Gabriele Santin and Bernard Haasdonk

9 Kernel methods for surrogate modeling

Abstract: This chapter deals with kernel methods as a special class of techniques for surrogate modeling. Kernel methods have proven to be efficient in machine learning, pattern recognition and signal analysis due to their flexibility, excellent experimental performance and elegant functional analytic background. These data-based techniques provide so called kernel-expansions, i. e., linear combinations of kernel functions which are generated from given input–output point samples that may be arbitrarily scattered. In particular, these techniques are meshless, do not require or depend on a grid, hence are less prone to the curse of dimensionality, even for high-dimensional problems.

In contrast to projection-based model reduction, we do not necessarily assume a high-dimensional model, but a general function that models input–output behavior within some simulation context. This could be some micro-model in a multiscale simulation, some submodel in a coupled system, some initialization function for solvers, coefficient function in Partial Differential Equations (PDEs), etc.

First, kernel surrogates can be useful if the input–output function is expensive to evaluate, e. g. as a result of a finite element simulation. Here, acceleration can be obtained by sparse kernel expansions. Second, if a function is available only via measurements or a few function evaluation samples, kernel approximation techniques can provide function surrogates that allow for global evaluation.

We present some important kernel approximation techniques, which are kernel interpolation, greedy kernel approximation and support vector regression. Pseudocode is provided for ease of reproducibility. In order to illustrate the main features, commonalities and differences, we compare these techniques on a real-world application. The experiments clearly indicate the enormous acceleration potential.

Keywords: regularized kernel interpolation, support vector regression, surrogate modeling, greedy approximation, reproducing kernel Hilbert spaces

MSC 2010: 65D05, 65D15, 46C05, 68T05

Acknowledgement: The authors acknowledge the support of the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2075. The authors would like to thank Florian Rieg for the careful proofreading of this manuscript.

Gabriele Santin, Bernard Haasdonk, Institute of Applied Analysis and Numerical Simulation, University of Stuttgart, Stuttgart, Germany

https://doi.org/10.1515/9783110498967-009

9.1 Introduction

This chapter deals with kernel methods as tools to construct surrogate models of arbitrary functions, given a finite set of arbitrary samples.

These methods generate approximants based solely on input–output pairs of the unknown function, without geometrical constraints on the sample locations. In particular, the surrogates do not necessarily depend on the knowledge of an high-dimensional model but only on its observed input–output behavior at the sample sites, and they can be applied on arbitrarily scattered points in high dimension.

These features are particularly useful when these methods are applied within some simulation context. For example, kernel surrogates can be useful if the input–output function is expensive to evaluate, e. g. is a result of a finite element simulation. Here, acceleration can be obtained by sparse kernel expansions. Moreover, if a function is available only via measurements or a few function evaluation samples, kernel approximation techniques can provide function surrogates that allow global evaluation.

Kernel methods are used with much success in Model Order Reduction, and far beyond the scope of this chapter. For example, they have been used in the modeling of geometry transformations and mesh coupling [3, 12, 13], and in mesh repair methods [33], or in the approximation of stability factors and error indicators [14, 32, 34], where only a few samples of the exact indicators are sufficient to construct an efficient surrogate to be used in the online phase. Moreover, kernel methods have been combined with projection-based MOR methods, e. g. to obtain simulation-based classification [60], or to derive multi-fidelity Monte Carlo approximations [40]. Kernel surrogates have been employed in optimal control problems [51, 59], in the coupling of multi-scale simulations in biomechanics [25, 69], in real time prediction for parameter identification and state estimation in biomechanical systems [29], in gas transport problems [22], in the reconstruction of potential energy surfaces [30], in the forecasting of time stepping methods [6], in the reduction of nonlinear dynamical systems [67], in uncertainty quantification [28], and for nonlinear balanced truncation of dynamical systems [5].

In further generality, there exist many kernel-based algorithms and application fields that we do not address here. Mainly, we address the solution of PDEs, in which several approaches have emerged in the last years, and which particularly allow one to solve problems with unstructured grids on general geometries, including high dimensional manifolds (see e. g. [11, 17]). Moreover, several other techniques are studied within Machine Learning, such as classification, density estimation, novelty detection or feature extraction (see e. g. [53, 54]).

Furthermore, we remark that these methods are members of the larger class of machine learning and approximation techniques, which are generally suitable to construct models based on samples to make prediction on new inputs. These models are

usually referred to as surrogates when they are then used as replacements of the model that generated the data, as they are able to provide an accurate and faster response. Some examples of these techniques are classical approximation methods such as polynomial interpolation, which are used in this context especially in combination with sparse grids to deal with high-dimensional problems (see [19]), and (deep) neural network models. The latter in particular have seen a huge increase in analysis and application in the recent years. For a recent treatment of deep learning, we refer e. g. to [21].

Despite these very diverse applications and methodologies, kernel methods can be analyzed to some extent in the common framework of Reproducing Kernel Hilbert spaces and, although the focus of this chapter will be on the construction of sparse surrogate models, parts of the following discussion can be the starting point for the analysis of other techniques.

In general terms, kernel methods can be viewed as nonlinear versions of linear algorithms. As an example, assume to have some set $X_n := \{x_k\}_{k=1}^n \subset \mathbb{R}^d$ of data points and target data values $Y_n := \{y_k\}_{k=1}^n \subset \mathbb{R}$. We can construct a surrogate $s : \mathbb{R}^d \to \mathbb{R}$ that predicts new data via linear regression, i. e., find $w \in \mathbb{R}^d$ s. t. $s(x) := \langle w, x\rangle$, where $\langle\cdot,\cdot\rangle$ is the scalar product in $\mathbb{R}^d$. A good surrogate model s will give predictions such that $|s(x_k) - y_k|$ is small. If we can write $w \in \mathbb{R}^d$ as $w = \sum_{j=1}^n \alpha_j x_j$ for a set of coefficients $(\alpha_i)_{i=1}^n \in \mathbb{R}^n$, then s can be rewritten as

$$s(x) := \sum_{j=1}^{n} \alpha_j \langle x_j, x\rangle .$$

Note that this formulation includes also regression with an offset (or bias) $b \neq 0$, which can be written in this form by an extended representation as

$$s(x) := \langle w, x\rangle + b =: \langle \bar{w}, \bar{x}\rangle ,$$

where $\bar{x} := (x, 1)^T \in \mathbb{R}^{d+1}$ and $\bar{w} := (w, b)^T \in \mathbb{R}^{d+1}$.

Using now the Gramian matrix $A \in \mathbb{R}^{n\times n}$ with entries $A_{ij} := \langle x_i, x_j\rangle$ and rows $A_i^T \in \mathbb{R}^n$, we look for the surrogate s which minimizes

$$\sum_{i=1}^{n} (s(x_i) - y_i)_2^2 = \sum_{i=1}^{n} (A_i^T \alpha - y_i)_2^2 = \|A\alpha - y\|_2^2.$$

Additionally, a regularization term can be added to keep the norm of α small, e. g. in terms of the value $\alpha^T A\alpha$. Thus, the surrogate can be characterized as the solution of the optimization problem

$$\min_{\alpha\in\mathbb{R}^n} \|A\alpha - y\|_2^2 + \lambda \alpha^T A\alpha,$$

i. e., $\alpha = (A + \lambda I)^{-1} y$ if $\lambda > 0$.

In many cases this (regularized) linear regression is not sufficient to obtain a good surrogate. A possible idea is to try to combine this linear, simple method with a nonlinear function which maps the data to a higher dimensional space, where the hope is that the image of the data can be processed linearly. For this we consider a so-called feature map $\Phi : \mathbb{R}^d \to H$, where H is a Hilbert space, and apply the same algorithm to the transformed data $\Phi(X_n) := \{\Phi(x_i)\}_{i=1}^n$ with the same values Y_n. Since the algorithm depends on X_n only via the Gramian A, it is sufficient to replace it with the new Gramian $A_{ij} := \langle \Phi(x_i), \Phi(x_j) \rangle_H$ to obtain a nonlinear algorithm.

We will see that $\langle \Phi(x), \Phi(y) \rangle_H$ defines in fact a positive definite kernel, and if any numerical procedure can be written in terms of inner products of the inputs, it can be transformed in the same way into a new nonlinear algorithm simply by replacing the inner products with kernel evaluations (the so-called kernel trick). We will discuss the details of this procedure in the next sections in the case of interpolation and Support Vector Regression, but this immediately gives a glance of the ample spectrum of algorithms in the class of kernel methods.

This chapter is organized as follows. Section 9.2 covers the basic notions on kernels and kernel-based spaces which are necessary for the development and understanding of the algorithms. The next Section 9.3 presents the general ideas and tools to construct kernel surrogates as characterized by the Representer Theorem, and these ideas are specialized to the case of kernel interpolation in Section 9.4 and Support Vector Regression in Section 9.5. In both cases, we provide the theoretical foundations as well as the algorithmic description of the methods, with particular attention to techniques to enforce sparsity in the model. These surrogates can be used to perform various analyses of the full model, and we give some examples in Section 9.6. Section 9.7 presents a general strategy to choose the various parameters defining the model, whose tuning can be critical for a successful application of the algorithms. Finally, we discuss in Section 9.8 the numerical results of the methods on a real application dataset, comparing training time (offline), prediction time (online), and accuracy.

9.2 Background on kernels

We start by introducing some general facts of positive definite kernels. Further details on the general analytical theory of reproducing kernels can be found e. g. in the recent monograph [45], while the books [15, 65] and [53, 55] contain a treatment of kernel theory from the point of view of pattern analysis and scattered data interpolation, respectively.

9.2.1 Positive definite kernels

Given a nonempty set Ω, which can be a subset of $\mathbb{R}^d$, $d \in \mathbb{N}$, but also a set of structured objects such as strings or graphs, a real- and scalar-valued kernel K on Ω is a bivariate

symmetric function $K : \Omega \times \Omega \to \mathbb{R}$, i. e., $K(x,y) = K(y,x)$ for all $x, y \in \Omega$. For our purposes, we are interested in (strictly) positive definite kernels, defined as follows.

Definition 9.1 (Positive definite kernels). Let Ω be a nonempty set. A kernel K on Ω is positive definite (PD) on Ω if for all $n \in \mathbb{N}$ and for any set of n pairwise distinct elements $X_n := \{x_i\}_{i=1}^n \subset \Omega$, the kernel matrix (or Gramian matrix) $A := A_{K,X_n} \in \mathbb{R}^{n\times n}$ defined as $A_{ij} := K(x_i, x_j)$, $1 \le i, j \le n$, is positive semidefinite, i. e., for all vectors $\alpha := (\alpha_i)_{i=1}^n \in \mathbb{R}^n$ we have

$$\alpha^T A \alpha = \sum_{i,j=1}^{n} \alpha_i \alpha_j K(x_i, x_j) \ge 0. \tag{9.1}$$

The kernel is strictly positive definite (SPD) if the kernel matrix is positive definite, i. e., (9.1) holds with strict inequality when $\alpha \neq 0$.

The further class of conditionally (strictly) positive definite kernels is also of interest in certain contexts. We refer to [65, Chapter 8] for their extensive treatment, and we just mention that they are defined as above, except that the condition (9.1) has to be satisfied only for the subset of coefficients α which match a certain orthogonality condition. When this condition is defined with respect to a space of polynomials of degree $m \in \mathbb{N}$, the resulting kernels are used e. g. to guarantee a certain polynomial exactness of the given approximation scheme, and they are often employed in certain methods for the solution of PDEs.

9.2.2 Examples and construction of kernels

Despite the abstract definition, there are several ways to construct functions $K : \Omega \times \Omega \to \mathbb{R}$ which are (strictly) positive definite kernels, and usually the proper choice of the kernel is a crucial step in the successful application of the method. We list here a general strategy to construct kernels, and some notable examples.

An often used, constructive approach to designing a new kernel is via feature maps as follows.

Proposition 9.1 (Kernels via feature maps). *Let Ω be a nonempty set. A feature map Φ is any function $\Phi : \Omega \to H$, where $(H, \langle\cdot,\cdot\rangle_H)$ is any Hilbert space (the feature space). The function*

$$K(x,y) := \big\langle \Phi(x), \Phi(y) \big\rangle_H \quad x, y \in \Omega,$$

is a PD kernel on Ω.

Proof. K is a PD kernel since it is symmetric and positive definite, because the inner product is bilinear, symmetric and positive definite. □

In many cases, H is either $\mathbb{R}^m$ with very large m or even an infinite dimensional Hilbert space. The computation of the possibly expensive m- or infinite-dimensional inner product can be avoided if a closed form for K can be obtained. This implies a significant reduction of the computational time required to evaluate the kernel and thus to execute any kind of algorithm.

We see now some examples.

Example 9.1 (Expansion kernels). The construction comprises finite dimensional linear combinations, i. e., for a set of functions $\{v_j\}_{j=1}^m : \Omega \to \mathbb{R}$, the function $K(x,y) := \sum_{k=1}^m v_k(x)v_k(y)$ is a positive definite kernel, having a feature map

$$\Phi(x) := \big(v_1(x), v_2(x), \dots, v_m(x)\big)^T \in H := \mathbb{R}^m. \tag{9.2}$$

This idea can be extended to an infinite number of functions provided $\{v_j(x)\}_{j=1}^\infty \in H := \ell_2(\mathbb{N})$ uniformly in Ω, and the resulting kernels are called Hilbert–Schmidt or expansion kernels, which can be proven to be even SPD under additional conditions (see [49]). As an example in $d = 1$, we mention the Brownian Bridge kernel $K(x,y) := \max(x,y) - xy$, defined with a feature map $v_j(x) := \sqrt{2}(j\pi)^{-1}\sin(j\pi x)$ for $j \in \mathbb{N}$, which is SPD on $\Omega := (0,1)$. We remark that the kernel can be extended to $(0,1)^d$ with $d > 1$ using a tensor product of one-dimensional kernels.

This feature map representation proves also that $\dim(H) =: m < \infty$ means that the kernel is not SPD in general: e. g., if X_n contains n pairwise distinct points and $m < n$, then the vectors $\{\Phi(x_i)\}_{i=1}^n$ cannot be linearly independent, and thus the kernel matrix is singular.

Example 9.2 (Kernels for structured data). Feature maps are also employed to construct positive definite kernels on sets Ω of structured data, such as sets of strings, graphs, or any other object. For example, the convolution kernels introduced in [20, 26] consider a finite set of features $v_1(x), \dots, v_m(x) \in \mathbb{R}$ of an object $x \in \Omega$, and define a feature map exactly as in (9.2).

Example 9.3 (Polynomial kernels). For $a \geq 0$, $p \in \mathbb{N}$, $x, y \in \mathbb{R}^d$, the polynomial kernel

$$K(x,y) := \big(\langle x, y\rangle + a\big)^p = \left(\sum_{i=1}^d x^{(i)}y^{(i)} + a\right)^p, \quad x := \big(x^{(1)}, \dots, x^{(d)}\big)^T, \tag{9.3}$$

is PD on any $\Omega \subset \mathbb{R}^d$. It is a d-variate polynomial of degree p, which contains the monomial terms of degrees $j := (j^{(1)}, \dots, j^{(d)}) \in J$, for a certain set $J \subset \mathbb{N}_0^d$. If $m := |J|$, a feature space is $\mathbb{R}^m$ with feature map

$$\Phi(x) := (\sqrt{a_1}x^{j_1}, \dots, \sqrt{a_m}x^{j_m})^T,$$

for some positive numbers $\{a_j\}_{j=1}^m$ and monomials $x^{j_m} := \prod_{i=1}^d (x^{(i)})^{j_m^{(i)}}$.

