Multi-fidelity surrogate modeling adapted to functional outputs for uncertainty quantification of complex models

Baptiste Kerleguer

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAX115

Thèse de doctorat

ÉCOLE POLYTECHNIQUE

IP PARIS

Multi-fidelity surrogate modeling adapted to functional outputs for uncertainty quantification of complex models

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École polytechnique

École doctorale n°574 École doctorale de mathématiques Hadamard (EDMH)
Spécialité de doctorat: Mathématiques appliquées

Thèse présentée et soutenue à Palaiseau, le 2 décembre 2022, par

BAPTISTE KERLEGUER

Composition du Jury :

Béatrice Laurent Professeure, IMT, INSA de Toulouse	Présidente et Rapporteure
Serge Guillas Professeur, University College London	Rapporteur
Erwan Scornet Professeur assistant, CMAP, École polytechnique	Examinateur
Merlin Keller Ingénieur chercheur, EDF - R&D	Examinateur
Anthony Nouy Professeur, LMJL, Centrale Nantes	Examinateur
Josselin Garnier Professeur, CMAP, École polytechnique	Directeur de thèse
Claire Cannamela Ingénieur chercheur, CEA	Co-encadrante de thèse

Résumé

Cette thèse porte sur l'approximation de la sortie d'un code de calcul complexe dans un cadre multi-fidélité, c'est-à-dire que le code peut être exécuté à différents niveaux de précision avec différents coûts de calcul. Ainsi, les qualités de prédictions du métamodèle de la sortie d'un code complexe peuvent être améliorées en utilisant en plus des simulations moins précises mais plus nombreuses (car moins coûteuses) .

Ce travail a pour objectif d'étendre les méthodes de métamodélisation multi-fidélité lorsque les sorties du code sont fonctionnelles. Tout d'abord, une première approche permettant de combiner les réseaux de neurones bayésiens et les processus gaussiens est proposée. Ce modèle est adapté lorsque les relations entre les codes basse et haute fidélité est non linéaire, mais n'est pas encore adapté aux sorties fonctionnelles.

Dans un second temps, une approche utilisant la décomposition en ondelettes et les processus gaussien est proposée. Cette approche permet de développer un processus gaussien dans l'espace des ondelettes qui est équivalent à un processus gaussien dans l'espace temporel. Cette méthode est naturellement adaptée aux sorties fonctionnelles.

Enfin, la troisième méthode proposée associe une méthode de réduction de dimension de la sortie avec une méthode de régression par processus gaussien à covariance tensorisée. Cette approche est développée dans le cadre de sortie de type série-temporelle. Les expressions analytiques de la moyenne et de la variance de prédiction sont également introduites.

Abstract

This thesis focuses on approximating the output of a complex computational code in a multi-fidelity framework, i.e. the code can be run at different levels of accuracy with different computational costs. Thus, the predictive qualities of the surrogate model of the output of a complex code can be improved by using in addition less accurate but more numerous (because less expensive) simulations.

This work aims to extend multi-fidelity surrogate modeling methods when the code outputs are functional. First, an approach allowing to combine Bayesian neural networks and Gaussian processes is proposed. This model is suitable when the relationship between the low- and high-fidelity codes is non-linear, but is not yet suitable for functional outputs.

In a second step, an approach using wavelet decomposition and Gaussian processes is proposed. This approach allows to develop a Gaussian process in the wavelet space which is equivalent to a Gaussian process in the time space. This method is naturally adapted to functional outputs.

Finally, the third proposed method combines an output dimension reduction method with a covariance tensorised Gaussian process regression method. This approach is developed in the context of time-series output. The analytical expressions for the predictive mean and variance are also introduced.

Remerciements

J'exprime ici ma profonde gratitude à madame Béatrice Laurent de m'avoir fait l'honneur de présider mon jury de thèse, mais aussi d'avoir accepté de rapporter ma thèse. Je remercie profondément monsieur Serge Guillas d'avoir accepté de rapporter ma thèse ainsi que pour les discussions sur des projets que j'espère voir aboutir bientôt. Je tiens ensuite à remercier messieurs Merlin Keller, Erwan Scornet et Anthony Nouy d'avoir accepté de participer à mon jury de thèse.

J'ai eu la chance de rencontrer Claire Cannamela et Josselin Garnier qui ont accepté de me prendre comme doctorant. Je tiens à les remercier de m'avoir introduit aux problématiques de la quantification d'incertitude de code de calcul. De plus, je souhaite relever leur implication humaine et notamment lors des moments difficiles de cette thèse, je pense notamment à la pandémie qui n'était pas prévue dans le programme de la thèse. Il reste encore beaucoup de choses à faire et j'ai hâte de savoir et pouvoir faire plus. Je souhaite également remercier les "permanents" des labos où j'ai eu l'occasion de faire ma thèse. Je pense en particulier à Cédric, Isabelle, Cécile, Guillaume, Gilles, Philippe, William, Gwenael, Aymeric, Cyril, Aline, Marine, Lucas, Nicole. La communauté Mascot-Num a été aussi l'occasion d'échanges très enrichissant pour moi lors de ces 3 ans et j'espère continuer à participer à cet environnement exceptionnel.

Que serait la thèse sans les autres doctorants ? J'ai eu la chance de pouvoir naviguer entre deux labos, le CEA et le CMAP. Je partageais, je partage et je partagerai ton bureau au CEA, je te remercie Cécile pour tout dans et hors du travail. J'espère que nos travaux futures seront encore plus féconds que le passé. Grace au CEA, j'ai eu la chance de travailler aux coté d'un grand nombre de doctorants dans la joie et la bonne humeurs, je tiens donc à remercier : Matthieu, Marie, Grégoire, Jean-Cédric, Christina, Eric, Ronnan, Victor, Olivier, Mathilde, Correntin, Albertine, Bastien, Mattéo, Sébastien, Corentin, Kévin, Joel, Ulrich, Léa et Marin. Un gros bisous aussi à Louise. Je n'oublie pas les post-docs Charles, Adrien et Etienne qui ont eu une place spéciale dans ma construction. Que serait la cave du bâtiment B sans les stagiaires ? Merci à vous tous d'avoir été présent à un moment pour moi, même ceux qui n'ont jamais passé la porte. En plus des doctorants du CEA, j'ai eu la chance d'aller à polytechnique deux fois par semaine, ce qui m'a permis d'enseigner, mais aussi de m'enrichir des rencontres de doctorants en math. En premier, je pense à Clément qui a squatté mon bureau, mon directeur de thèse et mes processus Gaussien ! Ce fut vraiment un plaisir de partager avec toi mes joies et, mais pênes lors de ces trois ans. Ma volonté encore fragile de faire des applications te doit beaucoup. J'espère qu'on pourra un jour faire de belles choses ensemble. Le CMAP a été agité lors de mes 3 années de thèse par le CJC-MA et j'ai eu la chance de travailler en étroite collaboration avec Pierre pour faire tenir debout cette fragile machine. Je pense aussi à Apolline et à Solange, le CJC vous doit tout ! J'ai également eu la chance de discuter avec les doctorants du groupe SIMPAS (ou sympa, ou sympas ...). Merci à vous. Je pense en particulier à Margaux, je n'ai rien de plus à te dire que merci, mais tu sais tout ce que je te dois. Il m'est très difficile de faire la liste des doctorants du CMAP, pas uniquement à cause de ma mauvaise mémoire, mais aussi car vous être tellement nombreux. Mais je tiens à remercier ceux dont j'ai été très proche lors de ces 3 années, notamment Arthur, Apolline, Benjamin, Cheikh, Clément($\times$ beaucoup), Corentin, Eugénie, Joffrey, Julie, Louis, Manon, Naoufal (team Josselin). Merci aussi à ceux que j'ai oubliés.

Il me vient maintenant une pensée pour tous les enseignants que j'ai eus lors de mes nombreuses années d'études. Contrairement à ce que j'essaie de faire croire dans mon CV le chemin n'a pas été si tracé et mes enseignants m'ont aidé Je tiens en particulier à remercier Madame Lilette qui a été ma prof de math en seconde et qui m'a donné le goût des problèmes difficiles pour moi. Il me semble essentiel de remercier l'ensemble de mes professeurs de classe préparatoire qui tous m'ont appris beaucoup. J'ai eu ensuite la chance d'être accueilli dans un endroit les plus extraordinaires pour un étudiant passionné de science comme moi : ENS Cachan. Je remercie mes camarades de classe et mes collèges "normaliens" pour l'ensemble de leur uvre. Je sais que vous changerez le monde tous les jours. Il me faut m'arrêter

un instant pour penser à Antonin Girardi, j'espère que d'où tu es tu nous pardonnes de ne pas faire assez pour les autres. Lors de mes 3 premières années à Cachan j'ai eu la chance d'avoir pour chef de département Thomas Rodet. Je te remercie d'avoir toujours eu ta porte ouverte quand on en avait besoin et j'en avais souvent besoin, merci.

Je tiens ensuite à remercier mes amis et tous ceux qui au-delà de ma thèse m'ont fait devenir la personne que je suis aujourd'hui. Il me faut commencer ce que j'ai rencontré il y a longtemps. Merci Guillaume, merci Matthieu, merci à Léa et Paul et à tous ceux que j'ai rencontrés à l'école et qui sont restés des gens géniaux ! Merci à toi Delphine, c'est un plaisir de te connaitre et de profiter de ton engagement. Merci Gwendoline d'avoir ajouté de l'aléatoire dans mon emploi du temps de thèse, et aussi d'avoir contribué à mon équilibre depuis des années. Merci Marie-Pierre, j'ai longtemps hésité à savoir si je te mettais dans la case des anciens profs, des cachanais ou des amis. Merci pour tout, tu ne peux pas savoir à quel point tu as été un soutien fort pour moi. Merci aussi pour les corrections d'Anglais ! Et au risque d'être redondant, merci à Cécile, tu as commencé par être ma cobureau mais tu es maintenant une amie. Grâce à toi, j'ai eu la chance de rencontrer deux personnes exceptionnelles : Nicolas et Léonard, en attend la suite ;). Merci aussi à Grégoire et Mathieu, j'ai eu beaucoup de plaisir à passer du temps avec vous.

Il faut aussi noter que j'ai eu la joie de naitre dans une famille exceptionnelle. Merci à mes oncles et tantes pour de m'avoir accompagné et de m'avoir beaucoup appris sur beaucoup de sujet. J'ai une pensée particulière pour Matthieu qui m'avait conseillé quand j'ai choisi de faire un stage puis une thèse au CEA. Cette thèse a été marquée par toi. Je remercie aussi mes deux grands-mères pour leur soutien tout au long de ma vie. J'espère aussi que mes deux grands-pères seraient heureux de me voir docteur en mathématiques appliquées. Merci aussi à vous mes cousins, des plus éloignées aux plus proches. Il y aura toujours une petite place pour vous dans mon coeur. En somme, merci à toute la famille Benoist/K-erleguer. J'en arrive à ma famille d'adoption, merci pour vos conseils avisés sur la thèse, mais pas que. Merci d'avoir une fille si extraordinaire. J'espère ne pas vous avoir déçu avec cette thèse. Merci Laurence et merci Hervé de m'avoir fait naitre, élevé et de m'avoir donné les clés pour faire ce que je souhaite. Je suis heureux d'être votre fils, j'espère que vous êtes heureux d'être mes parents. Merci à Soizic et Maëlle d'être mes super soeurs. J'ai eu beaucoup de chances d'être votre frère. Merci de m'avoir supporté pendant toute notre enfance.

Il me reste encore trop peu de mots pour remercier celle que j'ai la chance d'aimer et dont je crois qu'elle m'aime en retour. Merci à toi Aliénor, tu m'as permis d'être moi-même et de réaliser ce que je voulais. *J'aimerais quand même te dire ; Tout ce que j'ai pu écrire ; Je l'ai puisé à l'encre de tes yeux* (Francis Cabrel), ce n'est pas complètement vrai, moi tout ce que j'ai fait, je l'ai fait à l'ombre de tes yeux. Merci.

Summary

Introduction en Français

La simulation numérique est devenue un outil indispensable pour modéliser et prédire des phénomènes. Elle est basée sur l'utilisation de codes de calcul complexes et coûteux. Une simulation nécessite l'utilisation de paramètres d'entrée décrivant le phénomène étudié et renvoie des sorties permettant d'accéder aux caractéristiques d'intérêt du phénomène en question. Les codes de calcul sont basés sur la résolution d'équations mathématiques de plus en plus élaborées, c'est ainsi que les simulations sont devenues très précises mais aussi très coûteuses en temps de calcul. Par conséquent, l'utilisation d'un très grand nombre de simulations n'est souvent pas envisageable. Lorsque les paramètres d'entrée sont entachés d' incertitudes, il est nécessaire de quantifier l'impact de celles-ci sur les sortie du code de calcul. De telles études sont appelées analyses d'incertitude. Elles ne se limitent pas à quelques simulations autour d'un point nominal, mais nécessitent une utilisation massive du code de calcul.

Il devient alors essentiel d'approcher les sorties d'intérêt du code par une fonction à faible coût de calcul. De telles fonctions sont appelées métamodèles (ou modèles de substitution) et apprennent la relation entre les paramètres d'entrée et les sorties à partir d'un nombre limité d'appels au code de calcul.

La sortie du code est supposée être la fonction g :

$$\begin{aligned} \mathrm{g} \quad : Q \in \mathbb{R}^d \quad &\to \mathbb{R}^q \\ \mathbf{x} \qquad &\mapsto \mathrm{g}(\mathbf{x}) \end{aligned}$$

où Q est un ensemble non vide appelé espace d'entrée. L'objectif est de construire un métamodèle $\tilde{\mathrm{g}}$ de la fonction g. Dans ce manuscrit, on suppose qu'aucune information sur la fonction g n'est disponible, ce problème est appelé problème de " boîte noire ".

Les informations disponibles pour la construction du métamodèle sont définies par les N observations de la fonction g aux points du plan d'expériences $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\}$. Ainsi nous disposons des données suivantes pour construire le métamodèle $\left\{\left(\mathbf{x}^{(i)}, \mathrm{g}(\mathbf{x}^{(i)})\right), \text{for } i = 1, \cdots, N\right\}$ appelé ensemble d'apprentissage.

