latter has three fine-tuning parameters (β_1, β_2, μ), and it is possible to adjust those such that the method performs almost identically to the NK method. In particular, the results with the default parameters (as proposed in [34]) and the results with the fine-tuned parameters are provided in table 1. The fine-tuning has been done by the authors ‘knowing’ the exact solution; thus, it is only provided to show that the optimal values of the parameters exist. The performance of the ‘Adam’ SGD method with the default parameters is close to the performance of the NK method with $\mu = 0.1$ (although the NK method is noticeably better when α is large). The ‘Adam’ SGD method with the fine-tuned parameters is close to the NK method with $\mu = 1$. The small difference is not surprising given the relationship between the NK and the SGD methods.

There is significant sensitivity of the performance of the ‘Adam’ SGD method to the value of the learning rate. In particular, in table 2, the same accuracy metric (the average percentages of runs where the RMSE is below 5%) is compared across the methods with μ varied. For the considered problem, the ‘Adam’ SGD method provides good results only when the learning rate is between 10^{-2} and 10^{-3} .

Newton-Kaczmarz (NK) method					
μ	10^0	$10^{-0.5}$	10^{-1}	$10^{-1.5}$	10^{-2}
err. < 5%, $\alpha = 0.4$	96.6 ± 2.3	98.4 ± 0.9	98.4 ± 0.9	97.0 ± 2.2	92.6 ± 1.3
err. < 5%, $\alpha = 1.6$	50.6 ± 8.4	46.4 ± 6.5	35.8 ± 4.1	24.2 ± 1.3	9.4 ± 1.8
‘Adam’ SGD method, $\beta_1 = 0.9, \beta_2 = 0.999$					
μ	$10^{-1.5}$	10^{-2}	$10^{-2.5}$	10^{-3}	$10^{-3.5}$
err. < 5%, $\alpha = 0.4$	31.2 ± 3.6	75.6 ± 3.0	97.0 ± 1.2	98.6 ± 1.3	95.6 ± 3.0
err. < 5%, $\alpha = 1.6$	21.0 ± 3.7	45.6 ± 3.9	44.8 ± 7.0	28.0 ± 3.1	7.6 ± 2.7

Table 2: The average number of runs (executions of the code) out of 100, where the normalised root-mean-square error is below 5% for different amplitude of the perturbations of the initial guess, parameter α , and for different values of parameter μ for the NK method (lines 3-4) and for the ‘Adam’ SGD method (lines 7-8). In each cell, the mean and the standard deviation across 5 ensembles, each consisting of 100 runs, are given.

For $\mu = 10^{-3.5}$, there is a significant drop in the number of accurate runs when $\alpha = 1.6$, i.e. when the initial guess for the solution is relatively far from the exact solution. In this case, the performance of the method is even worse than that of the GN method (when compared to line 3 of table 1). For $\mu = 10^{-1.5}$, there is a very low percentage of accurate runs when $\alpha = 0.4$, i.e. when the initial guess for the solution is very close to the exact solution. For ‘good’ initial guesses, the Newton-type methods almost always converge well, while the ‘Adam’ SGD method with $\mu = 10^{-1.5}$ probably overshoots the optimal solution update due to the large step size.

The NK method has only one fine-tuning parameter — μ . In the ‘Adam’ SGD method it is referred to as the learning rate, while in the NK method it is more logical to call it the regularisation parameter (as done in the preceding sections). There is an intuitive understanding that for the exact data (i.e. the input-output data corresponding to the consistent system of non-linear equations that has a unique solution), the optimal value of μ in the NK method is 1. This follows from the error analysis (appendix A.1), where it is shown that such value of μ corresponds to the fastest error decay. As the example of this section deals with the exact data, $\mu = 1$ straightforwardly gives the best performance of the NK method, as seen in tables 1 and 2. Using $\mu < 1$ makes the method robust for the case of noisy data (some insights how this works are provided in [18], where such regularisation has been studied within the context of the Urysohn model with the Kaczmarz-based identification algorithm).

The percentages in tables 1 and 2 are calculated based on 5 ensembles, each consisting of 100 runs. Within each ensemble, the number of runs with the error below a given threshold is calculated; the mean and the standard deviation across the ensembles and are given in the tables. The same is done for the percentage of the converged runs of the GN method and the corresponding RMSEs.

5.2 Example 2: Approximation of a non-linear function

The purpose of this example is to demonstrate that the NK method can indeed solve the parameter identification problem for the Kolmogorov-Arnold model approximating a non-linear multivariate function. The code of this example is available at the GitHub repository¹. The ‘full’ model is taken with the piecewise-linear basis functions.

