ETH zürich

UQLab User Manual
Support Vector Machines For Regression

M. Moustapha, C. Lataniotis, S. Marelli, B. Sudret

Chair of Risk, Safety and Uncertainty Quantification
Stefano-Franscini-Platz 5
CH-8093 Zürich

Risk, Safety & Uncertainty Quantification

How to cite UQLAB

S. Marelli, and B. Sudret, UQLab: A framework for uncertainty quantification in Matlab, Proc. 2nd Int. Conf. on Vulnerability, Risk Analysis and Management (ICVRAM2014), Liverpool, United Kingdom, 2014, 2554-2563.

How to cite this manual

M. Moustapha, C. Lataniotis, S. Marelli, B. Sudret, UQLab user manual – Support vector machines for regression, Report # UQLab-V1.3-111, Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland, 2019

BIBTEX entry

```
@TechReport{UQdoc_13_111,
author = {Moustapha, M. and Lataniotis, C. and Marelli, S. and Sudret, B.},
title = {{UQLab user manual -- Support vector machines for regression}},
institution = {Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich,
Switzerland},
year = {2019},
note = {Report \# UQLab-V1.3-111}
}
```

Document Data Sheet

Document Ref.	UQLAB-V1.3-111
Title:	UQLAB user manual – Support vector machines for regression
Authors:	M. Moustapha, C. Lataniotis, S. Marelli, B. Sudret Chair of Risk, Safety and Uncertainty Quantification, ETH Zurich, Switzerland
Date:	19/09/2019

Doc. Version	Date	Comments
V1.3	19/09/2019	UQLAB V1.3 release
V1.2	22/02/2019	UQLAB V1.2 release • added automatic calculation of validation error if a validation set is provided
V1.1	05/07/2018	Initial release

Abstract

Support vector machines for regression is a metamodelling technique that can be used to approximate an unknown or expensive-to-evaluate model. The UQLAB implementation of support vector regression (SVR) allows the user to define a model given a set of training points in an easy fashion. A wide variety of options are available and can be configured to tune the fitting of the SVR model.
This manual is divided into three parts:

- A brief introduction to the main concepts of SVR together with the basic equations needed to build a model, which also includes relevant literature;
- A detailed usage of the UQLAB features for SVR throughout the implementation of basic examples;
- A comprehensive reference list of the available options for SVR in UQLAB.

Keywords: Machine learning, Surrogate modelling, Support vector machines, Regression

Contents

Chapter 1

Theory

1.1 Introduction

Learning from data has become a paramount task for many engineering applications that require the simulation of complex systems behavior. Despite the exponential growth in available computational power, the high level of fidelity required by designers has led to even more computationally demanding numerical models. This has resulted in a rise of simulation time, thus making engineering analyses such as reliability assessment or optimization unaffordable. Machine learning techniques provide tools to develop computationally inexpensive proxies to these complex numerical models. This is achieved by induction over a limited set of data, the so-called *experimental design*.

Support vector machines represent a class of learning techniques for classification and regression tasks developed by Vapnik (1995) over the principle of structural risk minimization. This principle provides support vector machines with significant generalization capabilities, thus making them less likely to *overfit* data. The mechanisms underlying the structural risk minimization principle are beyond the scope of this work. The reader may refer to Smola and Schölkopf (2004); Vapnik (1995) and Bourinet (2018) (in the context of surrogate modelling for uncertainty quantification) for an in-depth analysis. In the first part of this manual, we introduce the main concepts behind support vector machines for regression and provide the reader with the associated relevant literature. Chapters 2 and 3 respectively show the usage of SVM in UQLAB and a comprehensive list of available options.

1.2 SVR basics: linear regression

Support vector regression (SVR) attempts to learn a functional relationship between some pairs of inputs $\boldsymbol{x} \in \mathcal{D}_{\boldsymbol{X}} \subset \mathbb{R}^M$ and outputs $y \in \mathbb{R}$ given an experimental design $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and the corresponding model responses $\mathcal{Y} = \{y_1, \ldots, y_n\}$. In its simplest form, SVR achieves this through a linear relationship which can be cast as follows:

$$\mathcal{M}^{\text{SVR}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b, \tag{1.1}$$

where $\boldsymbol{w}$ is a vector of weight coefficients and b is an offset parameter to be estimated.

The parameters in Eq. (1.1) are set by minimizing a *loss function*. There exists a large variety of loss functions, each of them giving rise to different formulations of support vector regression (Gunn, 1998). We consider here the most widely used one, the so-called ε-insensitive loss function which reads for linear penalization (Smola and Schölkopf, 2004):

$$\mathcal{L}_1^{\varepsilon}(\boldsymbol{x};\, y) = \begin{cases} 0 & \text{if} \quad \left|\mathcal{M}^{\text{SVR}}(\boldsymbol{x}) - y\right| < \varepsilon, \\ \left|\mathcal{M}^{\text{SVR}}(\boldsymbol{x}) - y\right| - \varepsilon & \text{otherwise.} \end{cases} \tag{1.2}$$

A linear regressor considering this loss function is illustrated in Figure 1. The so-called ε-insensitive tube is highlighted in gray. Any point that falls within this tube is not penalized. In contrast, points whose discrepancy with the linear regressor is above ε are penalized linearly. This scheme is known as L_1-SVR.

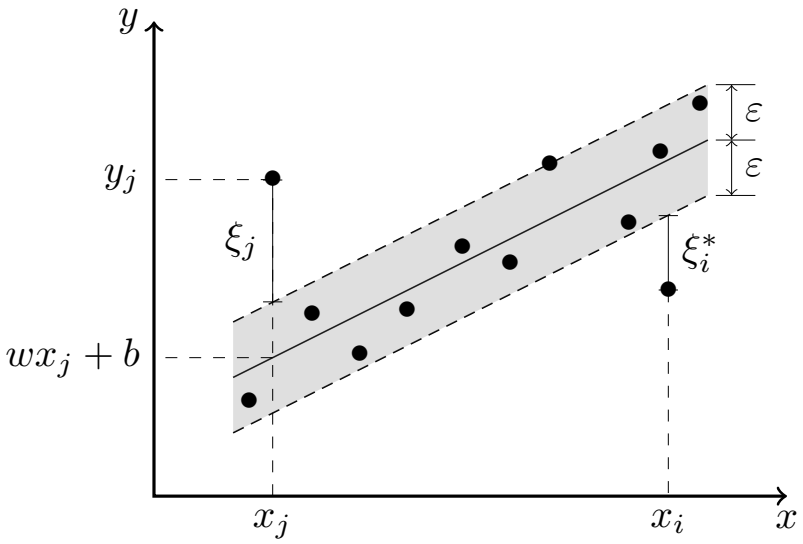

Figure 1: Concept of support vector regression: only the vectors outside the ε-insensitive tube (gray-shaded area) are penalized.

Similarly, L_2-SVR is another well-known formulation which results from a quadratic ε-insensitive loss function:

$$\mathcal{L}_2^{\varepsilon}(\boldsymbol{x}) = \begin{cases} 0 & \text{if} \quad \left|\mathcal{M}^{\text{SVR}}(\boldsymbol{x}) - y\right| < \varepsilon, \\ \left(\left|\mathcal{M}^{\text{SVR}}(\boldsymbol{x}) - y\right| - \varepsilon\right)^2 & \text{otherwise.} \end{cases} \tag{1.3}$$

The two penalization schemes are illustrated in Figure 2. To construct the estimator corresponding to L_1-SVR, one has to solve the following optimization problem (Smola and

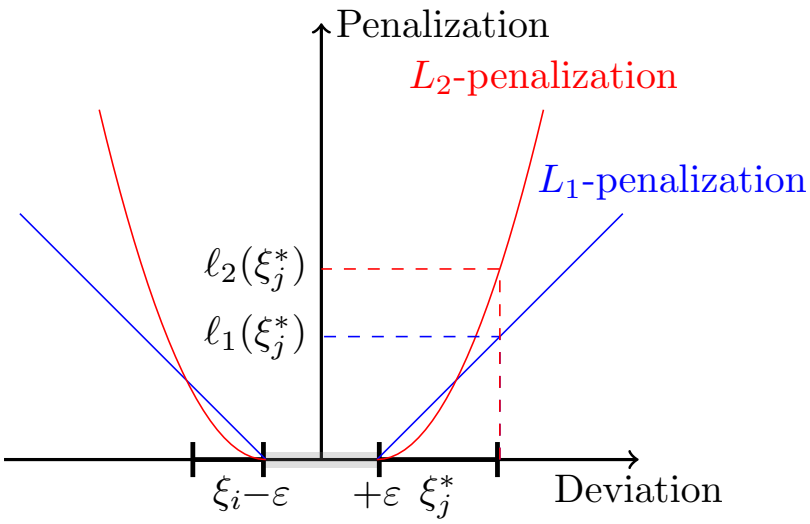

Figure 2: Penalization of deviations larger than ε for L_1- and L_2-SVR.

Schölkopf, 2004; Bourinet, 2018):

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\left(\xi_i + \xi_i^*\right), \\ \text{subject to} \quad & y_i - \boldsymbol{w}^T\boldsymbol{x}_i - b \leq \varepsilon + \xi_i, \\ & \boldsymbol{w}^T\boldsymbol{x}_i + b - y_i \leq \varepsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0, \end{aligned} \tag{1.4}$$

where ξ_i and ξ_i^* are the so-called *slack variables*, which measure the deviation from the insensitive tube and $C \in \mathbb{R}^+$ is a regularization parameter.
In practice, it is more convenient to solve this problem in its dual form, which reads:

$$\arg\max_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\alpha_i - \alpha_i^*\right)\left(\alpha_j - \alpha_j^*\right)\boldsymbol{x}_i^T\boldsymbol{x}_j - \sum_{i=1}^{N}\left(\alpha_i + \alpha_i^*\right)\varepsilon + \sum_{i=1}^{N}\left(\alpha_i - \alpha_i^*\right)y_i, \tag{1.5}$$

under the constraints:

$$\sum_{i=1}^{N}\left(\alpha_i - \alpha_i^*\right) = 0 \quad \text{and} \quad 0 \leq \alpha_i, \alpha_i^* \leq C, \quad i = \{1, \ldots, N\}, \tag{1.6}$$

where $\alpha_i \geq 0$ and $\alpha_i^* \geq 0$ are Lagrange multipliers.
The main asset of this formulation is that it is a quadratic convex problem which can be solved quite easily and whose solution is known to be global and unique.
Once the coefficients $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ are found, the prediction at any unknown point reads as the following expansion:

$$\mathcal{M}^{\text{SVR}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = \sum_{i=1}^{n}\left(\alpha_i - \alpha_i^*\right)\boldsymbol{x}_i^T\boldsymbol{x} + b. \tag{1.7}$$

The bias term b may be retrieved as a byproduct of the solution of Eq. (1.5) (Bompard, 2011). An important feature of support vector regression is that only a subset of the coefficients $\{\alpha_i, \alpha_i^*\}$ are non-zero. These coefficients are known as *support vectors* (SVs). For the linear penalization case, they can be classified into two groups: *unbounded* support vectors which correspond to the coefficients such that $0 < \alpha_i + \alpha_i^* < C$ and *bounded* SVs whose coefficients correspond to $\alpha_i + \alpha_i^* = C$. The former group corresponds to points which lie on the margin while the latter corresponds to those which lie outside the insensitive tube. The points inside the insensitive tube are not support vectors (*i.e.*, $\alpha_i = \alpha_i^* = 0$) and are therefore not relevant to the actual prediction given by the model. This gives SVR its sparsity property with respect to the training points.

1.3 Extension to non-linear regression

The extension to non-linear regression is performed in support vector machines by mapping the data into a high- or infinite-dimensional space known as the *feature space* through a mapping function Φ. In this space, the linear function becomes:

$$\mathcal{M}^{\text{SVR}}(\boldsymbol{x}) = \boldsymbol{w}^T \Phi(\boldsymbol{x}) + b = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}) + b. \tag{1.8}$$

Henceforth, the inner product is computed in the feature space. In practice, it is not necessary to explicitly carry out this mapping. Instead, the inner product $\Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_j)$ may be defined through a *kernel* $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. In fact, Vapnik (1995) demonstrated that any positive function satisfying so-called *Mercer's conditions* corresponds to an inner product in some feature space. Once a kernel function has been selected, the SVR expansion reads as follows:

$$\mathcal{M}^{\text{SVR}}(\boldsymbol{x}) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) k(\boldsymbol{x}_i, \boldsymbol{x}) + b, \tag{1.9}$$

where the coefficients α_i, α_i^* have been found by maximizing the Lagrangian function:

$$\begin{aligned} L(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = &-\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &- \sum_{i=1}^{N} (\alpha_i + \alpha_i^*) \varepsilon + \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) y_i, \end{aligned} \tag{1.10}$$

subject to the constraints in Eq. (1.6).

