A recipe for learning Variably Scaled Kernels via Discontinuous Neural Networks

Journal of Computational and Applied Mathematics 469 (2025) 116653

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

A recipe for learning Variably Scaled Kernels via Discontinuous Neural Networks

G. Audone, F. Della Santa, E. Perracchione *, S. Pieraccini

Dipartimento di Scienze Matematiche "Giuseppe Luigi Lagrange", Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129, Turin, Italy
Gruppo Nazionale per il Calcolo Scientifico INdAM, Piazzale Aldo Moro 5, 00185, Rome, Italy

ARTICLE INFO

MSC:
65D15
41A05
68Q32

Keywords:
Adaptive Kernel Bases
Variably Scaled Kernels
Meshfree approximation
Discontinuous Neural Networks
Deep learning

ABSTRACT

The efficacy of interpolating via Variably Scaled Kernels (VSKs) is known to be dependent on the definition of a *proper* scaling function, but no numerical recipes to construct it are available. Previous works suggest that such a function should mimic the target one, but no theoretical evidence is provided. This paper fills both the gaps: it proves that a scaling function reflecting the target one may lead to enhanced approximation accuracy, and it provides a user-independent tool for learning the scaling function by means of Discontinuous Neural Networks (δNN), i.e., NNs able to deal with possible discontinuities. Numerical evidence supports our claims, as it shows that the key features of the target function can be clearly recovered in the learned scaling function.

1. Introduction

Variably Scaled Kernels (VSKs) were introduced in 2015 [1] with the main purpose of improving the stability of kernel interpolants thanks to the definition of a scaling function which was originally thought of as a continuous version of the shape parameter, whose value affects accuracy and stability indicators (see e.g. [2–6]). Later works show that the use of VSKs might be valuable when the target function of the scattered data interpolation problem is characterized by discontinuities. Indeed in such cases, defining a scaling function with the same key features of the target one, e.g. the same discontinuities, results in a VSK basis sharing the same discontinuities with the target; e.g., we refer the reader to [7–9]. This fact suggests using VSKs with scaling functions which somehow mimic the target ones as, intuitively, this allows to directly insert in the kernel basis itself some prior information about the sought solution, and especially in applied fields, this turns out to be a relevant issue [10]. Nevertheless, this intuition lacks both theoretical and numerical evidence.

This work addresses these issues by providing theoretical justification for the claim, demonstrating by means of the Lebesgue functions that a scaling function reflecting the behavior of the target may lead to enhanced approximation accuracy. As a further confirmation of our claim, we propose a user-independent way of choosing the scaling function by training a Discontinuous Neural Network (δNN) able to learn also possibly discontinuous scaling functions directly from data [11]. Our results confirm the theoretical findings: the learned scaling function closely resembles the target function. This data-driven approach, namely in what follows δNN-VSKs, offers a user-independent and adaptive solution for meshfree approximation tasks.

* Corresponding author at: Dipartimento di Scienze Matematiche "Giuseppe Luigi Lagrange", Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129, Turin, Italy.
E-mail addresses: gianluca.audone@polito.it (G. Audone), francesco.dellasanta@polito.it (F. Della Santa), emma.perracchione@polito.it (E. Perracchione), sandra.pieraccini@polito.it (S. Pieraccini).

https://doi.org/10.1016/j.cam.2025.116653
Received 8 October 2024; Received in revised form 24 January 2025
Available online 1 April 2025

The paper is organized as follows. In Section 2 we recall the basics of kernel-based approximation. Section 3 is the core of the paper presenting the theory and the numeric intuition behind the δNN-VSKs. Experiments and conclusions are respectively offered in Sections 4 and 5.

2. Brief review of kernel-based interpolation

We consider symmetric reproducing kernels $\kappa : \Omega \times \Omega \longrightarrow \mathbb{R}$ for

$$H_\kappa = \text{span}\left\{\kappa(\cdot, \boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \subset \mathbb{R}^d\right\},$$

which is a pre-Hilbert space equipped with an inner product $(\cdot,\cdot)_{H_\kappa}$. Those are kernels so that (refer e.g. to [12, Definition 2.6, p. 32]) $\kappa(\boldsymbol{x}, \boldsymbol{z}) = \kappa(\boldsymbol{z}, \boldsymbol{x})$, $\boldsymbol{x}, \boldsymbol{z} \in \Omega$, with $\kappa(\cdot, \boldsymbol{x}) \in H_\kappa$, and

$$(f, \kappa(\cdot, \boldsymbol{x}))_{H_\kappa} = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega.$$

The native space $\mathcal{N}_\kappa$ of the kernel κ is then defined as the completion of H_κ with respect to the norm $\|\cdot\|_{H_\kappa} = \sqrt{(\cdot,\cdot)_{H_\kappa}}$, and hence for all $f \in H_\kappa$ we have that $\|f\|_{\mathcal{N}_\kappa} = \|f\|_{H_\kappa}$; here and throughout all this section, we refer the reader to the monographs [12,13] for more details.

A common subclass of reproducing kernels, which is frequently used in the framework of scattered data interpolation problems, is given by radial kernels. To any radial kernel, we are able to uniquely associate a Radial Basis Function (RBF) $\phi : \mathbb{R}_+ \longrightarrow \mathbb{R}$, where $\mathbb{R}_+ = [0, +\infty)$, and (possibly) a shape parameter $\varepsilon > 0$ such that, for all $\boldsymbol{x}, \boldsymbol{z} \in \Omega$,

$$\kappa(\boldsymbol{x}, \boldsymbol{z}) = \kappa_\varepsilon(\boldsymbol{x}, \boldsymbol{z}) = \phi(\varepsilon\|\boldsymbol{x} - \boldsymbol{z}\|_2) = \phi_\varepsilon(\|\boldsymbol{x} - \boldsymbol{z}\|_2) = \phi(r),$$

where $r = \|\boldsymbol{x} - \boldsymbol{z}\|_2$. For the sake of notation simplicity, we may omit the dependence of the kernel on the shape parameter, which is also referred to as scale parameter in the machine learning literature.

In order to formally introduce the scattered data interpolation problem, we consider a function $f : \Omega \longrightarrow \mathbb{R}$, with $\Omega \subset \mathbb{R}^d$, and an associated set of functional values $F_n = \{f(\boldsymbol{x}_i)\}_{i=1}^n$ sampled at a data nodes set $X_n = \{\boldsymbol{x}_i\}_{i=1}^n \subset \Omega$. Given these sets, the goal consists in computing an approximation of the unknown function f, called P_f, by imposing the n interpolation constraints, i.e.,

$$P_f(\boldsymbol{x}_i) = f(\boldsymbol{x}_i), \quad i = 1, \dots, n. \tag{1}$$

Assuming that the interpolating function P_f can be written as

$$P_f(\boldsymbol{x}) = \sum_{k=1}^{n} c_k \kappa\left(\boldsymbol{x}, \boldsymbol{x}_k\right), \quad \boldsymbol{x} \in \Omega, \tag{2}$$

the coefficients $\{c_i\}_{i=1}^n$ are the solution of the linear system

$$\mathsf{K}\boldsymbol{c} = \boldsymbol{f},$$

where $\boldsymbol{c} = \left(c_1, \dots, c_n\right)^\intercal$, $\boldsymbol{f} = \left(f_1, \dots, f_n\right)^\intercal$, being $f_i = f(\boldsymbol{x}_i)$, and $\mathsf{K}_{ik} = \kappa\left(\boldsymbol{x}_i, \boldsymbol{x}_k\right)$, $i, k = 1, \dots, n$. Letting κ be a symmetric and strictly positive definite kernel, the collocation matrix is positive definite, and equivalently to (2), we may write the nodal form of the interpolant as

$$P_f(\boldsymbol{x}) = \boldsymbol{\kappa}^\intercal(\boldsymbol{x})\mathsf{K}^{-1}\boldsymbol{f}, \quad \boldsymbol{x} \in \Omega,$$

where

$$\boldsymbol{\kappa}^\intercal(\boldsymbol{x}) = \left(\kappa(\boldsymbol{x}, \boldsymbol{x}_1), \dots, \kappa(\boldsymbol{x}, \boldsymbol{x}_n)\right).$$

Even if here we focus on interpolation, we point out that the kernel theory is not limited to interpolation, indeed the Ansatz conditions could be generalized to any linearly independent set of continuous linear functionals $\{\chi_1, \dots, \chi_n\}$, so that Eq. (1) becomes $\chi_i P_f = \chi_i f$, $i = 1, \dots, n$.

3. Investigating and learning the scaling function for VSKs

The idea behind VSKs, first introduced in the interpolation context in [1] and later adapted to least squares approximation [14], is to introduce a new kernel basis. This basis can be interpreted as a kind of feature augmentation strategy. Specifically, the original aim was to replace the shape parameter of the basis function with a continuous shape function. The authors of [1] demonstrated that this approach is equivalent to defining a standard kernel in a higher-dimensional space—effectively adding features via a scaling function denoted here by $\bar{f}$—and then projecting it back onto the original dimension.

More formally, let $\Sigma \subset \mathbb{R}^m$, $m > 0, m \in \mathbb{N}$, and let $\kappa : \tilde{\Omega} \times \tilde{\Omega} \longrightarrow \mathbb{R}$, $\tilde{\Omega} = \Omega \times \Sigma \subset \mathbb{R}^{d+m}$, be a continuous radial positive definite kernel. Given a scaling function $\bar{f} : \Omega \longrightarrow \Sigma$, a VSK $\kappa_{\bar{f}} : \Omega \times \Omega \longrightarrow \mathbb{R}$ is defined as

$$\kappa_{\bar{f}}(\boldsymbol{x}, \boldsymbol{z}) = \kappa\left((\boldsymbol{x}, \bar{f}(\boldsymbol{x})), (\boldsymbol{z}, \bar{f}(\boldsymbol{z}))\right) =: \kappa(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{z}}),$$

for $\boldsymbol{x}, \boldsymbol{z} \in \Omega$. Then, the VSK interpolant is simply given by $P_f^{\bar{f}}(\boldsymbol{x}) := P_f(\tilde{\boldsymbol{x}})$, or, equivalently,

$$P_f^{\bar{f}}(\boldsymbol{x}) := \sum_{k=1}^{n} c_k \kappa_{\bar{f}}(\boldsymbol{x}, \boldsymbol{x}_k) = \sum_{k=1}^{n} c_k \kappa(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}_k). \tag{3}$$

As radial kernels only depend on the distance of the nodes, the VSK interpolant turns out to be easy to implement. Indeed, the coefficients $c_1, \dots, c_n$ of the interpolant $P_f^{\bar{f}}$ are computed by solving

$$\underbrace{\begin{pmatrix} \kappa(\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_1) & \cdots & \kappa(\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_n) \\ \vdots & & \vdots \\ \kappa(\tilde{\boldsymbol{x}}_n, \tilde{\boldsymbol{x}}_1) & \cdots & \kappa(\tilde{\boldsymbol{x}}_n, \tilde{\boldsymbol{x}}_n) \end{pmatrix}}_{\mathsf{K}_{\bar{f}}} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}. \tag{4}$$

In the discrete setting, the approximant is then evaluated at N points denoted by $\Xi = \{\boldsymbol{\xi}_k\}_{k=1}^N$. Practically, we consider $m = 1$, and hence the VSK interpolant, after defining the augmented sets of nodes

$$\{\tilde{\boldsymbol{x}}_i\}_{i=1}^n = \{(\boldsymbol{x}_i, \bar{f}(\boldsymbol{x}_i))\}_{i=1}^n, \quad \text{and} \quad \{\tilde{\boldsymbol{\xi}}_k\}_{k=1}^N = \{(\boldsymbol{\xi}_k, \bar{f}(\boldsymbol{\xi}_k))\}_{k=1}^N \subset \mathbb{R}^{d+1}, \tag{5}$$

and computing the coefficients as in (4), is defined as the projection on $\mathbb{R}^d$ of a standard interpolant in $\mathbb{R}^{d+1}$, i.e,

$$P_f^{\bar{f}}(\boldsymbol{\xi}_k) = P_f(\tilde{\boldsymbol{\xi}}_k) = \sum_{i=1}^{n} c_i \kappa(\tilde{\boldsymbol{\xi}}_k, \tilde{\boldsymbol{x}}_i), \tag{6}$$

with $k = 1, \dots, N$.