The following non-linear function of $m = 5$ inputs is chosen:

$$y_i = \frac{2 + 2X_3^i}{3\pi} \left(\arctan \left(20 \left(X_1^i - \frac{1}{2} + \frac{X_2^i}{6} \right) \exp(X_5^i) \right) + \frac{\pi}{2} \right) + \frac{2 + 2X_4^i}{3\pi} \left(\arctan \left(20 \left(X_1^i - \frac{1}{2} - \frac{X_2^i}{6} \right) \exp(X_5^i) \right) + \frac{\pi}{2} \right). \quad (35)$$

The datasets consisting of $N = 10^4$ records for the training and of $N_{\text{val}} = 2000$ records for the validation are generated. This involves taking uniformly-distributed random inputs $X_j^i \sim \text{unif}(0, 1)$ and calculating outputs y_i using equation (35). Using the training datasets, the Kolmogorov-Arnold models with $n = 5$ inner and $s = 7$ outer piecewise-linear basis functions are identified.

¹<https://github.com/andrewpolar/UStandardM>

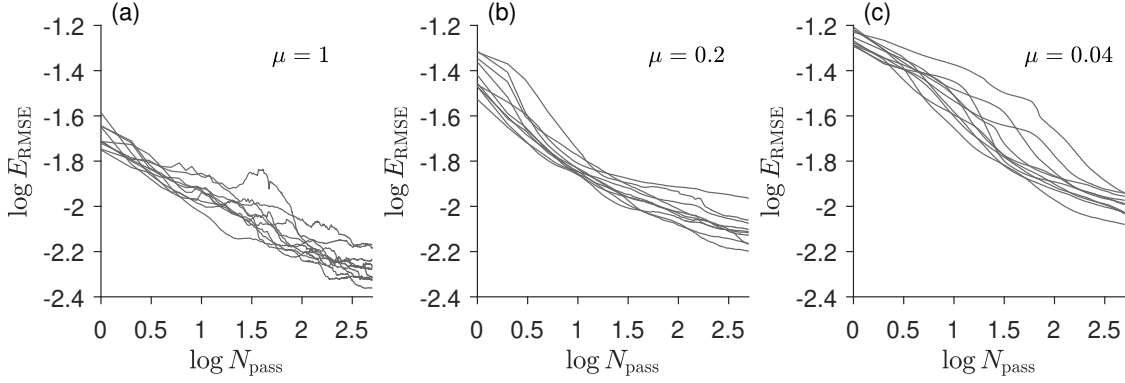


Figure 1: The dependence of the RMSE on the number of passes through the data for different values of μ .

Since the identification method is iterative, it is interesting to study the convergence rate with respect to the number of iterations. The number of iterations should be large, but each iteration is computationally cheap, as each iteration of the Newton-Kaczmarz method operates only with one record. Therefore, the algorithm goes through all the training records consecutively multiple times. After each pass¹ through the training data, the RMSE is calculated using the validation data. Thus, the RMSE can be plotted as a function of the number of passes through the data, which is illustrated in figure 1. Three different values of the regularisation parameter are used; for each case, 10 runs with different data and with different initial guesses for the parameters are performed.

It can be seen that for any μ , the method converges approximately logarithmically with respect to the number of passes, i.e. there is approximately linear dependence between $\log E_{\text{RMSE}}$ and $\log N_{\text{pass}}$. However, smaller μ leads to higher absolute value of E_{RMSE} for the same number of passes. The practical purpose of decreasing μ is to filter out the noise, which is absent in this example. A more detailed study of the noise filtering abilities of the Kaczmarz method has been performed for the case of the Urysohn model in [18]. Finally, it should be mentioned that the model obtained using the proposed identification technique is accurate, with the RMSE of around 0.5% for $\mu = 1$ after 500 passes, as seen in figure 1.

5.3 Example 3: Solution of a PDE

The purpose of this example is to demonstrate that the Kolmogorov-Arnold model with the Newton-Kaczmarz parameter identification method can indeed be used for data-driven solution of PDEs. The ‘full’ model is taken with the spline basis functions.

To test the approach properly, it is necessary to consider a PDE with an analytical solution. The following function is taken:

$$u(x, y) = x \exp(y - y^2). \quad (36)$$

It is easy to construct a second-order PDE, solution of which is function u above:

$$\frac{\partial^2 u}{\partial x^2} + 2xy \frac{\partial^2 u}{\partial x \partial y} + \frac{\partial^2 u}{\partial y^2} + 2x \frac{\partial u}{\partial x} - \frac{\partial u}{\partial y} = 0. \quad (37)$$

Domain $[0, 2] \times [0, 2]$ is considered, resulting in the boundary conditions for the PDE:

$$u|_{x=0} = 0, \quad u|_{x=2} = 2 \exp(y - y^2), \quad u|_{y=0} = x, \quad u|_{y=2} = x \exp(-2).$$

The structure of the Kolmogorov-Arnold model is chosen to consist of $n = 7$ inner and $s = 7$ outer spline basis functions. Regularisation parameter $\mu = 0.2$ is used. Batch model updates consisted of $N_b = 4$ boundary points (one point per boundary condition) and $N_i = 20$ internal points. Thus, weight factor ν is shifted towards the internal data.