Observe that using the closed form (9.3) of the kernel instead of the feature map is very convenient, since we work with d-dimensional instead of m-dimensional vectors, where possibly $m := |J| = \binom{d+p}{d} = \dim(\mathbb{P}_p(\mathbb{R}^d)) \gg d$.

Example 9.4 (RBF kernels). For $\Omega \subset \mathbb{R}^d$ in many applications the most used kernels are translational invariant kernels, i. e., there exists a function $\phi : \mathbb{R}^d \to \mathbb{R}$ with

$$K(x, y) := \phi(x - y), x, y \in \Omega,$$

and in particular radial kernels, i. e., there exists a univariate function $\phi : \mathbb{R}_{\geq 0} \to \mathbb{R}$ with

$$K(x, y) := \phi(\|x - y\|), x, y \in \Omega.$$

A radial kernel, or Radial Basis Function (RBF), is usually defined up to a shape parameter $\gamma > 0$ that controls the scale of the kernel via $K(x, y) := \phi(\gamma\|x - y\|)$.

The main example of such kernels is the Gaussian $K(x, y) := e^{-\gamma^2\|x-y\|^2}$, which is in fact strictly positive definite. An explicit feature map has been computed in [56]: If $\Omega \subset \mathbb{R}^d$ is nonempty, a feature map is the function $\Phi_\gamma : \Omega \to L_2(\mathbb{R}^d)$ defined by

$$\Phi_\gamma(x) := \frac{(2\gamma)^{\frac{d}{2}}}{\pi^{\frac{d}{4}}} \exp(-2\gamma^2\|x - \cdot\|^2), \quad x \in \Omega.$$

In this case it is even more evident how working with the closed form of K is much more efficient than working with a feature map and computing L_2-inner products.

RBF kernels offer a significant easiness of implementation in arbitrary space dimension d. The evaluation of the kernel $K(\cdot, x)$, $x \in \mathbb{R}^d$, on a vector of n points can indeed by realized by first computing a distance vector $D \in \mathbb{R}^n$, $D_i := \|x - x_i\|$, and then applying the univariate function ϕ on D. A discussion and comparison of different algorithms (in Matlab) to efficiently compute a distance matrix can be found in [15, Chapter 4], and most scientific computing languages comprise a built-in implementation (such as `pdist2`[1] in Matlab and `distance_matrix`[2] in Scipy).

Translational invariant and RBF kernels can be often analyzed in terms of their Fourier transforms, which provide proofs of their strict positive definiteness via the Bochner theorem (see e. g. [65, Chapter 6]), and connections to certain Sobolev spaces, as we will briefly see in Section 9.2.3.

Among various RBF kernels, there are also compactly supported kernels, i. e., $K(x, y) = 0$ if $\|x - y\| > 1/\gamma$, which produce sparse kernel matrices if γ is large enough. The most used ones are the Wendland kernels introduced in [63], which are even radial polynomial within their support.

1 https://www.mathworks.com/help/stats/pdist2.html

2 https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance_matrix.html

There are, in addition, various operations to combine positive definite kernels and obtain new ones. For example, sums and products of positive definite kernels and multiplication by a positive constant $a > 0$ produce again positive definite kernels. Moreover, if K' is a positive definite kernel and K'' is symmetric with $K' \preccurlyeq K''$ (i. e., $K := K'' - K'$ is PD) then also K'' is positive definite. Furthermore, if $\Omega = \Omega' \times \Omega''$ and K', K'' are PD kernels on Ω', Ω'', then $K(x,y) := K'(x',y')K''(x'',y'')$ and $K(x,y) := K'(x',y') + K''(x'',y'')$ are also PD kernels on Ω, i. e., kernels can be defined to respect tensor product structures of the input.

Further details and examples can be found in [45, Chapters 1–2].

9.2.3 Kernels and Hilbert spaces

Most of the analysis of kernel-based methods is possible through the connection with certain Hilbert spaces. We first give the following definition.

Definition 9.2 (Reproducing Kernel Hilbert Space). Let Ω be a nonempty set, $\mathcal{H}$ an Hilbert space of functions $f : \Omega \to \mathbb{R}$ with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. Then $\mathcal{H}$ is called a Reproducing Kernel Hilbert Space (RKHS) on Ω if there exists a function $K : \Omega \times \Omega \to \mathbb{R}$ (the reproducing kernel) such that

1. $K(\cdot, x) \in \mathcal{H}$ for all $x \in \Omega$,
2. $\langle f, K(\cdot, x) \rangle_{\mathcal{H}} = f(x)$ for all $x \in \Omega, f \in \mathcal{H}$ (reproducing property).

The reproducing property is equivalent to state that, for $x \in \Omega$, the x-translate $K(\cdot, x)$ of the kernel is the Riesz representer of the evaluation functional $\delta_x : \mathcal{H} \to \mathbb{R}$, $\delta_x(f) := f(x)$ for $f \in \mathcal{H}$, which is hence a continuous functional in $\mathcal{H}$. Also the converse holds, and the following result gives an abstract criterion to check if a Hilbert space is a RKHS.

Theorem 9.1. *An Hilbert space of functions $\Omega \to \mathbb{R}$ is a RKHS if and only if the point evaluation functionals are continuous in $\mathcal{H}$ for all $x \in \Omega$, i. e., $\delta_x \in \mathcal{H}'$, the dual space of $\mathcal{H}$. Moreover, the reproducing kernel K of $\mathcal{H}$ is strictly positive definite if and only if the functionals $\{\delta_x : x \in \Omega\}$ are linearly independent in $\mathcal{H}'$.*

Proof. The first part is clear from the reproducing property, while strict positive definiteness can be checked by verifying that the quadratic form in Definition 9.1 cannot be zero for $\alpha \neq 0$ if $\{\delta_x : x \in \Omega\}$ are linearly independent. □

We see two concrete examples.

Example 9.5 (Finite dimensional spaces). Any finite dimensional Hilbert space $\mathcal{H}$ of functions on a nonempty set Ω is a RKHS. If $m := \dim(\mathcal{H})$ and $\{v_j\}_{j=1}^m$ is an orthonormal

basis, then a reproducing kernel is given by

$$K(x,y) := \sum_{j=1}^{m} v_j(x)v_j(y), \quad x,y \in \Omega.$$

Indeed, the two properties of Definition 9.2 can be easily verified by direct computation.

Example 9.6 (The Sobolev space $H_0^1(0,1)$). The Sobolev space $H_0^1(0,1)$ with inner product $\langle f,g\rangle_{H_0^1} := \int_0^1 f'(y)g'(y)\,dy$ is a RKHS with the Brownian Bridge kernel

$$K(x,y) := \min(x,y) - xy, \quad x,y \in (0,1)$$

as reproducing kernel (see e. g. [8]). Indeed, $K(\cdot,x) \in H_0^1(0,1)$, and the reproducing property (2) follows by explicitly computing the inner product.

The following result proves that reproducing kernels are in fact positive definite kernels in the sense of Definition 9.1. Moreover, the first two properties are useful to deal with the various type of approximants of Section 9.4 and Section 9.5, which will be exactly of this form.

Proposition 9.2. *Let $\mathcal{H}$ be a RKHS on Ω with reproducing kernel K. Let $n,n' \in \mathbb{N}$, $\alpha \in \mathbb{R}^n$, $\alpha' \in \mathbb{R}^{n'}$, $X_n, X'_{n'} \subset \Omega$, and define the functions*

$$f(x) := \sum_{i=1}^{n} \alpha_i K(x,x_i), \quad g(x) := \sum_{j=1}^{n'} \alpha'_j K(x,x'_j), \quad x \in \Omega.$$

Then we have the following:

1. $f,g \in \mathcal{H}$,
2. $\langle f,g\rangle_{\mathcal{H}} = \sum_{i=1}^{n}\sum_{j=1}^{n'} \alpha_i\alpha'_j K(x_i,x'_j)$.
3. *K is the unique reproducing kernel of $\mathcal{H}$ and it is a positive definite kernel.*

Proof. The first two properties follow from Definition 9.2, and in particular from $\mathcal{H}$ being a linear space and from the bilinearity of $\langle\cdot,\cdot\rangle_{\mathcal{H}}$.

For Property (3), the fact that K is symmetric and positive definite, hence a PD kernel, follows from Property (1) of Definition 9.2, and from the symmetry and positive definiteness of the inner product. Moreover, the reproducing property implies that, if K,K' are two reproducing kernels of $\mathcal{H}$, then for all $x,y \in \Omega$ we have

$$K(x,y) = \big\langle K(\cdot,y), K'(\cdot,x)\big\rangle_{\mathcal{H}} = K'(x,y).$$

□

It is common in applications to follow instead the opposite path, i. e., to start with a given PD kernel, and try to see if an appropriate RKHS exists. This is in fact always the case, as proven by the following fundamental theorem from [2].

Theorem 9.2 (RKHS from kernels – Moore–Aronszajn theorem). *Let Ω be a nonempty set and $K : \Omega \times \Omega \to \mathbb{R}$ a positive definite kernel. Then there exists a unique RKHS $\mathcal{H} := \mathcal{H}_K(\Omega)$ with reproducing kernel K.*

Proof. The theorem was first proven in [2], to which we refer for a detailed proof. The idea is to deduce that, by Property (1) of Proposition 3, a candidate RKHS $\mathcal{H}$ of K needs to contain the linear space

$$\mathcal{H}_0 := \text{ span } \{K(\cdot, x) : x \in \Omega\}$$

of finite linear combinations of kernel translates. Moreover, from Property (2) of Proposition 9.2, the inner product on this $\mathcal{H}_0$ needs to satisfy

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^{n} \sum_{j=1}^{n'} \alpha_i \alpha'_j K(x_i, x'_j). \tag{9.4}$$

With this observation in mind, the idea of the construction of $\mathcal{H}$ is to start by $\mathcal{H}_0$, prove that (9.4) defines indeed an inner product on $\mathcal{H}_0$, and that the completion of $\mathcal{H}_0$ w. r. t. this inner product is a RKHS having K as reproducing kernel. Uniqueness then follows from Property (3) of the same proposition. □

As it is common in the approximation literature, we will sometimes refer to this unique $\mathcal{H}$ as the native space of the kernel K on Ω.

Remark 9.1 (Kernel feature map). Among other consequences, this construction allows one to prove that any PD kernel is generated by at least one feature map. Indeed, the function $\Phi : \Omega \to \mathcal{H}$, $\Phi(x) := K(\cdot, x)$, is clearly a feature map for K with feature space $\mathcal{H}$, since the reproducing property implies that

$$\big\langle \Phi(x), \Phi(y) \big\rangle_H = \big\langle K(\cdot, x), K(\cdot, y) \big\rangle_{\mathcal{H}} = K(x, y) \quad \text{for all } x, y \in \Omega.$$

Remark 9.2. For certain translational invariant kernels it is possible to prove that the associated native space is norm equivalent to a Sobolev spaces of the appropriate smoothness, which is related to the kernels' smoothness (see [65, Chapter 10]). This is particularly interesting since the approximation properties of the different algorithms, including certain optimality that we will see in the next sections, are in fact optimal in these Sobolev spaces (with an equivalent norm).

The various operations on positive definite kernels mentioned in Section 9.2.2 have an analogous effect on the corresponding native spaces. For example, the scaling by a positive number $a > 0$ does not change the native space, but scales the inner product correspondingly, and, if $K' \preccurlyeq K''$ are positive definite kernels, then $\mathcal{H}_{K'}(\Omega) \subset \mathcal{H}_{K''}(\Omega)$. We remark that the latter property has been used for example in [71] to prove inclusion relations for the native spaces of RBF kernels with different shape parameters.

9.2.4 Kernels for vector-valued functions

So far we only dealt with scalar-valued kernels, which are suitable to treat scalar-valued functions. Nevertheless, it is clear that the interest in model reduction is typically also on vector-valued or multi-output functions, which thus require a generalization of the theory presented so far. This has been done in [35], and it is based on the following definition of matrix-valued kernels.

Definition 9.3 (Matrix-valued PD kernels). Let Ω be a nonempty set and $q \in \mathbb{N}$. A function $K : \Omega \times \Omega \to \mathbb{R}^{q\times q}$ is a matrix-valued kernel if it is symmetric, i. e., $K(x,y) = K(y,x)^T$ for all $x, y \in \Omega$. It is a PD (resp., SPD) matrix-valued kernel if the kernel matrix $A \in \mathbb{R}^{nq\times nq}$ is positive semidefinite (resp., positive definite) for all $n \in \mathbb{N}$ and for all sets $X_n \subset \Omega$ of pairwise distinct elements.

This more general class of kernels is also associated to a uniquely defined native space of vector-valued functions, where the notion of RKHS is replaced by the following.

Definition 9.4 (RKHS for matrix-valued kernels). Let Ω be a nonempty set, $q \in \mathbb{N}$, $\mathcal{H}$ an Hilbert space of functions $f : \Omega \to \mathbb{R}^q$ with inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$. Then $\mathcal{H}$ is called a vector-valued RKHS on Ω if there exists a function $K : \Omega \times \Omega \to \mathbb{R}^{q\times q}$ (the matrix-valued reproducing kernel) such that

1. $K(\cdot, x)v \in \mathcal{H}$ for all $x \in \Omega$, $v \in \mathbb{R}^q$,
2. $\langle f, K(\cdot, x)v\rangle_{\mathcal{H}} = f(x)^T v$ for all $x \in \Omega$, $v \in \mathbb{R}^q$, $f \in \mathcal{H}$ (directional reproducing property).

A particularly simple version of this construction can be realized by considering separable matrix-valued kernels (see e. g. [1]), i. e., kernels that are defined as $K(x,y) := \tilde{K}(x,y)B$, where $\tilde{K}$ is a standard scalar-valued PD kernel, and $B \in \mathbb{R}^{q\times q}$ is a positive semidefinite matrix. In the special case $Q = I$ (the $q \times q$ identity matrix), in [70] it is shown that the native space of K is the tensor product of q copies of the native space of $\tilde{K}$, i. e.,

$$\mathcal{H}_K(\Omega) = \{f : \Omega \to \mathbb{R}^q : f_j \in \mathcal{H}_{\tilde{K}}(\Omega), 1 \le j \le q\}$$

with

$$\langle f, g\rangle_{\mathcal{H}_K} = \sum_{j=1}^{q} \langle f_j, g_j\rangle_{\mathcal{H}_{\tilde{K}}}.$$

This simplification will give convenient advantages when implementing some of the methods discussed in Section 9.4.

9.3 Data based surrogates

We can now introduce in general terms the two surrogate modeling techniques that we will discuss, namely (regularized) kernel interpolation and Support Vector Regression (SVR).

For both of them, the idea is to represent the expensive map to be reduced as a function $f : \Omega \to \mathbb{R}^q$ that maps an input $x \in \Omega$ to an output $y \in \mathbb{R}^q$. Here f is assumed to be only continuous, and the set Ω can be arbitrary as long as a positive definite kernel K can be defined on it. Moreover, the function does not need to be known in any particular way except than through its evaluations on a finite set $X_n := \{x_k\}_{k=1}^n \subset \Omega$ of pairwise distinct data points, resulting in data values $Y_n := \{y_k := f(x_k)\}_{k=1}^n \subset \mathbb{R}^q$.