Comme les codes de calcul intègrent une modélisation des phénomènes de plus en plus approfondie, les sorties qui en découlent sont elles aussi davantage détaillées. La sortie d'intérêt peut alors être de très

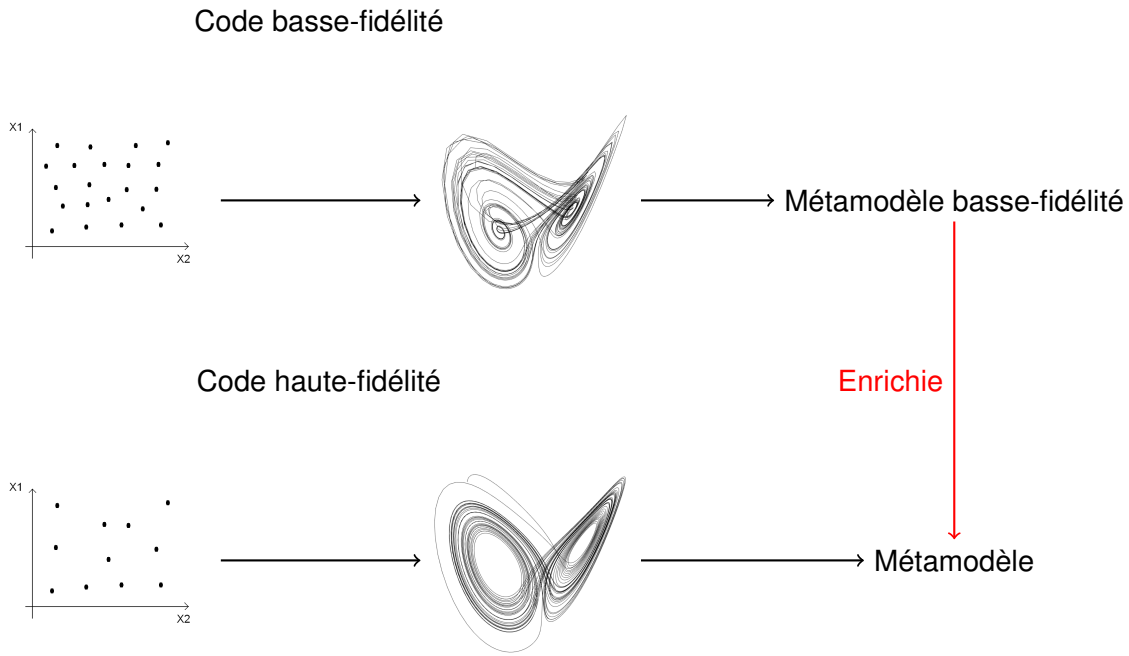


Figure 1: Illustration de la bi-fidélité avec l'exemple du système Lorentz 63. Pour la basse fidélité, une méthode d'Euler est utilisée pour résoudre le système, tandis qu'une methode de Runge-Kutta du quatrième ordre est utilisée pour la haute fidélité.

grande dimension, c'est notamment le cas des images ou des signaux. Le contexte de cette thèse s'inscrit dans le cadre où les sorties sont des fonctions dépendant du temps. Lorsqu'elles sont échantillonnées, elles sont appelées séries temporelles. On suppose que l'échantillonnage se fait sur une grille fine et régulière, ce qui implique que la sortie est de très haute dimension.

Par ailleurs, il peut exister des versions antérieures ou d'étude d'un code de calcul, qui sont moins précises mais aussi moins coûteuses. Une version moins précise sera naturellement moins coûteuse. Lorsque les versions peuvent être classées en fonction de leur précision et de leur coût, on parle de cadre hiérarchique multifidélité. L'objectif de la métamodélisation multi-fidélité est alors de construire un modèle de substitution de la sortie de la version la plus coûteuse du code de calcul tout en étant capable d'utiliser les informations obtenues à partir des versions plus rapides. C'est dans ce contexte que nous nous sommes placés dans ce manuscrit. La figure 1 illustre le principe de la multi-fidélité lorsque deux versions d'un code de calcul sont disponibles. Le cas de la métamodélisation de la sortie d'une seule version du code de calcul est appelé régression "simple-fidélité", par opposition à la multi-fidélité.

L'objectif principal de ce manuscrit est de proposer une méthodologie de construction d'un métamodèle multi-fidélité lorsque la sortie à approcher est une fonction dépendant du temps. Pour ce faire, il est incontournable de commencer par présenter les méthodes sur lesquelles seront basées nos approches. La première partie de ce manuscript s'y consacre et fait l'objet des trois premiers chapitres alors que la seconde partie, composée de trois chapitres également, expose plusieurs méthodes répondant globale-

ment ou en partie à l'objectif de départ.

Le chapitre 1 introduit le principe de la construction d'un modèle de substitution de la sortie d'un code de calcul et s'intéresse principalement à deux approches. Tout d'abord, la régression par processus gaussien, méthode populaire dans la communauté de la quantification dincertitudes de par ses expressions analytiques de prédiction et de quantification d'incertitudes, est introduite dans la section 1.1. Une extension de cette méthode aux sorties de type séries temporelles est ensuite exposée dans la section 1.1.3. Celle-ci est basée sur lexpression tensorisée de la fonction de covariance caractérisant la loi un processus gaussien. A l'instar de la régression par processus gaussien pour sorties scalaires, il est possible d'obtenir les formules analytiques de prédiction et de quantification dincertitudes. Dans un second temps, les réseaux de neurones, moins communément utilisés dans la communauté de la quantification d'incertitudes, sont présentés dans la section 1.2. Afin d'accéder aux incertitudes associées aux prédictions du métamodèle, plusieurs approches sont possibles, mais ce chapitre se focalise sur les réseaux de neurones bayésiens dans la section 1.2.2.2 ; En effet, l'idée derrière cette modélisation est proche de celle des processus gaussiens. Enfin, la section 1.3 propose différents critères dévaluation de métamodèles, utilisés tout au long du manuscrit, tant en termes de qualité de prédiction que de quantification des incertitudes.

La plupart des métamodèles présentés précédemment sont limités dans leur capacité à prédire en grande dimension. Une approche fréquemment utilisée est de réduire la dimension de la sortie et de réaliser la métamodélisation sur le problème de dimension réduite. Le chapitre 2 est ainsi dédié aux méthodes de réduction de dimension. La section 2.1 passe en revue les méthodes les plus classiques telles que la décomposition en valeurs singulières, section 2.1.1, ou les autoencodeurs, section 2.1.4. Une autre manière d'aborder la réduction de la dimension consiste à exprimer les données dans un espace où l'information est très concentrée. Une décomposition connue pour concentrer l'information est la décomposition en ondelettes. Il devient alors possible de choisir les points d'intérêt dans l'espace d'ondelettes afin de limiter la régression à ces points. C'est notamment le cas de la décomposition en ondelettes introduite dans la section 2.2.

Le chapitre 3 est dédié à la métamodélisation multi-fidélité. Le schéma auto-régressif AR(1) présenté par (Kennedy & O'Hagan, 2000) est le premier résultat important de la métamodélisation multi-fidélité. Il en est question à la section 3.1.1. Il permet de lier la sortie de plusieurs versions de codes en utilisant la régression par processus gaussien. D'autres métamodélisations utilisant des processus gaussiens peuvent être employées dans un contexte de multi-fidélité à l'image des DeepGP que nous présentons à la section 3.1.2. Comme pour la régression simple-fidélité, l'essor de la métamodélisation par réseaux de neurones conduit à la modélisation multi-fidélité par des réseaux de neurones. Plusieurs métamodélisations multi-fidélité avec réseaux de neurones sont proposées dans la section 3.2.

Le manuscrit se poursuit ensuite par l'exposition de troix méthodes originales sur la métamodèli-

sation multi-fidélité. Le premier modèle, présenté au chapitre 4, est la combinaison d'un processus gaussien et d'un réseau de neurones bayésien. L'idée est de combiner la flexibilité du réseau de neurones avec l'efficacité de la prédiction et de la quantification d'incertitudes des processus gaussiens. Pour ce faire, l'incertitude sur la sortie de processus gaussien doit être fournie en entrée du réseau de neurones bayésien. Ces travaux sont exposés dans (Kerleguer, Cannamela, & Garnier, 2022). Cependant, le passage à une grande dimension de sortie s'est avéré complexe et a imposé une limite à cette approche pour répondre à l'objectif global de départ. Cette approche est une alternative aux méthodes proposées au chapitre 3.

Ensuite, le chapitre 5 est consacré à la métamodèlisation par processus gaussien d'ondelettes. L'idée repose sur l'adaptation du modèle AR(1) à des sorties en grande dimension. Pour cela, on utilise la décomposition en ondelettes qui va concentrer les données afin de réduire la dimension de façon active grâce à la sélection de points. Pour cela, on a introduit un objet que nous avons appelé processus gaussien d'ondelettes. L'inversion de la décomposition en ondelettes permet d'obtenir la quantification des incertitudes de prédiction. Ce chapitre pose les bases nécessaires à la régression par processus gaussien sur une base d'ondelettes. Le principal apport est la modélisation dans un espace d'ondelettes avec le calcul de la fonction de covariance dans une espace d'ondelettes pour un processus gaussien de covariance connue dans l'espace temporel. La possibilité d'une extension à la multi-fidélité AR(1) est évoquée.

Enfin, le chapitre 6 présente un modèle utilisant une réduction de dimension pour construire un métamodèles multi-fidélité pour des sorties de type séries temporelles. La première approche envisagée est d'étendre la tensorisation de la covariance à la régression par processus gaussien dans un cadre multi-fidélité. Cependant, la section 6.1 aborde les problèmes de conditionnement des matrices de covariance qui entravent l'extension souhaitée. La seconde idée de ce chapitrer est d'utiliser la décomposition en valeurs singulières en tant que méthode de réduction de dimension et d'utiliser la régression par processus gaussien sur les sorties de dimension réduite. À cela est ajoutée une régression simple fidélité sur l'espace des fonctions non pris en compte dans la réduction de la dimension à l'aide une métamodélisation par processus gaussien avec covariance tensorisée. Cette méthode est présentée dans la section 6.3 et comparée à d'autres méthodes de l'état de l'art dans la section 6.4.

La métamodélisation multi-fidélité pour les sorties en grande dimension reste un problème particulièrement complexe. Ce manuscrit se concentre principalement sur les sorties séries temporelles, mais on peut aussi imaginer des images ou d'autres objets à haute dimension. De plus, seuls les réseaux de neurones et les processus gaussiens sont présentés pour la métamodélisation, mais il existe de nombreuses autres méthodes telles que les forêts aléatoires, présentées dans (Biau & Scornet, 2016), ou les polynomes de chaos, voir (Xiu & Karniadakis, 2002). Enfin, dans ce manuscrit, les métamodèles sont construits pour les performances d'emulation, alors que pour leur utilisation ils pourraient être associés à

des méthodes d’apprentissage actif, d’optimisation ou d’analyse de sensibilité par exemple.

Introduction

Numerical simulation has become an indispensable tool for modeling and predicting phenomena. It is based on the use of complex and expensive calculation codes. A simulation requires the use of input parameters describing the phenomenon under study and returns outputs allowing access to the characteristics of interest of the phenomenon in question. Computational codes are based on the resolution of increasingly elaborate mathematical equations, so that simulations have become very accurate but also very costly in terms of computing time. Therefore, the use of a very large number of simulations is often not feasible. When the input parameters are subject to uncertainties, it is necessary to quantify the impact of these uncertainties on the output of the calculation code. Such studies are called uncertainty analyses. They are not limited to a few simulations around a nominal point, but require a massive use of the calculation code.

It then becomes essential to approximate the outputs of interest of the code by a function with low computational cost. Such functions are called surrogate models (or metamodels) and learn the relationship between input parameters and outputs from a limited number of calls to the computational code.

The output of the code is assumed to be the function g :

$$\begin{aligned} \mathrm{g} \quad : Q \in \mathbb{R}^d \quad &\to \mathbb{R}^q \\ \mathbf{x} \qquad &\mapsto \mathrm{g}(\mathbf{x}) \end{aligned}$$

where Q is a non empty set called input space. The objective is to build a surrogate model $\tilde{\mathrm{g}}$ of the function g. In this manuscript, it is assumed that no information about the function g is available, this problem is called "black box" problem.

The information available for the construction of the surrogate model is defined by the N observations of the g function at the points of the experimental design $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\}$. Thus we have the following data to construct the surrogate model $\left\{\left(\mathbf{x}^{(i)}, \mathrm{g}(\mathbf{x}^{(i)})\right), \text{for } i = 1, \cdots, N\right\}$ called the learning set.

As the calculation codes incorporate more and more in-depth modeling of phenomena, the resulting outputs are also more detailed. The output of interest can then be very large, such as images or signals. The context of this thesis is where the outputs are time-dependent functions. When they are sampled,

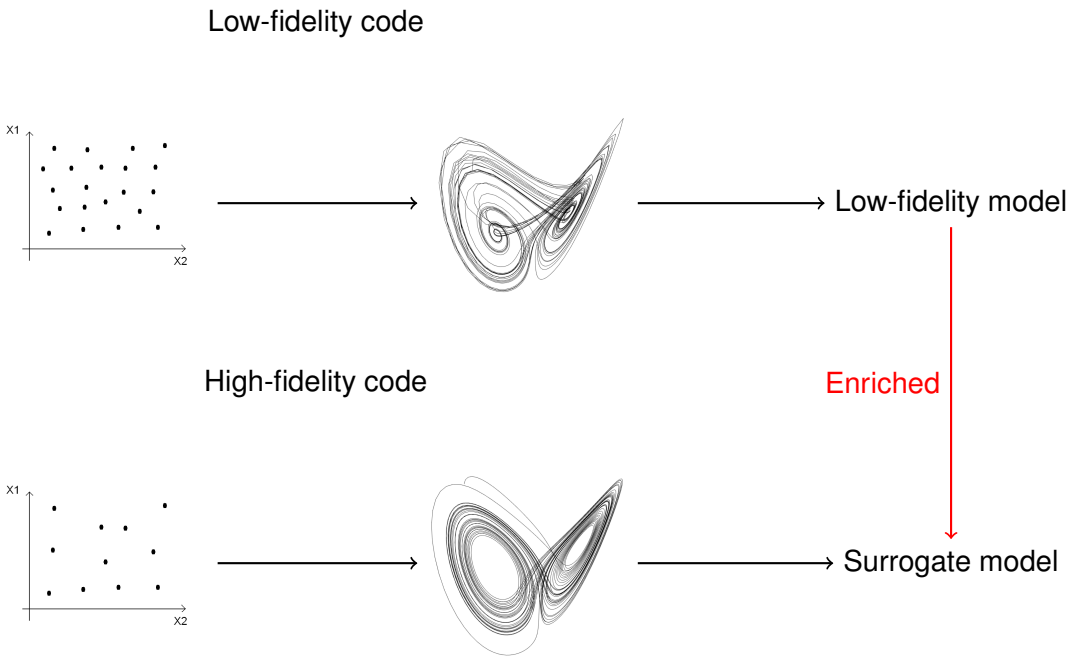


Figure 2: Illustration of bi-fidelity with the example of the Lorentz 63 system. For low-fidelity, an Euler method is used to solve the system, while a fourth order Runge-Kutta method is used for high-fidelity.

they are called time-series. It is assumed that the sampling is done on a fine and regular grid, which implies that the output is of high dimension.

Furthermore, there may be earlier or study versions of a calculation code, which are less accurate but also less expensive. A less accurate version will naturally be less expensive. When versions can be classified according to their accuracy and cost, this is called a hierarchical multi-fidelity framework. The objective of multi-fidelity surrogate modeling is then to build a surrogate model of the output of the most expensive version of the computational code while being able to use the information obtained from the faster versions. It is in this context that we have placed ourselves in this manuscript. The figure 2 illustrates the principle of multi-fidelity when two versions of a calculation code are available.

The case of surrogate modeling of the output of a single version of the computational code is called "single-fidelity" regression, as opposed to multi-fidelity.