The identification process starts with random parameters and converges as the number of batch updates increases. To illustrate this, the number of batch updates is varied and, in each case, 10 models

¹In machine learning literature, one ‘pass’ through the training data is also called ‘epoch’.

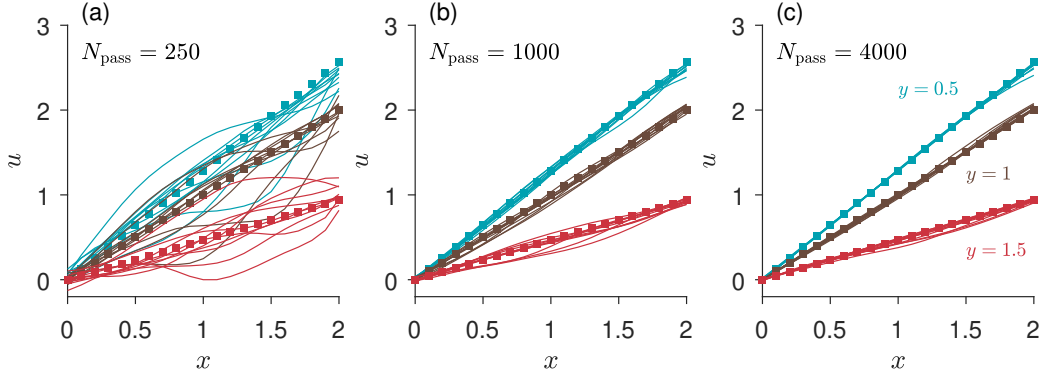


Figure 2: The sections of function $u(x, y)$ at $y = 0.5$ (blue), $y = 1$ (brown), and $y = 1.5$ (red). Solid lines are the numerical solutions of the second-order PDE, which are obtained by training the Kolmogorov-Arnold model using the different number of passes (batch updates). Symbols show the analytical solution.

are trained. Figure 2 shows the comparison between the analytical solution and the numerical solutions given by the Kolmogorov-Arnold model. It can be seen that the spread of the curves corresponding to the numerical solutions logarithmically decreases with the increase of the number of batch updates. The numerical solutions converge to the analytical solution — after 10^5 batch updates, the RMSE of $0.098\% \pm 0.021\%$ is obtained (the numbers are the mean and the standard deviation across the ensemble of 5 runs). The latter is calculated according to equation (34) on the structured grid with the spatial step of 0.1.

6 Conclusions

The Kolmogorov-Arnold model is a convenient tool for data modelling tasks that consist of mapping the input vectors into the output scalars. In the present paper, it has been proposed to represent the underlying functions constituting the model via the basis functions and the model parameters and to determine these parameters using the Newton-Kaczmarz (NK) method, which is locally convergent.

The approach has been tested using the numerical examples. First, it has been compared to the straightforward application of the Gauss-Newton method to identify the parameters of the ridge function model. A challenging example has been selected, where the non-linearity of the outer function creates convergence difficulties when the initial guess for the parameters is far from their exact values. It has been shown that the NK method is much less sensitive to the selection of the initial guess; therefore, it is more robust from a practical point of view. The NK method has also been compared to the ‘Adam’ stochastic gradient descent method (SGD), and it has been shown that although it is possible to get the same performance out of both methods, it is subjectively easier to fine-tune the NK method to achieve the best performance¹.

The second example has demonstrated that a challenging non-linear function can be approximated accurately by the Kolmogorov-Arnold model with the piecewise-linear basis functions, and the model parameters can be obtained efficiently using the proposed approach.

Furthermore, it has been shown that the Kolmogorov-Arnold model can be used for data-driven solution of partial differential equations (PDEs). There is crucial difference between the proposed approach and employing neural networks for such task — the underlying functions of the Kolmogorov-Arnold model are taken as products of the model parameters and the basis functions. The choice of the basis functions with compact support, similarly to e.g. finite-element methods, can be viewed as discretisation of the spatial and the temporal coordinates — the feature that physics-informed neural networks (PINNs) lack. Thus, first, the resulting approach can be viewed as a hybrid between the data-driven and the mesh-based solvers; second, it can be expected that the proposed approach will perform better than PINNs, which can struggle to approximate PDEs with strong nonlinearities, sharp gradients, or interfaces [35].

