Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Telecommunications Engineering
(XXXV cycle)

Machine Learning based Surrogate Modeling of Electronic Devices and Circuits

…

Nastaran Soleimani
* * * * *

Supervisors
Prof. Riccardo Trinchero, Prof. Flavio Canavero

Politecnico di Torino
May, 2023

. . . .

Nastaran Soleimani

Turin, May, 2023

List of Figures

List of Tables

List of Acronyms

UQ: Uncertainty Quantification
PDF: Probability Density Function
MC: Monte Carlo
ML: Machine Learning
PKBML: Prior Knowledge-Based Machine Learning
P.U.L.: Per-Unit-Length
KRR: Kernel Ridge Regression
RKHS: Reproducing Kernel Hilbert Space
ERM: Empirical Risk Minimization
LHS: Latin Hypercube Sampling
OLS: Ordinary Least Squares
MSE: Mean Square Error
CV: Cross-Validation
LOO: Leave-One-Out
ADAM: Adaptive Moment Estimation
SVM: Support Vector Machine
RBF: Radial Basis Function
LHS: Latin Hypercube Sampling
CV: Cross-Validation
NRMS: Normalized Root Mean Square
SD: Source Difference
PKI: Prior Knowledge Input
MTL: Multi-Conductor Transmission Line
FEM: Finite Element Method
MoM: Method of Moments
SI: Signal Integrity
CVCF: Complex Valued Complex Function
PCF: Pseudo Complex-Valued Function
DCK: Dual Channel Kernel
PCA: Principal Component Analysis
SVD: Singular Value Decomposition
PCB: Printed Circuit Board

NRMSE: Normalized Root Mean Square Error
GD: Gradient Descent
LNA: Low-Noise Amplifier

Contents

Dedicated to

My Lovely Mom

Acknowledgements

Fist of all, I would like to express my special thanks to my supervisor Prof. Flavio Canavero, who gave me the opportunity to come to Italy and study at Politecnico di Torino and supporting me during my whole Ph.D.

The completion of this study could not have been possible without my other supervisor Prof. Riccardo Trinchero, who gave me a lot of valuable help during this journey and without him it was almost impossible to I find my way.

A big debt of my life is also owed to my perfect Mom for teaching, loving and supporting me.

Last but not least, I would like to thank my sister and my brother (Nazanin and Amir Soleimani), my colleagues: Prof. Igor Simone Stievano, Felipe Treviso, Tommasso Brade, Marco De Stefano, Antonio Carlucci, Paolo Manfredi and my friends who have been kind with me and have supported me during this journey.

Summary

The availability of mathematical tools for understanding the actual relationship between the parameters and the outputs of complex dynamical system has become an important resource for the design of next-generation electrical and electronic equipment. The accurate knowledge of the relationship among inputs, outputs and parameters of the systems under analysis can be used to heavily speed up computational expensive design tasks, such as: uncertainty quantification (UQ) and optimization. Indeed, such knowledge can be used to guide the design space exploration and to reduce the number of experiments required by the above tasks.

Unfortunately, in realistic applications, the relationship between the parameters and the outputs of a generic electronic device or circuit is rather complicated and usually not explicit. For the above reason, UQ and optimization are usually performed via *brute force* approaches based on repeated experiments with the so-called computational model (i.e., physical experiments or computer simulations). However, physical experiments can be expensive and time consuming, since they would require the construction of several prototypes. Therefore, during the early design phase, parametric simulations are usually adopted.

In the above scenario, a surrogate model, also known as metamodel, allows to provide a closed-form and fast-to-evaluate approximation of the actual input-output relationship of the computational model. Among the several regression and interpolation techniques developed and used for the surrogate model construction, this dissertation mainly focuses on supervised machine learning (ML) regression techniques. Specifically, state-of-the-art ML regressions, such as: linear expansion of basis functions, artificial neural network (ANN) and kernel-machine regression are briefly presented with the aim of investigating their advantages and drawbacks for the surrogate model construction.

The above analysis shows that ANNs and kernel-machine regressions provide an effective solution for the surrogate model construction in regression problems with a "large" number of input parameters, since they allow mitigating the curse of dimensionality affecting conventional regression techniques based on linear expansion of basis functions. Also, kernel-machine regressions seem to provide the best accuracy with respect to the number of training samples, in regression problems in which *small* number of training samples is available, even if their applicability

is limited to real-valued scalar-output problem. However, for all the regression techniques considered in this dissertation, the computational cost required by the surrogate construction is dominated by the training set generation.

According to the above observations, this dissertation discusses and tries to address some relevant challenges related to the surrogate modeling construction in electronic applications.

First of all, an unconventional training scheme based on prior knowledge based machine learning approaches, in which the surrogate models are trained via a heterogeneous training set combining the predictions of a high- and low-fidelity model, is proposed with the aim of reducing the computational cost for the training set generation.

Then, a generalized mathematical framework is presented to extend the applicability of kernel-machine regressions to complex-valued problems.

Finally, data compression strategies and multi-output kernel formulations are developed with the aim of bridging the gap between ANN structures and kernel-machine regressions for the construction of vector-valued surrogate models.

The effectiveness, the strength and the performance of the above methodologies are investigated and discussed on several application examples by considering the UQ, the optimization and the parametric modeling of realistic electromagnetic structures and electronic devices.

Chapter 1

Introduction

1.1 Introduction

Understanding the link between the parameters and the responses of complex electrical and electronic systems and devices is a key aspect during the design phase. A deep knowledge of the system functioning can be used along with optimization and uncertainty quantification (UQ) tools, to optimize the product performance, to meet the design constraints and to assess the product reliability [1, 2]. Unfortunately, for realistic applications the relationship between the parameters and the outputs of the system is rather complicated and usually not explicit. Therefore, the above design tasks must be carried out by means of either physical (measurements) or computer experiments (simulations).

Computer experiments are usually preferred during the early design phase. Physical experiments can be expensive and time consuming, since they require the construction of several prototypes. As computing power increases, it has become possible to model and mimic the actual behavior of electronic circuits via sophisticated computer codes. Hence, computer experiments are now heavily adopted during the design phase [3].

Indeed, computer experiments also known as simulations, allow to synthetically explore the design space (i.e., the configurations of the input parameters) with the aim of evaluating their impact on the system performance and reliability. They rely on the so-called computational model. The computational model must provide an accurate parametric description of the actual behavior of the system under modeling, able to virtually compute, without the need of expensive prototypes, a prediction of the outputs of interest for any configuration of the system parameters.

As illustrated in Fig. 1.1, a computational model provides a forward map between the parameter space and the output of interest. Without loss of generality, the computational model can be interpreted as a function:

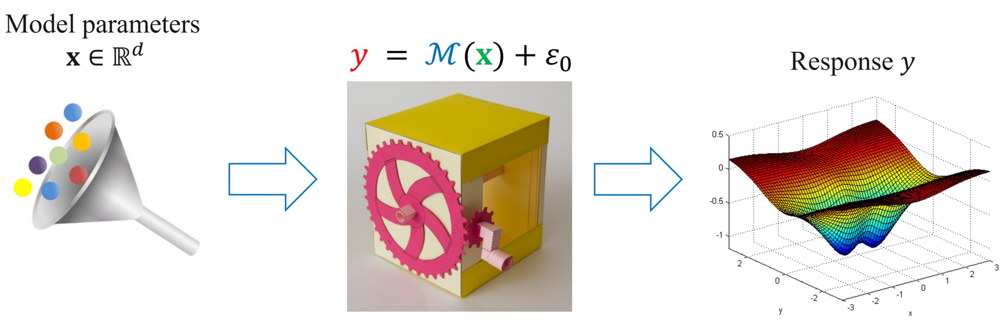
Model parameters
$\mathbf{x} \in \mathbb{R}^d$
$y = \mathcal{M}(\mathbf{x}) + \varepsilon_0$
Response y

Figure 1.1: Computational model determining the relative contribution of input variables

$$y = \mathcal{M}(\boldsymbol{x}) + \varepsilon_0, \tag{1.1}$$

such that $\mathcal{M}(\cdot) : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X} \subseteq \mathbb{R}^d$ represents the domain of d input parameters, $\mathcal{Y} \subseteq \mathbb{R}$ represents the corresponding scalar[1] output and ε_0 is the noise term corrupting the actual output values (e.g. numerical noise).

The computational model can have different levels of fidelity going from a simple closed-form analytical solution (usually available for simple devices) to the more sophisticated cases of physical-based models (e.g., the ones based on 3D full-wave solvers). It goes without saying that the model complexity heavily impacts its computational cost. A detailed physical-based computational model can be complicated to manage and analyze, thus making the design process lengthy [2, 3, 4, 5, 6].

1.1.1 Computational Model and Uncertainty Quantification

Uncertainty quantification (UQ) represents an essential task in the early-stage design of electronic components and devices, since it allows to account for the impact of process variation. In the above scenario, Monte Carlo (MC) simulation can be seen as the most straightforward way to deal with the inherently statistical nature of the problem at hand.

An illustration of the MC method is shown in Fig. 1.2. The MC sampling allows analyzing the impact of random input variables on a given set of outputs of interest by considering the results of a *large* number of parametric simulations (usually in the order of thousands), in which the value of the stochastic parameters is drawn

[1]A scalar output formulation is used for the sake of simplicity.

according to their probability density function (PDF). Such brute force approach turns out to be extremely robust and easy to implement within the simulation flow used by most of the commercial circuital and full-wave solvers, but it is also characterized by a low efficiency and by a non-negligible computational cost [7].

As an example, if a single simulation with the computational model requires 1h, the computational cost of a Monte Carlo (MC) simulation with 10k samples will be 1 month!!!

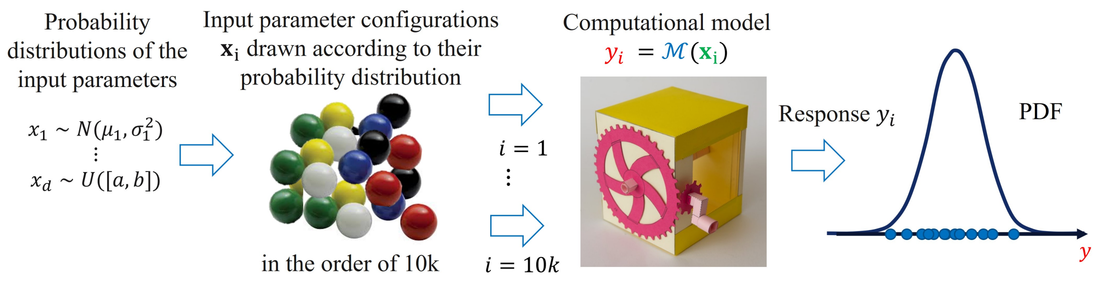
Probability distributions of the input parameters
$x_1 \sim N(\mu_1, \sigma_1^2)$
$\vdots$
$x_d \sim U([a,b])$
Input parameter configurations $\mathbf{x}_i$ drawn according to their probability distribution
in the order of 10k
$i = 1$
$\vdots$
$i = 10k$
Computational model
$y_i = \mathcal{M}(\mathbf{x}_i)$
Response y_i
PDF
y

Figure 1.2: Illustration of the Monte Carlo method for calculating probabilities of the model output of interest in an uncertain process via repeated random sampling.

1.1.2 Computational Model and Optimization

Undoubtedly, optimization can be considered as the crucial step during the design of electrical and electronic devices, since it allows to optimize the product performance and to meet the design constraints. The underlying idea is to explore the parameter space seeking a configuration of the design parameter able to maximize or minimize a given cost function. As an example, we can think of optimizing the electrical parameters of an amplifier to get a given gain in a specific bandwidth or to optimize the parameters of a filter to keep the conductive emissions of a device below a given threshold.

Figure 1.3 provides a simple illustration of the optimization process. It is usually based on an iterative scheme. During the first iteration, an initial configuration $\boldsymbol{x}_0$ of the optimization parameters is randomly or quasi-randomly chosen according to their range of variation, indeed such parameters can be interpreted as uniform distributed random variables. Then the initial guess $\boldsymbol{x}_0$ is used as input for a simulation run with the computational model providing as a result the output or a set of outputs of interest. If we are so lucky that the simulation result meets the optimization constraints, the algorithm can be stopped. However, in most of the cases the optimization algorithm requires several iterations in which the simulation results obtained in the previous steps are used together with an optimization scheme (e.g., gradient descent algorithm, particle swarm optimizer, etc...) in order to generate a new configuration of the input parameters to be simulated until all the optimization constraints have been satisfied.

It is important to remark that the number of iterations, and thus the number of simulations to be run with the computational model can be huge (e.g. in the order of 1000) for complicated problem [3]. This makes the optimization procedure described above rather expensive in term of computational time, especially when accurate and computational expensive models are used to carried out the simulations at each iteration.

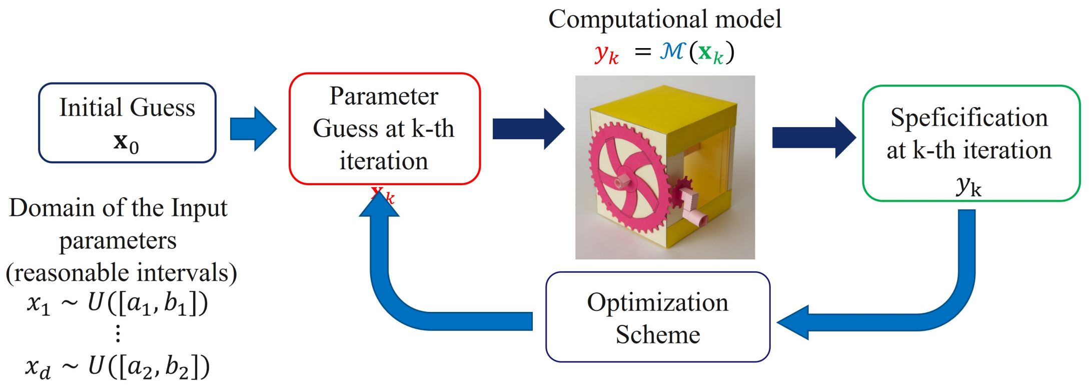
Computational model
$y_k = \mathcal{M}(\mathbf{x}_k)$
Initial Guess $\mathbf{x}_0$
Parameter Guess at k-th iteration
$\mathbf{x}_k$
Speficification at k-th iteration y_k
Domain of the Input parameters (reasonable intervals)
$x_1 \sim U([a_1, b_1])$
$\vdots$
$x_d \sim U([a_2, b_2])$
Optimization Scheme

Figure 1.3: Illustration of a generic optimization scheme based on the computational model.

1.2 Surrogate Models

1.2.1 What is a Surrogate Model?

Surrogate models, also known as metamodels, provide an efficient alternative to the computational model in computationally expensive tasks such as UQ and optimization, thus providing an effective solution able to alleviate their computational cost [3].

A surrogate model is "a model of a model". It allows to provide a closed-form and fast-to-evaluate model able to mimic the actual input-output relationship provided by the computational model. Once the surrogate model for a given structure is available, it can be easily embedded within UQ and optimization tasks presented in Sec. 1.1.1 and 1.1.2 as an efficient alternative to the computational expensive computational model.

As depicted in Fig. 1.4, a surrogate model is built via a regression or interpolation technique based on the information provided by a *small* set of training samples calculated via the computationally expensive computational model. The data set used for training the surrogate model is referred to as training set. The samples belonging to the training set are carefully selected in order to explore the design space (i.e., the domain of the input parameters) as much as possible [3, 8].

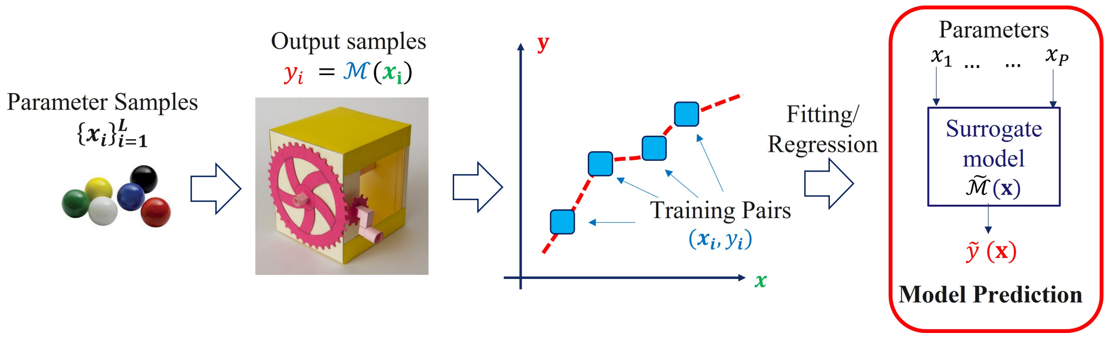

Figure 1.4: Surrogate model training by using a data-driven approach

After the training, since the resulting surrogate model is known in a closed-form, it can be suitably embedded with the optimization and UQ framework as an efficient alternative to the plain computational model. It is important to stress that the performance of the resulting surrogate model, in terms of accuracy with respect to the number of training samples, unavoidably depends on the model structure and the regression technique used to build it [7].

1.2.2 Modeling Challenges and Thesis Motivations

Without loss of generality, the “ideal” interpolation or regression technique for constructing the above-mentioned surrogate models should have the following features [3]:

- Learn complex non-linear input-output relationship;
- Handle a large number of input variables with large parameter variations;
- Deal with possible multi-output or vector-valued problems;
- Converge fast in terms of accuracy w.r.t. the number of training samples;
- Deal with complex-valued problems.

Despite the enormous efforts which have been spent by the academic community in order to develop advance regression and interpolation techniques, no state-of-the-art technique exists complying with all the aforementioned features, since each technique has its own advantages and drawbacks.

In the above scenario, Machine Learning (ML) inspired and data-driven techniques have shown some interesting performance and features for the surrogate model construction [9, 10] and macromodels [11, 12]. The aim of this thesis is to explore and in same cases to enhance state-of-the-art ML techniques with the aim of providing a set of effective and unconventional solutions to the above challenges.

1.3 Thesis Organization

The thesis is organized as follows. Chapter.2 presents a quick overview of state-of-the-art supervise machine learning regressions with the aim of highlighting their advantages and drawbacks for the surrogate modelling construction. Chapter.3 investigates the performance of prior knowledge-based machine learning (PKBML) surrogate modeling technique for the prediction of the per-unit-length (p.u.l.) parameters of the hybrid copper-graphene on-chip interconnects. Chapter.4 discusses an extension of the mathematical framework of the least-square support vector machine (LS-SVM) regression with the aim of extending its applicability to complex- and vector-valued problems via data compression. Chapter.5 presents an alternative generalized vector-valued formulation of a well-consolidated kernel-machine regression, such as the kernel ridge regression (KRR). The proposed methodology is based on a generalized definition of the reproducing kernel Hilbert space (RKHS) and kernel functions to the case of vector-valued learning problem. And finally, conclusion and future work are drawn in Chapter.6.

Chapter 2

State-of-the-art Supervised Machine Learning Regressions

This chapter presents an overview of state-of-the-art supervised machine learning regressions for the surrogate modeling construction. Specifically, three types of regression techniques, classified according to their model structure, will be discussed along this chapter, such as:

- Linear basis function regression
- Artificial neural network
- Kernel machine regression

The above regression techniques will be briefly presented to highlight their advantages and drawbacks for the construction of surrogate models. The performance of the considered methods in terms of efficiency and accuracy with respect to the number of training samples are investigated and discussed on realistic applications with the help of two illustrative examples.

2.1 Learning from Data: Learning Paradigm & Surrogate Model

The aim of this section is to address in a rigours way the problem of constructing a surrogate model. First of all, let us briefly recall the definition of the computational model provided in the previous chapter. From a mathematical point of view, a generic scalar-valued computational model $\mathcal{M}$, provides a forward map describing the input-output behavior of parametric system, such as:

$$y = \mathcal{M}(\boldsymbol{x}) + \varepsilon_0, \tag{2.1}$$

where $\boldsymbol{x} = [x_1, \dots, x_d]^T \in \mathcal{X}$, with $\mathcal{X} \subseteq \mathbb{R}^d$, represents the input design parameters, $\mathcal{M}(\boldsymbol{x}) \in \mathcal{Y}$ with $\mathcal{Y} \subseteq \mathbb{R}$ is the actual noiseless system output, ε_0 represents a possible random noise superimposed to the actual system output and $y \in \mathbb{R}$ is corresponding noisy response.

Our goal is to use a regression or interpolation technique in order to estimate the optimal values of the unknown coefficients collected in the vector $\boldsymbol{w}^*$ characterizing a surrogate model $\tilde{\mathcal{M}}(\boldsymbol{x}; \boldsymbol{w})$, such that:

$$\mathcal{M}(\boldsymbol{x}) \approx \tilde{\mathcal{M}}(\boldsymbol{x}; \boldsymbol{w}^*), \tag{2.2}$$

for any $\boldsymbol{x} \in \mathcal{X}$.

The meaning of the above equation is rather quantitative, but also quite *weak* from the mathematical point of view due to the symbol "≈". Indeed, what does it mean approximating a function?

The above training process can be recast in a more rigorous way in terms of an optimization problem. Specifically, all the regression and interpolation techniques can be formulated in terms of the so-called empirical risk minimization (ERM). Given a set of training pairs $\mathcal{D} = \{(\boldsymbol{x}_l, y_l)\}_{l=1}^{L}$, where $\boldsymbol{x}_l \in \mathcal{X} \subseteq \mathbb{R}^p$ represents the configuration of the training input and $y_l \in \mathcal{Y} \subseteq \mathbb{R}$ are the corresponding scalar outputs computed via the computational model in. (2.1), we seek the optimal configuration of the coefficient vector $\boldsymbol{w}$, characterizing a generic surrogate model $\mathcal{M}$, as the solution of the following ERM [13]:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{l=1}^{L} \ell\left(\tilde{\mathcal{M}}(\boldsymbol{x}_l; \boldsymbol{w}), y_l\right), \tag{2.3}$$

where $\ell(\cdot)$ is generic loss function measuring the "error" between the training outputs and the predictions of the surrogate model $\tilde{\mathcal{M}}$ evaluated on the corresponding training inputs.

The above ERM estimates the best set of model coefficients $\boldsymbol{w}$ as the ones that minimizes the empirical mean of the cost function ℓ computed on the available training set $\mathcal{D}$. Several loss-function are available in the literature. Among them, the most common one is the squared loss, which measure the square of the model error on the training set, i.e.,

$$\ell_S\left(\tilde{\mathcal{M}}(\boldsymbol{x}_l; \boldsymbol{w}), y_l\right) = (y_l - \tilde{\mathcal{M}}(\boldsymbol{x}_l; \boldsymbol{w}))^2, \tag{2.4}$$

for $l = 1, \dots, L$.

The training set $\mathcal{D}$ is obtained by drawing samples $\boldsymbol{x}_i$ of the input variables randomly or pseudo-randomly, and computing the corresponding system responses y_i. As an example, the latin hypercube sampling (LHS) represents a widely used and effective strategy to randomly sample the design space with good exploration properties [8]. Once the model $\tilde{\mathcal{M}}(\boldsymbol{x};\boldsymbol{w}^*)$ is built via the training set $\mathcal{D}$, its accuracy is investigated on the test set $\mathcal{T} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_T, y_T)\}$, which collects an unseen set of sample pairs, which were not been used during the training phase, i.e., $\mathcal{D} \cap \mathcal{T} = \emptyset$ [14].

2.2 Standard Regressions based on Basis Function

Regression models based on basis function can be seen as the most straightforward way to built a surrogate model. The underlying idea is to apply the ERM minimization of Eq. (2.3), in order to estimate the coefficient of a regression model $\mathcal{M}(\boldsymbol{x};\boldsymbol{w})$ defined as a linear combination of basis function and regression coefficients, such as:

$$\tilde{\mathcal{M}}(\boldsymbol{x};\boldsymbol{w}) = \sum_{n=1}^{D} \phi_n(\boldsymbol{x}) w_n = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}) \rangle, \tag{2.5}$$

where $\phi_n(\boldsymbol{x})$ are the basis functions and w_n are the corresponding regression coefficients (i.e., $\boldsymbol{w} = [w_1, \dots, w_D]^T$).

2.2.1 Ordinary Least Squares (OLS) Regression

Ordinary Least Squares (OLS) regression undoubtedly represents the most intuitive and well-known technique for the construction of a surrogate model in terms of a weighted linear combination of basis functions. For the case of the OLS regression, the ERM in 2.3 writes:

$$\min_{\boldsymbol{w}} \sum_{l=1}^{L} \left(y_l - \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_l) \rangle\right)^2. \tag{2.6}$$

The above optimization problem can be rewritten in its matrix form as:

$$\min_{\boldsymbol{w}} \left(\boldsymbol{y} - \boldsymbol{\Phi}^T \boldsymbol{w}\right)^T \left(\boldsymbol{y} - \boldsymbol{\Phi}^T \boldsymbol{w}\right), \tag{2.7}$$

where $\boldsymbol{\Phi}^T \in \mathbb{R}^{L \times D}$ is a matrix collecting the basis functions evaluated on the training inputs, i.e.,:

$$\boldsymbol{\Phi}^T = [\boldsymbol{\phi}(\boldsymbol{x}_1), \dots, \boldsymbol{\phi}(\boldsymbol{x}_L)]^T, \tag{2.8}$$

and $\boldsymbol{y} = [y_1, \dots, y_L]^T \in \mathbb{R}^L$ is a vector collecting the training outputs. According to the above definition the l-th training output is approximated as:

$$y_l \approx \left[\boldsymbol{\Phi}^T \boldsymbol{w}\right]_l = \boldsymbol{\phi}(\boldsymbol{x}_l)^T \boldsymbol{w}. \tag{2.9}$$

The solution of Eq. (2.7) leads to the well-known closed-form solution of the OLS regression based on the pseudo inverse matrix [4]:

$$\boldsymbol{w} = (\boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1}\boldsymbol{\Phi}\boldsymbol{y}. \tag{2.10}$$

The OLS regression is extremely flexible, since the only restriction for the selection of the basis functions is that the squares matrix ($\boldsymbol{\Phi}^T\boldsymbol{\Phi}$) must be invertible. Moreover, the model parameters for the OLS regression can be estimated in a closed form via the simple matrix relation in Eq.(2.10). The above features motivate the popularity of the OLS techniques. On the other hand, such simple approach has two main limitations, restricting its applicability and effectiveness in complex regression problems. First of all, the OLS regression provides a parametric model in which the number of regression unknowns (i.e., the regression coefficients w_n) is equal to the number of basis functions (i.e., the dimensionality of $\boldsymbol{\phi}$). This leads to the infamous *curse of dimensionality* [4] (i.e. a reduction of the model efficiency, when the model complexity increases), thus making the application of such method impractical for systems with many input parameters or requiring a large set of basis functions. In addition to that, the OLS metamodels suffer from high variance and are usually affected by overfitting.

2.2.2 Ridge Regression

Ridge regression shares the same model structure used by the OLS regression, but it allows to limit the overfitting issue of the OLS via a regularization process. The *regularization process* or *regularizer* is a penalty, which is added to the loss function to limit the value of the regression coefficients, thus improving the model generalization and limiting its variance, mitigating the overfitting issue [15].

The regression training reduces to the solution of the following penalized ERM:

$$\min_{\boldsymbol{w}} \sum_{l=1}^{L} \left(y_l - \langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}_l)\rangle\right)^2 + \lambda \|\boldsymbol{w}\|^2, \tag{2.11}$$

where the extra term $\|\boldsymbol{w}\|^2$ is the Tikhonov regularizer used to limit the L2-norm of the regression coefficients and $\lambda \geq 0$ is the regularizerd parameters acting as a weight for the regularizer contribution. Such parameter λ is usually referred to as hyperparameter, since, different from the regression coefficients $\boldsymbol{w}$, its value cannot be estimated via the training set.

From Eq. (2.11), it is easy to see that by increasing the value of the hyperparameter λ, we are limiting the maximum norm allowed by the regression coefficients in $\boldsymbol{w}$, thus penalizing the error on the training samples. In such a way, by changing the value of λ, we are able to tune the model bias and variance.

Similar to the OLS, for any given value of λ, the optimal coefficients $\boldsymbol{w}^*(\lambda)$ can be computed in a closed form as:

$$\boldsymbol{w}^*(\lambda) = (\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}\boldsymbol{y}, \tag{2.12}$$

where $\boldsymbol{I} \in \mathbb{R}^{(N+1)\times(N+1)}$ is the identity matrix.

Thanks to the regularizer, Ridge regression allows to overcome the overfitting issue, even if a dedicated algorithm must be implemented for the tuning of the regularizer hyperparameters. However, similar to the OLS, the obtained model is still a parametric one.

2.2.3 Hyperparameter Tuning

The tuning of the hyperparameter λ is a key step of the training phase since it allows to reduce the overfitting and to improve the model robustness to the noise. The tuning of hyperparamaters, such as the λ in Eq. (2.12) must be carried out via dedicated algorithms based on two different strategies: the validation set and the cross-validation [15].

2.2.4 Validation Set

The underlining idea used by the validation set is to partition the available data into 3 sets: the training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{L}$, the validation set $\mathcal{V} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{V}$ and the test set $\mathcal{T} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{T}$ such that $\mathcal{D} \cap \mathcal{V} \cap \mathcal{T} = \emptyset$, and to use the validation set to tune the hyperparameters of the model.

For doing that, a possible set of values for the hyperparameter λ (e.g., log-spaced values $\lambda = 10^{-2}$,10^{-1},$1, \dots$,10^5) is defined. In the next step, the training set is used to build a metamodel $\tilde{\mathcal{M}}^{(\lambda)}(\boldsymbol{x})$ for each value of λ defined in the previous step. For each metamodel $\tilde{\mathcal{M}}^{(\lambda)}(\boldsymbol{x})$, the error on the validation set $\mathcal{V}$ is calculated by using a given figure of merit, e.g., for the mean square error (MSE) writes:

$$MSE_{\text{validation}}^{(\lambda)} = \frac{1}{V} \sum_{(\boldsymbol{x},y)\in\mathcal{V}} (y - \tilde{\mathcal{M}}^{(\lambda)}(\boldsymbol{x}))^2. \tag{2.13}$$

The optimal value λ^* of gthe hyperparameter is chosen as the one which minimizes the above validation error, such as:

$$\lambda^* = \arg\min_{\lambda} MSE_{validation}^{(\lambda)}. \tag{2.14}$$

The main limitation of the above approach is that by partitioning the available data into three sets, we are drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the samples pairs in the train and validation set [15, 16].

2.2.5 Cross Validation

Cross-validation (CV) can be seen as an effective alternative strategy in which the validation set is no longer needed. The basic approaches are the k-fold and Leave-one-out CV. Such strategies can be computationally expensive, but they do not waste too many data, which is a major advantage in problems where the number of samples is "small". In the k-fold CV, the training set is split into k smaller sets, called "folds". For each of the k folds, a metamodel is trained using $k-1$ folds as training data and the remaining fold is used as validation set. The above scheme is iterated for all the k-fold. For each value of the hyperparameter λ, the overall model performance calculated by the k-fold CV as the average of the values computed during the k iterations. The common choice for k is usually 5 or 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance. When $k=1$ the k-fold validation is called leave-one-out (LOO) cross-validation. An illustration of the k-fold CV for the case in which we have $L=100$ training samples and 5 folds (i.e., $k=5$) is shown in Fig. 2.1.

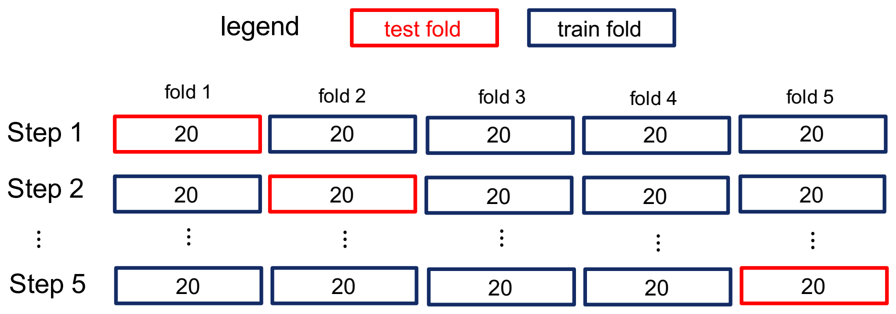

Figure 2.1: Illustration of the k-fold cross-validation with K=5 and L=100 training samples.