1.3.1 Kernel function families

Examples of valid kernel functions are presented in Vapnik (1995). Among the most widely used, the following ones are implemented in UQLAB:

- Non-stationary linear:

$$k_{\text{lin-ns}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\boldsymbol{x}^T\boldsymbol{x}'; \tag{1.11}$$

- Polynomial:

$$k_{\text{poly}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\left(\boldsymbol{x}^T\boldsymbol{x}'+d\right)^p, \tag{1.12}$$

  where $d\geq 0$ and $p\in\mathbb{N}^*$ are the kernel parameters;

- Sigmoid:

$$k_{\text{sigmoid}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\tanh\left(\frac{\boldsymbol{x}^T\boldsymbol{x}'}{a}+b\right), \tag{1.13}$$

  where $a>0$ and $b\leq 0$ are the kernel parameters;

- Stationary linear:

$$k_{\text{lin-s}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\max\left(0,1-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right). \tag{1.14}$$

- Gaussian:

$$k_{\text{Gaussian}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right); \tag{1.15}$$

- Exponential:

$$k_{\text{exp}}\left(\boldsymbol{x},\boldsymbol{x}'\right)=\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right); \tag{1.16}$$

- Matérn $3/2$ and Matérn $5/2$:

$$\begin{aligned} k_{3/2}\left(\boldsymbol{x},\boldsymbol{x}'\right)&=\left(1+\sqrt{3}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right)\exp\left(-\sqrt{3}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right),\\ k_{5/2}\left(\boldsymbol{x},\boldsymbol{x}'\right)&=\left(1+\sqrt{5}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}+\frac{5}{3}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{\sigma^2}\right)\exp\left(-\sqrt{5}\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{\sigma}\right), \end{aligned} \tag{1.17}$$

  where $\sigma>0$ corresponds to the characteristic length-scale.

Note: The most popular kernel for support vector machines is the Gaussian kernel. Once a kernel is chosen, the most important step is to properly fit the hyperparameters.

1.3.2 Anisotropic kernel functions

The expressions of the different kernel functions that were given in Section 1.3.1 correspond to the *isotropic* case. Generally speaking, a kernel function is called *isotropic* when it has the same behavior over all dimensions. In that sense, for each type of kernel function, the same parameter is used in every dimension. Alternatively, one may need to define a different parameter in each dimension. In UQLAB, *anisotropy* is implemented for the stationary kernels, *i.e.* linear, Gaussian, exponential, Matérn $3/2$ and Matérn $5/2$. The following form is used:

$$k\left(\boldsymbol{x},\boldsymbol{x}'\right)=\prod_{i=1}^{M}k_1\left(x_i,{x'}_i;\,\sigma_i\right), \tag{1.18}$$

where $\{\sigma_i, i = 1, \ldots, M\}$ are the kernel parameters in each dimension and $k_1\left(x_i, x'_i, \sigma_i\right)$ are the one-dimensional kernel functions as defined in Eqs. (1.14),(1.15),(1.16) and (1.17).

1.4 Solving the SVR problem

Two versions of SVR are implemented in UQLAB, namely L_1-SVR and L_2-SVR as introduced above. Other formulations derived from different loss functions exist but they are not considered here (See for instance Gunn (1998) for quadratic loss or Saunders et al. (1998) for least-square SVR and the comprehensive review in Bourinet (2018)).

1.4.1 L_1-SVR formulation

The optimization problem in Eq. (1.10) is a quadratic convex problem which may be equivalently written in a matrix form as:

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix}^T \begin{bmatrix} \boldsymbol{K} & -\boldsymbol{K} \\ -\boldsymbol{K} & \boldsymbol{K} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} + \begin{bmatrix} \boldsymbol{\varepsilon} - \boldsymbol{y} \\ \boldsymbol{\varepsilon} + \boldsymbol{y} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} \\ \text{subject to:} \quad & \begin{bmatrix} \mathbf{1} \\ -\mathbf{1} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \qquad 0 \leq \alpha, \alpha^* \leq C, \end{aligned} \tag{1.19}$$

where $\boldsymbol{K}$ is the Gram matrix of size $N \times N$ whose components read $K_{ij} = k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)$.
The solution to this problem may be given by general-purpose quadratic programming solvers. Decomposition methods taking advantage of the sparsity of SVM have been widely used (Platt, 1999; Chang and Lin, 2005). Here we consider MATLAB built-in algorithms such as the interior-point (IP) (Vanderbei, 1994), the *sequential minimal optimization* (SMO) (Platt, 1999) and the *iterative single data algorithm* (ISDA) (Kecman et al., 2005). An important feature of the first approach is that the bias term b may be directly retrieved as a by-product of the results (Bompard, 2011). The latter two are algorithms that have been developed specifically for SVM. For problems with medium to large datasets or problems where sparsity is a desirable property, these two algorithms are expected to be more efficient than IP and should be used. For extremely large datasets, IP may fail due to the large size of the dense kernel matrix that needs to be computed.

1.4.2 L_2-SVR formulation

In L_2-SVR, penalization of deviations larger than ε is quadratic, *i.e.* the optimization problem in Eq. (1.4) becomes:

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{N}\left(\xi_i^2 + \xi_i^{*2}\right), \\ \text{subject to} \quad & y_i - \boldsymbol{w}^T\boldsymbol{x}_i - b \leq \varepsilon + \xi_i, \\ & \boldsymbol{w}^T\boldsymbol{x}_i + b - y_i \leq \varepsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0. \end{aligned} \tag{1.20}$$

It can be shown after some algebra that this problem may be cast as a quadratic optimization problem whose matrix form reads:

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}\begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix}^T \begin{bmatrix} \widetilde{\boldsymbol{K}} & -\widetilde{\boldsymbol{K}} \\ -\widetilde{\boldsymbol{K}} & \widetilde{\boldsymbol{K}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} + \begin{bmatrix} \boldsymbol{\varepsilon} - \boldsymbol{y} \\ \boldsymbol{\varepsilon} + \boldsymbol{y} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} \\ \text{subject to:} \quad & \begin{bmatrix} \mathbf{1} \\ -\mathbf{1} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\alpha}^* \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \qquad \alpha, \alpha^* \geq 0. \end{aligned} \tag{1.21}$$

A modified Gram matrix $\widetilde{\boldsymbol{K}} = \boldsymbol{K} + (1/C)\,\boldsymbol{I}$ has been introduced, where $\boldsymbol{I}$ is the identity matrix of size $N \times N$.

This problem is quite close to L_1-SVR, even though they slightly differ in two points. First, $1/C$ is added at the diagonal of the Gram matrix. For large values of C, it behaves as a ridge making the optimization problem easier to solve. Second, the coefficients of the support vectors are no more upper bounded. Only the interior-point algorithm can used to solve this problem in UQLAB.

1.5 Error estimation

The generalization capacity of the metamodel is usually assessed through resampling techniques. The basic idea is to partition the experimental design into *training* and *validation* sets. A popular approach is *K-fold cross validation*, which consists in partitioning the data into K random sets then using $(K-1)$ sets for training and the remaining one for validation. The procedure is repeated K times by changing the validation set and the resulting error is averaged, thus yielding a measure of the model accuracy. The particular case when $K = N$, also known as *leave-one-out* cross-validation, reads:

$$e_{\text{LOO}} = \frac{1}{N}\sum_{i=1}^{N} \left(\mathcal{M}_{\sim i}^{\text{SVR}}\left(\boldsymbol{x}_i\right) - y_i\right)^2, \tag{1.22}$$

where $\mathcal{M}_{\sim i}^{\text{SVR}}$ denotes the SVR model built by withdrawing the training point $(\boldsymbol{x}_i, y_i)$ from the experimental design. Eq. (1.22) gives a rather good measure of the model accuracy. However its computation requires building as many models as the number of training points. This is a cumbersome task which has been addressed by the development of bounds and approximations of this leave-one-out error, *e.g.* the *radius margin* or *span bounds* (Chang and Lin, 2005). The later is convenient and has been shown to be quite effective in this context (Vapnik and Chapelle, 2000; Chapelle et al., 2002; Chang and Lin, 2005).

1.5.1 Span estimate of the LOO error

The span bound approximation of the LOO error has been proposed by Vapnik and Chapelle (2000) for classification and adapted by Chang and Lin (2005) for regression problems. For

L_1-SVR with the ε-insensitive loss, it reads:

$$\widehat{e}_{\text{LOO}} = \frac{1}{N}\left(\sum_{i\in\mathcal{I}_{\text{sv}}}\left(\alpha_i+\alpha_i^*\right)S_i^2 + \sum_{i\in\mathcal{I}_{\text{sv}}}\left(\xi_i+\xi_i^*\right)\right)+\varepsilon, \tag{1.23}$$

where S_i^2 is the *span* of the support vectors $\boldsymbol{x}_i$ (see below), $\mathcal{I}_{\text{sv}}$ is the set of support vectors of size N_{sv}, ξ_i and ξ_i^* are the slack variables introduced earlier. Hence, the only additional operation required to estimate the LOO error approximation is that of computing S_i^2. The calculation depends on the type of support vector considered (Chapelle et al., 2002), namely:

- if $\boldsymbol{x}_i$ is an unbounded support vector ($0<\alpha_i<C$), then:

$$S_i^2 = \frac{1}{\left(\widetilde{\boldsymbol{K}}_{\text{usv}}^{-1}\right)_{ii}}, \tag{1.24}$$

  where $\widetilde{\boldsymbol{K}}_{\text{usv}} = \begin{bmatrix} \boldsymbol{K}_{\text{usv}} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix}$ and $\boldsymbol{K}_{\text{usv}} = \left[k\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)\right]_{i,j\in\mathcal{I}_{\text{usv}}}$, with $\mathcal{I}_{\text{usv}}$ being the set of unbounded support vectors;

- if $\boldsymbol{x}_i$ is a bounded support vector ($\alpha_i = C$), then:

$$S_i^2 = k(\boldsymbol{x}_i,\boldsymbol{x}_i) - \widetilde{\boldsymbol{v}}_i^T\boldsymbol{K}_{\text{usv}}^{-1}\widetilde{\boldsymbol{v}}_i \tag{1.25}$$

  where $\widetilde{\boldsymbol{v}}_i$ is the i-th column of $\boldsymbol{K}_{\text{usv}}$.

As for L_2-SVR, the span estimate of the leave-one-out error simply reduces to:

$$\widehat{e}_{\text{LOO}} = \frac{1}{N_{\text{sv}}}\left(\sum_{i\in\mathcal{I}_{\text{sv}}}\left(\alpha_i+\alpha_i^*\right)S_i^2\right)+\varepsilon, \tag{1.26}$$

where the span is defined by considering the set of all support vectors (Chapelle et al., 2002):

$$S_i^2 = \frac{1}{\left(\widetilde{\boldsymbol{K}}_{\text{sv}}^{-1}\right)_{ii}}, \tag{1.27}$$

with $\widetilde{\boldsymbol{K}}_{\text{sv}} = \begin{bmatrix} \boldsymbol{K}_{\text{sv}} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix}$ and $\widetilde{\boldsymbol{K}}_{\text{sv}} = \left[k\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)+1/C\delta_{ij}\right]_{i,j\in\mathcal{I}_{\text{sv}}}$. δ_{ij} is the Kronecker symbol, *i.e.* $\delta_{ij}=1$ if $i=j$ and $\delta_{ij}=0$ otherwise.

1.5.2 Smoothed span estimate of the LOO error

For efficiency in large dimensional problems, Chapelle et al. (2002) considers optimizing the LOO error estimate through a gradient descent algorithm. In order to facilitate the gradient computation, they proposed a smooth version of the LOO error estimate based on a modified definition of the span:

$$S_i^2 = \frac{1}{\left(\widetilde{\boldsymbol{K}}_{\text{sv}}+\widetilde{\boldsymbol{D}}\right)^{-1}_{ii}} - \boldsymbol{D}_{ii}, \tag{1.28}$$

where $\widetilde{\boldsymbol{D}} = \begin{bmatrix} \boldsymbol{D} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix}$, with $\boldsymbol{D}$ being a $N_{\text{SV}} \times N_{\text{SV}}$ diagonal matrix whose elements read $\boldsymbol{D}_{ii} = \eta/\alpha_i$ and $\boldsymbol{D}_{ij} = 0$ for $i, j \neq 0$.
The computation of the error is thus carried out using Eqs. (1.23) and (1.26) by replacing the span with the modified version of Eq. (1.28).