The goal is to construct a function $s \in \mathcal{H}$ such that $s(x)$ is a good approximation of $f(x)$ for all $x \in \Omega$ (and not only for $x \in X_n$), while being significantly faster to evaluate. The process of computing s from the data (X_n, Y_n) is often referred to as training of the surrogate s, and the set (X_n, Y_n) is thus called training dataset.

The computation of the particular surrogate is realized as the solution of an infinite dimensional optimization problem. In general terms, we define a loss function

$$L : \mathcal{H} \times \Omega^n \times \left(\mathbb{R}^q\right)^n \to \mathbb{R}_{\geq 0} \cup \{+\infty\},$$

which takes as input a candidate surrogate $g \in \mathcal{H}$ and the values $X_n \in \Omega^n$, $Y_n \in (\mathbb{R}^q)^n$, and returns a measure of the data-accuracy of g. Then the surrogate s is defined as a minimizer, if it exists, of the cost function

$$J(g) := L(g, X_n, Y_n) + \lambda \|g\|_{\mathcal{H}}^2,$$

where the second part of J is a regularization term that penalizes solutions with large norm. The tradeoff between the data-accuracy term and the regularization term is controlled by the regularization parameter $\lambda \geq 0$.

For the sake of presentation, we restrict in the remaining of this section to the case of scalar-valued functions, i. e., $q = 1$. The general case follows by using matrix valued kernels as introduced in Section 9.2.4, and the corresponding definition of orthogonal projections.

The following fundamental Representer Theorem characterizes exactly some solutions of this problem, and it proves that the surrogate will be a function

$$s \in V(X_n) := \operatorname{span}\{K(\cdot, x_i), x_i \in X_n\}$$

i. e., a finite linear combination of kernel translates on the training points. A first version of this result was proven in [27], while we refer to [52] for a more general statement.

Theorem 9.3 (Representer Theorem). *Let Ω be a nonempty set, K a PD kernel on Ω, $\lambda > 0$ a regularization parameter, and let (X_n, Y_n) be a training set. Assume that $L(s, X_n, Y_n)$ depends on s only via the values $s(x_i)$, $x_i \in X_n$.*

Then, if the optimization problem

$$\operatorname*{argmin}_{g\in\mathcal{H}} J(g) := L(g, X_n, Y_n) + \lambda\|g\|_{\mathcal{H}}^2 \tag{9.5}$$

has a solution, it has in particular a solution of the form

$$s(x) := \sum_{j=1}^{n} \alpha_j K(x, x_j), \quad x \in \Omega, \tag{9.6}$$

for suitable coefficients $\alpha \in \mathbb{R}^n$.

Proof. We prove that for every $g \in \mathcal{H}$ there exists $s \in V(X_n)$ such that $J(s) \le J(g)$. To see this, we decompose $g \in \mathcal{H}$ as

$$g = s + s^{\perp}, \quad s \in V(X_n),\ s^{\perp} \in V(X_n)^{\perp}.$$

In particular, since $K(\cdot, x_i) \in V(X_n)$, we have by the reproducing property of the kernel

$$s^{\perp}(x_i) = \left\langle s^{\perp}, K(\cdot, x_i)\right\rangle_{\mathcal{H}} = 0, \quad 1 \le i \le n,$$

thus $g(x_i) = s(x_i) + s^{\perp}(x_i) = s(x_i)$ for $1 \le i \le n$, and it follows that $L(g, X_n, Y_n) = L(s, X_n, Y_n)$. Moreover, again by orthogonal projection we have $\|g\|_{\mathcal{H}}^2 = \|s\|_{\mathcal{H}}^2 + \|s^{\perp}\|_{\mathcal{H}}^2$. Since $\lambda \ge 0$, we obtain

$$\begin{aligned} J(s) &= L(s, X_n, Y_n) + \lambda\|s\|_{\mathcal{H}}^2 = L(g, X_n, Y_n) + \lambda\|s\|_{\mathcal{H}}^2 \\ &= L(g, X_n, Y_n) + \lambda\|g\|_{\mathcal{H}}^2 - \lambda\|s^{\perp}\|_{\mathcal{H}}^2 = J(g) - \lambda\|s^{\perp}\|_{\mathcal{H}}^2 \le J(g). \end{aligned}$$

Thus, if $g \in \mathcal{H}$ is a solution then $s \in V(X_n)$ is also a solution. □

The existence of a solution will be guaranteed by choosing a convex cost function J, i. e., since the regularization term is always convex, by choosing a convex loss function. Then the theorem states that solutions of the infinite dimensional optimization problem can be computed by solving a finite dimensional convex one.

This is a great result, but observe that the evaluation of $s(x)$, $x \in \Omega$, requires the evaluation of the n-terms linear combination (9.6), where n is the size of the dataset. Assuming that the kernel can be evaluated in constant time, the complexity of this operation is $\mathcal{O}(n)$. Thus, to achieve the promised speedup in evaluating the surrogate in place of the function f, we will consider in the following methods that enforce sparsity in s, i. e., which compute approximate solution where most of the coefficients α_j are zero. If the nonzero coefficients correspond to an index set $I_N := \{i_1, \dots, i_N\} \subset \{1, \dots, n\}$, the complexity is reduced to $\mathcal{O}(N)$.

Taking into account this sparsity and denoting $X_N := \{x_i \in X_n : i \in I_N\}$ and $\alpha := (\alpha_i : i \in I_N)$, we can summarize in Algorithm 9.1 the online phase for any of the following algorithms, consisting in the evaluation of s on a set of points $X_{\text{te}} \subset \Omega$. Here and in the following, we denote by $s(X) := (s(x_1), \dots, s(x_m))^T \in \mathbb{R}^m$ the vector of evaluations of s on a set of points $X := \{x_i\}_{i=1}^m \subset \Omega$.

Algorithm 9.1: Kernel surrogate – online phase.

1: Input: $X_N \in \Omega^N, \alpha \in \mathbb{R}^N$, kernel K (and kernel parameters), test points $X_{\text{te}} := \{x_i^{\text{te}}\}_{i=1}^{n_{\text{te}}} \in \Omega^{n_{\text{te}}}$
2: Compute the kernel matrix $A_{\text{te}} \in \mathbb{R}^{n_{\text{te}} \times N}$, $(A_{\text{te}})_{ij} := K(x_i^{\text{te}}, x_{i_j})$.
3: Evaluate the surrogate $s(X^{\text{te}}) = A_{\text{te}}\alpha$.
4: Output: evaluation of the surrogate $s(X^{\text{te}}) \in \mathbb{R}^{n_{\text{te}}}$.

Remark 9.3 (Normalization of the cost function). It is sometimes convenient to weight the loss term in the cost function (9.5) by a factor $1/n$, which normalizes its value with respect to the number of data. We do not use this convention here, and we only remark that this is equivalent to the use of a regularization parameter $\lambda = n\lambda'$ for a given $\lambda' > 0$.

9.4 Kernel interpolation

The first method that we discuss is (regularized) kernel interpolation. In this case, we consider the square loss function

$$L(s, X_n, Y_n) := \sum_{i=1}^{n} \big(s(x_i) - y_i\big)^2,$$

which measures the pointwise distance between the surrogate and the target data. We have then the following special case of the Representer Theorem. We denote by $y \in \mathbb{R}^n$ the vector of output data, assuming again for now that $q = 1$.

Corollary 9.1 (Regularized interpolant). *Let Ω be a nonempty set, K a PD kernel on Ω, $\lambda \geq 0$ a regularization parameter. For any training set (X_n, Y_n) there exists an approximant of the form*

$$s(x) = \sum_{j=1}^{n} \alpha_j K(x, x_j), \quad x \in \Omega, \tag{9.7}$$

where the vector of coefficients $\alpha \in \mathbb{R}^n$ is a solution of the linear system

$$(A + \lambda I)\alpha = y, \tag{9.8}$$

where $A \in \mathbb{R}^{n \times n}$, $A_{ij} := K(x_i, x_j)$, is the kernel matrix on X_n. Moreover, if K is SPD this is the unique solution of the minimization problem (9.5).

Proof. The loss L is clearly convex, so there exists a solution of the optimization problem, and by Theorem 9.3 we know that we can restrict to solutions in $V(X_n)$.

We then consider functions $s := \sum_{j=1}^{n} \alpha_j K(\cdot, x_j)$ for some unknown $\alpha \in \mathbb{R}^n$. Computing the inner product as in Proposition 9.2, we obtain

$$s(x_i) = \sum_{j=1}^{n} \alpha_j K(x_i, x_j) = (A\alpha)_i, \quad \|s\|_{\mathcal{H}}^2 = \sum_{i,j=1}^{n} \alpha_i \alpha_j K(x_i, x_j) = \alpha^T A \alpha.$$

The functional J restricted to $V(X_n)$ can be parametrized by $\alpha \in \mathbb{R}^n$, and thus it can be rewritten as $\tilde{J} : \mathbb{R}^n \to \mathbb{R}$ with

$$\begin{aligned}\tilde{J}(\alpha) &= \|A\alpha - y\|_2^2 + \lambda \alpha^T A \alpha = (A\alpha - y)^T (A\alpha - y) + \lambda \alpha^T A \alpha \\ &= \alpha^T A^T A \alpha - 2\alpha^T A^T y + y^T y + \lambda \alpha^T A \alpha,\end{aligned}$$

which is convex in α since A is positive semidefinite. Since A is symmetric, its gradient is

$$\nabla_\alpha \tilde{J}(\alpha) = 2A^T A\alpha - 2A^T y + 2\lambda A\alpha = 2A(A\alpha - y + \lambda\alpha),$$

i. e., $\nabla_\alpha \tilde{J}(\alpha) = 0$ if and only if $A(A+\lambda I)\alpha = Ay$, which is satisfied by α such that $(A+\lambda I)\alpha = y$. If K is SPD then both A and $A + \lambda I$ are invertible, so this is the only solution. □

The extension to vector-valued functions, i. e. $q > 1$, is straightforward using the separable matrix-valued kernels with $B = I$ of Section 9.2.4. Indeed, in this case the data values are vectors $y_i := f(x_i) \in \mathbb{R}^q$, and thus in the interpolant (9.7) also the coefficients are vectors $\alpha_j \in \mathbb{R}^q$. The linear system (9.8) has the same matrix, but instead $\alpha, y \in \mathbb{R}^{n \times q}$ are defined as

$$\alpha := (\alpha_1, \ldots, \alpha_n)^T, \quad y := (y_1, \ldots, y_n)^T. \tag{9.9}$$

We remark that in the following the values x_i, y_i, $s(x)$, and α_k have always to be understood as row vectors when $q > 1$. This notation is very convenient when representing the coefficients as the solution of a linear system. Furthermore, the representation of the dataset samples (x, y) is quite natural when dealing with tabular data, where each column represents a feature and each row a sample vector.

For K SPD and pairwise distinct sample locations X_n we can also set $\lambda := 0$ and obtain pure interpolation, i. e., the solution satisfies $L(s, X_n, Y_n) = 0$, or

$$s(x_i) = y_i, \quad 1 \le i \le n.$$

Observe that this means that with this method we can exactly interpolate arbitrary continuous functions on arbitrary pairwise distinct scattered data in any dimension, as opposite to many other techniques which require complicated conditions on the interpolation points or a grid structure. Moreover, this approximation process has several optimality properties in $\mathcal{H}$, which remind one of similar properties of spline interpolation.

Proposition 9.3 (Optimality of kernel interpolation). *Let K be SPD, $f \in \mathcal{H}$, and $\lambda = 0$. Then s is the orthogonal projection of f in $V(X_n)$, and in particular*

$$\|f - s\|_{\mathcal{H}} = \min_{g \in V(X_n)} \|f - g\|_{\mathcal{H}}.$$

Moreover, if $S := \{g \in \mathcal{H} : g(x_i) = f(x_i), 1 \leq i \leq n\}$, then

$$\|s\|_{\mathcal{H}} = \min_{g \in S} \|g\|_{\mathcal{H}},$$

i. e., s is the minimal norm interpolant of f on X_n.

Proof. The proof is analogous to the proof of the Representer Theorem, using a decomposition $f = g + g^{\perp}$, and proving that $s = g$. □

We will see in Section 9.7 a general technique to tune λ using the data, which should return $\lambda = 0$ (or very small) when this is the best option. Nevertheless, also for an SPD kernel there are at least two reasons to still consider regularized interpolation. First, the data can be affected by noise, and thus an exact pointwise recovery does not make much sense. Second, a positive parameter $\lambda > 0$ improves the condition number of the linear system, and thus the stability of the solution. Indeed, the 2-condition number of $A + \lambda I$ is

$$\kappa(\lambda) := \frac{\lambda_{\max}(A + \lambda I)}{\lambda_{\min}(A + \lambda I)} = \frac{\lambda_{\max}(A) + \lambda}{\lambda_{\min}(A) + \lambda},$$

which is a strictly decreasing function of λ, with $\kappa(0) = \kappa(A)$ and $\lim_{\lambda \to \infty} \kappa(\lambda) = 1$. Moreover (see [66]) this increased stability can be achieved by still controlling the pointwise accuracy. Namely, if $f \in \mathcal{H}$, we have

$$\left\|y_i - s(x_i)\right\|_2 \leq \sqrt{\lambda} \|f\|_{\mathcal{H}} \quad 1 \leq i \leq n.$$

We can then summarize the offline phase for regularized kernel interpolation in Algorithm 9.2.

Algorithm 9.2: Regularized Kernel interpolation – offline phase.

1: Input: training set $X_n \in \Omega^n$, $Y_n \in (\mathbb{R}^q)^n$, kernel K (and kernel parameters), regularization parameter $\lambda \geq 0$.
2: Compute the kernel matrix $A \in \mathbb{R}^{n \times n}$, $A_{ij} := K(x_i, x_j)$.
3: Solve the linear system $(A + \lambda I)\alpha = y$.
4: Output: coefficients $\alpha \in \mathbb{R}^{n \times q}$.

Remark 9.4 (Flat limit). The matrix A can be seriously ill-conditioned for certain kernels, and this constitutes a problem at least in the case of pure interpolation. It can also be proven that kernels which guarantee a faster error convergence result in a worse conditioned matrix [48].

For RBF kernels, this happens especially for $\gamma \to 0$ (the so called flat limit), and it is usually not a good idea to directly solve the linear system. In the last years there has been very active research to compute s via different formulations, which rely on different representations of the kernel. We mention here mainly the RBF-QR algorithm[3] [18, 31] and the Hilbert–Schmidt SVD[4] [16] . Both methods are limited so far to only some kernels, but they manage to achieve a great accuracy, which is usually impossible to obtain with the direct solution of the linear system.

Remark 9.5 (Error estimation). For SPD translational invariant kernels there is a very detailed error analysis of the interpolation process ($\lambda = 0$), for which we refer to [65, Chapter 11]. We only mention that these error bounds assume that $f \in \mathcal{H}$, and are of the form

$$\|f - s\|_{L_\infty(\Omega)} \leq C h_n^p \|f\|_{\mathcal{H}},$$

where $C > 0$ is a constant independent of f, and h_n is the fill distance of X_n in Ω, i. e.,

$$h_n := h_{X_n,\Omega} := \sup_{x\in\Omega} \min_{x_j \in X_n} \|x - x_j\|,$$

which is the analogue of the mesh width for scattered data. Moreover, the order of convergence $p > 0$ is dependent on the smoothness of the kernel. In particular, these error bounds can be proven to be optimal when the native space of K is a Sobolev space.

Moreover, these results have been recently extended to the case of regularized interpolation ($\lambda > 0$) in [43, 66].