The main objective of this manuscript is to propose a methodology for building a multi-fidelity surrogate model when the output to be approximated is a time-dependent function. In order to do so, it is essential to start by presenting the methods on which our approaches will be based. The first part of this manuscript is devoted to this and is the subject of the first three chapters, while the second part, also composed of three chapters, presents several methods that meet the initial objective in whole or in part.

First of all, methods for surrogate modeling of the output of a calculation code are presented in the chapter 1. An extension of this method to time-series output is given in section 1.1.3. It is based on the tensorised expression of the covariance function allowing to define a Gaussian process. As for the Gaus-

sian process regression, we have all the analytical formulas for prediction and uncertainty quantification. Then, we present more recent methods in the world of uncertainty quantification, such as neural networks in the section 1.2. Several methods allow to access the uncertainties associated with the predictions of the surrogate model. We focus rather on the Bayesian neural network, section 1.2.2.2, because the idea of modeling is close to the Gaussian processes. Finally, the section 1.3 proposes different criteria for evaluating surrogate models both in terms of quality and quantification of prediciton uncertainties.

Most of the surrogate models presented previously are limited in their capacity to predict in high dimension. A frequently used approach is to reduce the dimension of the output and perform the surrogate modeling on the reduced dimension problem. The chapter 2 is thus dedicated to dimension reduction methods. The section 2.1 reviews the most classical methods such as singular value decomposition, section 2.1.1, or autoencoders, section 2.1.4. Another way of approaching dimension reduction is to express the data in a space where the information is highly concentrated. One decomposition known to concentrate information is the wavelet decomposition. It then becomes possible to choose the points of interest in the wavelet space in order to limit the regression to these points. This is notably the case for the wavelet decomposition introduced in section 2.2.

The chapter 3 is dedicated to multi-fidelity surrogate modeling. The AR(1) auto-regressive scheme presented by (Kennedy & O'Hagan, 2000) is the first important result of multi-fidelity surrogate modeling. It is discussed in section 3.1.1. It allows the output of several versions of codes to be linked using Gaussian process regression. Other surrogate modeling using Gaussian processes can be used in a multi-fidelity context, such as DeepGPs, which we present in section 3.1.2. As for single-fidelity regression, the rise of neural network surrogate modeling leads to multi-fidelity modeling by neural networks. Several multi-fidelity surrogate models with neural networks are proposed in the section 3.2.

The manuscript then continues with the exposition of three methods on multi-fidelity surrogate modelling. The first model, presented in chapter 4, is the combination of a Gaussian process and a Bayesian neural network. The idea is to combine the flexibility of the neural network with the efficiency of the prediction and uncertainty quantification of Gaussian processes. To do this, uncertainty on the Gaussian process output must be provided as input to the Bayesian neural network. This work is outlined in (Kerleguer et al., 2022). However, the transition to a large output dimension proved to be complex and imposed a limit on this approach to meet the overall objective. This approach is an alternative to the methods proposed in the chapter 3.

Then, the chapter 5 is devoted to wavelet Gaussian processes for surrogate modeling. The idea is based on the adaptation of the AR(1) model to high dimensional outputs. For this purpose, wavelet decomposition is used to concentrate the data in order to actively reduce the dimension through point selection. For this purpose, an object has been introduced which we have called the Gaussian wavelet process. The inversion of the wavelet decomposition allows us to obtain the quantification of the prediction

uncertainty. This chapter lays the foundation for Gaussian process regression on a wavelet basis. The main contribution is the modeling in a wavelet space with the calculation of the covariance function in a wavelet space for a Gaussian process of known covariance in the time space. The possibility of an extension to AR(1) multi-fidelity is discussed.

Finally, the chapter 6 presents a model using a dimension reduction to build a multi-fidelity surrogate model for time-series outputs. The first approach considered is to extend covariance tensorization to Gaussian process regression in a multi-fidelity framework. However, the section 6.1 discusses the problems of conditioning the covariance matrices which hinder the desired extension. The idea of the second part of this chapter is to use the singular value decomposition as a dimension reduction method and to use Gaussian process regression on the reduced dimensional outputs. To this is added a simple-fidelity regression on the function space not considered in the dimension reduction using Gaussian process surrogate modeling with tensorised covariance. This method is presented in section 6.3 and compared to other state-of-the-art methods in section 6.4.

Multi-fidelity surrogate model for high dimensional outputs remains a particularly complex problem. This manuscript focuses mainly on time-series outputs, but one can also imagine images or other high-dimensional objects. Moreover, only neural networks and Gaussian processes are presented for surrogate modeling, but there are many other methods such as random forests, presented in (Biau & Scornet, 2016), or chaos polynomes, see (Xiu & Karniadakis, 2002). Finally, in this manuscript, surrogate models are built for emulation performances, while for their use they could be associated to active learning, optimisation or sensitivity analysis methods for instance.

Part I

State of the Art

Chapter 1

Regression for uncertainty quantification

Numerical simulation has become an indispensable tool for bridging theory and experiments. Simulation relies on the use of complex and expensive computer codes, which take as input parameters describing the simulated phenomenon and return as output quantities characterizing it. The computation codes are based on increasingly complex mathematical equations; they have therefore become more accurate but also more expensive in terms of calculation time. As a result, the code can only be used to a limited extent. Many studies, such as optimization, uncertainty quantification, sensitivity analysis or rare event estimation, require a lot of simulations. To carry out these studies, it is necessary to approximate the output(s) of interest of the code by a low-cost function, i.e. a function that can be executed an almost unlimited number of times. These functions are commonly called surrogate models and learn the relationship between input parameters and output(s) from a limited number of simulations.

The mathematical formalization of the problem is given as follows. The function we search to approach by a surrogate model is called g :

$$\begin{aligned} \mathrm{g} \quad &: Q \subset \mathbb{R}^d \quad \to \mathbb{R}^q \\ &\mathbf{x} \qquad\quad \mapsto \mathrm{g}(\mathbf{x}) \end{aligned}$$

where Q is a non-empty set called the input parameters space.

We assume that we do not know the g function apart from the simulations we have done at specific inputs. So, we have a N-sized experimental design set noted $\mathcal{D} = \left\{\mathbf{x}^{(1)} =, \ldots, \mathbf{x}^{(N)}\right\}$ and the values of the function g at points in $\mathcal{D}$, noted $(\mathbf{z}^{(i)} = \mathrm{g}(\mathbf{x}^{(i)}))_{i=1,\ldots,N}$.

In addition to having to predict the computer code output, we are also interested in the accuracy of this prediction. In other words, what error is made when using the surrogate model instead of the code to predict a new observation? We will call this the uncertainty of prediction, and we will focus in this chapter on surrogate models that can quantify (give estimations of) their predictive uncertainties (without exhaustiveness). We choose to focus only on two types of methods: Gaussian process regression (GPR)

and neural networks (NN). Our choice is mainly motivated by the possible extension of these methods to a multi-fidelity framework introduced in chapter 3.

The section 1.1 recalls the basic properties of Gaussian processes by introducing the covariance function. Then, different families of covariance functions used for regression are presented. Then, we present Gaussian process regression and several criteria to evaluate prediction capacities. To conclude this section we present an extension of GPR dealing with time-series output.

The section 1.2 describes the neural network model and different ways to formulate it. We recall a classical learning approach to compute the model. As neural networks are not natively able to quantify the uncertainties associated to a prediction, we present several methods to address this issue.

1.1 Gaussian process regression (GPR)

The method that first comes to mind when looking to do regression is linear regression. Linear regression is indeed a regression method that can be adapted to many situations, see (Hastie, Tibshirani, & Friedman, 2009, sec. 3-4). However, the assumptions required to perform a linear regression are very strong and often difficult to verify in practice. Starting from linear regression, which is very easy to use, see (James, Witten, Hastie, & Tibshirani, 2021b), it has been possible to construct non-linear regressions. In particular there is a method that is widely used for uncertainty quantification: Gaussian process regression, otherwise known as Kriging. In this section, the basics of Gaussian process regression will be presented. GPR addresses the problem of modeling for a moderate number of input parameters. It also performs well in terms of quantifying uncertainty.

In this section we first present the tools needed to emulate computer codes. The tools are in link with stochastic processes and in particular Gaussian processes. A key element of this section is the presentation of the covariance kernels of the processes. Secondly, the Gaussian process regression model is presented. Gaussian process regression is highlighted in this section as it gives a prediction with quantified uncertainty.

1.1.1 Some useful elements for Gaussian process regression

Before presenting the regression method it is necessary to introduce some mathematical tools. As these tools are basics of probability and statistics, definitions and properties are proposed in the following. For more detail than the quick introduction that follows many books and courses are available, we refer to (Grimmett & Stirzaker, 2020; Karatzas & Shreve, 1998).

1.1.1.1 From random variable to random process

In order to predict a function, it is necessary to characterize it. Surrogate modeling can be seen as a completely deterministic point of view, but this is very limited, and the possible models are limited, see (Hastie et al., 2009, sec. 2.3). This is why we emulate a stochastic model from deterministic data.

Definition 1.1.1. *A probability space is a triple $(\Omega, \mathcal{F}, P)$ such that:*
- *Ω is the sample set.*
- *$\mathcal{F}$ is a σ-algebra. $\mathcal{F}$ contains the sample set $\Omega \in \mathcal{F}$. $\mathcal{F}$ is closed under complements and countable unions.*
- *P is the probability measure. P is σ-additive and $P(\Omega) = 1$.*

For a point to be predicted the value of the output is not known. We model our non-knowledge by a random variable which represents our prediction at this point.

Definition 1.1.2. *A random vector (or variable when $q = 1$) Y is a measurable function $Y : \Omega \to S \subset \mathbb{R}^q$ from the probability space $(\Omega, \mathcal{F}, P)$ to the measurable space $(S, \mathcal{B}(S))$ where S is a measurable subspace of $\mathbb{R}^q$ and $\mathcal{B}(S)$ is the Borel σ-algebra of S.*

Our model, which we seek to approximate, is a function from $\mathbb{R}^d$ to $\mathbb{R}^q$, we need to go further in the modeling.

Definition 1.1.3. *A $\mathbb{R}^q$-valued stochastic process Z on Q is an application, that associates a random vector $Z(\mathbf{x})$ to each $\mathbf{x} \in Q$. The random vectors $Z(\mathbf{x})$, for $\mathbf{x} \in Q$, are defined on a common probability space $(\Omega, \mathcal{F}, P)$.*

For each fixed $\omega \in \Omega$, the $\mathbb{R}^q$-valued function $\mathbf{x} \to Z(\omega, \mathbf{x})$ is called a realization or a trajectory of the random process Z.

1.1.1.2 Gaussian vectors

The objects presented above are useful for describing computer code, but they do not allow predictions to be made. Indeed, stochastic processes are very general and are not easily exploitable for prediction and uncertainty quantification. It is therefore necessary to add assumptions to our calculation code. However, we know a law that is commonly found in physics and other sciences: the normal or Gaussian distribution. This part presents the basic properties of the Gaussian distribution for a random vector.

Definition 1.1.4. *The normal distribution or Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \sigma^2)$ of mean $\boldsymbol{\mu}$ and variance $\sigma^2 > 0$ is a continuous probability distribution for a real-valued random variable. Its probability density function (PDF) is:*

$$f(\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(\mathbf{x} - \boldsymbol{\mu})^2}{2\sigma^2} \tag{1.1}$$

A random variable that is constant equal to $\boldsymbol{\mu}$ is said to be Gaussian with mean $\boldsymbol{\mu}$ and variance $\sigma^2 = 0$. The univariate Gaussian distribution, presented in definition 1.1.4 is generalized for vectors in the following definition.

Definition 1.1.5. *A n-dimensional random vector $X = (x_1, \cdots, x_n)$,$n \geq 1$, is a Gaussian vector with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ if and only if for any $a \in \mathbb{R}^n$, $a^T U$ is a Gaussian variable with mean $a^T \mu$ and variance $a^T \Sigma a$.*

It means that all linear combinations of Gaussian vectors are Gaussian variables. We can therefore note $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$.

We can retain that:

1. the components of a Gaussian vector X are independent if they are uncorrelated (Σ is diagonal),
2. if the components of a vector are Gaussian and independent then this vector is Gaussian,
3. the sum of two independent Gaussian vectors is a Gaussian vector.

In the case where X is a Gaussian vector, the condition of invertibility of its covariance matrix is sufficient to establish the existence of a density:

Theorem 1.1.6. *Let $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ be a Gaussian vector of size n with Σ invertible. The PDF f_X of X is:*

$$f_X(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \det \Sigma}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^t \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right). \tag{1.2}$$

We can also deduce that Gaussian vectors are stable by affine transformation.

Properties 1.1.7. *Let X be a Gaussian vector in $\mathbb{R}^n$, with mean $\boldsymbol{\mu}$ and covariance Σ. Let $A \in \mathcal{M}_{p,n}(\mathbb{R})$ and $b \in \mathbb{R}^p$. Then the random vector $AX + b$ is a Gaussian vector in $\mathbb{R}^p$. Its mean is $A\boldsymbol{\mu} + b$ and its covariance is $A\Sigma A^T$.*

The following theorem is essential because it is central to the Gaussian process regression. It is about the stability of Gaussian density distribution by conditioning.

Theorem 1.1.8 (Theorem of Gaussian conditioning)**.** *Let (Y, X) be a Gaussian vector with mean $(\boldsymbol{\mu}, \nu)$ and covariance $\mathrm{Cov}(Y, X) = \begin{pmatrix} W & C^T \\ C & V \end{pmatrix}$ with V invertible. The distribution of $Y|X = \mathbf{x}$ is Gaussian with mean $\boldsymbol{\mu}^\star = \boldsymbol{\mu} + C^T V^{-1}(\mathbf{x} - \nu)$ and covariance $W^\star = W - C^T V^{-1} C$.*

Corollary 1.1.8.1. *To generate realizations from a Gaussian vector of size n $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, we have:*

$$X = AZ + \boldsymbol{\mu}, \tag{1.3}$$

with A the square root matrix of Σ and Z a random vector with independent components $Z \sim \mathcal{N}(0, \mathbb{I}_n)$. It is then possible to generate realizations of X from realizations of Z.

1.1.1.3 Gaussian processes

Gaussian processes are the generalization of the Gaussian distribution from a random variable to a stochastic process.

Definition 1.1.9. *A stochastic process $Z(\mathbf{x})$ on $Q \subset \mathbb{R}^d$ valued in $\mathbb{R}$ is a Gaussian process when for all $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)} \in Q$, the random vector $(Z(\mathbf{x}^{(1)}), \ldots, Z(\mathbf{x}^{(n)}))$ is a Gaussian vector.*

A Gaussian process is characterized by its mean and covariance function. The mean function of a Gaussian process $Z(\mathbf{x})$ is the function :

$$\begin{aligned} m \quad &: Q \subset \mathbb{R}^d \quad \to \mathbb{R} \\ &\mathbf{x} \qquad \mapsto \mathbb{E}(Z(\mathbf{x})) \end{aligned}$$

The covariance function of a Gaussian process $Z(\mathbf{x})$ is the function:

$$\begin{aligned} k \quad &: Q \times Q \quad \to \mathbb{R} \\ &(\mathbf{x}, \mathbf{x}') \quad \mapsto \mathrm{Cov}\left[Z(\mathbf{x}), Z(\mathbf{x}')\right] \end{aligned}$$

The covariance function k must be positive semi-definite.