1.5.3 *A posteriori* error estimation

After the metamodel is set up, its predictive accuracy on new data can be assessed by using the so-called *validation error*. It is calculated as the relative generalization error on an independent set of inputs and outputs $[\boldsymbol{\mathcal{X}}_{Val},\ \mathcal{Y}_{Val} = \mathcal{M}(\boldsymbol{\mathcal{X}}_{Val})]$:

$$\epsilon_{Val} = \frac{N-1}{N} \left[\frac{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}_{Val}^{(i)}) - \mathcal{M}^{\text{SVR}}(\boldsymbol{x}_{Val}^{(i)}) \right)^2}{\sum_{i=1}^{N} \left(\mathcal{M}(\boldsymbol{x}_{Val}^{(i)}) - \hat{\mu}_{Y_{Val}} \right)^2} \right] \tag{1.29}$$

where $\hat{\mu}_{Y_{Val}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{M}(\boldsymbol{x}_{Val}^{(i)})$ is the sample mean of the validation set response. This error measure is useful to compare the performance of different surrogate models when evaluated on the same validation set. This measure is not used in the optimization algorithms for the hyperparameter calibration.

1.6 Hyperparameters calibration

1.6.1 SVR hyperparameters

Model selection in SVM consists in finding the hyperparameters that lead to the most accurate model in terms of generalization *i.e.*, its ability to predict points that are yet to be observed. For SVR with ε-insensitive loss, these hyperparameters are:

- The penalty term C: as shown in Evgeniou et al. (2000), this parameter may be seen as a regularization term for the regression problem. It trades the complexity of the metamodel with deviations from the training points.

- The insensitive tube width ε: this parameter provides SVM with its sparsity property. Larger values of ε allow for larger deviations from the training points. As it increases, the number of support vectors reduces.

- The parameters of the kernel function: their number depends on the type of kernel function that has been chosen. These parameters are defined in Section 1.3.1. In the sequel, they will be gathered in the vector $\boldsymbol{\theta}$ of size N_θ.

In this manual, the set of hyperparameters is denoted by the vector $\boldsymbol{\gamma} = \{C, \varepsilon, \boldsymbol{\theta}\}^T$ of size $N_\gamma = N_\theta + 2$.

1.6.2 Optimization algorithms

To calibrate the SVR model, the optimal values of the hyperparameters may be found using any general-purpose optimization algorithm to minimize either the estimated leave-one out error (Eqs. (1.23) or (1.26)) or a K-fold cross validation error. In UQLAB, both local and global algorithms are offered. The former, despite being less computationally demanding, may converge to a local minimum. Besides, the error metric used as objective function may be noisy and therefore mislead the algorithm. On the other hand, global search algorithms generally produce more robust results, at a higher cost though.
The following algorithms are currently used for hyperparameters optimization:

- Interior-point BFGS algorithm: to increase robustness, it is possible to specify multiple starting points (MATLAB built-in);
- Grid search ;
- Genetic algorithm (MATLAB built-in);
- Cross-entropy method for optimization (Kroese et al., 2006);
- Covariance matrix adaptation - evolution scheme (CMA-ES) (Hansen and Ostermeier, 2001);
- Hybrid algorithms: Genetic algorithm, cross-entropy or CMA-ES optimization method followed by a gradient-based refinement of the solution.

The BFGS and genetic algorithms used in UQLAB are the ones implemented in MATLAB (respectively in the Optimization Toolbox and the Global Optimization Toolbox). For further details, the reader is referred to MATLAB's documentation. We describe briefly the cross-entropy method for optimization in the sequel.

Note: For practical purpose, the optimization is carried out in the $\log_{10}$ space for the penalty term C and the tube width ε.

1.6.2.1 Cross-entropy method for optimization

The cross-entropy method is a global search algorithm which consists in iteratively sampling points in the search space so as to converge to the optimal solution. The implementation in UQLAB considers a normal distribution for sampling candidates. The algorithm is summarized below (Bourinet, 2015; Moustapha, 2016):

1. Initialize the algorithm as follows:
   - Define the initial PDF parameters, *i.e.* the mean $\boldsymbol{\mu}_\gamma^{(0)} = \{C_0, \varepsilon_0, \boldsymbol{\theta}_0\}^T$ and the standard deviation $\boldsymbol{\sigma}_\gamma^{(0)}$;
   - Set the bounds of the search space $\boldsymbol{\gamma}_{\min}$ and $\boldsymbol{\gamma}_{\max}$ and define the convergence criterion;

- Set internal parameters of the algorithm such as the number of points per generation N_{pop}, the smoothing parameters α^{CE}, β^{CE} and q^{CE} and the number of points in the elite sample $N_{el} = \lfloor \rho N_{\text{pop}} \rfloor$, where $\lfloor \, \rfloor$ denotes the floor function and ρ is a coefficient such that $0 < \rho < 1$;
- Set $t = 1$.

2. Sample N_{pop} points $\left\{\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2, \ldots, \boldsymbol{\gamma}_{N_{\text{pop}}}\right\}$ following a truncated normal distribution $\mathcal{N}_{[\boldsymbol{\gamma}_{\min}, \boldsymbol{\gamma}_{\max}]}\left(\boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(t-1)}, \boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(t-1)}\right)$.

3. Train N_{pop} SVR models with hyperparameters $\boldsymbol{\gamma}_i, i = \left\{1, \ldots, N_{\text{pop}}\right\}$ and evaluate the associated error.

4. Select the N_{el} best models with respect to the computed error metrics. The associated parameters are denoted $\boldsymbol{\gamma}_{(1)}, \ldots, \boldsymbol{\gamma}_{(N_{el})}$.

5. Compute the component-wise mean and standard deviations of the elite sample (the following equations should be understood component-wise for each optimized parameter $\gamma_{,i}$ of $\boldsymbol{\gamma}$):

$$\begin{aligned} \widetilde{\boldsymbol{\mu}}_{\boldsymbol{\gamma}}^{(t)} &= \frac{1}{N_{el}} \sum_{j=1}^{N_{el}} \boldsymbol{\gamma}_{(j)}, \\ \widetilde{\boldsymbol{\sigma}}_{\boldsymbol{\gamma}}^{(t)} &= \sqrt{\frac{1}{N_{el}} \sum_{j=1}^{N_{el}} \left(\boldsymbol{\gamma}_{(j)} - \widetilde{\boldsymbol{\mu}}_{\boldsymbol{\gamma}}^{(t)}\right)^2}. \end{aligned} \tag{1.30}$$

6. Update the parameters of the normal distribution as follows:

$$\begin{aligned} \boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(t)} &= \alpha^{\text{CE}} \widetilde{\boldsymbol{\mu}}_{\boldsymbol{\gamma}}^{(t)} + (1 - \alpha^{\text{CE}}) \boldsymbol{\gamma}_{\text{best}}, \\ \boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(t)} &= \beta_t^{\text{CE}} \widetilde{\boldsymbol{\sigma}}_{\boldsymbol{\gamma}}^{(t)} + (1 - \beta_t^{\text{CE}}) \boldsymbol{\gamma}_{\text{best}}, \end{aligned} \tag{1.31}$$

   where

$$\beta_t^{\text{CE}} = \beta^{\text{CE}} + \beta^{\text{CE}} \left(1 - 1/t\right)^{q^{\text{CE}}} \tag{1.32}$$

   is a dynamic smoothing parameter and $\boldsymbol{\gamma}_{\text{best}}$ is the best set of hyperparameters found so far.

7. Stop if convergence is achieved, otherwise set $t \leftarrow t + 1$ and go to 2. The following convergence criteria are considered:

   - Number of stall generations: The algorithm stops if the number of successive iterations without sampling a point that improves the current best solution reaches a given threshold;
   - Stagnation of the cost: The algorithm stops if the relative change in the cost function values within a given number of iterations is below a given threshold;

- Stagnation of the solution: The algorithm stops if the possible change in the solution becomes extremely small, *i.e.* the current normal distribution can only sample points extremely close to its mean.
- Number of function evaluations: The algorithm stops if the number of calls to the cost function reaches a given threshold.

All the parameters of this algorithm are tunable. It is however recommended to keep the default parameters given in Chapter 3.

1.6.2.2 Covariance Matrix adaptation - Evolution scheme (CMA-ES)

The Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) is a derandomized stochastic search algorithm introduced by Hansen and Ostermeier (2001). It proceeds by adapting the covariance matrix of a normal distribution such that directions that have improved the cost in the recent past iterations are more likely to be sampled again.
In the so-called (μ, λ)-CMA-ES strategy, the candidate solutions of the next generation consist of μ points sampled using a normal distribution considering only the λ best points of the current generation (Hansen and Kern, 2004; Hansen, 2001). Such a strategy is considered in UQLAB, with $\mu = N_{\text{pop}}$ and $\lambda = N_{\text{el}}$. The important steps of the algorithm are presented in the sequel. The actual implementation is more complex, the user can refer to Hansen (2001) for more details.

1. Initialize the algorithm:
   - Set the initial hyperparameters mean $\boldsymbol{\mu}_{\gamma}^{(0)} = \{C_0, \varepsilon_0, \boldsymbol{\theta}_0\}^T$ and corresponding standard deviation $\boldsymbol{\sigma}_{\gamma}^{(0)}$ which will be referred to as the *global step size*;
   - Initialize the covariance matrix $\boldsymbol{C}^{(0)} = \boldsymbol{I}_{N_\gamma \times N_\gamma}$ where N_γ is the number of hyperparameters to optimize;
   - Set the internal CMA-ES parameters, *i.e.* the weight coefficients $\{w_i,\ i = 1, \ldots, \lambda\}$, the so-called *evolution path* $\boldsymbol{p}_c$ and the coefficients c_σ, d_σ, c_c, c_{cov} and μ_{cov};

     Note: All these parameters have been fine-tuned for CMA-ES (Hansen and Ostermeier, 2001). It is not advised to modify their default values.

   - Set $t = 1$.
2. Sample $\mu = N_{\text{pop}}$ points $\left\{\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2, \ldots, \boldsymbol{\gamma}_{N_{\text{pop}}}\right\}$ such that

$$\boldsymbol{\gamma}_i = \boldsymbol{\mu}_{\gamma}^{(t-1)} + \boldsymbol{\varepsilon}^{(t-1)}; \tag{1.33}$$

   where $\boldsymbol{\varepsilon}^{(t-1)} \sim \mathcal{N}\left(\mathbf{0}, \left(\boldsymbol{\sigma}_{\gamma}^{(t-1)}\right)^T \cdot \boldsymbol{C}^{(t-1)} \cdot \boldsymbol{\sigma}_{\gamma}^{(t-1)}\right)$
3. Train N_{pop} SVR models with hyperparameters $\boldsymbol{\gamma}_i, i = \left\{1, \ldots, N_{\text{pop}}\right\}$ and evaluate the associated error;

4. Select the $\lambda = N_{el}$ best models with respect to the computed error metrics. The associated parameters are denoted $\boldsymbol{\gamma}_{(1)}, \ldots, \boldsymbol{\gamma}_{(\lambda)}$;

5. Update the mean of the distribution as follows:

$$\boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(t)} = \sum_{i=1}^{\lambda} w_i \boldsymbol{\gamma}_{(i)}; \tag{1.34}$$

6. Update the global step size

$$\boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(t)} = \boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(t-1)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_c^{(t)}\|}{\sqrt{N_\gamma}\left(1 - \frac{1}{4N_\gamma} + \frac{1}{21N_\gamma^2}\right)} - 1\right)\right), \tag{1.35}$$

where

$$\boldsymbol{p}_c^{(t)} = (1-c_c)\boldsymbol{p}_c^{(t-1)} + \sqrt{c_\sigma\,(2-c_\sigma)}\boldsymbol{B}^{(t-1)}\boldsymbol{D}^{(t-1)}\boldsymbol{B}^{(t-1)}\frac{\sqrt{\mu_{eff}}}{\boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(t-1)}}\left(\boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(t)} - \boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(t-1)}\right). \tag{1.36}$$

$\boldsymbol{B}^{(t-1)}$ and $\boldsymbol{D}^{(t-1)}$ are obtained using a principal component analysis, *i.e.* $\boldsymbol{C}^{(t-1)} = \left(\boldsymbol{B}^{(t-1)}\right)^T \left(\boldsymbol{D}^{(t-1)}\right)^2 \boldsymbol{B}^{(t-1)}$ and $\mu_{eff} = 1/\sum_{i=1}^{\mu} w_i^2$ is the so-called *variance effective selection mass*;

7. Update the covariance matrix

$$\begin{aligned}\boldsymbol{C}^{(t)} = {} & (1 - c_{cov})\,\boldsymbol{C}^{(t-1)} + \frac{c_{cov}}{\mu_{cov}}\boldsymbol{p}_c^{(t)}\left(\boldsymbol{p}_c^{(t)}\right)^T \\ & + c_{cov}\left(1 - \frac{1}{\mu_{cov}}\right)\sum_{i=1}^{\mu}\frac{w_i}{\left(\boldsymbol{\sigma}_{\boldsymbol{\gamma}}^{(t-1)}\right)^2}\left(\boldsymbol{\gamma}_{(i)} - \boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(t-1)}\right)\left(\boldsymbol{\gamma}_{(i)} - \boldsymbol{\mu}_{\boldsymbol{\gamma}}^{(t-1)}\right)^T.\end{aligned} \tag{1.37}$$

8. Stop if convergence is achieved, otherwise set $t \leftarrow t+1$ and go to 2. The following convergence criteria are considered:

   - Number of stall generations: The algorithm stops if the number of successive iterations without sampling a point that improves the current best solution reaches a given threshold;
   - Stagnation of the cost: The algorithm stops if the relative change in the cost function values within a given number of iterations is below a given threshold;
   - Stagnation of the solution: The algorithm stops if the possible change in the solution becomes extremely small, *i.e.* the current normal distribution can only sample points extremely close to its mean.
   - Number of function evaluations: The algorithm stops if the number of calls to the cost function reaches a given threshold.