9.4.1 Kernel greedy approximation

The surrogate constructed via Corollary 9.1 involves a linear combination of n terms, where n is the size of the dataset. In general, there is no reason to assume that the result has any sparsity, i. e., in general all the α_j will be nonzero, and it is thus necessary to introduce some technique to enforce this sparsity.

A very effective way to achieve this result is via greedy algorithms. The idea is to select a small subset $X_N \subset X_n$, $N \ll n$, given by indices $I_N \subset \{1, \dots, n\}$, and to solve the

3 http://www.it.uu.se/research/scientific_computing/software/rbf_qr

4 http://math.iit.edu/~mccomic/gaussqr/

corresponding restricted problem with the dataset (X_N, Y_N) to compute a surrogate

$$s_N(x) := \sum_{k \in I_N} \alpha_k K(x, x_k), \tag{9.10}$$

where the coefficient vectors are computed based on (9.8), and are in general different from the ones of the full surrogate. If we manage to select I_N in a proper way, we will obtain $s_N(x) \approx f(x)$ for all $x \in \Omega$, while the evaluation of $s_N(x)$ is now only of order $\mathcal{O}(N)$.

An optimal selection of X_N is a combinatorial problem and thus is very expensive and in practice computationally intractable. The idea of greedy algorithms is instead to perform this selection incrementally, i. e., adding at each iteration only the most promising new point, based on some error indicator.

The general structure of the algorithm is described in Algorithm 9.3. For the moment, we consider a generic selection rule $\eta : X_n \times \mathbb{N} \times \Omega^n \times (\mathbb{R}^q)^n \to \mathbb{R}_{\geq 0}$ that selects points based on the value $\eta(x, N, X_n, Y_n)$. This is a compact notation to denote that the selection rule assigns a score to a point $x \in \Omega$, and it is computed using various quantities that depend on the dataset (X_n, Y_n) and on the iteration number N, including in particular the surrogate computed at the previous iteration. The algorithm is terminated by means of a given tolerance $\tau > 0$.

Algorithm 9.3: Kernel greedy approximation – offline phase.

1: Input: training set $X_n \in \Omega^n$, $Y_n \in (\mathbb{R}^q)^n$, kernel K (and kernel parameters), regularization parameter $\lambda \geq 0$, selection rule η, tolerance τ.
2: Set $N := 0$, $X_0 := \emptyset$, $V(X_0) := \{0\}$, $s_0 := 0$.
3: **repeat**
4: Set $N := N + 1$
5: Select $x_N := \operatorname{argmax}_{x \in X_n \setminus X_{N-1}} \eta(x, N, X_n, Y_n)$.
6: Define $X_N := X_{N-1} \cup \{x_N\}$ and $V(X_N) := \operatorname{span}\{K(\cdot, x_i), x_i \in X_N\}$
7: Compute the surrogate s_N with dataset (X_N, Y_N) with (9.8).
8: **until** $\eta(x_N, N, X_n, Y_n) \leq \tau$
9: Output: surrogate s_N (i. e. coefficients $\alpha \in \mathbb{R}^{N \times q}$).

Remark 9.6. In the case that the maximizer of η the line 5 of Algorithm 9.3 is not unique, only one of the multiple points is selected and included in X_N.

In line 7 of the algorithm, we need to compute the surrogate s_N with dataset (X_N, Y_N). This step can be highly simplified by reusing s_{N-1} as much as possible, thus improving the efficiency of the algorithm. As a side effect, with this incremental procedure it is easy to update the surrogate if the accuracy has to be improved.

This can be achieved using the Newton basis, which is defined in analogy to the Newton basis for polynomial interpolation. It has been introduced in [37, 39] for K SPD, and extended to the case of K PD and $\lambda > 0$ in [47], and we refer to these papers for the proof of the following result.

Proposition 9.4 (Newton basis). *Let Ω be non empty, $\lambda \geq 0$, K be PD on Ω or SPD when $\lambda = 0$. Let $X_n \subset \Omega$ be pairwise distinct, and let $I_N \subset \{1, \dots, n\}$. Let moreover $K_\lambda(x, y) := K(x, y) + \lambda \delta_{xy}$ for all $x, y \in \Omega$, and denote its RKHS as $\mathcal{H}_\lambda$.*

The Newton basis $\{v_j\}_{j=1}^N$ is defined as the Gram–Schmidt orthonormalization of $\{K_\lambda(\cdot, x_i)\}_{i \in I_N}$ in $\mathcal{H}$, i. e.,

$$
\begin{aligned}
v_1(x) &:= \frac{K_\lambda(x, x_{i_1})}{\|K_\lambda(\cdot, x_{i_1})\|_{\mathcal{H}_\lambda}} = \frac{K_\lambda(x, x_{i_1})}{\sqrt{K_\lambda(x_{i_1}, x_{i_1})}}, \\
\tilde{v}_k(x) &:= K_\lambda(x, x_{i_k}) - \sum_{j=1}^{k-1} v_j(x_{i_k}) v_j(x), \\
v_k(x) &:= \frac{\tilde{v}_k(x)}{\|\tilde{v}_k\|_{\mathcal{H}_\lambda}} = \frac{\tilde{v}_k(x)}{\sqrt{\tilde{v}_k(x_{i_k})}}, \quad 1 < k \leq N.
\end{aligned}
$$

Moreover, for all $1 \leq k \leq N$, we have

$$
v_k(x) = \sum_{j=1}^{N} \beta_{jk} K_\lambda(x, x_{i_j}),
$$

and, if $B \in \mathbb{R}^{N \times N}$, $B_{jk} := \beta_{jk}$, and $V \in \mathbb{R}^{N \times N}$, $V_{jk} := v_k(x_j)$, then B, V are triangular, $B = V^{-T}$, and

$$
A_N + \lambda I = V V^T
$$

is the Cholesky decomposition of the regularized kernel matrix $A_N + \lambda I \in \mathbb{R}^{N \times N}$, $A_{jk} := K(x_{i_j}, x_{i_k})$, with pivoting given by I_N.

Observe that this basis is nested, i. e., we can incrementally add a new element without recomputing the previous ones. Even more, with this basis the surrogate can be computed as follows.

Proposition 9.5 (Incremental regularized interpolation). *Let Ω be non empty, $\lambda \geq 0$, K be PD on Ω or SPD when $\lambda = 0$. Let (X_N, Y_N) be the subset of (X_n, Y_n) corresponding to indices I_N, for all $N \leq n$.*

Let $\tilde{s}_0 := 0$, and, for $N \geq 1$, compute the following incremental function

$$
\tilde{s}_N(x) = \sum_{k=1}^{N} c_k v_k(x) = c_N v_N(x) + s_{N-1}(x), \quad c_N := \frac{y_{i_N} - \tilde{s}_N(x_{i_N})}{v_N(x_{i_N})}. \tag{9.11}
$$

Then, for all N, the regularized interpolant can be computed as

$$s_N(x) = \sum_{j=1}^{N} \alpha_j K(x, x_{i_j}) \quad \text{where } \alpha := V^{-T} c.$$

Remark 9.7. In the case $\lambda = 0$ and K SPD, the function $\tilde{s}_N$ coincides with the interpolant s_N. We refer to [39, 47] for the details.

We are now left to define the selection rules, represented by η, to select the new point at each iteration.

For this, we first need to define the power function, which gives an upper bound on the interpolation error, and it can be defined using the Newton basis as

$$P_N(x)^2 := K_\lambda(x,x) - \sum_{j=1}^{N} v_j(x)^2. \tag{9.12}$$

Its relevance is due to the fact that it provides an upper bound on the pointwise (regularized) interpolation error, i. e., if $x \notin X_n$, and K is PD, or SPD when $\lambda = 0$, we have for all $f \in \mathcal{H}$ that

$$|f(x) - s_N(x)| \leq P_n(x) \|f\|_{\mathcal{H}}. \tag{9.13}$$

This function is well known and has been studied in the case of pure interpolation (see e. g. [65, Chapter 11]), for which the upper bound holds for all $x \in \Omega$, and it can be easily extended to the case of regularized interpolation (see [47]). In both cases, it can be proven that $P_n(x) = 0$ if and only if $x \in X_n$, and its maximum is strictly decreasing with N.

Remark 9.8. This interpolation technique is strictly related to the kriging method and to Gaussian Process Regression (see e. g. [38, 42]). In this case the kernel represents the covariance kernel of the prior distribution, and the power function is the Kriging variance, or variance of the posterior distribution (see [50]).

We can then define the following selection rules. We assume to have a dataset (X_n, Y_n), and to have already selected N points corresponding to indices I_{N-1}. We use the notation $[1,n] := \{1,\dots,n\}$, and we have

- P-greedy: $i_N := \operatorname{argmax}_{i \in [1,n] \setminus I_{N-1}} P_{N-1}(x_i)$;
- f-greedy: $i_N := \operatorname{argmax}_{i \in [1,n] \setminus I_{N-1}} |y_i - s_{N-1}(x_i)|$;
- f/P-greedy: $i_N := \operatorname{argmax}_{i \in [1,n] \setminus I_{N-1}} \frac{|y_i - s_{N-1}(x_i)|}{P_{N-1}(x_i)}$.

Observe that all the selections are well defined, since $P_{N-1}(x_i) \neq 0$ for all $i \notin I_{N-1}$ if X_N are pairwise distinct, and they can be efficiently implemented by using the update rules (9.11) for s_N and (9.12) for P_N. Moreover, they are motivated by different ideas: The P-greedy selection tries to minimize the Power function, thus providing a uniform upper bound on the error for any function $f \in \mathcal{H}$ via (9.13); the f- and f/P-greedy (which

reads "f-over-P-greedy"), on the other hand, use also the output data, and produce points which are suitable to approximate a single function and thus are expected to result in a better approximation. In the case of f-greedy this is done by including in the set of points the location where the current largest error is achieved, thus reducing the maximum error. The f/P-greedy selection, instead, reduces the error in the $\mathcal{H}$-norm, and indeed it can be proven to be locally optimal, i. e., it guarantees the maximal possible reduction of the error, in the $\mathcal{H}$-norm, at each iteration.

We can now describe the full computation of the greedy regularized interpolant in Algorithm 9.4. It realizes the computation of the sparse surrogate s_N by selecting the points X_N via the index set I_N, and computing only the nonzero coefficients α. Moreover, using the nested structure of the Newton basis and the incremental computation of Proposition 9.5, the algorithm needs only to compute the columns of the full kernel matrix corresponding to the index set I_N, and thus there is no need to compute nor store the full $n \times n$ matrix, i. e., the implementation is matrix-free. In addition, again using Proposition 9.5 most of the operations are done in-place, i. e., some vectors are used to store and update the values of the Power Function and of y. In the algorithm, we use a Matlab-like notation, i. e., $A(I_N, :)$ denotes the submatrix of A consisting of rows I_N and of all the columns. Moreover, the notation v^2 denotes the pointwise squaring of the entries of the vector v.

Algorithm 9.4: Kernel greedy approximation – offline phase.

Input: training set $X_n \in \Omega^n$, $Y_n \in (\mathbb{R}^q)^n$, kernel K (and kernel parameters), regularization parameter $\lambda \geq 0$, selection rule η, tolerance τ.
Set $N := 0$, $I_0 := \emptyset$, $V := [\cdot] \in \mathbb{R}^{n \times 0}$, $p := \mathrm{diag}(K_\lambda(X_n, X_n)) \in \mathbb{R}^n$
repeat
    Set $N = N + 1$
    Select $i_N := \mathrm{argmax}_{i \in [1,n] \setminus I_{N-1}} \ \eta(x_i, N, X_n, Y_n)$.
    Generate column $v := K_\lambda(X_n, x_{i_N})$
    Project $v := v - VV(i_N, :)^T$
    Normalize $v = v/\sqrt{v(i_N)}$
    Compute $c_N := y(i_N)/v(i_N)$
    Update the power function $p := p - v^2$
    Update the residual $y := y - c_N v$
    Update $I_N := I_{N-1} \cup \{i_N\}$
    Add the column $V = [V, v_N]$
    Update the inverse $C^T = V(I_N, :)^{-1}$
    Add the coefficient $c = [c^T, c_N]^T$
until $\eta(x_N, N, X_n, Y_n) \leq \tau$
Set $\alpha = Cc$
Output: $\alpha \in \mathbb{R}^{N \times q}$, I_N.

The set of points X_N defined by I_N, and the coefficients α, can then be used in the online phase of Algorithm 9.1.

Remark 9.9 (Vector-valued functions and implementation details). Algorithm 9.4 and the overall procedure are well defined for arbitrary $q \geq 1$. Indeed, using the separable matrix-valued kernel of Section 9.2.4, the Newton basis only depends on the scalar-valued kernel K, while the computation of the coefficients is valid by considering that now c, α are matrices instead of vectors. In particular, the computation of c_N (line 14) and the update of y (line 11) has to be done via column-wise multiplications.

Moreover, observe that in line 12 we employ a standard technique to update the inverse of a lower triangular matrix, i. e., given $V_N \in \mathbb{R}^{N\times N}$ lower triangular with inverse V_N^{-1}, we define

$$V_{N+1} = \begin{bmatrix} V_N & 0 \\ v^T & w \end{bmatrix}$$

for $v \in \mathbb{R}^N$, $w \in \mathbb{R}$, and compute V_{N+1}^{-1} by a simple row-update as

$$V_{N+1}^{-1} = \begin{bmatrix} V_N^{-1} & 0 \\ -v^T V_N^{-1}/w & 1/w \end{bmatrix}.$$

The present version of the algorithm for vector-valued functions has been introduced in [68] and named Vectorial Kernel Orthogonal Greedy Algorithm (VKOGA). We keep the same abbreviation also for the regularized version, which has been studied in [47].

Remark 9.10 (Convergence rates). When the greedy algorithm is run by selecting points over Ω instead of X_N, there are also convergence rates for the resulting approximation processes. For pure interpolation (i. e., K SPD, $\lambda = 0$) convergence of f-greedy has been proven in [36], of P-greedy in [46], and of f/P-greedy in [68], while in [47] the convergence rate of P-greedy has been extended to regularized interpolation. All the results make additional assumptions on the kernels, for which we refer to the cited literature. Nevertheless, we remark that the convergence rates for interpolation with P-greedy are quasi-optimal for translational invariant kernels, while the results for the other algorithms guarantee only a possibly significantly slower convergence rate. These results are believed to be significantly sub-optimal, since extensive experiments indicate that f- and f/P-greedy cases behave much better. This seems to suggest that there is space for a large improvement in the theoretical understanding of the methods.

Remark 9.11 (Other techniques). There are other techniques that can be applied to reduce the complexity of the evaluation of the surrogate s, which do not use greedy algorithms but instead different approaches. First, there is a domain decomposition technique, known as Partition of Unity Method, which partitions Ω into subdomains,

solves the (regularized) interpolation problem restricted to each patch, and then combines the results by a weighted sum of the local interpolants to obtain a global approximant. This method has the advantage that this offline phase can be completely parallelized. Moreover, when evaluating the surrogate only the few local interpolant having a support containing the test points have to be evaluated, thus requiring the evaluation of a few, small kernel expansions, thus providing a significant speedup. The efficiency of this technique relies on an efficient search procedure to determine the local patches including the given points, which is the only limitation in the application to high dimensional problems. Both theoretical results and efficient implementations are available [7, 64].

Moreover, other sparsity-inducing techniques have been proposed, namely, the use of an ℓ_1-regularization term [10], and the method of the Least Absolute Shrinkage and Selection Operator (LASSO) [61].