The advantages of using VSKs, i.e., their easy implementation and the fact that we can superimpose some prior information, are briefly summarized in the following remark.

Remark 3.1. Because of their easy implementation, VSKs have already been used in many applications whose goals consist in recovering functions with steep gradients, high oscillations and/or discontinuities. In case of discontinuities, intuitively, if the scaling function $\bar{f}$ is chosen so that it has discontinuities at the same edges of f, so is for the radial kernel $\kappa_{\bar{f}}$ and hence for $P_f^{\bar{f}}$. We refer the reader to [7] for further details on the characterization of native spaces induced by the discontinuous kernels.

Hence, following the intuition of Remark 3.1, we are interested in proposing bounds that highlight the dependence of the error on the scaling function (see Section 3.1) and in providing an algorithm to automatically define such a scaling function (see Section 3.2). In order to make the presentation clearer we want to stress the fact that the scheme used to find the nearly optimal scaling function $\bar{f}$, based on the so-called δNNs, could be thought of as an independent step of the analysis carried out in the next subsection. Indeed, the use of δNNs is primarily due to numerically support the claim that the scaling function should resemble the target one, even if we do not have theoretical evidence about the fact that such choice is definitely the best one. Nevertheless, independently of the results reported in the next subsection, our δNN algorithm returns a learned function $\bar{f}$ that definitely has the same key features of the target.

Before going into details and in order to set our final goal, we summarize, in Flowchart 1, the main steps needed to compute the VSK interpolant (well-known in literature already), with the difference that here the scaling function will be no longer an input of the scheme but the result of the δNN optimization procedure.

3.1. VSKs: investigating the scaling function via Lagrange bases

Following the insights of Remark 3.1, the authors of previous papers (e.g., see [7,9,10]) claim that if the scaling function $\bar{f}$ *somehow mimics* the target f or some of its key features, then the VSK interpolant outperforms the standard one. Nevertheless, no theoretical evidence has been provided and no numerical tests were conducted to determine the *optimal* scaling function. For the first item, we point out that for any set of distinct points $\{\boldsymbol{x}_i\}_{i=1}^n$, there exist functions (known as Lagrange or cardinal bases) $\varphi_j^{\bar{f}} \in \text{span}\{\kappa_{\bar{f}}(\cdot, \boldsymbol{x}_j), j = 1, \dots, n\}$ so that the VSK interpolant can be written as

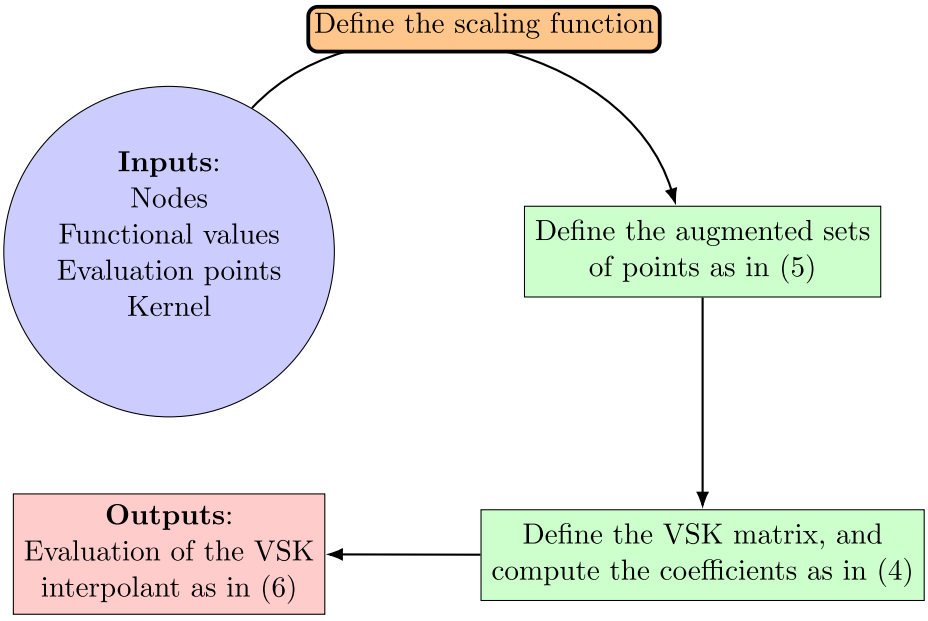
$$P_f^{\bar{f}}(\boldsymbol{x}) = \sum_{j=1}^{n} f_j \varphi_j^{\bar{f}}(\boldsymbol{x}),$$

being $\varphi_j^{\bar{f}}(\boldsymbol{x}) := \varphi_j(\tilde{\boldsymbol{x}})$. Such functions are computed by solving the linear system

$$\mathsf{K}_f \tilde{\boldsymbol{\varphi}} = \tilde{\boldsymbol{\kappa}},$$

where $\tilde{\boldsymbol{\varphi}} = (\varphi_1(\tilde{\boldsymbol{x}}), \dots, \varphi_n(\tilde{\boldsymbol{x}}))^\intercal$ and the vector on the right-hand side is given by $\tilde{\boldsymbol{\kappa}} = (\kappa(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}_1), \dots, \kappa(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}_n))^\intercal$.

Such Lagrange basis allows us to formally introduce error bounds for the VSK interpolant that depend on the *similarities* between $\bar{f}$ and f.



Flowchart 1. Diagram for the VSK-based algorithm.

Proposition 3.1. *Let $\bar{f}$ be a scaling function and let $\kappa_{\bar{f}} : \Omega \times \Omega \longrightarrow \mathbb{R}$ be the VSK corresponding to the strictly positive definite and symmetric kernel $\kappa : \tilde{\Omega} \times \tilde{\Omega} \longrightarrow \mathbb{R}$. Let $P_f^{\bar{f}} : \Omega \longrightarrow \mathbb{R}$ be the VSK interpolant. Then, the following point-wise error bound holds true*

$$\left|\left(f - P_f^{\bar{f}}\right)(\boldsymbol{x})\right| \leq \left|\left(f - P_{\bar{f}}^{\bar{f}}\right)(\boldsymbol{x})\right| + \|\boldsymbol{f} - \bar{\boldsymbol{f}}\|_\infty \lambda_{\bar{f}}(\boldsymbol{x}), \tag{7}$$

where

$$\lambda_{\bar{f}}(\boldsymbol{x}) = \sum_{j=1}^{n} |\varphi_j^{\bar{f}}(\boldsymbol{x})|,$$

is the Lebesgue function for the VSK interpolant, and $\boldsymbol{f} - \bar{\boldsymbol{f}} = (f(\boldsymbol{x}_1) - \bar{f}(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_n) - \bar{f}(\boldsymbol{x}_n))^\intercal = (f_1 - \bar{f}_1, \dots, f_n - \bar{f}_n)^\intercal$.

Proof. We trivially have that

$$\left|\left(f - P_f^{\bar{f}}\right)(\boldsymbol{x})\right| = \left|\left(f - P_{\bar{f}}^{\bar{f}}\right)(\boldsymbol{x}) + \left(P_{\bar{f}}^{\bar{f}} - P_f^{\bar{f}}\right)(\boldsymbol{x})\right| \leq \left|\left(f - P_{\bar{f}}^{\bar{f}}\right)(\boldsymbol{x})\right| + \left|\left(P_f^{\bar{f}} - P_{\bar{f}}^{\bar{f}}\right)(\boldsymbol{x})\right|.$$

To bound the second term in the above inequality, we observe that

$$\left|\left(P_f^{\bar{f}} - P_{\bar{f}}^{\bar{f}}\right)(\boldsymbol{x})\right| = \left|P_{f-\bar{f}}^{\bar{f}}(\boldsymbol{x})\right| = \sum_{j=1}^{n} |(f_j - \bar{f}_j)\varphi_j^{\bar{f}}(\boldsymbol{x})| \leq \|\boldsymbol{f} - \bar{\boldsymbol{f}}\|_\infty \lambda_{\bar{f}}(\boldsymbol{x}),$$

and the thesis follows. □

Equivalently to (7), we might write

$$\left|\left(f - P_f^{\bar{f}}\right)(\boldsymbol{x})\right| \leq \left|\left(f - P_{\bar{f}}^{\bar{f}}\right)(\boldsymbol{x})\right| + \|\boldsymbol{f} - \bar{\boldsymbol{f}}\|_\infty \Lambda_{\bar{f}},$$

where $\Lambda_{\bar{f}}$ is the Lebesgue constant given by [15]

$$\Lambda_{\bar{f}} = \sup_{\boldsymbol{x} \in \mathbb{R}^d} \sum_{j=1}^{n} |\varphi_j^{\bar{f}}(\boldsymbol{x})|.$$

We point out that to get rid of the second term in the right-hand side of (7), we only need to set the nodal values of $\bar{f}$ equal to the ones of f; this could be trivially achieved since the interpolation conditions are known. Nevertheless, the first term suggests that the scaling function $\bar{f}$ should be close to f on the whole domain Ω, as in this way $P_{\bar{f}}^{\bar{f}}$ should approach $P_f^{\bar{f}}$ and f in turn, following classical error bounds based either on the power function or on the fill-distance (see [13, Theorem 11.4 and Theorem 11.13]). Moreover, we stress that taking $f \equiv \bar{f}$ is an unrealistic choice, as f is unknown, and we do not have any theoretical nor numerical evidence that $|(f - P_f^f)(\boldsymbol{x})| \leq |(f - P_f^{\bar{f}})(\boldsymbol{x})|$ for each $\bar{f}$ and $\boldsymbol{x} \in \Omega$.

Finally, in terms of native spaces error bounds, we have the following corollary.