Chapter 2

Usage

In this section, a reference problem will be set up to showcase how each of the techniques in Chapter 1 can be deployed in UQLAB.

2.1 Reference problem

In the context of this manual, SVR is used for building either a surrogate of some model given a set of observations and model responses or a metamodel given only a set of available data. In this section, the following one-dimensional function is used as a true model:

$$\mathcal{M}(x) = x\,\sin(x)\;,\;\; x \sim \mathcal{U}\,(0, 15) \tag{2.1}$$

2.2 Problem set-up

The SVR module creates a MODEL object that can be used later on exactly as any other MODEL. The basic options common to any SVR metamodel read:

```
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'SVR';
```

As introduced in Eq. (1.9), the SVR expansion reads:

$$\mathcal{M}^{\text{SVR}}\,(\boldsymbol{x}) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*)\, k\,(\boldsymbol{x}_i, \boldsymbol{x}) + b, \tag{2.2}$$

where the coefficients $\alpha_i^{(*)}$ and the offset parameter b are solutions of the quadratic optimization problem in Eq. (1.19) (resp. Eq. (1.21)) for L_1-SVR (resp. L_2-SVR).
To solve these equations, the following minimal ingredients are needed:

- An experimental design $\mathcal{X}$, and the corresponding set of model responses $\mathcal{Y}$;
- An appropriate loss function;
- An appropriate kernel function $k\,(\boldsymbol{x}, \boldsymbol{x}')$;

- Hyperparameters specification, *i.e.* the penalty term C, the insensitive tube width ε and the kernel parameters $\boldsymbol{\theta}$;
- A quadratic optimization problem solver.

Note that the minimal amount of information that need to be supplied by the user is the experimental design. The rest of the ingredients can either be selected and tuned by the user or just left to their default values.

2.3 Fitting the SVR metamodel

Default values are assigned within UQLAB to most of these ingredients. In practice, only the experimental design is required. A minimal working example can be set up in UQLAB with the following commands:

```
% Initialize the UQLab framework
uqlab;
% Create the data X, Y used for the minimal working example
X = transpose(0 : 2 : 14);
Y = X.*sin(X);

% Create the SVR metamodel
MetaOpts.Type = 'Metamodel';
MetaOpts.MetaType = 'SVR';
MetaOpts.ExpDesign.X = X;
MetaOpts.ExpDesign.Y = Y;

mySVR = uq_createModel(MetaOpts);
```

Note: When not specified by the user, the following default values are used:

- Loss function: Linear ϵ-insensitive loss;
- Kernel function: Anisotropic Gaussian;
- Quadratic optimization solver: Interior-point (SMO if the size of the data set N is greater than 300);
- Error metric for hyperparameters calibration: Span estimate of the LOO ;
- Optimization algorithm for hyperparameters calibration: CMA-ES

Once the model is created, a summary of its main features can be printed using the command `uq_print`.

```
uq_print(mySVR);

%------------------------ SVR metamodel ------------------------%
Object Name:      Model 1
Input Dimension:  1

Experimental Design
Sampling:     User
X size:       [8x1]
```

```
Y size:        [8x1]

Loss function:     L_1-epsilon-insensitive
QP Solver:         Quadprog's IP
Kernel:            Gaussian

Hyperparameters
C:                          2.822360e+03
epsilon:                    2.817716e-02
kernel params:              4.907561e-01
Estimation method:          Leave-one-out span estimate
Optim. method:              CMA-ES

Number of SVs (USV,BSV):                  8    (8,0)
Leave-one-out error (normalized):         4.8951173e-01

%-----------------------------------------------------------------%
```

It can be observed that the default values regarding the loss function, the kernel and the optimization method have been assigned.

A visual representation of the metamodel can be obtained by:

```
uq_display(mySVR);
```

Note that `uq_display` for SVR `MODEL` objects can only be used for one- and two-dimensional inputs. The plot produced by `uq_display` is shown in Figure 3.

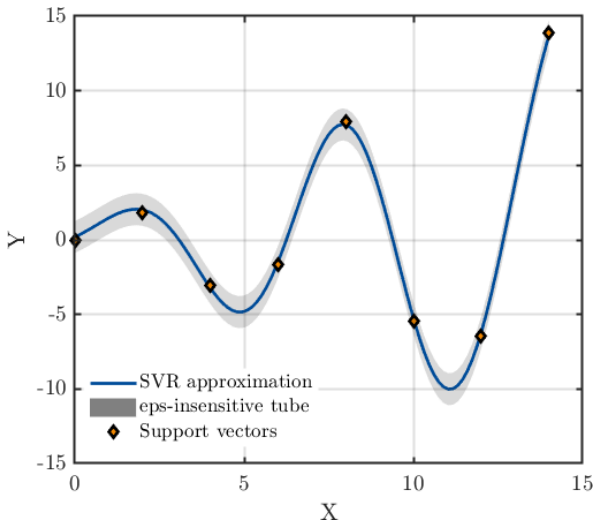

Figure 3: Illustration of the SVR model as displayed by the `uq_display` command: The default parameters are considered, *i.e.* the hyperparameters are calibrated by minimizing the span estimate of the LOO error with the CMA-ES algorithm.

In this figure, the support vectors are plotted as green diamonds while the remaining training points are black dots (In this example, all the training points are support vectors). In one-dimensional cases, the ε-insensitive tube is also illustrated and depicted as a gray-shaded area.

2.3.1 Accessing the results

The SVR model object is created using the options given by the user. When options are not provided, default parameters are set such that all the necessary information required to build the model is defined. The resulting parameters are structured in the SVR object, herein `mySVR`, and can be accessed by the user.

SVR parameters

The basic results of the of the SVR metamodel, namely the hyperparameters values (Section Section 1.6.1) and the coefficients of the expansion in Eq. 1.9 are given in the field `mySVR.SVR`

```
mySVR.SVR

ans =

struct with fields:

  Hyperparameters: [1x1 struct]
     Coefficients: [1x1 struct]
           Kernel: [1x1 struct]
```

The field `.Hyperparameters` contains the values of the penalty term C, the insensitive tube width ε and the kernel parameters $\boldsymbol{\theta}$:

```
mySVR.SVR.Hyperparameters

ans =

  struct with fields:

           C: 2.8224e+03
     epsilon: 0.0282
       theta: 0.4908
```

The field `.Coefficients` contains the following information:

```
mySVR.SVR.Coefficients

ans =

  struct with fields:

     alpha: [16x1 double]
      beta: [8x1 double]
      bias: 0.3268
     SVidx: [8x1 double]
    USVidx: [8x1 double]
    BSVidx: [8x1 double]
```

`C` is the penalty term, `alpha` is the vector containing the coefficients α_i and α_i^* of the model, `beta` is a vector whose elements correspond to the term $\alpha_i - \alpha_i^*$ of Eq. (1.9), and `SVidx` are the indices of the all SVs while `USVidx` and `BSVidx` represent respectively the indices of

the unbounded and bounded SVs. The latter two are available only when using the linear $\varepsilon-$insensitive loss function. Furthermore, `bias`, `eloo` and `eloo_norm` respectively stand for the bias, the leave-one-out and the normalized leave-one-out error estimates.
Finally, `Kernel` is a structure containing the kernel handle, the kernel family and when applicable, the isotropy/anisotropy boolean. These values can be accessed as follows:

```
mySVR.SVR.Kernel

      Handle:  @uq_eval_Kernel
      Family: 'Gaussian'
   Isotropic: 0
```

Model evaluations

The experimental design can be accessed in the substructure `mySVR.ExpDesign`. The following information is displayed:

```
mySVR.ExpDesign

ans =

           X: [8x1 double]
           Y: [8x1 double]
    Sampling: 'User'
    NSamples: 8
           U: [8x1 double]
          Yu: [8x1 double]
```

The type and size of the experimental design are given respectively in the fields `ExpDesign.Sampling` and `ExpDesign.NSamples`. In the case when an INPUT object is used to create the experimental design, the related information is displayed in the field `ExpDesign.Input`. The remaining fields are:

- `ExpDesign.X`: The experimental design $\mathcal{X}$.
- `ExpDesign.Y`: The model responses corresponding to the inputs.
- `ExpDesign.U`: The scaled experimental design. Refer to Section Section 2.9 for more information.
- `ExpDesign.Yu`: The scaled experimental design model responses. Refer to Section Section 2.9.2 for more information.

A posteriori error estimates

The leave-one-out (LOO) error of the SVR model as estimated using Eq. (1.23) is stored in `mySVR.Error`:

```
mySVR.Error

  struct with fields:
```

```
        LOO: 0.4895
   LOO_norm: 0.4895
```

`LOO_norm` is the normalized LOO error, *i.e.* $e_{\text{LOO\_norm}} = e_{\text{LOO}}/\text{Var}\,(\mathcal{Y})$. When `.OutputScaling` is set to `true`, then $e_{\text{LOO\_norm}} = e_{\text{LOO}}$ as the variance of the scaled output is 1.

2.4 Evaluating the model

To evaluate the model for a given set of points, the function `uq_evalModel` shall be used. In the following code, the built SVR model is used to evaluate a set of validation data points `Xval`.

```
Yval = uq_evalModel(mySVR,Xval);
```

2.5 Set-up of the SVR metamodel

In the following subsections, the various configuration options of each of the ingredients of a SVR metamodel are presented.

2.5.1 Generation/Specification of the experimental design

2.5.1.1 Using already existing data

Depending on where the experimental design data is stored the following alternatives are offered:

- If data is stored in the MATLAB workspace, *e.g.* in the variables `X`, `Y`:

  ```
  Metaopts.ExpDesign.X = X;
  Metaopts.ExpDesign.Y = Y;
  ```

- If data is stored in a .mat file, *e.g.* `mydata.mat`:

  ```
  Metaopts.ExpDesign.DataFile = 'mydata.mat' ;
  ```

  Note: The experimental design that is contained in `mydata.mat` must be saved with variable names `X` and `Y` respectively.

The input and output dimensions M and N_{out} of the problem are automatically retrieved by the number of columns of `X` and `Y`, respectively.

2.5.1.2 Using INPUT and MODEL objects

First we have to create the probabilistic INPUT vector. For the reference problem in Eq. (2.1) this reads:

```
IOpts.Marginals.Type = 'Uniform';
IOpts.Marginals.Parameters = [0 15];
myInput = uq_createInput(IOpts);
```

The input and output dimensions of the problem are automatically retrieved by the configuration of the INPUT object `myInput` and MODEL object `myModel`. For more information about the configuration options available of an INPUT and a MODEL object, please refer to the UQLAB User Manual – the INPUT module and the UQLAB User Manual – the MODEL module respectively.