9.5 Support vector regression

The second method that we present is Support Vector Regression (SVR) [53], which is based on different premises, but it still fits in the general framework of Section 9.3. In this case, we consider the ε-insensitive loss function

$$L(s, X_n, Y_n) := \sum_{i=1}^{n} L_\varepsilon(s(x_i), y_i), \quad L_\varepsilon(s(x_i), y_i) := \max(0, |s(x_i) - y_i| - \varepsilon),$$

which is designed to linearly penalize functions s which have values outside of an ε-tube around the data, while no distinction is made between function values that are inside this tube.

In this setting it is common to use the regularization parameter to scale the cost by a factor $1/\lambda$, and not the regularization term by a factor λ. The two choices are clearly equivalent, but we adopt here this different normalization to facilitate the comparison with the existing literature, and because this offers additional insights in the structure of the surrogate.

Since the problem is not quadratic (and not smooth), we first derive an equivalent formulation of the optimization problem (9.5). Assuming again that the output is scalar, i. e., $q = 1$, the idea is to introduce non-negative slack variables $\xi^+, \xi^- \in \mathbb{R}^n$ which represent upper bounds on L via

$$\begin{aligned} \xi_i^+ &\geq \max(0, s(x_i) - y_i - \varepsilon), \quad 1 \leq i \leq n, \\ \xi_i^- &\geq \max(0, y_i - s(x_i) - \varepsilon), \quad 1 \leq i \leq n, \end{aligned} \tag{9.14}$$

and to minimize them in place of the original loss. With these new variables we can rewrite the optimization problem in the following equivalent way.

Definition 9.5 (SVR – primal form). Let Ω be a nonempty set, K a PD kernel on Ω, $\lambda > 0$ a regularization parameter. For a training set (X_n, Y_n) the SVR approximant $(s, \xi^+, \xi^-) \in \mathcal{H} \times \mathbb{R}^{2n}$ is a solution of the quadratic optimization problem

$$\begin{aligned} \min_{s\in\mathcal{H},\, \xi^+,\, \xi^-\in\mathbb{R}^n} & \frac{1}{\lambda}\mathbb{1}_n^T(\xi^+ + \xi^-) + \|s\|_{\mathcal{H}}^2 \\ \text{s.t. } & s(x_i) - y_i - \varepsilon \le \xi_i^+, \quad 1 \le i \le n \\ & -s(x_i) + y_i - \varepsilon \le \xi_i^-, \quad 1 \le i \le n \\ & \xi_i^+, \xi_i^- \ge 0, \quad 1 \le i \le n, \end{aligned} \tag{9.15}$$

where $\mathbb{1}_n := (1, \dots, 1)^T \in \mathbb{R}^n$.

For this rewriting of the optimization problem, we can now specialize the Representer Theorem as follows.

Corollary 9.2 (SVR – alternative primal form). *Let Ω be a nonempty set, K a PD kernel on Ω, $\lambda > 0$ a regularization parameter. For any training set (X_n, Y_n) there exists an SVR approximant of the form*

$$s(x) = \sum_{j=1}^{n} \alpha_j K(x, x_j), \quad x \in \Omega, \tag{9.16}$$

where $(\alpha, \xi^+, \xi^-) \in \mathbb{R}^{3n}$ is a solution of the quadratic optimization problem

$$\begin{aligned} \min_{\alpha,\xi^+,\xi^-\in\mathbb{R}^n} & \frac{1}{\lambda}\mathbb{1}_n^T(\xi^+ + \xi^-) + \alpha^T A\alpha \\ s.t. \ & (A\alpha)_i - y_i - \varepsilon \le \xi_i^+, \quad 1 \le i \le n \\ & -(A\alpha)_i + y_i - \varepsilon \le \xi_i^-, \quad 1 \le i \le n \\ & \xi_i^+, \xi_i^- \ge 0, \quad 1 \le i \le n, \end{aligned} \tag{9.17}$$

with $\mathbb{1}_n := (1, \dots, 1)^T \in \mathbb{R}^n$, and $A \in \mathbb{R}^{n\times n}$, $A_{ij} := K(x_i, x_j)$, the kernel matrix on X_n. Moreover, if K is SPD this is the unique solution of the minimization problem (9.5).

Proof. The result is an immediate consequence of Proposition 9.5, where we use the form (9.16) for s and compute its squared norm via Proposition 9.2. □

The slack variables (9.14) have a nice geometric interpretation. Indeed, the optimization process clearly tries to reduce their value as much as possible, while respecting the constraints. We state a more precise result in the following proposition, and give a schematic illustration in Figure 9.1.

Proposition 9.6 (Slack variables). *Let $\alpha, \xi^+, \xi^- \in \mathbb{R}^n$ be a solution of* (9.17)*, and let s be the corresponding surrogate* (9.16)*. Then, for each index $i \in \{1, \dots, n\}$, the values ξ_i^+, ξ_i^- represent the distance of $s(x_i)$ from the ε-tube around y_i, and in particular*

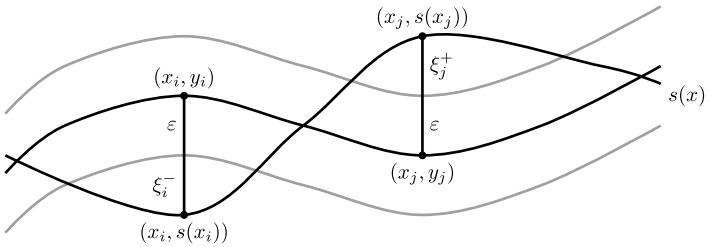

Figure 9.1: Illustration of the role of the slack variables in (9.17).

1. *If* $s(x_i) > y_i + \varepsilon$, *then* $\xi_i^+ > 0$ *and* $\xi_i^- = 0$.
2. *If* $s(x_i) < y_i - \varepsilon$, *then* $\xi_i^+ = 0$ *and* $\xi_i^- > 0$.
3. *If* $y_i - \varepsilon \le s(x_i) \le y_i + \varepsilon$, *then* $\xi_i^+ = 0$ *and* $\xi_i^- = 0$.

In particular, only one of ξ_i^+ *and* ξ_i^- *can be nonzero.*

Instead of solving the primal problem of Corollary 9.2, it is more common to derive and solve the following dual problem. Here again we denote by $y \in \mathbb{R}^n$ the vector of all scalar training target values.

Proposition 9.7 (SVR – dual form). *Let* Ω *be a nonempty set,* K *a PD kernel on* Ω, $\lambda > 0$ *a regularization parameter. For any training set* (X_n, Y_n) *there exists a solution* $(\alpha^+, \alpha^-) \in \mathbb{R}^{2n}$ *of the problem*

$$\min_{\alpha^+, \alpha^- \in \mathbb{R}^n} \frac{1}{4}(\alpha^- - \alpha^+)^T A(\alpha^- - \alpha^+) + \varepsilon \mathbb{1}_n^T(\alpha^+ + \alpha^-) + y^T(\alpha^+ - \alpha^-)$$
$$s.t.\ \alpha^+, \alpha^- \in [0, 1/\lambda]^n, \tag{9.18}$$

which is unique if K *is SPD. Moreover, a solution of* (9.17) *is given by*

$$s(x) := \sum_{j=1}^{n} \frac{\alpha_j^- - \alpha_j^+}{2} K(x, x_j), \quad x \in \Omega, \tag{9.19}$$

with $\xi_i^+ := \max(0, s(x_i) - y_i - \varepsilon)$, $\xi_i^- := \max(0, y_i - s(x_i) - \varepsilon)$.

Proof. We give a sketch of the proof, although a formal derivation requires more care, and we refer to [53, Chapter 9] for the details. The idea is to first derive the Lagrangian $\mathcal{L} := \mathcal{L}(\alpha, \xi^+, \xi^-; \alpha^+, \alpha^-, \mu^+, \mu^-)$ for the primal problem (9.17) using non-negative Lagrange multipliers α^+, α^-, μ^+, $\mu^- \in \mathbb{R}^n$ for the inequality constraints, and then derive the dual problem by imposing the Karush–Kuhn–Tucker (KKT) conditions (see e. g. Chapter 6 in [53]).

The Lagrangian is defined as

$$\mathcal{L} = \frac{1}{\lambda}\mathbb{1}_n^T(\xi^+ + \xi^-) + \alpha^T A\alpha + (\mu^+)^T(-\xi^+) + (\mu^-)^T(-\xi^-) \tag{9.20}$$
$$+ (A\alpha - y - \varepsilon\mathbb{1}_n - \xi^+)^T\alpha^+ + (y - A\alpha - \varepsilon\mathbb{1}_n - \xi^-)^T\alpha^-$$

$$= (\alpha + \alpha^+ - \alpha^-)^T A\alpha + \left(\frac{1}{\lambda}\mathbb{1}_n - \alpha^+ - \mu^+\right)^T \xi^+ + \left(\frac{1}{\lambda}\mathbb{1}_n - \alpha^- - \mu^-\right)^T \xi^- - \varepsilon \mathbb{1}_n^T(\alpha^+ + \alpha^-) - y^T(\alpha^+ - \alpha^-).$$

Using the symmetry of A, the partial derivatives of $\mathcal{L}$ with respect to the primal variables can be computed as

$$\nabla_\alpha \mathcal{L} = 2A\alpha + A(\alpha^+ - \alpha^-), \quad \nabla_{\xi^+}\mathcal{L} = \frac{1}{\lambda}\mathbb{1}_n - \alpha^+ - \mu^+, \quad \nabla_{\xi_i^-}\mathcal{L} = \frac{1}{\lambda}\mathbb{1}_n - \alpha^- - \mu^-, \tag{9.21}$$

and setting these three equalities to zero we obtain equations for α, μ^+, μ^-, where in particular $\alpha = \frac{1}{2}(\alpha^- - \alpha^+)$ (which is the unique solution if A is invertible). Substituting these values in the Lagrangian we get

$$\begin{aligned}\mathcal{L} &= (\alpha + \alpha^+ - \alpha^-)^T A\alpha - \varepsilon \mathbb{1}_n^T(\alpha^+ + \alpha^-) - y^T(\alpha^+ - \alpha^-) \\ &= -\frac{1}{4}(\alpha^- - \alpha^+)^T A(\alpha^- - \alpha^+) - \varepsilon \mathbb{1}_n^T(\alpha^+ + \alpha^-) - y^T(\alpha^+ - \alpha^-).\end{aligned}$$

The remaining conditions in (9.18) stem from the requirements that the Lagrange multipliers are non-negative, and in particular $0 \le \mu_i^+ = 1/\lambda - \alpha_i^+$, i. e., $\alpha_i^+ \le 1/\lambda$, and similarly for α_i^-. □

This dual formulation is particularly convenient to explain that the SVR surrogate has a built-in sparsity, i. e., the optimization process provides a solution where possibly many of the entries of $\alpha = \frac{1}{2}(\alpha^- - \alpha^+)$ are zero. This behavior is in strong contrast with the case of interpolation of Section 9.4 where we needed to adopt special techniques to enforce this property. The points $x_i \in X_n$ with $\alpha_i \neq 0$ are called support vectors, which gives the name to the method.

In particular, as for the slack variables there is a clean geometric description of this sparsity pattern, this gives additional insights into the solution. To see this we remark that, in addition to the stationarity KKT conditions (9.21), an optimal solution satisfies also the complementarity KKT conditions

$$\alpha_i^+(s(x_i) - y_i - \varepsilon - \xi_i^+) = 0, \quad \alpha_i^-(y_i - s(x_i) - \varepsilon - \xi_i^-) = 0, \tag{9.22}$$

$$\xi_i^+(1/\lambda - \alpha_i^+) = 0, \quad \xi_i^-(1/\lambda - \alpha_i^-) = 0. \tag{9.23}$$

We then have the following:

1. Equation (9.22) states that $\alpha_i^+ \neq 0$ only if $s(x_i) - y_i - \varepsilon - \xi_i^+ = 0$, and similarly for α_i^-. Since $\xi_i^+ \ge 0$, this happens only when $s(x_i) - y_i \ge \varepsilon$, i. e., only for points $(x_i, s(x_i))$ which are outside or on the boundary of the ε-tube.
2. In particular, if $\alpha_i^+ \neq 0$ it follows that $s(x_i) - y_i \ge \varepsilon$, and thus $y_i - s(x_i) - \varepsilon - \xi_i^- \neq 0$, and then necessarily $\alpha_i^- = 0$. Thus, at most one of α_i^+ and α_i^- can be nonzero.
3. Equation (9.23) implies that $\alpha_i^+, \alpha_i^- = 1/\lambda$ whenever ξ_i^+, ξ_i^- is nonzero, i. e., whenever $s(x_i)$ is strictly outside of the ε-tube. The corresponding x_i are called bounded

support vectors, and the value of the corresponding coefficients is indeed kept bounded by the value of the regularization parameter. Reducing λ, i. e., using less regularization, allows solutions with coefficients of larger magnitude.

In summary, we can then expect that, if s is a good approximation of the data, it will be also a sparse approximation.

We summarize the offline phase for SVR in Algorithm 9.5. We remark that in this case the extension to vector-valued functions is not as straightforward as for kernel interpolation, and it is thus common to train a separate SVR for each output component.

Algorithm 9.5: SVR – offline phase.

1: Input: training set $X_n \in \Omega^n$, $Y_n \in \mathbb{R}^n$, kernel K (and kernel parameters), regularization parameter $\lambda \geq 0$, tube width $\varepsilon > 0$.
2: Compute the kernel matrix $A \in \mathbb{R}^{n \times n}$, $A_{ij} := K(x_i, x_j)$.
3: Solve the quadratic problem (9.18).
4: Set $I_N := \{i : \alpha_i^- \neq 0 \text{ or } \alpha_i^+ \neq 0\}$.
5: Set $\alpha_i := (\alpha_i^- - \alpha_i^+)/2$ for $i \in I_N$.
6: Output: $\alpha \in \mathbb{R}^N$, I_N.

Remark 9.12 (General Support Vector Machines). SVR is indeed one member of a vast collection of algorithms related to Support Vector Machines (SVMs). Standard SVMs solve classification problems, i. e., $Y_n \subset \{0, 1\}$. The original algorithm has been introduced as a linear algorithm (or, in the present understanding, as limited to the linear kernel, i. e., the polynomial kernel with $a = 0$, $p = 1$), and it has later been extended via the kernel trick to its general kernel version in [4]. The SVR algorithms have instead been introduced in [53].

Moreover, the version presented here is usually called ε-SVR. There exists also another non equivalent version called ν-SVR, which adds another term in the cost function multiplied by a factor $\nu \in [0, 1]$. This plays the role of giving an upper bound on the number of support vectors and on the fraction of training data which are outside of the ε-tube (see Chapter 9 in [53]).

We also remark that it is sometimes common to include in any SVM-based algorithm also an offset or bias term $b \in \mathbb{R}$, i. e., to obtain a surrogate $s(x) = \sum_{j=1}^n \alpha_j K(x, x_j) + b$. This changes in an obvious way the primal problem (9.17), while the dual contains also the constraint $\sum_{i=1}^n (\alpha_i^+ + \alpha_i^-) = 0$. However, we stick here to this formulation and refer to [57] for a discussion of statistical and numerical benefits of not using this offset term, at least in the case of SPD kernels.

Remark 9.13 (Error estimation). Also for SVR there is a detailed error theory, usually formulated in the framework of statistical learning theory (see [62]). Results are ob-

tained by assuming that the dataset (X_n, Y_n) is drawn from a certain unknown probability distribution, and then quantifying the approximation power of the surrogate. For a detailed treatment of this theory, we refer to [53, 55]. Moreover, recently also deterministic error bounds for translational invariant kernels have been proven in [43, 44].