Definition 1.1.10. *An $n \times n$ symmetric real matrix M is said to be positive-definite if $\mathbf{x}^T M \mathbf{x} > 0$ for all non-zero $\mathbf{x}$ in $\mathbb{R}^n$.*

Definition 1.1.11. *An $n \times n$ symmetric real matrix M is said to be positive semi-definite if $\mathbf{x}^T M \mathbf{x} \geq 0$ for all non-zero $\mathbf{x}$ in $\mathbb{R}^n$.*

Definition 1.1.12. *We define the covariance matrix at the N points $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N)}\right\}$. The covariance matrix $\mathbf{K} \in \mathcal{M}_{N,N}(\mathbb{R})$ is the matrix defined by $K_{i,j} = k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$ with $(i,j) = 1, \cdots, N$, with $\mathbf{K} = \{K_{i,j}\}_{i,j}$.*

Definition 1.1.13. *A symmetric kernel $k : Q \times Q \to \mathbb{R}$ is positive semi-definite on Q if:*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \geq 0 \tag{1.4}$$

for any $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(n)} \in Q$, $c_1, \cdots, c_n \in \mathbb{R}$, given $n \in \mathbb{N}$.

The following theorem shows that to create a Gaussian process, it is sufficient to create a mean function and a covariance function.

Theorem 1.1.14. *Let Q be any set, m any function from Q to $\mathbb{R}$ and k any symmetric positive semi-definite function from $Q \times Q$ to $\mathbb{R}$. Then there exists a Gaussian process $Z(\mathbf{x})$ on Q with mean function and covariance function k.*

The covariance kernel describes relations between $Z(\mathbf{x})$ and $Z(\mathbf{x}')$. For examples, let $k_1(\mathbf{x}, \mathbf{x}') = 1$, a Gaussian process $Z(\mathbf{x})$ with mean zero and covariance function k_1 is constant (i.e., $P(Z(\mathbf{x}) = Z(\mathbf{x}_0)) = 1$ for all, $\mathbf{x}, \mathbf{x}_0 \in Q$). Let $k_2(\mathbf{x}, \mathbf{x}') = \mathbf{1}_{\{\mathbf{x}=\mathbf{x}'\}}$, a Gaussian process $Z(\mathbf{x})$ with mean zero and covariance function k_2 is composed of independent Gaussian values.

Stationary "Kernel" covariance function A GP is said to be stationary when its covariance function is stationary.

Definition 1.1.15. *A covariance function k is stationary when for any $\mathbf{x}, \mathbf{x}\prime \in Q$, k is a function of τ, with $\tau = \mathbf{x} - \mathbf{x}'$.*

In this paragraph we are interested in stationary covariance kernels, and we use the term stationary Gaussian process when such functions allow to define the GP. These kernels are used in GP regression to emulate computer codes.

It exists a property of stationary covariance kernel that says that they can be represented as the Fourier transform of a positive measure. The definition of the Fourier transform used in the following is given with all properties in section 2.2.1. The following theorem is given in (Rasmussen & Williams, 2006, sec. 4.2.1) and allows to have convenient characterization of stationary covariance functions with Fourier transform of k.

Theorem 1.1.16. *(Bochner's theorem) For any continuous positive definite function $k(\tau)$ from $\mathbb{R}^d$ into $\mathbb{R}$, there exists a unique positive measure μ on $\mathbb{R}^d$ such that:*

$$k(\tau) = \frac{1}{(2\pi)^d} \int_{\mathbb{R}^d} e^{i<\tau,\xi>} \mathrm{d}\mu(\xi), \tag{1.5}$$

with $< \cdot, \cdot >$ the scalar product.

Proof. A proof can be found in (Gihman & Skorokhod, 2004, pp. 208-212). The expression of the theorem we use is from (Stein, 1999). □

In the case where $\mathrm{d}\mu(\xi)$ has a density we can introduce the spectral density $S(\xi)$ (or power spectrum) of the covariance kernel $k(\tau)$. It can be useful for the representation of GP in Fourier domain, see chapter 5.

Theorem 1.1.17. *(Wiener-Khintchine theorem) In the case that the spectral density $S(\xi)$ exists, the covariance function and the spectral density are Fourier duals of each other:*

$$k(\tau) = \frac{1}{(2\pi)^d} \int S(\xi) e^{i<\tau,\xi>} \mathrm{d}\xi, \tag{1.6}$$

$$S(\xi) = \int k(\tau) e^{-i<\tau,\xi>} \mathrm{d}\tau. \tag{1.7}$$

The spectral density must be integrable because $k(0) = \frac{1}{(2\pi)^d} \int S(\xi) \mathrm{d}\xi$ and $S(\xi) \geq 0$ for all ξ.

1.1.1.4 Examples of stationary covariance kernel on $\mathbb{R}$

The following paragraphs are a list of standard stationary covariance kernels. For a more complete list, see (Rasmussen & Williams, 2006, sec. 4.2; Stein, 1999; K. Tiwari & Chong, 2019, ch. 6).

Gaussian kernel

Definition 1.1.18. *The Gaussian or squared exponential covariance function is defined as:*

$$k(\tau) = \exp\left(-\frac{|\tau|^2}{2\theta^2}\right), \tag{1.8}$$

where θ is the correlation length. The spectral density of the Gaussian covariance function is:

$$S(\xi) = \sqrt{2\pi}\theta \exp\left(-\frac{1}{2}\theta^2|\xi|^2\right). \tag{1.9}$$

Gaussian processes associated with Gaussian kernel are infinitely differentiable almost surely, thanks to Kolmogorov-Chentsov theorem.

ν-Matérn kernel

Definition 1.1.19. *Matérn class of covariance function is defined as:*

$$k(\tau) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}|\tau|}{\ell}\right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}|\tau|}{\ell}\right), \tag{1.10}$$

where K_ν is a modified Bessel function (see (Olver, Lozier, Boisvert, & Clark, 2010)), Γ is the Gamma function, ν is a positive parameter and ℓ is the correlation length. The spectral density of this covariance function is:

$$S_\nu(\xi) = \frac{2\pi^{\frac{1}{2}}\Gamma\left(\nu+\frac{1}{2}\right)(2\nu)^\nu}{\Gamma(\nu)\ell^{2\nu}} \left(\frac{2\nu}{\ell^2} + \xi^2\right)^{-\left(\nu+\frac{1}{2}\right)}. \tag{1.11}$$

A Gaussian process $Z(x)$ with a ν-Matérn covariance kernel is ν-Hölder continuous in mean square. This expression being particularly complex, we limit ourselves to a reduced number of ν.

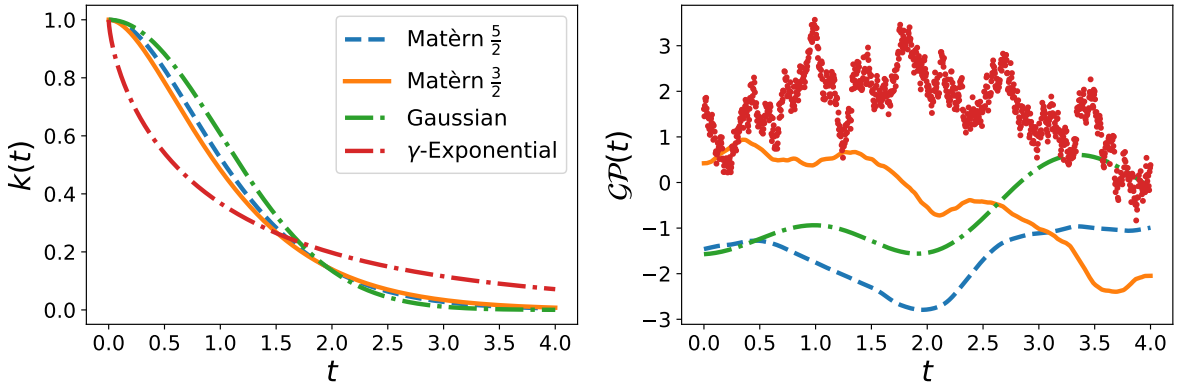


Figure 1.1: Comparison of the stationary kernels. Both pictures have the same legend. On the right there is a realization of GP with the covariance on the left. It can be seen that the more the covariance is peaked in 0 the rougher the realization. For this figure $\gamma = 0.8$.

Remark 1.1.20. *The most common ν values give:*

- $\nu = \frac{1}{2}$*, then the expression is equivalent to exponential kernel*

$$k(\tau) = \exp\left(-\frac{|\tau|}{\ell}\right), \quad S_{\frac{1}{2}}(\xi) = \frac{2}{\ell}\left(\frac{1}{\ell^2} + \xi^2\right)^{-1}. \tag{1.12}$$

- $\nu = \frac{3}{2}$*, the simplification gives:*

$$k(\tau) = \left(1 + \frac{\sqrt{3}|\tau|}{\ell}\right)\exp\left(-\frac{\sqrt{3}|\tau|}{\ell}\right), \quad S_{\frac{3}{2}}(\xi) = \frac{12\sqrt{3}}{\ell^3}\left(\frac{3}{\ell^2} + \xi^2\right)^{-2}. \tag{1.13}$$

- $\nu = \frac{5}{2}$*, the simplification gives:*

$$k(\tau) = \left(1 + \frac{\sqrt{5}|\tau|}{\ell} + \frac{5|\tau|^2}{3\ell^2}\right)\exp\left(-\frac{\sqrt{5}|\tau|}{\ell}\right), \quad S_{\frac{5}{2}}(\xi) = \frac{400\sqrt{5}}{\ell^5}\left(\frac{3}{\ell^2} + \xi^2\right)^{-3}. \tag{1.14}$$

The Matèrn-$\frac{5}{2}$ is the most used for computer code uncertainty quantification.

γ-exponential kernel The γ-exponential kernel is defined as:

$$k(\tau) = \exp\left(-\left(\frac{|\tau|}{\ell}\right)^{\gamma}\right), \quad 0 < \gamma \leq 2. \tag{1.15}$$

Comparison of the kernels The covariance kernels are associated with GP. Figure 1.1 shows the differences between kernels and associated GP.

1.1.1.5 Stationary covariance functions on $\mathbb{R}^d$

From covariance functions defined of $\mathbb{R}$ it is possible to construct covariance functions defined on $\mathbb{R}^d$. Three methods are presented below.

Tensorized covariance function It is possible to build a kernel on $\mathbb{R}^d$ by tensorization of one-dimensional kernels. Let us assume that, for $i = 1, \cdots, d$, k_i is a one-dimensional stationary kernel. Then:

$$k(\mathbf{x} - \mathbf{x}') = \prod_{i=1}^{d} k_i(x_i - x_i'), \tag{1.16}$$

is a stationary kernel on $\mathbb{R}^d$. For example the tensorized square exponential covariance kernel on $\mathbb{R}^d$ is:

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \prod_{i=1}^{d} \exp\left(-\frac{(x_i - x_i')^2}{\ell_i^2}\right). \tag{1.17}$$

Isotropic covariance function Another method to build stationary kernels on $\mathbb{R}^d$ is to apply a one-dimensional stationary kernel k_1 to the Euclidean norm on $\mathbb{R}^d$. If k_1 is a one-dimensional stationary kernel, then

$$k(\mathbf{x} - \mathbf{x}') = k_1(|\mathbf{x} - \mathbf{x}'|), \tag{1.18}$$

is a stationary kernel on $\mathbb{R}^d$. For example the isotropic square exponential covariance kernel on $\mathbb{R}^d$ is:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_1^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\ell_1^2}\right).$$

Geometric covariance function The previous method can be extended by using a weighted l^2-norm in $\mathbb{R}^d$. If k_1 is a one-dimensional stationary kernel, then:

$$k(\mathbf{x} - \mathbf{x}') = k_1\left(\sqrt{\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{\ell_i^2}}\right), \tag{1.19}$$

is a stationary kernel on $\mathbb{R}^d$. For example the geometric square exponential covariance kernel on $\mathbb{R}^d$ is:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_1^2 \exp\left(-\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{\ell_i^2}\right),$$

with $\ell = (\ell_1, \cdots, \ell_d)$, which is equivalent to eq. (1.17).

1.1.2 Gaussian process modeling

The approach comes from (Krige, 1951). The regression by Gaussian process in the current formalism is defined in (Rasmussen & Williams, 2006). There are many recent extensions, for example in (Gu & Berger, 2016; Rullière, Durrande, Bachoc, & Chevalier, 2018) Gaussian process regression is adapted for a large number of data.

1.1.2.1 Gaussian process prediction

In this section, we seek to approximate a scalar output of a computational code g. The function g we wish to learn is deterministic (noisy free case) and defined on $Q \subset \mathbb{R}^d$ into $\mathbb{R}$. Let us assume that the emulator is a Gaussian process $Z(\mathbf{x})$ with mean function $\mu(\mathbf{x})$ and with covariance function $k(\mathbf{x}, \mathbf{x}')$. We want to predict the value of $\mathrm{g}(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^d$ from a few observations of g at points in $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$. We introduce the vector $\mathbf{Z} = (Z^{(1)}, \ldots, Z^{(N)})$, it is the Gaussian vector observed in the learning set.

A common choice for the mean $\mu(\mathbf{x})$ is to take a linear form : $\mu(\mathbf{x}) = f^T(\mathbf{x})\beta$, where $f(\mathbf{x})$ is a column vector of p known functions and β the corresponding coefficients. In the following, we introduce the correlation function $r(\mathbf{x}, \mathbf{x}')$ such that $k(\mathbf{x}, \mathbf{x}'; \sigma, \ell) = \sigma^2 r(\mathbf{x}, \mathbf{x}'; \ell)$, where ℓ and σ are correlation length and standard deviation of the function k. The kernel is supposed to be in $\mathbb{R}^d$ so that the correlation length is the vector ℓ.

If we know the mean and the covariance functions, we can express the joint distribution:

$$\begin{pmatrix} Z(\mathbf{x}) \\ \mathbf{Z} \end{pmatrix} \sim \mathcal{N}\left(\left(\begin{pmatrix} f^T(\mathbf{x})\beta \\ \mathbf{F}\beta \end{pmatrix}, \sigma^2 \begin{pmatrix} 1 & \mathbf{r}^T(\mathbf{x}) \\ \mathbf{r}(\mathbf{x}) & \mathbf{R} \end{pmatrix}\right)\right), \tag{1.20}$$

where $\mathbf{F} = \begin{pmatrix} f_1(\mathbf{x}^{(1)}) & \cdots & f_p(\mathbf{x}^{(1)}) \\ \vdots & & \vdots \\ f_1(\mathbf{x}^{(N)}) & \cdots & f_p(\mathbf{x}^{(N)}) \end{pmatrix}$ is the matrix of regressors, $\mathbf{r}(\mathbf{x}) = \left[r(\mathbf{x}, \mathbf{x}^{(i)})\right]_{i=1,\cdots,N}$ is the correlation vector between $Z(\mathbf{x})$ and the observation points. The correlation matrix between the observation points is $\mathbf{R} = \left[r(\mathbf{x}^{(j)}, \mathbf{x}^{(i)})\right]_{i,j=1,\cdots,N}$.