Model training via the NK method is ideally-suited for parallel implementation. Each outer function of the model is independent of other outer functions; therefore, within each step, the outer functions

¹In particular, the ‘Adam’ SGD method has three fine-tuning parameters, while the NK method has only one.

and the corresponding to them inner functions can be updated in parallel on different CPU cores. A possibility of such parallel implementation is a significant advantage of the proposed approach.

Finally, it should be mentioned that many more examples of application of the Urysohn and the Kolmogorov-Arnold models to various synthetic and real datasets, as well as all codes of all examples above, are available at the GitHub repositories of the authors¹². The readers will also find there MATLAB, Python and C# implementations of the models and the proposed identification techniques.

Appendix A The Newton-Kaczmarz method

For system of non-linear equations $L(Z) = 0$, where $L : D \rightarrow \mathbb{R}^N$ is continuously differentiable in open set $D \subset \mathbb{R}^C$, the Newton-Kaczmarz method [25–27] consists in iterations

$$Z^{q+1} = Z^q - \mu \frac{L^i(Z^q)}{\|\nabla L^i(Z^q)\|^2} \nabla L^i(Z^q), \quad \nabla L^i(Z^q) = \left[\frac{\partial L^i}{\partial Z_1} \quad \dots \quad \frac{\partial L^i}{\partial Z_C} \right]^T \Big|_{Z=Z^q}, \quad (38)$$

where function L^i denotes the i -th component of vector function L , scalar Z_j denotes the j -th component of vector Z , parameter $\mu \in (0, 2)$ is introduced for regularisation, and vector Z^q is the approximation of the solution at iteration q . Index i changes each iteration. The method requires an initial guess and continues until some convergence criteria are reached.

This method can also be interpreted as an iterative scheme, where at each step, a single equation is linearised and one step of the Kaczmarz method [22, 23] is performed. Indeed, to write an iterative scheme, an expression for Z^{q+1} must be found, assuming that Z^q is known. Considering single equation $L^i(Z^{q+1}) = 0$, the linearisation is performed:

$$L^i(Z^{q+1}) = L^i(Z^q) + \nabla L^i(Z^q)^T \Delta Z + O(\|\Delta Z\|^2) = 0, \quad \Delta Z = Z^{q+1} - Z^q. \quad (39)$$

When the second-order term is neglected, linear equation

$$M^i \Delta Z = Y_i \quad (40)$$

is obtained, where $M^i = \nabla L^i(Z^q)^T$ and $Y_i = -L^i(Z^q)$. Equation (40) with respect to ΔZ describes a hyperplane in \mathbb{R}^C . Using the key idea of the Kaczmarz method (i.e. the projection descend), the origin is projected onto this hyperplane to obtain the solution:

$$\Delta Z = \frac{Y_i}{\|M^i\|^2} M^{iT}. \quad (41)$$

Adding regularisation parameter μ to limit the magnitude of change of Z over one step, leads to equation (38).

A.1 On local convergence of the Newton-Kaczmarz method

As the Newton-Kaczmarz method is a version of the Newton's method, it is not expected to be globally convergent (apart from isolated cases, e.g. when $L(Z)$ are linear equations), but it is locally convergent, given a 'good' initial guess. It is easy to see this by performing a simple error analysis.

System of equations $L(Z) = 0$ is assumed to be consistent, i.e. there exists at least one \bar{Z} that satisfies all equations. It is now assumed that $L : D \rightarrow \mathbb{R}^N$ is twice continuously differentiable in open set $D \subset \mathbb{R}^C$. Initial guess Z^0 is assumed to be sufficiently close to \bar{Z} . Using the Taylor's expansion results in

$$L^i(\bar{Z}) = -Y_i + M^i E^q + R_i = 0, \quad E^q = \bar{Z} - Z^q, \quad R_i = O(\|E^q\|^2), \quad (42)$$

where the notation introduced in equation (40) is utilised. Using iterative formula (38) and subtracting \bar{Z} from both sides results in

$$-E^{q+1} = -E^q + \mu \frac{Y_i}{\|M^i\|^2} M^{iT}. \quad (43)$$

¹<https://github.com/andrewpolar>

²<https://github.com/mpoluektov>

It is useful to remind that for any vector A , the norm is defined as $\|A\|^2 = A^T A$, if A is a column, and $\|A\|^2 = A A^T$, if A is a row. Equation (43) then gives

$$\begin{aligned}\|E^{q+1}\|^2 &= \|E^q\|^2 + \mu^2 \frac{Y_i^2}{\|M^i\|^2} - 2\mu \frac{Y_i}{\|M^i\|^2} M^i E^q = \\ &= \|E^q\|^2 - \frac{\mu}{\|M^i\|^2} ((2 - \mu) Y_i^2 - 2Y_i R_i),\end{aligned}\tag{44}$$