Corollary 3.1. *Let $\kappa_{\bar{f}} : \Omega \times \Omega \longrightarrow \mathbb{R}$ be the VSK associated to $\kappa : \tilde{\Omega} \times \tilde{\Omega} \longrightarrow \mathbb{R}$ and $f \in N_\kappa^{\bar{f}}$, then*

$$\left| \left(f - P_{\bar{f}}^{\bar{f}} \right)(\boldsymbol{x}) \right| \leq \left(\|f\|_{N_\kappa^{\bar{f}}} + \bar{\boldsymbol{f}}^\intercal \mathsf{K}_{\bar{f}}^{-1} \bar{\boldsymbol{f}} \right) \|\kappa_{\bar{f}}\|_{N_\kappa^{\bar{f}}} + \|\boldsymbol{f} - \bar{\boldsymbol{f}}\|_\infty \lambda_{\bar{f}}(\boldsymbol{x}).$$

Proof. In [1, Theorem 2] the authors proved that the native spaces N_κ and $N_\kappa^{\bar{f}}$ are isometric. Then, by the Cauchy–Schwarz inequality and the reproducing property, we have that

$$\left| \left(f - P_{\bar{f}}^{\bar{f}} \right)(\boldsymbol{x}) \right| = \left(f - P_{\bar{f}}^{\bar{f}} \, , \, \kappa_{\bar{f}}(\cdot, \boldsymbol{x}) \right)_{N_\kappa^{\bar{f}}} \leq \|f - P_{\bar{f}}^{\bar{f}}\|_{N_\kappa^{\bar{f}}} \|\kappa_{\bar{f}}\|_{N_\kappa^{\bar{f}}} \leq \left(\|f\|_{N_\kappa^{\bar{f}}} + \|P_{\bar{f}}^{\bar{f}}\|_{N_\kappa^{\bar{f}}} \right) \|\kappa_{\bar{f}}\|_{N_\kappa^{\bar{f}}},$$

and as $\|P_{\bar{f}}^{\bar{f}}\|_{N_\kappa^{\bar{f}}} = \bar{\boldsymbol{f}}^\intercal \mathsf{K}_{\bar{f}}^{-1} \bar{\boldsymbol{f}}$, the thesis follows. □

We conclude this subsection by pointing out that we have some theoretical evidence of the fact that the function $\bar{f}$ should approximate f. However, it is worth clarifying that no optimal properties have been proved and, as the function f is in practice unknown, we do not have any criteria or algorithm to automatically define the *best* scaling function. Hence we implement a NN that confirms our findings and returns a safe (i.e., nearly-optimal) scaling function $\bar{f}$. More precisely, numerically we will observe that for the practical implementation of the δNN-VSKs the demanding pointwise convergence of the function $\bar{f}$ to f can typically be relaxed, as reproducing some of the key features of the target usually leads to accurate approximations. Further comments are provided later in Section 4.1.

3.2. δNN-VSKs: learning nearly-optimal scaling functions via δNNs

The NN that we consider is referred to as δNN and its first usage [11] was devoted to detect the discontinuity interfaces while approximating discontinuous functions. Such NN is characterized by the introduction of learnable discontinuities in the layers, legitimated by the universal approximation theorems [16], and hence might return a discontinuous output. Before introducing the discontinuous layers that characterize δNNs, we recall that a standard Fully-Connected (FC) layer, from n_{in} units to n_{out} units, can be described by the function

$$\mathcal{L}(\boldsymbol{x}) = \boldsymbol{\sigma}\left(\mathsf{W}^\intercal \boldsymbol{x} + \boldsymbol{b} \right), \quad \boldsymbol{x} \in \mathbb{R}^{n_{\text{in}}}, \tag{8}$$

where $\mathsf{W} \in \mathbb{R}^{n_{\text{in}} \times n_{\text{out}}}$ is the weight matrix, $\boldsymbol{b} \in \mathbb{R}^{n_{\text{out}}}$ is the bias vector, and $\boldsymbol{\sigma}$ is the element-wise application of the activation function $\sigma : \mathbb{R} \to \mathbb{R}$.

The characterizing function of a discontinuous layer is a function $\delta\mathcal{L} : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$ such that

$$\mathcal{L}(\boldsymbol{x}) = \boldsymbol{\sigma}\left(\mathsf{W}^\intercal \boldsymbol{x} + \boldsymbol{b} \right) + \boldsymbol{\alpha} \odot \boldsymbol{\mathcal{H}}\left(\mathsf{W}^\intercal \boldsymbol{x} + \boldsymbol{b} \right), \quad \boldsymbol{x} \in \mathbb{R}^{n_{\text{in}}}, \tag{9}$$

where $\boldsymbol{\mathcal{H}}$ is the element-wise application of the Heaviside function and $\boldsymbol{\alpha} \in \mathbb{R}^{n_{\text{out}}}$ is the vector of *trainable discontinuity jumps*. In brief, for each $j = 1, \dots, n_{\text{out}}$, Eq. (9) is equivalent to add to the jth element of (8) a jump of height α_j if the jth row of $\mathsf{W}^\intercal \boldsymbol{x} + \boldsymbol{b}$ is non-negative (otherwise, no jump is applied). See Fig. 1 for a visual example of discontinuous layers with $n_{\text{in}} = n_{\text{out}} = 1$.

The main property of a NN embedded with discontinuous layers (i.e., a δNN) is the ability to return both an approximation of the target function and an approximation of its discontinuity interfaces. Specifically, in a δNN, the weight matrices and the bias vectors of the discontinuous layers have the role of learning the discontinuity interfaces, while the discontinuity jump parameters have the role of learning the height of the jump along the discontinuity interfaces. For further theoretical or practical details about δNNs, we refer the reader to [11].

The reason why we are interested in δNNs for defining the scaling function $\bar{f}$ lies in the fact that, in principle, $\bar{f}$ can be discontinuous, and we already have numerical evidence of the fact that if $\bar{f}$ mimics the discontinuities of f then the Gibbs phenomenon observed in the approximations is drastically reduced. The novelty in this case is that we do not learn the discontinuities of f, we learn directly the function $\bar{f}$ without imposing any constraints on its discontinuities.

We define the δNN as a parametric scaling function for the VSKs, aiming to find the *optimal* one for the interpolant of f; i.e., we train a δNN $\bar{f}_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^m$, with trainable parameters denoted by $\boldsymbol{\theta} \in \mathbb{R}^p$, and we adapt the interpolation coefficients $\boldsymbol{c} \in \mathbb{R}^n$ by minimizing the interpolation error of the VSK interpolant.

More precisely, we build a NN model $\mathcal{M}_{\boldsymbol{\theta},\boldsymbol{\eta}} : \mathbb{R}^d \to \mathbb{R}^m \times \mathbb{R}^n$ pairing the δNN $\bar{f}_{\boldsymbol{\theta}}$ and a model $\boldsymbol{c}_{\boldsymbol{\eta}} := \boldsymbol{c}(\boldsymbol{\eta})$ for the coefficients, with $\boldsymbol{\theta} \in \mathbb{R}^p, \boldsymbol{\eta} \in \mathbb{R}^q$ trainable parameters, such that

$$\mathcal{M}_{\boldsymbol{\theta},\boldsymbol{\eta}}(\boldsymbol{x}) = (\bar{f}_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{c}_{\boldsymbol{\eta}}),$$

for each $\boldsymbol{x} \in \mathbb{R}^d$.

The training of $\mathcal{M}_{\boldsymbol{\theta},\boldsymbol{\eta}}$ is obtained by minimizing the loss

$$\ell_{X_n}(\boldsymbol{\theta}, \boldsymbol{\eta}) := \frac{1}{n} \|\boldsymbol{f} - \mathsf{K}_{\bar{f}_{\boldsymbol{\theta}}} \boldsymbol{c}_{\boldsymbol{\eta}}\|_2^2 = \frac{1}{n} \sum_{i=1}^{n} \left(f_i - \sum_{k=1}^{n} c_k(\boldsymbol{\eta}) \, \kappa\left((\boldsymbol{x}_i, \bar{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) \, , \, (\boldsymbol{x}_k, \bar{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) \right) \right)^2, \tag{10}$$

where a global minimum $(\boldsymbol{\theta}^*, \boldsymbol{\eta}^*) \in \mathbb{R}^p \times \mathbb{R}^q$ for (10) is such that $\ell_{X_n}(\boldsymbol{\theta}^*, \boldsymbol{\eta}^*) = 0$ (i.e., interpolation satisfied).

We drive the reader's attention towards the fact that (10) is defined using the whole training data $\{(\boldsymbol{x}_i, f_i)\}_{i=1}^n$ and not using random mini-batches sampled from them; indeed, since the main task is the interpolation, we prefer a deterministic training procedure instead of a stochastic one.

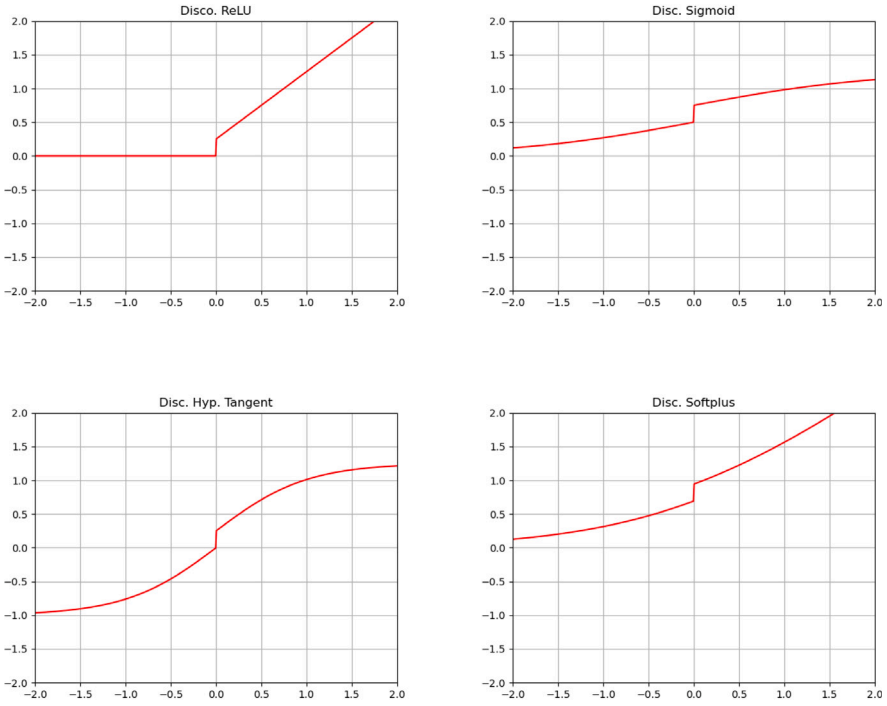


Fig. 1. Examples of discontinuous layers ($n_{\text{in}} = n_{\text{out}} = 1$) with different activation functions.

Remark 3.2. Let $(\boldsymbol{\theta}^{\text{fin}}, \boldsymbol{\eta}^{\text{fin}})$ be the parameters obtained at the end of the training by minimizing (10). It is likely that the interpolation conditions corresponding to $(\boldsymbol{\theta}^{\text{fin}}, \boldsymbol{\eta}^{\text{fin}})$ are only approximately satisfied; indeed, stopping criteria are adopted and it is possible that the training stops before reaching a given tolerance. Therefore, it is recommended to build a model based on $\boldsymbol{\theta}^{\text{fin}}$ and an updated version of $\boldsymbol{c}$, namely $\boldsymbol{c} = \mathsf{K}^{-1}_{\bar{f}_{\boldsymbol{\theta}^{\text{fin}}}} \boldsymbol{f}$. In case an iterative solver is used to solve the linear system, $\boldsymbol{c}_{\boldsymbol{\eta}^{\text{fin}}}$ can be used as a starting guess.