9.5.1 Sequential minimal optimization

Although the optimization problem (9.18) can in principle be solved with any quadratic optimization method, there exists a special algorithm, called Sequential Minimal Optimization (SMO) that is designed for SVMs and that performs possibly much better.

SMO is an iterative method which improves an initial feasible guess for $\alpha^+, \alpha^- \in \mathbb{R}^n$ until convergence, and the update is made such that the minimal possible number of entries of α are affected. In this way, very large problems can be efficiently solved. The original version of the algorithm has been introduced in [41] for SVM, and it has later been adapted to more general methods such as SVR, which we use here to illustrate the structure of its implementation.

The idea is to find at each iteration $\ell \in \mathbb{N}$ a minimal set of indices $I^\ell \subset \{1, \dots, n\}$ and optimize only the variables with indices in I^ℓ. The procedure is then iterated until the optimum is reached. If the SVR includes an offset term, as explained in the previous section we have constraints

$$\begin{aligned} &\alpha_i^+, \alpha_i^- \in [0, 1/\lambda], \quad 1 \le i \le n, \\ &\sum_{i=1}^{n} (\alpha_i^+ + \alpha_i^-) = 0. \end{aligned} \tag{9.24}$$

Given a feasible solution $(\alpha_i^+, \alpha_i^-)^{(\ell)}$ at iteration $\ell \in \mathbb{N}$, it is thus not possible to update a single entry of α_i^+ or α_i^- without violating the KKT conditions (since at most one between α_i^+ and α_i^- need to be nonzero) or violating the second constraint. It is instead possible to select two indices $I^\ell := \{i, j\}$ and in this case we have variables $\alpha_i^+, \alpha_i^-, \alpha_j^+, \alpha_j^-$ and we can solve the restricted quadratic optimization problem under the constraints

$$\alpha_i^+, \alpha_i^- \in [0, 1/\lambda], i \in I^\ell, \quad \sum_{i \in I^\ell} (\alpha_i^+ + \alpha_i^-) = R^\ell := -\sum_{i \notin I^\ell} (\alpha_i^+ + \alpha_i^-),$$

which can be solved analytically.

The crucial step is to select I^ℓ, and this is done by finding a first index that does not satisfy the KKT conditions and a second one with some heuristic. It can be proven that, if at least one of the two violates the KKT conditions, then the objective is strictly decreased and convergence is obtained. Moreover, the vectors $\alpha^+ = \alpha^- = 0 \in \mathbb{R}^n$ are always feasible and can thus be used as a first guess. In practice, the iteration is stopped when a sufficiently small value of the cost function is reached.

In the case of SVR without offset discussed in the previous section the situation is even simpler, since the second constraint in (9.24) is not present and it is thus possible to update a single pair (α_i^+, α_i^-) at each iteration. Nevertheless, it has been proven in [57] that using also in this case two indices improves significantly the speed of convergence. Moreover, the same paper introduces several additional details to select the pair, to optimize the restricted cost function, and to establish termination conditions.

A general version of SMO for SVR is summarized in Algorithm 9.6, where we assume that the function $\eta : \{1, \dots, n\} \to \{1, \dots, n\}$ implements the selection rule of I^ℓ.

Algorithm 9.6: SMO.

Input: training set $X_n \in \Omega^n$, $Y_n \in \mathbb{R}^n$, kernel K (and kernel parameters), regularization parameter $\lambda \geq 0$, tube width $\varepsilon > 0$, selection rule η, tolerance τ.
Set $\ell := 0$ and $(\alpha^+, \alpha^-)^{(0)} := (0, 0)$.
while $(\alpha^+, \alpha^-)^{(\ell)}$ does not satisfy KKT conditions within tolerance τ. **do**
    Set $\ell = \ell + 1$.
    Set $I^\ell := \{i, j\} := \eta(\{1, \dots, n\})$.
    Set $(\alpha_k^+, \alpha_k^-)^{(\ell)} := (\alpha_k^+, \alpha_k^-)^{(\ell-1)}$ for $k \notin I^\ell$.
    Solve the optimization problem restricted to I^ℓ.
end while
Set $I_N := \{i : \alpha_i^- \neq 0 \text{ or } \alpha_i^+ \neq 0\}$.
Set $\alpha_i := (\alpha_i^- - \alpha_i^+)/2$ for $i \in I_N$.
Output: $\alpha \in \mathbb{R}^N$, I_N.

Remark 9.14 (Reference implementations). We remark that there exist commonly used implementations of SVR (and other SVM-related algorithms), which are available in several programming languages and implement also some version of this algorithm. We mention especially LIBSVM[5] [9] and liquidSVM[6] [58].

9.6 Model analysis using the surrogate

Apart from predicting new inputs with good accuracy and a significant speedup, the surrogate model can be used to perform a variety of different tasks related to metamodeling, such as uncertainty quantification and state estimation. This can be done in a non-intrusive way, meaning that the full model is employed as a black-box that provides input–output pairs to train the surrogate, but is not required to be modified.

5 https://www.csie.ntu.edu.tw/~cjlin/libsvm/
6 https://github.com/liquidSVM/liquidSVM

In principle, any kind of analysis that requires multiple evaluations can be significantly accelerated by the use of a surrogate, including the ones that are not computationally feasible due to the high computational cost of the full model. An example is uncertainty quantification, where the expected value of f can be approximated by a Monte Carlo integration of s using a set $X_m \subset \Omega$ of integration points, i. e.,

$$\int_\Omega f(x)\,dx \approx \frac{1}{m}\sum_{i=1}^{m} s(x_i).$$

Once the surrogate is computed using a training set (X_n, Y_n), this approximate integral can be evaluated also for $m \gg n$ with a possibly very small cost, since the evaluation of s is significantly cheaper than the one of f.

Another example, which we describe in detail in the following, is the solution of an inverse problem to estimate the input parameter which generated a given output, i. e., from a given vector $\overline{y} \in \mathbb{R}^q$ we want to estimate $x \in \Omega$ such that $f(\overline{x}) = \overline{y}$. This can be done by considering a state-estimation cost function $C : \Omega \to \mathbb{R}$ defined by

$$C(x) := \frac{1}{2\|\overline{y}\|_2^2}\|s(x) - \overline{y}\|_2^2, \tag{9.25}$$

and estimating $\overline{x}$ by the value x^* defined as

$$x^* := \min_{x\in\Omega} C(x).$$

In principle, we could perform the same optimization also using f instead of s in (9.25), but the surrogate allows a rapid evaluation of C. Moreover, if K is at least differentiable, then also s is differentiable, and thus we can use gradient-based methods to minimize C.

To detail this approach, we assume $f : \Omega \to \mathbb{R}^q$ and to have a surrogate obtained as in Section 9.4.1 with the separable matrix-valued kernel of Section 9.2.4, i. e., from (9.10) we have

$$s_N(x) = \sum_{k\in I_N} \alpha_k K(x, x_k).$$

As explained in (9.9), in the vector-valued case $q > 1$ we always assume that the output $s_N(x)$ and the coefficients α_k are row vectors, and in particular $\alpha \in \mathbb{R}^{N\times q}$ and $s_N(x) \in \mathbb{R}^{1\times q}$. In this case we have the following.

Proposition 9.8 (Gradient of the state-estimation cost). *For $x \in \Omega \subset \mathbb{R}^d$ and $\overline{y} \in \mathbb{R}^q$, the gradient of the cost* (9.25) *can be computed in $x \in \Omega$ as*

$$\nabla C(x) = \frac{1}{\|\overline{y}\|_2^2}(D\alpha)E^T,$$

where $D \in \mathbb{R}^{d\times N}$ with columns $D_j := \nabla_x K(x, x_j)$, and $E := s_N(x) - \overline{y} \in \mathbb{R}^{1\times q}$.

Proof. By linearity, the gradient of s_N in x can be computed as

$$\nabla s_N(x) = \sum_{j=1}^{n} \alpha_j \nabla_x K(x, x_j) = D\alpha \in \mathbb{R}^{d\times q},$$

and thus

$$\begin{aligned}\nabla C(x) &= \frac{1}{\|\bar{y}\|_2^2}\big(s_N(x) - \bar{y}\big)\nabla_x\big(s_N(x) - \bar{y}\big) = \frac{1}{\|\bar{y}\|_2^2}\big(s_N(x) - \bar{y}\big)\nabla s(x) \\ &= \frac{1}{\|\bar{y}\|_2^2}(D\alpha)E^T.\end{aligned}$$

□

Observe in particular that whenever K is known in closed form the matrix D can be explicitly computed, and thus the gradient can be assembled using only matrix-vector multiplications of matrices of dimensions N, d, q, but independent of n. The solution x^* can then be computed by any gradient-based optimization method, and each iteration can be performed in an efficient way.

9.7 Parameter and model selection

For all the methods that we have seen the approximation quality of the surrogate depends on several parameters, which need to be carefully chosen to obtain good results. There are both parameters defining the kernel, such as the shape parameter $\gamma > 0$ in a RBF kernel, and model parameters such as the regularization parameter $\lambda \geq 0$. To some extent, also the selection of the kernel itself can be considered as a parametric dependence of the model. Moreover, it is essential to test the quality of the surrogate on an independent test set of data, since tuning it on the training set alone can very likely lead to overfitting, i. e., to obtain a model that is excessively accurate on the training set, while failing to generalize its prediction capabilities to unseen data.

In practical applications the target function f is unknown, so it cannot be used to check if the approximation is good, and all we know is the training set (X_n, Y_n). In this case the most common approach is to split the sets into train, validation and test sets in the following sense. We permute (X_n, Y_n), fix numbers n_{tr}, n_{val}, n_{te} such that $n = n_{\text{tr}} + n_{\text{val}} + n_{\text{te}}$, and define a partition of the dataset as

$$\begin{aligned}X_{\text{tr}} &:= \{x_i, 1 \leq i \leq n_{\text{tr}}\}, \\ X_{\text{val}} &:= \{x_i, n_{\text{tr}} + 1 \leq i \leq n_{\text{tr}} + n_{\text{val}}\}, \\ X_{\text{te}} &:= \{x_i, n_{\text{tr}} + n_{\text{val}} + 1 \leq i \leq n\},\end{aligned}$$

and similarly for Y_{tr}, Y_{val}, Y_{te}.

The idea is then to use the validation set $(X_{\text{val}}, Y_{\text{val}})$ to validate (i. e., choose) the parameters, and the test set $(X_{\text{te}}, Y_{\text{te}})$ to evaluate the error. Having disjoint sets allows one to have a fair way to test the algorithm.

For the process we also need an error function that returns the error of the surrogate s evaluated on a generic set of points $X := \{x_i\}_i \subset \Omega$ w. r. t. the exact values $Y := \{y_i\}_i$. We denote by $|X|$ the number of elements of X. Common examples are the maximal error and the Root Mean Square Error (RMSE) defined as

$$E(s, X, Y) := \max_{1 \le i \le |X|} \|s(x_i) - y_i\|_2 \quad \text{or} \quad E(s, X, Y) := \sqrt{\frac{1}{|X|} \sum_{i=1}^{|X|} \|s(x_i) - y_i\|_2^2}. \tag{9.26}$$

Then one chooses a set of possible parameter instantiations $\{p_1, \dots, p_{n_p}\}$, $n_p \in \mathbb{N}$ that has to be checked. A common choice for positive numerical parameters is to take them logarithmically equally spaced, since the correct scale is not known in advance, in general.

The training and validation process is described in Algorithm 9.7, where we denote by $s(p_i)$ the surrogate obtained with parameter p_i. It works as an outer loop with respect to the training of any of the surrogates that we have considered, and it has thus to be understood as part of the offline phase.

Algorithm 9.7: Model selection by validation.

1: Input: $X_{\text{tr}}, X_{\text{val}}, X_{\text{te}}, Y_{\text{tr}}, Y_{\text{val}}, Y_{\text{te}}, \{p_1, \dots, p_{n_p}\}$
2: **for** $i = 1, \dots, n_p$ **do**
3: Train surrogate $s(p_i)$ with data $(X_{\text{tr}}, Y_{\text{tr}})$
4: Compute validation error $e_i := E(s(p_i), X_{\text{val}}, Y_{\text{val}})$
5: **end for**
6: Choose parameter $\bar{p} := p_i$ with $i := \operatorname{argmin} e_i$
7: Train surrogate $s(\bar{p})$ with data $(X_{\text{tr}} \cup X_{\text{val}}, Y_{\text{tr}} \cup Y_{\text{val}})$
8: Compute test error $\bar{E} = E(s(\bar{p}), X_{\text{te}}, Y_{\text{te}})$
9: Output: surrogate $s(\bar{p})$, optimal parameter $\bar{p}$, test error $\bar{E}$

A more advanced way to realize the same idea is via k-fold cross validation. To have an even better selection of the parameters, one can repeat the validation step (lines 2–6 in the previous algorithm) by changing the validation set at each step. To do so we do not select a validation set (so $n = n_{\text{tr}} + n_{\text{te}}$), and instead consider a partition of $X_{\text{tr}}, Y_{\text{tr}}$ into a fixed number $k \in \{1, \dots, n_{\text{tr}}\}$ of disjoint subsets, all approximately of the same size, i. e.,

$$X_{\text{tr}} := \{x_i, 1 \le i \le n_{\text{tr}}\} := \cup_{i=1}^{k} X_i$$
$$X_{\text{te}} := \{x_i, n_{\text{tr}} + 1 \le i \le n\},$$

and similarly for $Y_{\text{tr}} := \cup_{i=1}^{k} Y_i$ and for Y_{te}. In the validation step each of the X_i is used as a validation set, and the validation is repeated for all $i = 1, \dots, k$. In this case the error e_i for the parameter p_i is defined as the average error over all these permutations, as described in Algorithm 9.8.

Algorithm 9.8: Model selection by k-fold cross validation.

Input: $X_{\text{tr}} = \cup_{i=1}^{k} X_i, X_{\text{te}}$, $Y_{\text{tr}} = \cup_{i=1}^{k} Y_i, Y_{\text{te}}$, $\{p_1, \dots, p_{n_p}\}$
for $i = 1, \dots, n_p$ **do**
    for $j = 1, \dots, k$ **do**
        Train surrogate $s(p_i)$ with data $(\cup_{\ell \neq j} X_\ell, \cup_{\ell \neq j} Y_\ell)$
        Compute error $e^{(j)} := E(s(p_i), X_j, Y_j)$
    end for
    $e_i := \text{mean}\{e^{(j)}, 1 \leq j \leq k\}$
end for
Choose parameter $\bar{p} := p_i$ with $i := \text{argmin}\, e_i$
Train surrogate $s(\bar{p})$ with data $(X_{\text{tr}}, Y_{\text{tr}})$
Compute test error $\bar{E} = E(s(\bar{p}), X_{\text{te}}, Y_{\text{te}})$
Output: surrogate $s(\bar{p})$, optimal parameter $\bar{p}$, test error $\bar{E}$

We remark that, in the extreme case $k = N$, this k-fold cross validation is usually called Leave One Out Cross Validation (LOOCV).

9.8 Numerical examples

For the testing and illustration of the two methods of Section 9.4 and Section 9.5, we consider a real-world application dataset describing the biomechanical modeling of the human spine introduced and studied in [69]. We refer to that paper for further details and we just give a brief description in the following.