where $M^i E^q$ is substituted from (42). For convergence, it is sufficient that

$$(2 - \mu) Y_i^2 - 2Y_i R_i > 0,$$

as error norm $\|E^q\|^2$ decreases each iteration in this case. If $0 < \mu \leq 1$, then

$$(2 - \mu) Y_i^2 - 2Y_i R_i \geq Y_i^2 - 2Y_i R_i = (M^i E^q)^2 - R_i^2,\tag{45}$$

where Y_i is substituted from (42). Assuming that ∇L^i is finite in the neighbourhood of solution \bar{Z} leads to

$$(M^i E^q)^2 = O(\|E^q\|^2),$$

the first and the second terms of the right-hand side of equation (45) become second-order and fourth-order, respectively, with respect to $\|E^q\|$. For small $\|E^q\|$, the second-order term is dominant; hence, the right-hand side of equation (45) is positive when Z^q is close to \bar{Z} . The latter means that the iterative scheme is locally convergent.

It is useful to note that when equation (44) is considered, it can be seen that the leading-order term in the right-hand side has multiplier $(2\mu - \mu^2)$. Thus, neglecting the higher-order terms, the fastest error decay is achieved when this multiplier takes the maximum value, i.e. when $\mu = 1$. The purpose of introducing μ within the iterative scheme is to make the method robust for noisy data, i.e. when system of equations $L(Z) = 0$ is not consistent, in which case, the decrease of μ acts as noise filtering.

A.2 On relation to the stochastic gradient descent method

It can be useful to note that iterative formula (38) has the same structure as the well-known stochastic gradient descent (SGD) method. The SGD method is written for minimisation of objective function

$$Q(Z) = \frac{1}{N} \sum_{i=1}^N Q^i(Z),$$

where Z is the vector of parameters to be found by the minimisation of $Q(Z)$. Addends $Q^i(Z)$ are typically associated with the i -th record of the dataset. The SGD method is then written as the following iterative formula:

$$Z^{q+1} = Z^q - \tilde{\mu} \nabla Q^i(Z^q),$$

where $\tilde{\mu}$ is referred to as the “learning rate”. Index i changes each iteration. Iterative formula (38) is obtained by defining

$$Q^i(Z) = \frac{1}{2} (L^i(Z))^2, \quad \tilde{\mu} = \mu \frac{1}{\|\nabla L^i(Z^q)\|^2}.$$

Hence, the iterations of the Newton-Kaczmarz method can be viewed as the iterations of the SGD method with an adaptive learning rate. It should be emphasised that the convergence analysis of the SGD method with convex and non-convex objective functions is well studied, e.g. [36].

Appendix B Derivatives of the Kolmogorov-Arnold representation

The implementation of the Newton-Kaczmarz method to find the parameters of the Kolmogorov-Arnold model requires expressions for the derivatives of the output by the parameters. Furthermore, solving PDEs

using the proposed approach additionally requires expressions for the mixed derivatives of the output by the parameters and the inputs. These derivatives are summarised in this section. The differentiation with respect to the inputs is limited to the second order.

It is useful to start by writing the Kolmogorov-Arnold model with substituted basis function — substituting equations (10) and (11) into (5):

$$\hat{y} = \sum_{k,l} G_{kl} \psi^l \left(\sum_{j,p} H_{kjp} \phi^p(X_j) \right),$$

where superscript i is omitted for brevity, as well as the ranges of the indices used in the summations. The following short notation is introduced for the basis functions evaluated at the inputs and at the intermediate variables:

$$\Lambda_{pj} = \phi^p(X_j), \quad \Psi_{lk} = \psi^l \left(\sum_{j,p} H_{kjp} \Lambda_{pj} \right).$$

The derivatives of the basis functions evaluated at the same points are denoted with the prime symbol, for example,

$$\Lambda'_{pj} = \left. \frac{d\phi^p(x)}{dx} \right|_{x=X_j}.$$

The derivatives of \hat{y} by the parameters and by the inputs are the following:

$$\begin{aligned} \frac{\partial \hat{y}}{\partial G_{kl}} &= \Psi_{lk}, \\ \frac{\partial \hat{y}}{\partial H_{kjp}} &= \sum_l G_{kl} \Psi'_{lk} \Lambda_{pj}, \\ \frac{\partial \hat{y}}{\partial X_a} &= \sum_{k,l,p} G_{kl} H_{kap} \Psi'_{lk} \Lambda'_{pa}. \end{aligned}$$