Summarizing the procedure, the δNN-VSK interpolant for a target function f is constructed as in (3) where the coefficients $\boldsymbol{c} = (c_1, \dots, c_n)^\intercal$ are computed as reported in Remark 3.2 above.

The numerical experiments will show that the scaling function learned via the δNN-VSK method is, indeed, a nearly-optimal choice and that it closely mimics the target's shape. Therefore, in the next subsection, we propose an alternative way for computing the VSK interpolant, namely the VSK-f approach, that directly learns the target and uses such an approximation as a scaling function.

3.3. VSKs-f: using an approximation of f as scaling function

Another possible approach for defining a proper scaling function can be more focused on the claim stated at the end of Section 3.1. Specifically, Proposition 3.1 suggests that a good approximation of the target can be used as a "safe" scaling function for a VSK interpolant. Therefore, we can renounce to find a nearly-optimal scaling function minimizing a loss like (10), preferring to train a δNN for directly approximating f, instead. Nonetheless, we recall that the quantity of data typically used for interpolation (i.e., n) rarely is enough for a fine approximation of f via δNN models; therefore, finding a VSK interpolant of f is still necessary.

In details, given a δNN $\bar{f}_{\boldsymbol{\theta}}$, we train it for approximating f, using the interpolation data as the training set and via a classic stochastic training procedure. Therefore, the trained model $\bar{f}_{\boldsymbol{\theta}^{\text{fin}}}$ will be an approximation of f, and the larger is n (i.e., the amount of interpolation data used for the training) the finer the approximation quality.

In conclusion, the VSK-f method consists in computing the VSK interpolant (3) of f using $\bar{f}_{\boldsymbol{\theta}^{\text{fin}}}$ as scaling function.

Remark 3.3. The main advantage of using VSKs-f with respect to δNN-VSKs is that training $\bar{f}_{\boldsymbol{\theta}}$ for approximating f is easier than training the model $\mathcal{M}_{\boldsymbol{\theta},\boldsymbol{\eta}}$ that learns simultaneously the coefficients of the approximant and the scaling function. Of course, the price of an easier procedure is the risk of using a safe scaling function, which however is not tailored for the considered kernel basis.

4. Numerical experiments

We test the proposed procedure both on synthetic datasets (Section 4.1) obtained by sampling three selected functions at given sets of quasi-uniform nodes, and on data describing the real-world phenomenon of the acetone's phase transition (Section 4.2). For

all the numerical experiments, we have used the same residual δNN architecture and the same training options. Details are provided in Appendix.

4.1. Tests with synthetic datasets

In the numerical experiments that follow we take as test functions the well-known *Franke's* function

$$f_1(x_1,x_2)=\frac{3}{4}e^{-[(9x_1-2)^2+(9x_2-2)^2]/4}+\frac{3}{4}e^{-(9x_1+1)^2/49-(9x_2+1)/10}+\frac{3}{4}e^{-[(9x_1-2)^2+(9x_2-2)^2]/4}+\frac{3}{4}e^{-(9x_1+1)^2/49-(9x_2+1)/10},$$

which is continuous, being the sum of exponential terms, and the discontinuous test functions:

$$f_2(x_1,x_2)=\begin{cases}-e^{(-(x_1-0.5)^2+(x_2-0.5)^2)}, & \text{if} \quad \|\boldsymbol{x}-(0.5,0.5)\|_2^2\geq 0.08\\ \sin(x_1)+4\sin(x_2), & \text{otherwise},\end{cases}$$

and

$$f_3(x_1,x_2)=\begin{cases}\sin(0.4\pi(x_1+x_2)), & \text{if} \quad x_2\geq e^{x_1},\\ \sin(0.7\pi(x_1+x_2))-4, & \text{if} \quad x_2<e^{x_1}-1,\\ \sin(\pi(x_1+x_2))+4, & \text{if} \quad \text{otherwise}.\end{cases}$$

We sample these functions at $n=[27^2,33^2,39^2]$ scattered Halton data in $[0,1]^2$ and we interpolate them with respect to:

- classical Fixed Scale Kernels (FSKs), see (2);
- δNN-VSKs (see Section 3.2);
- VSKs-f (see Section 3.3).

The dataset size considered here is rather small, indeed, in order to reconstruct the target when the dataset is large enough, one may use local and greedy methods [17,18] or directly the δNN.

For all the VSK interpolants, we consider a scalar scaling function $\bar{f}_{\theta}$; i.e., $\bar{f}_{\theta}:\mathbb{R}^2\to\mathbb{R}$.

All the interpolants are evaluated on a $\sqrt{N}\times\sqrt{N}$ regular grid of evaluation points $\boldsymbol{\xi}_k$, $k=1,\dots,N$, with $\sqrt{N}=100$. In particular, we compute the following performance indicators for the interpolation accuracy of a given approximant P:

- Mean Absolute Error (MAE):

$$\mathrm{MAE}(P)=\frac{1}{N}\sum_{k=1}^{N}|f(\boldsymbol{\xi}_k)-P(\boldsymbol{\xi}_k)|;$$

- Mean Squared Error (MSE):

$$\mathrm{MSE}(P)=\frac{1}{N}\sum_{k=1}^{N}(f(\boldsymbol{\xi}_k)-P(\boldsymbol{\xi}_k))^2.$$

Moreover, we also compute the Structural Similarity Index Measure (SSIM) [19] with respect to the greyscale image I_f generated by the evaluations of f on the $\sqrt{N}\times\sqrt{N}$ grid and the greyscale image I_P generated by the evaluations of the interpolant P on the same grid. The value $\mathrm{SSIM}(I_f,I_P)$ is in $[-1,1]$, where 1 denotes perfect similarity, 0 denotes no similarity, and -1 denotes perfect anti-correlation. The SSIM index is an accuracy score popular in many imaging applications, such as image processing, super-resolution, image resizing, image rotation and registration, and whose convergence in the context of image interpolation has been recently studied in [20]. This index is computed via the *tf.image.ssim* function of Tensorflow [21]. For more details about SSIM refer to [19,21].

The kernel used to approximate the function f_1 (independently of the method used) is the Gaussian C^{∞} function:

$$\phi_{\varepsilon}(r)=e^{-\varepsilon^2r^2},$$

while for f_2 and f_3 we use the Matérn C^2 kernel

$$\phi_{\varepsilon}(r)=e^{-\varepsilon r}(1+\varepsilon r).$$

The results are reported in Table 1. All results are reproducible as the Python code is freely available at

https://github.com/Fra0013To/VSKlearning/tree/paper2024.

In Figs. 2–7, we depict all the obtained surfaces. In particular, we plot both their top view, which is essential to clearly see the improvements in the discontinuous case, and their 3D counterpart false colored with the absolute error.

In all the considered test cases, we observe (see Table 1) that the δNN-VSK and VSK-f methods behave better than FSKs. Precisely, for the continuous test function f_1, the FSKs already give good approximations, while in the discontinuous cases the δNN-VSK and VSK-f methods provide a consistent improvement on the approximation accuracy (see Table 1 and Figs. 2–7); moreover, in the latter case, the improvement in terms of SSIM (refer to Table 1) of the VSKs-based methods is rather impressive. Such results are

Table 1
MAEs, MSEs, and SSIMs for the test functions f_1, f_2, and f_3.

	Interpolation			MAE			MSE			SSIM		
	n	Kernel	ϵ	FSKs	δNN-VSKs	VSKs-f	FSKs	δNN-VSKs	VSKs-f	FSKs	δNN-VSKs	VSKs-f
f_1	729	Gauss.	0.6	4.99e–2	8.98e–3	3.57e–3	4.24e–3	1.42e–4	2.43e–5	0.8712	0.9820	0.9943
	1089	Gauss.	1.2	2.38e–2	3.48e–3	2.11e–3	9.88e–4	2.33e–5	8.25e–6	0.9533	0.9969	0.9980
	1521	Gauss.	4.8	3.60e–4	3.60e–4	2.47e–4	3.84e–7	3.83e–7	2.70e–7	0.9997	0.9997	0.9999
f_2	729	Mat.C^2	0.06	1.97e–1	2.09e–2	1.65e–2	5.76e–2	3.72e–3	3.84e–3	0.5883	0.9074	0.9253
	1089	Mat.C^2	0.12	1.45e–1	8.06e–3	1.29e–2	3.37e–2	2.12e–3	2.18e–3	0.5885	0.9556	0.9486
	1521	Mat.C^2	0.48	6.65e–2	5.68e–3	8.37e–3	1.33e–2	1.89e–3	2.37e–3	0.6453	0.9696	0.9559
f_3	729	Mat.C^2	0.06	2.09e–1	4.30e–2	1.40e–2	5.84e–2	3.82e–3	2.42e–3	0.6594	0.7928	0.9495
	1089	Mat.C^2	0.12	1.34e–1	3.65e–2	1.98e–2	3.08e–2	3.58e–3	2.71e–3	0.6411	0.8993	0.9312
	1521	Mat.C^2	0.48	6.28e–2	1.00e–2	6.48e–3	1.20e–2	1.66e–3	1.19e–3	0.7242	0.9657	0.9760

due to the fact that when data are smooth, VSKs might only improve already quite good approximation scores returned by FSKs. On the other hand, in the discontinuous cases, the hardest task is to identify and reconstruct the faults; therefore, any discontinuous scaling function sharing the same discontinuities of the target provides a positive contribution.

For all the cases, we recall that no information is provided about the continuity or discontinuity of the target, nor is information provided for selecting the scaling function. Therefore, the continuous case is a rather interesting test case, because it shows the flexibility of the δNN architecture in learning also continuous targets. In other words, if one has no information about the smoothness of data, then the δNN-VSK approach is a practical and robust choice for approximating the target.

Moreover, concerning the scaling function learned via δNNs, we observe that for both the continuous test (f_1) and all the other discontinuous tests, the δNN-VSK method learns nearly-optimal scaling functions that mimic the targets, while the VSK-f method supports the theoretical claims of the safety in selecting a scaling function that not only mimics but approximates the target; refer to the bottom rows of Figs. 2, 3, and 4.

Summarizing, from the analysis of the interpolation results, we observe the following:

- Our VSKs-based methods provide reliable approximations both for the continuous case and for the discontinuous test functions (see Table 1 and Figs. 5–7);
- The δNN-VSK method proves to be able to learn a nearly-optimal scaling function automatically, and as a numerical confirmation of our claims, the identified scaling functions $\bar{f}_{\theta^{\text{fin}}}$ are such that they *mimic* the target f. This mimicking behavior can be detailed (e.g., see Fig. 2) or "generic" (e.g., see Figs. 3–4);
- The results obtained via the VSK-f method support our theoretical claim reported at the end of Section 3.1; i.e., using a scaling function $\bar{f}_{\theta^{\text{fin}}} \approx f$ is a safe choice that guarantees a good interpolation accuracy, even if not necessarily the *optimal one* (see Table 1).