Specifically, the step-by-step procedure for the above algorithm can be summarized as follows. At first, the training data $\mathcal{D}=\{(\boldsymbol{x}_i,y_i)\}_{i=1}^{L}$ is split into K separate datasets of equal size $\mathcal{D}_k=\{(\boldsymbol{x}_{k,i},y_{k,i})\}_{i=1}^{\tilde{L}}$ with $\tilde{L}=L/K$, such that $\mathcal{D}=\cup_{k=1}^{K}\mathcal{D}_k$. Then for each $k=1\ldots,K$, we fit the model $\tilde{\mathcal{M}}_{-k}^{(\lambda)}$ by considering the training set $\mathcal{D}$ excluding the k-th fold. We use the model $\tilde{\mathcal{M}}_{-k}^{(\lambda)}$ to compute the fitted values on the observations $\mathcal{D}_k$. The latter can be considered as the validation set, since the model $\tilde{\mathcal{M}}_{-k}^{(\lambda)}$ has been trained excluding this fold.

The CV error can computes for the k-fold as following:

$$CV^{(\lambda)}_{\text{error},k} = \frac{1}{\tilde{L}} \sum_{(\boldsymbol{x},y)\in\mathcal{D}_k} \left(y - \tilde{\mathcal{M}}^{(\lambda)}_{-k}(\boldsymbol{x}) \right)^2 . \tag{2.15}$$

The overall model CV error for a given value of the hyperparameter λ is defined as the average of the CV error computed on the available folds, and writes:

$$CV^{(\lambda)}_{\text{error}} = \frac{1}{K} \sum_{k=1}^{K} CV^{(\lambda)}_{\text{error},k} \tag{2.16}$$

The optimum value of the hyperparameter λ^* is selected as the one that minimizes the corresponding overall CV error $CV^{(\lambda)}_{\text{error}}$, i.e.,

$$\lambda^* = \arg\min_{\lambda} CV^{(\lambda)}_{\text{error}}. \tag{2.17}$$

After selecting the optimal value λ^*, the performances of the metamodel $\tilde{\mathcal{M}}^{(\lambda^*)}$ are evaluated by computing the error on the test samples.

2.3 Artificial Neural Network

ANN regressions can be seen as viable way to overcome to the inherent parametric nature shared by regression models based on linear basis function expansion. Indeed, ANN allows to train non-parametric models in which the model parameters to be estimated during the training phase depends on the network structure and turns out to be independent from the number of input parameters.

According to the universal approximation theorem [17], ANN structures can approximate any non-linear function, or a set of functions for the multi-output case, via a collection of artificial neurons connected together and organized in layers. In such structure the role of regression unknowns is played the bias term b and by the synaptic weights w_{ij}. The overall structure turns out to be extremely flexible, without any limitation in terms of number of layers, neurons per layer, number of outputs, etc... The first and last layers are called the input and output layer, respectively, while the other layers in between are named hidden layers. If there is more than 1 hidden layer the neural network is called deep neural networks.

As an example, a generic NN structure with $n_i = d$ input parameters and a single hidden layer with $n_h = 3$ neurons is depicted in Fig. 2.2. For the proposed example, the input-output map obtained by the ANN writes [18]:

$$\tilde{\mathcal{M}}_{ANN}(\boldsymbol{x};\boldsymbol{w}) = \sigma_o \left(\sum_{j=0}^{n_h=3} w^{(2)}_j \sigma_h \left(\sum_{i=0}^{n_i=d} w^{(1)}_{ji} x_i \right) \right), \tag{2.18}$$

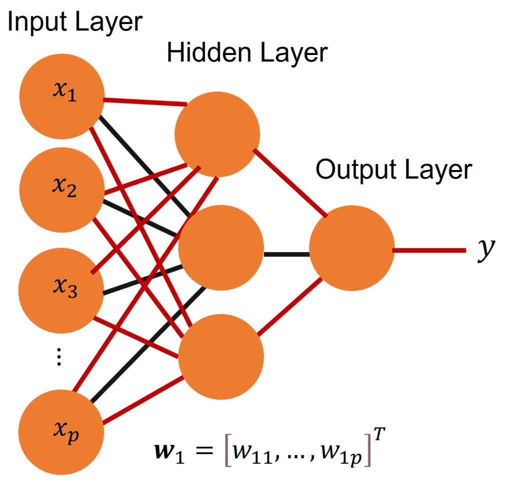

Figure 2.2: Basic ANN architecture with a single hidden layer.

where the vector $\boldsymbol{w}$ collects the set of weights $w_{ji}^{(1)}$ and $w_j^{(2)}$ of the input and hidden layer, respectively, and the corresponding bias parameters (i.e., $w_{j0}^{(1)} = b_j$ and $w_o^{(2)} = b^{(2)}$) to be estimated during the training phase. The functions $\sigma_h(\cdot)$ and $\sigma_o(\cdot)$ are the activation functions (usually non-linear functions) associated to each neuron belonging to the hidden and output layer, respectively.

Several activation functions are available in the literature, such as: linear, sigmoid, tangent hyperbolic, rectified linear unit and softmax function [18]. For instance, one of the activation function is the logistic sigmoid function [17, 19, 20], which writes:

$$\sigma(s \cdot v) = \frac{1}{1 + e^{-s \cdot v}}. \tag{2.19}$$

Similar to the previous methods, the model training can be written in terms of the following ERM:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{L} E_i(\boldsymbol{w}) = \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{L} \left(y_i - \tilde{\mathcal{M}}_{ANN}\left(\boldsymbol{x}_i; \boldsymbol{w}\right) \right)^2. \tag{2.20}$$

However, different from the OLS and ridge regression presented in the previous section, the above optimization can not be solved in a closed-form. Indeed, the mathematical model describing the input-output map obtained by the ANN is usually not linear with respect to the model unknowns (i.e., the weights and bias), since they appear within the argument of the activation functions. This allows learning very complex non-linear behaviors, but on the other hand, the non-linear structure of the ANN model makes the ERM in Eq. (2.20) a non-convex optimization problem with several local minima. Such optimization is usually solved with the help of advance optimization algorithms, such as:

- Gradient Descent [18]: $w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E\left(w^{(\tau)}\right)$
- Stochastic Gradient Descent [21]: $w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_i\left(w^{(\tau)}\right)$
- ADAM (Adaptive Moment Estimation) [21]: adaptive algorithm (in which η can be different at each iteration) for stochastic optimization

in which the gradients are calculated via the Back propagation algorithm [22].

It is important to remark that the non-convex nature of the ERM associated to the ANN makes the training phase rather complicated and data-hungry [23, 24]. Thus limiting the accuracy of ANN-based regression in surrogate modeling applications in which a small set of training sample is available. On the other hand, ANN structures are extremely flexible, since users can easily add or remove neurons and layers.

2.4 Kernel Based Regressions

Similar to ANN regression, kernel-machine regressions allow overcoming the curse of dimensionality shared by regression models based on basis functions, since they allow to build non-parametric models. Kernel methods in machine learning owe their name to the use of kernel functions. The kernel quantifies similarities between input samples and thanks to the kernel tricks, they allow to build non-parametric models in which the number of regression unknowns to be estimated turns out to be independent from the model complexity and thus from the number of input parameters.

2.4.1 Support Vector Machine Regressions

Support vector machine (SVM) represents one of the most popular machine learning tool for both classification and regression. Within the proposed regression scenario, SVM regression has shown interesting features such as: robustness to noise and model sparsity.

The primal space formulation for the SVM regression can be written as a standard basis function expansion, i.e., [25, 26]:

$$\tilde{\mathcal{M}}_{SVM}(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}) \rangle + b = \sum_{n=1}^{D} w_i \Phi_i(\boldsymbol{x}) + b \tag{2.21}$$

where $\boldsymbol{w} \in \mathbb{R}^D$ is a vector collecting the regression coefficients, $b \in \mathbb{R}$ is the bias term and $\boldsymbol{\Phi}(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \ldots, \phi_D(\boldsymbol{x})]^T$ is a nonlinear map $\boldsymbol{\Phi}(\cdot) : \mathbb{R}^d \to \mathbb{R}^D$ which maps the parameter space of dimension d into the feature space of dimension D.

It is important to remark that the definition of the primal space formulation of the SVM regression in Eq. (2.21) turns out to be equivalent to a classical linear regression, like the ones presented in the previous Sections (e.g., OLS and ridge regressions). Indeed, also in this case, the number of regression unknown (i.e., the cardinality of $\boldsymbol{w}$) is equal to the dimensionality of the feature space (i.e., D) defined by the nonlinear map $\Phi(\boldsymbol{x})$.

For the primal space formulation of the SVM regression, the unknowns $(\boldsymbol{w}, b)$ in Eq. (2.21) is obtained via the following regularized ERM:

$$\begin{aligned} &\min_{\boldsymbol{w},b} \underbrace{\frac{1}{2}\|\boldsymbol{w}\|_2^2}_{\text{Regularizer}} + \underbrace{\frac{C}{L}\sum_{i=1}^{L} |y_i - \tilde{\mathcal{M}}_{SVM}(\boldsymbol{x}_i; \boldsymbol{w}, b)|_\varepsilon}_{\text{Empirical Risk}} && (2.22) \\ &= \min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \frac{C}{L}\sum_{i=1}^{L} |y_i - (\langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b)\,|_\varepsilon, && (2.23) \end{aligned}$$

where, different from the Ridge regression, the above optimization uses as loss function the linear ε-insensitive loss function denoted as $|\cdot|_\varepsilon$, defined as [25]:

$$|y_i - \tilde{\mathcal{M}}_{SVM}(\boldsymbol{x}_i)|_\varepsilon = \begin{cases} 0, & \text{if } |y_i - \tilde{\mathcal{M}}_{SVM}(\boldsymbol{x}_i)| \le \varepsilon \\ |y_i - \tilde{\mathcal{M}}_{SVM}(\boldsymbol{x}_i)| - \varepsilon, & \text{otherwise.} \end{cases} \tag{2.24}$$

Figure 2.3 (left panel) provides a graphical interpretation of the above optimization problem. The underlying idea is to minimize the upper ξ_i and lower ξ_i^* deviations of the training samples which lay outside the ε-insensitive zone (gray area), but at the same time with the help of a ridge regularizer, we are also maximizing the model flatness to avoid overfitting.

Minimizing the risk function Eq. (2.24) with respect to the ε-norm is equivalent to find the pair $(\boldsymbol{w}, b)$ in Eq. (2.23), that minimizes the deviation of the model from the training samples outside the ε-insensitive zone. Therefore, the ERM in Eq. (2.23), can be recast as [26]:

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{L}(\xi_i + \xi_i^*) \\ &\text{subject to } \begin{cases} y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle - b \le \varepsilon + \xi_i \\ \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0, \end{cases} \end{aligned} \tag{2.25}$$

where ξ_i, ξ_i^* are slack variables that indicate a positive and negative deviation of the training samples lying outside of the ε-intensive zone, whereas C is the hyperparameters of the regularizer (i.e, it is equivalent to the $\frac{1}{\lambda}$ used in the Ridge regression) providing a trade-off between the accuracy of the model and its flatness empirically chosen by the user [27].

The optimization problem with inequality constraints in Eq. (2.25) is transformed into its dual problem [26, 27] and then solved by minimizing the corresponding Lagrangian function. The solution allows estimating the optimum $\boldsymbol{w}$ via a linear combination of the training patterns $\boldsymbol{\Phi}(\boldsymbol{x}_i)$, and writes,

$$\boldsymbol{w} = \sum_{i=1}^{L} (\alpha_i - \alpha_i^*) \boldsymbol{\Phi}(\boldsymbol{x}_i), \tag{2.26}$$

where $\alpha_i, \alpha_i^* \in [0, C]$ are the pertinent Lagrange multipliers related to the constraints of the optimization problem. Additional details on the Lagrangian minimization and on the dual problem formulation are available in [26, 28].

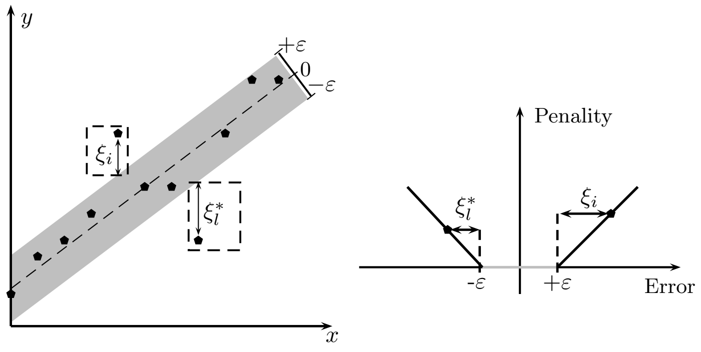

Figure 2.3: Graphical interpretation of the minimization problem Eq. (2.23) and of the role of ε-intensive loss function in Eq. (2.24) (inspired by [27, 29]).

Substituting Eq. (2.26) in Eq. (2.21) leads to

$$\begin{aligned}\tilde{\mathcal{M}}_{SVM}(\boldsymbol{x}) &= \sum_{i=1}^{\#SV} \beta_i \langle \boldsymbol{\Phi}(\boldsymbol{x}_i) \boldsymbol{\Phi}(\boldsymbol{x}) \rangle + b \\ &= \sum_{i=1}^{\#SV} \beta_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b, \end{aligned} \tag{2.27}$$

where $\beta_i = (\alpha_i - \alpha_i^*)$, $\#SV$ is the number of the support vectors (i.e., the training configurations $\boldsymbol{x}_i$ for which either α_i or α_i^* do not vanish) which are always less or equal to the number of training samples L and $K(\boldsymbol{x}_i, \boldsymbol{x}) = \langle \boldsymbol{\Phi}(\boldsymbol{x}_i), \boldsymbol{\Phi}(\boldsymbol{x}) \rangle$ is the so-called kernel functions defined as the inner product between the basis functions $\boldsymbol{\Phi}(\boldsymbol{x})$ evaluated at a given training sample $\boldsymbol{x}_i$ and the same function at a generic point $\boldsymbol{x} \in \mathbb{R}^d$.

Equation (2.27) does not require an explicit calculation of the inner product $\langle \boldsymbol{\Phi}(\boldsymbol{x}_i), \boldsymbol{\Phi}(\boldsymbol{x}) \rangle$ which is extremely inefficient when the dimensionality of the D-space increases. Indeed, the SVM regression is completely defined by the kernel K

without requiring an explicit definition of the nonlinear transformation $\boldsymbol{\Phi}(\boldsymbol{x})$. This is the so-called *kernel trick*.

The SVM regression is already implemented within the MATLAB machine learning toolbox, which includes the following three classes of kernels:

- linear: $K(\boldsymbol{x}_i, \boldsymbol{x}) = \boldsymbol{x}_i^T \boldsymbol{x}$;
- polynomial (of order q): $K(\boldsymbol{x}_i, \boldsymbol{x}) = (1 + \boldsymbol{x}_i^T \boldsymbol{x})^q$, where q is the kernel hyperparameter;
- Gaussian Radial Basis Function (RBF): $K(\boldsymbol{x}_i, \boldsymbol{x}) = \exp\left(-\|\boldsymbol{x}_i - \boldsymbol{x}\|^2 / \sigma\right)$, where σ is the kernel hyperparameter.

In he above

Thanks to the kernel trick, the number of regression unknowns in Eq. (2.27) (i.e., the coefficients β_i) to be estimated by solving the SVM optimization is always equal to the number of support vectors $\#SV$. In fact, the dual space formulation of the SVM regression of Eq. (2.27) provides *non-parametric* model in which the number of model unknowns are independent from the number of input parameters d and the dimensionality of the feature space $\mathcal{D}$. Such property can be extremely useful, since it alleviates the effects of the curse of dimensionality. Also, it is important to note, the resulting SVM expansion turns out to be sparse. Indeed, when $|y_i - \mathcal{M}_{SVM}(\boldsymbol{x}_i)| \leq \varepsilon$, the corresponding Lagrange multipliers α_i, α_i^* turn out to be equal to 0 [28].

The parameter ε in SVM can be introduce as the regression tolerance. As a result, this kind of regression can be extremely useful to deal with noisy dataset.

2.4.2 Least Square Support Vector Machine Regressions

The LS-SVM regression provides an alternative solution with respect to the standard SVM regression, which allows recasting the quadratic optimization problem used by the SVM regression, in terms of a more standard least squares formulation [30].

Similar to the SVM regression, the primal space formulation of the LS-SVM regression writes:

$$\tilde{\mathcal{M}}_{LS-SVM}(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}) \rangle + b. \tag{2.28}$$

Like the Ridge regression, the minimization problem for the LS-SVM regression uses a squared loss function and a Tihkonov regularization:

$$\min_{\boldsymbol{w},b} \underbrace{\frac{1}{2}\|\boldsymbol{w}\|_{L_2}^2}_{\text{Regularizer}} + \underbrace{\frac{\gamma}{L}\sum_{i=1}^{L}\left(y_i - \tilde{\mathcal{M}}_{LS-SVM}(\boldsymbol{x}_i; \boldsymbol{w}, b)\right)^2}_{\text{Empirical Risk Function}}, \tag{2.29}$$

The formulation is equivalent to the following one:

$$\min_{\boldsymbol{w},b,\boldsymbol{e}} \frac{1}{2}\|\boldsymbol{w}\|_{L_2}^2 + \gamma\frac{1}{2}\sum_{i=1}^{L} e_i^2 \tag{2.30}$$
$$\text{subject to } y_i = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b + e_i, \text{ for } i = 1,\dots,L,$$

where the terms $e_i = y_i - \tilde{\mathcal{M}}_{LS-SVM}(\boldsymbol{x}_i;\boldsymbol{w},b)$ with $i = 1,\dots,L$ are the error variables and γ is the regularizer hyperparameters which provides a trade-off between the accuracy of the model and its flatness, playing the same role of the parameter C in the SVM primal optimization of Eq. (2.25).

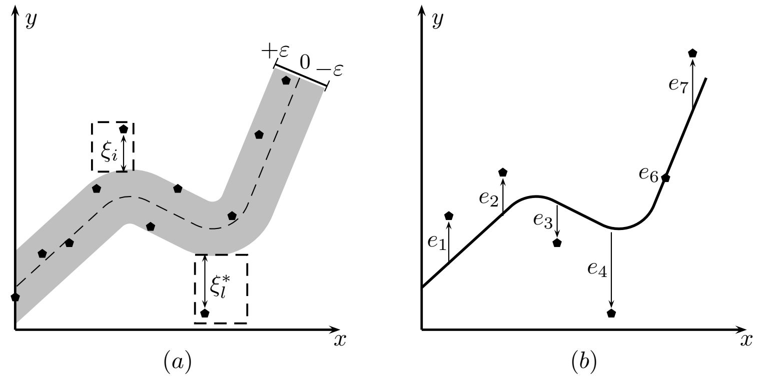

Figure 2.4: Panel (a): graphical interpretation of the SVM regression optimization problem in Eq. (2.25) (inspired by [27, 29]). Panel (b): illustration of the corresponding least square formulation in Eq. (2.30) for the LS-SVM regression.

The LS-SVM regression admits a dual space formulation, which writes:

$$\tilde{\mathcal{M}}_{LS\text{-}SVM}(\boldsymbol{x}) = \sum_{i=1}^{L} \alpha_i K(\boldsymbol{x}_i,\boldsymbol{x}) + b, \tag{2.31}$$

where α_i and b are the regression coefficients and bias term, respectively and K is the kernel function.

Similar to the SVM regression, the above LS-SVM formulation in the dual space is a non-parametric model in which the number of regression unknowns α_i is independent from both the dimensionality of the dual space $\mathcal{D}$ and from the number of parameters d, but it is equal to the number of training samples L. However, since a square loss function is used in the penalized ERM, such regression unknowns can be computed in closed-form as the solution of the following linear system of equations:

$$\begin{bmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & \boldsymbol{\Omega} + \boldsymbol{I}/\gamma \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{y} \end{bmatrix} \tag{2.32}$$

where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_L]^T$, $\boldsymbol{y} = [y_1, \ldots, y_L]^T$, $\mathbf{1}^T = [1, \ldots, 1] \in \mathbb{R}^{1\times L}$, $\boldsymbol{I} \in \mathbb{R}^{L\times L}$ is the identity matrix and $\boldsymbol{\Omega} \in \mathbb{R}^{L\times L}$ is the Gram kernel matrix for which the element $\Omega_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for any $i, j = 1, \ldots, L$. Similar to the SVM regression, the most common kernel functions adopted in the dual space formulation of the LS-SVM regression are the linear, polynomial and Gaussian RBF kernel functions.

The LS-SVM regression is already implemented within LS-SVM Lab Toolbox version 1.8 [31], which is fully compatible with the MATLAB environment.

2.5 Example

The performances of the state-of-the-art regression techniques presented in the previous Sections are investigated on two different applications consisting of a real dataset for the wet human skin permittivity [32] and of a hybrid copper-graphene on-chip interconnect.

2.5.1 Example I: Skin permittivity Dataset

First of all, let us compare standard linear regressions (i.e., OLS and ridge regression) with kernel machine regressions (i.e., SVM and LS-SVM regression) to better understand the role of the regularizer. Without loss of generality, the features of above techniques are investigated on an illustrative example with a single input parameter. Indeed, working with a 1D-input space will allow us to provide a clear understanding of the model features via graphical illustrations.

Specifically, the accuracy of OLS, ridge, LS-SVM and SVM regressions are compared on an experimental dataset gathered from [32], collecting the measurements of the wet human skin permittivity as a function of frequency. The dataset includes 171 permittivity values in a frequency bandwidth from $\sim$ 20 Hz to 20GHz with a large amount of variability. A small subset of 10 measurement data has been used as training samples, and the remaining 161 samples are used as test samples. The selected samples have been organized in the training set $D = \{(x_i, y_i)\}_{i=1,\ldots,10}$, in which x_i and y_i are the log base 10 of the frequency and permittivity values, respectively.

The training set D is then used to train surrogate models based on a polynomial expansion for the OLS and ridge regression and Gaussian RBF kernel expansion for the LS-SVM and SVM regression. Figure 2.5 shows the prediction obtained by the above surrogate models. The results highlight the detrimental effect of the overfitting for the OLS surrogate when a high-order polynomial expansion (order

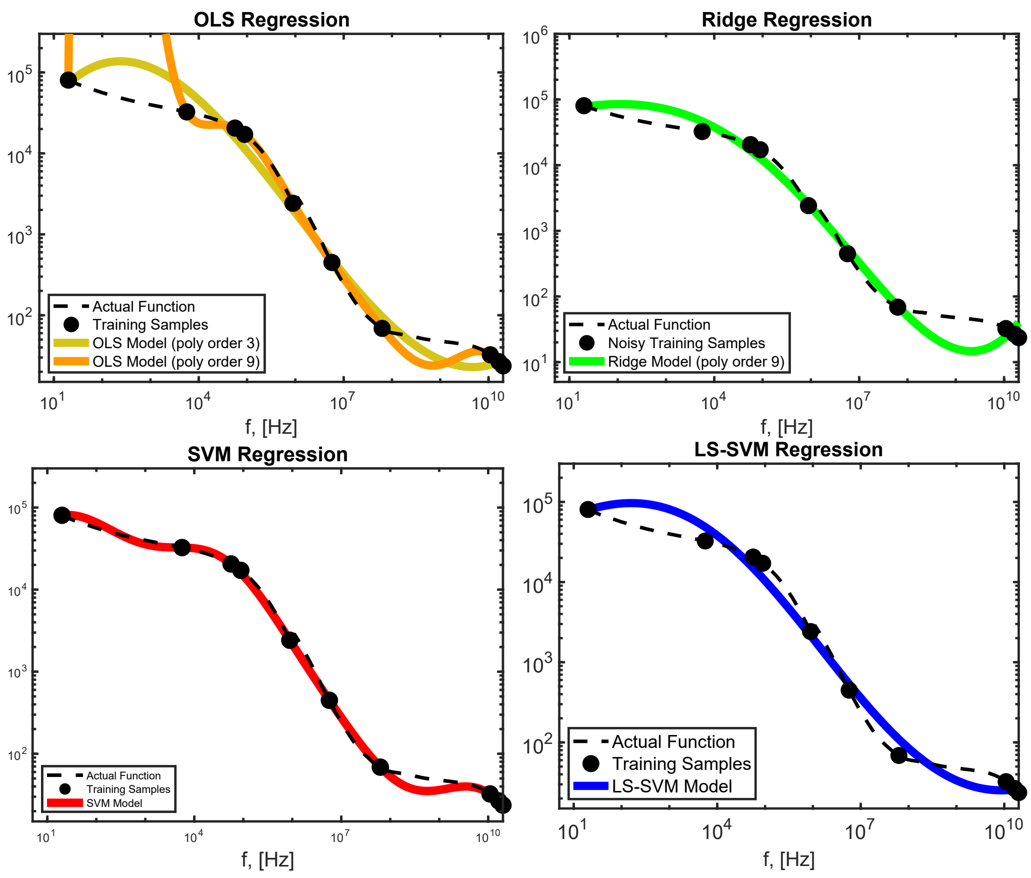

Figure 2.5: Permittivity of the wet human skin vs frequency predict by the OLS, Ridge, SVM and LS-SVM surrogate models

9) is considered. In particular, when a order 9 polynomial expansion is used, the obtained model fits perfectly the training set, but does not generalized well on the test set. On the other hand, the plots show the capability of the Ridge, LS-SVM and SVM regression to provide accurate surrogate models, since the over fitting issue is limited by the regularizer.

As a further validation, the outputs of the training set are corrupted by an additive Gaussian noise mimicking measurement error, as follows:

$$\tilde{y}_i = y_i \times (1 + \varepsilon), \tag{2.33}$$

where $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is a Gaussian variable with zero mean and standard deviation $\sigma_\varepsilon = 0.05$.

The predictions obtained by the surrogate models built with the noisy training set are shown in Fig. 2.6. The plots show again the loss of accuracy of the surrogate model based on OLS with order 9, as well as the beneficial effect introduced by the

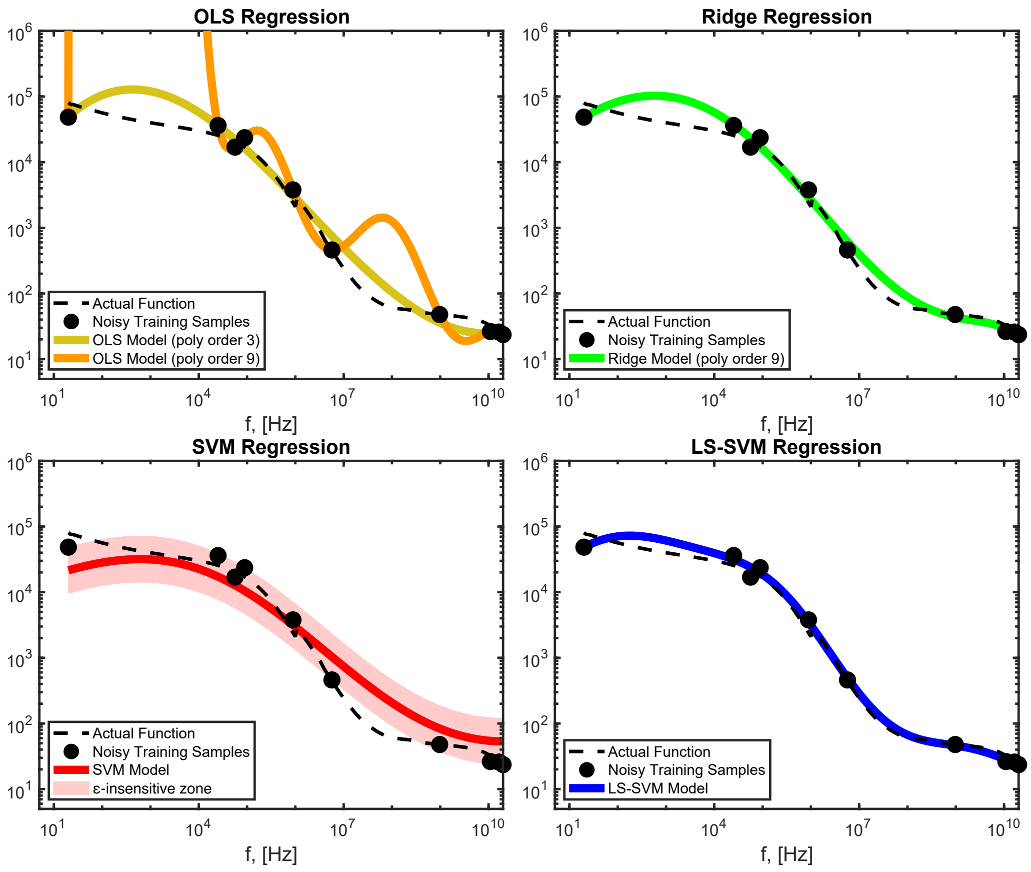

Figure 2.6: Permittivity of the wet human skin vs frequency which corrupted by noise predicted by the OLS, SVM and LS-SVM surrogate models .

regularizer adopted by the Ridge, SVM and LS-SVM. Due to the overfitting, the model trained via the OLS fits perfectly the noisy training samples, but it does not generalize well on the test samples. Also, the ε-insensitive loss function (light red area in Fig. 2.6) makes the SVM regression to be more effective and accurate with noisy samples.

It is important to remark, that the above analysis does not investigate the curse of dimensionality issue affecting standard linear regressions, being the latter issue harder to illustrate (i.e., the dimensionality of the input space must be heavily increased) and already addressed in several publications [32].

2.5.2 Example II: Hybrid Copper-Graphene On-Chip Interconnects

This section aims at comparing the performance of the ANN and kernel-machine regressions (i.e., the SVM and LS-SVM regression) based on a realistic test case consisting of a hybrid copper-graphene on-chip interconnect. Specifically, the above techniques are hereafter adopted in order to build a surrogate model able to predict the impact of 8 geometrical parameters on the p.u.l. resistance (R), inductance (L), and capacitance (C) matrices characterising the 3-conductor hybrid copper graphene interconnect shown in Fig. 2.7. The considered 8 geometrical parameters and their range of variation are provided in Tab. 2.1.

For the considered structure, the computational model consists in a parametric implementation in ANSYS Q3D. Additional detail on this regard will be given in Chap. 3. The ANSYS Q3D implementation allows computing the p.u.l. parameters specified above for any configuration of the considered input parameters within their range of variability and it is used together with a latin hypercube sampling (LHS) to generate the training and test samples.

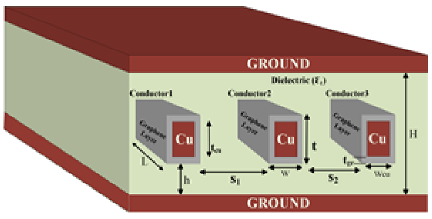

Figure 2.7: Cross-section view of the hybrid interconnect structure considered in this example.

Table 2.1: Design parameters for the hybrid copper graphene interconnect in Sec. 2.5.2.

	Design Parameters	Nominal Values (nm)	Variation
1	Width (w)	18	30%
2	Thickness (t)	37.8	30%
3	Spacing between Line 1 and 2 (S_1)	13	30%
4	Spacing between Line 2 and 3 (S_2)	17	30%
5	Height of Interconnect from GND Layer (h_1)	37	30%
6	Total Height of Dielectric (H)	113.8	20%
7	Barrier Layer Thickness ($t_g r$)	1	30%
8	Dielectric Constant (ϵ_r)	3.9	30%

The training set has been used to train three different surrogate models based on the ANN, SVM and LS-SVM regressions. The ANN surrogates used a single hidden layer and are trained using the Levenberg-Marquardt optimizer with back-propagation as available in the ML toolbox in MATLAB. The SVM and the LS-SVM based surrogate models are trained in MATLAB via built-in functions and the LS-SVM lab v1.8 toolbox [33]. Specifically, the SVM-based surrogates are optimized using a Bayesian optimizer with 5-fold cross-validation (CV) error [34] for a maximum of 30 iterations, while the LS-SVM surrogates use the same Bayesian optimizer with a leave-one-out CV error [33] for a maximum of 25 optimization steps.

For each of the above regression techniques, the corresponding surrogate model has been trained by considering an increasing number of training samples, i.e., $L =$ 35, 140 and 700 samples. Then, the model performance are evaluated on the same test set consisting of 500 samples. Table 2.2 provides an exhaustive comparison among the considered techniques in terms of training time, average and maximum normalized root mean square (NRMS) computed on the 9 p.u.l. parameters for $L = 35$, 140, and 700 training samples. The NRMS error is computed as:

$$\varepsilon_{\text{NRMS}} = \frac{1}{N_{\text{test}}}\sqrt{\frac{\sum_{l=1}^{N_{\text{test}}} \left(y\left(\boldsymbol{x}_l\right) - \tilde{y}\left(\boldsymbol{x}_l\right)\right)^2}{\sum_{l=1}^{N_{\text{test}}} y\left(\boldsymbol{x}_l\right)^2}} \times 100, \tag{2.34}$$

where $y(\boldsymbol{x}_l)$ is the true result obtained from the full-wave EM solver and $\tilde{y}(\boldsymbol{x}_l)$ is the result predicted by the considered surrogate model.