The second-order derivatives are the following:

$$\begin{aligned} \frac{\partial^2 \hat{y}}{\partial X_a \partial G_{kl}} &= \sum_p H_{kap} \Psi'_{lk} \Lambda'_{pa}, \\ \frac{\partial^2 \hat{y}}{\partial X_a \partial H_{kjp}} &= \sum_l G_{kl} \delta_{aj} \Psi'_{lk} \Lambda'_{pa} + \sum_{l,u} G_{kl} H_{kau} \Psi''_{lk} \Lambda'_{ua} \Lambda_{pj}, \\ \frac{\partial^2 \hat{y}}{\partial X_a \partial X_b} &= \sum_{k,l,p,u} G_{kl} H_{kap} H_{kbu} \Psi''_{lk} \Lambda'_{pa} \Lambda'_{ub} + \sum_{k,l,p} G_{kl} H_{kap} \delta_{ab} \Psi'_{lk} \Lambda''_{pa}, \end{aligned}$$

where δ_{ab} is the Kronecker delta. The third-order derivatives are the following:

$$\begin{aligned} \frac{\partial^3 \hat{y}}{\partial X_a \partial X_b \partial G_{kl}} &= \sum_{p,u} H_{kap} H_{kbu} \Psi''_{lk} \Lambda'_{pa} \Lambda'_{ub} + \sum_p H_{kap} \delta_{ab} \Psi'_{lk} \Lambda''_{pa}, \\ \frac{\partial^3 \hat{y}}{\partial X_a \partial X_b \partial H_{kjp}} &= \sum_{l,u} G_{kl} H_{kbu} \delta_{aj} \Psi''_{lk} \Lambda'_{pa} \Lambda'_{ub} + \sum_{l,u} G_{kl} H_{kau} \delta_{bj} \Psi''_{lk} \Lambda'_{pb} \Lambda'_{ua} + \\ &+ \sum_{l,u,v} G_{kl} H_{kbu} H_{kav} \Psi'''_{lk} \Lambda'_{va} \Lambda'_{ub} \Lambda_{pj} + \sum_l G_{kl} \delta_{aj} \delta_{ab} \Psi'_{lk} \Lambda''_{pa} + \sum_{l,u} G_{kl} H_{kau} \delta_{ab} \Psi''_{lk} \Lambda''_{ua} \Lambda_{pj}. \end{aligned}$$

Appendix C Identification of the ridge function using the Gauss-Newton method

The ridge function model with the outer function represented via the parameters and the basis functions is given by

$$\hat{y}_i = \sum_{l=1}^s G_l \psi^l \left(\sum_{j=1}^m c_j X_j^i \right), \quad (46)$$

which is obtained by substituting equation (11) with omitted index k into equation (6). The sum of squares of the residuals is introduced as

$$S = \frac{1}{2} \sum_{i=1}^N (\hat{y}_i - y_i)^2. \quad (47)$$

The Gauss-Newton method consists in minimisation of S with respect to the model parameters using the Newton-Raphson method and neglecting the second-order derivative terms. However, to obtain the most accurate Hessian matrix of S and to maximise the performance of the method, the second-order derivatives are not neglected here. Thus, although the method deviates from the classical Gauss-Newton algorithm, for the purposes of this paper, it is still referred to as such.

The minimisation of S gives

$$\frac{\partial S}{\partial G_l} = \sum_{i=1}^N r_i \frac{\partial r_i}{\partial G_l} = 0, \quad \frac{\partial S}{\partial c_j} = \sum_{i=1}^N r_i \frac{\partial r_i}{\partial c_j} = 0, \quad l \in \{1, \dots, s\}, \quad j \in \{1, \dots, m\}, \quad (48)$$

with the following additionally-introduced quantities:

$$r_i = \hat{y}_i - y_i, \quad \theta_i = \sum_{j=1}^m c_j X_j^i, \quad \eta^l = \frac{d\psi^l}{dt}, \quad (49)$$

$$\frac{\partial r_i}{\partial G_l} = \psi^l(\theta_i), \quad \frac{\partial r_i}{\partial c_j} = X_j^i \sum_{l=1}^s G_l \eta^l(\theta_i). \quad (50)$$

The evaluation of the second derivatives of S leads to

$$\frac{\partial^2 S}{\partial G_l \partial G_p} = \sum_{i=1}^N \frac{\partial r_i}{\partial G_l} \frac{\partial r_i}{\partial G_p}, \quad \frac{\partial^2 S}{\partial G_l \partial c_j} = \sum_{i=1}^N \left(\frac{\partial r_i}{\partial G_l} \frac{\partial r_i}{\partial c_j} + r_i \frac{\partial^2 r_i}{\partial G_l \partial c_j} \right), \quad (51)$$

$$\frac{\partial^2 S}{\partial c_j \partial c_k} = \sum_{i=1}^N \left(\frac{\partial r_i}{\partial c_j} \frac{\partial r_i}{\partial c_k} + r_i \frac{\partial^2 r_i}{\partial c_j \partial c_k} \right), \quad p \in \{1, \dots, s\}, \quad k \in \{1, \dots, m\}, \quad (52)$$

with the following additionally-introduced quantities:

$$\frac{\partial^2 r_i}{\partial G_l \partial c_j} = X_j^i \eta^l(\theta_i), \quad \frac{\partial^2 r_i}{\partial c_j \partial c_k} = X_j^i X_k^i \sum_{l=1}^s G_l \xi^l(\theta_i), \quad \xi^l = \frac{d^2 \psi^l}{dt^2}. \quad (53)$$