Then we create a MODEL object as follows:

```
MOpts.mString = 'X.*sin(X)' ;
myModel = uq_createModel(MOpts);
```

Now we include the INPUT and MODEL objects to the SVR metamodel configuration:

```
Metaopts.Input = myInput;
Metaopts.FullModel = myModel;
```

Finally, we need to define the size of the experimental design, *e.g.* for $N = 8$ samples:

```
Metaopts.ExpDesign.NSamples = 8;
```

Note that, by default crude Monte Carlo sampling is used in order to obtain the sample points $\mathcal{X}$. However other sampling strategies can be selected through the option `Metaopts.ExpDesign.Sampling`, see Table 3 in Section 3.1.1 for a list of the available sampling strategies.

2.5.2 Loss function

Two types of loss functions are available in UQLAB, namely the linear ε-insensitive and the quadratic ε-insensitive loss functions. For instance, one can set the quadratic loss function with the following command:

```
MetaOpts.Loss = 'l2-eps' ;
```

Additional information is given in Table 2.

2.5.3 Kernel functions

2.5.3.1 Standard options

The main ingredients for specifying the kernel function are:

- **Family:** The most widely used families of kernels are implemented in UQLAB. An extensive list of the available kernel functions families can be found in Table 4. For instance, in order to use the Matérn-$5/2$ family, the following option can be set:

```
MetaOpts.Kernel.Family = 'Matern-5_2' ;
```

- **Isotropic:** This is a Boolean flag (with `true` or `false` value) that specifies whether the kernel function is isotropic or not. By default, the kernel function is considered isotropic, unless the following option is set:

```
MetaOpts.Kernel.Isotropic = false ;
```

Note that this option is only available for the radial-basis families of kernels, namely, Gaussian, exponential, Matérn $3/2$ and Matérn $5/2$.

2.5.3.2 Advanced options

- **User-defined kernel family**

Apart from the built-in kernel families described above, user-defined *one-dimensional* kernels can be used. They are expected to be of the form $k_1\left(x, x'; \theta\right)$ where x, x' are two input vectors and $\boldsymbol{\theta}$ is a vector of parameters. For example, to define the following one-dimensional kernel family

$$K_1(x, x'; \theta) = \exp\left[-\left(\frac{x - x'}{\theta}\right)^{3/2}\right],$$

the following option should be provided:

```
MetaOpts.Kernel.Family = @(x1,x2,th) exp(-((x1-x2)/th).^(3/2));
```

Note: The user-defined kernel family is expected to be *vectorized*, *i.e.* it should be able to handle inputs `x1` and `x2` that are vectors with the same lengths.

- **User-specified routine to calculate the kernel matrix:** All the aforementioned options regarding the kernel function are handled by the function `uq_SVR_eval_K`. If the user wants to use a fully custom kernel function in order to obtain the kernel matrix $\boldsymbol{K}$ the following rules need to be followed:

  - The function should return the kernel matrix $\boldsymbol{K}$ and accept x_1, x_2, θ and a structure of options like the following:

```
function K = my_eval_K(x1,x2,theta,options)
...
```

  - Inputs `x1`, `x2` are matrices with arbitrary number of rows and M number of columns (where M is the input dimension).
  - When the custom kernel function `my_eval_K` is called, *e.g.* during the creation or usage of a SVR metamodel, the structure `options` will contain all options that were set inside `MetaOpts.Kernel`.
  - The parameter `theta` of the function `my_eval_K` corresponds to the hyperparameters $\boldsymbol{\theta}$ and it is expected to be a vector of arbitrary length. There is no limitation

regarding its length but its dimension should be reflected in the choice of the initial value and/or optimization bounds (see Section 2.5.7.2 for more information about the optimization options).

This user-defined function can be used for calculating the SVR metamodel as long as the following option has been set:

```
MetaOpts.Kernel.Handle = @my_eval_K;
```

An example with custom kernel using a mixed kernel function made of a linear combination of the Gaussian and polynomial families can be found in the example script `uq_Example_SVR_03_MixedKernel`.

2.5.4 Hyperparameters estimation methods

2.5.5 Standard options

The hyperparameters estimation method affects the type of the objective function that is minimized to calibrate the hyperparameters. Currently, the leave-one-out and cross-validation methods are available for estimating the SVR parameters. They can be selected by setting `'SpanLOO'`, `'SmoothLOO'` or `'CV'` as the value of the field `MetaOpts.EstimMethod`.

Note: If not specified otherwise by the user, the span bound leave-one-out error (`'SpanLOO'`) is used by default.

2.5.5.1 Advanced Options

Smooth leave-one-out error

When the smooth leave-one-out error method is considered, it is used with its default parameters. One can define a specific value for the regularization term η of Eq. (1.28) as follows:

```
MetaOpts.SmoothLOO.eta = eta_value;
```

where $\eta =$ `eta_value` is a positive real.

Cross-validation

The cross-validation option implements K-fold cross validation where the number of folds is by default $K = 3$. It can be set to leave-one-out or to any other arbitrary value of $K \geq 2$ with the following command:

```
MetaOpts.CV.NumOfFolds = K;
```

where K is an integer that should be smaller than the size of the experimental design. The true leave-one-out error (in contrast to the span approximate) corresponds to $K = N$ where N is the experimental design size.

2.5.6 SVR quadratic problem solver

The coefficients of the SVR expansion are found by solving the quadratic optimization problem in Eq.(1.19) or (1.21). To set one algorithm, say SMO, on can use the following syntax:

```
MetaOpts.QPSolver = 'SMO';
```

Note: SMO and ISDA cannot be used for L_2-SVR. If they are selected, a warning will be issued and the program will proceed using interior-point convex algorithm.

Note: For medium to large datasets ($N > 300$), the interior-point convex algorithm may be extremely slow or even fail to converge since it does not account for any sparsity and builds the Gram matrix at once using all the training points. It is advised to use SMO for medium to large dataset problems. This is set by default if the user does not specify the solver.

2.5.7 Hyperparameters calibration

2.5.7.1 Default values

Default values are given to the hyperparameters in UQLAB following rules of thumb or heuristics. They are defined as follows:

- **Penalty term C:** as proposed in Cherkassky and Ma (2004), a reasonable value of C is the range of the model response over the experimental design, *i.e.*:

$$C_0 = \max \mathcal{Y} - \min \mathcal{Y}. \tag{2.3}$$

  For noisy models, they proposed a formulation which allows one to minimize the importance of any possible outlier:

$$C_0 = \max \left(\left| \mu_{\boldsymbol{\mathcal{Y}}} + 3\sigma_{\boldsymbol{\mathcal{Y}}} \right|, \left| \mu_{\boldsymbol{\mathcal{Y}}} - 3\sigma_{\boldsymbol{\mathcal{Y}}} \right| \right), \tag{2.4}$$

  where $\mu_{\boldsymbol{\mathcal{Y}}}$ and $\sigma_{\boldsymbol{\mathcal{Y}}}$ are respectively the mean and standard deviations of the experimental design's response values. Eq. (2.4) is used to set the default value of C in UQLAB.

- **The insensitive tube width ε:** the default value is computed based on the interquartile range of the dataset:

$$\varepsilon_0 = \text{iqr}\left(\boldsymbol{\mathcal{Y}}\right)/13.49, \tag{2.5}$$

  where iqr is the interquartile operator. Note that it is the same as in the MATLAB's built-in implementation of support vector machines.

- **Kernel parameters $\boldsymbol{\theta}$:** The computation of the default kernel parameters depends on the chosen family:

  - Radial basis families of kernel: The characteristic length-scale σ for radial basis kernel functions. The default value depends on the input. It is computed as the

average distance between the training points and reads:

$$\sigma_0 = \frac{2}{N\,(N-1)} \sum_{j,k,j>k}^{N} d_{jk}, \tag{2.6}$$

where

$$d_{jk} = \|\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)}\|, \quad j = 1,, \ldots ,, N, \quad k = 1,, \ldots ,, N, \tag{2.7}$$

> **Note:** When the size of the training set is larger than 500, the average distance is computed on a random subset of the training set containing only 500 points

In case of anisotropy, the same parameter is computed independently for each dimension.

– For the other kernels, the default values are independent of the experimental design and are given values specified in Table 5.

These values are used to build a model when the user does not specify any value *and* does not carry out optimization of the hyperparameters (which can be done by setting the option `.Optim.Method` to `'none'` as will be shown in the sequel). Figure 4 shows the resulting model for the example in Section 2.3.

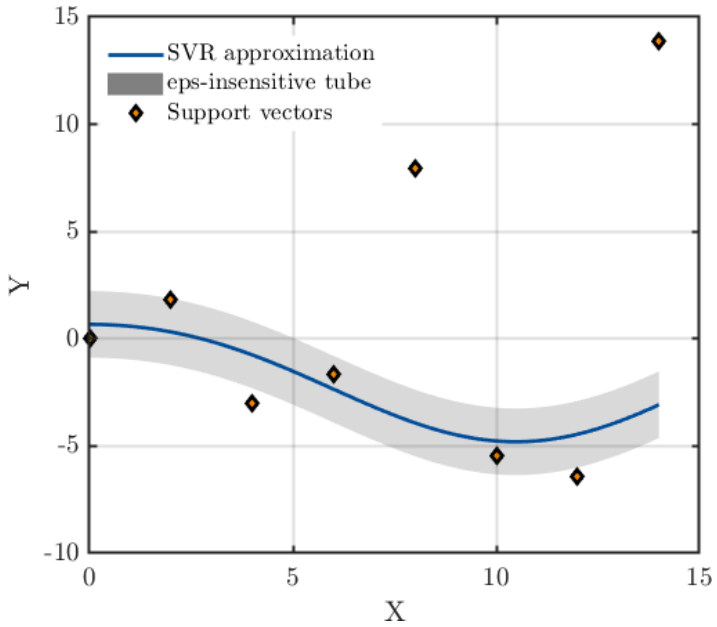

Figure 4: Illustration of the SVR model as displayed by the `uq_display` command: The UQLAB default parameters are considered here without any calibration, which leads to an extremely poor model.

> **Note:** As can be seen by comparing Figure 4 with Figure 3, a SVR metamodel obtained without optimizing the hyperparameters is usually of mediocre quality.

Instead of relying on the default values, one may also set values of the hyperparameters directly as follows:

```
MetaOpts.Hyperparameters.C = C_user;
MetaOpts.Hyperparameters.epsilon = eps_user;
MetaOpts.Hyperparameters.theta = theta_user;
```

For anisotropic kernels, the user may specify a scalar which will then be replicated in all dimensions or a vector of size $1 \times M$ to define `theta_user`.
When using a polynomial kernel, the polynomial order can be given using the following command:

```
MetaOpts.Hyperparameters.degree = 2;
```

If the value set is a vector, UQLAB will calibrate the model using each of the specified polynomial orders and keep as final model the one with the lowest error. For instance, to calibrate a model with possible polynomial orders going from 2 to 5, the following command can be used:

```
MetaOpts.Hyperparameters.degree = 2:5;
```

2.5.7.2 Optimization algorithms

Various configuration options are available regarding the method to solve the optimization problem used to calibrate the hyperparameters. The available optimization methods can be divided into three categories:

- **Gradient-based methods**

  Currently the BFGS method is available. In order to use this method the following syntax is used:

  ```
  MetaOpts.Optim.Method = 'BFGS';
  ```

  Additional options specific to BFGS can be set. For example, one can set the maximum number of iterations as follows:

  ```
  MetaOpts.Optim.MaxIter = 10;
  ```

- **Global methods**

  Currently the Genetic Algorithm (GA), the cross-entropy method for optimization (CE), the covariance matrix adaptation - evolution scheme (CMA-ES) and their hybrid counterparts (respectively HGA, HCE and HCMAES) are available in UQLAB. To use one of these methods for solving the optimization of the hyperparameters, *e.g.* the Genetic Algorithm, one sets:

  ```
  MetaOpts.Optim.Method = 'GA';
  ```

Additional options which are specific to each method exist. For example, when using the Genetic Algorithm method one might need to set a different value of stall generations, *e.g.* 20. This can be accomplished by setting:

```
MetaOpts.Optim.GA.nStall = 20;
```

- **Grid search method**

  In this approach, a regular grid is created on the search space ($\log_{10}$-space for C and ε). An SVR model is then built for each combination of hyperparameters and the one with the lowest error (according to the chosen estimation method) is selected as final model.

  To define the number of discretization points for each variable, one can use the following command:

  ```
  MetaOpts.Optim.GS.NumOfDiscPoints = 5;
  ```

  When a scalar is given, the same value is used in all directions. In contrast, the user can also provide a row vector, the size of which should be equal to the number of hyperparameters to optimize, in order to specify different values in each direction.