The results highlight the improved convergence of the test error with respect to the number of training samples (i.e., L) achieved by the surrogate model trained via kernel-machine regressions (i.e., SVM and LS-SVM) compared to the ANN. Moreover, the training time for the kernel-machine regressions turns out to be extremely faster compared to the corresponding models trained via ANNs, especially when a relatively small training set is considered. Such training costs can be considered negligible compared to the ones related to the training set generation via the computational model (see the first row in Tab. 2.2). It is important to remark that after the training, the evaluation of each of the proposed surrogate models on 500 configuration is in the order of seconds, with a huge speed up compared to the corresponding evaluations with the computational model.

Table 2.2: Accuracy and time cost comparison between the considered surrogate models for a training dataset with $L = 35$, 140, and 700 samples

Methods	L=35(Cost=105 Minutes)			L=140(Cost=420 Minutes)			L=700(Cost=2100 Minutes)		
	Ave RMSE(%)	Max RMSE	Training Time (s)	Ave RMSE(%)	Max RMSE	Training Time (s)	Ave RMSE(%)	Max RMSE	Training Time (s)
SVM	0.85	1.57	190	0.63	1.43	241	0.63	1.42	1804
LS-SVM	0.81	1.5	2.38	0.62	1.41	5.47	0.64	1.46	57.6
ANN	2.28	4.5	3468	1.38	2.79	5994	1.16	2.45	8554

Moreover, Fig. 2.8 and Fig. 2.9 shows the accuracy of the proposed surrogate models in terms of NRMSE computed on the test set for an increasing number of training samples. The results highlight again the improved convergence of the test error with respect to the number of training samples (i.e., L) achieved by the surrogate model trained via kernel-machine regressions (i.e., SVM and LS-SVM) compared to the ANN.

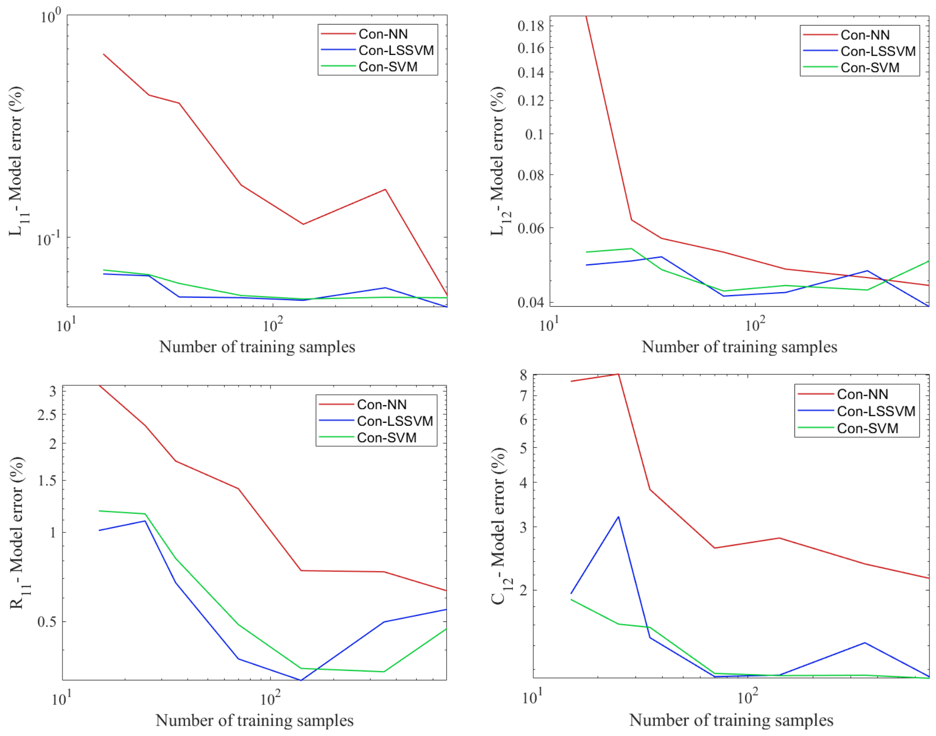

Figure 2.8: Scaling of the testing NRMSE with increasing number of training points for the values of the L_{11},L_{12},R_{11} and C_{11} p.u.l. parameters using the different machine learning metamodel.

The improved performance of the kernel-machine regressions compared with the ANN in terms of accuracy and computational training cost is motivated by the different complexity of the optimization problem to be solved during the model training. Indeed, as pointed out in the previous Sections, the training of kernel-machine regressions requires the solution of a convex optimization, which is unavoidably simpler to solved compared to the non-convex optimization required by the ANN.

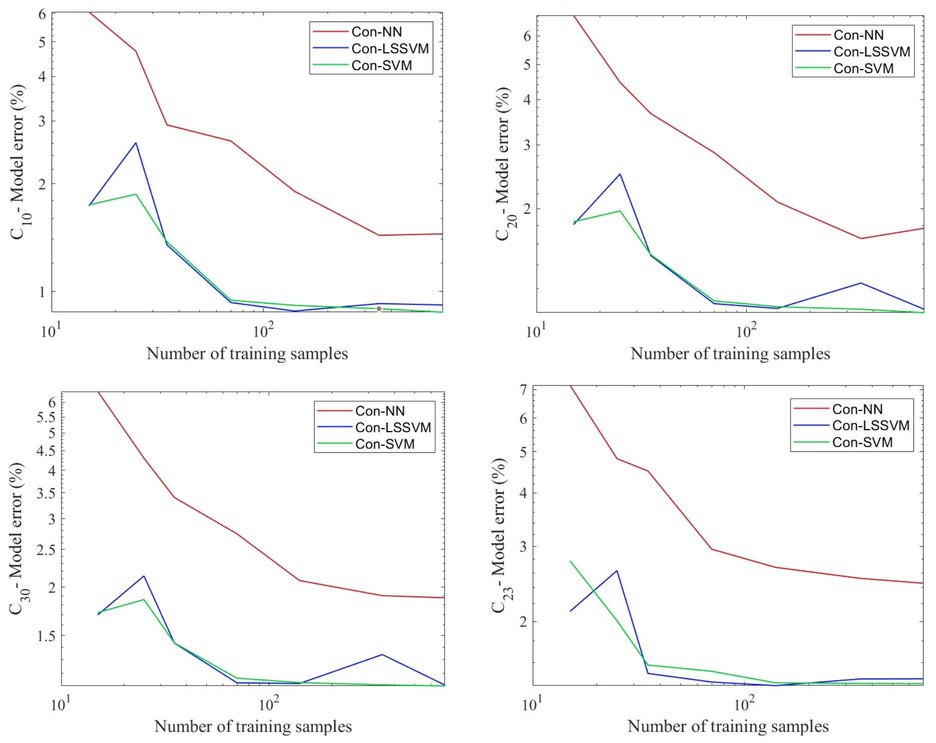

Figure 2.9: Scaling of the testing NRMSE with increasing number of training points for the values of the C_{12}, C_{20}, C_{23} and C_{30} p.u.l. parameters using the different machine learning metamodel.

2.6 Summary

This Chapter presented an overview of three state-of-the-art regression techniques for the construction of surrogate models such as: linear basis function regressions, artificial neural network and kernel machine regressions. The mathematical formulation of the above techniques has been briefly presented to highlight their advantages and drawbacks. Moreover, the performance of the above techniques is quantitatively investigated on two illustrative examples.

The obtained results can be summarized as follows:

- linear basis function regressions are easy to code, but their performance in regression problems with several input parameters are limited by the curse of dimensional and for the OLS regression by overfitting issue;
- ANN-based regression is extremely flexible and allows to lean complex non-linear input-output behavior providing as a result a non-parametric model. On the other hand, due to the non-convex nature of the optimization to be solved during the training phase, the convergence of the ANN accuracy

with respect to the number of training samples is usually quite slow. Thus compromising its applicability in regression problem in which a “small” set of training data is available;

- kernel machine regressions can be seen as a good compromise between the previous two techniques. Similar to the ANN, they allow building non-parametric models, thus mitigating the curse of dimensionality. Moreover, similar to linear basis function regressions, the linear structure of the regression model leads to a convex training problem, which sometimes admits a closed-form solution. For the above reasons, kernel machine regressions are characterized by a promising error convergence with respect to the number of training samples and by a fast training time. Thus making such techniques a good candidate for the construction of surrogate models in the considered modeling scenario.

Chapter 3

Prior Knowledge Based Machine Learning Surrogate Models

As shown in Sec. 2.5.2, for realistic applications the computational cost related to the construction of a surrogate model is not dominated by the actual model training, but by the generation of the training samples. The latter task is usually carried out via a design space exploration based on several simulations performed with the computational expensive computational model. As an example, for EM applications, the training set is usually generated by using many repeated high-fidelity and detailed full-wave EM simulations, thus leading to a possible high computational cost. Therefore, reducing the computational cost for the training set generation can be seen as one of the most important challenge to be addressed in order to efficiently construct a surrogate model.

To this aim, this Chapter presents a promising solution to the above problem based on prior knowledge-based machine learning (PKBML) metamodels. Without loss of generalities, the PKBML surrogate modeling technique is applied on the example presented in Sec. 2.5.2 by considering the prediction of the per-unit-length (p.u.l.) parameters of a hybrid copper-graphene on-chip interconnect as a function of several input parameters. Two different implementations of PKML are considered and combined with the LS-SVM, SVM and ANN regression.

3.1 PKBML Methods

PKBML methods can be seen as a viable solution for reducing the computational cost required by the generation of the training set. The underlying idea is to work with a training set generated as the combination of the results obtained with an accurate high-fidelity computational model and the ones provided by a course low-fidelity one. In the considered application, simulations carried out with accurate and computationally expensive full-wave EM solver will be used as the high-fidelity models, while computationally cheap and approximate empirical formulas will be used as low-fidelity models [24, 35, 17, 36]. The key idea behind PBKML is to develop advanced regression solutions aimed at working in an efficient and accurate way with an heterogeneous training set combining the results obtain by the high- and low-fidelity computational model.

Indeed, since data generation with the low-fidelity model is unavoidably cheaper than a corresponding simulation carried out with the high-fidelity one, the possibility of combining data extracted from both of them enables a better accuracy versus training time compared with conventional training approaches using data from the high-fidelity model alone. The metamodels constructed from such PKBML frameworks are called PKBML metamodels. Specifically, this work presents two PKBML frameworks for hybrid copper-graphene interconnect modeling based on the source difference (SD) technique and the prior knowledge input (PKI) technique.

3.1.1 Source Difference (SD) Technique

The source difference (SD) technique uses the predictions of the low-fidelity model as prior knowledge to accelerate the training of the ML-based surrogate models. In particular, the surrogate model is trained to learn the difference between the predictions of the high-fidelity model $\mathcal{M}(\boldsymbol{x})$ and the low-fidelity model $F_y(\boldsymbol{x})$, as illustrated in Fig. 3.1.

To this aim, let us define the training set used by the SD technique as:

$$\mathcal{D}_{SD} = \{(\boldsymbol{x}_l, E(\boldsymbol{x}_l))\}_{l=1}^{L} \tag{3.1}$$

with

$$E(\boldsymbol{x}_l) = y_l - F_y(\boldsymbol{x}_l) \tag{3.2}$$

where $y_l = \mathcal{M}(\boldsymbol{x}_l)$ is the prediction computed via the high-fidelity model and $F_y(\boldsymbol{x}_l)$ is the corresponding output calculated via the low-fidelity model for the same input parameters $\boldsymbol{x}_l$. This means that both the high- and low-fidelity models should be evaluated on the same set of training input samples.

The error data of Eq. (3.2) is then used as output training samples to train the metamodels. It is important to point out that the variance of the training output samples (i.e., $E(\boldsymbol{x}_l)$) in (3.1), writes:

$$\operatorname{Var}(E(\boldsymbol{x})) = \operatorname{Var}(y(\boldsymbol{x})) + \operatorname{Var}(F_y(\boldsymbol{x})) - 2\operatorname{Cov}(y(\boldsymbol{x}), F_y(\boldsymbol{x})) \tag{3.3}$$

where $\operatorname{Var}(y(\boldsymbol{x}))$ is the variance of the actual output samples predicted by the high-fidelity model, $\operatorname{Var}(F_y(\boldsymbol{x}))$ is the corresponding one related to the low-fidelity model and $\operatorname{Cov}(y(\boldsymbol{x}), F_y(\boldsymbol{x}))$ is covariance between them.

Equation (3.3) shows that as the covariance between the predicted outputs of the low and high-fidelity models increases, the variance of the error quantity in (3.2) decreases. This, in turn, implies that a smaller number of training samples in the dataset $\mathcal{D}_{SD}$ will be sufficient to capture the variability of the error quantity of Eq. (3.2). So, for an appropriate low-fidelity model, the number of training samples required to train a ML metamodel able to emulate the error quantity of Eq. (3.2) will be much lower than required to emulate the true output $y = \mathcal{M}(\boldsymbol{x})$.

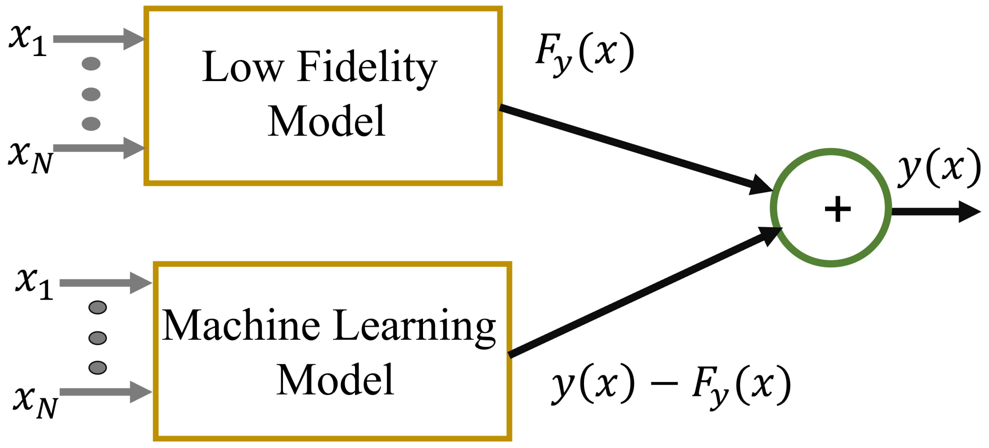

Figure 3.1: Block diagram illustrating how a low-fidelity model and a ML metamodel can be utilized in a SD technique.

Once the error quantity of Eq. (3.2) is emulated by any surrogate model, the true output can be recovered simply as the sum of the predicted outputs from the surrogate and the low-fidelity model as shown Fig. 3.1.

$$y(\boldsymbol{x}) = E(\boldsymbol{x}) + F_y(\boldsymbol{x}) \tag{3.4}$$

Therefore, in summary, when the correlation between the high-fidelity model and the low-fidelity model $F_y(\mathbf{x})$ is high, the SD approach enables a dramatic reduction of the number of training samples required during the training phase [17, 35, 36].

3.1.2 Prior Knowledge Input (PKI) Technique

Based on Eq. (3.4), it is observed that the SD technique assumes a linear correlation between the low and high-fidelity models. However, such a strong assumption might lead to loss of convergence of the SD technique when the correlation between the low and high-fidelity models is actually nonlinear. In such scenarios, the PKI framework shown in Fig. 3.2 provides a promising alternative to SD able to overcome the above limitation [23]. The underlying concept of the PKI technique is to represent the true output as a nonlinear function of the predictions of the low-fidelity model $F_y(\boldsymbol{x})$ to which is added the error quantity previously defined in (3.2), such as:

$$y(\boldsymbol{x}) = G\left(F_y(\boldsymbol{x})\right) + E(\boldsymbol{x}), \tag{3.5}$$

where $G(.)$ is an appropriate nonlinear function to be learn during the regression model training. The expression of (3.5) can be expressed even more compactly using a new nonlinear function $H(.)$ as

$$y(\boldsymbol{x}) = H\left(F_y(\boldsymbol{x}), \boldsymbol{x}\right), \tag{3.6}$$

in which the output of the high-fidelity model $y = \mathcal{M}(\boldsymbol{x})$ is obtained as a nonlinear combination of the input parameters $\boldsymbol{x}$ and the output of the low-fidelity model $F_y(\boldsymbol{x})$. The nonlinear map $H(\cdot)$ in (3.6) can be learnt via a generic ML regression using a new training dataset:

$$\mathcal{D}_{PKI} = \{(\tilde{\boldsymbol{x}}_l, y_l)\}_{l=1}^{L}, \tag{3.7}$$

where the new input space $\tilde{\boldsymbol{x}}$ is augmented as:

$$\tilde{\boldsymbol{x}}_l = \begin{bmatrix} \boldsymbol{x}_l \\ F_y\left(\boldsymbol{x}_l\right) \end{bmatrix}. \tag{3.8}$$

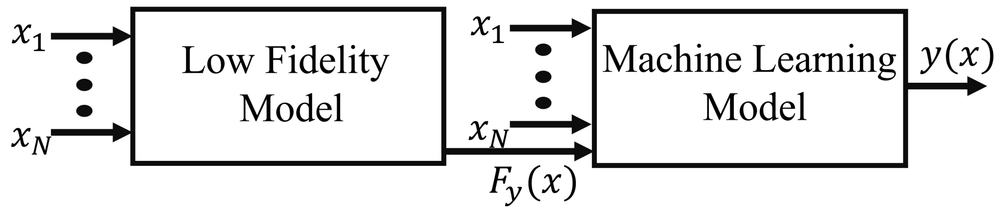

Figure 3.2: Block diagram illustrating how a low-fidelity model and a ML metamodel can be utilized in a PKI technique.

As illustrated in Fig. 3.2, and in (3.6) and (3.8), the PKI formulation uses the prior knowledge of the low-fidelity model as an additional input for the ML surrogate model. This prior knowledge guides the ML metamodel to learn the nonlinear function of Eq. (3.6) using much fewer training data points than what is conventionally required.

3.2 Example: Surrogate Models for Hybrid Copper-Graphene On-Chip Interconnects

In this section, the numerical example of Sec. 2.5.2 is used to demonstrate the advantages of the proposed PKBML techniques of Sec. 3.1, over conventional

ML metamodels and full-wave EM simulations for the signal integrity verification of hybrid copper-graphene interconnects. In this example, at the first step, the SD and PKI metamodels are applied to predict the p.u.l. parameters of hybrid copper-graphene interconnects. Then, the obtained p.u.l. parameters are embedded in SPICE multi-conductor transmission line (MTL) models to perform transient analysis and signal integrity verification of the interconnects.

The performance and the capability of the proposed PKBML modeling framework has been investigated by considering the impact of 8 geometrical parameters on the p.u.l. resistance (R), inductance (L), and capacitance (C) matrices of the 3-conductor hybrid copper graphene interconnect structure shown in Fig. 2.7. Additional details on the considered input parameters are provided in Sec. 2.5.2.

The above structure has been implemented as a parametric simulation in ANSYS Q3D. The ANSYS Q3D Extractor employs a quasi-static 2D EM solver using the finite element method (FEM) technique to extract the p.u.l. capacitance parameters of the network. Next, it employs a quasi-static 3D EM solver using the method of moments (MoM) technique accelerated by the fast multiple method to extract the p.u.l. inductance parameters of the network. For all these calculations, the tool ensures that the calculated values are accurate up-till a user-defined maximum frequency limit of operation. The maximum frequency limit of operation was chosen to be 50GHz. The above implementation will be use hereafter as a high-fidelity computational model and as reference model for the assessing the model accuracy.

3.2.1 Appropriate Low Fidelity Model for Copper Graphene Interconnects

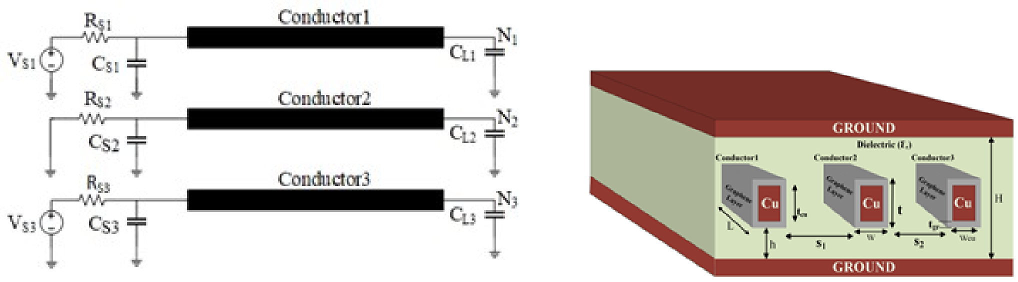

Figure 3.3: Left panel: schematic of the hybrid interconnect network showing the driver and load circuits modeled as linear RC circuits. Right panel: cross-sectional view of the hybrid interconnect structure.

The use of PKBML requires the availability of a low-fidelity computational model. For the considered test case, such low-fidelity model for the R, L, and C

p.u.l. parameters have been built by using the results available in [37]. Specifically, the results presented in [37] allow constructing a closed-form analytical approximation able to efficiently predict the p.u.l. matrices of the considered in hybrid copper-graphene interconnects [35]. The resulting empirical model is then used as low-fidelity model together with the PKI and SD techniques. For the sake of brevity, the detailed description of the empirical models [35, 37] have been avoided.

In order to evaluate the degree of accuracy of the p.u.l. matrices predicted by the considered low-fidelity model, it is important to assess its correlation with the results of the high-fidelity one. Indeed, as pointed out in Sec. 3.1, a higher correlation between the high-fidelity and low-fidelity model will lead to beneficial effects on the performance and effectiveness of the proposed PKBML framework.

Table 3.1: Statistical analysis of the correlation among the prediction of the considered high- and low-fidelity models.

Parameters	High-Fidelity Model		Low-Fidelity Model		Corr.Coef
	Mean	Std.	Mean	Std.	
R_{11}	108.9	10.39	86.08	13.37	0.99
C_{10}	77.37	8.06	61.58	8.67	0.67
C_{12}	114.24	15.2	148.6	22.34	0.97
C_{20}	62.9	7.14	61.25	8.59	0.66
C_{23}	90.80	12.36	110.66	17.08	0.97
C_{30}	84.89	9.01	69.03	9.54	0.69
L_{11}	4.02	0.03	3.48	0.17	-0.2
L_{12}	3.67	0.02	1.86	0.01	0.93
L_{13}	3.45	0.02	1.71	0.01	0.99

Table 3.1 investigates the correlation between the two models by considering 500 random configurations of the input parameters. While the accuracy of the low fidelity models is far from perfect, it can be seen that for R_{11}, C_{12}, C_{23}, L_{12} and L_{13}, there is a very large correlation between the predictions obtained by the two models, while for C_{10}, C_{20} and C_{30}, this correlation is good, and for L_{11} it is poor. The poor correlation observed for the p.u.l. inductance parameters is motivated by a standard deviation of less than 1% of its mean value, thus highlight a small variability of the inductance p.u.l. parameter value as a function of the considered input parameters. Moreover, Fig. 3.4 shows the resulting scatter plots providing a visual understanding of the correlation between low-fidelity and high-fidelity models computed for the same 500 configurations of the input parameters. The scatter plots confirm the observations made before by showing a strong linear correlation between the high- and low-fidelity models for the p.u.l. parameters associated to R_{11}, C_{12}, C_{23}, L_{12} and L_{13}, a quite strong nonlinear correlation for

the C_{10}, C_{20} and C_{30} p.u.l. parameters and a weak non-linear correlation for the L_{11} p.u.l. parameter.

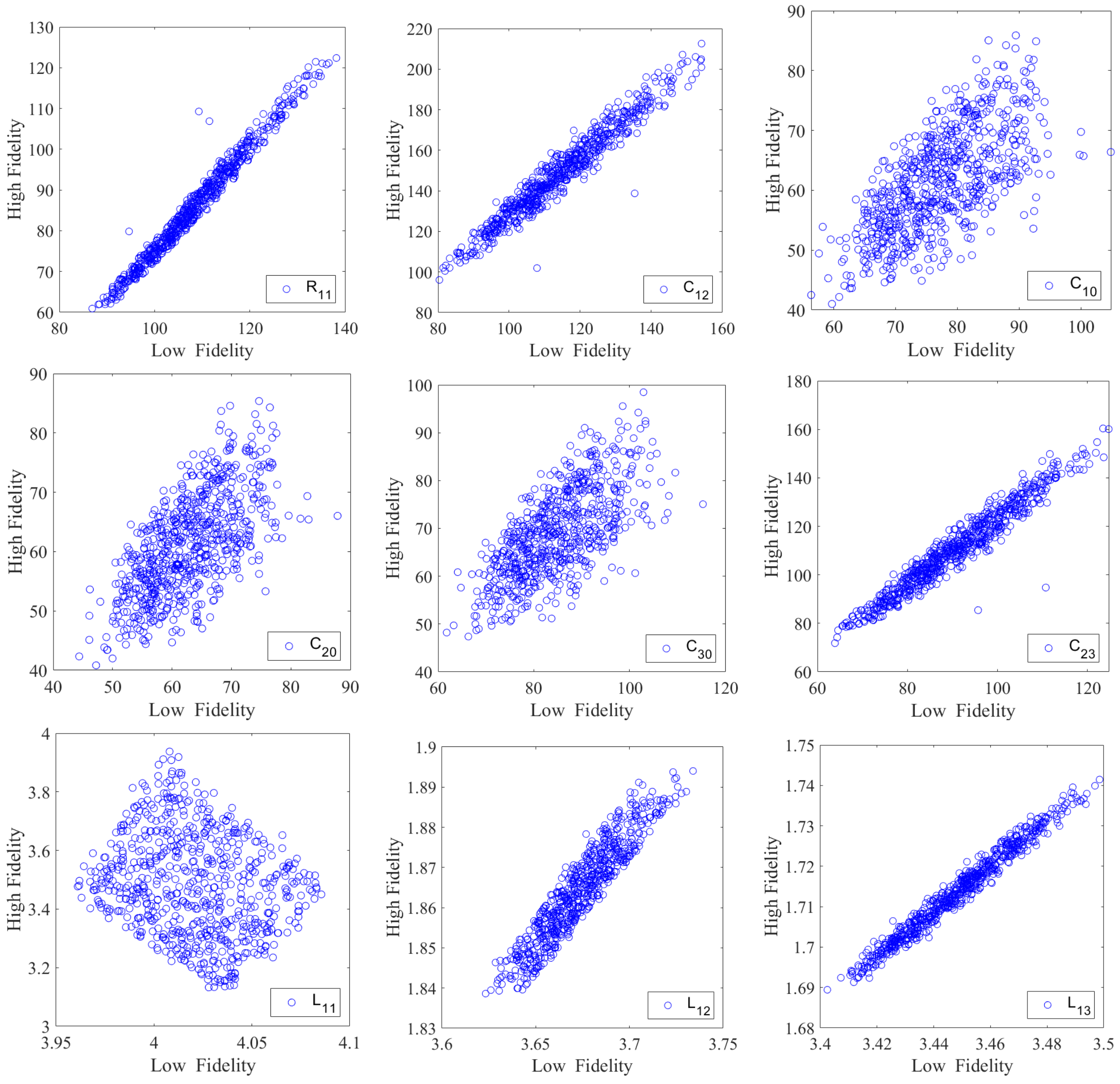

Figure 3.4: Scatter plots showing the correlation between the predictions of the low- and high- fidelity model for the RLC p.u.l. parameters.

3.2.2 Numerical Validations

This section quantifies the benefits of the proposed PKBML metamodels over conventional ML metamodels and full-wave EM simulations for the signal integrity verification of hybrid copper-graphene interconnects by considering the scenario presented in Sec. 2.5.2.

For this purpose, several modeling techniques have been considered and compared, such as: conventional ANN, SVM, and LS-SVM metamodels trained by using the data generated with the high-fidelity model only, and the proposed SD and PKI variants of the ANN, SVM, and LS-SVM metamodels trained using a combination of data obtained from the ANSYS Q3D Extractor tool and the low-fidelity model [37]. All the ML-based metamodels are trained using the same training dataset, where the number of points in the dataset is progressively increased as $L = \{15, 25, 35, 70, 140, 350, 700\}$. All these training points are selected using the LHS scheme. For all of the models, the accuracy has been assessed on the same test set generated via the high-fidelity model consisting of 500 points uniformly distributed over the entire input parameter space.

Figure 3.6 and 3.5 show the the decay of the testing errors computed as the NRMSE, defined in (2.34), for the L_{11}, L_{12}, R_{11}, C_{11}, C_{12}, C_{20}, C_{23} and C_{30} p.u.l. parameters achieved by the considered modeling schemes. From Figure 3.6 and 3.5, we can draw the following observations:

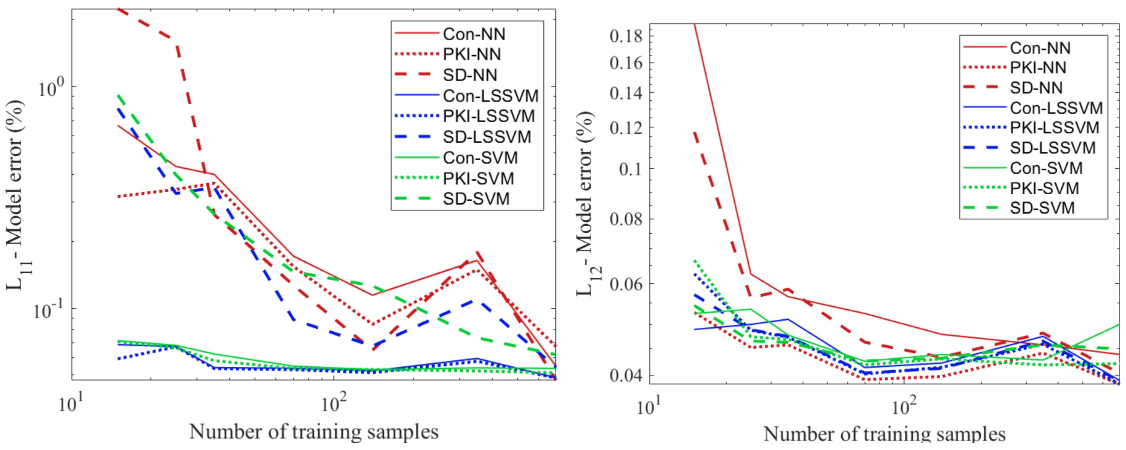

Figure 3.5: Scaling of the testing NRMSE with increasing number of training points for the values of the L_{11} and L_{12} p.u.l. parameters using the different ML metamodel and their PKI and SD variants.

- the SD and PKI formulations converge much faster than their conventional formulations presented in Sec. 2.5.2. This is to be expected given that the SD and PKI approaches exploit the correlation between the results of the low-fidelity model and the full-wave EM simulations.
- For most ML metamodels, the PKI approach shows a faster convergence than the SD approach, except for ANNs. This is also expected given that the PKI approach assumes a general nonlinear correlation between the results of the low-fidelity model and the full-wave EM simulations as opposed to the restrictive linear correlation assumed by the SD approach.

Table 3.2: Accuracy and time cost comparison between different metamodels for training dataset with $L = 35$, 140, and 700 samples

Methods		L=35(Cost=105 Minutes)			L=140(Cost=420 Minutes)			L=700(Cost=2100 Minutes)		
		Ave RMSE(%)	Max RMSE	Training Time (s)	Ave RMSE(%)	Max RMSE	Training Time (s)	Ave RMSE(%)	Max RMSE	Training Time (s)
SVM	CON	0.85	1.57	190	0.63	1.43	241	0.63	1.42	1804
	PKI	0.77	1.56	205	0.62	1.42	253	0.62	1.41	1790
	SD	0.82	1.61	220	0.63	1.41	285	0.62	1.40	2645
LS-SVM	CON	0.81	1.5	2.38	0.62	1.41	5.47	0.64	1.46	57.6
	PKI	0.77	1.56	2.38	0.62	1.41	5.96	0.62	1.41	154
	SD	0.81	1.61	2.07	0.62	1.40	5.41	0.64	1.46	145
ANN	CON	2.28	4.5	3468	1.38	2.79	5994	1.16	2.45	8554
	PKI	1.24	2.42	3213	1	1.84	6287	0.92	1.8	8175
	SD	1.17	2.71	3435	0.90	1.69	5948	0.89	1.68	8329

- For a fixed number of training samples, the SVM and LS-SVM metamodels, whether of the conventional or the SD and PKI variety, are much more accurate than their ANN counterparts. Thus confirming the conclusions drawn in Sec. 2.5.2.