Using the obtained relations, equations (48) are solved using the Newton-Raphson method, consisting in the following iterative scheme:

$$Z^{q+1} = Z^q - \mu M^{-1} L|_{Z=Z^q}, \quad (54)$$

where Z is the vector containing the model parameters, with superscript indicating the iteration number, M and L are the matrix and the vector containing the second and the first derivatives of S with respect to the model parameters, respectively, and μ is the regularisation parameter added for the robustness of

the scheme¹. These quantities are defined as

$$Z = [G_1 \quad \dots \quad G_s \quad c_1 \quad \dots \quad c_m]^T, \quad (55)$$

$$L = \left[\frac{\partial S}{\partial G_1} \quad \dots \quad \frac{\partial S}{\partial G_s} \quad \frac{\partial S}{\partial c_1} \quad \dots \quad \frac{\partial S}{\partial c_m} \right]^T, \quad (56)$$

$$M_{l,p} = \frac{\partial^2 S}{\partial G_l \partial G_p}, \quad M_{l,s+j} = M_{s+j,l} = \frac{\partial^2 S}{\partial G_l \partial c_j}, \quad M_{s+j,s+k} = \frac{\partial^2 S}{\partial c_j \partial c_k}, \quad (57)$$

where comma is added between the subscripts indicating the components of matrix M for clarity. The iterations are performed until either the maximum number of iterations is reached or $\|Z^{q+1} - Z^q\| < \delta$ is fulfilled².

Appendix D Summary of the ‘Adam’ SGD method

The ‘Adam’ SGD method has been proposed in [34]. Using the notation of appendix A.2, the method constitutes the following iterative sequence:

$$G^q = \nabla Q^i(Z^q), \quad (58)$$

$$A^{q+1} = \beta_1 A^q + (1 - \beta_1) G^q, \quad (59)$$

$$B_j^{q+1} = \beta_2 B_j^q + (1 - \beta_2) (G_j^q)^2, \quad j \in \{1, \dots, r\}, \quad (60)$$

$$\hat{A}^{q+1} = \frac{A^{q+1}}{1 - (\beta_1)^q}, \quad \hat{B}^{q+1} = \frac{B^{q+1}}{1 - (\beta_2)^q}, \quad (61)$$

$$Z_j^{q+1} = Z_j^q - \mu \frac{\hat{A}_j^{q+1}}{\sqrt{\hat{B}_j^{q+1} + \varepsilon}}, \quad j \in \{1, \dots, r\}, \quad (62)$$

where $Z^q \in \mathbb{R}^C$ is the approximation of the sought solution at iteration q (as in the previous sections), $G^q \in \mathbb{R}^C$ is the gradient of objective function Q^i (index i changes each iteration), vectors A^q and B^q are referred to as the biased first and second moment estimates, respectively, vectors \hat{A}^q and \hat{B}^q are the corresponding bias-corrected moments, parameters $\beta_1, \beta_2 \in [0, 1)$ are referred to as the ‘forgetting’ factors, small scalar parameter ε is introduced to avoid division by zero. Scalar B_j^q denotes the j -th component of vector B^q (the same for A^q and G^q). The initial values for A^q and \hat{B}^q are the all-zero vectors.

References

- [1] F. Musil, A. Grisafi, A. P. Bartók, C. Ortner, G. Csányi, and M. Ceriotti. Physics-inspired structural representations for molecules and materials. *Chemical Reviews*, 121(16):9759–9815, 2021.
- [2] V. I. Arnold. On functions of three variables. *Doklady Akademii Nauk SSSR*, 114(4):679–681, 1957.
- [3] A. N. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114(5):953–956, 1957.
- [4] D. Bryant. *Analysis of Kolmogorov’s superposition theorem and its implementation in applications with low and high dimensional data*. PhD thesis, University of Central Florida, 2008.
- [5] X. Liu. *Kolmogorov superposition theorem and its applications*. PhD thesis, Imperial College London, 2015.
- [6] D. A. Sprecher. A numerical implementation of Kolmogorov’s superpositions. *Neural Networks*, 9(5):765–772, 1996.

¹In the example of section 5.1, the initial value of μ is 0.1. As soon as $\|Z^{q+1} - Z^q\| < 0.1$ is fulfilled, the value of μ is changed 1.