We can therefore predict the probability distribution of Z conditioned by the data and the hyperparameters: $Z(\mathbf{x})|\mathbf{Z} = \mathbf{z}^N, \beta, \sigma, \ell$, with $\mathbf{z}^N = \left\{\mathrm{g}(\mathbf{x}^{(1)}), \cdots, \mathrm{g}(\mathbf{x}^{(N)})\right\}$. Because the vector $(Z(\mathbf{x}), \mathbf{Z})$ is Gaussian conditionally to the hyperparameters we can use the theorem of Gaussian conditioning. Then, the distribution $Z(\mathbf{x})|\mathbf{Z} = \mathbf{z}^N, \beta, \sigma, \ell$ is Gaussian with mean $\mu^\star(\mathbf{x})$ and variance $v^\star(\mathbf{x})$:

$$\mu^\star(\mathbf{x}) = f^T(\mathbf{x})\beta + \mathbf{r}^T(\mathbf{x})\mathbf{R}^{-1}\left(\mathbf{z}^N - \mathbf{F}\beta\right), \tag{1.21}$$

$$v^\star(\mathbf{x}) = \sigma^2\left(1 - \mathbf{r}^T(\mathbf{x})\mathbf{R}^{-1}\mathbf{r}(\mathbf{x})\right). \tag{1.22}$$

This model is called the simple Kriging.

Thanks to the Gaussian property of the posterior distribution and section 1.1.2.1, we can define the prediction interval at confidence level α:

$$I_\alpha(\mathbf{x}) = \left[\mu^\star(\mathbf{x}) - C(\alpha)v^\star(\mathbf{x})^{\frac{1}{2}}, \mu^\star(\mathbf{x}) + C(\alpha)v^\star(\mathbf{x})^{\frac{1}{2}}\right], \tag{1.23}$$

with $C(\alpha)$ the $1 - \frac{\alpha}{2}$ quantile of the standard Gaussian distribution. For $\alpha = 95\%$ we have $C(95\%) \approx 1.96$.

Remark 1.1.21.

- *The Kriging mean interpolates the observation points. Also, the variances at observation points are null. This important property of Kriging means that the model is interpolating at observation points.*
- *The Kriging mean does not depend on the standard deviation σ.*
- *The Kriging variance at a point is independent of the observed values. This makes it possible to predict the reduction in variance when adding a new observation point even before having the value of the observation. This is useful for active learning.*
- *In real cases it can be of interest to product non-interpolating models. In particular for noisy measurements, it is possible to add a 'nugget' effect so that the model is not interpolating. This can be useful for optimization, see (Picheny & Ginsbourger, 2014).*

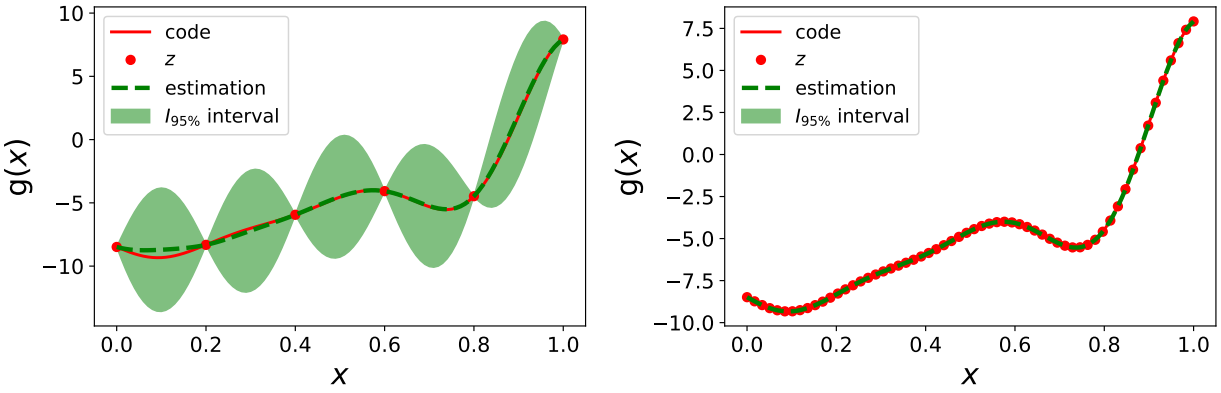
There are other ways to express the Kriging problem. One way is to express it as the best linear unbiased predictor (BLUP). It is also possible to express the prediction mean as the sum of kernels centred at the observation point. More details can be found in (Robinson, 1991). This form is the solution of the regularization problem on a reproducing kernel Hilbert space (RKHS). The first elements are proposed in (Kimeldorf & Wahba, 1971). The whole framework is described in (Rasmussen & Williams, 2006).

Example in 1d We assume that our computer code is the function:

$$\mathrm{g}(x) = 0.5(6x-2)^2\sin(12x-4) + 10(x-0.5) - 5. \tag{1.24}$$

The experimental design is $\mathcal{D} = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. A second learning set with more data ($\mathrm{card}(\mathcal{D}) = 60$) on a regular grid is also proposed.

We can see the interpolative nature of the Gaussian process regression. The result is presented in fig. 1.2. The prediction intervals are larger when fewer points are known. The shape of the prediction intervals is typical of Gaussian process regression. We can therefore see that this method can be very satisfactory for fairly regular functions and when we have few observations of them. The prediction interval is presented in eq. (1.23). The interval is larger when the data are far from the training points and the interval is smaller when there are more points. The model is very accurate, and the prediction interval is representative of the information we have about the function.



a) 6 points z b) 60 points z

Figure 1.2: Two regression problems solved with GP regression. The proposed covariance kernel is Matèrn $5/2$. The regression is performed for two training sets. It can be seen that the prediction performance and the uncertainty quantification (in the form of prediction intervals) improve with the increase of the number of training points.

1.1.2.2 Parameters estimation

In section 1.1.2.1, we assumed that the mean and covariance functions were known. However, although we can indeed know their parametric form, it is necessary to estimate their parameters. For the linear mean function, the β coefficients must be estimated and for the covariance function, the σ^2 variance and the correlation lengths ℓ must also be estimated.

Bayesian Estimation The parameters to estimate are (β, σ^2). With this method we consider ℓ fixed. A Bayesian estimation of ℓ can be proceeded as presented in (Le Gratiet, 2013b, sec. 1.3.1).

Definition 1.1.22. *The prior distributions of the hyperparameters* (β, σ^2) *are:*

$$\sigma^2 \sim \mathcal{IG}\left(\alpha, \gamma\right), \tag{1.25}$$

$$\beta|\sigma^2 \sim \mathcal{N}\left(b_0, \sigma^2 V_0\right), \tag{1.26}$$

with b_0 *a vector,* V_0 *a covariance matrix and* $\mathcal{IG}(\alpha, \gamma)$ *stands for the inverse gamma distribution with density function:*

$$p(\mathbf{x}) = \frac{\gamma^\alpha e^{-\gamma/\mathbf{x}}}{\mathbf{\Gamma}(\alpha)\mathbf{x}^{\alpha+1}}\mathbf{1}_{\mathbf{x}>0}. \tag{1.27}$$

This form of priors distributions allows obtaining a closed form expression for the predictive distribution. These priors are conjugate priors.

Property 1.1.23. *Parameter posterior distributions The observed data are* $\mathbf{z}^N$ *a vector containing the observed output values. The likelihood of* $\mathbf{z}^N$ *knowing* β *and* σ^2 *can be expressed, thanks to the multivariate*

normal assumption:

$$p(\mathbf{z}^N|\beta,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{N}{2}}\sqrt{|\mathbf{R}|}}\exp\left(-\frac{(\mathbf{z}^N-\mathbf{F}\beta)^T\mathbf{R}^{-1}(\mathbf{z}^N-\mathbf{F}\beta)}{2\sigma^2}\right) \tag{1.28}$$

with $|\mathbf{R}|$ *the determinant of the matrix* $\mathbf{R}$, $\mathbf{F} = \begin{pmatrix} f_1(\mathbf{x}^{(1)}) & \cdots & f_p(\mathbf{x}^{(1)}) \\ \vdots & & \vdots \\ f_1(\mathbf{x}^{(N)}) & \cdots & f_p(\mathbf{x}^{(N)}) \end{pmatrix}$ *is the matrix of regressors,* $\mathbf{r}(\mathbf{x}) = \left[r(\mathbf{x},\mathbf{x}^{(i)})\right]_{i=1,\cdots,N}$ *is the correlation between* $Z(\mathbf{x})$ *and the vector* $\left(Z(\mathbf{x}^{(i)})\right)_{i=1}^N$*. The correlation matrix of this vector is* $\mathbf{R} = \left[r(\mathbf{x}^{(j)},\mathbf{x}^{(i)})\right]_{i,j=1,\cdots,N}$*. The Bayes rule gives us that* $\beta|\mathbf{z}^N,\sigma^2 \sim \mathcal{N}(\mathbf{A}\theta,\mathbf{A})$*, with:*

$$\mathbf{A}^{-1} = \left[\mathbf{F}^T\mathbf{R}^{-1}\mathbf{F} + V_0^{-1}\right]\frac{1}{\sigma^2}$$

and

$$\theta = \left[\mathbf{F}^T\mathbf{R}^{-1}\mathbf{z}^N + V_0^{-1}b_0\right]\frac{1}{\sigma^2}.$$

The Bayes rule gives $\sigma^2|\mathbf{z}^N \sim \mathcal{IG}\left(\frac{N}{2}+\alpha, Q_\sigma\right)$*, with:*

$$Q_\sigma = 2\gamma + (b_0-\hat{\beta})^T(V_0 + \left[\mathbf{F}^T\mathbf{R}^{-1}\mathbf{F}\right]^{-1})(b_0-\hat{\beta}) + \tilde{Q_\sigma}, \tag{1.29}$$

with $\hat{\beta} = \left(\mathbf{F}^T\mathbf{R}^{-1}\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{R}^{-1}\mathbf{z}^N$ *the generalized least square estimator and*

$$\tilde{Q_\sigma} = (\mathbf{z}^N)^T\left[\mathbf{R}^{-1} - \mathbf{R}^{-1}\mathbf{F}(\mathbf{F}^T\mathbf{R}^{-1}\mathbf{F})^{-1}\mathbf{F}^T\mathbf{R}^{-1}\right]\mathbf{z}^N.$$

Details on the previous property can be found in (Santner, Williams, & Notz, 2003).

It should be noted that we can introduce priors that reflect the fact that we have no information a priori. These priors are called non-informative priors. Jeffreys' priors are an example of objective prior. There are other types of priors such as reference priors, see (Berger, Bernardo, & Sun, 2009). Non-informative priors are usually improper distributions obtained from Fisher information matrices, see (Fisher, 1955). For more details on non-informative priors see (Gu, Wang, & Berger, 2018, 6A).

Maximum likelihood estimation In this paragraph we seek to estimate β,σ and ℓ. One of the most popular methods for estimating parameters is maximum likelihood. This approach has the advantage of being very fast in computation time. As with Bayesian estimation, the starting point is the observations $\mathbf{z}^N$. Thanks to the multivariate Gaussian assumption we have:

$$p(\mathbf{z}^N|\beta,\sigma^2,\ell) = \frac{1}{(2\pi\sigma^2)^{\frac{N}{2}}\sqrt{|\mathbf{R}_\ell|}}\exp\left(-\frac{(\mathbf{z}^N-\mathbf{F}\beta)^T\mathbf{R}_\ell{}^{-1}(\mathbf{z}^N-\mathbf{F}\beta)}{2\sigma^2}\right), \tag{1.30}$$

with $\mathbf{R}_\ell$ the correlation matrix depending on ℓ. The Maximum Likelihood Estimator (MLE) of β conditionally to σ, l and $\mathbf{z}^N$ is:

$$\hat{\beta} = \left(\mathbf{F}^T {\mathbf{R}_\ell}^{-1} \mathbf{F}\right)^{-1} \mathbf{F}^T {\mathbf{R}_\ell}^{-1} \mathbf{z}^N. \tag{1.31}$$

This is the generalized least squares estimator of β. By substitution of β by $\hat{\beta}$ in the likelihood we get the optimum with respect to σ^2:

$$\hat{\sigma^2} = \frac{(\mathbf{z}^N - \mathbf{F}\hat{\beta})^T {\mathbf{R}_\ell}^{-1} (\mathbf{z}^N - \mathbf{F}\hat{\beta})}{N}. \tag{1.32}$$

The MLE of ℓ is obtained by minimizing the log-likelihood:

$$\mathcal{L}(\ell, \mathbf{z}^N) = N \log(\hat{\sigma}^2) + \log(|\mathbf{R}_\ell|). \tag{1.33}$$

This MLE gives estimation of the parameters. This marginal likelihood has to be numerically minimized in ℓ in order to estimate the best value of ℓ.

More details can be found in (Bachoc, 2013) and (Santner et al., 2003).

1.1.3 Tensorized non-stationary covariance for time-series output

This subsection is highlighting a method that can be seen as high-dimensional Gaussian process regression. This methods is dealing with time-series output. As the problem is slightly different, we start by defining it. The code is the function z depending on two variables, $\mathbf{x}$ and t. For each point $\mathbf{x}$ we have the time-series $z(\mathbf{x}, \cdot)$. The goal is to emulate the code using GP regression.

For the calculation of surrogate models with functional outputs, there are two different techniques. The first is dimension reduction method as presented in chapter 2. An alternative is presented here, the second method is GP regression with covariance tensorization. The method is presented in (Rougier, 2008) and the estimation of the hyper-parameters is from (Perrin, 2020).

In this subsection we consider that the output is a time-dependent function observed on a fixed time grid $\{t_u\}_{u=1,\cdots,N_t}$, with $N_t \gg 1$, which is called a time-series.

The experimental design in a times-series output case is very different from a scalar output case. In particular, for a value $\mathbf{x}$ in the experimental design D, all the t of the time grid are in the experimental design. The $N_t \times N_x$ matrix containing the observations is $\mathbf{Z}_{\text{obs}} = \left(z(\mathbf{x}^{(i)}, t_u)\right)_{\substack{u=1,\ldots,N_t \\ i=1,\ldots,N_x}}$. In GP regression we model the prior knowledge of the code output as a Gaussian process $Z(\mathbf{x}, t_u)$ with $\mathbf{x} \in Q$ and $u = 1, \ldots, N_t$ with a covariance function C given by eq. (1.35) and a mean function μ given by eq. (1.34). We focus our attention to the case $N_t > N_x$. We assume that the covariance structure can be decomposed into two different functions representing the correlation in $\mathbf{x}$ and the correlation in t. If we choose well both functions, the Kriging calculation is possible (Perrin, 2020; Rougier, 2008).