As further proof of the above observations, Tab. 3.2 provides an exhaustive comparison among all the considered surrogate models and p.u.l. parameters in terms of their average and maximum NRMSE computed over all the entries of the p.u.l. matrices and their training time costs for $L = 35$, 140, and 700 samples. The training time costs are divided into two parts: the computational time required to generate the training dataset (given at the top of the table) and the training time. The results of Tab. 3.2 clearly show that for the same number of training samples, the proposed PKI and SD techniques provide a lower testing error compared to the conventional metamodels. This clearly underlines the universal improved convergence of the proposed PKI and SD techniques. In fact, the PKI and SD variants of ANN metamodels are sufficiently well trained using only 35 training samples as opposed to the conventional ANN metamodel that requires roughly 140 (i.e., 4 times larger) number of training samples.

Regarding the computational time costs, Tab. 3.2 shows that the computational cost for the construction of the proposed surrogate models is dominated by the generation of the training set. Indeed, the computational cost required for the generation of the training dataset is undoubtedly higher than the one required for the model training. This is motivated by the fact that for the generation of the training dataset repeated computationally expensive simulations with the high-fidelity modes (i.e., full-wave EM simulations) have to be performed. For example, the computational time cost required by the ANSYS Q3D Extractor tool for generating 35 training samples is around 6300 seconds, which is more than 2 times the time required to train the slowest metamodel. On the other hand, the remarkable

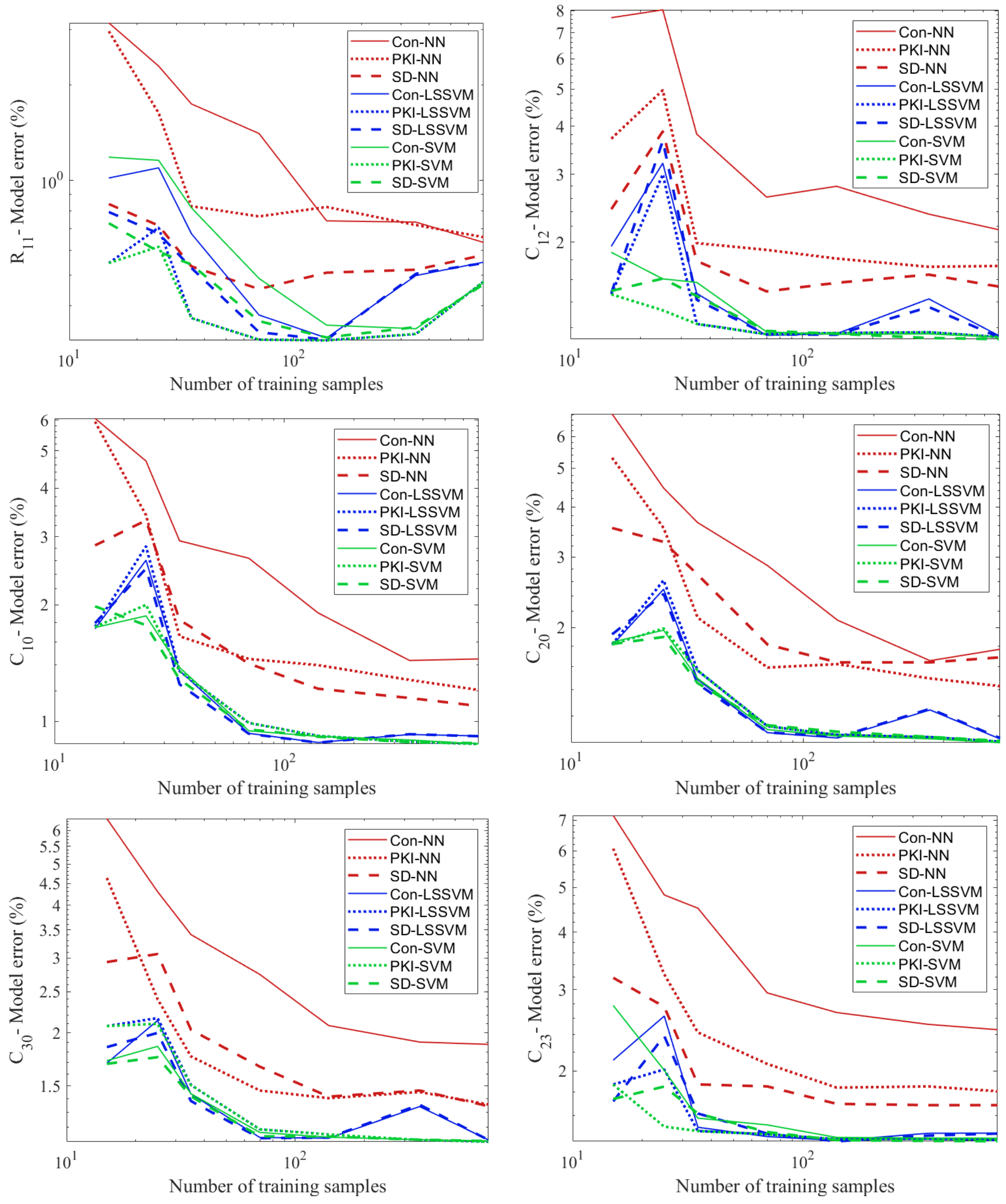

Figure 3.6: Scaling of the testing NRMSE with increasing number of training points for the values of the R_{11}, C_{11}, C_{12}, C_{20}, C_{23} and C_{30} p.u.l. parameters using the different ML metamodel and their PKI and SD variants.

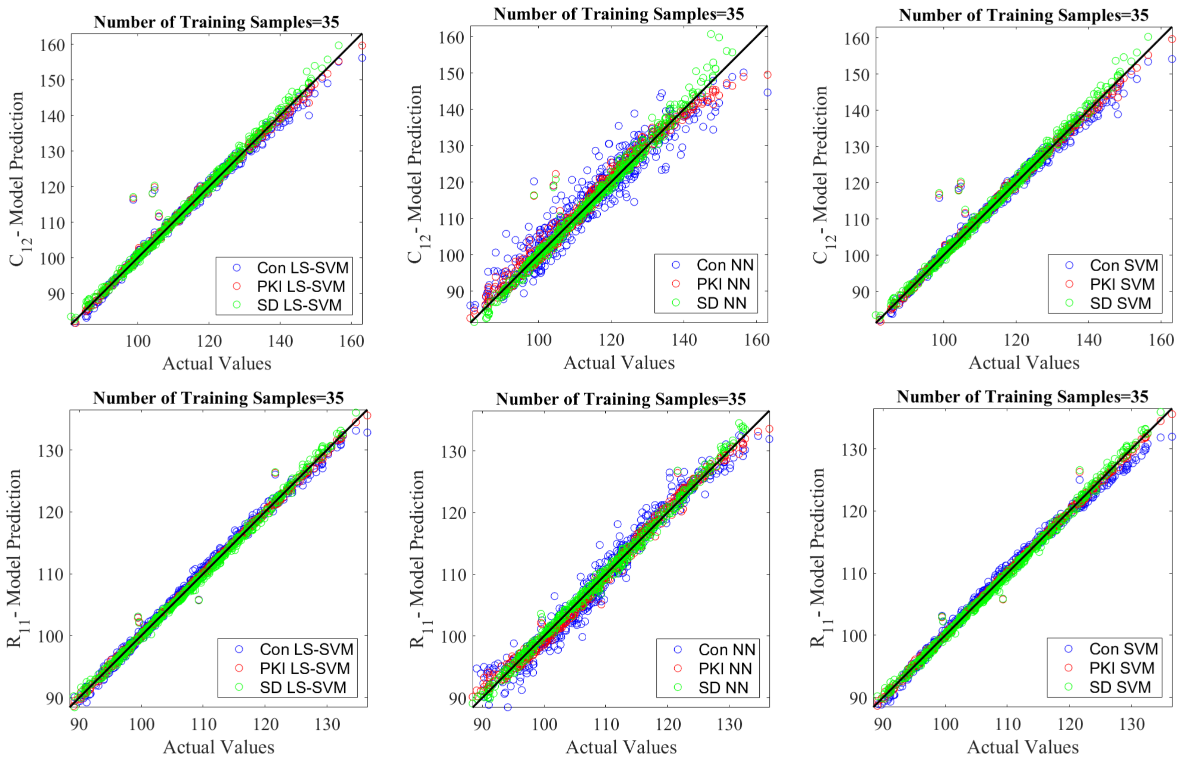

Figure 3.7: Comparison of the scatter plots for the C_{12} and R_{12} p. u. l. parameters computed from the prediction of the proposed metamodels by using as reference the corresponding values computed via a full-wave EM simulation in ANSYS Q3D Extractor.

fact of all these techniques is that the time taken by the metamodels to predict their outputs is extremely small. Indeed, once trained, all of the metamodels take less than one second for predicting the p.u.l. parameter at any design point. This makes these metamodels far better suited than full-wave EM simulators for tasks that require several evaluations, e.g., uncertainty quantification, design space exploration, sensitivity analysis, etc. The results of Tab. 3.2 are further validated using the scatter plots of Fig. 3.7.

The accuracy of the resulting model is then investigated with a SPICE transient analysis of the interconnects structure in Fig. 3.3. To this aim the values of the p.u.l. parameters predicted by the different ML metamodels showing an accuracy below 2% NRMS error threshold for ANN metamodels and 1.5% testing error for the SVM and LS-SVM metamodels are used as input parameters for a SPICE transient analysis. For this purpose, lines 1 and 3 of the interconnects are excited by voltage sources with saturated ramp wave forms of rise time $T_r = 0.1$ ps and amplitude 1 V. Line 2 is the victim line. In Fig. 3.3 (left panel), the values of resistance, $R_{s1} = R_{s1} = R_{s1} = 138.6\,\Omega$. The values of capacitor, $C_{s1} = C_{s2} = C_{s3} = 9.4\,\text{fF}$ and

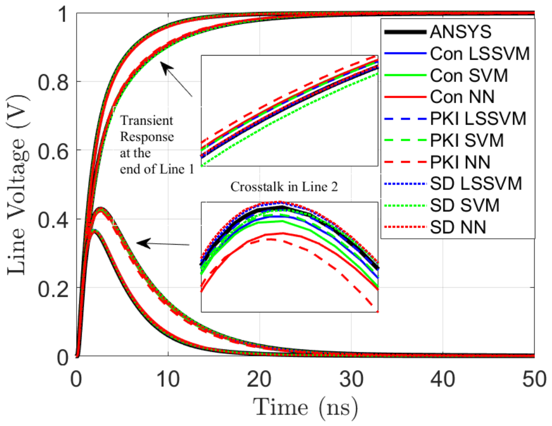

Figure 3.8: Transient responses at the end of line 1 and crosstalk in the at the end of line 2 simulated in SPICE and based on the p. u. l. parameters predicted by the different metamodels for the two different design points.

the values of the capacitor, $C_{L1} = C_{L2} = C_{L3} = 7.78\,\text{fF}$. They provide a suitable linear model for the CMOS driver and load according to the technology node of the interconnects considered. The length of interconnects is set to be $500\,\mu\text{m}$. Outputs of interest for this example are the transient responses at the far-end of line 1 and 2. Figure 3.8 shows the transient responses of the interconnects by using the p.u.l. parameters predicted by the ML metamodels is illustrated for two arbitrary points in the input parameter space. In addition, Fig. 3.8 also includes the transient responses obtained using the p.u.l. parameters extracted from full-wave EM simulations for the same two configurations.

It is noted from Fig. 3.8 that the results obtained from ML metamodels exhibit very good agreement with the corresponding ones obtained from the full-wave EM simulations. This demonstrates that the accelerated training of the ML metamodels provided by the proposed SD and PKI approaches does not lead to any loss in model accuracy either when predicting the p.u.l. parameters of the interconnects or when performing the SPICE simulations.

Further to go into greater detailed analysis of the transient responses shown in Fig. 3.8, with the SPICE simulation setup, a design space exploration is performed using 1000 randomly sampled points. Regarding signal integrity (SI) quantities, the SPICE simulations compute the 50% delay in line 1, 50% delay of line 3 and peak crosstalk of line 2. A summary of this SI data extracted from the transient simulation for all the metamodels is shown in Tab. 3.3. This table shows the minimum and maximum values of the investigated SI quantities obtained among the 1000 simulations. Data shown in Tab. 3.3 clearly indicates that all the proposed metamodels are in good agreement with the full-wave simulation results. Results predicted by conventional metamodels also matches with full wave simulation results as plotted in Fig. 3.8, but are omitted from Tab. 3.3 for the sake of brevity.

Table 3.3: Table for the quantities for PKBML metamodels

SI Quantity	ANSYS		PKI-ANN		SD-ANN		PKI-ASVM		SD-SVM		PKI-LSSVM		SD-LSSVM	
	min	max	min	max	min	max	min	max	min	max	min	max	min	max
50% Delay Line 1 (ns)	1.39	2.19	1.38	2.21	1.32	2.25	1.38	2.22	1.31	2.28	1.38	2.22	1.37	2.22
50% Delay Line 3 (ns)	1.30	2.15	1.32	2.06	1.26	2.20	1.29	2.13	1.31	2.13	1.27	2.14	1.31	2.15
Peak Crosstalk Line 2 (V)	0.34	0.44	0.35	0.42	0.34	0.44	0.34	0.43	0.34	0.44	0.34	0.43	0.34	0.44

3.3 Summary

This chapter investigated two PKBML techniques aimed at improving the performance in terms of accuracy with respect to the number of training samples of ML-based surrogate models, and to reduce overhead and the computational cost required by the generation of the training set. Specifically, two implementations of the PKBML have been investigated, such as the PKI and SD techniques. The improved performance of the proposed modeling framework, as well its advantages over conventional ML approaches have been investigated together with the SVM, LS-SVM and ANN regression for the signal integrity verification of hybrid copper-graphene interconnects for which a low-fidelity model is available. The results clearly highlight the capability of the proposed solution of reducing the computational cost of the training phase. Remarkable results have been achieved for ANN-based regressions.

Chapter 4

Kernel-Machine Regressions in Complex- and Vector-Output Regression Problems

As pointed out in Chapter 2 and 3, kernel-machine regressions have shown interesting features in terms of accuracy and convergence with respect to the number of training samples. Indeed, thanks to their linear structure and to the beneficial effect of the kernel trick, such approaches can be seen as a promising candidate for the development of surrogate models in EM applications with dozens of parameters by using a limited number of training samples. Moreover, as discussed in Chap. 2, for the specific case of kernel-machine regressions with a square loss-function, the regularized ERM can be recast in terms of a standard least-square problem for which the unknowns can be estimated in closed-form, thus making such technique easy to implement and fast to train.

On the other hand, plain formulations of kernel regressions are restricted to real-valued single-output problem. Indeed, a conventional formulation of kernel-based regressions allows to learn any possible nonlinear map between real input parameters and a generic real-valued scalar output of interest. Unfortunately, complex- and vector-valued data are quite common in electronic applications. As an example we can think about building a surrogate model able to approximate the real and imaginary part of the scattering parameters defined at several frequency points of an EM structure as a function of its geometrical and material parameters.

This chapter aims at addressing the above two challenges shared by kernel machine regressions:

- Challenge #1 - Complex Output: presenting a possible generalization of the mathematical framework for kernel-machine regressions to complex-valued problems. The proposed formulation, based on the results presented in [38, 39, 40], is applied to the LS-SVM regression introduced in Chap. 2, even if it can be possible extended to a generic kernel-machine regression.

- Challenge #2: Vector-Valued Output: discussing an efficient modeling scheme for adopting scalar-valued regressions in vector-valued problems based on data compression.

The above aspects will be formally presented and discussed along the Chapter. Moreover, the effectiveness and performance of the proposed solutions will be then investigated on two application examples.

4.1 Challenge #1: Complex-Valued LS-SVM Regression

The simplest strategy for extending the applicability of real-valued ML techniques to the case of complex-valued data is based on the so-called dual-channel formulation [38, 39, 41]. The underlying idea is to recast the complex-valued problem into two uncorrelated real-valued ones, by stacking the real and imaginary part of the complex input and output values. The main advantage of the above procedure is that plain real-valued ML techniques can be directly adopted without requiring any generalization or improvement. However, even if such approach looks quite intuitive, it is usually adopted in a naive way, without any rigorous proof or explanation. Moreover, such approach completely ignores any possible correlation among the real and imaginary part of the complex-valued output, thus leading to possible accuracy and robustness to noise issues [38, 42]. For the above reasons, alternative pure complex-valued formulations have been proposed for several ML techniques, such as ANN [42], SVM regression [43], kernel Ridge regression [38, 39, 41, 44, 40], LS-SVM regression [45] and Gaussian process regression [46].

This Section discusses a possible generalization of the real-valued LS-SVM regressions presented in Chapter 2 to the more general case of complex-data problem. It is important to remark that the proposed mathematical formulation can be easily extended to the case of kernel Ridge regression, and some considerations apply to the SVM regression as well.

Starting from a set of complex-valued dataset $\mathcal{D} = \{(\mathbf{x}_l, y_l)\}_{l=1}^{L}$ where $\mathbf{x}_i \in \mathbb{C}^d$ and $y(\mathbf{x}_i) \in \mathbb{C}$, the primal space formulation of the LS-SVM regression writes:

$$\tilde{\mathcal{M}}(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i^*(\mathbf{x}) + b = \langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}) \rangle + b, \tag{4.1}$$

where $\mathbf{w} = [w_1, \ldots, w_N]^T = \mathbf{w}_R + j\mathbf{w}_I \in \mathbb{C}^N$ are complex regression coefficients (i.e., $w_i = w_{i,R} + jw_{i,I})$, $\mathbf{\Phi}(\mathbf{x}) = \mathbf{\Phi}_R(\mathbf{x}) + j\mathbf{\Phi}_\mathrm{I}(\mathbf{x}) = [\phi_1(\mathbf{x}), \ldots, \phi_N(\mathbf{x})]^T$), is a vector-valued complex function $\mathbf{\Phi}(\cdot) : \mathbb{C}^d \to \mathbb{C}^N$ collecting the complex-valued basis functions $\phi_\mathrm{i}(\mathbf{x})$ that provide a map between the parameter space and the feature space, $b = b_R + jb_I$ is the bias term, and $\langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}) \rangle = \mathbf{\Phi}^\mathrm{H}(\mathbf{x})\mathbf{w} = \mathbf{w}^T \mathbf{\Phi}^*(\mathbf{x})$ is the inner product in the complex domain.

The primal space formulation of the complex-valued regression in Eq. 4.1 can be written in term of its real and imaginary components:

$$\begin{aligned}\tilde{\mathcal{M}}(\boldsymbol{x}) &= \tilde{\mathcal{M}}_R(\boldsymbol{x}) + j\tilde{\mathcal{M}}_I(\mathbf{x}) \\ &= \Big(\boldsymbol{w}_R\boldsymbol{\Phi}_R^T(\mathbf{x}) + \mathbf{w}_I\boldsymbol{\Phi}_I^T(\mathbf{x}) + b_R\Big) + j\left(\mathbf{w}_\mathrm{I}\boldsymbol{\Phi}_R^T(\mathbf{x}) - \boldsymbol{w}_R\boldsymbol{\Phi}_I^T(\mathbf{x}) + b_I\right).\end{aligned} \tag{4.2}$$

In the above primal space formulation, the regression unknowns (i.e., the coefficients $\mathbf{w}_R$ and $\mathbf{w}_I$ and the bias terms b_R and b_I) are estimated by solving the following convex optimization problem:

$$\min_{\mathbf{w}_R,\mathbf{w}_I,b_R,b_I,\mathbf{e}_R,\mathbf{e}_I} \frac{1}{2}\mathbf{w}_R^T\mathbf{w}_R + \frac{1}{2}\mathbf{w}_I^T\mathbf{w}_I + \frac{\gamma_R}{2}\sum_{l=1}^{L} e_{R,l}^2 + \frac{\gamma_I}{2}\sum_{l=1}^{L} e_{I,l}^2, \tag{4.3}$$

such that, for $l = 1, \ldots, L$:

$$\mathbf{e}_{R,l} = \Re\{y_l - \widetilde{\mathcal{M}}\left(\mathbf{x}_l\right)\} = y_{R,l} - \left(\mathbf{w}_R\boldsymbol{\Phi}_R\left(\mathbf{x}_l\right) + \mathbf{w}_I\boldsymbol{\Phi}_I\left(\mathbf{x}_l\right) + b_R\right) \tag{4.4}$$

$$\mathbf{e}_{R,l} = \Im\{y_l - \widetilde{\mathcal{M}}\left(\mathbf{x}_l\right)\} = y_{I,l} - \left(\mathbf{w}_I\boldsymbol{\Phi}_R\left(\mathbf{x}_l\right) - \mathbf{w}_R\boldsymbol{\Phi}_I\left(\mathbf{x}_l\right) + b_I\right) \tag{4.5}$$

where $\boldsymbol{w}_R^T\boldsymbol{w}_R + \boldsymbol{w}_I^T\boldsymbol{w}_I = \|\boldsymbol{w}\|_2^2$ is the L2 regularizer and the terms $e_{R,l}^2$ and $e_{I,l}^2$ provide the error of a squared loss function.

The Lagrangian for the above constraint optimization problem writes:

$$\begin{aligned}&\mathcal{L}\left(\boldsymbol{w}_R, \boldsymbol{w}_I, b_R, b_I, \boldsymbol{e}_R, \boldsymbol{e}_I; \boldsymbol{\alpha}_R, \boldsymbol{\alpha}_I\right) = \\ &= \frac{1}{2}\mathbf{w}_R^T\mathbf{w}_R + \frac{1}{2}\mathbf{w}_I^T\mathbf{w}_I + \frac{\gamma_R}{2}\sum_{l=1}^{L} e_{R,l}^2 + \frac{\gamma_I}{2}\sum_{l=1}^{L} e_{I,l}^2 + \\ &- \sum_{l=1}^{L}\alpha_{R,l}\left\{e_{R,l} - y_{R,l} + \left(\mathbf{w}_R^T\boldsymbol{\Phi}_R\left(\mathbf{x}_l\right) + \mathbf{w}_I^T\boldsymbol{\Phi}_I\left(\mathbf{x}_l\right) + b_R\right)\right\} + \\ &- \sum_{l=1}^{L}\alpha_{I,l}\left\{e_{I,l} - y_{I,l} + \left(\mathbf{w}_I^T\boldsymbol{\Phi}_R\left(\mathbf{x}_l\right) - \mathbf{w}_R^T\boldsymbol{\Phi}_I\left(\mathbf{x}_l\right) + b_I\right)\right\},\end{aligned} \tag{4.6}$$

where $\alpha_l = \alpha_{R,l} + j\alpha_{I,l}$ are the Lagrangian multipliers such that $\alpha_{R,l}, \alpha_{l,l} \geq 0$ for $l = 1, \ldots, L$.

By computing the partial derivatives of the Lagrangian $\mathcal{L}\left(w_R, w_I, b_R, b_I, e_R, e_I; \alpha_R, \alpha_I\right)$ with respect to its parameters:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}_R} = 0 \rightarrow \mathbf{w}_R = \sum_{l=1}^{L}\left[\alpha_{R,l}\boldsymbol{\Phi}_R\left(\mathbf{x}_l\right) - \alpha_{I,l}\boldsymbol{\Phi}_I\left(\mathbf{x}_l\right)\right] \tag{4.7}$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}_I} = 0 \rightarrow \mathbf{w}_I = \sum_{l=1}^{L}\left[\alpha_{R,l}\boldsymbol{\Phi}_I\left(\mathbf{x}_l\right) + \alpha_{I,l}\boldsymbol{\Phi}_R\left(\mathbf{x}_l\right)\right] \tag{4.8}$$

$$\frac{\partial \mathcal{L}}{\partial b_R} = 0 \rightarrow \sum_{l=1}^{L} \alpha_{R,l} = 0 \tag{4.9}$$

$$\frac{\partial \mathcal{L}}{\partial b_I} = 0 \rightarrow \sum_{l=1}^{L} \alpha_{I,l} = 0 \tag{4.10}$$

$$\frac{\partial \mathcal{L}}{\partial e_{R,l}} = 0 \rightarrow \gamma_R e_{R,l} = \alpha_{R,l} \tag{4.11}$$

$$\frac{\partial \mathcal{L}}{\partial e_{I,l}} = 0 \rightarrow \gamma_I e_{I,l} = \alpha_{I,l} \tag{4.12}$$

$$\frac{\partial \mathcal{L}}{\partial \alpha_{R,l}} = 0 \rightarrow e_{R,l} - y_{R,l} + \left(\mathbf{w}_R^T \Phi_R\left(\mathrm{x}_l\right) + \mathrm{w}_I^T \Phi_I\left(\mathrm{x}_l\right) + b_R\right) = 0 \tag{4.13}$$

$$\frac{\partial \mathcal{L}}{\partial \alpha_{I,l}} = 0 \rightarrow e_{I,l} - y_{I,l} + \left(\mathrm{w}_I^T \Phi_R\left(\mathrm{x}_l\right) - \mathrm{w}_R^T \Phi_I\left(\mathrm{x}_l\right) + b_I\right) = 0 \tag{4.14}$$

for $l = 1, \ldots, L$.

By substituting Eq. 4.7, 4.8, 4.9, 4.10, 4.11 and 4.12 in Eq. 4.13 and 4.14, we get the following linear system of equations:

$$\begin{cases} \frac{\alpha_{R,l}}{\gamma_R} - y_{R,l} + \sum_{i=1}^{L} \left[\alpha_{R,i} \mathbf{\Phi}_{\mathrm{R}}\left(\mathbf{x}_i\right) - \alpha_{I,i} \mathbf{\Phi}_I\left(\mathbf{x}_i\right)\right]^T \mathbf{\Phi}_R\left(\mathbf{x}_l\right) + \\ + \sum_{i=1}^{L} \left[\alpha_{R,i} \mathbf{\Phi}_{\mathrm{I}}\left(\mathbf{x}_i\right) + \alpha_{I,i} \mathbf{\Phi}_R\left(\mathbf{x}_i\right)\right]^T \mathbf{\Phi}_I\left(\mathbf{x}_l\right) + b_R = 0 \\ \frac{\alpha_{I,l}}{\gamma_I} - y_{I,l} + \sum_{i=1}^{L} \left[\alpha_{R,i} \mathbf{\Phi}_{\mathrm{I}}\left(\mathbf{x}_i\right) + \alpha_{I,i} \mathbf{\Phi}_R\left(\mathbf{x}_i\right)\right]^T \mathbf{\Phi}_R\left(\mathbf{x}_l\right) + \\ - \sum_{i=1}^{L} \left[\alpha_{R,i} \mathbf{\Phi}_{\mathrm{R}}\left(\mathbf{x}_i\right) - \alpha_{I,i} \mathbf{\Phi}_I\left(\mathbf{x}_i\right)\right]^T \mathbf{\Phi}_I\left(\mathbf{x}_l\right) + b_I = 0 \\ \sum_{l=1}^{L} \alpha_{R,l} = 0 \\ \sum_{l=1}^{L} \alpha_{I,l} = 0 \end{cases} \tag{4.15}$$

for $l = 1, \ldots, L$.

The above system of equations is the dual-form representation of the optimization problem in Eq. 4.3, in which the original regression coefficients collected in the vectors $\mathbf{w}_R$ and $\mathbf{w}_I$ are replaced by the Lagrangian multipliers $\boldsymbol{\alpha}_R$ and $\boldsymbol{\alpha}_I$. It is important to remark that similar to the plain real-valued LS-SVM regression, although the number of unknowns in the primal space (i.e., the dimensionality of $|\mathbf{w}| = N$) is given by the number of basis functions, for the above dual formulation the number of unknowns (i.e., the Lagrangian multipliers collected in the vectors $\alpha = \alpha_R + j\alpha_I$) is always equal to the number of the training samples L. This means the resulting model is non-parametric, i.e., a model in which its complexity is independent from the number of both the input parameters and the basis functions. In order to define the kernel function and the dual formulation of the complex-valued LS-SVM, the first two equations in Eq. 4.15 can be rewritten as follows:

$$\frac{\alpha_{R,l}}{\gamma_R} - y_{R,l} + \sum_{i=1}^{L} \alpha_{R,i} \left[\boldsymbol{\Phi}_{\mathrm{R}}^T(\mathbf{x}_i)\,\boldsymbol{\Phi}_R(\mathbf{x}_l) + \boldsymbol{\Phi}_{\mathrm{I}}^T(\mathbf{x}_i)\,\boldsymbol{\Phi}_{\mathrm{I}}(\mathbf{x}_l)\right] + \\ + \sum_{i=1}^{L} \alpha_{1,i} \left[\boldsymbol{\Phi}_R^{\mathrm{T}}(\mathbf{x}_i)\,\boldsymbol{\Phi}_I(\mathbf{x}_l) - \boldsymbol{\Phi}_{\mathrm{I}}^T(\mathbf{x}_i)\,\boldsymbol{\Phi}_R(\mathbf{x}_l)\right] + b_R = 0 \tag{4.16}$$

and

$$\frac{\alpha_{I,l}}{\gamma_I} - y_{I,l} + \sum_{i=1}^{L} \alpha_{R,i} \left[\boldsymbol{\Phi}_{\mathrm{I}}^{\mathrm{T}}(\mathbf{x}_i)\,\boldsymbol{\Phi}_R(\mathbf{x}_l) - \boldsymbol{\Phi}_{\mathrm{R}}^{\mathrm{T}}(\mathbf{x}_i)\,\boldsymbol{\Phi}_I(\mathbf{x}_l)\right] + \\ + \sum_{i=1}^{L} \alpha_{I,i} \left[\boldsymbol{\Phi}_R^{\mathrm{T}}(\mathbf{x}_i)\,\boldsymbol{\Phi}_R(\mathbf{x}_l) + \boldsymbol{\Phi}_I^T(\mathbf{x}_i)\,\boldsymbol{\Phi}_I(\mathbf{x}_l)\right] + b_I = 0 \tag{4.17}$$

For $l = 1, \ldots, L$.

Now, let us define a complex-valued kernel $k_c(\mathbf{x}, \mathbf{x}')$:

$$\begin{aligned} \boldsymbol{k}_c(\mathbf{x}, \mathbf{x}') &= \langle \boldsymbol{\Phi}(\mathbf{x}), \boldsymbol{\Phi}(\mathbf{x}') \rangle \\ &= \boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{x}) \cdot \boldsymbol{\Phi}^*(\mathbf{x}') \\ &= (\boldsymbol{\Phi}_R(\mathbf{x}) + j\Phi_I(\mathbf{x}))^T \cdot (\Phi_R(\mathbf{x}') - j\boldsymbol{\Phi}_I(\mathbf{x}')) \\ &= \left[\boldsymbol{\Phi}_R^T(\mathbf{x})\boldsymbol{\Phi}_R(\mathbf{x}') + \boldsymbol{\Phi}_{\mathbf{I}}^T(\mathbf{x})\boldsymbol{\Phi}_I(\mathbf{x}')\right] + \boldsymbol{j}\left[\boldsymbol{\Phi}_I^T(\mathbf{x})\boldsymbol{\Phi}_{\mathbf{R}}(\mathbf{x}') - \boldsymbol{\Phi}_{\mathbf{R}}^T(\mathbf{x})\boldsymbol{\Phi}_I(\mathbf{x}')\right] \\ &= \boldsymbol{k}_{\mathbb{R}}(\mathbf{x}, \mathbf{x}') + j\boldsymbol{k}_{\mathrm{I}}(\mathbf{x}, \mathbf{x}') \end{aligned} \tag{4.18}$$

Similar to the real-valued formulation, the kernel function is defined as the inner product in the complex feature space of the basis functions evaluated at $\mathbf{x}$ and $\mathbf{x}'$, where $k_{\mathbb{R}}(\mathbf{x}, \mathbf{x}') = \boldsymbol{\Phi}_R^T(\mathbf{x})\boldsymbol{\Phi}_R(\mathbf{x}') + \boldsymbol{\Phi}_{\mathrm{I}}^T(\mathbf{x})\boldsymbol{\Phi}_I(\mathbf{x}')$ and $k_{\mathbb{I}}(\mathbf{x}, \mathbf{x}') = \boldsymbol{\Phi}_I^T(\mathbf{x})\boldsymbol{\Phi}_{\mathrm{R}}(\mathbf{x}') - \boldsymbol{\Phi}_{\mathrm{R}}^T(\mathbf{x})\boldsymbol{\Phi}_I(\mathbf{x}')$.

Using the above definition, Eq. 4.16 and Eq. 4.17 can be rewritten as follows:

$$\frac{\alpha_{R,l}}{\gamma_R} - y_{R,l} + \sum_{i=1}^{L} \left[\alpha_{R,i} k_{\mathbb{R}}(\mathbf{x}_i, \mathbf{x}_l) - \alpha_{I,i} k_{\mathbb{I}}(\mathbf{x}_i, \mathbf{x}_l)\right] + b_R = 0 \tag{4.19}$$

$$\frac{\alpha_{I,l}}{\gamma_I} - y_{I,l} + \sum_{i=1}^{L} \left[\alpha_{I,i} k_{\mathbb{R}}(\mathbf{x}_i, \mathbf{x}_l) + \alpha_{R,i} k_{\mathbb{I}}(\mathbf{x}_i, \mathbf{x}_l)\right] + b_I = 0 \tag{4.20}$$

for $l = 1, \ldots, L$.