The input–output function $f : \mathbb{R}^3 \to \mathbb{R}^3$ represents the coupling between a global multibody system (MBS) and a Finite Elements (FEM) submodel. The human spine is represented as a MBS consisting of the vertebra, which are coupled by the interaction through intervertebral disks (IVDs). The PDE representing the behavior of each IVD is approximated by a FEM discretization, and it has the input geometry parameters as boundary conditions, and computes the output mechanical response as a result of the simulation. In particular, the three inputs are two spatial displacements and an angular inclination of a vertebra, and the three outputs are the corresponding two force components and the momentum which are transferred to the next vertebra. The dataset is generated by running the full model for $n := 1370$ different input parameters X_n and generating the corresponding set of outputs Y_n.

The dataset, as described in Section 9.7, is first randomly permuted and then divided in training and test datasets $(X_{n_{\text{tr}}}, Y_{n_{\text{tr}}})$, $(X_{n_{\text{te}}}, Y_{n_{\text{te}}})$ with $n_{\text{tr}} := 1238$ and $n_{\text{te}} = 132$, corresponding to roughly 90 % and 10 % of the data. We remark that the full model predicts a value $(0, 0, 0)^T$ for the input $(0, 0, 0)^T$ and this sample pair is present in the dataset. We thus manually include it in the training set independently of the permutation. The training and test sets can be seen in Figure 9.2.

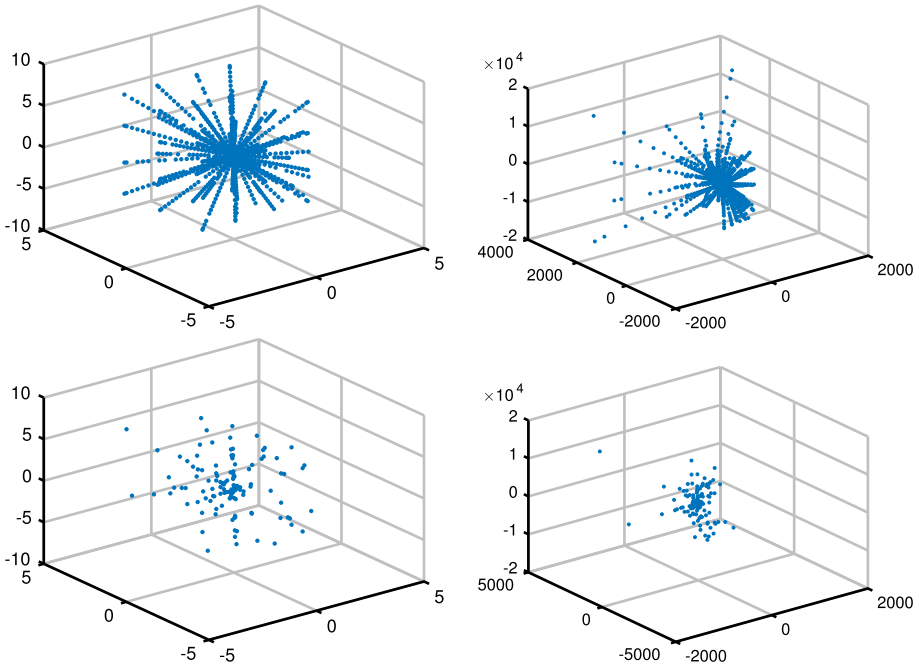

Figure 9.2: Input parameters (left) and corresponding outputs (right) for the training (top row) and test set (bottom row).

The models are trained using a Matlab implementation of the algorithms. For VKOGA we use an own implementation,[7] while for SVR we employ the KerMor package,[8] which provides an implementation of the 2-index SMO for the SVR without offset that is discussed in Section 9.5.1. We remark that this implementation requires the output data to be scaled in $[-1, 1]$, and thus we perform this scaling for the training and validation, while the testing is executed by scaling back the predictions to the original range. To have a fair comparison, we use the same data normalization also for the VKOGA models.

7 https://gitlab.mathematik.uni-stuttgart.de/pub/ians-anm/vkoga

8 https://www.morepas.org/software/kermor/index.html

The regularized VKOGA (with f-, P-, and f/P-greedy selection rules) and the SVR models are trained with the Gaussian kernel. Both algorithms depend on the shape parameter γ of the kernel and on the regularization parameter λ, while SVR additionally depends on the width ε of the tube. These parameters are selected by k-fold cross validation as described in Section 9.7. The values of k and of the parameter samples used for validation are reported in Table 9.1, where each parameter set is obtained by generating logarithmically equally spaced samples in the given interval, i. e., 400 parameter pairs are tested for VKOGA and 4000 triples for SVR. As an error measure we use the max error in (9.26). We remark that the SVR surrogate is obtained by training a separate model for each output, as described in Section 9.5, but only one cross validation is used. This means that for each parameter triple three models are trained, and then the parameter is evaluated in the prediction of the three-dimensional output.

Table 9.1: Parameters ranges and sample numbers used in the k-fold cross validation.

k	$\gamma_{\min}$	$\gamma_{\max}$	n_γ	$\lambda_{\min}$	$\lambda_{\max}$	n_λ	$\varepsilon_{\min}$	$\varepsilon_{\max}$	n_ε
5	10^{-2}	10^{1}	20	10^{-16}	10^{3}	20	10^{-10}	10^{-3}	10

Moreover, the training of the VKOGA surrogates is terminated when the square of the power function is below the tolerance $\tau_P := 10^{-12}$, or when the training error is below the tolerance $\tau_f := 10^{-6}$. Additionally, it would be possible to use a maximal number of selected points as stopping criterion, and this offers the significant advantage of directly controlling the expansion size, which could be reduced to any given number (of course at the price of a reduced accuracy). In the case of SVR, instead, the number N is a result of the tuning of the remaining parameters.

In Table 9.2 we report the values of the parameters selected by the validation procedure for the four models, as well as the number N of nonzero coefficients in the trained kernel expansions. Observe that for SVR the three values of N refer to the number of support vectors for the three scalar-valued models. Moreover, the number of support vectors or kernel centers is only slightly larger for SVR than for the VKOGA models, but, as discussed in the following, the VKOGA models give prediction errors which are up to two orders of magnitude smaller than the ones of the SVR model.

We can now test the four models in the prediction on the test set. Table 9.3 contains various error measures between the prediction of the surrogates and the exact data. We report the values of the maximum error $E_{\max}$ and the RMSE E_{RMSE} defined in (9.26), and the relative maximum error $E_{\max,\mathrm{rel}}$ obtained by scaling each error by the norm of the exact output.

To provide a better insight in the approximation quality of the methods, we show in Figure 9.3 the distribution of the error over the test set. The plots show, for each sam-

Table 9.2: Selected parameters and number of nonzero coefficients in the kernel expansions.

Method	N	$\bar{\gamma}$	$\bar{\lambda}$	$\bar{\varepsilon}$
VKOGA P-greedy	1000	$4.9 \cdot 10^{-2}$	10^{-11}	–
VKOGA f-greedy	879	$4.3 \cdot 10^{-2}$	10^{-11}	–
VKOGA f/P-greedy	967	$6.2 \cdot 10^{-2}$	10^{-9}	–
SVR, output 1	359	$1.8 \cdot 10^{-1}$	10^{2}	$7.7 \cdot 10^{-7}$
output 2	378			
output 3	405			

Table 9.3: Test errors: maximum error $E_{\max}$, RMSE error E_{RMSE}, maximum relative error $E_{\max,\mathrm{rel}}$.

Method	$E_{\max}$	E_{RMSE}	$E_{\max,\mathrm{rel}}$
VKOGA P-greedy	$1.6 \cdot 10^{2}$	22.3	$2.2 \cdot 10^{-1}$
VKOGA f-greedy	$1.6 \cdot 10^{2}$	22.4	$2.0 \cdot 10^{-1}$
VKOGA f/P-greedy	$1.6 \cdot 10^{2}$	23.2	$8.8 \cdot 10^{-1}$
SVR	$1.3 \cdot 10^{3}$	$1.6 \cdot 10^{2}$	$1.4 \cdot 10^{1}$

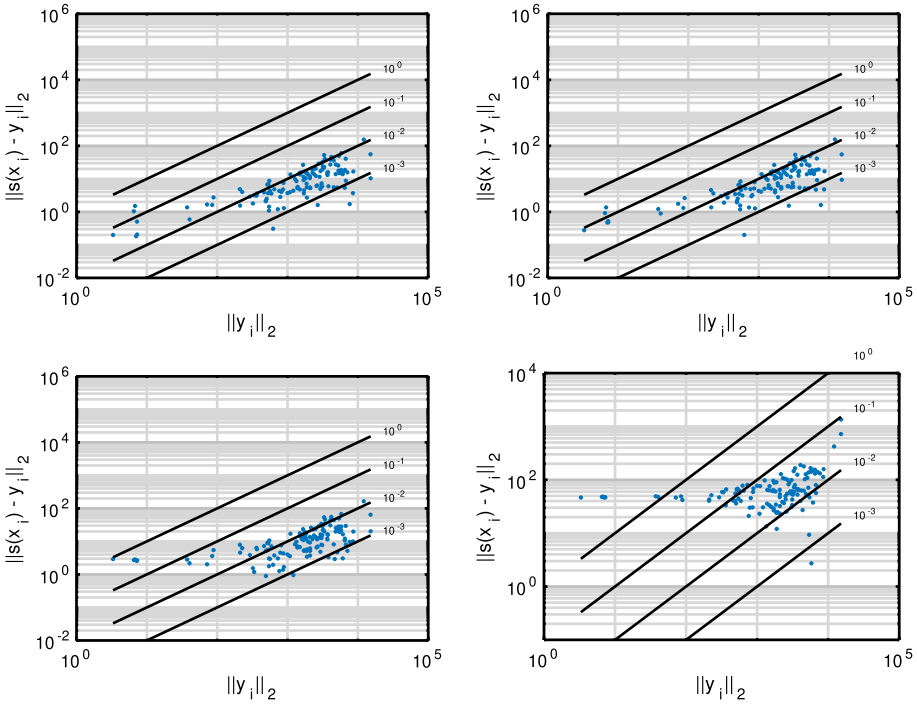

Figure 9.3: Absolute errors as functions of the magnitude of the output, and relative error levels from 10^{0} to 10^{-3} for the surrogates obtained with P-greedy VKOGA (top left), f-greedy VKOGA (top right), f/P-greedy VKOGA (bottom left) and SVR (bottom right).

ple (x_i, y_i) in the test set, the absolute error $\|y_i - s(x_i)\|_2$ as a function of the magnitude $\|y_i\|_2$ of the output. Moreover, the black lines represent a relative error from 10^0 to 10^{-3}. It is clear that in all cases the maximum and RMS errors of Table 9.3 are dominated by the values obtained for outputs of large norm, where the VKOGA models obtain a much better accuracy than SVR. The relative errors, on the other hand, are not evenly distributed for SVR, where most of the test set is approximated with a relative error between 10^1 and 10^{-2} except for the samples with small magnitude of the output. For these data, the model gives increasingly bad predictions as the magnitude is smaller, reaching a relative error much larger than 1. The VKOGA models, instead, obtain a relative error smaller than 10^{-2} on the full test set except for the entries of small magnitude. For these samples, the f- and P-greedy versions of the algorithm perform almost the same and better than the f/P-greedy variant, thus giving an overall smaller relative error in Table 9.3. Moreover, these results are obtained with a significantly smaller expansion size for f-greedy than for P-greedy. Indeed, even if the SVR surrogates for the individual output components are smaller than the VKOGA ones, the overall number of nonzero coefficients is $359 + 378 + 405 = 1142$, i. e., more than the one of each of the three VKOGA models, thus leading to a less accurate and more expensive surrogate.

Regarding the runtime requirements, we can now estimate both the offline (training) and the online (prediction) times. The offline time required for the validation and training of the models is essentially determined by the number of parameters tested in the k-fold cross validation, while the training time of a single model is almost negligible. As a comparison, we report in Table 9.4 the average runtime $\tilde{T}_{offline}$ for 10 runs of the training of the models for the fixed set of parameters of Table 9.2. All the reported times are in the ranges of seconds (for VKOGA) and below one minute (for SVR). We remark that this timing is only a very rough indication and not a precise comparison, since the times highly depends on the number of selected points (for VKOGA) and the number of support vectors for SVR, and both are dependent on the used parameters. For example, we repeated the experiment for SVR with the same parameter set but with $\varepsilon = 10^{-1}$. In this case this value of ε is overly large (if compared to the one selected by cross validation) and it likely produces a useless model, but nevertheless we obtain an average training time of 0.03 sec.

Table 9.4: Average offline time (training only), online time, and projected speedup factor for the four different models.

Method	N	$\tilde{T}_{\text{offline}}$	$\tilde{T}_{\text{online}}$	$\tilde{T}_{\text{full}}/\tilde{T}_{\text{online}}$
VKOGA P-greedy	1000	1.67 sec	$9.97 \cdot 10^{-6}$ sec	$3.01 \cdot 10^5$
VKOGA f-greedy	879	1.41 sec	$9.44 \cdot 10^{-6}$ sec	$3.18 \cdot 10^5$
VKOGA f/P-greedy	967	1.66 sec	$9.92 \cdot 10^{-6}$ sec	$3.02 \cdot 10^5$
SVR (3 models)	1142	52.0 sec	$2.28 \cdot 10^{-5}$ sec	$1.32 \cdot 10^5$

A more interesting comparison is the online time, which directly determines the efficiency of the surrogate models in the replacement of the full simulation. In this case, we evaluate the models 5000 times on the full test set consisting of $n_{te} = 132$ samples, and we report the average online time $\tilde{T}_{online}$ per single test sample in Table 9.4. The table contains also again the number N of elements of the corresponding kernel expansions, and it is evident that a smaller value leads to a faster evaluation of the model.

In the original paper [69], it has been estimated that a 30 sec full simulation with 24 IVDs with a timestep $\Delta t = 10^{-3}$ sec requires $7.2 \cdot 10^5$ evaluations of the coupling function f, and these were estimated to require 600 h. This corresponds to an average of $\tilde{T}_{\text{full}} = 3$ sec per evaluation of f, giving a speedup $\tilde{T}_{\text{full}}/\tilde{T}_{\text{online}}$ as reported in Table 9.4.

These surrogates can now be employed to solve different tasks that require multiple evaluations of f. As an example, we employ the f-greedy model (as the most accurate and most efficient) to solve a parameter estimation problem as described in Section 9.6. We consider the output values $Y_{n_{\text{te}}}$ in the test set as a set of measures that have not been used in the training of the model, and we try to estimate the values of $X_{n_{\text{te}}}$. For each output vector $y_i \in \mathbb{R}^3$ we define a target value $\bar{y} := y_i + \eta \|y_i\|_2 \nu$ to define the cost (9.25), where $\nu \in \mathbb{R}^3$ is a uniform random vector representing some noise, and $\eta \in [0, 1]$ is a noise level. We then use a built-in Matlab optimizer with the gradient of Proposition 9.8, with initial guess $x_0 := 0 \in \mathbb{R}^3$, to obtain an estimate x_i^* of x_i. The results of the estimate for each output value in the test set are depicted in Figure 9.4 for $\eta = 0, 0.1$, where we report also the final value of the cost function $C(x_i^*)$. In all cases, the optimizer seems to converge, since the value of the cost function is in all cases smaller than 10^{-4}, which represents a relative value smaller than 10^{-3} with respect to the magnitude of the input values. The maximum absolute error in the estimations is quite uniform for all the samples in the test set, and this results in a good relative error of about 10^{-1} for large inputs, while for inputs of very small magnitude the relative error is larger than 1, and a larger noise level leads to less accurate predictions. This behavior is coherent with the analysis of the test error discussed above, since the approximant is less accurate on inputs of small magnitude, and thus it provides a less reliable surrogate in the cost function.