Concerning the second item above, it is worth remarking that the function $\bar{f} := \bar{f}_{\theta^{\text{fin}}}$ of the δNN-VSK method is such that $\bar{f} \approx \gamma f$, where $\gamma : \mathbb{R}^d \longrightarrow \mathbb{R}$ is a continuous function. This is coherent with our implementation of the δNN and its loss (10). Let us write the VSKs with their explicit dependence on the shape parameter and on the radial distance, i.e., for $\boldsymbol{x}, \boldsymbol{z} \in \Omega$:

$$\kappa_{\gamma f}(\boldsymbol{x}, \boldsymbol{z}) = \phi\left(\varepsilon\sqrt{\sum_{k=1}^{d}(x^{(k)} - z^{(k)})^2 + ((\gamma f)(\boldsymbol{x}) - (\gamma f)(\boldsymbol{z}))^2}\right) = \phi\left(\sqrt{\sum_{k=1}^{d}\varepsilon^2(x^{(k)} - z^{(k)})^2 + \varepsilon^2(\gamma(\boldsymbol{x})f(\boldsymbol{x}) - \gamma(\boldsymbol{z})f(\boldsymbol{z}))^2}\right).$$

Then, the fact that the function $\bar{f}$ mimics the shape of the function f but sometimes possibly out of scale is a consequence of the fact that the δNN itself tries to *adjust*, via the function γ, the fixed shape parameter ε. In principle, we could optimize the shape parameter too, indeed methods for optimizing ε could be integrated with our scheme [2,4,22,23]; nevertheless, this is beyond the scope of the current paper as here the focus is on learning $\bar{f}$; such feature can be addressed in future work. However, the most relevant result of these experiments is that we always observe some sort of similarities between the targets and the recovered scaling functions (coherently with Proposition 3.1) and that the proposed VSK-based methods work properly for automatically identifying nearly-optimal/safe scaling functions.

Concluding the analyses of the interpolation results for the three test functions f_1, f_2, f_3, we measure the NN models training time, for both the δNN-VSK method and the VSK-f method. All the training procedures have been performed on a Laptop LENOVO 21HFS04Q00 ThinkPad P14s Gen 4, 12-core CPU (4-mt/8-st) model 13th Gen Intel Core i7-1360P (bits: 64), 32 GB RAM, NVIDIA RTX A500 Laptop GPU. The NN model training time is shown in Table 2 where we report the training time as the average over 10 runs together with their standard deviation.

We measure the time spent for training the NN models because it is the only extra computational cost with respect to FSKs. Moreover, we recall that the δNN-VSK method is trained in a deterministic way, on the whole interpolation data; then, all the available training epochs are used, because no other stopping criteria are considered. On the other hand, the training of the δNN as scaling function in the VSK-f methods follows the classic NN training rules (see Appendix for details); it is then a stochastic training, and early stopping regularization can activate before the maximum number of epochs is reached (see the cases f_2, $n = 1521$, and f_3, $n = 1089$ in Table 2).

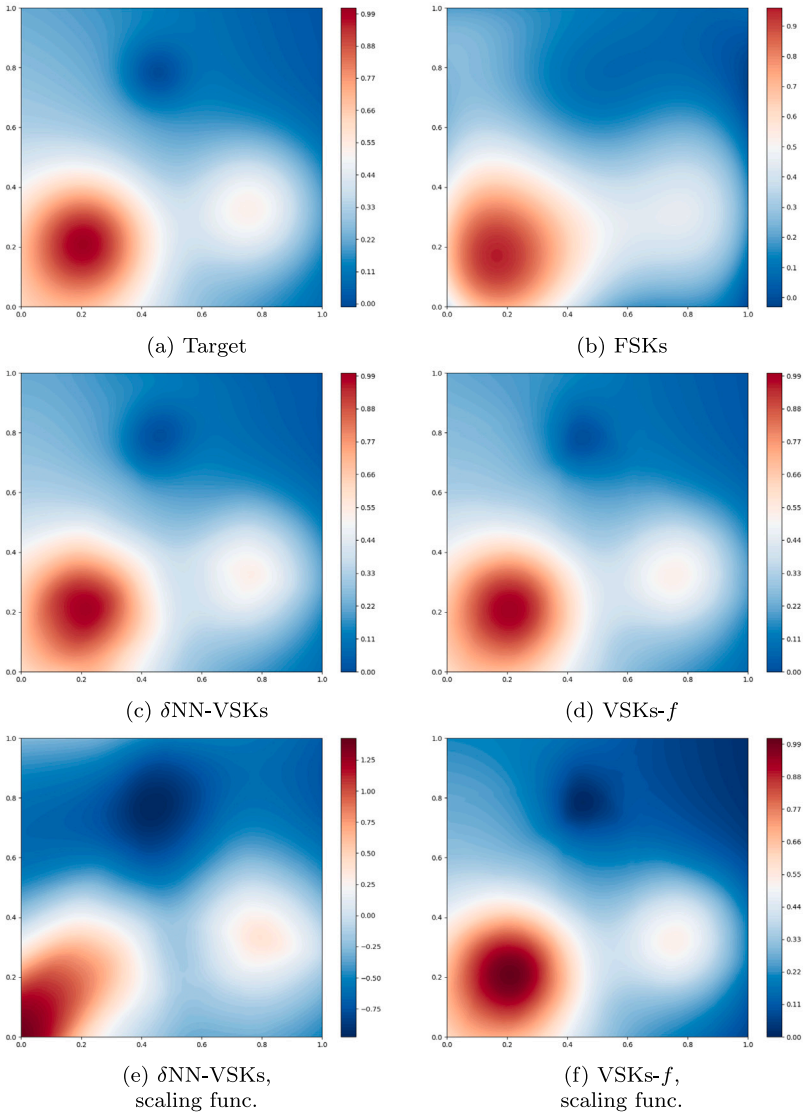


Fig. 2. Interpolation results for f_1. Case $n = 33^2 = 1089$ interpolation points, Gaussian Kernel. Shared color bars for subfigures (*a*)–(*d*); custom color bars for the scaling functions (subfigures (*e*) and (*f*)).

Table 2
Training times in minutes and seconds for the methods δNN-VSKs and VSKs-f on the test functions f_1, f_2, and f_3. Training times were computed by running the methods 10 times and computing the average and the standard deviation.

	n	Training time (min:s)		Training epochs (done/max)	
		δNN-VSKs	VSKs-f	δNN-VSKs	VSKs-f
f_1	729	01:04.28 ± 0.30 s	02:41.68 ± 0.79 s	2000/2000	1000/1000
	1089	01:36.96 ± 0.22 s	03:47.06 ± 0.98 s	2000/2000	1000/1000
	1521	02:19.92 ± 0.26 s	05:06.02 ± 0.99 s	2000/2000	1000/1000
f_2	729	01:09.13 ± 1.17 s	02:41.77 ± 0.60 s	2000/2000	1000/1000
	1089	01:44.41 ± 0.32 s	03:46.98 ± 0.48 s	2000/2000	1000/1000
	1521	02:33.06 ± 0.28 s	04:44.27 ± 0.63	2000/2000	928/1000
f_3	729	01:06.84 ± 0.12 s	02:42.33 ± 0.47 s	2000/2000	1000/1000
	1089	01:43.88 ± 0.25 s	03:34.77 ± 0.99 s	2000/2000	947/1000
	1521	02:32.31 ± 0.28 s	05:05.35 ± 0.87 s	2000/2000	1000/1000

In general, we observe that these training procedures are relatively fast: at most 2 min and a half for δNN-VSKs and at most 5 min for VSKs-f. Therefore, the best results obtained using the proposed methods, especially for discontinuous functions, are worth

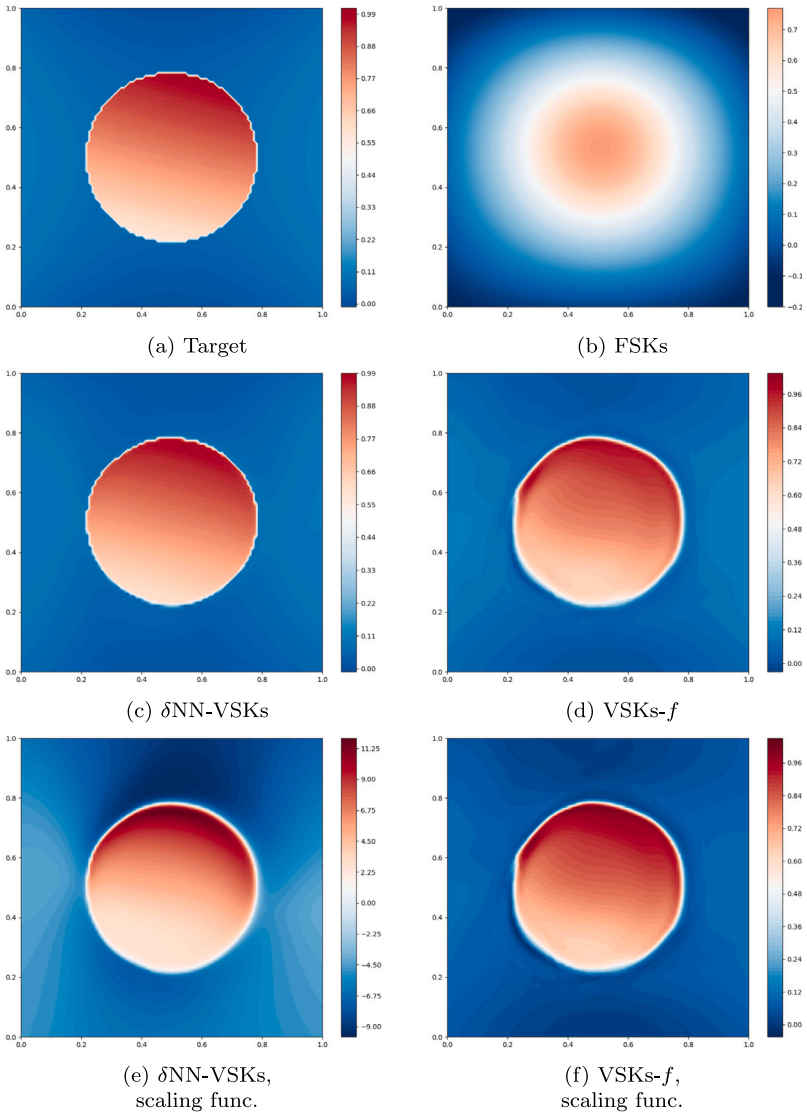


Fig. 3. Interpolation results for f_2. Case $n = 33^2 = 1089$ interpolation points, Matérn-C^2 Kernel. Shared color bars for subfigures (*a*)–(*d*); custom color bars for the scaling functions (subfigures (*e*) and (*f*)).

the extra computational cost in terms of time. Concerning the scalability with respect to n, the δNN-VSK method performs better, because it is deterministic and not based on mini-batches (mini-batches may increase the number of inner iteration for epochs in NN training procedures).