- **General options**

  Moreover, regardless of the method, the user can manually specify the following options:

  - The initial search point (not necessary for GA):

    ```
    MetaOpts.Optim.InitialValue.C = C_0 ;
    MetaOpts.Optim.InitialValue.epsilon = eps_0 ;
    MetaOpts.Optim.InitialValue.theta = theta_0 ;
    ```

  - The number of iterations or generations (depending on the optimization method):

    ```
    MetaOpts.Optim.MaxIter = 100;
    ```

  - The convergence tolerance:

    ```
    MetaOpts.Optim.Tol = 1e-5;
    ```

  - The verbosity of the optimization process, *e.g.* showing only the final result:

    ```
    MetaOpts.Optim.Display = 'final';
    ```

For a list of all the available options for each method, please refer to Table 8 in Section 3.1.5. A comparison of the resulting SVR models using different optimization (and estimation) methods can be found in the example script `uq_Example_SVR_02_VariousMethods`.

2.5.7.3 Default bounds of the search space

Some default values are independent on the experimental design and are given numeric values specified in Table 8. The others are defined by the following empirical rules:

- Lower and upper C

$$C_{\min} = C_0/10 \quad \text{and} \quad C_{\max} = C_0 \times 10^5, \tag{2.8}$$

  where C_0 is the user-given or default value of the penalty coefficient as computed in Eq. (2.4);

- Lower and upper bounds on ε:

$$\varepsilon_{\min} = \varepsilon_0/1000 \quad \text{and} \quad \varepsilon_{\max} = \varepsilon_0 \times 10, \tag{2.9}$$

  where ε_0 is user-given or default value of the insensitive tube width as computed in Eq. (2.5);

- Lower bounds on the kernel parameters for the radial basis family of kernels:

$$\sigma_{\min} = \min_{j,k} d_{jk}/M, \tag{2.10}$$

  where d_{jk} is given by Eq. (2.7) and M is the number of hyperparameters;

- Upper bounds on the kernel parameters for the radial basis families of kernel:

$$\sigma_{\max} = M \times \max_{j,k} d_{jk}, \tag{2.11}$$

  where d_{jk} is given by Eq. (2.7);

Note: For large data sets, the default settings of the UQLAB SVR module are modified in order to increase the calibration speed and favor sparsity. Hence when the experimental design is of size larger than 300 the following adaptations to the default values are made:

- the default QP solver becomes SMO;
- the upper bound of C becomes $100 \times C_0$ (See Eq. (2.8));
- the default lower bound of ε becomes $\epsilon_0/100$ (See Eq. (2.9)).

2.5.7.4 Specifying a subset of hyperparameters to calibrate

It is possible within UQLAB to calibrate a subset of hyperparameters while keeping the remaining ones to their user-given or default values. By default, all parameters are calibrated. To set off the calibration of one particular parameter, say the penalty term C, the following command should be used:

```
MetaOpts.Optim.Calibrate.C = false ;
```

2.6 Use of a validation set

If a validation set is provided (see Table 21 in Section 3.1.6), UQLAB automatically computes the validation error given in Eq. (1.29). To provide a validation set, the following command shall be used:

```
MetaOpts.ValidationSet.X = XVal;
MetaOpts.ValidationSet.Y = YVal;
```

The value of the validation error is stored in `mySVR.Error.Val` (see Table 26) and will also be displayed when typing `uq_print(mySVR)`.

2.7 SVR metamodels of vector-valued models

All the examples presented so far in this chapter dealt with scalar-valued models. In case the model (or the experimental design, if manually specified) produces multi-component outputs, UQLAB performs an independent SVR metamodel for each output component on the shared experimental design. No additional configuration is needed to enable this behaviour. A SVR example with multi-component outputs can be found in the UQLAB example script `uq_Example_SVR_04_MultipleOutputs`.

2.7.1 Accessing the results

Building a SVR surrogate on a multi-component output model will result in a multi-component output structure. As an example, a model with 9 outputs will produce the following output structure:

```
mySVR.SVR
ans =
1x9 struct array with fields:

  Hyperparameters: [1x1 struct]
     Coefficients: [1x1 struct]
           Kernel: [1x1 struct]
```

Each element of the `SVR` structure is functionally identical to its scalar counterpart in Section 2.3.1. Similarly, the `mySVR.Error` structure becomes a multi-element structure:

```
mySVR.Error
ans =
1x9 struct array with fields:
  LOO
  LOO_norm
```

2.8 Using SVR with constant parameters

In some analyses, one may need to assign a constant value to one or more parameters. When this is the case, the SVR metamodel is designed so as to internally remove the constant parameters from the inputs. This process is transparent to the users as they shall still evaluate the model using the full set of parameters (including those which were set constant). UQLAB will automatically and appropriately account for the set of input parameters which were declared constant.
To set a parameter to constant, the following command can be used (See UQLAB User Manual – the INPUT module):

```
inputOpts.Marginals.Type = 'Constant' ;
inputOpts.Marginals.Parameters = value;
```

Furthermore, when the standard deviation of a parameter is set to zero, UQLAB automatically sets this parameter's marginal to the type `Constant`. For example, the following uniformly distributed variable whose upper and lower bounds are identical is automatically set to a constant with value 1:

```
inputOpts.Marginals.Type = 'Uniform' ;
inputOpts.Marginals.Parameters = [1 1];
```

Note: A metamodel built with a given parameter set to constant shall not be evaluated using a different value of that parameter. It is however possible to build multiple metamodels by setting different constant values for some parameters.

2.9 Performing SVR on an Auxiliary space (Scaling)

The SVR module offers various options for scaling the experimental design before computing the metamodel.

2.9.1 Input scaling

Note: The SVR module uses by default the *scaled* experimental design $\mathcal{U}$ instead of the original values in $\mathcal{X}$.

Depending on the option `Metaopts.Scaling` and whether a probabilistic input model has been defined (in `Metaopts.Input`), the transformation $\mathcal{X} \mapsto \mathcal{U}$ varies. All the possible cases are summarised in Table 1.

Table 1: Available experimental design transformations		
`Input` / `Scaling`	No INPUT defined	INPUT defined
`1`	$\mathcal{U} = (\mathcal{X} - \mu_{\mathcal{X}})/\sigma_{\mathcal{X}}$	$\mathcal{U} = (\mathcal{X} - \mu_{\boldsymbol{X}})/\sigma_{\boldsymbol{X}}$
`0`	$\mathcal{U} = \mathcal{X}$	$\mathcal{U} = \mathcal{X}$
INPUT object	-	$\mathcal{U} = \mathcal{T}(\mathcal{X})$

By setting `Metaopts.Scaling = 1`, the experimental design is scaled so that it has zero mean and unit variance. When no input distribution has been specified, $(\mu_{\mathcal{X}}, \sigma_{\mathcal{X}})$ are empirically estimated from the available data specified in `Metaopts.ExpDesign.X`. When an input distribution has been specified via an INPUT object, say `myInput`, as follows:

```
Metaopts.Input = myInput;
```

then $(\mu_{\boldsymbol{X}}, \sigma_{\boldsymbol{X}})$ are estimated from the moments of the marginal distribution of each component in $\boldsymbol{X}$.
When an additional INPUT object, say `myScaledInput`, is set as follows:

```
Metaopts.Scaling = myScaledInput;
```

then $\mathcal{U} = \mathcal{T}(\mathcal{X})$ where $\mathcal{T}$ denotes the generalised *isoprobabilistic transformation* between the two probability spaces (for more information refer to the UQLAB User Manual – the INPUT module).

2.9.2 Output scaling

Instead of using $\mathcal{Y}$, the module uses a *scaled* output $\mathcal{Y}_U$ which is obtained by the following linear transformation:

$$\mathcal{Y}_U = (\mathcal{Y} - \mu_{\mathcal{Y}})/\sigma_{\mathcal{Y}}, \tag{2.12}$$

where $\mu_{\mathcal{Y}}$ and $\sigma_{\mathcal{Y}}$ are respectively the mean and standard deviation of the outputs in the physical space, empirically estimated from the available data.

Note: By default `Metaopts.OutputScaling = 1`.
The user can disable output scaling by setting this option to `0`.

Chapter 3

Reference List

How to read the reference list

Structures play an important role throughout the UQLAB syntax. They offer a natural way to semantically group configuration options and output quantities. Due to the complexity of the algorithms implemented, it is not uncommon to employ nested structures to fine-tune the inputs and outputs. Throughout this reference guide, a table-based description of the configuration structures is adopted.
The simplest case is given when a field of the structure is a simple value or array of values:

Table X: `Input`			
●	`.Name`	String	A description of the field is put here

which corresponds to the following syntax:

```
Input.Name = 'My Input';
```

The columns, from left to right, correspond to the name, the data type and a brief description of each field. At the beginning of each row a symbol is given to inform as to whether the corresponding field is mandatory, optional, mutually exclusive, etc. The comprehensive list of symbols is given in the following table:

●	Mandatory
□	Optional
⊕	Mandatory, mutually exclusive (only one of the fields can be set)
⊞	Optional, mutually exclusive (one of them can be set, if at least one of the group is set, otherwise none is necessary)

When one of the fields of a structure is a nested structure, a link to a table that describes the available options is provided, as in the case of the `Options` field in the following example:

Table X: `Input`

●	.Name	String	Description
☐	.Options	Table Y	Description of the Options structure

Table Y: Input.Options			
●	.Field1	String	Description of Field1
☐	.Field2	Double	Description of Field2

In some cases, an option value gives the possibility to define further options related to that value. The general syntax would be:

```
Input.Option1 = 'VALUE1' ;
Input.VALUE1.Val1Opt1 = ...;
Input.VALUE1.Val1Opt2 = ...;
```

This is illustrated as follows:

Table X: Input			
●	.Option1	String	Short description
		'VALUE1'	Description of 'VALUE1'
		'VALUE2'	Description of 'VALUE2'
⊞	.VALUE1	Table Y	Options for 'VALUE1'
⊞	.VALUE2	Table Z	Options for 'VALUE2'

Table Y: Input.VALUE1			
☐	.Val1Opt1	String	Description
☐	.Val1Opt2	Double	Description

Table Z: Input.VALUE2			
☐	.Val2Opt1	String	Description
☐	.Val2Opt2	Double	Description

Note: In the sequel, `double` and `doubles` mean a real number represented in double precision and a set of such real numbers, respectively.

3.1 Create the SVR metamodel

Syntax

```
mySVR = uq_createModel(Metaopts)
```

Input

The struct variable `Metaopts` contains the configuration information for a support vector regression metamodel. The detailed list of available options is reported in Table 2.

Table 2: `Metaopts`

●	`.Type`	`'Metamodel'`	Select the metamodeling tool
●	`.MetaType`	`'SVR'`	Select support vector regression
☐	`.Name`	String	Unique identifier for the metamodel
☐	`.Display`	String default: `'standard'`	Level of information displayed by the methods
		`'quiet'`	Minimum display level, displays nothing or very few information.
		`'standard'`	Default display level, shows the most important information.
		`'verbose'`	Maximum display level, shows all the information on runtime, like updates on iterations, etc.
⊞	`.ExpDesign`	Table 3	Experimental design options
⊞	`.Input`	String	The name of the INPUT object that describes the inputs of the metamodel
		INPUT object	The INPUT object that describes the inputs of the metamodel
☐	`.FullModel`	String	The name of the MODEL object that is used to calculate the model responses
		MODEL object	The MODEL object that is used to calculate the model responses
☐	`.Loss`	String default: `'l1-eps'`	Loss function (Section 1.2)
		`'l1-eps'`	Linear ε-insensitive
		`'l2-eps'`	Quadratic ε-insensitive
☐	`.Kernel`	Table 4	Kernel function used to compute the Gram matrix (Section 1.3.1)

☐	.Hyperparameters	Table 5	Values of the hyperparameters (Section 1.6.1)
☐	.QPSolver	String default: 'IP'	Quadratic programming algorithm used to estimate the coefficients of the expansion (Eqs. (1.19) and (1.21))
		'IP'	Interior-point method
		'SMO'	Sequential minimal optimization using MATLAB's fitrsvm (becomes the default when the experimental design size $N > 300$) • fitrsvm is only available in MATLAB versions older than R2015a.
		'ISDA'	Iterative single data algorithm using MATLAB's fitrsvm • fitrsvm is only available in MATLAB versions older than R2015a.
☐	.Scaling	Logical / Integer default: true	An integer defining whether the input data should be scaled or not, see details in Table Table 1
		true / 1	Scale the input
		false / 0	Do not scale the input
☐	.OutputScaling	Logical / Integer default: true	An integer defining whether the output data should be scaled or not
		true / 1	Scale the output
		false / 0	Do not scale the output
☐	.EstimMethod	String default: 'SpanLOO'	Hyperparameters estimation method (Section 1.6.1)
		'SpanLOO'	Minimization of the span estimate of the leave-one-out error (Eq. (1.23;1.26;1.24))
		'SmoothLOO'	Minimization of a smoothed span estimate of the leave-one-out error (Eq. (1.23;1.26;1.28))
		'CV'	Minimization of a K-fold cross validation generalization error
☐	.SmoothLOO	Table 6	Smooth span LOO related options
☐	.CV	Table 7	Cross-validation related options
☐	.Optim	Table 8	Hyperparameters optimization-related options (Section 1.6.2)
☐	.ValidationSet	Table 21	Validation set components (Section 2.6)