9.9 Conclusions and outlook

In this chapter we discussed the use of kernel methods to construct surrogate models based on scattered data samples. These methods can be applied to data with general structure, and they scale well with the dimension of the input and output values. In particular, we analyzed issues and methods to obtain sparse solutions, which are then extremely fast to evaluate, while still being very accurate. These properties have been further demonstrated on numerical tests on a real application dataset. These

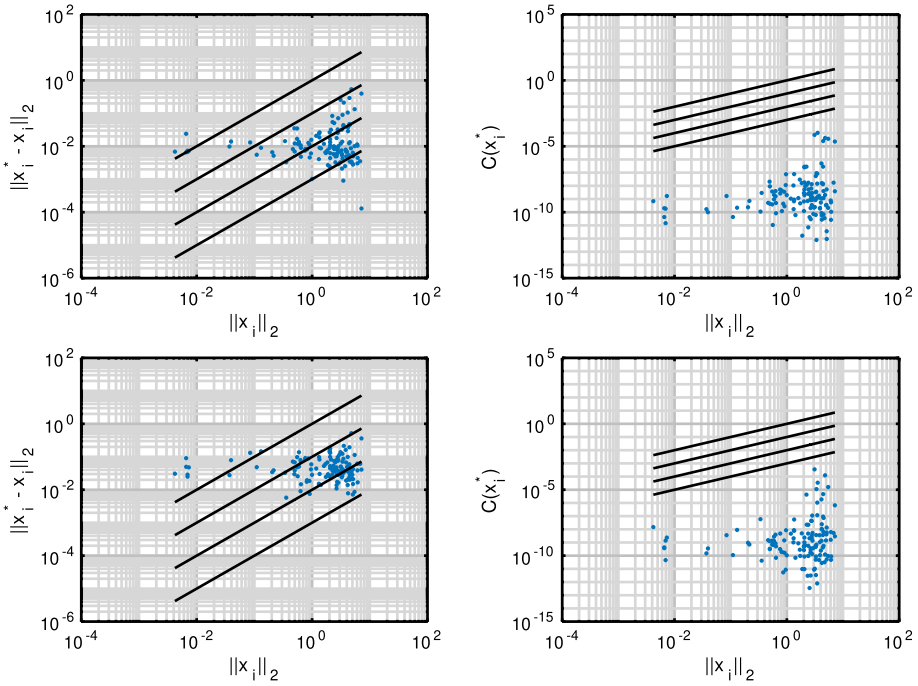

Figure 9.4: Absolute errors of the input estimation as functions of the magnitude of the output (left), and value of the cost function at the estimated input (right) for a noise level $\eta = 0$ (top row) and $\eta = 0.1$ (bottom row) using the f-greedy VKOGA model. The dotted lines represent relative error levels from 10^0 to 10^{-3}.

methods can be analyzed in the common framework of Reproducing Kernel Hilbert Spaces, which provides solid theoretical foundations and a high flexibility to derive new algorithms.

The integration of machine learning and model reduction is promising and many interesting aspects have still to be investigated. For example, surrogate models have been used in [23, 24] to learn a representation with respect to projection-based methods, and generally a more extensive application of machine learning to dynamical systems requires additional understanding and the derivation of new techniques. Moreover, the field of data-based numerics is very promising, where classical numerical methods are integrated or accelerated with data-based models.

Bibliography

[1] M. Alvarez, L. Rosasco, and N. D. Lawrence. Kernels for vector-valued functions: a review. *Found. Trends Mach. Learn.*, 4(3):195–266, 2012.

[2] N. Aronszajn. Theory of reproducing kernels. *Trans. Am. Math. Soc.*, 68:337–404, 1950.

[3] A. Beckert and H. Wendland. Multivariate interpolation for fluid-structure-interaction problems using radial basis functions. *Aerosp. Sci. Technol.*, 5(2):125–134, 2001.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92*, pages 144–152. ACM, New York, NY, USA, 1992.

[5] J. Bouvrie and B. Hamzi. Kernel methods for the approximation of nonlinear systems. *SIAM J. Control Optim.*, 55(4):2460–2492, 2017.

[6] T. Brünnette, G. Santin, and B. Haasdonk. Greedy kernel methods for accelerating implicit integrators for parametric ODEs. In F. A. Radu, K. Kumar, I. Berre, J. M. Nordbotten and I. S. Pop, editors, *Numerical Mathematics and Advanced Applications - ENUMATH 2017*, pages 889–896. Springer, Cham, 2019.

[7] R. Cavoretto, A. De Rossi, and E. Perracchione. Efficient computation of partition of unity interpolants through a block-based searching technique. *Comput. Math. Appl.*, 71(12):2568–2584, 2016.

[8] R. Cavoretto, G. Fasshauer, and M. McCourt. An introduction to the Hilbert-Schmidt SVD using iterated Brownian bridge kernels. *Numer. Algorithms*, 68(2):1–30, 2014.

[9] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[10] S. Chen, D. Donoho, and M. Saunders. Atomic decomposition by basis pursuit. *SIAM J. Sci. Comput.*, 20(1):33–61, 1998.

[11] W. Chen, Z.-J. Fu, and C.-S. Chen. *Recent advances in radial basis function collocation methods*. Springer, 2014.

[12] S. Deparis, D. Forti, and A. Quarteroni. A rescaled localized Radial Basis Function interpolation on non-Cartesian and nonconforming grids. *SIAM J. Sci. Comput.*, 36(6):A2745–A2762, 2014.

[13] S. Deparis, D. Forti, and A. Quarteroni. *A Fluid–Structure Interaction Algorithm Using Radial Basis Function Interpolation Between Non-Conforming Interfaces*, pages 439–450. Springer, Cham, 2016.

[14] M. Drohmann and K. Carlberg. The ROMES method for statistical modeling of Reduced-Order-Model error. *SIAM/ASA J. Uncertain. Quantificat.*, 3(1):116–145, 2015.

[15] G. E. Fasshauer and M. McCourt. *Kernel-Based Approximation Methods Using MATLAB*. *Interdisciplinary Mathematical Sciences*, volume 19. World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2015.

[16] G. E. Fasshauer and M. J. McCourt. Stable evaluation of Gaussian radial basis function interpolants. *SIAM J. Sci. Comput.*, 34(2):A737–A762, 2012.

[17] B. Fornberg and N. Flyer. *A primer on radial basis functions with applications to the geosciences*. SIAM, 2015.

[18] B. Fornberg, E. Larsson, and N. Flyer. Stable computations with Gaussian radial basis functions. *SIAM J. Sci. Comput.*, 33(2):869–892, 2011.

[19] J. Garcke and M. Griebel. *Sparse grids and applications*, volume 88. Springer, 2012.

[20] T. Gärtner, J. W. Lloyd, and P. A. Flach. Kernels for structured data. In S. Matwin and C. Sammut, editors, *Inductive Logic Programming*, pages 66–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[22] S. Grundel, N. Hornung, B. Klaassen, P. Benner, and T. Clees. *Computing Surrogates for Gas Network Simulation Using Model Order Reduction*, pages 189–212. Springer New York, New York, NY, 2013.

[23] M. Guo and J. S. Hesthaven. Reduced order modeling for nonlinear structural analysis using Gaussian process regression. *Comput. Methods Appl. Mech. Eng.*, 341:807–826, 2018.

[24] M. Guo and J. S. Hesthaven. Data-driven reduced order modeling for time-dependent problems. *Comput. Methods Appl. Mech. Eng.*, 345:75–99, 2019.
[25] B. Haasdonk and G. Santin. Greedy kernel approximation for sparse surrogate modeling. In W. Keiper, A. Milde and S. Volkwein, editors, *Reduced-Order Modeling (ROM) for Simulation and Optimization: Powerful Algorithms as Key Enablers for Scientific Computing*, pages 21–45. Springer, Cham, 2018.
[26] D. Haussler. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, UC Santa Cruz, 1999.
[27] G. S. Kimeldorf and G. Wahba. A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *Ann. Math. Stat.*, 41(2):495–502, 1970.
[28] M. Köppel, F. Franzelin, I. Kröker, S. Oladyshkin, G. Santin, D. Wittwar, A. Barth, B. Haasdonk, W. Nowak, D. Pflüger, and C. Rohde. Comparison of data-driven uncertainty quantification methods for a carbon dioxide storage benchmark scenario. *Comput. Geosci.*, 23(2):339–354, 2019.
[29] T. Köppl, G. Santin, B. Haasdonk, and R. Helmig. Numerical modelling of a peripheral arterial stenosis using dimensionally reduced models and kernel methods. *Int. J. Numer. Methods Biomed. Eng.*, 34(8):e3095, 2018. cnm.3095.
[30] M. Kowalewski, E. Larsson, and A. Heryudono. An adaptive interpolation scheme for molecular potential energy surfaces. *J. Chem. Phys.*, 145(8):084104, 2016.
[31] E. Larsson, E. Lehto, A. Heryudono, and B. Fornberg. Stable computation of differentiation matrices and scattered node stencils based on Gaussian radial basis functions. *SIAM J. Sci. Comput.*, 35(4):A2096–A2119, 2013.
[32] A. Manzoni and F. Negri. Heuristic strategies for the approximation of stability factors in quadratically nonlinear parametrized PDEs. *Adv. Comput. Math.*, 41(5):1255–1288, 2015.
[33] E. Marchandise, C. Piret, and J.-F. Remacle. CAD and mesh repair with Radial Basis Functions. *J. Comput. Phys.*, 231(5):2376–2387, 2012.
[34] I. Martini. Reduced Basis Approximation for Heterogeneous Domain Decomposition Problems. PhD thesis, IANS, University of Stuttgart 2017.
[35] C. A. Micchelli and M. Pontil. On learning vector-valued functions. *Neural Comput.*, 17(1):177–204, 2005.
[36] S. Müller. Komplexität und Stabilität von kernbasierten Rekonstruktionsmethoden (Complexity and Stability of Kernel-based Reconstructions). PhD thesis, Fakultät für Mathematik und Informatik, Georg-August-Universität Göttingen 2009.
[37] S. Müller and R. Schaback. A Newton basis for kernel spaces. *J. Approx. Theory*, 161(2):645–655, 2009.
[38] R. A. Olea. *Geostatistics for engineers and earth scientists*. Springer, 2012.
[39] M. Pazouki and R. Schaback. Bases for kernel-based spaces. *J. Comput. Appl. Math.*, 236(4):575–588, 2011.
[40] B. Peherstorfer and Y. Marzouk. A transport-based multifidelity preconditioner for Markov chain Monte Carlo. *Adv. Comput. Math.*, 45(5):2321–2348, 2019.
[41] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, April 1998.
[42] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
[43] C. Rieger and B. Zwicknagl. Sampling inequalities for infinitely smooth functions, with applications to interpolation and machine learning. *Adv. Comput. Math.*, 32(1):103–129, 2008.
[44] C. Rieger and B. Zwicknagl. Deterministic error analysis of support vector regression and related regularized kernel methods. *J. Mach. Learn. Res.*, 10:2115–2132, 2009.
[45] S. Saitoh and Y. Sawano. *Theory of Reproducing Kernels and Applications*. *Developments in Mathematics*, volume 44. Springer, Singapore, 2016.

[46] G. Santin and B. Haasdonk. Convergence rate of the data-independent P-greedy algorithm in kernel-based approximation. *Dolomites Res. Notes Approx.*, 10:68–78, 2017.
[47] G. Santin, D. Wittwar, and B. Haasdonk. Greedy regularized kernel interpolation/ University of Stuttgart, 2018. ArXiv preprint 1807.09575.
[48] R. Schaback. Error estimates and condition numbers for radial basis function interpolation. *Adv. Comput. Math.*, 3(3):251–264, 1995.
[49] R. Schaback and H. Wendland. Approximation by positive definite kernels. In M. Buhmann and D. Mache, editors, *Advanced Problems in Constructive Approximation. International Series in Numerical Mathematics*, volume 142, pages 203–221. 2002.
[50] M. Scheuerer, R. Schaback, and M. Schlather. Interpolation of spatial data – a stochastic or a deterministic problem? *Eur. J. Appl. Math.*, 24(4):601-–629, 2013.
[51] A. Schmidt and B. Haasdonk. Data-driven surrogates of value functions and applications to feedback control for dynamical systems. *IFAC-PapersOnLine*, 51(2):307–312, 2018. 9th Vienna International Conference on Mathematical Modelling.
[52] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In D. Helmbold and B. Williamson, editors, *Computational Learning Theory*, pages 416–426. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.
[53] B. Schölkopf and A. Smola. *Learning with Kernels*. The MIT Press, 2002.
[54] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
[55] I. Steinwart and A. Christmann. *Support Vector Machines*. Springer, 2008.
[56] I. Steinwart, D. Hush, and C. Scovel. An explicit description of the Reproducing Kernel Hilbert Spaces of Gaussian RBF kernels. *IEEE Trans. Inf. Theory*, 52(10):4635–4643, 2006.
[57] I. Steinwart, D. Hush, and C. Scovel. Training SVMs Without Offset. *J. Mach. Learn. Res.*, 12:141–202, 2011.
[58] I. Steinwart and P. Thomann. liquidSVM: A fast and versatile SVM package, 2017. arXiv:1702.06899.
[59] J. Suykens, J. Vanderwalle, and B. D. Moor. Optimal control by least squares support vector machines. *Neural Netw.*, 14:23–35, 2001.
[60] T. Taddei, J. D. Penn, M. Yano, and A. T. Patera. Simulation-based classification; a model-order-reduction approach for structural health monitoring. *Arch. Comput. Methods Eng.*, 1–23, 2016.
[61] R. Tibshirani. Regression shrinkage and selection via the LASSO. *J. R. Stat. Soc. B*, 58(1):267–288, 1996.
[62] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998.
[63] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Adv. Comput. Math.*, 4(1):389–396, 1995.
[64] H. Wendland. Fast evaluation of radial basis functions: methods based on partition of unity. In *Approximation theory, X (St. Louis, MO, 2001). Innov. Appl. Math.*, pages 473–483 Vanderbilt Univ. Press, Nashville, TN, 2002.
[65] H. Wendland. *Scattered Data Approximation. Cambridge Monographs on Applied and Computational Mathematics*, volume 17. Cambridge University Press, Cambridge, 2005.
[66] H. Wendland and C. Rieger. Approximate interpolation with applications to selecting smoothing parameters. *Numer. Math.*, 101(4):729–748, 2005.
[67] D. Wirtz and B. Haasdonk. A-posteriori error estimation for parameterized kernel-based systems. In *Proc. MATHMOD 2012 - 7th Vienna International Conference on Mathematical Modelling*, 2012.
[68] D. Wirtz and B. Haasdonk. A vectorial kernel orthogonal greedy algorithm. *Dolomites Res. Notes Approx.*, 6:83–100, 2013.

[69] D. Wirtz, N. Karajan, and B. Haasdonk. Surrogate modelling of multiscale models using kernel methods. *Int. J. Numer. Methods Eng.*, 101(1):1–28, 2015.
[70] D. Wittwar, G. Santin, and B. Haasdonk. Interpolation with uncoupled separable matrix-valued kernels. *Dolomites Res. Notes Approx.*, 11:23–29, 2018.
[71] H. Zhang C and L. Zhao. On the inclusion relation of reproducing kernel Hilbert spaces. *Anal. Appl.*, 11, 2013.