4.2. Real-world test case

In this subsection, we apply the proposed δNN-based VSK methods to approximate a discontinuous function obtained from real data. Specifically, we consider the phase transition phenomenon of acetone. For this medium (like other ones), the state of matter is characterized by some parameters, such as the density ρ. Nonetheless, from a physical point of view, the density can be described as a function of the temperature T and the pressure p; then, we can define the function f_4:

$$\rho = f_4(T, p)\,.$$

We are interested in approximating f_4 because, for a wide range of media, it has been observed via experimental measurements that the density often presents discontinuity interfaces that separate two or more regions of the (T, p) plane, representing different states of matter. The points belonging to these discontinuity interfaces are called equilibrium points for the states of matter they separate. However, phase transitions can happen also continuously, crossing a region of so-called supercritical points of the (T, p)

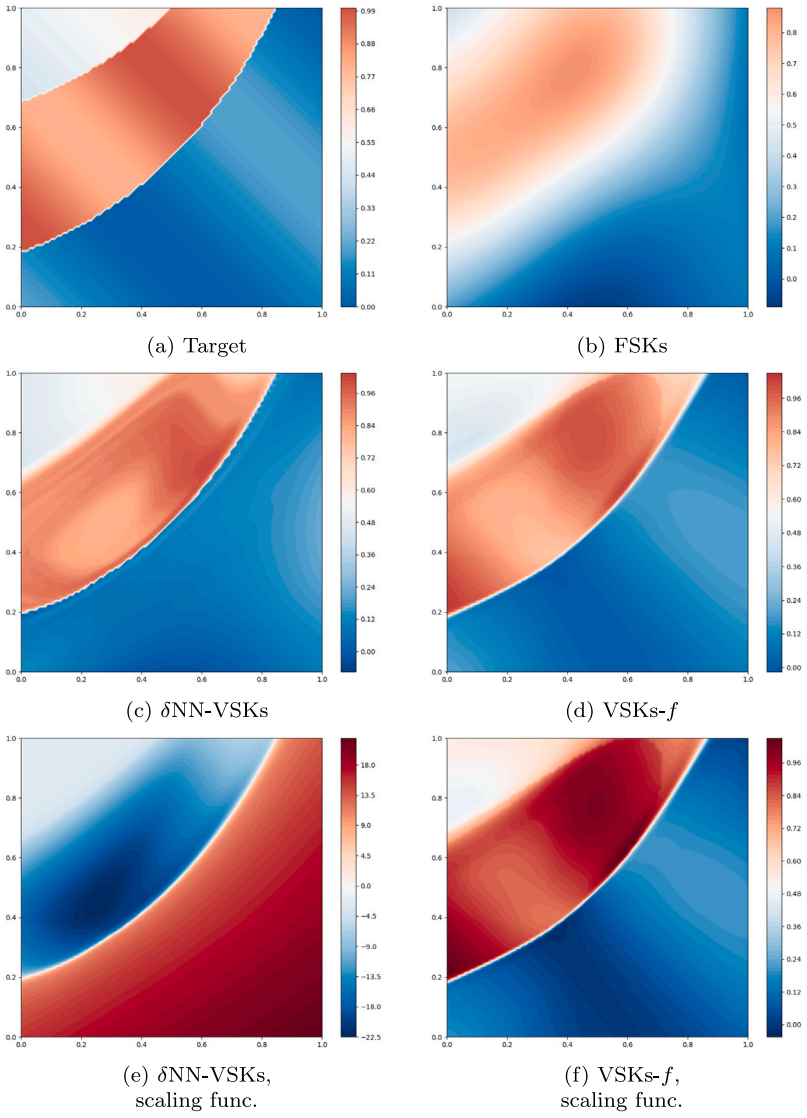


Fig. 4. Interpolation results for f_3. Case $n = 33^2 = 1089$ interpolation points, Matérn-C^2 Kernel. Shared color bars for subfigures (*a*)–(*d*); custom color bars for the scaling functions (subfigures (*e*) and (*f*)).

plane. This classification of phase transition phenomena is called Ehrenfest classification [24]. See Figs. 9-(a) and 10-(a) for a reconstruction of the density function f_4 of acetone (T and p are normalized); see Fig. 8 for an example of experimental measures and a description of the state of matter. Looking at these figures, we observe that the function is characterized by a discontinuity interface that is a sort of nonlinear "rip".

The discontinuous behavior of f_4 makes it an interesting new test function, useful for extending the analysis to a real-world scenario. Then, we approximate the acetone's density function by using FSK, δNN-VSK, and VSK-f methods, with respect to the same values n of interpolation points used in Section 4.1.

Also for this case, we observe that the δNN-VSK and VSK-f methods improves on FSKs in the approximation accuracy (see Table 3 and Figs. 9 and 10). In particular, even if the approximation errors of FSKs are not particularly bad, the VSKs-based methods show very good abilities in correctly approximating the nonlinear rip characterizing the discontinuity. This property is very important for the real-world scenario we are considering.

Concerning the scaling function learned via δNN, we observe similar behaviors to the ones observed in Section 4.1. The δNN-VSK method learns a nearly-optimal scaling functions that resemble the target in the form of $\bar{f} \approx \gamma f$, with $\gamma < 0$. On the other hand, the VSK-f method continues to support the theoretical claims of the safety in having $\bar{f} \approx f$; refer to the bottom rows of Fig. 9.

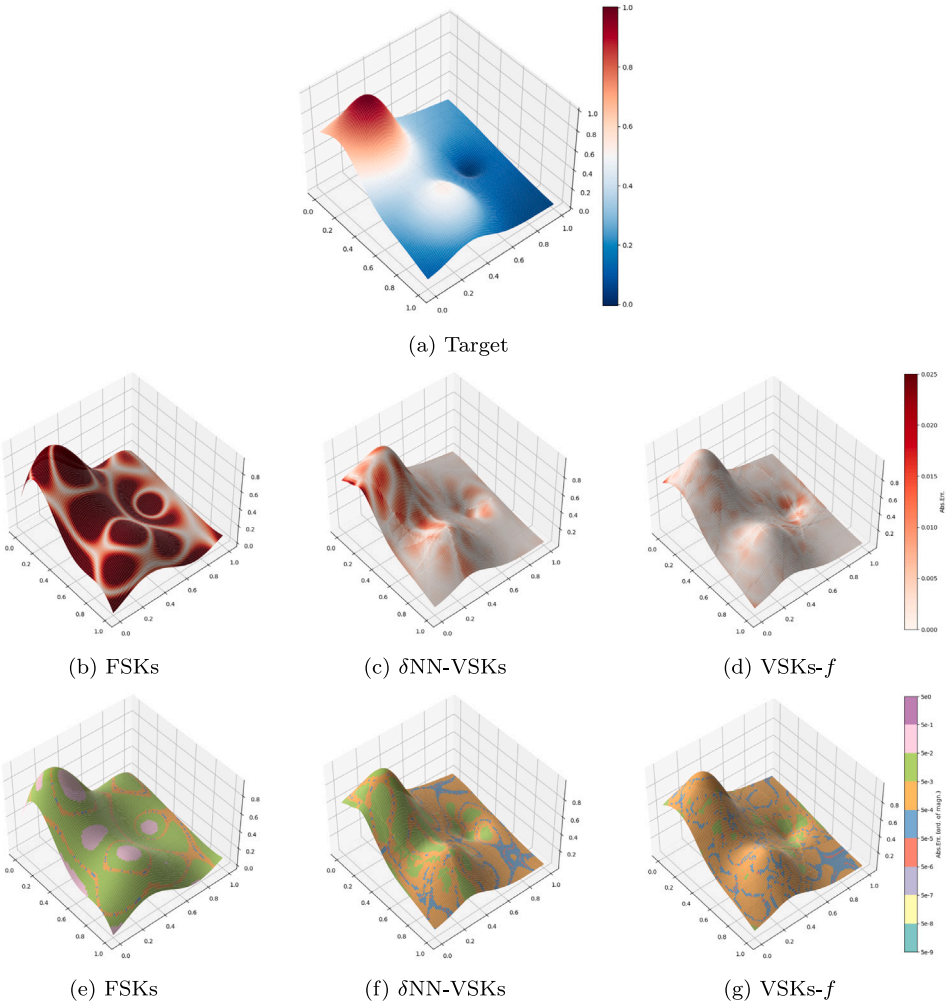


Fig. 5. Interpolation results for f_1 (see Fig. 2), colored w.r.t. interpolation error (absolute error). In the bottom row, colors describe the order of magnitude of the absolute error. Shared colorscales for subfigures (*b*)–(*d*) and for subfigures (*e*)–(*g*).

Table 3
MAEs, MSEs, and SSIMs for the test function f_4.

	Interpolation			MAE			MSE			SSIM		
	n	Kernel	ϵ	FSKs	δNN-VSKs	VSKs-f	FSKs	δNN-VSKs	FSKs	FSKs	δNN-VSKs	VSKs-f
f_4	729	Mat.C^2	0.06	7.16e–2	1.03e–2	9.23e–3	1.38e–2	3.34e–3	3.26e–3	0.9000	0.9777	0.9853
	1089	Mat.C^2	0.12	4.94e–2	7.98e–3	8.29e–3	1.05e–2	3.45e–3	3.35e–3	0.9158	0.9852	0.9902
	1521	Mat.C^2	0.48	2.54e–2	8.10e–3	7.22e–3	6.06e–3	3.43e–3	2.84e–3	0.9251	0.9856	0.9883

5. Conclusions

In this work, by taking advantage of the cardinal form of the VSK interpolant, we introduce new error bounds that relate the accuracy of the VSK interpolant to the definition of the associated scaling function. These bounds point out that the closer the scaling function is to the target, the smaller the pointwise error. We further propose a way to learn the scaling function using δNNs, and such a data-driven approach offers a user-independent solution for enhancing the flexibility of VSKs in meshfree approximation tasks and numerically shows that the learned scaling function captures the key features of the target, as expected.