²In the example of section 5.1, a more strict convergence criteria, also requiring $\|L^q\| < \delta$, were investigated additionally (results are not shown). However, no significant change of the results was observed.

- [7] D. A. Sprecher. A numerical implementation of Kolmogorov's superpositions II. *Neural Networks*, 10(3):447–457, 1997.
- [8] M. Köppen. On the training of a Kolmogorov network. In J. R. Dorronsoro, editor, *Artificial Neural Networks — ICANN 2002*, pages 474–479. Springer Berlin Heidelberg, 2002.
- [9] J. Braun and M. Griebel. On a constructive proof of Kolmogorov's superposition theorem. *Constructive Approximation*, 30(3):653–675, 2009.
- [10] J. Actor and M. J. Knepley. An algorithm for computing Lipschitz inner functions in Kolmogorov's superposition theorem. arXiv:1712.08286, 2017.
- [11] J. Actor. Computation for the Kolmogorov superposition theorem. Master's thesis, Rice University, 2018.
- [12] M. Nees. Approximative versions of Kolmogorov's superposition theorem, proved constructively. *Journal of Computational and Applied Mathematics*, 54(2):239–250, 1994.
- [13] B. Igel'nik and N. Parikh. Kolmogorov's spline network. *IEEE Transactions on Neural Networks*, 14(4):725–733, 2003.
- [14] M. Coppejans. On Kolmogorov's representation of functions of several variables by functions of one variable. *Journal of Econometrics*, 123(1):1–31, 2004.
- [15] A. Polar and M. Poluektov. A deep machine learning algorithm for construction of the Kolmogorov-Arnold representation. *Engineering Applications of Artificial Intelligence*, 99:104137, 2021.
- [16] M. A. Krasnoselskii. *Topological methods in the theory of nonlinear integral equations*. Pergamon, 1964.
- [17] V. V. Krylov. Modeling the internal structure of dynamical systems from input-output relationships (abstract realization theory) 2. *Automation and Remote Control*, 45(3):277–288, 1984.
- [18] M. Poluektov and A. Polar. Modelling non-linear control systems using the discrete Urysohn operator. *Journal of the Franklin Institute*, 357:3865–3892, 2020.
- [19] T. J. Hastie and R. J. Tibshirani. *Generalized additive models*. Chapman and Hall/CRC, 1990.
- [20] A. Ismayilova and V. E. Ismailov. On the Kolmogorov neural networks. *Neural Networks*, 176:106333, 2024.
- [21] V. E. Ismailov. A review of some results on ridge function approximation. *Azerbaijan Journal of Mathematics*, 3(1):3–51, 2013.
- [22] S. Kaczmarz. Angenäherte auflösung von systemen linearer gleichungen. *Bulletin international de l'Académie polonaise des sciences et des lettres. Classe des sciences mathématiques et naturelles. Série A, Sciences mathématiques*, 35:355–357, 1937.
- [23] R. P. Tewarson. Projection methods for solving sparse linear systems. *The Computer Journal*, 12(1):77–80, 1969.
- [24] C. Popa. Convergence rates for Kaczmarz-type algorithms. *Numerical Algorithms*, 79(1):1–17, 2018.
- [25] K.-H. Meyn. Solution of underdetermined nonlinear equations by stationary iteration methods. *Numerische Mathematik*, 42(2):161–172, 1983.
- [26] J. M. Martinez. Solving systems of nonlinear equations by means of an accelerated successive orthogonal projections method. *Journal of Computational and Applied Mathematics*, 16(2):169–179, 1986.
- [27] J. M. Martinez and R. J. B. de Sampaio. Parallel and sequential Kaczmarz methods for solving underdetermined nonlinear equations. *Journal of Computational and Applied Mathematics*, 15(3):311–321, 1986.

- [28] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley, 2009.
- [29] H. van Deventer, P. J. van Rensburg, and A. Bosman. KASAM: Spline additive models for function approximation. *arXiv:2205.06376*, 2022.
- [30] S. Lane, M. Flax, D. Handelman, and J. Gelfand. Multi-layer perceptrons with B-spline receptive field functions. In R. P. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 684–692, 1990.
- [31] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.
- [32] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [33] M. Raissi, A. Yazdani, and G. E. Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.
- [34] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- [35] V. Dwivedi, N. Parashar, and B. Srinivasan. Distributed learning machines for solving forward and inverse problems in partial differential equations. *Neurocomputing*, 420:299–316, 2021.
- [36] B. Fehrman, B. Gess, and A. Jentzen. Convergence rates for the stochastic gradient descent method for non-convex objective functions. *Journal of Machine Learning Research*, 21(136):1–48, 2020.