In the above equations, the regression unknowns can be computed by solving the following linear system:

$$\begin{bmatrix} \boldsymbol{K}_{RR} + \frac{\boldsymbol{I}_L}{\gamma_R} & \boldsymbol{K}_{RI} & \mathbf{1} & \mathbf{0} \\ \boldsymbol{K}_{IR} & \boldsymbol{K}_{II} + \frac{\boldsymbol{I}_L}{\gamma_I} & \mathbf{0} & \mathbf{1} \\ \mathbf{1}^T & \mathbf{0}^T & 0 & 0 \\ \mathbf{0}^T & \mathbf{1}^T & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_R \\ \boldsymbol{\alpha}_I \\ b_R \\ b_I \end{bmatrix} = \begin{bmatrix} \mathbf{y}_R \\ \mathbf{y}_I \\ 0 \\ 0 \end{bmatrix} \tag{4.21}$$

where $\mathbf{I}_L$ is the identity matrix of size $L \times L$, $\mathbf{1}^T = [1, \dots, 1]^T \in \mathbb{R}^{1\times L}, \mathbf{0^T} = [0, \dots, 0]^T \in \mathbb{R}^{1\times L}$, $\boldsymbol{\alpha}_R, \boldsymbol{\alpha}_I \in \mathbb{R}^{L\times 1}$ are vectors collecting the real and imaginary part of the regression coefficients, $b = b_R + jb_I$ is the bias term and $\boldsymbol{K}_{RR}, \boldsymbol{K}_{RI}, \boldsymbol{K}_{IR}, \boldsymbol{K}_{II} \in \mathbb{R}^{L\times L}$ are kernel matrices defined as:

$$K_{RR}^{(i,j)} = K_{II}^{(i,j)} = k_{\mathbb{R}}(\mathbf{x}_i, \mathbf{x}_j) \tag{4.22}$$

$$K_{IR}^{(i,j)} = -K_{RI}^{(i,j)} = k_{\mathbb{I}}(\mathbf{x}_i, \mathbf{x}_j) \tag{4.23}$$

for any $i, j = 1, \dots, L$, where the parameters γ_R and γ_I are the regularizer hyper-parameters tuned by the user and provide a trade-off between the model flatness and its accuracy [25].

Similar to the real-valued case, the dual space formulation for complex-valued of LS-SVM regression writes:

$$\mathrm{y}(\mathrm{x}) = \sum_{l=1}^{L} \alpha_l \mathrm{k}_C(\mathbf{x}_l, \mathbf{x}) + b. \tag{4.24}$$

By substituting Eq. 4.18, this gives:

$$\mathrm{y}(\mathbf{x}) = \sum_{l=1}^{L} \left(\alpha_{R,l}\mathrm{k}_{\mathbb{R}}(\mathbf{x}_l, \mathbf{x}) - \alpha_{I,l}\mathrm{k}_{\mathbb{I}}(\mathbf{x}_l, \mathbf{x}) + b_R\right) \tag{4.25}$$

$$+ j\left(\alpha_{R,l}\mathrm{k}_{\mathbb{I}}(\mathbf{x}_l, \mathbf{x}) + \alpha_{I,l}\mathrm{k}_{\mathbb{R}}(\mathbf{x}_l, \mathbf{x}) + b_I\right). \tag{4.26}$$

From the above formulation, it is clear that by means of the complex kernel k_C, the complex-valued LS-SVM regression in the dual space is able to account for possible correlation between the real and imaginary part of $y(\mathbf{x})$.

4.1.1 Complex-Valued Kernel

There are several strategies to construct a complex kernel k_C. Here, we will investigate two of them, the independent kernel, referred as the complex valued complex function (CVCF) [38], and the pseudo kernel, referred to as the pseudo complex-valued function (PCF) [47, 48]. A generic CVCF kernel can be constructed starting from a real-valued kernel k_R as follows:

$$\mathrm{k}_{\mathrm{C}}(\mathbf{x}, \mathbf{x}') = \mathrm{k}_{\mathbb{R}}(\mathbf{x}_R, \mathbf{x}'_{\mathrm{R}}) + \mathrm{k}_{\mathbb{R}}(\mathbf{x}_I, \mathbf{x}'_I) + j\left(\mathrm{k}_{\mathbb{R}}(\mathbf{x}_R, \mathbf{x}'_I) + \mathrm{k}_{\mathbb{R}}(\mathbf{x}_I, \mathbf{x}'_{\mathrm{R}})\right), \tag{4.27}$$

The above complex kernel is fully compliant with the definition provided in Eq. 4.18. The real kernel k_R can be any real kernel function, e.g., linear, radial basis function (RBF) and polynomial kernel. Hereafter, for the CVCF kernel we will use k_R as the RBF kernel, i.e.,

$$\mathrm{k}_{\mathbb{R}}\left(\mathbf{x},\mathbf{x}'\right) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{x}'\|^2\right) \tag{4.28}$$

where σ is the kernel hyperparameter, which will be tuned, along with the regularizer hyperparameters, during the model training by combining cross validation (CV) with a Bayesian optimizer [49, 50]. As an alternative, a family of kernels based on the PCF can be suitably generated from the isotropic complex covariance function, such that (additional mathematical details are provided in [38, 48]):

$$\mathrm{k_c}\left(\mathbf{x},\mathbf{x}'\right) = \cos\left(c\left\|\mathbf{x}-\mathbf{x}'\right\|\right) k_{\mathbb{R}}\left(\mathbf{x},\mathbf{x}'\right) + \mathrm{j}\sin\left(c\left\|\mathbf{x}-\mathbf{x}'\right\|\right) \mathrm{k}_{\mathbb{R}}\left(\mathbf{x},\mathbf{x}'\right), \tag{4.29}$$

where $\mathrm{k}_{\mathbb{R}}\left(\mathbf{x},\mathbf{x}_{\boldsymbol{k}}\right)$ can be selected as any kernel function and c is a new hyperparameter.

In this specific case, a rational quadratic kernel is adopted:

$$\mathrm{k}_{\mathbb{R}}\left(\mathbf{x},\mathbf{x}'\right) = \sigma^2\left(1+\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2al^2}\right)^{-a} \tag{4.30}$$

where a, l and σ are additional hyperparameters.

4.1.2 Dual Channel Kernel (DCK) LS-SVM for Complex-Valued Data

The dual channel kernel (DCK) formulation can be seen as a special case of the general mathematical framework presented in the Section. 4.1.1. The underlying idea is to recast the complex variables in terms of their real and imaginary part and to work with a standard real kernel, i.e., $k_c = k_{\mathbb{R}} : \mathbb{R}^{d\times d} \to \mathbb{R}$.

Under the above assumption, the regression problem in Eq. 4.21 can be simplified as follows:

$$\begin{bmatrix} \boldsymbol{K}_{RR}+\frac{\boldsymbol{I}_L}{\gamma_R} & \mathbf{0}_L & \mathbf{1} & \mathbf{0} \\ \mathbf{0}_{\boldsymbol{L}} & \boldsymbol{K}_{RR}+\frac{\boldsymbol{I}_L}{\gamma_I} & \mathbf{0} & \mathbf{1} \\ \mathbf{1}^{\boldsymbol{T}} & \mathbf{0}^{\boldsymbol{T}} & 0 & 0 \\ \mathbf{0}^{\boldsymbol{T}} & \mathbf{1}^{\boldsymbol{T}} & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_{\boldsymbol{R}} \\ \boldsymbol{\alpha}_{\boldsymbol{I}} \\ b_R \\ b_I \end{bmatrix} = \begin{bmatrix} \mathbf{y}_R \\ \mathbf{y}_{\boldsymbol{I}} \\ 0 \\ 0 \end{bmatrix}, \tag{4.31}$$

where $\mathbf{0}_L$ is the $L\times L$ null matrix, $\boldsymbol{K}_{RR} \in \mathbb{R}^{L\times L}$, such that $K_{RR}^{(i,j)} = k_{\mathbb{R}}\left(\mathbf{x}_i,\mathbf{x}_j\right)$, whilst the matrices $\boldsymbol{K}_{IR} = -\boldsymbol{K}_{RI} = \mathbf{0}_L$.

It is important to remark that, in the above formulation, there is no coupling between the real and imaginary coefficients α_R and α_I, and bias terms $\mathbf{b}_R$ and $\mathbf{b}_I$. Indeed, the solution of the above linear system is equivalent to solving two decoupled ones, accounting for the real and imaginary parts of the regression unknowns independently, such as:

$$\begin{bmatrix} \boldsymbol{K}_{RR}+\frac{\boldsymbol{I_L}}{\gamma_R} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha_R} \\ b_R \end{bmatrix} = \begin{bmatrix} \boldsymbol{y_R} \\ 0 \end{bmatrix} \tag{4.32}$$

$$\begin{bmatrix} \boldsymbol{K}_{RR}+\frac{\boldsymbol{I_L}}{\gamma_I} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha_I} \\ b_I \end{bmatrix} = \begin{bmatrix} \boldsymbol{y_I} \\ 0 \end{bmatrix}. \tag{4.33}$$

In this chapter, a standard RBF Gaussian kernel is considered as real kernel $k_{\mathbb{R}}(\mathbf{x}_i, \mathbf{x}_j)$. In the above scenario, the dual space formulation of the LS-SVM is written [51]:

$$\mathrm{y_R}(\mathbf{x}) = \sum_{l=1}^{L} \alpha_{R,l} k_{\mathbb{R}}(\mathbf{x}_l, \mathbf{x}) + b_R, \tag{4.34}$$

and

$$\mathrm{y_I}(\mathbf{x}) = \sum_{l=1}^{L} \alpha_{I,l} k_{\mathbb{R}}(\mathbf{x}_l, \mathbf{x}) + b_I. \tag{4.35}$$

It is important to note that, in the above formulation, the model for the real and imaginary part of y are built separately, thus ignoring any possible correlation between them. Furthermore, the above models can be trained using the LS-SVM Lab toolbox for the LS-SVM available in MATLAB [31].

4.2 Challenge #2: Vector-valued Surrogate model based on data-Compression

Let us now consider a training set $\mathcal{D} = \{(\mathbf{x}_i, y_i(f_k))\}_{i,k=1}^{L,K}$, where $\mathbf{x}_\mathrm{i} = [x_{i,1}, \ldots, x_{i,d}]^T$, with $\mathcal{X} \in \mathbb{C}^d$, is a vector collecting the i-th configuration of the input parameters and $y_i(f_k) \in \mathbb{C}$ is the corresponding output computed by a full-computational model (i.e., $y_i(f_k) = \mathcal{M}(f_k; \mathbf{x}_i)$) for a set of values of the independent variable f_k (e.g., the frequency points or time instants). Given the information provided by the training set, our goal is to build a surrogate model $\tilde{\mathcal{M}}$ that approximates the training data and is able to generalize well on the "unseen" test samples, such as:

$$y(f_k; \mathbf{x}) \approx \tilde{\mathcal{M}}(f_k; \mathbf{x}), \tag{4.36}$$

for $k = 1, \ldots, K$.

It is important to remark that also in this case the symbol "$\approx$" has been used in an informal way with the aim of providing a more "practical" and "engineering oriented" understanding of the problem. The problem in Eq. (4.36) can be interpreted in term of learning K complex-valued functions $\tilde{\mathcal{M}}(f_k; \mathbf{x})$, with $k = 1, \ldots, K$.

In the above scenario, a possible modeling solution consists of considering the free variable f_k as an extra input parameter. This means that we are seeking a "single" model able to represent in a closed-form, the impact of both the input parameters $\boldsymbol{x}$ and the frequency points f_k on the system output. As an example, such a model can be obtained via a plain or recurrent ANN [19, 9]. However, as pointed out in Chap. 2 despite their flexibility, the training of ANN-based structures requires the solution of a non-convex optimization leading to training issues such as: expensive training time and/or huge number of training samples [9, 23].

As an alternative, the above problem can be tackled via a scalar-valued regression, but it would require to train a possible *huge* number of uncorrelated scalar-output models, one for each output components, thus making the training process extremely expensive and cumbersome when a large number of frequency or time points are considered. Moreover, the above approach unavoidably ignores any correlation among output components, compromising its accuracy and robustness to noise [38].

A clever workaround to the above issues consists in compressing the output-dimension via a compression technique such as the principal component analysis (PCA). The resulting compressed representation of the output components allows to heavily reduce the number of single output regression problems to be solved, with beneficial effects on the training cost [52]. Specifically, the PCA [53, 54, 55, 56] allows extracting the inherent correlation existing among several realizations of output data samples at different frequency or time points, thus leading to a compressed representation of the frequency spectra or wave forms. In such a scenario, the number of actual single-output models required to represent the data can be heavily reduced.

To this aim let us recast the output dataset $\{y_i\left(f_k\right)\}_{i,k=1}^{L,K}$ collecting L realizations computed for different configurations of the input parameters (i.e., the number of training outputs), each having K output components as $K \times L$ matrix $\mathbf{Y}$, such that the element $Y_{i,k} = y_i(f_k)$. For normalization purposes, a zero-mean matrix is considered:

$$\widetilde{\mathbf{Y}} = \mathbf{Y} - \mu \tag{4.37}$$

where μ is the mean value of $\mathbf{Y}$, calculated row-wise and subtracted column-wise.

The new rectangular matrix $\widetilde{\mathbf{Y}} \in \mathbb{C}^{K\times L}$ can be decomposed via the singular value decomposition (SVD) [54, 56]:

$$\widetilde{\mathbf{Y}} = \mathbf{USV}^{\mathrm{H}} \tag{4.38}$$

where, assuming that there are L columns of $\mathbf{U}$ and $\mathbf{V}$ associated with non-zero singular values, $\mathbf{U} = [\mathbf{u_1}, \ldots \mathbf{u_L}] \in \mathbb{C}^{K\times L}$,$\mathbf{V} = [v_1, \ldots, v_L] \in \mathbb{C}^{L\times L}$ are orthogonal matrices, such that $\mathbf{U}^{\mathrm{H}}\mathbf{U} = \mathbf{V}^{\mathrm{H}}\mathbf{V} = \boldsymbol{I}_{L\times L}$, collecting the left and right singular vectors, and $\mathbf{S} = \operatorname{diag}\{(\sigma_1, \ldots \sigma_{\mathrm{L}})\} \in \mathbb{C}^{L\times L}$ is a diagonal matrix collecting the singular values sorted in a decreasing order $\sigma_1 \gg \sigma_2 \gg \cdots \gg \sigma_L$.

Now, a compressed approximation of the actual matrix $\widetilde{\mathbf{Y}}$ can be obtained as follows:

$$\widetilde{\mathbf{Y}} \approx \widetilde{\mathbf{U}}\widetilde{\mathbf{S}}\widetilde{\mathbf{V}}^{\mathrm{H}} \tag{4.39}$$

where $\widetilde{\mathbf{U}} = [\mathbf{u}_1, \ldots, \mathbf{u}_n] \in \mathbb{C}^{K\times\tilde{n}}$ with $\mathbf{u}_i \in \mathbb{C}^{K\times 1}$ and $\widetilde{\mathbf{V}} = [\mathbf{v}_1, \ldots, \mathbf{v}_n] \in \mathbb{C}^{\tilde{n}\times L}$ with $\mathbf{v}_i \in \mathbb{C}^{L\times 1}$ are the reduced left and right-eigenvector matrices collecting only the first $\tilde{n}$ components (i.e., the first $\tilde{n}$ columns of the original matrices $\mathbf{U}$ and $\mathbf{V}$, and $\widetilde{\mathbf{S}} = \operatorname{diag}\{(\sigma_1, \ldots, \sigma_{\tilde{n}})\}$ is a reduced diagonal matrix containing the first $\tilde{n}$ singular values.

The above relationship can be used to obtain a compressed representation $\mathbf{Z} \in \mathbb{C}^{\tilde{n}\times L}$ of the original matrix $\mathbf{Y}$, such that:

$$\mathbf{Z} = \widetilde{\mathbf{U}}^{H}\widetilde{\mathbf{Y}} = \widetilde{\mathbf{S}}\mathbf{V}^{H}. \tag{4.40}$$

It is important to note the resulting compressed matrix $\mathbf{Z} \in \mathbb{C}^{\tilde{n}\times L}$ is smaller than the original matrix $\widetilde{\mathbf{Y}}$, since usually $\tilde{n} \ll K$. Moreover, the rows of the compressed matrix $\mathbf{Z}$ can be considered to be the realizations of a new set of output variables $\{\mathbf{z}(\mathbf{x}_l)\}_{l=1}^{L}$, with $\boldsymbol{z}(\boldsymbol{x}_l) \in \mathbb{C}^{\tilde{n}\times 1}$, which can be considered as the collection of L samples of a compressed $\tilde{n}$-dimensional output variable.

Specifically, the (i, j)-element of the matrix $\mathbf{Z}$, corresponds to the i-th output of the compressed representation evaluated at the j-th configuration of the input parameters, i.e., $Z_{ij} = z_i(\mathbf{x}_j)$. This means that only $\tilde{n}$ single-output models need to be trained to represent the whole dataset provided by the matrix $\mathbf{Y}$, thus leading to a substantial improvement in the training time.

Once a surrogate model for each of the $\tilde{n}$ components of the compressed multi-output representation in $\mathbf{Z}$ is available, the overall compress surrogate can be inexpensively used to predict the system output $\mathbf{y}(\overline{\mathbf{x}}) \in \mathbb{C}^{K\times 1}$ for a generic test sample $\overline{\mathbf{x}} \in \mathcal{X}$ as follows:

$$\boldsymbol{y}(\overline{\mathbf{x}}) \approx \mu + \widetilde{\mathbf{U}}\bar{Z} \tag{4.41}$$

where $\overline{\boldsymbol{Z}} = [z_1(\overline{\mathbf{x}}), \ldots, z_{\tilde{n}}(\overline{\mathbf{x}})] \in \mathbb{C}^{1\times\tilde{n}}$.

It is important to remark that the compressed representation $\boldsymbol{z}(\boldsymbol{x})$ provided by the PCA compression allows approximating the actual data $\boldsymbol{y}(\boldsymbol{x})$ with a tunable accuracy depending on the number of PCA components $\tilde{n}$, such that [54]:

$$\left(\frac{\sigma_{\tilde{n}+1}}{\sigma_1}\right)^2 \leq \varepsilon^2 \tag{4.42}$$

where ε is a given error threshold tuned by the user.

4.3 Application Examples

This section compares the accuracy and the robustness against noise of the three implementations of the complex- and vector-valued LS-SVM regression provided in this Chapter by considering two different application examples. Specifically, the proposed approaches are applied to predict the scattering parameters of a serpentine structure with three parameters and the transfer function of a high-speed link with four parameters.

4.3.1 Example I

As a first test case, the complex-valued LS-SVM regression is applied to calculate the scattering parameters of a serpentine delay line structure is presented. Serpentine lines are widely used in printed circuit board (PCB) design to compensate time delays introduced by the trace routing. However, the frequency-domain behavior of such a structure is heavily affected by its geometrical and electrical parameters and should be carefully assessed during the design phase to avoid signal and power integrity issues and to meet design constraints [57].

The structure of the serpentine delay line considered in this example is shown in Figure 4.1 (inspired by [57]). The S_{21} scattering parameters of the above structure are investigated as a function of three parameters (i.e., $\boldsymbol{x} = [\varepsilon_r, LL, SW]^T$) in a frequency bandwidth from 1 MHz to 1 GHz. Table 4.1 shows the range of variability serpentine delay line parameters that are simulated to produce the training and test data. The whole dataset consists of 3000 samples (1000 training data and 2000 test samples). The samples were generated via LHS by assuming a uniform variability between their maximum and minimum value. For each configuration of the geometrical parameters, the corresponding scattering parameters were computed for 5000 linearly spaced frequency sample points. The samples were generated via a set of parametric simulations with the full-wave solver available in CST.

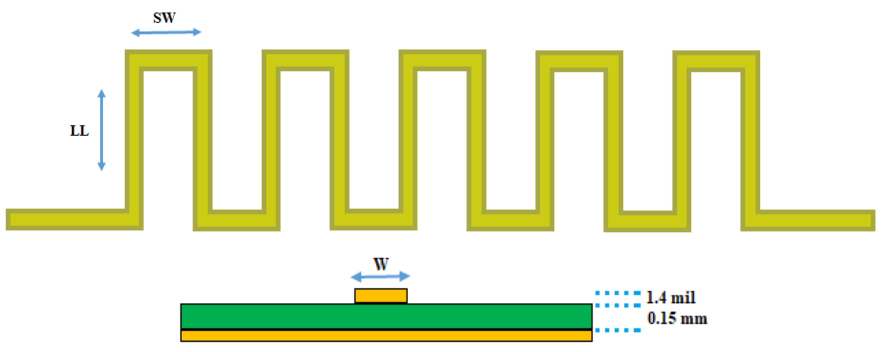

Figure 4.1: Panel (a): design parameters of the serpentine line to be analyzed; Panel (b): cross-sectional view of the serpentine line.

The PCA compression is then used to compress the number of frequency components in the training set with the aim of simplifying the model training. Figure 4.2 shows the behavior of the normalized singular values of the frequency points of

Table 4.1: Serpentine delay line parameters for the training and test dataset.

Training Ranges	**Test Ranges**
4.5 mm $\leqslant$ LL $\leqslant$ 5.1 mm	4.5 mm $\leqslant$ LL $\leqslant$ 5.1 mm
$3.9 \leqslant \epsilon_r \leqslant 4.5$	$3.9 \leqslant \epsilon_r v \leqslant 4.5$
0.13 mm $\leqslant$ SW $\leqslant$ 0.17 mm	0.13 mm $\leqslant$ SW $\leqslant$ 0.17 mm
1 MHz $\leqslant$ f $\leqslant$ 3 GHz	1 MHz $\leqslant$ f $\leqslant$ 3 GHz
1000 samples for each frequency	2000 samples for each frequency

the training dataset. The plot shows that $\bar{n} = 9$ is enough to represent the whole training set with a 0.001% threshold.

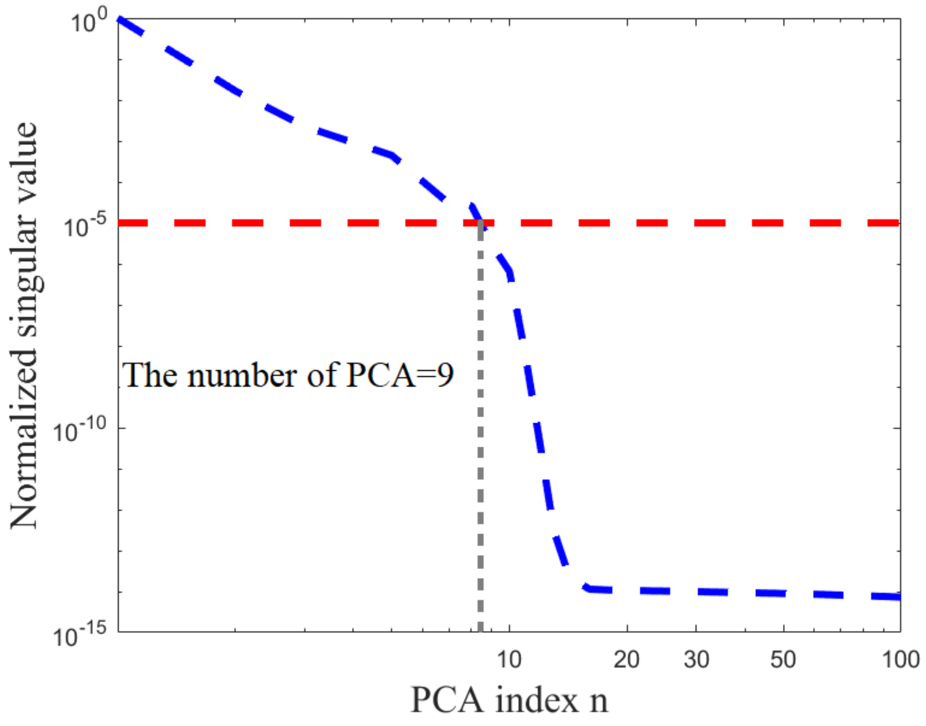

Figure 4.2: Normalized singular value plot of the serpentine delay line for the considered dataset with 5000 frequency points (blue line). The horizontal line shows the 0.001% threshold for the PCA truncation

After applying the PCA, the performances of the LS-SVM regression using the PCF and the CVCF complex kernel function (see Sec. 4.1.1), and the DCK LS-SVM regression (see Sec. 4.1.2), were assessed on the test samples for the S_{21} parameter. To investigate the performance of the mentioned methods, the normalized root mean square error (NRMSE), which writes:

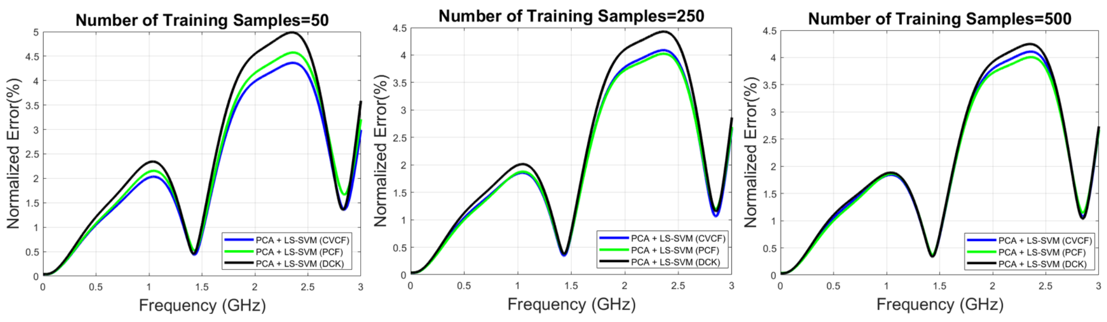

Figure 4.3: Comparison of the relative NRMSE values computed by the proposed approaches on the test samples by considering the real part of the S21 parameter of the serpentine delay line structure for an increasing number of training samples (i.e., $L = 50, 250, 500$)

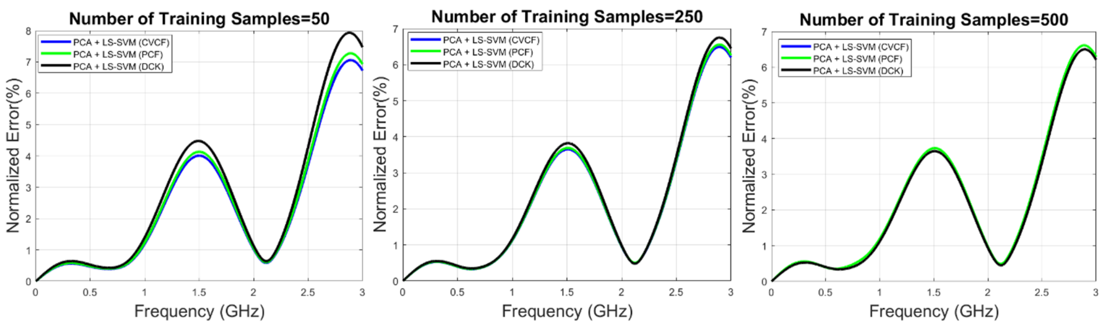

Figure 4.4: Comparison of the relative NRMSE values computed by the proposed approaches on the test samples by considering the imaginary part of the S21 parameter of the serpentine delay line structure for an increasing number of training samples (i.e., $L = 50, 250, 500$).

$$NRMSE\% = 100 \cdot \frac{\frac{1}{T}\sqrt{\sum_{t=1}^{T} \left(x_y - X_{\hat{y}}\right)^2}}{\frac{1}{T}\sqrt{\sum_{t=1}^{T} X_y^2}} \tag{4.43}$$

where X_y can be either the real or imaginary part of the actual test samples, $X_{\hat{y}}$ is the corresponding prediction estimated via the proposed metamodels, and T is the number of test samples.

Figures 4.3 and 4.4 show the normalized error for real and imaginary parts of these three regression approaches for an increasing number of training samples (i.e., L=50, 250, and 500). The plots highlight the improved accuracy of the CVCF and PCF with respect to the DCK. Indeed, with the DCK-based model, we are implicitly neglecting any kind of correlation between the real and imaginary parts of the S-parameters, and this lack of complexity becomes even more evident when a low number of training samples is used to train the model.

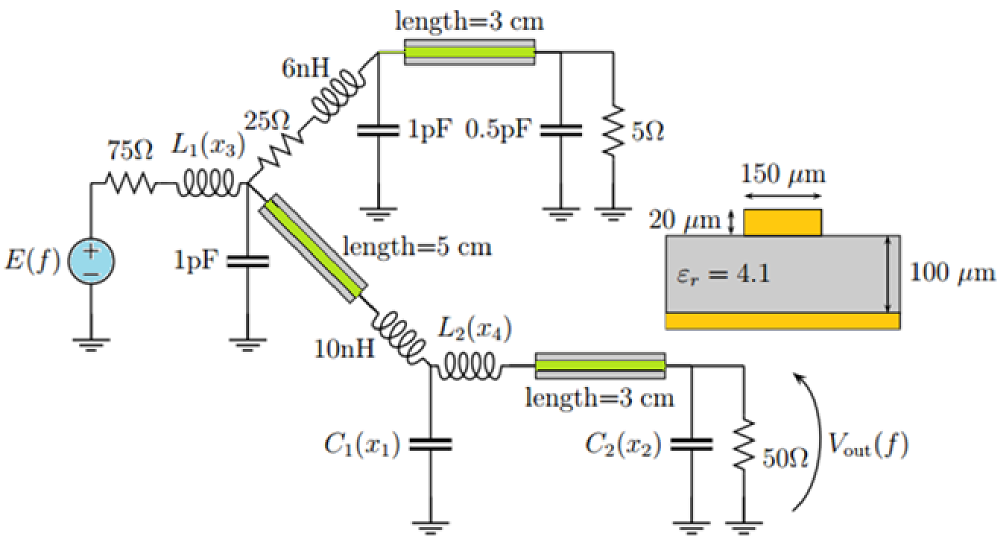

Figure 4.5: Schematic of the high-speed interconnect link.

Table 4.2: High-speed interconnect link parameters for the training and test datasets.

Training and Test ranges	
$C_1(x_1)$	$(1 \pm 0.5\,x_1)$pF
$C_2(x_2)$	$(0.5 \pm 0.25\,x_2)$pF
$L_1(x_3)$	$(10 \pm 5\,x_3)$nH
$L_2(x_4)$	$(10 \pm 5\,x_4)$nH

4.3.2 Example II

The second application example is based on the high-speed link depicted in Figure 4.5. The link represents a signal distribution on a PCB. Similar to the previous example, the frequency response of the link, and thus its performance, can be greatly influenced by possible variations of its internal parameters [58].

The proposed modeling approaches are here adopted to build a surrogate model for the frequency-domain behavior of the complex-valued transfer function, in which $y(\boldsymbol{x}; f) = \frac{\text{Vout}(f;\boldsymbol{x})}{E(f)}$, as a function of the values of four lumped components $C_1\,(x_1)$, $C_2\,(x_2)$, $L_1\,(x_3)$, $L_2\,(x_4)$ defined by four uniformly distributed normalized random

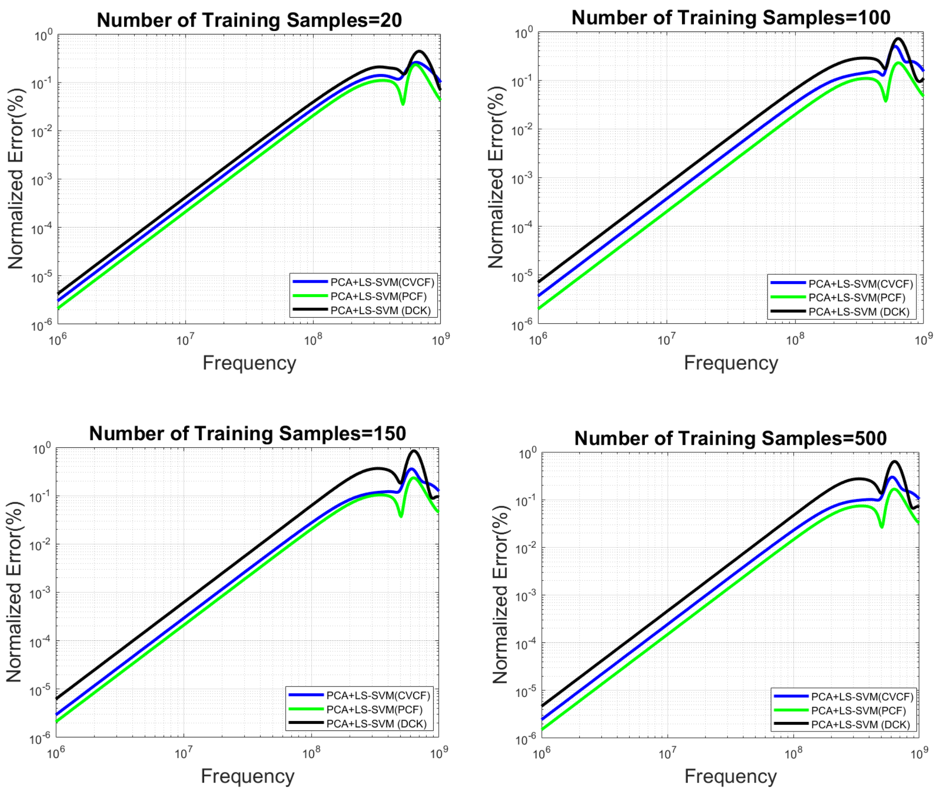

Figure 4.6: Comparison of the relative NRMSE values computed by the proposed approaches and a feed-forward multi-output neural network on the test samples by considering the real part of the transfer function of a high-speed link for an increasing number of training samples (i.e., $L = 20$, 100, 150 and 500)

variables $\boldsymbol{x} = [x_1, x_2, x_3, x_4]^T$ with a variation of $\pm 50\%$ around their central value (additional details are provided in Table 4.2).