3.1.1 Experimental design options

Table 3: `Metaopts.ExpDesign`			
☐	`.Sampling`	String default: `'MC'`	Sampling type
		`'MC'`	Monte Carlo sampling
		`'LHS'`	Latin Hypercube sampling
		`'Sobol'`	Sobol sequence sampling
		`'Halton'`	Halton sequence sampling
⊞	`.DataFile`	String	A string containing the name of the mat file that contains the experimental design.
⊞	`.X`	$N \times M$ Double	User defined model input X.
⊞	`.Y`	$N \times O$ Double	User defined model response Y.
☐	`.NSamples`	Integer	The number of samples to draw

3.1.2 Kernel function options

Table 4: `Metaopts.Kernel`			
☐	`.Family`	String or function handle default: `'Gaussian'`	The kernel family, *i.e.* the elementary 1-D function that is used by the kernel function. For a description of each family, refer to Section 1.3.1
		`'Linear-NS'`	Non-stationary Linear kernel function
		`'Polynomial'`	Polynomial kernel function
		`'Sigmoid'`	Sigmoid kernel function
		`'Linear'`	Stationary linear kernel function
		`'Exponential'`	Exponential kernel function.
		`'Gaussian'`	Gaussian kernel function
		`'Matern-3_2'`	Matérn-3/2 kernel function
		`'Matern-5_2'`	Matérn-5/2 kernel function
		function handle	The user defined kernel family function handle
☐	`.Isotropic`	Logical default: true	Determines whether the kernel function is isotropic or anisotropic (only valid for stationary kernels)
☐	`.Handle`	function handle default: `@uq_SVR_eval_K`	The handle of the function which is used to calculate the kernel matrix $\boldsymbol{K}$ (Section 2.5.3.2)

3.1.3 Hyperparameters specification

Table 5: Metaopts.Hyperparameters			
☐	.C	Double	Penalty term. The default value depends on the experimental design (See Eq. (2.4))
☐	.epsilon	Double	Insensitive tube width. The default value depends on the experimental design (See Eq. (2.5))
☐	.theta	Variable size Double	Kernel function parameter. • For stationary kernels, the default value depends on the experimental design (See Section 2.5.7.1) • For the non-stationary linear kernel, there is no hyperparameter (Eq. (1.11)) • For the polynomial kernel the defaults are $d = 1$ and $p = 2$ (Eq. (1.12)) • For the sigmoid kernel, the defaults are $a = 1$ and $b = -1$ (Eq. (1.13))

3.1.4 Estimation method options

Table 6: Metaopts.SmoothLOO			
☐	.eta	Positive Double default: 0.1	Regularization term η for the computation of the span following Eq. (1.28)

Table 7: Metaopts.CV			
☐	.Folds	Positive Double default: 3	• Number of folds for the cross-validation procedure. It can be any integer between 2 and N • The case with the value N corresponds to the well-known leave-one-out error

3.1.5 Hyperparameters optimization options

Table 8: `Metaopts.Optim`			
☐	`.Method`	String default: `'CMAES'`	Optimization algorithm for the hyperparameters calibration (Section 1.6.2)
		`'none'`	No selected optimization algorithm. Default or user-given values of the hyperparameters will be considered (Section 2.5.7.1)
		`'BFGS'`	Gradient-based optimization (BFGS) using MATLAB's built-in `fmincon` function
		`'CMAES'`	Covariance matrix adaptation - evolution scheme
		`'CE'`	Cross-entropy method for optimization (Section 1.6.2.1)
		`'GA'`	Genetic algorithm using MATLAB's built-in `ga` function
		`'HCMAES'`	Hybrid CMA-ES: CMA-ES followed by a gradient-based refinement of the solution using MATLAB's built-in `fmincon` function
		`'HCE'`	Hybrid cross-entropy: Cross-entropy method for optimization followed by a gradient-based refinement of the solution using MATLAB's built-in `fmincon` function
		`'HGA'`	Hybrid genetic algorithm: Genetic algorithm followed by a gradient-based refinement of the solution using MATLAB's built-in `ga` and `fmincon` functions
☐	`.InitialValue`	Table 9	A structure containing the initial values of the hyperparameters (See Section 1.6.1)
☐	`.MaxIter`	Integer default: 10	Maximum number of iterations allowed in the optimization algorithms
☐	`.Bounds`	Table 10	A structure containing the bounds of the search space for the calibration of the hyperparameters
☐	`.Calibrate`	Table 11	A structure containing flag specifying which hyperparameters should be calibrated (by default all parameters are calibrated)

☐	.Display	String default: matches with global display level	Level of information displayed by the methods.
		'none'	Minimum display level, displays nothing.
		'iter'	Maximum display level, shows detailed information in each iteration of the algorithm
		'final'	Shows information regarding only the convergence of the algorithm, if any.
☐	.Tol	Double default: 1e-3	Termination tolerance on the objective function Only applies when gradient-based algorithm is considered
⊞	.CMAES	Table 12	Options relevant to the CMA-ES optimization method
⊞	.HCMAES	Table 13	Options relevant to the HCMAES optimization method
⊞	.GA	Table 14	Options relevant to the GA optimization method
⊞	.HGA	Table 15	Options relevant to the HGA optimization method
⊞	.CE	Table 16	Options relevant to the CE optimization method
⊞	.HCE	Table 17	Options relevant to the HCE optimization method
⊞	.BFGS	Table 18	Options relevant to the BFGS optimization method

Table 9: Metaopts.Optim.InitialValue

☐	.C	Positive Double default: See Eq. (2.4)	Initial value of the penalty term • For BFGS, the default value is given by Table 5 • For the other algorithms, the default value is the center of the search space
☐	.epsilon	Positive Double	Initial value of the insensitive tube width • For BFGS, the default value is given by Table 5 • For the other algorithms, the default value is the center of the search space

☐	`.theta`	$1 \times N_\theta$ Double	Initial values of the kernel parameters • For BFGS, the default value is given in Table 5 • For the other algorithms, the default value is the center of the search space

Table 10: `Metaopts.Optim.Bounds`

☐	`.C`	2×1 Double default: See Eq. (2.8)	A vector in the form `[lower_C; upper_C]` • The default upper bound differs from that of Eq. (2.8) when the sample size is larger than 300 (See note in Section 2.5.7.3)
☐	`.epsilon`	2×1 Double default: See Eq. (2.9)	A vector in the form `[lower_eps; upper_eps]` • The default upper bound differs from that of Eq. (2.9) when the sample size is larger than 300 (See note in Section 2.5.7.3)
☐	`.theta`	$2 \times N_\theta$ Double	A vector in the form `[lower_theta; upper_theta]` • Defaults for radial basis families of kernel (σ): lower bound using Eq. (2.10), upper bound using Eq. (2.11) • Default for polynomial kernel: Lower d: 10^{-3}, upper d: 10 (Eq. (1.12)) • Default for sigmoid kernel: Lower a: 10^{-1}, upper a: 10, lower b: -5 and upper b: 0 (Eq. (1.13))

Table 11: `Metaopts.Optim.Calibrate`

☐	`.C`	Logical default: true	A logical defining whether the penalty term C should be calibrated or not
☐	`.epsilon`	Logical default: true	A logical defining whether the insensitive tube width ε should be calibrated or not
☐	`.theta`	Logical default: true	A logical defining whether the kernel parameters $\boldsymbol{\theta}$ should be calibrated or not

Table 12: `Metaopts.Optim.CMAES`

☐	`.nPop`	Positive integer default: $10 \times (\text{floor}(4 + 3\log N_\gamma))$	The population size N_{pop} of each generation (See μ in Section 1.6.2.2)
☐	`.ParentNumber`	Positive integer default: $\lfloor N_{pop}/2 \rfloor$	Number of points in the current population selected to generate the offsprings (See λ in Section 1.6.2.2)
☐	`.sigma`	$1 \times N_\gamma$ Double default: $\boldsymbol{\sigma}_0 = (\boldsymbol{\gamma}_{\max} - \boldsymbol{\gamma}_{\min})/3$	Initial value of the global step size of the CMA-ES algorithm (See Section 1.6.2.2) • The given vector should concatenate the values for all the hyperparameters, *i.e.* in the form `[sigma_C, sigma_eps, ... sigma_theta]` • This should be consistent with the actual parameters to optimize, *e.g.*if `.Calibrate.C = false`, then `.sigma` is in the form `sigma_eps, sigma_theta`.
☐	`.nStall`	Positive integer default: $5\lceil 30 \times N_\gamma/N_{pop} \rceil$	The maximum number of stall generations
☐	`.TolFun`	Double default: 1e-3	Termination tolerance on the cost function: The algorithm stops if the average relative change in the best cost function value over `.nStall` generations is less than or equal to `.TolFun`
☐	`.TolX`	Double default: 1e-3	Termination tolerance on X: The algorithm stops if the average relative change in the best solution over `.nStall` generations is less than or equal to `.TolX`
☐	`.FvalMin`	Double default: 1e-12	Termination tolerance on cost function: The algorithm stops if the cost function value in a given iteration is less or equal to `.FvalMin`

Table 13: `Metaopts.Optim.HCMAES`

☐	`.nPop`	Positive integer default: $10 \times (\text{floor}(4 + 3\log N_\gamma))$	The population size N_{pop} of each generation (See μ in Section 1.6.2.2)
☐	`.ParentNumber`	Positive integer default: $N_{pop}/2$	Number of points in the current population selected to generate the offsprings (See λ in Section 1.6.2.2)

☐	`.sigma`	$1 \times N_\gamma$ Double default: $\boldsymbol{\sigma}_0 = (\boldsymbol{\gamma}_{\max} - \boldsymbol{\gamma}_{\min})/3$	Initial value of the global step size of the CMAES algorithm. (See Section 1.6.2.2) • The given vector should concatenate the values for all the hyperparameters, *i.e.* in the form `[sigma_C, sigma_eps, ... sigma_theta]` • This should be consistent with the actual parameters to optimize, *e.g.*if `.Calibrate.C = false`, then `.sigma` is in the form `sigma_eps, sigma_theta`.
☐	`.nStall`	Positive integer default: $5\lceil 30 \times N_\gamma / N_{pop} \rceil$	The maximum number of stall generations
☐	`.TolFun`	Double default: 1e-3	Termination tolerance on the cost function: The algorithm stops if the average relative change in the best cost function value over `.nStall` generations is less than or equal to `.TolFun`
☐	`.TolX`	Double default: 1e-3	Termination tolerance on X: The algorithm stops if the average relative change in the best solution over `.nStall` generations is less than or equal to `.TolX`
☐	`.FvalMin`	Double default: 1e-12	Termination tolerance on cost function: The algorithm stops if the cost function value in a given iteration is less or equal to `.FvalMin`
☐	`.MaxFunEvals`	Integer default: 200	The maximum number of function evaluations in the BFGS part
☐	`.nLM`	Positive integer default: 5	The number of past iterations remembered when computing the Hessian by a limited-memory, large-scale quasi-Newton solver (BFGS algorithm).

Table 14: `Metaopts.Optim.GA`

☐	`.nPop`	Positive integer default: $\min(20, \text{floor}(4 + 3\log N_\gamma))$	The population size of each generation
☐	`.nStall`	Positive integer default: 2	The maximum number of stall generations

Table 15: `Metaopts.Optim.HGA`			
☐	`.nPop`	Positive integer default: $\min\left(20, \text{floor}(4 + 3\log N_\gamma)\right)$	The population size of each generation
☐	`.nStall`	Positive integer default: 2	The maximum number of stall generations
☐	`.nLM`	Positive integer default: 5	The number of past iterations remembered when computing the Hessian by a limited-memory, large-scale quasi-Newton solver (BFGS algorithm).