In the following we present a simplification of the method proposed in (Perrin, 2020). The a priori

$\mathbb{R}^{N_t}$-valued mean function is assumed to be of the form:

$$\mu(\mathbf{x}) = Bf(\mathbf{x}) \tag{1.34}$$

where $f(\mathbf{x})$ is a given $\mathbb{R}^M$-valued function and $B \in \mathcal{M}_{N_t \times M}(\mathbb{R})$ is to be estimated. We define by F the $N_x \times M$ matrix $[f^T(\mathbf{x}^{(i)})]_{i=1,\dots,N_x}$.

The a priori covariance function $C(t_u, t_{u'}, \mathbf{x}, \mathbf{x}')$ can be expressed with the $N_t \times N_t$ matrix R_t and the correlation function $C_x : Q \times Q \to [0,1]$ with $C_x(\mathbf{x}, \mathbf{x}) = 1$:

$$C(t_u, t_{u'}, \mathbf{x}, \mathbf{x}') = R_t(t_u, t_{u'}) C_x(\mathbf{x}, \mathbf{x}'). \tag{1.35}$$

The covariance in time is expressed as a matrix because the temporal grid is finite and fixed. The covariance "matrix" (here a tensor) of $\left(Z(\mathbf{x}^{(j)}, t_u)\right)_{\substack{u=1,\dots,N_t \\ j=1,\dots,N_x}}$ is

$$R = R_t \otimes R_x, \tag{1.36}$$

with $(R_x)_{k,l} = C_x(\mathbf{x}^{(k)}, \mathbf{x}^{(l)})$ $k, l = 1, \dots, N_x$.

If R_x and R_t are not singular, then the a posteriori distribution of the $\mathbb{R}^{N_t}$-valued process Z given the covariance functions and the observations $\mathbf{Z}_{\text{obs}}$ is Gaussian:

$$(Z(\mathbf{x}, t_u))_{u=1,\dots,N_t} | R_t, C_x, \mathbf{Z}_{\text{obs}} \sim \mathcal{GP}(\mu_\star(\mathbf{x}), R_\star(\mathbf{x}, \mathbf{x}') R_t), \tag{1.37}$$

with the N_t-dimensional posterior mean:

$$\mu_\star(\mathbf{x}) = \mathbf{Z}_{\text{obs}} R_x^{-1} r_x(\mathbf{x}) + B_\star u(\mathbf{x}) \tag{1.38}$$

where $r_x(\mathbf{x})$ is the N_x-dimensional vector $\left(C_x(\mathbf{x}, \mathbf{x}^{(j)})\right)_{j=1,\dots,N_x}$. The posterior covariance function $R_\star(\mathbf{x}, \mathbf{x}')$ is :

$$R_\star(\mathbf{x}, \mathbf{x}') = c_\star(\mathbf{x}, \mathbf{x}') \left(1 + v_\star(\mathbf{x}, \mathbf{x}')\right). \tag{1.39}$$

The functions that are used in the regression are:

$$\begin{cases} u(\mathbf{x}) = f(\mathbf{x}) - F^T R_x^{-1} r_x(\mathbf{x}) \\ c_\star(\mathbf{x}, \mathbf{x}') = C_x(\mathbf{x}, \mathbf{x}') - r_x(\mathbf{x})^T R_x^{-1} r_x(\mathbf{x}') \\ v_\star(\mathbf{x}, \mathbf{x}') = u(\mathbf{x})^T (F^T R_x^{-1} F)^{-1} u(\mathbf{x}') c_\star^{-1}(\mathbf{x}, \mathbf{x}') \end{cases}, \tag{1.40}$$

and

$$B_\star = \mathbf{Z}_{\text{obs}} R_x^{-1} F (F^T R_x^{-1} F)^{-1}. \tag{1.41}$$

The correlation function C_x is assumed to be a Matèrn $\frac{5}{2}$ kernel with a tensorized form, see (Rasmussen & Williams, 2006, ch. 4):

$$C_x(\mathbf{x}, \mathbf{x}') = \prod_{i=1}^{d} \left(1 + \frac{\sqrt{5}|x_i - x_i'|}{\ell_{x_i}} + \frac{5|x_i - x_i'|^2}{3\ell_{x_i}^2}\right) \exp\left(-\frac{\sqrt{5}|x_i - x_i'|}{\ell_{x_i}}\right), \tag{1.42}$$

with $\ell_{\mathbf{x}} = (\ell_{x_1}, \ldots, \ell_{x_d})$ the vector of correlation lengths. Other choices are of course possible. R_t is estimated using R_x^{-1} and the observations $\mathbf{Z}_{\text{obs}}$ by maximum likelihood, as in (Perrin, 2020):

$$\widehat{R_t} = \frac{1}{N_x}\left(\mathbf{Z}_{\text{obs}} - \hat{\mathbf{Z}}\right) R_x^{-1} \left(\mathbf{Z}_{\text{obs}} - \hat{\mathbf{Z}}\right)^T, \tag{1.43}$$

with $\hat{\mathbf{Z}}$ being the $N_t \times N_x$ matrix of empirical means $\hat{Z}_{u,i} = \frac{1}{N_x}\sum_{j=1}^{N_x} (\mathbf{Z}_{\text{obs}})_{u,j}$, $\forall i = 1, \ldots, N_x$ and $u = 1, \ldots, N_t$.

It remains only to estimate the vector of correlation lengths $\ell_{\mathbf{x}} = (\ell_{x_1}, \ldots, \ell_{x_d})$ to determine the function C_x. As presented in (Perrin, 2020), the maximum likelihood estimation is not well-defined for $\ell_{\mathbf{x}}$. Indeed, the estimation of R_t by eq. (1.43) is singular because $N_x < N_t$. In fact, we do not need to invert R_t as seen in Equations (1.37) to (1.41). The method generally used to estimate the correlation lengths is cross-validation and, in our case, Leave-One-Out (LOO). LOO consists in building a surrogate model with all the data except one point, given the values of the correlation lengths. We compare the prediction of the surrogate model with the data at this point, which gives us an error. We build as many surrogate models as there are points, and we can therefore evaluate the quality of our surrogate model as a function of the values of the correlation lengths. The LOO mean square error that needs to be minimized is:

$$\varepsilon^2(\ell_{\mathbf{x}}) = \sum_{k=1}^{N_x} \|\mu_\star^{(-k)}(\mathbf{x}^{(k)}|\mathbf{Z}_{\text{obs}}^{(-k)}, \ell_{\mathbf{x}}) - \mathbf{Z}_{\text{obs}}(\mathbf{x}^{(k)})\|^2, \tag{1.44}$$

where $\mu_\star^{(-k)}(\mathbf{x}^{(k)}|\mathbf{Z}_{\text{obs}}^{(-k)}, \ell_{\mathbf{x}})$ is the $\mathbb{R}^{N_t}$-valued prediction mean obtained with the correlation length vector $\ell_{\mathbf{x}}$, using all observations except the k-th, which is $\left(z(\mathbf{x}^{(k)}, t_u)\right)_{u=1}^{N_t}$ (note that we remove a whole time series from the data in this special LOO procedure), at the point $\mathbf{x}^{(k)}$ and $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^{Nt}$. The LOO learning set is $\mathbf{Z}_{\text{obs}}^{(-k)} = \left(z(\mathbf{x}^{(i)}, t_u)\right)_{\substack{u=1,\ldots,N_t \\ i=1,\ldots,N_x; i\neq k}}$ and $\mathbf{Z}_{\text{obs}}(\mathbf{x}^{(k)}) = \left(z(\mathbf{x}^{(k)}, t_u)\right)_{u=1,\ldots,N_t}$. We can use an expression of $\varepsilon^2(\ell_{\mathbf{x}})$ that does not require multiple regression, as in (Bachoc, 2013; Dubrule, 1983). For more detail see appendix A.

1.2 Uncertainty quantification with Neural Network

The section describes the neural network model and different ways to formulate it. We recall a classical learning approach to compute the model. As neural networks are not natively able to predict uncertainties

associated to a prediction, we present several methods to address this issue.

A neural network (NN) is a surrogate model that links linear regression by an activation function. The composition of a linear regression and an activation function is called a neuron. Neurons are organized in layers, as presented in fig. 1.3.

The name "neural network" came from thinking of these different layers operations as analogous to neurons in brains. And in particular activation is seen as a trigger that is also present in neurons. However, the more complex the methods become, the further away the NN are from brain models.

Thanks to the impressive achievements of the NN, see (LeCun et al., 1989) for example, it has been possible to construct uncertainty quantification methods from these models. (Kabir, Khosravi, Hosen, & Nahavandi, 2018) provides an overview of existing methods for uncertainty quantification with neural networks. As NN surrogate models have been constructed to be accurate on high dimension, (Tripathy & Bilionis, 2018) shows an example of surrogate model for high dimension with NN. In (Pasini, 2015) the author shows that NN can be used in a small data context but also for critical applications. This has also been proposed for more complex models as in (Liu & Deng, 2015).

Parallels have also been drawn with other regression methods. The link between NN and linear regression is more than obvious as NN are composed of elements related to linear regression. (Cohen, Sharir, & Shashua, 2016; Michel & Nouy, 2022) shows the link between NN and tree tensor format. In (Williams, 1996) it is shown that infinitely wide NN with distributions placed over their weights converge to Gaussian process. This allows us to be confident about the ability of NN to take uncertainty into account.

1.2.1 Definition for NN surrogate models

In this section we will use the formalism given in (James, Witten, Hastie, & Tibshirani, 2021a). This is a simplified version of (Hastie et al., 2009). In this section the NN emulates an unknown function g with input $\mathbf{x} \in \mathbb{R}^d$ and output $\mathbf{y} \in \mathbb{R}^q$. A neural network takes an input vector $\mathbf{x}$ and builds a nonlinear approximation NN of g with the help of data. We called $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N)}\right\}$ the learning set and the data set consists of the observations $\left\{\mathrm{g}(\mathbf{x}^{(1)}), \cdots, \mathrm{g}(\mathbf{x}^{(N)})\right\}$. The specificity of the NN compared to the GP is its structure. The structure of the NN is composed of many simple linear regressions and then passes into an activation function which introduces a non-linearity. The element called input layer consists in decomposing the input vector into a number of scalars corresponding to the input dimension. Then comes the hidden layer which takes the output of the previous layer as input and builds a linear regression and then activates it using the activation function. A set of regressions and activations is called a neuron (or node). When the output of a hidden layer is of size K it is said to be composed of K neurons. Then the output layer takes in the output of the last hidden layer and makes a linear regression corresponding to the size of the output. Figure 1.3 is an illustration of a fully connected NN with 1 hidden layer and $q = 1$.

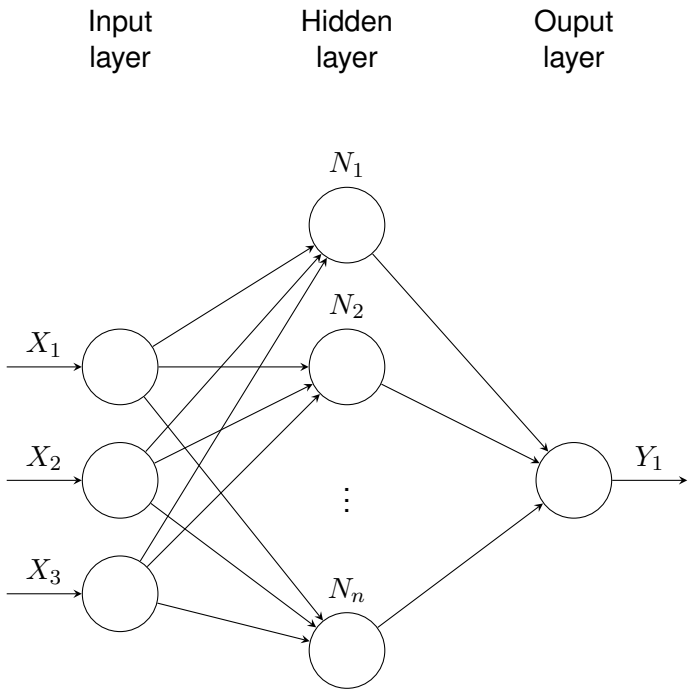


Figure 1.3: Neural network with a single hidden layer. The inputs respectively the outputs are $\mathbf{x} = (x_1, x_2, x_3)$, respectively $\mathbf{y} = (y_1)$. The number of neurons in the hidden layer is N_n.

The NN model for one hidden layer and for $q = 1$ has the form:

$$\mathrm{NN}(\mathbf{x}) = \beta_0 + \sum_{k=1}^{K} \beta_k \varsigma \left(w_{k0} + \sum_{j=1}^{d} w_{kj} \mathbf{x}_j \right), \tag{1.45}$$

with $\beta_0, \cdots, \beta_K$ parameters of the linear regression of the output layer, with $\beta_i \in \mathbb{R}$, $\{w_{kj}, k = 1, \cdots, K; j = 1, \cdots, d\}$ parameters of the linear regression of the hidden layer and ς the activation function. The function ς is chosen in advance and a specific discussion is made in section 1.2.1.1. The parameters of the different regressions are estimated from the data. To get a fit of the NN we need to have an error loss and a fitting algorithm. Fitting algorithms are presented in section 1.2.1.2.

It is possible to add as many hidden layers as one wishes. These are multi-layer neural networks. For some applications, especially in image processing, we will try to have a certain structure in the neural networks. Many coefficients w and β are therefore fixed at 0. An example is convolutional neural networks, used for example in (LeCun et al., 1989). There are also other structures of NN that do not rely only on fixing some parameters to 0, for example recursive neural network, see (Frasconi, Gori, Member, & Sperduti, 1998). In (Creswell et al., 2018), the authors present different structures but with the same idea of learning. In the following we will only focus on fully connected NN.

1.2.1.1 Activation function

The activation function ς is essential. Without an activation function or with a linear one the model proposed in eq. (1.45) would collapse into a linear model. The parameter conditioning of this linear model would also be very bad, so it is best to avoid it as much as possible. (Ramachandran, Zoph, & Le, 2017; Rasamoelina, Adjailia, & Sinák, 2020) propose a comparison of most of the activation functions available. We have decided to limit ourselves to the most common activation functions. In this section we present: *Sigmoid*, $\tanh$, ReLU and PReLU.

Definition 1.2.1. *The Sigmoid activation function is a function from* $\mathbb{R}$ *to* $[0,1]$*. The function can be expressed:*

$$\textit{Sigmoid}(z) = \frac{1}{1+e^{-z}}, \tag{1.46}$$

with $z \in \mathbb{R}$*.*

This function is the same function used in logistic regression. Sigmoid was widely used in the early instances of NN. The major advantage is that the function is infinitely differentiable. This makes it easier to implement. The main problem with this function is the output range. It does squash the output and cause the gradient to vanish in deep networks. Then a function with a steeper gradient is constructed to simplify the optimization.

Definition 1.2.2. *The Hyperbolic Tangent function is a function from* $\mathbb{R}$ *to* $[-1,1]$*. The function can be expressed:*

$$\tanh(z) = \frac{2}{1+e^{-2z}} - 1, \tag{1.47}$$

with $z \in \mathbb{R}$*.*

This activation function suffers from the same vanishing gradient problem, but it is much more limited for small networks. $\tanh$ is also infinitely differentiable. $\tanh$ will be used for BNN in section 1.2.2.2.