Four input sets with $L = 20$, 100, 150 and 500 training input configurations were generated via an LHS and used as input for a computational model implemented in MATLAB, which provides as output the corresponding transfer function evaluated at 200 frequency points. The PCA is applied to remove redundant information, leading to a compressed representation of the original dataset with only $\bar{n} = 10$ components using a threshold of 0.01%. The compressed training sets are then used to train three different surrogate models based on the DCK, PCF, and CVCF LS-SVM regressions.

Figures 4.6 and 4.7 show the performance of each method on a test set consisting of 1000 samples assessed via the relative NRMSE in (4.43) for the real and imaginary parts, respectively. The results clearly highlight the improved accuracy achieved via the PCF. Moreover, as expected, due to its simplified formulation, the DCK again provides the lowest accuracy.

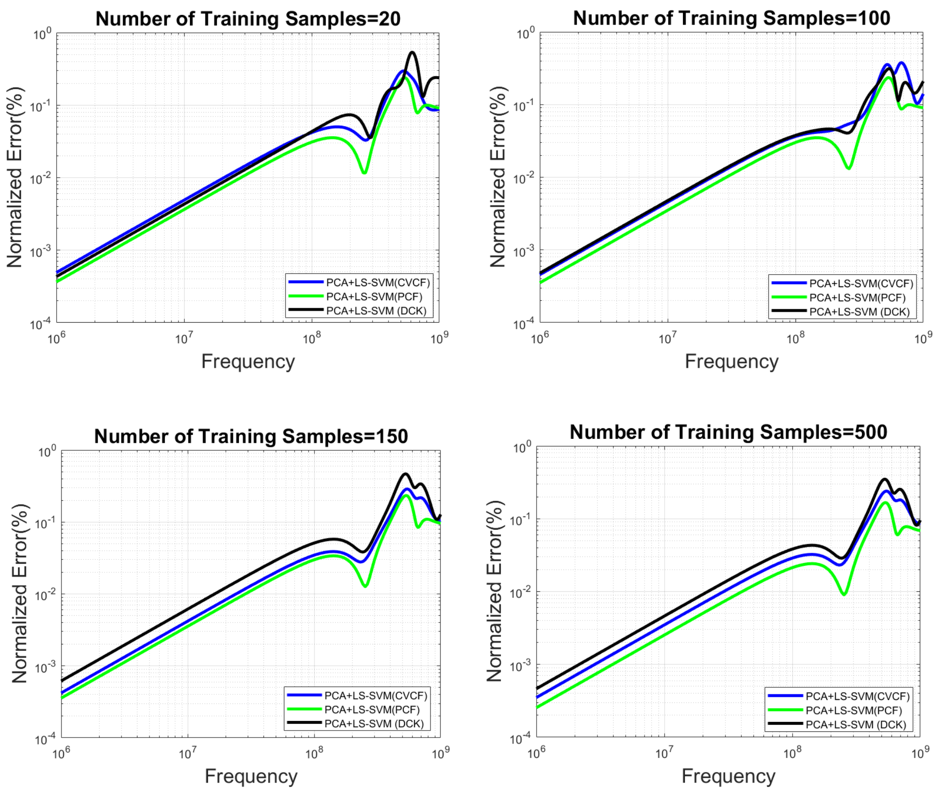

Figure 4.7: Comparison of the relative NRMSE values computed via the proposed approaches and a feed-forward multi-output neural network on the test samples by considering the imaginary part of the transfer function of high-speed link for an increasing number of training samples (i.e., $L = 20$, 100, 150 and 500).

Moreover, in order to stress the reliability of the proposed techniques, the training output $\{\boldsymbol{y}_l\}_{l=1}^{L}$ was corrupted with Gaussian noise, such that:

$$\boldsymbol{y}_{l,\text{ noisy}}\ (\boldsymbol{x}_l) = \boldsymbol{y}_l\,(\boldsymbol{x}_l) \times (1 + \zeta_n)\,, \tag{4.44}$$

where $\zeta_n \sim \mathcal{N}\,(0, \sigma_n^2)$ is a Gaussian random variable with standard deviation $\sigma_n =$ [0.01,0.03].

Figures 4.8 and 4.9 compare the relative NRMSE computed at a single frequency point selected as the one providing the maximum error via the proposed approaches for different values of the noise standard deviation σ_n and the number of training samples L for the real and imaginary parts, respectively. Among the introduced methods, CVCF shows the better performance and robustness against noise, both for real and imaginary parts.

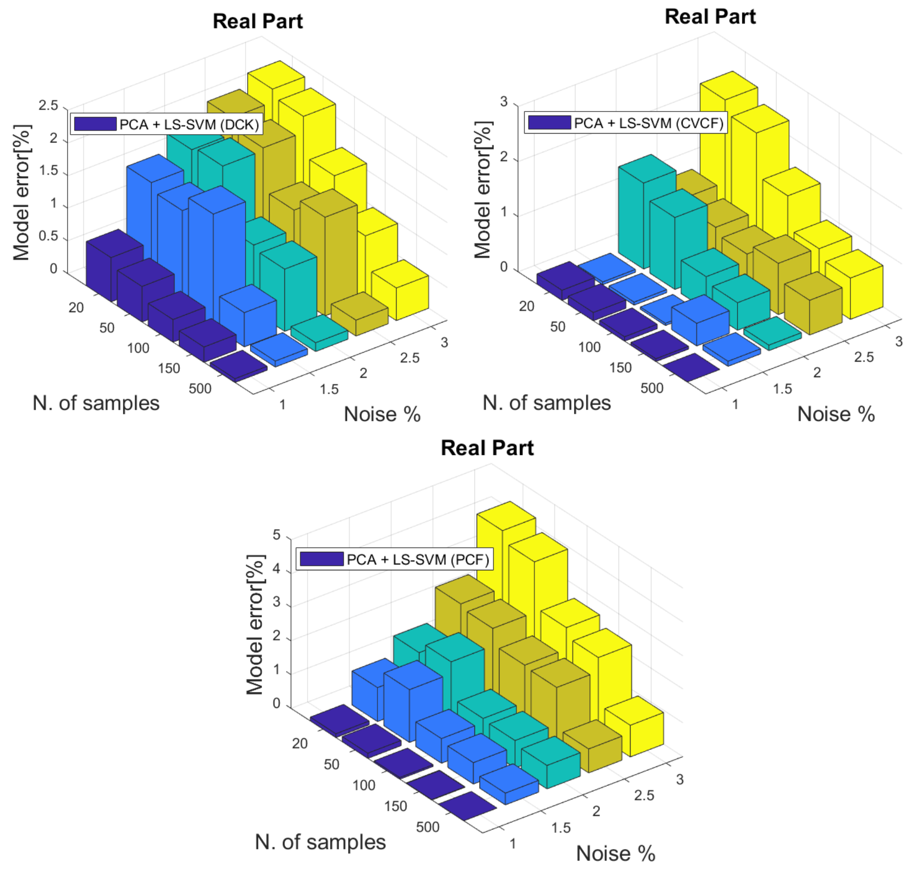

Figure 4.8: Three-dimensional plot of the relative NRMSE computed on the real part of the test samples at a single frequency point selected as the one providing the maximum error via the proposed approaches (see top left panel for the DCK, top right for CVCF and central bottom panel for PCF) for different values of the noise standard deviation σ_n and the number of training samples L.

4.4 Summary

This Chapter addressed two challenges related to the constructions of surrogate model in electronic applications with a complex- and vector-valued output. First of all, it provided a generalized complex formulation of the LS-SVM regression by investigating the performance of several real and complex-valued kernel functions. Then, it presented an efficient and effective solution based on data compression aimed at extending the applicability of standard single-output kernel regression to vector-valued regression problems. The performance of the above techniques have been investigated on two examples: a serpentine EM structure with three parameters and a high-speed link with four parameters.

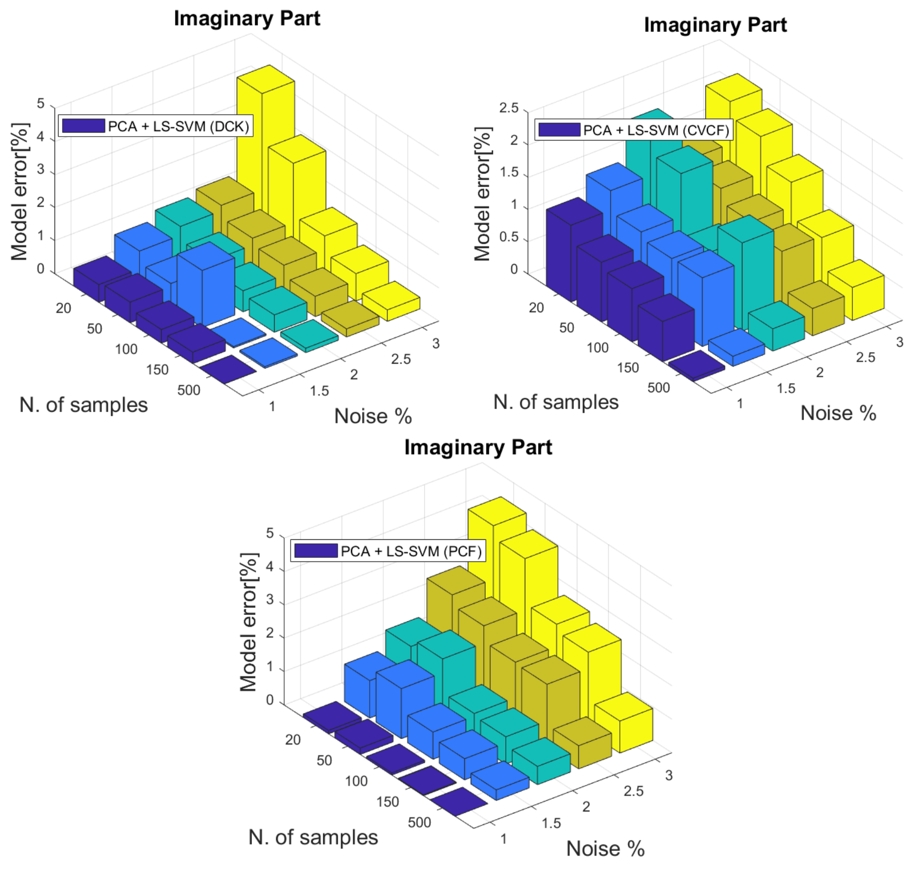

Figure 4.9: Three-dimensional plots of the relative NRMSE computed on the imaginary part of the test samples at a single frequency point selected as the one providing the maximum error via the proposed approaches (see top left panel for the DCK, top right for CVCF and central bottom panel for PCF) for different values of the noise standard deviation σ_n and the number of training samples L.

Chapter 5

Bridging the Gap between ANNs and Kernel-Machine Regressions in Vector-Value EM Applications

This chapter presents an alternative strategy for the employment of the kernel machine regression in multi- or vector-output regression problems with the aim of building closed-form and efficient vector-valued surrogate models for electronic devices and circuits. In contrast to the data compression strategy presented before (see Sec. 4.2), this chapter proposes a generalized vector-valued formulation of the kernel ridge regression (KRR). Such approach can be directly applied to tackle multi-output regression problems without requiring any data manipulation or compression, thus mitigating possible generalization issues due to the selection of the model components and improving the robustness to noise [59]. Moreover, the proposed mathematical framework allows at bridging the gap in terms of flexibility between kernel-based regressions and multi-output ANN structures. The effectiveness and the performance of the proposed vector-valued K-formulation of the KRR will be investigated on several examples and compared with the performance achieved by state-of-the-art techniques such as the PCA+LS-SVM presented in Chapter. 4.

5.1 Scalar-Output Kernel Ridge Regression

Kernel Ridge Regression (KRR) is a kernel-machine regression having several similarities with the LS-SVM regression presented in Sec. 2.4.2.

Similar to the LS-SVM regression, let us consider the problem of building a function $\tilde{f}$ by approximating a set of training pairs $\mathcal{S} = \{(\boldsymbol{x}_l, y_l)\}_{l=1}^{L}$, where $\boldsymbol{x}_l \in \mathcal{X} \subseteq \mathbb{R}^p$ represents the training input and $y_l \in \mathcal{Y} \subseteq \mathbb{R}$ are the corresponding scalar outputs. In particular, knowing the training set, we seek the "best" structure of a generic function $\hat{f}(\boldsymbol{x})$ able to approximate $f(\boldsymbol{x})$ for any $\boldsymbol{x} \in \mathcal{X}$ by minimizing the following penalized ERM [13]:

$$\hat{f} = \underset{\tilde{f} \in \mathcal{H}}{\arg\min} \sum_{l=1}^{L} (y_l - \tilde{f}(\boldsymbol{x}_l))^2 + \lambda \|\tilde{f}\|_{\mathcal{H}}^2, \tag{5.1}$$

where, similar to the LS-SVM regression, we are using a squared loss function together with a Tickonov regularizer and λ is the hyperparameter associated to it.

According to the representation theorem [13], any optimal solution $\hat{f}(\boldsymbol{x})$ of (5.1) can be written as:

$$\hat{f}(\boldsymbol{x}) = \sum_{l=1}^{L} \alpha_l k(\boldsymbol{x}_l, \boldsymbol{x}), \tag{5.2}$$

where $k(\cdot, \cdot) : \mathbb{R}^{p \times p} \rightarrow \mathbb{R}$ is the kernel function (additional details are provided in Sec. 2.4).

Plugging (5.2) into (5.1), we can write:

$$\min_{\boldsymbol{\alpha}} \sum_{l=1}^{L} (y_l - \sum_{m=1}^{L} \alpha_m k(\boldsymbol{x}_m, \boldsymbol{x}_l))^2 + \lambda \| \sum_{l=1}^{L} \alpha_l k(\boldsymbol{x}_l, \boldsymbol{x}) \|_{\mathcal{H}}^2, \tag{5.3}$$

where according to the kernel properties [13]:

$$\begin{aligned} \|\hat{f}\|_{\mathcal{H}}^2 &= \| \sum_{l=1}^{L} \alpha_l k(\boldsymbol{x}_l, \boldsymbol{x}) \|_{\mathcal{H}}^2 \\ &= \sum_{l,m=1}^{L} \alpha_l \alpha_m k(\boldsymbol{x}_l, \boldsymbol{x}_m) = \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}. \end{aligned} \tag{5.4}$$

In the above equation $\mathbf{K} \in \mathbb{R}^{L \times L}$ is the empirical kernel matrix, also known as kernel Gram matrix, defined by evaluating the kernel function on each configuration pairs belonging to the training input set, such that:

$$[\mathbf{K}]_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{5.5}$$

for any $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the training input set.

The optimization problem in (5.1) can be written in its matrix form as:

$$\min_{\boldsymbol{\alpha}} (\boldsymbol{y} - \mathbf{K}\boldsymbol{\alpha})^T (\boldsymbol{y} - \mathbf{K}\boldsymbol{\alpha}) + \lambda \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}, \tag{5.6}$$

where $\boldsymbol{y} = [y_1, \ldots, y_L]^T$ is a vector collecting the training outputs, whereas $\mathbf{K}\boldsymbol{\alpha}$ represents the corresponding predictions computed via (5.2).

Also for the KRR, the above optimization admits a closed-form solution, which writes:

$$-\mathbf{K}\boldsymbol{y} + \mathbf{K}^2\boldsymbol{\alpha} + \lambda\mathbf{K}\boldsymbol{\alpha} = 0, \tag{5.7}$$

which can be recast in terms of the following linear system:

$$(\mathbf{K} + \lambda\mathbf{I}_L)\boldsymbol{\alpha} = \boldsymbol{y}, \tag{5.8}$$

where $\mathbf{I}_L$ refers to the $L \times L$ identity matrix.

Therefore, the model coefficients in the vector $\boldsymbol{\alpha}$ can be suitably computed by solving the above linear system of equations, i.e.,

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I}_L)^{-1}\boldsymbol{y}. \tag{5.9}$$

It is important to point out that the KRR turns out to be equivalent to the dual space formulation of the LS-SVM regression presented in Sec. 2.4.2 with a null bias term (i.e., $b = 0$).

5.2 ANNs vs. Scalar Kernel-Machine Regressions

As briefly presented in Sec. 2.3, ANN structures can approximate any set of non-linear functions via a collection of artificial neurons connected together and organized in layers [18]. The overall structure turns out to be extremely flexible, without any limitation in terms of number of layers, neurons per layer, number of outputs, etc. Moreover, the mathematical model describing the input-output map obtained by the ANN is usually not linear with respect to the model unknowns (i.e., the weights and bias), since they appear within the argument of non-linear functions (i.e., the activation functions). This allows learning very complex non-linear behaviors, but on the other hand, the non-linear structure of the ANN model leads to a non-convex optimization with several local minima. Such non-convex optimization makes the training phase for the ANN rather complicated and data-hungry [23, 24].

As shown in Fig. 5.1, a generic kernel model is equivalent to an ANN structure with a single hidden layer and in which the unknown weights (i.e., $\{\alpha_1 \ldots, \alpha_L\}$) appear linearly as the connection between the hidden and the output layer [25, 60]. It is important to remark that in such structure the number of both weights and

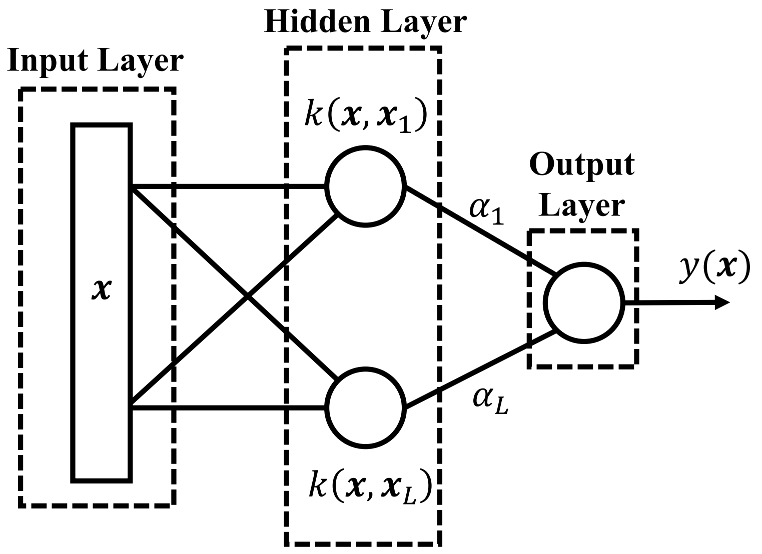

Figure 5.1: ANN interpretation of a scalar-output kernel regression (the picture is inspired by [60]).

neurons in the hidden layer are fixed, and turns out to be equal to the number of training samples [60] (or less for the support vector machine regression [25, 26]). This means that the overall model complexity in terms of regression unknowns, turns out to be independent of the number of input parameters [25]. Unlike ANNs, the linear model structure adopted by kernel regressions (i.e., the model unknowns appear linearly) has the key advantage of heavily simplifying the training phase, which reduces to the solution of a standard convex optimization [61], thus leading to several advantages in terms of training time and accuracy with respect to the number of training samples [23, 24, 61, 62].

Conversely, in the advocated scalar kernel regressions, the lack of flexibility needed to guarantee the linear structure leads to some limitations with respect to ANN. In fact, their implementations are usually limited to scalar-output regression problems [63].

5.3 From Scalar- to Vector-Valued KRR

This Section aims at providing a generalized formulation of the scalar-output KRR presented in Sec. 5.1 for vector-valued output or multi-task regression problems. For the sake of simplicity, we will focus on the specific case of vector-valued regressions for which the training set is defined as $\mathcal{S} = \{(\boldsymbol{x}_l, \boldsymbol{y}_l)\}_{l=1}^{L}$, in which $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^p$ is a vector collecting the configurations of the input parameters (e.g., geometrical and electrical parameters of an EM structure) and $\boldsymbol{y}_i = [y_i^{(1)}, \ldots, y_i^{(D)}]^T \in \mathbb{R}^D$ is a vector collecting the corresponding vector-valued training outputs (e.g., the frequency samples of a frequency response). The above training set can be rewritten in its compact form as $\mathcal{S} = \{(\mathbf{X}, \mathbf{Y})\}$ where $\mathbf{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_L]^T$ is a $L \times p$ matrix collecting the configurations of the training input and $\mathbf{Y} = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_L]^T$ is a $L \times D$ matrix associated to the training outputs.

Given the information available in the training set $\mathcal{S}$, our goal is to learn D scalar functions $\hat{f}^{(d)} : \mathcal{X} \to \mathbb{R}$ with $d = 1, \dots, D$, able to provide an accurate prediction of the actual output vector $\boldsymbol{y}(\boldsymbol{x})$ for any configuration of the parameters $\boldsymbol{x} \in \mathcal{X}$. [1] In order to deal with the above vector-valued regression problem, the learning problem in Eq. (5.1) must be generalized as follows:

$$\hat{\boldsymbol{f}} = \arg\min_{\tilde{\boldsymbol{f}} \in \mathcal{H}} \sum_{d=1}^{D} \sum_{l=1}^{L} (y_l^{(d)} - \tilde{f}^{(d)}(\boldsymbol{x}_l))^2 + \lambda \|\tilde{\boldsymbol{f}}\|_{\mathcal{H}}^2, \tag{5.10}$$

where $y_l^{(d)}$ and $\tilde{f}^{(d)}(\boldsymbol{x}_l)$ represent the d-th component of the $l-$th training output and the corresponding model prediction, respectively.

According to the represented theorem for vector-valued regression problem presented in [64], any solution $\hat{\boldsymbol{f}}$ of Eq. (5.10) takes the form:

$$\hat{\boldsymbol{f}}(\boldsymbol{x}) = \sum_{l=1}^{L} \mathbf{K}(\boldsymbol{x}, \boldsymbol{x}_l)\boldsymbol{c}_l, \tag{5.11}$$

where $\mathbf{K}(\cdot,\cdot) : \mathbb{R}^{p\times p} \to \mathbb{R}^{D\times D}$ is a matrix multi-output kernel acting on the column vectors $\boldsymbol{c}_l \in \mathbb{R}^D$ collecting the regression unknowns for $l = 1, \dots, L$. For a generic output n, the above equation writes:

$$\begin{aligned}\hat{f}^{(n)}(\boldsymbol{x}) &= \sum_{l=1}^{L} [\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}_l)]_{[n,:]}\boldsymbol{c}_l \\ &= \sum_{d=1}^{D} \sum_{l=1}^{L} [\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}_l)]_{[n,d]} c_{d,l},\end{aligned} \tag{5.12}$$

where $[\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}_l)]_{[n,:]}$ and $[\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}_l)]_{[n,d]}$ denotes the n-th row and the (n, d)-element of the matrix kernel $\mathbf{K}(\cdot,\cdot)$, respectively, and $c_{d,l}$ is the d-th element of the vector $\boldsymbol{c}_l$.

Equation (5.12) can be rewritten in its scalar form, i.e.,

[1]The proposed formulation can be extended to the more general case of multi-task formulation in which the number of training samples L_d can vary for each output d, as well as the number of parameters p_d.

$$f^{(n)}(\boldsymbol{x}) = \sum_{d=1}^{D}\sum_{l=1}^{L} k((\boldsymbol{x}, n), (\boldsymbol{x}_l, d))c_{d,l}, \tag{5.13}$$

where $k((\boldsymbol{x}, n), (\boldsymbol{x}_l, d)) : \mathbb{R}^{p\times p} \times \mathbb{R}^{\{1,\dots,D\}\times\{1,\dots,D\}} \to \mathbb{R}$ represents the (n, d) entry of the multi-output kernel matrix $\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}_l)$, such that $k((\boldsymbol{x}, n), (\boldsymbol{x}_l, d)) = [\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}_l)]_{[n,d]}$.

5.3.1 Separable Multi-Output Kernels for Vector-Valued Regression

The kernel structure in Eq. (5.12) and Eq. (5.13) was introduced by [65]. The multi-output kernel should be able to account for the correlation in both the parameter space and output components. Unfortunately, there do not exist off-the-shelf kernel functions which can be directly applied in such context. The simplest solution is to work on a specific class of multi-output kernels such as the separable kernel or sum of separable kernels [65, 64]. Specifically, we will consider matrix kernel functions $\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}')$, obtained as the product between two scalar kernels acting either on the input space or on the output dimensions, such that:

$$\begin{aligned}[\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}')]_{[d,d']} &= k((\boldsymbol{x}, d), (\boldsymbol{x}_l, d')) \\ &= k_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{x}')k_o(d, d'),\end{aligned} \tag{5.14}$$

where $k_{\boldsymbol{x}}$ and k_o are scalar kernels acting independently on the input space (i.e., $k_{\boldsymbol{x}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$) and output dimensions (i.e., $k_o : \{1, \dots, D\} \times \{1, \dots, D\} \to \mathbb{R}$).

Therefore, for each pairs $\boldsymbol{x}$ and $\boldsymbol{x}'$ belonging to the input space $\mathcal{X}$, the resulting multi-output kernel matrix $\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}')$ can be written as:

$$\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}') = k_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{x}')\mathbf{B}, \tag{5.15}$$

where $\mathbf{B} \in \mathbb{R}^{D\times D}$ is a symmetric semi-definite matrix completely independent from the input parameters $\boldsymbol{x}$ and $\boldsymbol{x}'$, in which its elements are obtained by evaluating the scalar kernel k_o on the output dimensions (i.e., $\{1, \dots, D\} \times \{1, \dots, D\}$). The overall kernel matrix $\mathbf{K}(\boldsymbol{x}, \boldsymbol{x}')$ is a $D \times D$ symmetric matrix by construction, since it is the product of a symmetric function $k_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{x}')$ with a symmetric matrix $\mathbf{B}$.

By combining the optimal solution in Eq. (5.11) for the vector-output scenario, with the separable kernel structure in Eq. (5.14), we can write:

$$\hat{\boldsymbol{f}}(\boldsymbol{x}) = \sum_{l=1}^{L} \mathbf{K}(\boldsymbol{x}, \boldsymbol{x}_l)\boldsymbol{c}_l = \sum_{l=1}^{L} k_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{x}_l)\mathbf{B}\boldsymbol{c}_l. \tag{5.16}$$

5.3.2 Matrix Formulation for Vector-Valued KRR with Separable Kernel

Let us now consider the following matrix formulation of the ERM in Eq. (5.10) developed for the vector-valued scenario:

$$\min_{\boldsymbol{f}\in\mathcal{H}} \|\mathbf{Y}-\mathbf{F}\|_F^2 + \lambda\|\boldsymbol{f}\|_{\mathcal{H}}^2, \tag{5.17}$$

where $\mathbf{F} = [\boldsymbol{f}_1^T, \dots, \boldsymbol{f}_L^T]$ is a $L \times D$ matrix collecting the model predictions for the samples in the training set, such that $[\mathbf{F}]_{ij} = f^{(j)}(\boldsymbol{x}_i)$, and $\|\cdot\|_F$ is the Frobenius norm defined as:

$$\|A\|_F^2 = \sum_{i=1}\sum_{j=1} a_{ij}^2 = \mathrm{Tr}(\mathbf{A}\mathbf{A}^T). \tag{5.18}$$

According to Eq. (5.12) and Eq. (5.15), the n-row of the matrix $\mathbf{F}$ in Eq. (5.17) writes:

$$\begin{aligned}[\mathbf{F}]_{[n,:]} = \hat{\boldsymbol{f}}(\boldsymbol{x}_n)^T &= \sum_{l=1}^{L} k_x(\boldsymbol{x}_n, \boldsymbol{x}_l)\boldsymbol{c}_l^T\mathbf{B}^T \\ &= \sum_{l=1}^{L} k_x(\boldsymbol{x}_n, \boldsymbol{x}_l)\boldsymbol{c}_l^T\mathbf{B}.\end{aligned} \tag{5.19}$$

Since $\mathbf{B}$ is a symmetric matrix (i.e., $\mathbf{B} = \mathbf{B}^T$), the matrix $\mathbf{F}$ can be rewritten as [66]:

$$\mathbf{F} = \mathbf{K}_{\boldsymbol{x}}\mathbf{C}\mathbf{B}, \tag{5.20}$$

where $\mathbf{K}_x \in \mathbb{R}^{L\times L}$ with $[\mathbf{K}_{\boldsymbol{x}}]_{[ij]} = k_{\boldsymbol{x}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the Gram matrix associated to the kernel $k_{\boldsymbol{x}}$ evaluated on the input training samples and $\mathbf{C} \in \mathbb{R}^{L\times D}$ is a matrix collecting the regression unknowns $\boldsymbol{c}_l$, such that $\mathbf{C} = [\boldsymbol{c}_1, \dots, \boldsymbol{c}_L]^T$

By substituting the above model structure in Eq. (5.17), we get the following optimization problem:

$$\min_{\mathbf{C}} \|\mathbf{Y} - \mathbf{K}_{\boldsymbol{x}}\mathbf{C}\mathbf{B}\|_F^2 + \lambda\langle\mathbf{C}^T\mathbf{K}_{\boldsymbol{x}}\mathbf{C}, \mathbf{B}\rangle_F, \tag{5.21}$$

in which $\langle\cdot,\cdot\rangle_F$ is the inner Frobenius product, which for the case of matrices $\mathbf{A}$ and $\mathbf{B}$ writes:

$$\langle \mathbf{A}, \mathbf{B} \rangle_F = \sum_{i,j} A_{ij} B_{ij} = \mathrm{Tr}(\mathbf{A}^T \mathbf{B}). \tag{5.22}$$

According to [67] and the references within, the optimal values for the entries of the coefficient matrix **C** can be estimated as the ones which satisfied the following discrete-time Sylvester equation:

$$\mathbf{K}_{\boldsymbol{x}} \mathbf{C} \mathbf{B} + \lambda \mathbf{C} = \mathbf{Y}. \tag{5.23}$$

After computing the unknown matrix **C** by solving the above equation, Eq. (5.16) use to make predictions for a generic input configuration $\boldsymbol{x} \in \mathcal{X}$:

$$\hat{\boldsymbol{f}}(\boldsymbol{x}) = \sum_{l=1}^{L} k_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{x}_l) \mathbf{B} \boldsymbol{c}_l = \sum_{l=1}^{L} \mathbf{K}(\boldsymbol{x}, \boldsymbol{x}_l) \boldsymbol{c}_l. \tag{5.24}$$

5.4 Training Algorithms

5.4.1 Kronecker formulation & Gradient Descent

The simplest way to solve the discrete-time Sylvester equation in (5.23) comes from its Kronecker formulation [68], which writes [66]:

$$\underbrace{(\mathbf{B} \otimes \mathbf{K}_{\boldsymbol{x}} + \lambda \mathbf{I}_{LD})}_{\mathbf{A}} \mathrm{vec}(\mathbf{C}) = \mathrm{vec}(\mathbf{Y}), \tag{5.25}$$

where $\otimes$ is the Kronecker product, $\mathbf{I}_{LD}$ refers to the $LD \times LD$ identity matrix and the vec$(\cdot)$ operator stacks column of its argument matrix into a column vector and therefore vec$(\mathbf{C}) \in \mathbb{R}^{LD}$ is a vector collecting the regression coefficients $\boldsymbol{c}_l$ in Eq. (5.11), with $\mathbf{C} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_L]^T \in \mathbb{R}^{L \times D}$. Like the scalar case, the above equation can be rewritten in terms of the Gram vector-valued matrix **K**, such that:

$$(\mathbf{K} + \lambda \mathbf{I}_{LD}) \mathrm{vec}(\mathbf{C}) = \mathrm{vec}(\mathbf{Y}), \tag{5.26}$$

where the Gram vector-valued matrix $\mathbf{K} \in \mathbb{R}^{(LD) \times (LD)}$ associated to the whole input training dataset **X** and output components writes:

$$\mathbf{K} = \mathbf{K}(\mathbf{X}, \mathbf{X}) = \mathbf{B} \otimes k_{\boldsymbol{x}}(\boldsymbol{X}, \boldsymbol{X}) = \mathbf{B} \otimes \mathbf{K}_{\boldsymbol{x}}. \tag{5.27}$$

It is straightforward to see that the unknown coefficients collected in the vector vec($\mathbf{C}$) can be computed by solving a linear system of equations, which writes:

$$\text{vec}(\mathbf{C}) = (\mathbf{B} \otimes \mathbf{K}_{\boldsymbol{x}} + \lambda \mathbf{I}_{LD})^{-1} \, \text{vec}(\mathbf{Y}). \tag{5.28}$$

Therefore, the model training can be recast as the solution of a linear system of equations with LD equations in LD unknowns. A direct solution of such linear systems requires the inversion of a possible coupled matrix $\mathbf{A}$ in Eq. (5.25) of dimension $LD \times LD$, for which the computational complexity scales as $\mathcal{O}(L^3D^3)$. This makes the direct inversion of the matrix $\mathbf{A}$ extremely inefficient or intractable in a standard laptop when the product between the number of training samples L and the output dimensionality D, becomes in the order of thousand.