Table 16: `Metaopts.Optim.CE`			
☐	`.nPop`	Positive integer default: 50	The population size N_{pop} of each generation (See Section 1.6.2.1)
☐	`.qElite`	Double (between 0 and 1) default: 0.05	Ratio of points in the current population selected to generate the offsprings (ρ in Section 1.6.2.1)
☐	`.sigma`	$1 \times N_\gamma$ Double default: $\boldsymbol{\sigma}_0 = (\boldsymbol{\gamma}_{\max} - \boldsymbol{\gamma}_{\min})/3$	Initial value of the global step size of the CE algorithm (See Section 1.6.2.1) • The given vector should concatenate the values for all the hyperparameters, *i.e.* in the form `[sigma_C, sigma_eps, ... sigma_theta]` • This should be consistent with the actual parameters to optimize, *e.g.*if `.Calibrate.C = false`, then `.sigma` is in the form `sigma_eps, sigma_theta`.
☐	`.nStall`	Positive integer default: 2	The maximum number of stall generations
☐	`.TolFun`	Positive integer default: 1e-3	Termination tolerance on the cost function: The algorithm stops if the average relative change in the best cost function value over `.nStall` generations is less than or equal to `.TolFun`
☐	`.TolSigma`	Double or $1 \times N_\gamma$ Double default: 1e-2	Termination tolerance on the global step length: The algorithm stops if the ratio between the current global step length and the initial one (`.sigma`) is below or equal to `.TolFun`
☐	`.alpha`	Double default: 0.4	Smoothing parameter of the updating scheme (α^{CE} in Eq. (1.31))

☐	`.beta`	Double default: 0.4	Smoothing parameter of the updating scheme (β^{CE} in Eq. (1.32))
☐	`.q`	Double default: 10	Smoothing parameter of the updating scheme (q^{CE} in Eq. (1.32))

Table 17: `Metaopts.Optim.HCE`

☐	`.nPop`	Positive integer default: 50	The population size N_{pop} of each generation (See Section 1.6.2.1)
☐	`.qElite`	Double (between 0 and 1) default: 0.05	Ratio of points in the current population selected to generate the offsprings (ρ in Section 1.6.2.1)
☐	`.sigma`	$1 \times N_\gamma$ Double default: $\boldsymbol{\sigma}_0 = (\boldsymbol{\gamma}_{\max} - \boldsymbol{\gamma}_{\min})/3$	Initial value of the global step size of the CE algorithm (See Section 1.6.2.1) • The given vector should concatenate the values for all the hyperparameters, *i.e.* in the form `[sigma_C, sigma_eps, ... sigma_theta]` • This should be consistent with the actual parameters to optimize, *e.g.*if `.Calibrate.C = false`, then `.sigma` is in the form `sigma_eps, sigma_theta`.
☐	`.nStall`	Positive integer default: 2	The maximum number of stall generations
☐	`.TolFun`	Double default: 1e-3	Termination tolerance on the cost function: The algorithm stops if the average relative change in the best cost function value over `.nStall` generations is less than or equal to `.TolFun`
☐	`.TolSigma`	Double or $1 \times N_\gamma$ Double default: 1e-2	Termination tolerance on the global step length: The algorithm stops if the ratio between the current global step length and the initial one (`.sigma`) is below or equal to `.TolFun`
☐	`.alpha`	Double default: 0.4	Smoothing parameter of the updating scheme (α^{CE} in Eq. (1.31))
☐	`.beta`	Double default: 0.4	Smoothing parameter of the updating scheme (β^{CE} in Eq. (1.32))
☐	`.q`	Double default: 10	Smoothing parameter of the updating scheme (q^{CE} in Eq. (1.32))

☐	.nLM	Positive integer default: 5	The number of past iterations remembered when computing the Hessian by a limited-memory, large-scale quasi-Newton solver (BFGS algorithm).

Table 18: Metaopts.Optim.BFGS

☐	.nLM	Positive integer default: 5	The number of past iterations remembered when computing the Hessian by a limited-memory, large-scale quasi-Newton solver (BFGS algorithm).
☐	.StartPoints	Positive integer default: 1	The number of starting points for the BFGS algorithm. The default starting point, either given by the user or heuristically calculated, is always considered. The remaining points are sampled uniformly in the search space.

Table 19: Metaopts.Optim.GS

☐	.DiscPoints	Positive integer or $1 \times N_\gamma$ Positive integer default: 5	Number of discretization point per optimization dimension when using grid search

Table 20: Metaopts.Optim.HGS

☐	.DiscPoints	Positive integer or $1 \times N_\gamma$ Positive integer default: 5	Number of discretization point per optimization dimension in the grid search part of the algorithm
☐	.nLM	Positive integer default: 5	The number of past iterations remembered when computing the Hessian by a limited-memory, large-scale quasi-Newton solver (BFGS algorithm).

3.1.6 Validation Set

If a validation set is provided, UQLAB automatically calculates the validation error of the created SVR model. The required information is listed in Table 21.

Table 21: MetaOpts.ValidationSet

●	.X	$N \times M$ Double	User-specified validation set $\boldsymbol{\mathcal{X}}_{Val}$
●	.Y	$N \times N_{Out}$ Double	User-specified validation set response $\mathcal{Y}_{Val}$

3.2 Accessing the results

Syntax

```
mySVR = uq_createModel(MetaOpts) ;
```

Output

Regardless on the configuration options given at creation time in the `MetaOpts` structure, all SVR metamodels share the same output structure, given in Table 22.

Table 22: `mySVR`		
`.Name`	String	Unique name of the SVR metamodel
`.Options`	Table 2	Copy of the `Metaopts` structure used to create the metamodel
`.SVR`	Table 23	Information about the final model
`.ExpDesign`	Table 28	Experimental design used for calculating the coefficients
`.Error`	Table 26	Error estimates of the metamodels's accuracy
`.Internal`	Table 29	Internal state of the MODEL object (useful for debug/diagnostics)

Table 23: `mySVR.SVR`		
`.Hyperparameters`	Table 24	The values the hyperparameters of γ (See Section 1.6.1)
`.Coefficients`	Table 25	The values of the different coefficients in the expansion in Eq. (1.9)
`.Kernel`	Table 27	Information about the kernel function

Table 24: `mySVR.SVR.Hyperparameters`		
`.C`	Double	The value of the penalty term C (See Section 1.6.1)
`.epsilon`	Double	The value of the ε-insensitive width tube (Eq. (1.2))
`.theta`	Variable size double	The value of the kernel hyperparameters (See Section 1.3.1)

Table 25: `mySVR.SVR.Coefficients`		
`.alpha`	$2N \times 1$ Double	The value of the coefficients of the expansion in Eq. (1.9)
`.beta`	$N \times 1$ Double	The value of the vector $\boldsymbol{\alpha} - \boldsymbol{\alpha}^*$ in Eq. (1.9)

.bias	Double	The value of the bias term in Eq. (1.9)
.SVidx	Variable size double	The index of the support vectors (non-zero coefficients)

Table 26: mySVR.Error		
.LOO	Double	The Leave-One-Out error (See Eqs. (1.23,1.26))
.LOO_norm	Double	The normalized Leave-One-Out error
.Val	Double	Validation error (see Eq. (1.29) and Section 2.6). Only available if a validation set is provided (see Table 21).

Note: In general the fields mySVR.SVR and mySVR.Error are structure arrays with length equal to the number of outputs N_{out} of the metamodel.

Table 27: mySVR.SVR.Kernel		
.Handle	function handle	The user defined kernel family function handle (See Section 2.5.3.2)
.Family	String or function handle	The chosen kernel family, *i.e.* the elementary 1-D function that is used by the kernel function (See Section 1.3.1)
.Isotropic	boolean	A boolean showing whether the kernel is isotropic or not (only available for radial basis families of kernels)

Table 28: mySVR.ExpDesign		
.NSamples	Integer	The number of samples
.Sampling	String	The sampling method
.Input	INPUT object	The input module that was used to generate the experimental design (X)
.X	$N \times M$ Double	The experimental design values
.U	$N \times M$ Double	The experimental design values in the reduced space (See Section 2.9.1)
.Y	$N \times N_{out}$ Double	The output Y that corresponds to the input X

Table 29: mySVR.Internal		
.Runtime	Structure	Variables that are used during the calculation of the SVR metamodel

`.SVR`	Structure	All the parameters set and calculated to completely define the generated SVR model
`.AuxSpace`	INPUT object	Auxiliary space where SVR is performed (See Section 2.9.1)

Note: The internal fields of the SVR module are not intended to be accessed or changed for most typical usage scenarios of the module.

3.3 Evaluating the model

Syntax

```
Y = uq_evalModel(X)
Y = uq_evalModel(mySVR, X)
```

Description

`Y = uq_evalModel(X)` returns the value of the SVR prediction ($N \times N_{\text{out}}$ Double) on the points `X` ($N \times M$ Double) using the last created SVR model.

> **Note:** by default, the *last created* model or surrogate model is the currently active model.

`Y = uq_evalModel(mySVR,X)` returns the value of the SVR prediction ($N \times N_{\text{out}}$ Double) on the points `X` ($N \times M$ Double) using the SVR model object `mySVR`.

References

Bompard, M. (2011). *Modèles de substitution pour l'optimisation globale de forme en aérodynamique et méthode locale sans paramétrisation*. Ph. D. thesis, Université de Nice Sophia-Antipolis, France. 4, 6

Bourinet, J.-M. (2015). Reliability assessment with adaptive surrogates based on support vector machine regression. In G. S. e. M. Papadrakakis, V. Papadopoulos (Ed.), *Proc. 1st ECCOMAS Thematic Conference on Uncertainty Quantification in Computational Sciences and Engineering, Crete Island, Greece, May 25-27*. 10

Bourinet, J.-M. (2018). *Reliability analysis and optimal design under uncertainty - Focus on adaptive surrogate-based approaches*. Université Blaise Pascal, Clermont-Ferrand, France. Habilitation à diriger des recherches, 243 pages. 1, 3, 6

Chang, M.-W. and C.-J. Lin (2005). Leave-one-out bounds for support vector regression model selection. *Neural Computation 17*(5), 1188–1222. 6, 7

Chapelle, O., V. Vapnik, and Y. Bengio (2002). Model selection for small sample regression. *Journal of Machine Learning Research 48*(1), 9–23. 7, 8

Chapelle, O., V. Vapnik, O. Bousquet, and S. Mukherje (2002). Choosing multiple parameters for support vector machines. In N. Cristianini (Ed.), *Machine Learning*, Volume 46, pp. 131 – 159. 8

Cherkassky, V. and Y. Ma (2004). Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks 17*(1), 113–126. 24

Evgeniou, T., M. Pontil, and T. Poggio (2000). Regularization networks and support vector machines. *Advances in Computational Mathematics 13*(1), 1–50. 9

Gunn, S. (1998). Support vector machines for classification and regression. Technical Report ISIS-1-98, Dpt. of Electronics and Computer Science, University of Southampton. 2, 6

Hansen, N. (2001). The CMA evolution strategy: A tutorial. Technical report, Institut National de Recherche en Informatique et en Automatique (INRIA), France. 12

Hansen, N. and S. Kern (2004). Evaluating the CMA evolution strategy on multimodal test functions. *Evolutionary Computation 9*(2), 282–221. 12

Hansen, N. and A. Ostermeier (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation 9*(2), 159–195. 10, 12

Kecman, V., T.-M. Huang, and M. Vogt (2005). *Iterative single data algorithm for training kernel machines from huge data sets: theory and performance*, pp. 255–274. Springer Berlin Heidelberg. 6

Kroese, D. P., S. Porotsky, and R. Y. Rubinstein (2006). The cross-entropy method for continuous multi-extremal optimization. *Methodology and Computing in Applied Probability 8*, 383–407. 10

Moustapha, M. (2016). *Adaptive surrogate models for the reliable lightweight design of automotive body structures*. Ph. D. thesis, Université Blaise Pascal, Clermont-Ferrand, France. 10

Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in kernel methods*, pp. 185–208. MIT Press. 6

Saunders, C., A. Gammerman, and V. Vovk (1998). Ridge regression learning algorithm in dual variables. In *Proc. of the 15th International Conference on Machine Learning*, pp. 515–521. Morgan Kaufmann Publishers Inc. 6

Smola, A. and B. Schölkopf (2004). A tutorial on support vector regression. *Statistics and Computing 14*, 199–222. 1, 2

Vanderbei, R. J. (1994). LOQO: an interior point code for quadratic programming. Technical report, Optimization Methods and Software. 6

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York. 1, 4

Vapnik, V. and O. Chapelle (2000). Bounds on error expectation for support vector machines. *Neural Computation 12*(9), 2013–2036. 7