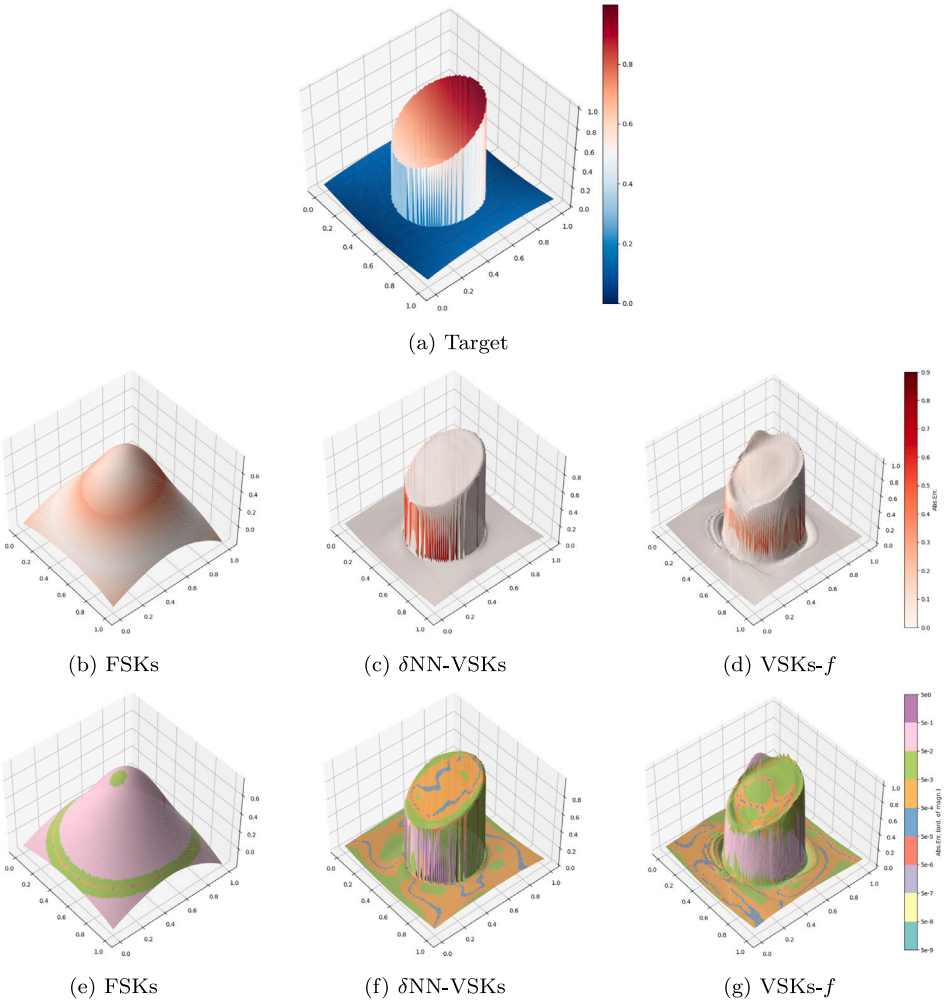


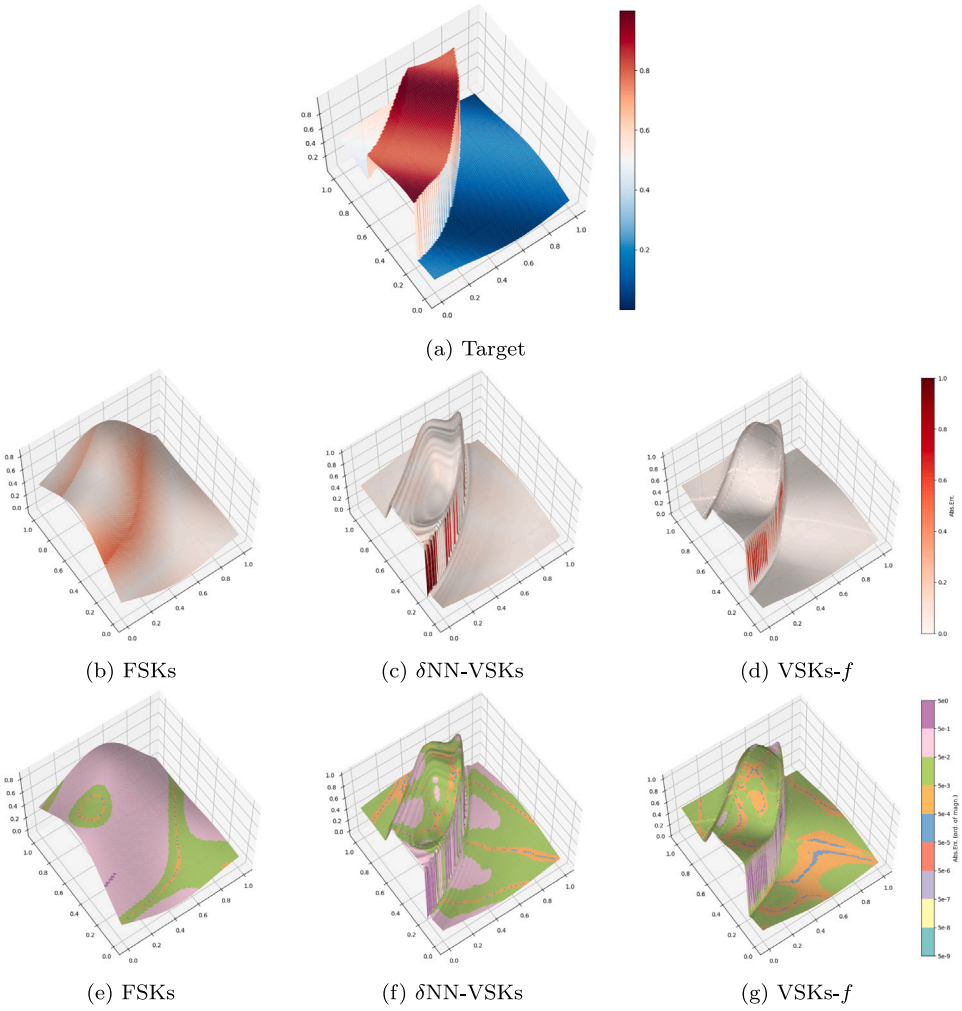
Fig. 6. Interpolation results for f_2 (see Fig. 3), colored w.r.t. interpolation error (absolute error). In the bottom row, colors describe the order of magnitude of the absolute error. Shared color bars for subfigures (*b*)–(*d*) and for subfigures (*e*)–(*g*).

The numerical tests show that our two possible implementations of the VSKs, namely the δNN-VSKs and the VSKs-f, behave better than the classical kernel interpolant. The main advantage of the VSKs-f lies in its easy implementation because it directly learns the target which is then used as a scaling function. Nevertheless, the justification for using such a scheme is subordinated to the theoretical results and to the fact that δNN-VSKs, which are able to learn the nearly-optimal scaling function without any constraints, return scaling functions that truly resemble the targets.

In future work, we plan to learn simultaneously both the scaling function for VSKs and the *optimal* sampling locations producing in that way sparse models (see e.g. [18,26,27]). Moreover, the same data-driven learning algorithm could be extended to the context of kernel collocation models for approximating PDEs, with a particular focus on the most local promising techniques as RBF-FD (refer to [28] for a general overview).

Acknowledgments

Gianluca Audone and Emma Perracchione kindly acknowledge the support of the Fondazione Compagnia di San Paolo, Italy within the framework of the Artificial Intelligence Call for Proposals, AIxtreme project (ID Rol: 71708). Emma Perracchione further acknowledges the support of the GOSSIP project (“Greedy Optimal Sampling for Solar Inverse Problems”) – funded by the Ministero dell’Università e della Ricerca (MUR), Italy - within the PRIN 2022 program (D.D.104 - 02/02/2022, CUP: E53C24002330001). Sandra Pieraccini acknowledges the support of the FaReX project (“Full and Reduced order modelling of coupled systems: focus on



(a) Target (b) FSKs (c) δNN-VSKs (d) VSKs-f (e) FSKs (f) δNN-VSKs (g) VSKs-f

Fig. 7. Interpolation results for f_3 (see Fig. 4), colored w.r.t. interpolation error (absolute error). In the bottom row, colors describe the order of magnitude of the absolute error. Shared color bars for subfigures (b)–(d) and for subfigures (e)–(g).

non-matching methods and automatic learning") – funded by MUR, Italy – within the PRIN 2022 program (D.D.104 - 02/02/2022, CUP: E53D23005510006). Francesco Della Santa and Sandra Pieraccini acknowledge that this study was carried out within the FAIR-Future Artificial Intelligence Research and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR)–MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3—D.D. 1555 11/10/2022, PE00000013). This manuscript reflects only the authors' views and opinions; neither the European Union nor the European Commission can be considered responsible for them.

Appendix. δNN architecture in the numerical experiments

In this section, we list the details about the δNN architecture and the training options used.

Architecture:

1. Input layer ($\mathbb{R}^2$ inputs);

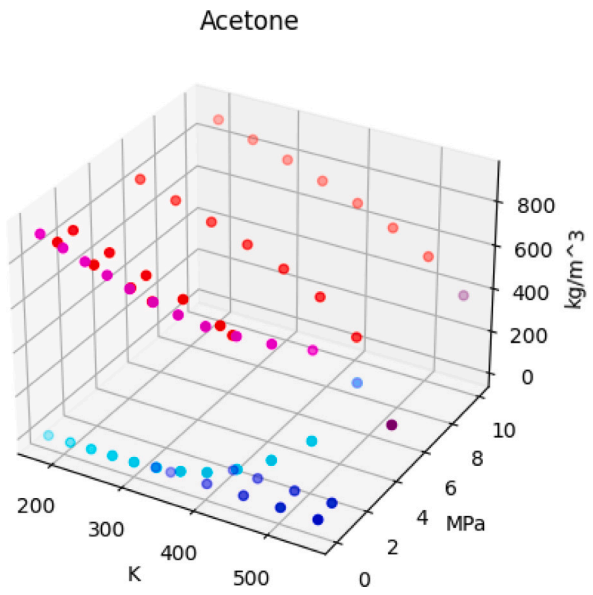


Fig. 8. (T, p, ρ) points of acetone taken from [25]; red dots correspond to liquid state, magenta to liquid state at equilibrium, blue to gas state, cyan to gas state at equilibrium, and purple to supercritical points.

2. Two FC layers, 128 units, elu activation function;
3. One residual block made of:
   (a) FC layer, 128 units, elu activation function;
   (b) FC layer, 128 units, linear activation function;
   (c) Sum of the two layers above;
   (d) elu activation function;
4. Three blocks of layers, each block made of:
   (a) FC layer, 128 units, elu activation function;
   (b) Residual block as described at item 3;
   (c) Discontinuous layer, 16 units, elu activation function;
5. FC layer, 128 units, elu activation function;
6. FC layer, 1 unit, linear activation function.

Training Options:

- δNN-VSKs:
  - maximum epochs: 2000;
  - Adam optimizer [29], starting learning rate 10^{-4}, reduce learning rate on plateau (patience 75 epochs, factor 0.5);
- VSKs-f:
  - maximum epochs: 1000;
  - Validation set: 20% of the n points;
  - Mini-batch size: 32;
  - Adam optimizer [29], starting learning rate 10^{-4}, reduce learning rate on plateau (patience 75 epochs, factor 0.5);
  - Early stopping (patience 550 epochs).