To mitigate the above limitation, the linear system in Eq. (5.25) can be solved in a more efficient way via an iterative procedure based on the gradient descent (GD) algorithm [61, 69]:

$$\text{vec}(\mathbf{C})_k = \text{vec}(\mathbf{C})_{k-1} - \alpha[\mathbf{A} \, \text{vec}(\mathbf{C})_{k-1} - \text{vec}(\mathbf{Y})], \tag{5.29}$$

where vec($\mathbf{C}$)$_k$ represents the unknown regression coefficients estimated at the k-th step and α is a scalar number, known as the learning rate, defining the step-size at each iteration.

Specifically, the proposed modeling framework implements the conjugate gradient method [69], which provides an efficient version of the above algorithm tailored for semi-definite matrices, such as the matrix $\mathbf{A}$ [59]. Such implementation allows reducing the computational complexity of the training phase from $\mathcal{O}(L^3D^3)$ to $\mathcal{O}(KL^2D^2)$, where K is the number of iterations required by the GD algorithm to converge. It is important to remark that thanks to the benefits in terms of computational cost of the GD algorithms with respect to the plain inversion algorithms, the above inversion scheme implemented in standard laptop allows to deal with regression problems in which $LD \leq 10$k.

5.4.2 Diagonalization Procedure

Diagonaliazation can be seen as an more efficient way with respect to the Kronecker formulation presented in the previous Section for the solution of the discrete-time Sylvester equation (5.23) based on a representation of the Gram kernel matrices $\mathbf{K}_{\boldsymbol{x}}$ and $\mathbf{B}$ in terms on their eigenvector and eigenvalue matrices [70], i.e.,:

$$\mathbf{K}_{\boldsymbol{x}} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T \text{ and } \mathbf{B} = \mathbf{T}\mathbf{M}\mathbf{T}^T, \tag{5.30}$$

where $\mathbf{U} \in \mathbb{R}^{L\times L}$ and $\mathbf{T} \in \mathbb{R}^{D\times D}$ are matrices collecting the eigenvectors of the matrices $\mathbf{K}_{\boldsymbol{x}}$ and $\mathbf{B}$, respectively, whereas $\boldsymbol{\Lambda} \in \mathbb{R}^{L\times L}$ and $\mathbf{M} \in \mathbb{R}^{D\times D}$ are diagonal matrices collecting the corresponding eigenvalues.

Using the definitions in Eq. (5.30), the discrete-time Sylvester equation in Eq. (5.23) can be rewritten as:

$$\mathbf{\Lambda}\tilde{\mathbf{C}}\mathbf{M} + \lambda\tilde{\mathbf{C}} = \tilde{\mathbf{Y}}, \tag{5.31}$$

where $\tilde{\mathbf{C}} = \mathbf{U}^T\mathbf{C}\mathbf{T}$ and $\tilde{\mathbf{Y}} = \mathbf{U}^T\mathbf{Y}\mathbf{T}$ are new transformed matrices collecting a transformed version of regression unknowns and source term, respectively.

Due to the diagonal structure of Eq. (5.31), a generic entry of the unknown matrix $[\tilde{\mathbf{C}}]_{ij} = c_{ij}$ can be suitably computed via a scalar equation defined by the diagonal eigenvector matrices $\mathbf{\Lambda}$ and $\mathbf{M}$, such as:

$$\tilde{c}_{ij} = \frac{\tilde{y}_{ij}}{[\mathbf{\Lambda}]_{ii}[\mathbf{M}]_{jj} + \lambda}. \tag{5.32}$$

Once the entries of the matrix $\tilde{\mathbf{C}}$ has been computed via the above equation, the original unknown matrix $\mathbf{C}$ can be reconstructed as:

$$\mathbf{C} = \mathbf{U}\tilde{\mathbf{C}}\mathbf{T}^T. \tag{5.33}$$

The above approach for solving the discrete-time Sylvester equation turns out to be more efficient than its equivalent solution based on the Kronecker formulation [68] presented in [59]. Indeed, since the diagonalization is applied on the matrices $\mathbf{K}_{\boldsymbol{x}}$ and $\mathbf{B}$ separately, the computational cost required for the model training reduces from $\mathcal{O}(L^3D^3)$ to $\mathcal{O}(L^3 + D^3 + L^2D + LD^2)$, thus leading to beneficial effect on the training time when the product $L \times D$ is large.

5.5 Separable Kernels for Vector-Valued KRR

This Section aims at discussing possible solutions for the design of separable kernel functions tailored for vector-valued KRR.

5.5.1 Block-diagonal Multi-Output Kernel Matrix

The discussion starts considering a special case of the separable kernel function in Eq. (5.14), in which the kernel acting on the output dimensions $k_o(d, d') = \delta_{d,d'}$, such that:

$$k_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{x}')k_o(d, d') = k_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{x}')\delta_{d,d'}, \tag{5.34}$$

where $\delta_{d,d'}$ is the Kronecker delta. This means that in Eq. (5.16) we are considering $\mathbf{B} = \mathbf{I}_D$ (i.e., the identity matrix).

In the above case, the overall regression problem turns out to be equivalent to train D scalar regression problems using the same kernel function $k_{\boldsymbol{x}}$. Therefore, the associated Gram kernel matrix $\mathbf{K}(\mathbf{X},\mathbf{X})$ becomes a $LD \times LD$ block diagonal matrix:

$$\mathbf{K}(\mathbf{X},\mathbf{X}) = diag(\mathbf{K}_{\boldsymbol{x}},\dots,\mathbf{K}_{\boldsymbol{x}}) = \begin{bmatrix} \mathbf{K}_{\boldsymbol{x}} & \mathbf{0} & \vdots \\ \mathbf{0} & \ddots & \mathbf{0} \\ \vdots & \mathbf{0} & \mathbf{K}_{\boldsymbol{x}} \end{bmatrix}. \tag{5.35}$$

Such decoupled interpretation of the vector-valued KRR has several advantages with respect to the standard modeling scheme in which a plain scalar kernel regression is applied to construct a set of independent surrogate models, one for each output dimensions. Indeed, even if the multi-output kernel in Eq. (5.34) still considers the output dimensions to be independent, it allows to learn them in one shoot via the solution of single optimization problem. This means that the number of hyperparameters to be tuned during the model training is independent from the number of output dimensions, since it is determined by the structure of scalar kernel $k_{\boldsymbol{x}}$, only. Also, possible correlations among the output dimensions are inherently accounted during the training phase by means of the hyperparameters tuning, since the latter operation is carried out on the whole training set and output dimensions.

It is important to notice that thanks to the block-diagonal structure of the Gram kernel matrix $\mathbf{K}(\mathbf{X},\mathbf{X})$ in Eq. (5.35), the regression training turns out to be extremely efficient. Indeed, the overall inversion of the $LD \times LD$ Gram matrix $\mathbf{K}$ reduces to invert a $L \times L$ matrix, i.e.,

$$\begin{aligned}[\mathbf{K} + \lambda\mathbf{I}_{LD}]^{-1} &= [diag((\mathbf{K}_x + \lambda\mathbf{I}_L),\dots,(\mathbf{K}_x + \lambda\mathbf{I}_L))]^{-1} \\ &= diag([\mathbf{K}_x + \lambda\mathbf{I}_L]^{-1},\dots,[\mathbf{K}_x + \lambda\mathbf{I}_L]^{-1}) \\ &= \mathbf{I}_D \otimes [\mathbf{K}_x + \lambda\mathbf{I}_L]^{-1}. \end{aligned} \tag{5.36}$$

Due to the block diagonal structure of the vector-valued kernel matrix $\mathbf{K}$ in Eq. (5.35), the overall computational complexity required by the matrix inversion is $\mathcal{O}(L^3)$ (i.e., the computational cost required to invert the sub-matrix $[\mathbf{K}_x + \lambda\mathbf{I}_L]$), since the hyperparameters of the kernel $k_{\boldsymbol{x}}$ and λ are shared by all the output dimensions.

5.5.2 Coupled Multi-Output Kernel Matrix

A possible alternative for the kernel k_o acting on the output dimensions is provided by the so-called *mixed kernel* [65], which writes:

$$k_o(d,d') = \omega + (1-\omega)\delta_{d,d'}, \tag{5.37}$$

or equivalently to a matrix $\mathbf{B}$ in Eq. (5.16):

$$\mathbf{B} = \omega \mathbf{1} + (1-\omega)\mathbf{I}_D, \tag{5.38}$$

where $\mathbf{1}$ is a $D \times D$ matrix whose entries are equal to 1 and $\omega \in [0,1]$ is the kernel hyperparameter.

The resulting Gram kernel matrix $\mathbf{K}(\mathbf{X}, \mathbf{X})$ is a coupled matrix accounting for a possible uniform correlation among all the output components. It is important to point out that by setting $\omega = 0$, the learning problem turns out to be equivalent to the block diagonal formulation presented before.

A separable kernel structure based on the product of standard radial basis function (RBF) kernels [59] can be used as a trade-off between the uncoupled and mixed kernel function. In this case, the scalar kernels $k_{\boldsymbol{x}}$ and k_o write:

$$k_{\boldsymbol{x}/o}(\boldsymbol{\theta}, \boldsymbol{\theta}') = \exp\left(-\frac{\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|}{\sigma_{\boldsymbol{x}/o}}\right)^2, \tag{5.39}$$

where the pair $(\boldsymbol{\theta}, \boldsymbol{\theta}')$ can be any combination of input or output pairs, $\sigma_{\boldsymbol{x}}$ and σ_o are the hyperparameters of the scalar kernels $k_{\boldsymbol{x}}$ and k_o, respectively. Such hyperparameters are shared by all the output dimensions and can be tuned once via either cross validation or validation set [14], for instance via Bayesian optimization [49].

The idea of using a Gaussian RBF function for the kernel k_o acting on the output components allow to account for possible correlation in the input and output dimension, thus providing an interesting alternative to the uncoupled and mixed kernel functions. Indeed, a large value of σ_o will lead to a strong coupling among the output components, while a small value leads to a block diagonal problem. For the sake of illustration Fig. 5.2 provides a graphical interpretation of block-diagonal (left panel), weakly coupled (central panel) and strongly coupled (right panel) kernel Gram matrices. In such scenario, the coupled matrix $\mathbf{A}$ in Eq. (5.25) of dimension $LD \times LD$ can be inverter via the algorithms presented in Sec. 5.4.1 and Sec. 5.4.2.

5.6 Illustrative Example

This Section provides a more practical interpretation of the mathematical formulation presented from Sec. 5.3 to Sec. 5.5.2 by means of an illustrative example, with the aim of discussing the advantages and drawbacks of the proposed vector-valued KRR. Without loss of generality, the proposed results will focus on the high-speed link in Fig. 5.3.

Specifically, the proposed vector-valued KRR is applied to predict the parametric behavior of the magnitude of the frequency response $y(\boldsymbol{x}; f) = |H(f; \boldsymbol{x})| = |V_{out}(f; \boldsymbol{x})/E(f)|$, as a function of 11 normalized parameters collected in the vector $\boldsymbol{x} = [x_1, \ldots, x_{11}]^T$, in which each parameter $x_i \sim \mathcal{U}([-1, +1])$ is modeled as

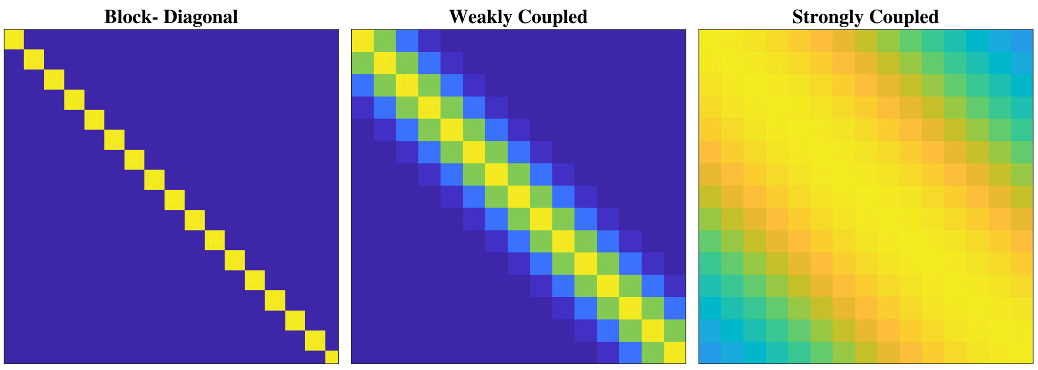

Figure 5.2: Graphical interpretation of the resulting block-diagonal (left panel), weakly coupled (central panel) and strongly coupled (right panel) kernel Gram matrix. Dark color is used for matrix entries with smaller values and bright color is used for matrix entries with higher values.

a normalized uniformly distributed random variable. Additional details about the variability and mean value of the 11 parameters are provided in Tab. 5.1. The high-speed link has been implemented by means of a parametric simulation in MATLAB. Such implementation is then used to generate the training, validation and test set based on a latin hypercube sampling (LHS).

Table 5.1: Mean value and corresponding relative range of variation of the 11 parameters considered for the illustrative example in Sec. 5.6.

Parameter	Mean Value	Uniform Variation
$C_1(x_1)$	1 pF	20%
$C_2(x_2)$	0.5 pF	20%
$L_1(x_3)$	10 nH	20%
$L_2(x_4)$	10 nH	20%
$\varepsilon_r(x_5)$	4.1	1%
$w(x_6)$	252 μm	1%
$t(x_7)$	35 μm	1%
$h(x_8)$	60 μm	1%
$Len_1(x_9)$	5 cm	5%
$Len_2(x_{10})$	3 cm	5%
$Len_3(x_{11})$	3 cm	5%

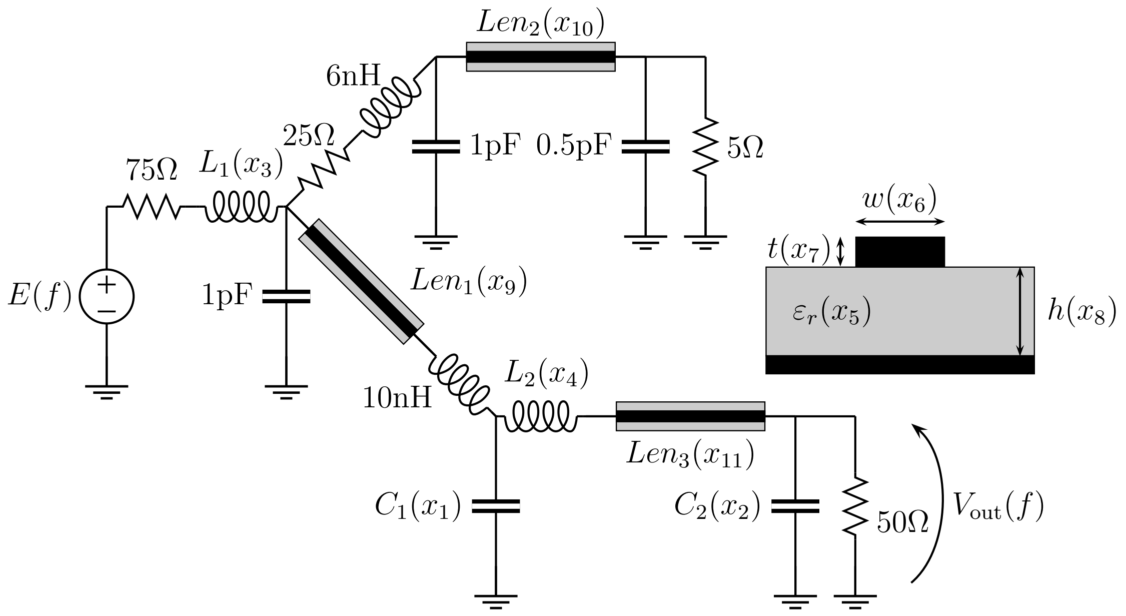

Figure 5.3: Schematic of the high-speed link considered as illustrative example in Sec. 5.6.

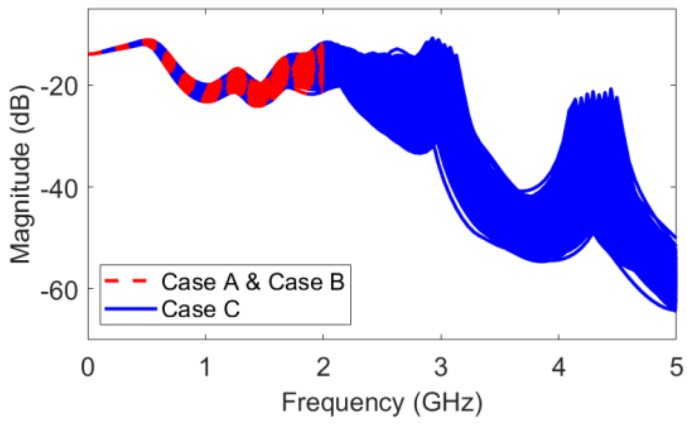

Figure 5.4: Parametric behavior of the magnitude of transfer function $y(\boldsymbol{x}; f)$ of the high-speed link in Fig. 5.3 computed on 1000 test samples for CASE A, B and C.

5.6.1 Performance Analysis

The performance of the proposed KRR are investigated on three different configurations of the proposed test-case:

- CASE A: noise-free training set in a frequency band from 1MHz to 2GHz;

- CASE B: noisy training set[2] in a frequency band from 1MHz to 2GHz;
- CASE C: noise-free training set in a wider frequency band from 1MHz to 5GHz.

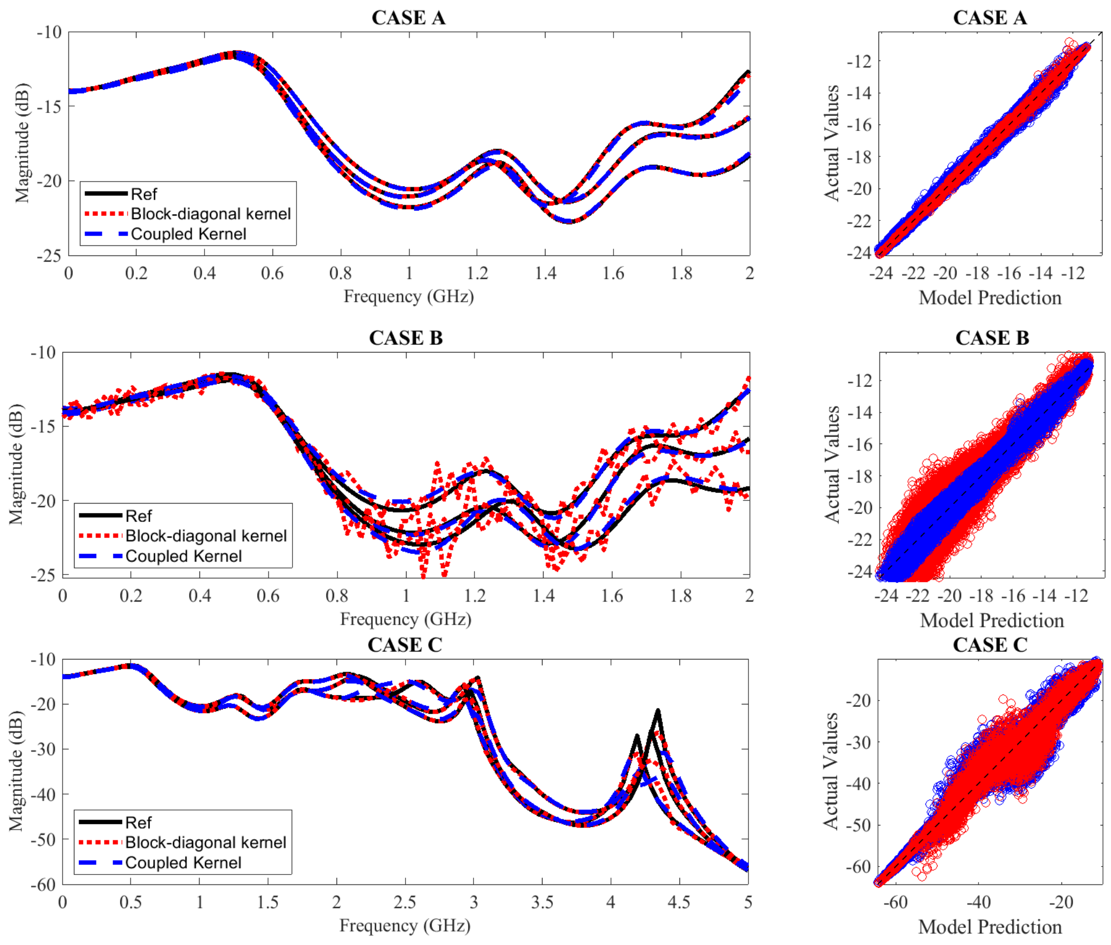

Figure 5.5: Parametric and scatter plots comparing the prediction of the proposed vector-valued KRR with block-diagonal and coupled kernel with the corresponding ones obtained from the computational model for CASE A, B and C on 1000 test-samples.

For each of the above configurations, the parametric behavior of the magnitude of the frequency response $H(f;\boldsymbol{x})$ is investigated for 100 equally spaced frequency points (i.e., the number of outputs is $D = 100$). For the sake of illustration,

[2]Uniformly distributed and uncorrelated noise terms affecting the real and imaginary part of the frequency response $H(f;\boldsymbol{x})$ with an absolute level of 0.05.

Table 5.2: Comparison of training time t_{train}, and relative L_2- and L_∞- error computed for the coupled and uncoupled kernel implementation of the proposed vector-valued KRR. The study was conducted on the illustrative example of Fig. 5.3, for 1000 test samples.

Kernel Matrix	Error	Case A ($L = 150$)	Case B ($L = 150$)	Case C ($L = 300$)
Block-Diagonal Kernel (see Sec. 5.5.1)	L_2	0.43%	5.77%	4.1%
	L_∞	15.69%	31.66%	35%
	t_{train}	13s	16s	23s
Coupled Kernel (see Sec. 5.5.2)	L_2	1.32%	2.54%	6.7%
	L_∞	11.51%	16.74%	48%
	t_{train}	812s	1547s	1527s

Fig. 5.4 shows the spread of 1000 realizations (use as test samples) of the frequency responses for the three considered configurations: CASE A, B and C. The plots highlight the complexity of the three datasets, as well as the strong sensitivity of the model output to the considered parameters.

Two vector-valued KRRs with a block-diagonal and a coupled kernel are trained with L training samples by merans of the GD alghoritm. A validation set [14] with 150 samples is used within a Bayesian optimization [49] to tune the regression hyperparmaters by considering the following intervals: $\sigma_{\boldsymbol{x}} = [10^{-2}, 10^{2}]$, $\sigma_o = [10^{-4}, 10^{-2}]$ and $\lambda = [10^{-3}, 10^{-1}]$ for the coupled kernel matrix and $\sigma_{\boldsymbol{x}} = [10^{-2}, 10^{2}]$, $\sigma_o = [10^{-11}, 2 \cdot 10^{-11}]$ and $\lambda = [10^{-5}, 10^{-2}]$ for the block-diagonal one.

Figure 5.5 provides a comparison between the predictions obtained by the proposed vector-valued KRR trained with a block-diagonal and coupled kernel matrix (see Fig. 5.2) for three different configurations of the input parameters $\boldsymbol{x}$ and the corresponding scatter plots computed on the 1000 test samples. The comparison highlights the excellent capability of trained models to capture the actual variation of the transfer function under modeling for the three considered test-case configurations. Moreover, Table 5.2 presents a quantitative comparison among the proposed implementations in terms of training time t_{train}, relative L_2- and L_∞-error computed in linear scale from the predictions in dB provided by the proposed models on 1000 test samples. The figures of merit provided in the table lead to the following observations:

- *Training time*: as shown in the rows labeled with t_{train} in Tab. 5.2, the computational cost for the training of the vector-valued KRR with coupled kernel is higher than the one required by the block-diagonal implementation. Indeed, as discussed in Sec. 5.5, the computational complexity of the model training depends on the structure of the matrix $\mathbf{A}$ to be inverted in Eq. (5.25), and it is proportional to $\mathcal{O}(KL^2D^2)$ for the implementation of the proposed vector-valued KRR with a coupled kernel and reduces to $\mathcal{O}(KL^2)$ for the uncoupled one based on the GD.

- *Model accuracy*: the KRR implementation based on the block diagonal kernel provides the most accurate model for CASE A and C with a L_2-error below 5%. The high value of the L_∞-error (i.e., the worst-case error) for the CASE C is motivated by the inherently resonance behavior of the frequency response under modeling in the considered frequency bandwidth. On the other hand, the results for CASE B highlight the benefits of the regularization effect on the output dimension introduced by the coupled kernel [70]. Such regularization allows to suppress the sharp fluctuations induced by the noise, thus leading to a more accurate prediction on the noiseless test set.

Summarizing, the block diagonal kernel provides the best trade-off between efficiency and accuracy for noiseless multi-output regression problems, but it is also extremely sensitive to noise. Indeed, the block-diagonal formulation does not directly account for a possible correlation among the output dimensions (i.e., the frequency points of the frequency response), thus increasing the model variance and leading to overfitting issue in the output space. On the contrary, the coupled formulation introduces a regularization effect on the output dimensions, leading to a smoother model in the output space able of heavily suppressing noise fluctuations.

5.6.2 Comparison with State-of-the-art Techniques

The training and validation set generated for the CASE B, are hereafter used to compare the performance of the proposed vector-valued KRR with the ones of a corresponding surrogate model constructed via the combination of the PCA+LS-SVM regression. Also in this case the vector-valued KRR is trained by using a Gaussian RBF kernel for both input parameters and output dimensions and the model hyperparameters are tuned via a 3-fold cross-validation. For the model based on the PCA+LS-SVM regression two different compression levels have been considered by using the relative tolerance of 0.6% and 0.01% for the PCA compression, leading to a compressed model with either 2 or 100 components, respectively.

Table 5.3 provides an exhaustive comparison between the above methods in terms of training time and relative $L2$-norm error computed in dB on 1000 test samples for an increasing number of the training samples (i.e., $L = 30$, 90 and 150). The results show that the computational cost required to built a vector-valued model with the proposed efficient implementation of the multi-output KRR turns out to be comparable with the one required by the PCA+LS-SVM with a

Table 5.3: Relative L2-error and training time computed from the predictions in dB obtained by the proposed vector-valued KRR and PCA+LS-SVM regression trained with increasing number of noisy training samples.

Methods	$L = 30$		$L = 90$		$L = 150$	
	ε_{L2}	t_{train}	ε_{L2}	t_{train}	ε_{L2}	t_{train}
KRR (Proposed)	4.0%	25s	2.8%	38s	2.5%	51s
PCA+LS-SVM (Rel.Tol.=0.01%)	7.5%	6s	5.0%	33s	4.3%	63s
PCA+LS-SVM (Rel.Tol.=0.6%)	6.8%	1.5s	5.9%	1.5s	4.6%	2s

relative tolerance of 0.01%. It is important to notice that the proposed implementation of the vector-valued KRR has a speed up of $\times 30$ with respect to its GD implementation proposed in the previous session and in [67].

Concerning the model accuracy, the errors reported in the table clearly highlight the improved performance of the proposed vector-valued KRR with respect to the ones achieved by the PCA+LS-SVM for all the considered training sets. The above statement is further supported by the parametric plot in Fig. 5.6 computed from the predictions of the considered methods trained with $L = 150$ training samples for two random configurations of the input parameters belonging to the test set. The plot clearly highlights the limited capability of the PCA compression to learn the actual correlation among the output components when the data are corrupted by noise. Indeed, the compressed representation of the training set obtained from the PCA still contains a non-negligible level of noise, which cannot be filtered out even if a small number of components is considered. On the other hand, thanks to the output dimension regularization provided by the kernel k_o in Eq. (5.14), the corresponding model trained via the proposed vector-valued KRR turns out to be more accurate and robust to noise.

Moreover, Figure 5.7 shows a statistical comparison among the methods in terms of the probability density functions (PDFs) computed on 1000 test samples for all the frequency points. Also in this case, it is possible to notice the detrimental effect of the noise on the predictions obtained by the PCA+LS-SVM models, which is responsible for the spurious peaks visible in the corresponding PDFs around -18 and -16 dB.

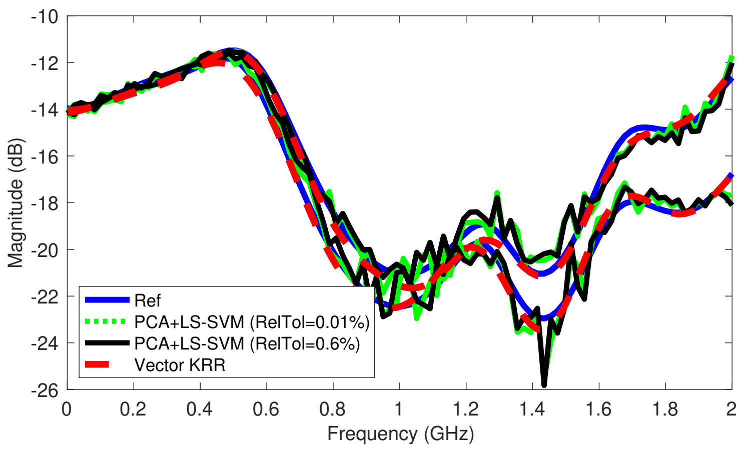

Figure 5.6: Parametric plots comparing the frequency responses predicted by the proposed method and the PCA+LS-SVM surrogate models for 2 different realizations of the input parameters.

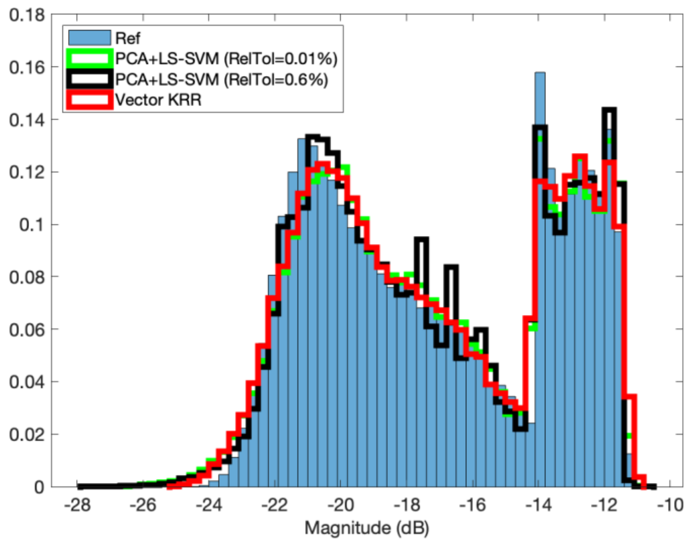

Figure 5.7: Comparison of the PDFs computed from the predictions of the surrogate models built via the proposed and PCA+LS-SVM regression with Tol.= 0.6% and 0.01% on 1000 test samples and for all the frequency points.