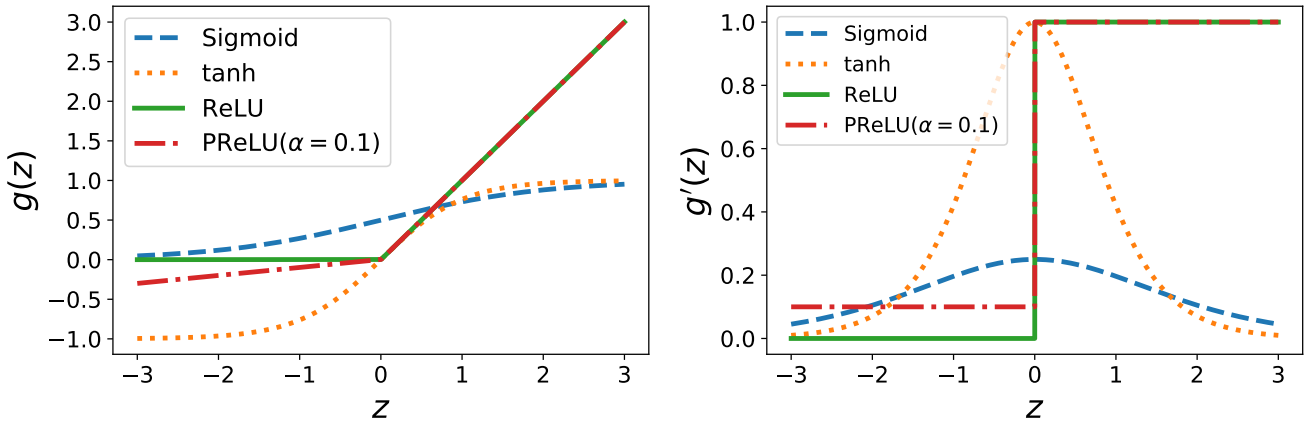
In modern neural networks the preferred choice is the ReLU function. This function is linear for all positive values and null elsewhere.

Definition 1.2.3. *The rectified linear unit* ReLU *function is a function from* $\mathbb{R}$ *to* $[0,+\infty)$*. The function can be expressed:*

$$\mathrm{ReLU}(z) = \max(0,z), \tag{1.48}$$

with $z \in \mathbb{R}$*.*

The ReLU function has the advantage to get true zero values. This can help regularization in a context that requires sparsity. The disadvantage to get values in $\mathbb{R}^+$ is that for certain types of NN (such as Recurrent NN), layers can have large number of neurons. The major downside of ReLU is vanishing



a) b)

Figure 1.4: a) the 4 activation functions proposed. b) the derivative of the activation functions.

gradient for negative values. This is called "dying ReLU", see (Trottier, Giguere, & Chaib-draa, 2017). The optimization of the parameters is then hard because the neurons that are far in the negative will have null gradient and null output.

Multiple techniques have been used to improve ReLU, see (Trottier et al., 2017). One classical attempt to fix "dying ReLU" is proposed in (He, Zhang, Ren, & Sun, 2015). The idea is to combine two linear functions: one for positive and one for negatives values.

Definition 1.2.4. *The parametric leaky version of a Rectified Linear Unit* PReLU *is a function from* $\mathbb{R}$ *to* $\mathbb{R}$*. The function can be expressed:*

$$\mathrm{PReLU}(z) = \max(0, z) + \alpha \min(0, z), \tag{1.49}$$

with $z \in \mathbb{R}$ *and* $\alpha > 0$ *a learnable parameter of the activation function.*

This function has the advantage to be able to learn negative values. The back propagation, defined in the following, will also be available to learn the coefficient α. This does not solve the problem of extreme values but an appropriate architecture can.

All the activation functions presented in this section are listed in table 1.1, with their main properties. According to current knowledge, none of the activation functions is superior to the others for all applications. As the work on activation functions is evolving very fast, see (Rasamoelina et al., 2020), it is possible that new and more efficient functions for our application cases will emerge. Activation functions presented in this section and their derivative are shown in table 1.1. We can see that the major variations are near zero: this has an influence on the initialization of the NN weights but should not have one on the final prediction.

Function	formula	Derivative	Output range
Sigmoid(z)	$\frac{1}{1+e^{-z}}$	$\frac{e^{-z}}{(1+e^{-z})^2}$	$[0,1]$
$\tanh(z)$	$\frac{2}{1+e^{-2z}}-1$	$\frac{1}{\cosh^2 z}$	$[-1,1]$
$\mathrm{ReLU}(z)$	$\max(0,z)$	$\begin{cases}1 & \text{if } z>0,\\ 0 & \text{otherwise.}\end{cases}$	$\mathbb{R}^+$
$\mathrm{PReLU}(z)$	$\max(0,z)+\alpha\min(0,z)$	$\begin{cases}1 & \text{if } z>0,\\ \alpha & \text{otherwise.}\end{cases}$	$\mathbb{R}$

Table 1.1: Activations functions

1.2.1.2 Fitting a Neural Network

In this section we present a method to fit NN. For this section we set $q = 1$. We restrict ourselves to a hidden layer for illustration in the following. As in (Hastie et al., 2009), the chosen method is back propagation because we can learn a lot from it. Having chosen a structure for our NN, it remains to estimate the weights β_i and w_{kj}. The set of weights to be estimated is the following:

$$\begin{aligned}&\{w_{kj};\ \ k=1,\cdots,K;\ \ j=1,\cdots,d\}\\ &\{\beta_i;\ \ i=0,\cdots,K\}.\end{aligned} \tag{1.50}$$

In the following the set of coefficients β_i and w_{kj} will be called θ. The first thing to do in order to build a model is to build an error function. We use one of the functions proposed in section 1.3. However, the name of the function is adapted to the use for the learning of NN.

Sum of squared error

Definition 1.2.5. *The sum of squared error $R(\theta)$ for the parameter θ and the learning set $\mathcal{D} = \left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}$ (remenber that $\mathbf{y}^{(i)} \in \mathbb{R}$) is:*

$$R(\theta) = \sum_{i=1}^{N}\left(\mathbf{y}^{(i)} - \mathrm{NN}_\theta(\mathbf{x}^{(i)})\right)^2. \tag{1.51}$$

The sum of squared error is the MSE times the number of samples N. Then it is also related to $\ell_{2,\text{loss}}$. We should note that the error is constructed with training data. This is coherent with the fact that this method is used during training.

The goal of fitting is then to minimize the function $R(\theta)$. The method used to minimize $R(\theta)$ is gradient descent. Gradient descent is known to be proposed by Cauchy and one of the first use for non-linear function is (Curry, 1944). I also must mention that Hadamard has proposed a similar method at the beginning of the 20$^{\text{th}}$ century. The back propagation is the name of the gradient descent but in the context of optimizing a NN.

However, we want our model to be well adapted to all data sets and not just to the learning set we use. One can imagine a model that knows the training points perfectly well to have a zero error at these points but whose prediction outside these points is meaningless. This problem is called overfitting. The problem of the overfitting will be addressed in the last paragraph.

Back propagation An overview of the gradient descent is available in (Ruder, 2017). The generic approach to get the gradient of the sum squared error loss of the NN is the back propagation. The general idea of back propagation is to express the derivative of the loss function and to minimize it using an iterative method.

The error can be expressed as a sum of errors:

$$R(\theta) = \sum_{i=1}^{N} R_i(\theta) = \sum_{i=1}^{N} (\mathbf{y}^{(i)} - \mathrm{NN}_\theta(\mathbf{x}^{(i)}))^2, \tag{1.52}$$

The back propagation algorithm is extremely simple:

1. Initialization: we start with an initial value of θ^0 for the parameters in θ.
2. Iteration on t:
   (a) Compute a vector δ that improves θ, i.e. with $\theta^{t+1} = \theta^t + \delta$ gives $R(\theta^{t+1}) < R(\theta^t)$.
   (b) Set $t \leftarrow t + 1$.

We can estimate the vector δ by calculating the gradient of $R(\theta)$, evaluated at θ^t:

$$\nabla R(\theta^t) = \left. \frac{\partial R(\theta)}{\partial \theta} \right|_{\theta=\theta^t} \tag{1.53}$$

This gives the direction along which $R(\theta)$ increases more rapidly. We will then move θ in the opposite direction:

$$\theta^{t+1} \leftarrow \theta^t - \rho \nabla R(\theta^t), \tag{1.54}$$

with ρ the learning rate. We choose ρ such that $R(\theta^{t+1}) \leq R(\theta^t)$. Since $R(\theta)$ is a sum of $R_i(\theta)$ we can just examine one of the terms. The expression of $R_i(\theta)$ depends on the parameters:

$$R_i(\theta) = \left(\mathbf{y}^{(i)} - \beta_0 - \sum_{k=1}^{K} \beta_k \varsigma(w_{k0} + \sum_{j=1}^{d} w_{kj} \mathbf{x}_j^{(i)}) \right)^2, \tag{1.55}$$

with $i = 1, \cdots, N$. In the following we will write $z_{ik} = w_{k0} + \sum_{j=1}^{d} w_{kj} \mathbf{x}_j^{(i)}$. We have the derivative with respect to β_k:

$$\frac{\partial R_i(\theta)}{\partial \beta_k} = -2(\mathbf{y}^{(i)} - \mathrm{NN}_\theta(\mathbf{x}^{(i)}))\varsigma(z_{ik}), \tag{1.56}$$

with ς the activation function, $i = 1, \cdots, N$ and $k = 1, \cdots, K$. we take the derivative with respect to w_{kj}:

$$\frac{\partial R_i(\theta)}{\partial w_{kj}} = -2(\mathbf{y}^{(i)} - \mathrm{NN}_\theta(\mathbf{x}^{(i)}))\beta_k \varsigma'(z_{ik})\mathbf{x}_j^{(i)}, \tag{1.57}$$

with ς' the derivative of the activation function, $i = 1, \cdots, N$, $j = 1, \cdots, d$ and $k = 1, \cdots, K$. With this we can have access the computation of $\nabla R(\theta^t)$.

The back propagation algorithm is composed of two passes. The forward pass is the first pass that computes the estimation of the function for a set of parameters θ. The backward pass is the pass that computes the error and computes the new values of the parameters.

The advantages of the back propagation method is its simplicity and its ability to be easily parallelized. But this method can be slow in particular for deep learning. A method that is commonly used to fit NN is BroydenFletcherGoldfarbShanno (BFGS) algorithm.

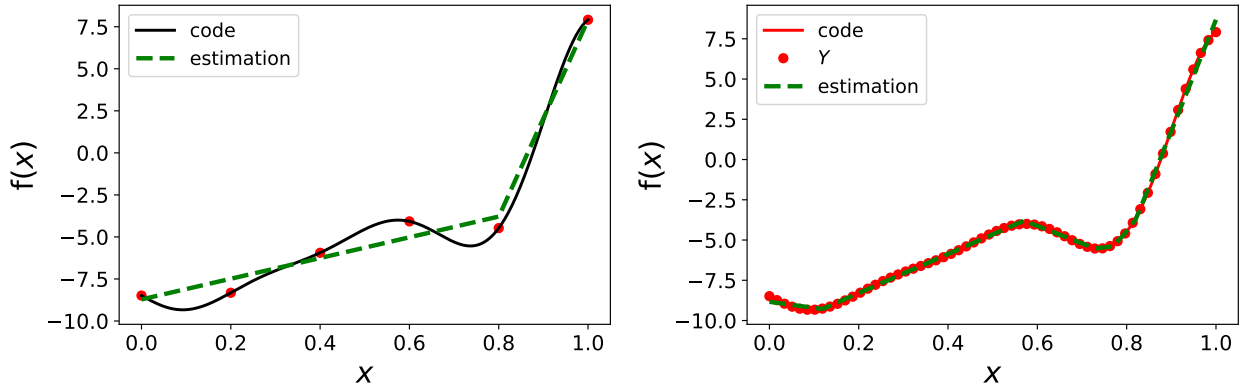
Initialization of the learning process To initialize the optimization algorithm we need initial values for β and w. The most natural way of doing it is to use null values but the NN will be symmetric and will never move from this initial position. Another idea is to compute with large random values but according to (Hastie et al., 2009, p. 398) it leads to poor optimization performances. The optimal method is to initialize the parameters with values close to 0 but randomly chosen. As exposed in (Nguyen & Widrow, 1990), the initial weights can have an influence on the learning speed, and so on the prediction in the industrial time-contrained world.

Prevent overfitting Often the dimension of θ is larger than the number of learning points. This can lead to the overfitting of the NN. The parameter θ that optimizes the function $R(\theta)$ is known to overfit. One way of solving this problem is to use a validation set. A validation set is a set that is not used to train the NN but only to evaluate its accuracy. If the accuracy is constant considering the validation set error, but the back propagation is still optimizing the error, it may mean that we have to stop optimization to avoid overfitting.

Another method to prevent overfitting is to add a penalty to the error function such that:

$$J(\theta) = \sum_i \beta_i^2 + \sum_{i,j} w_{ij}^2, \tag{1.58}$$

with J the penalty. The function to optimize is $R(\theta) + \lambda J(\theta)$ with $\lambda \in \mathbb{R}^{+\star}$ a parameter. This can be seen as an analogy of the ridge regression, see (Hastie et al., 2009, p. 223). Other forms of penalty can be imagined.



a) 6 points with 20 neurons b) 60 points with 100 neurons

Figure 1.5: Two regression problems solved with two different NN.

1.2.1.3 Implementation

The advances in this field require a great deal of computer knowledge to be used in practice. In what is presented below we have used the (Bingham et al., 2019; Paszke et al., 2019) python libraries. Pytorch (Paszke et al., 2019) allows to build a NN knowing its structure. Moreover, all the most advanced optimization tools are implemented, which allows to use NN efficiently. Pyro (Bingham et al., 2019) is a less mature library that allows the use of Bayesian formalism in the Pytorch library.

An example in the context of small data We assume that our computer code is the function:

$$\mathrm{g}(x) = 0.5(6x-2)^2 \sin(12x-4) + 10(x-0.5) - 5. \tag{1.59}$$

The experimental design is $\mathcal{D} = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. This function comes from (Le Gratiet, 2013b, p. 98). In this paragraph we construct a surrogate model based on NN fully connected presented in this section.

The result is given in fig. 1.5. The prediction of the surrogate model is very similar to local linear regression. This type of shape is common in applications.

1.2.2 Methods for uncertainty quantification with neural network

Neural network models have proven to be accurate for regression. The variety of applications that use NN for modeling makes it one of the most studied methods. Unlike GP, linear regression or polynomial chaos expansion, they do not have a native uncertainty computation. However, the uncertainty of the surrogate model is a key element in the study of the computer codes. In this subsection we present methods for uncertainty prediction of a NN. The first method presented is the dropout. In second, Bayesian neural network are presented and finally three others methods are introduced.