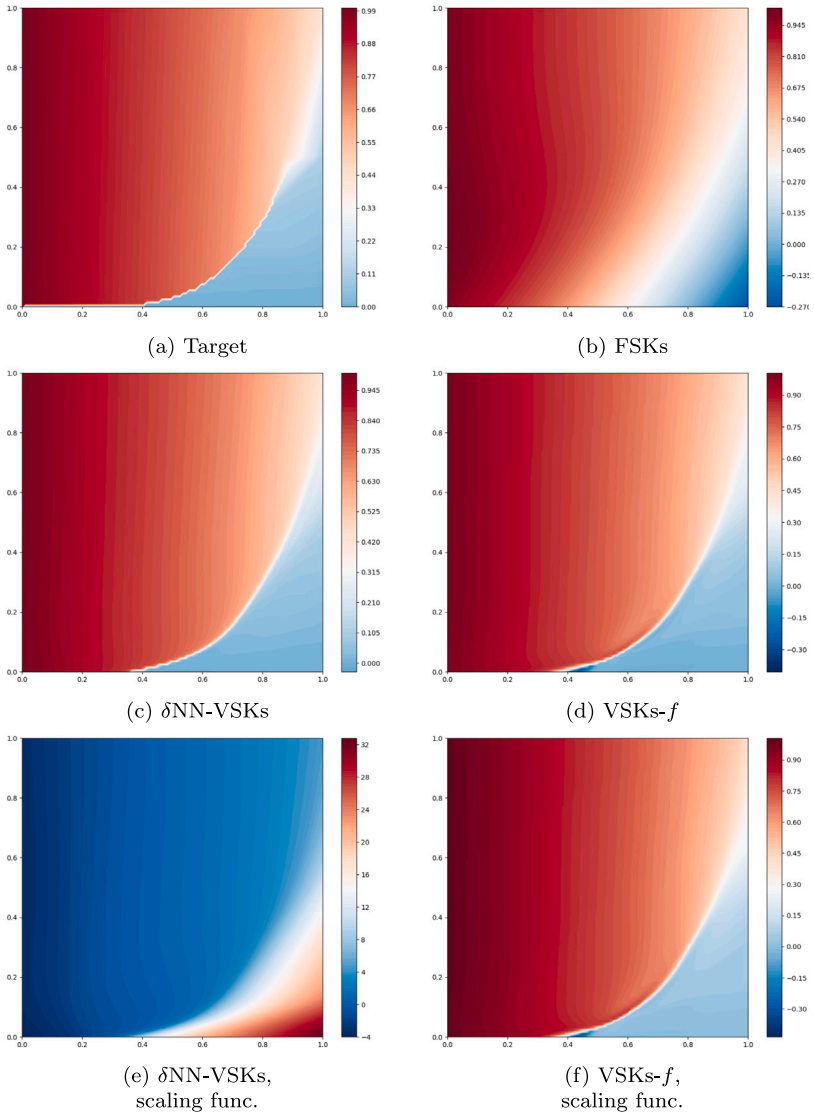
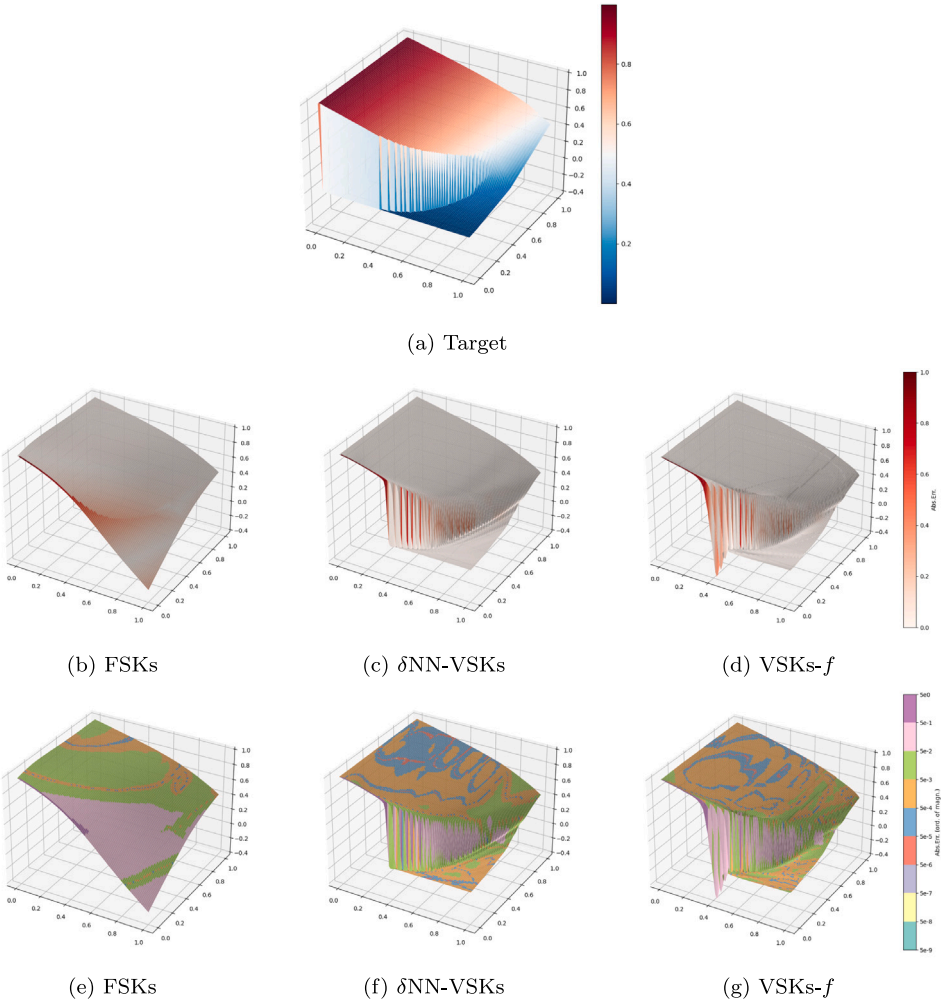


Fig. 9. Interpolation results for f_4. Case $n = 33^2 = 1089$ interpolation points, Matérn-C^2 Kernel. Shared color bars for subfigures (*a*)–(*d*); custom color bar for the scaling functions (subfigures (*e*) and (*f*)).

Data availability

Data will be made available on request.



(a) Target (b) FSKs (c) δNN-VSKs (d) VSKs-f (e) FSKs (f) δNN-VSKs (g) VSKs-f

Fig. 10. Interpolation results for f_4 (see Fig. 9), colored w.r.t. interpolation error (absolute error). In the bottom row, colors describe the order of magnitude of the absolute error. Shared colorscales for subfigures (b)–(d) and for subfigures (e)–(g).

References

[1] M. Bozzini, L. Lenarduzzi, M. Rossini, R. Schaback, Interpolation with variably scaled kernels, IMA J. Numer. Anal. 35 (1) (2015) 199–219.
[2] T.A. Driscoll, B. Fornberg, Interpolation in the limit of increasingly flat radial basis functions, Comput. Math. Appl. 43 (3–5) (2002) 413–422.
[3] B. Fornberg, G. Wright, Stable computation of multiquadric interpolants for all values of the shape parameter, Comput. Math. Appl. 48 (5–6) (2004) 853–867.
[4] L. Ling, F. Marchetti, A stochastic extended Rippa's algorithm for LpOCV, Appl. Math. Lett. 129 (2022) 107955.
[5] F. Marchetti, The extension of Rippa's algorithm beyond LOOCV, Appl. Math. Lett. 120 (2021) 107262.
[6] S. Rippa, An algorithm for selecting a good value for the parameter c in radial basis function interpolation, Adv. Comput. Math. 11 (2) (1999) 193–210.
[7] S. De Marchi, W. Erb, F. Marchetti, E. Perracchione, M. Rossini, Shape-driven interpolation with discontinuous kernels: Error analysis, edge extraction, and applications in magnetic particle imaging, SIAM J. Sci. Comput. 42 (2) (2020) B472–B491.
[8] L. Romani, M. Rossini, D. Schenone, Edge detection methods based on RBF interpolation, J. Comput. Appl. Math. 349 (2019) 532–547.
[9] M. Rossini, Interpolating functions with gradient discontinuities via variably scaled kernels, Dolomites Res. Notes Approx. 17 (2018) 3–14.
[10] E. Perracchione, F. Camattari, A. Volpara, P. Massa, A.M. Massone, M. Piana, Unbiased CLEAN for STIX in solar orbiter, Astrophys. J. Suppl. S. 268 (2) (2023).
[11] F. Della Santa, S. Pieraccini, Discontinuous neural networks and discontinuity learning, J. Comput. Appl. Math. 419 (2023).
[12] G.E. Fasshauer, M. McCourt, Kernel-Based Approximation Methods using MATLAB, World scientific, 2015.
[13] H. Wendland, Scattered Data Approximation, Cambridge University Press, 2004.
[14] M.K. Esfahani, S. De Marchi, F. Marchetti, Moving least squares approximation using variably scaled discontinuous weight function, Constr. Math. Anal. 6 (1) (2023) 38–54.
[15] L. Brutman, On the lebesgue function for polynomial interpolation, SIAM J. Num. Anal. 15 (4) (1978) 694–704.
[16] P. Kidger, T. Lyons, Universal approximation with deep narrow networks, 2020, 125, 2306–2327.
[17] R. Cavoretto, A. De Rossi, E. Perracchione, Efficient computation of partition of unity interpolants through a block-based searching technique, Comput. Math. Appl. 71 (12) (2016) 2568–2584.

[18] T. Wenzel, G. Santin, B. Haasdonk, Analysis of target data-dependent greedy kernel algorithms: Convergence rates for f-, $f \cdot p$- and f/p-greedy, Constr. Approx. 57 (1) (2023) 305–327.
[19] Z. Wang, A. Bovik, H. Sheikh, E. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process. 13 (4) (2004) 600–612, http://dx.doi.org/10.1109/TIP.2003.819861.
[20] F. Marchetti, G. Santin, Convergence results in image interpolation with the continuous ssim, SIAM J. Imag. Sci. 15 (4) (2022) 1977–1999.
[21] Tf.image.ssim, tensorflow, 2024, https://www.tensorflow.org/api_docs/python/tf/image/ssim. (Accessed 20 September 2024).
[22] R. Cavoretto, A. De Rossi, M. Mukhametzhanov, Y. Sergeyev, On the search of the shape parameter in radial basis functions using univariate global optimization methods, J. Global Optim. 39 (9) (2021) 305–327.
[23] T. Wenzel, F. Marchetti, E. Perracchione, Data-driven kernel designs for optimized greedy schemes: A machine learning perspective, SIAM J. Sci. Comput. 46 (1) (2024) C101–C126.
[24] G. Jaeger, The Ehrenfest Classification of Phase Transitions: Introduction and Evolution, Arch. Hist. Exact Sci. 53 (1998) 51–81, http://dx.doi.org/10.1007/s004070050021.
[25] Acetone - density and specific weight, 2022, https://www.engineeringtoolbox.com/acetone-2-propanone-density-specific-weight-temperature-pressure-d_2038.html. (Accessed 11 March 2022).
[26] F. Camattari, S. Guastavino, F. Marchetti, M. Piana, E. Perracchione, Classifier-dependent feature selection via greedy methods, Statist. Comput. 34 (151) (2024) 1–12.
[27] S. De Marchi, R. Schaback, H. Wendland, Near-optimal data-independent point locations for radial basis function interpolation, Adv. Comput. Math. 23 (3) (2005) 317–330.
[28] B. Fornberg, N. Flyer, Solving PDEs with radial basis functions, Acta Numer. 24 (2015) 215–258.
[29] D.P. Kingma, J.L. Ba, Adam: A method for stochastic optimization, in: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 2015, pp. 1–15, arXiv:1412.6980.