5.7 Application Example: Doherty Amplifier

This Section discusses the performance of the proposed method by considering the optimization of the power splitter of the Doherty amplifier illustrated in Fig. 5.8 [71]. Specifically, the proposed vector-valued KRR is used to train a parametric model able to predict the S_{11} and S_{21} of the amplifier, as a function of several coupled and uncoupled parameters listed in Tab. 5.4, characterizing the working point of the amplifier and the geometry of the power splitter (see the red square in Fig. 5.8).

Table 5.4: Mean value and corresponding relative range of variation of the parameters considered for the optimization of the Doherty amplifier in Sec. 5.7.

Parameter	Mean Value	Uniform Variation
W_{TL1}	29.44 mil	50%
W_{TL2}	50.78 mil	50%
W_{TL3}	29.44 mil	50%
W_{TL4}	29.44 mil	50%
W_{TL5}	50.78 mil	50%
W_{TL6}	29.44 mil	50%
W_{TL7}	29.44 mil	50%
W_{TL8}	29.44 mil	50%
L_{TL1} & L_{TL2}	646.85 mil	2.5%
L_{TL4} & L_{TL6}	620.9 mil	2.5%
L_{TL3} & L_{TL5}	646.85 mil	2.5%
L_{TL7} & L_{TL8}	646.85 mil	2.5%
V_{dc1}	2.45 v	5%
V_{dc2}	7 v	5%
V_{dc3} & V_{dc4}	28 v	5%

First of all, the schematic in Fig. 5.8 has been implemented as a parametric simulation in ADS. For any configuration of the input parameters, the ADS simulation provides the frequency responses of the scattering parameters S_{11} and S_{21} computed for $D = 1101$ frequency points in a bandwidth from 1.9 GHz to 3 GHz. A set of $L = 700$ training samples and 100 validation samples have been generated via a LHS.

Such samples have been used to train a parametric model for the scattering parameters of interest via the KRR with the block-diagonal kernel. The model training takes 220 s. The obtained models are then used together with a "brute force" optimization algorithm based on a random grid search [72] implemented in MATLAB, with the aim of optimizing the amplifier parameters in order to meet the following constraints:

$$S_{11} \leq -10\text{dB for } 2.4\text{GHz} \leq f \leq 2.6\text{GHz} \tag{5.40a}$$

$$10\text{dB } \leq S_{21} \leq 12\text{dB for } 2.1\text{GHz} \leq f \leq 2.9\text{GHz}. \tag{5.40b}$$

Figure 5.9 compares the S_{11} and S_{21} scattering parameters estimated after the optimization via the proposed vector-valued model with those obtained from the

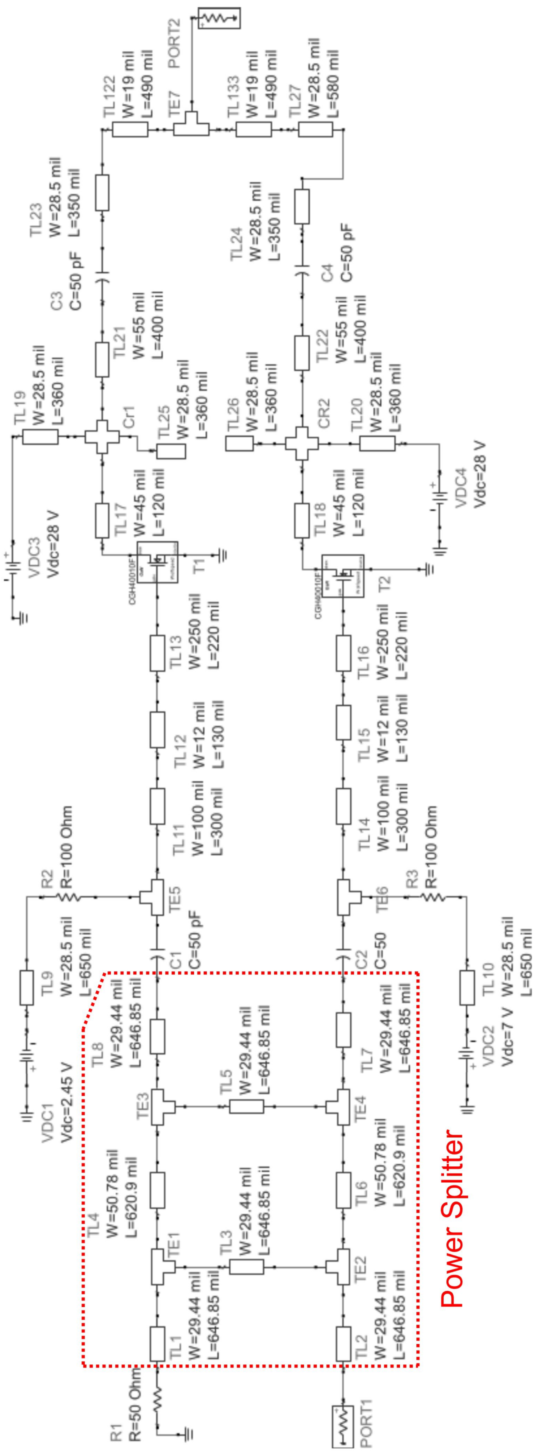

Figure 5.8: Schematic of the Doherty amplifier considered in Sec. 5.7 (inspired by [71]).

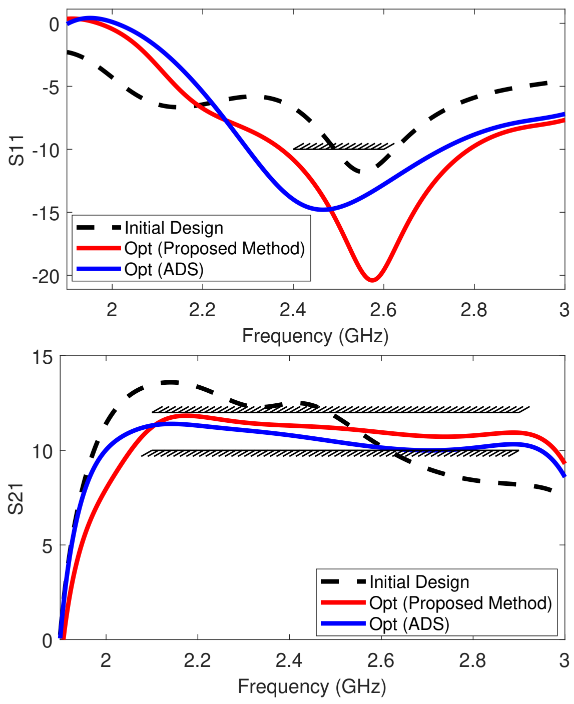

Figure 5.9: Scattering parameters of the Doherty amplifier presented in Sec. 5.7 obtained from the initial design (dashed black line) and after optimization carried out via the ADS random optimizer (solid red line) and the proposed model (solid blue line).

initial design. Moreover, the plots show the corresponding results obtained from the random optimizer (default option) in ADS after 2800 iterations. The results clearly highlight the strong agreement and consistence between the optimization results obtained via the proposed modeling scheme and the ones obtained from the ADS optimization. Concerning the computational cost, the overall optimization with our advocated model takes 25s. On the other hand, the corresponding optimization in ADS requires 2800 iterations and takes 386 seconds. The proposed simulation approach leads to a speed up 15× with respect to ADS. It is important to stress that the obtained speed up is mitigated by the relatively fast simulation time required by the ADS circuital solver when it is used in small-signal analysis. Moreover, unlike the ADS optimizer, the obtained model turns out to be independent from the optimization constraints and therefore can be suitably adopted as it is to meet different optimization constraints, as well as for the stochastic analysis within the uncertainty quantification scenario [7, 59].

5.8 Application Example: 2-GHz Low-Noise Amplifier (LNA)

The performance of the proposed technique has been further evaluated on the UQ of the scattering parameters of the 2-GHz low-noise amplifier (LNA) shown in Fig. 5.10 by considering 25 Gaussian stochastic variables affecting the parasitic resistances, capacitances and inductances of the BJT, its forward current gain, all the lumped components in the amplifier schematic, and the widths of the microstrip lines, each with a 10% relative standard deviation [73, 74].

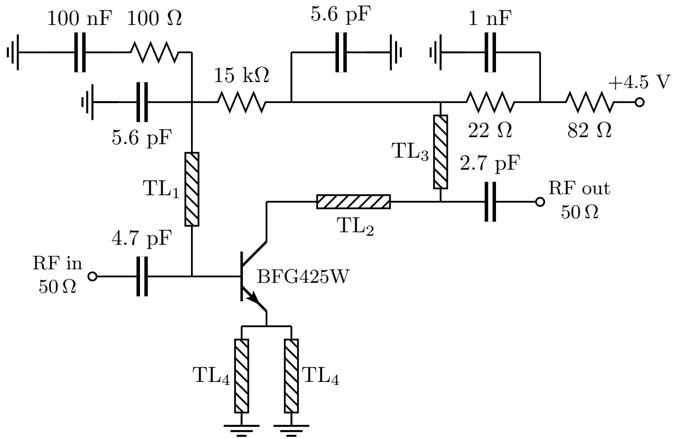

Figure 5.10: 2-GHz BJT LNA.

The considered test case has been implemented as a parametric small-signal AC analysis in HSPICE. Such implementation allows computing the two-port scattering parameters S_{11} and S_{21} of the LNA at 201 frequency points for any configuration of the 25 random parameters. The HSPICE simulations have been used to generate three training sets with an increasing number of samples (i.e., $L = 30$, 50, and 100) based on a LHS and to run a 1000-sample MC simulation that is used hereafter as a reference for the proposed statistical analysis. Two independent surrogate models, one for S_{11} and one for S_{21}, have been trained via the proposed implementation of the vector-valued KRR presented in Section 5.3.

Table 5.5 reports the performance of the obtained surrogate models in terms of training time and relative L2-norm error computed on a test set collecting the results of a 1000-sample MC simulation for the parameters S_{11} and S_{21} by considering an increasing number of the training samples (i.e., $L = 30$, 50, and 100). The results show a constant reduction of the model error (i.e., the relative L2-error) with respect to the number of training samples (i.e., L), thus highlighting the capability of the proposed vector-valued KRR of learning the actual information provided by the training set. Concerning the computational cost, the training time required to build the proposed vector-valued surrogate models is less than 7 min for all the

Table 5.5: Training time and relative L2-error computed from the frequency-domain samples of the predictions of the surrogate models trained with an increasing number of training samples via the proposed implementation of the vector-valued KRR for the S_{11} and S_{21} parameters by considering 1000 test samples.

Parameter	$L = 30$		$L = 50$		$L = 100$	
	t_{train}	L2-error	t_{train}	L2-error	t_{train}	L2-error
S_{11}	74s	6.15%	115s	3.58%	205s	2.70%
S_{21}	78s	1.96%	118s	1.54%	216s	1.16%

considered modeling scenarios. After the training, the evaluation of the obtained model on the 1000 test samples required less than 20s, while the corresponding MC simulation requires 1131s.

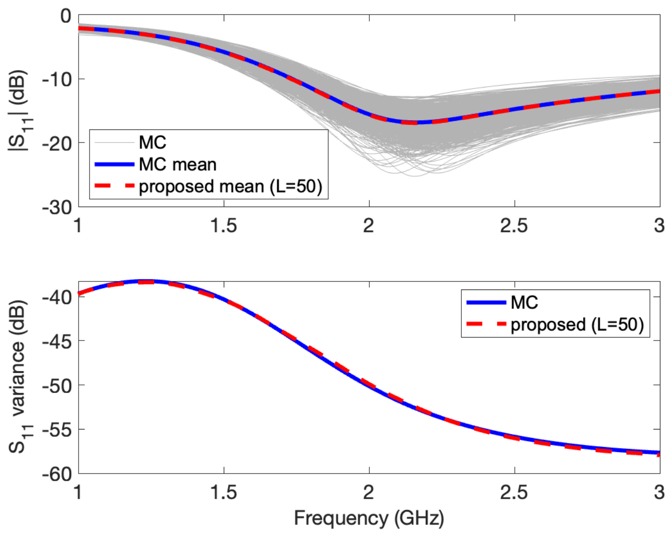

Figure 5.11: S_{11} parameter of the LNA in Fig. 5.10. Top panel: the gray lines show the magnitude of the S_{11} computed from a 1000-sample MC simulation. The blue solid and the red dashed lines are the magnitude of the average S_{11} obtained from the MC samples and the proposed model, respectively. Bottom panel: the blue solid and the red dashed lines are the magnitude of the S_{11} variance obtained from the MC samples and the proposed models, respectively.

Figures 5.11 and 5.12 compare the mean (top panels) and the variance (bottom panels) of the magnitude value of the S_{11} and S_{21} parameters predicted by the proposed surrogate models trained with $L = 50$ training samples against the corresponding results computed via the MC samples. The results show an excellent

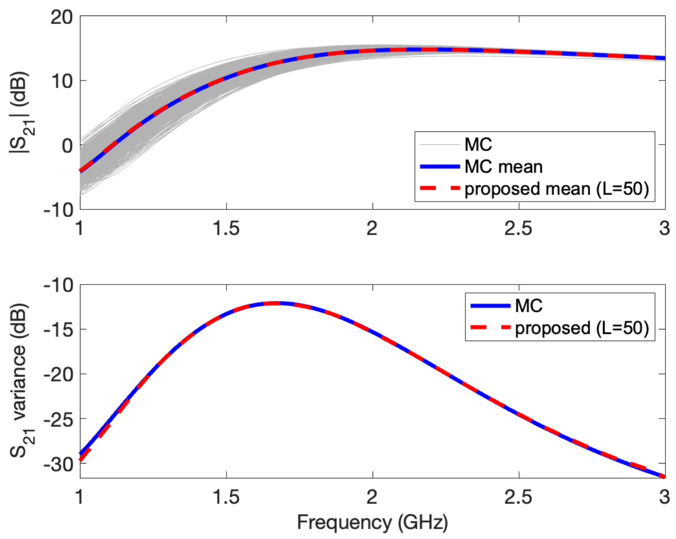

Figure 5.12: S_{21} parameter of the LNA in Fig. 5.10. Top panel: the gray lines show the magnitude of the S_{21} computed from a 1000-sample MC simulation. The blue solid and the red dashed lines are the magnitude of the average S_{21} obtained from the MC samples and the proposed models, respectively. Bottom panel: the blue solid and the red dashed lines are the magnitude of the S_{21} variance obtained from the MC samples and the proposed models, respectively.

agreement between the mean value and the variance predicted by the proposed surrogate model and the ones computed from the MC samples, being the two sets of curves almost perfectly overlapped. Furthermore, Fig. 5.13 shows the probability density function (PDF) of the S_{11} and S_{21} parameters computed at $f_0 = 2\,\text{GHz}$. Also in this case, the histograms predicted by the proposed models are in perfect agreement with the corresponding ones calculated from the MC simulation, thus confirming the excellent performance of the proposed approach in the UQ scenario.

5.9 Summary

This Chapter presented a generalized vector-valued formulation of the KRR, able to deal with the inherently multi-output nature shared by most of the electronic applications. The proposed vector-valued KRR provides a generalization of the mathematical framework used by state-of-the-art scalar kernel regressions and can be seen as an alternative to the data compression strategy presented in Chapter 4. The mathematical framework of the vector-valued KRR has been discussed in detail, together with possible alternatives for the kernel functions and training schemes. The feasibility and the strength of the proposed approach have been investigated on an illustrative example consisting of an high-speed link, the

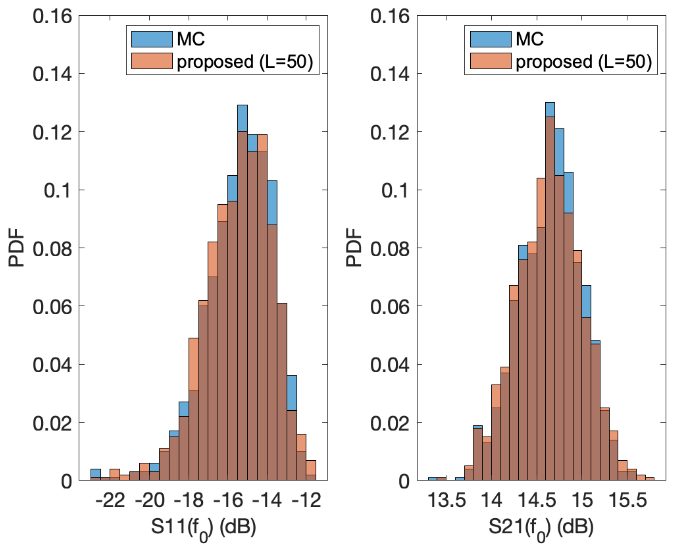

Figure 5.13: Comparison of the PDFs of the S_{11} and S_{21} parameters at the frequency $f_0 = 2\,\text{GHz}$ computed from the predictions of the surrogate models built via the proposed vector-valued KRR evaluated on 1000 test samples and the corresponding ones computed from a 1000-sample MC simulation.

optimization of a Doherty amplifier and the UQ of the scattering parameters of a LNA.

Chapter 6

Conclusions & Future Work

This dissertation focused on the development of modeling techniques for the construction of efficient and accurate surrogate models for the outputs of interest in electronic devices and circuits. The problem statement and the considered modeling framework have been introduced in Chapter. 1. A brief overview of state-of-the-art regression approaches such as: standard regressions based on linear model, kernel machines regression and ANN has been presented in Chapter. 2, along with several considerations about their advantages and drawback supported by two examples.

Chapter 3 addressed a very relevant issue for the surrogate modeling construction such as the possibility of reducing the computational cost arising from the training set generation. The proposed solution is based on the idea PKBML. the underlying idea is to train the surrogate model with an heterogeneous training set combining the prediction of a computational expensive high-fidelity model and the ones of a more efficient low-fidelity model. The obtained dataset, after some manipulation, can be used as a training set for any regression algorithm. Two different implementation of the PKBML have been investigated, such as the PKI and SD, and their performance have been evaluated on a realistic example consisting of a hybrid copper-graphene on-chip interconnects.

Chapter 4 provides an unconventional solutions for two main limitations related to the use of kernel-machine regression in EM applications. Indeed, despite the advantages shown by kernel-machine regressions with respect to other regression techniques, in regression problems with dozens input parameters and with a "small" number of training samples, such as: their cheap computational cost and improved accuracy, their plain implementation is limited to scalar- and real-valued problems. The above issues motivated the complex formulation of the LS-SVM regression and the data compression strategy developed within the above Chapter. The performance and the effectiveness of the proposed solutions are then investigated on two realistic examples consisting of a serpentine EM structure and a high-speed link.

Chapter 5 presents a generalized vector-valued formulation of the KRR able to address the inherent vector-valued structure characterizing most of the electronic and EM applications, thus bridging the gap in terms of flexibility with respect to ANN structures. The proposed formulation has been widely discussed along the Chapter, together with several training schemes and multi-output kernel functions. Also in this case, the effectiveness of the proposed methodology has been investigated via several examples such as: the parametric modeling of an high speed link, the optimization of a Doherty amplifier and the UQ of a LNA.

Summarizing, this dissertation discussed and tried to address some relevant issues and challenges encountered during the construction of surrogate models of the responses of electronic devices and circuits with the help of ML-based techniques. Despite the encouraging results, additional work must be done in order to provide completely automatic, general purpose and robust modeling framework, such as:

- is there any clever strategy to understand how many training and testing samples must be used and how to select them?
- what is the limitation of the proposed techniques in terms of number of input parameters?
- how large can be the range of variability of the input parameters?
- can the multi-output kernel be automatically adapted to the training dataset?

The above questions are rather challenging and can be seen as stimuli for future works and research activities.

References

[1] Robert Spence and Randeep Singh Soin. *Tolerance design of electronic circuits*. World Scientific, 1997.

[2] Bobak Shahriari et al. “Taking the human out of the loop: A review of Bayesian optimization”. In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[3] Kai-Tai Fang, Runze Li, and Agus Sudjianto. *Design and modeling for computer experiments*. Chapman and Hall/CRC, 2005.

[4] Jean-Marc Bourinet. “Reliability analysis and optimal design under uncertainty-Focus on adaptive surrogate-based approaches”. PhD thesis. Université Clermont Auvergne, 2018.

[5] Bruno Sudret et al. “Surrogate models for uncertainty quantification and design optimization”. In: *14ème colloque national en calcul des structures (CSMA 2019)*. ETH Zurich, Chair of Risk, Safety and Uncertainty Quantification. 2019.

[6] RY Rubinstein. *Simulation and monte carlo method. new york: John & wiley & sons*. 1981.

[7] Riccardo Trinchero and Flavio Canavero. “Machine learning regression techniques for the modeling of complex systems: An overview”. In: *IEEE Electromagnetic Compatibility Magazine* 10.4 (2021), pp. 71–79.

[8] Michael D McKay, Richard J Beckman, and William J Conover. “A comparison of three methods for selecting values of input variables in the analysis of output from a computer code”. In: *Technometrics* 42.1 (2000), pp. 55–61.

[9] Jing Jin et al. “Deep neural network technique for high-dimensional microwave modeling and applications to parameter extraction of microwave filters”. In: *IEEE Transactions on Microwave Theory and Techniques* 67.10 (2019), pp. 4140–4155.

[10] Slawomir Koziel, Anna Pietrenko-Dabrowska, et al. “Design-oriented two-stage surrogate modeling of miniaturized microstrip circuits with dimensionality reduction”. In: *IEEE Access* 8 (2020), pp. 121744–121754.

[11] Tommaso Bradde et al. “Data-driven extraction of uniformly stable and passive parameterized macromodels”. In: *IEEE Access* 10 (2022), pp. 15786–15804.

[12] Alessandro Zanco and Stefano Grivet-Talocia. “Toward fully automated high-dimensional parameterized macromodeling”. In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 11.9 (2021), pp. 1402–1416.

[13] Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. “A generalized representer theorem”. In: *Computational Learning Theory: 14th Annual Conference on Computational Learning Theory, COLT 2001 and 5th European Conference on Computational Learning Theory, EuroCOLT 2001 Amsterdam, The Netherlands, July 16–19, 2001 Proceedings 14*. Springer. 2001, pp. 416–426.

[14] Benyamin Ghojogh and Mark Crowley. “The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial”. In: *arXiv preprint arXiv:1905.12787* (2019).

[15] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[16] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, 2013. ISBN: 978-1-4614-6848-6.

[17] Qi-jun Zhang and Kuldip C Gupta. *Neural networks for RF and microwave design (Book+ Neuromodeler Disk)*. Artech House, Inc., 2000.

[18] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[19] Madhavan Swaminathan et al. “Demystifying machine learning for signal and power integrity problems in packaging”. In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 10.8 (2020), pp. 1276–1295.

[20] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[21] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[22] DE Rumelhart. “GE H inton, and RJ Williams”. In: *Learning internal representations by error propagation. InD. E. Rumelhart and JL McClelland, editors, Parallel Distribu ted Processing: Ex plorations in the Microstr u ct u reof C ognition. B radford B ooks/MITPress, Cambridge, Mass* (1986).

[23] Suyash Kushwaha et al. “Fast extraction of per-unit-length parameters of hybrid copper-graphene interconnects via generalized knowledge based machine learning”. In: *2021 IEEE 30th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*. IEEE. 2021, pp. 1–3.

[24] Suyash Kushwaha et al. “Comparative Analysis of Prior Knowledge-Based Machine Learning Metamodels for Modeling Hybrid Copper–Graphene On-Chip Interconnects”. In: *IEEE Transactions on Electromagnetic Compatibility* 64.6 (2022), pp. 2249–2260.

[25] Vladimir Vapnik. *The nature of statistical learning theory.* Springer science & business media, 1999.

[26] Vladimir N Vapnik. “An overview of statistical learning theory”. In: *IEEE transactions on neural networks* 10.5 (1999), pp. 988–999.

[27] VR Kohestani and M Hassanlourad. “Modeling the mechanical behavior of carbonate sands using artificial neural networks and support vector machines”. In: *International Journal of Geomechanics* 16.1 (2016), p. 04015038.

[28] Alex J Smola and Bernhard Schölkopf. “A tutorial on support vector regression”. In: *Statistics and computing* 14 (2004), pp. 199–222.

[29] Vladimir Cherkassky and Yunqian Ma. “Practical selection of SVM parameters and noise estimation for SVM regression”. In: *Neural networks* 17.1 (2004), pp. 113–126.

[30] Johan AK Suykens and Joos Vandewalle. “Least squares support vector machine classifiers”. In: *Neural processing letters* 9 (1999), pp. 293–300.

[31] version 1.8 LS-SVMlab. *Available online: http://www.esat.kuleuven.be/sista/lssvmlab/.* Department of Electrical Engineering (ESAT), Katholieke Universiteit Leuven: Leuven, Belgium, 2011.

[32] Camelia Gabriel. *Compilation of the dielectric properties of body tissues at RF and microwave frequencies.* Tech. rep. King’s coll london (United Kingdom) dept of physics, 1996.

[33] Kris De Brabanter et al. *LS-SVMlab toolbox user’s guide: version 1.7.* Katholieke Universiteit Leuven, 2010.

[34] Anand Veluswami, Michel S Nakhla, and Qi-Jun Zhang. “The application of neural networks to EM-based simulation and optimization of interconnects in high-speed VLSI circuits”. In: *IEEE transactions on Microwave Theory and Techniques* 45.5 (1997), pp. 712–723.

[35] Joongheon Kim, Giuseppe Caire, and Andreas F Molisch. “Quality-aware streaming and scheduling for device-to-device video delivery”. In: *IEEE/ACM Transactions on Networking* 24.4 (2015), pp. 2319–2331.

[36] PM Watson, KC Gupta, and RL Mahajan. "Development of knowledge based artificial neural network models for microwave components". In: *1998 IEEE MTT-S International Microwave Symposium Digest (Cat. No. 98CH36192).* Vol. 1. IEEE. 1998, pp. 9–12.

[37] Zi-Han Cheng et al. "Analysis of Cu-graphene interconnects". In: *IEEE Access* 6 (2018), pp. 53499–53508.

[38] Rafael Boloix-Tortosa et al. "Widely linear complex-valued kernel methods for regression". In: *IEEE Transactions on Signal Processing* 65.19 (2017), pp. 5240–5248.

[39] Rafael Boloix-Tortosa, Juan José Murillo-Fuentes, and Sotirios A Tsaftaris. "The generalized complex kernel least-mean-square algorithm". In: *IEEE Transactions on Signal Processing* 67.20 (2019), pp. 5213–5222.

[40] Tokunbo Ogunfunmi and Thomas Paul. "On the complex kernel-based adaptive filter". In: *2011 IEEE International Symposium of Circuits and Systems (ISCAS).* IEEE. 2011, pp. 1263–1266.

[41] Felipe A Tobar, Anthony Kuh, and Danilo P Mandic. "A novel augmented complex valued kernel LMS". In: *2012 IEEE 7th Sensor Array and Multichannel Signal Processing Workshop (SAM).* IEEE. 2012, pp. 473–476.

[42] Akira Hirose. "Complex-valued neural networks: Advances and applications". In: (2013).

[43] Pantelis Bouboulis et al. "Complex support vector machines for regression and quaternary classification". In: *IEEE transactions on neural networks and learning systems* 26.6 (2014), pp. 1260–1274.

[44] Simone Scardapane et al. "Widely linear kernels for complex-valued kernel activation functions". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE. 2019, pp. 8528–8532.

[45] Felipe Treviso, Riccardo Trinchero, and Flavio G Canavero. "Multiple delay identification in long interconnects via LS-SVM regression". In: *IEEE Access* 9 (2021), pp. 39028–39042.

[46] Rafael Boloix-Tortosa et al. "Complex Gaussian processes for regression". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.11 (2018), pp. 5499–5511.

[47] Donato Posa. "Parametric families for complex valued covariance functions: Some results, an overview and critical aspects". In: *Spatial Statistics* 39 (2020), p. 100473.

[48] Sandra De Iaco, Monica Palma, and Donato Posa. “Covariance functions and models for complex-valued random fields”. In: *Stochastic Environmental Research and Risk Assessment* 17 (2003), pp. 145–156.

[49] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. “Practical bayesian optimization of machine learning algorithms”. In: *Advances in neural information processing systems* 25 (2012).

[50] Jing Geng et al. “Support vector machine regression (SVR)-based nonlinear modeling of radiometric transforming relation for the coarse-resolution data-referenced relative radiometric normalization (RRN)”. In: *Geo-Spatial Information Science* 23.3 (2020), pp. 237–247.

[51] Nastaran Soleimani, Riccardo Trinchero, and Flavio Canavero. “Application of different learning methods for the modelling of microstrip characteristics”. In: *2020 IEEE Electrical Design of Advanced Packaging and Systems (EDAPS)*. IEEE. 2020, pp. 1–3.

[52] Paolo Manfredi and Riccardo Trinchero. “A data compression strategy for the efficient uncertainty quantification of time-domain circuit responses”. In: *IEEE Access* 8 (2020), pp. 92019–92027.

[53] Athanasios Papaioannou and Stefanos Zafeiriou. “Principal component analysis with complex kernel: The widely linear model”. In: *IEEE Transactions on Neural networks and Learning systems* 25.9 (2013), pp. 1719–1726.

[54] Paolo Manfredi and Stefano Grivet-Talocia. “Compressed Stochastic Macromodeling of Electrical Systems via Rational Polynomial Chaos and Principal Component Analysis”. In: *2021 Asia-Pacific International Symposium on Electromagnetic Compatibility (APEMC)*. IEEE. 2021, pp. 1–3.

[55] Mohsen Ahmadi et al. “Detection of brain lesion location in MRI images using convolutional neural network and robust PCA”. In: *International journal of neuroscience* 133.1 (2023), pp. 55–66.

[56] René Vidal et al. *Principal component analysis*. Springer, 2016.

[57] Wei-Shan Soh et al. “Comprehensive analysis of serpentine line design”. In: *2009 Asia Pacific Microwave Conference*. IEEE. 2009, pp. 1285–1288.

[58] Riccardo Trinchero et al. “Machine learning and uncertainty quantification for surrogate models of integrated devices with a large number of parameters”. In: *IEEE Access* 7 (2018), pp. 4056–4066.

[59] Nastaran Soleimani, Riccardo Trinchero, and Flavio Canavero. “Vector-Valued Kernel Ridge Regression for the Modeling of High-Speed Links”. In: *2022 IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO)*. IEEE. 2022, pp. 1–4.

[60] JAK Suykens et al. "Least squares support vector machines, World Scientific Publishing, Singapore". In: (2002).

[61] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. "Falkon: An optimal large scale kernel method". In: *Advances in neural information processing systems* 30 (2017).

[62] Nastaran Soleimani and Riccardo Trinchero. "Compressed complex-valued least squares support vector machine regression for modeling of the frequency-domain responses of electromagnetic structures". In: *Electronics* 11.4 (2022), p. 551.

[63] Mauricio A Alvarez, Lorenzo Rosasco, Neil D Lawrence, et al. "Kernels for vector-valued functions: A review". In: *Foundations and Trends® in Machine Learning* 4.3 (2012), pp. 195–266.

[64] Charles A Micchelli and Massimiliano Pontil. "On learning vector-valued functions". In: *Neural computation* 17.1 (2005), pp. 177–204.

[65] Charles Micchelli and Massimiliano Pontil. "Kernels for Multi–task Learning". In: *Advances in neural information processing systems* 17 (2004).

[66] Francesco Dinuzzo et al. "Learning output kernels with block coordinate descent". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 49–56.

[67] Nastaran Soleimani, Riccardo Trinchero, and Flavio G Canavero. "Bridging the Gap Between Artificial Neural Networks and Kernel Regressions for Vector-Valued Problems in Microwave Applications". In: *IEEE Transactions on Microwave Theory and Techniques* (2023).

[68] Roger A Horn, Roger A Horn, and Charles R Johnson. *Topics in matrix analysis*. Cambridge university press, 1994.

[69] Magnus R Hestenes, Eduard Stiefel, et al. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49.6 (1952), pp. 409–436.

[70] Luca Baldassarre et al. "Multi-output learning via spectral filtering". In: *Machine learning* 87 (2012), pp. 259–301.

[71] Andrei Grebennikov and James Wong. "A dual-band parallel Doherty power amplifier for wireless applications". In: *IEEE Transactions on Microwave Theory and Techniques* 60.10 (2012), pp. 3214–3222.

[72] Antonio Guarino et al. "A fast fuel cell parametric identification approach based on machine learning inverse models". In: *Energy* 239 (2022), p. 122140.

[73] Paolo Manfredi et al. "Generalized decoupled polynomial chaos for nonlinear circuits with many random parameters". In: *IEEE Microwave and Wireless Components Letters* 25.8 (2015), pp. 505–507.

[74] Paolo Manfredi. “Probabilistic Uncertainty Quantification of Microwave Circuits Using Gaussian Processes”. In: *IEEE Transactions on Microwave Theory and Techniques* (2022).

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.