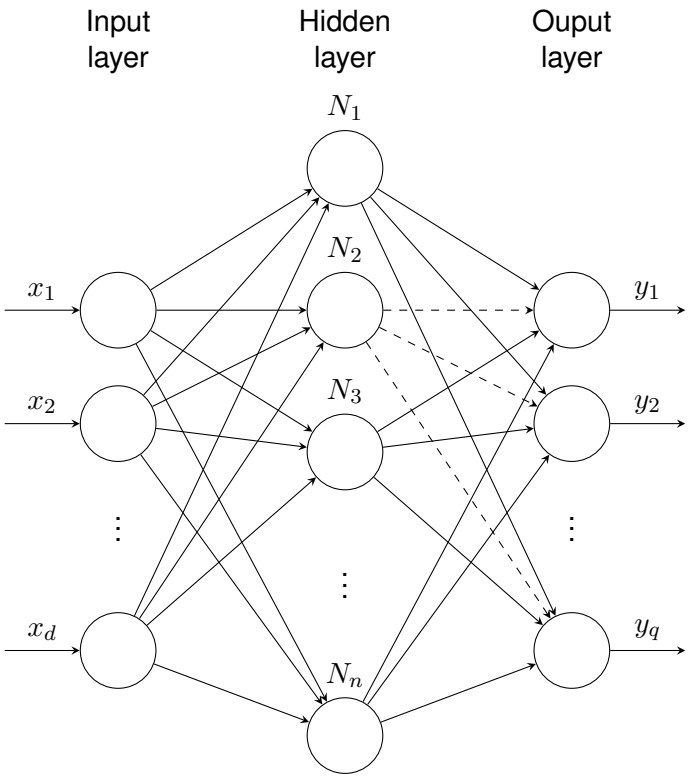


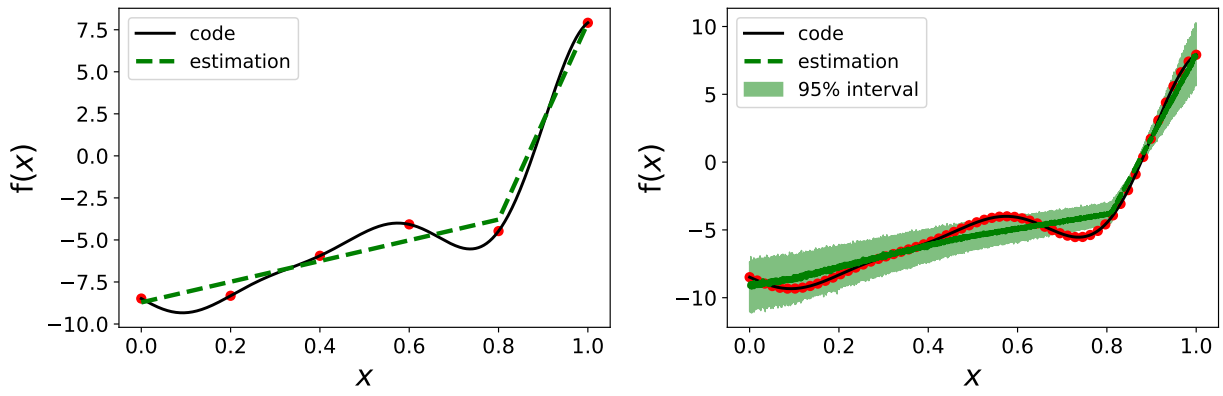
Figure 1.6: Illustration of the dropout in a particular NN. The coefficients are learnt by using only coefficients that are linked with the solid lines. But during the learning phase, neurons (here the second) have their links with the following layers removed. The dashed lines represent the coefficients that are put to zero during an iteration of the learning.

1.2.2.1 Dropout

According to (James et al., 2021a, p. 438), dropout is an efficient form of regularization, close to ridge regularization. The idea is to remove a fraction β of the nodes in a layer when fitting the model. This operation is reproduced for different training instances. Nodes are randomly selected and ignored in an instance of training. Then other nodes are randomly selected, the NN is trained and so on. To compensate the loss of nodes the weights are scaled for each layer by a factor $\frac{1}{1-\beta}$. This is supposed to prevent nodes to be overspecialized. In practice, dropout is easy to implement (just an activation function with Bernoulli trial has to be implemented) and has shown relative efficiency (James et al., 2021a, Table 10.1).

This method is presented here because it is theoretically similar for an infinite number of neurons with the deep Gaussian process, see (Damianou & Lawrence, 2013) and (Gal & Ghahramani, 2016) for details.

This does not make the dropout a method of uncertainty quantification but the NN has a non-deterministic output during the learning phase. The idea presented in (Gal & Ghahramani, 2016)is to activate dropout during the testing period to effortlessly provide an ensemble of multiple predictions. This model has shown its efficiency in (Althoff, Rodrigues, & Bazame, 2021). This model is working well for complex structures of NN. The dropout also needs more data to fit than for classical regularization methods.



a) 6 points with 20 neurons b) 60 points with 20 neurons

Figure 1.7: Two regression problems solved with two different NN with Dropout. On the left we removed the 95% interval because it was too large. The figure shows two different training sets with the NN prediction with dropout.

Example We use the same data as for NN. We assume that our computer code is the function:

$$\mathrm{g}(x) = 0.5(6x-2)^2 \sin(12x-4) + 10(x-0.5) - 5. \tag{1.60}$$

The experimental design is $\mathcal{D} = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. A second learning set with more data is also proposed. We see that the model is giving a prediction interval. The model has shown a poor prediction, see fig. 1.7. The average prediction is indeed worse than for the GP and the prediction interval is much larger. However, there is not enough data to make the dropout work properly.

1.2.2.2 Bayesian Neural Network

Neural networks have been used to emulate unknown functions based on data (Cigizoglu & Alp, 2006), and in particular computer codes (Tripathy & Bilionis, 2018). Our goal, however, is also to quantify the uncertainty of the emulation. BNN makes it possible to quantify predictive uncertainties. Below we present the BNN structure and the priors for the parameters.

The BNN model In order to simplify the notations, we present a BNN with one hidden layer. It can be easily extended for multi-layer BNN. Let N_l be the number of neurons in the hidden layer. The function g to emulate goes from $\mathbb{R}^d$ to $\mathbb{R}$.

The output of the first layer is

$$\mathbf{y}_1 = G\left(w_1 \mathbf{x} + \beta_1\right), \tag{1.61}$$

with $\mathbf{x} \in \mathbb{R}^d$ the input vector of the BNN, $\beta_1 \in \mathbb{R}^{N_l}$ the bias vector, $w_1 \in \mathbb{R}^{N_l \times d}$ the weight matrix and $\mathbf{y}_1 \in \mathbb{R}^{N_l}$ the output of the hidden layer. The function $G : \mathbb{R}^{N_l} \to \mathbb{R}^{N_l}$ is of the form $G(\beta) = (\varsigma(\beta_j))_{j=1}^{N_l}$,

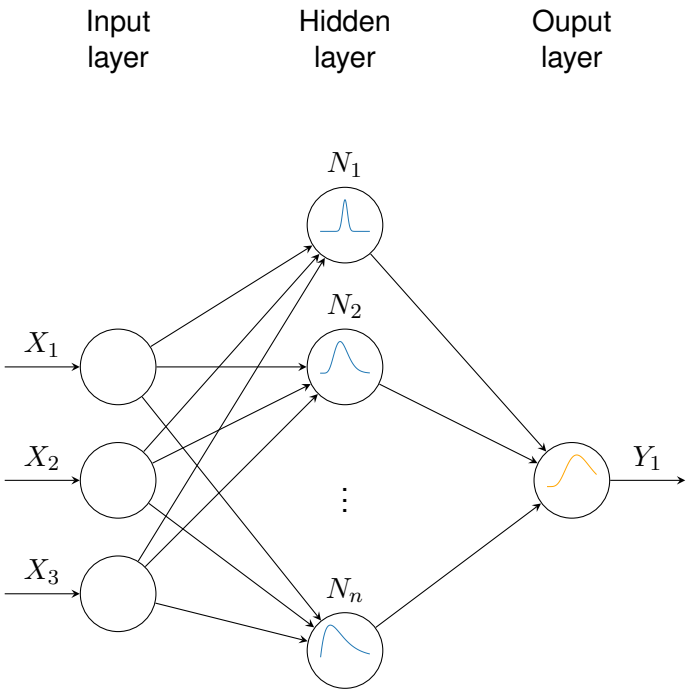


Figure 1.8: Representation of the BNN. The law of each neuron is represented at its centre. This means that in order to have a realization of the output, the parameters are drawn from the associated law represented. For simplicity, we have only represented one law per neuron and not all the parameter laws.

where the activation function ς can be hyperbolic tangent or ReLU for instance. The second (and last) layer is fully linear:

$$BNN(\mathbf{x}) = w_2^T \mathbf{y}_1 + \beta_2, \tag{1.62}$$

with $w_2 \in \mathbb{R}^{N_l}$ the weight matrix, $\beta_2 \in \mathbb{R}$ the bias vector and $BNN(\mathbf{x}) \in \mathbb{R}$ the scalar output of the BNN at point $\mathbf{x}$.

We use a Bayesian framework similar to the one presented in (Jospin, Buntine, Boussaid, Laga, & Bennamoun, 2020). Let $\boldsymbol{\theta}$ denote the parameter vector of the BNN, which is here $\boldsymbol{\theta} = (w_i, \beta_i)_{i=1,2}$. The probability distribution function (PDF) of the output given $\mathbf{x}$ and $\boldsymbol{\theta}$ is

$$p(y|\mathbf{x}, \boldsymbol{\theta}, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(- \frac{(y - BNN_{\boldsymbol{\theta}}(\mathbf{x}))^2}{2\sigma^2} \Big), \tag{1.63}$$

where σ^2 is the variance of the random noise added to account for the fact that the neural network is an approximation. $BNN_{\boldsymbol{\theta}}(\mathbf{x})$ is the output of the neural network with parameter $\boldsymbol{\theta}$ at point $\mathbf{x}$.

Applying Bayes' theorem, the posterior PDF of $(\boldsymbol{\theta}, \sigma)$ given the data $\mathcal{D} = (\mathbf{x}^{(i)}, y_i)_{i=1}^N$ is

$$p(\boldsymbol{\theta}, \sigma|\mathcal{D}) = \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \boldsymbol{\theta}, \sigma) p(\boldsymbol{\theta}, \sigma) \tag{1.64}$$

up to a multiplicative constant, where $p(\boldsymbol{\theta}, \sigma)$ is the prior distribution of $\boldsymbol{\theta}, \sigma$ described below. The posterior

distribution of the output at $\mathbf{x}$ has pdf

$$p(y|\mathbf{x},\mathcal{D}) = \iint p(y|\mathbf{x},\boldsymbol{\theta},\sigma)p(\boldsymbol{\theta},\sigma|\mathcal{D})d\boldsymbol{\theta}d\sigma, \tag{1.65}$$

and the two first moments are:

$$\mathbb{E}_{\text{post}}[Y] = \iint BNN_{\boldsymbol{\theta}}(\mathbf{x})p(\boldsymbol{\theta},\sigma|\mathcal{D})d\boldsymbol{\theta}d\sigma, \tag{1.66}$$

$$\mathbb{E}_{\text{post}}\left[Y^2\right] = \iint \left(BNN_{\boldsymbol{\theta}}(\mathbf{x})^2 + \sigma^2\right)p(\boldsymbol{\theta},\sigma|\mathcal{D})d\boldsymbol{\theta}d\sigma. \tag{1.67}$$

Contrarily to GP regression, the prediction of a BNN cannot be expressed analytically as shown by (1.65) but there exist efficient sampling methods. To sample the posterior distribution of the BNN output, we need to sample the posterior distribution of $(\boldsymbol{\theta},\sigma)$. This is done in a following paragraph. By eqs. (1.66) and (1.67), the estimated mean $\tilde{f}$ and variance $\tilde{V}$ of the output at point $\mathbf{x}$ are:

$$\tilde{f}(\mathbf{x}) = \frac{1}{N_v}\sum_{i=1}^{N_v} BNN_{\boldsymbol{\theta}_i}(\mathbf{x}), \tag{1.68}$$

$$\tilde{V}(\mathbf{x}) = \frac{1}{N_v}\sum_{i=1}^{N_v}\left[BNN_{\boldsymbol{\theta}_i}(\mathbf{x}) - \tilde{f}(\mathbf{x})\right]^2 + \frac{1}{N_v}\sum_{i=1}^{N_v}\sigma_i^2, \tag{1.69}$$

where the $(\boldsymbol{\theta}_i,\sigma_i)_{i=1}^{N_v}$ is the sample of $(\boldsymbol{\theta},\sigma)$ with its posterior distribution.

Priors for BNN Here we choose a prior distribution for $(\boldsymbol{\theta},\sigma)$ that is classical in the field of BNN (Jospin et al., 2020, p. 10). The prior laws of the parameters $(w_i,\beta_i)_{i=1,2}$ are:

$$w_i \sim \mathcal{N}\left(\mathbf{0},\sigma_{w_i}^2\mathbf{I}\right), \quad \mathbf{b}_i \sim \mathcal{N}\left(\mathbf{0},\sigma_{\beta_i}^2\mathbf{I}\right), \quad i=1,2, \tag{1.70}$$

with $\sigma_{w,\beta_{1,2}}$ the prior standard deviations. The prior for σ is the standard Gaussian $\mathcal{N}(0,1)$ (assuming the function g has been normalized to be of order one). All parameters are assumed to be independent. The choice of priors in BNN is discussed in (Karaletsos & Bui, 2020). The authors use GP as prior of BNN. Another way to see prior for BNN came from (Matsubara, Oates, & Briol, 2020). It is the same as presented here for one hidden layer.

Sampling of the posterior law In this paragraph we study the sampling algorithm for the posterior law. The Markov Chain Monte Carlo (MCMC) method aims at generating the terms of an ergodic Markov chain $(X_n)_{n\in\mathbb{N}}$ whose invariant measure is the target law with density $p(\mathbf{x})$ which is known up to a multiplicative constant. This Markov chain is specially constructed for this purpose, in the sense that its transition

kernel is defined such that p is its unique invariant probability. There are several possible variations of this principle, such as the Metropolis-Hastings algorithm, see (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953; Robert, 2004, sec. 6.3.2).

The Metropolis-Hastings algorithm is as follows. We give ourselves a starting point $\mathbf{x}_0$ and an exploration law, i.e. a family $(q(\mathbf{x}', \mathbf{x}))_{\mathbf{x}\in\mathbb{R}^d}$ of probability densities on $\mathbb{R}^d$ parametrized by $\mathbf{x}' \in \mathbb{R}^d$ that are easily simulated. We assume that we have simulated the n-th term of the chain X_n.

1. We make a proposition X'_{n+1} drawn according to the density law $q(X_n, \cdot)$.

2. We calculate the acceptance rate $a(X_n, X'_{n+1}) = \min(1, \rho(X_n, X'_{n+1}))$ with,

$$\rho(\mathbf{x}_n, \mathbf{x}'_{n+1}) = \frac{p(\mathbf{x}'_{n+1})q(\mathbf{x}'_{n+1}, \mathbf{x}_n)}{p(\mathbf{x}_n)q(\mathbf{x}_n, \mathbf{x}'_{n+1})}. \tag{1.71}$$

3. We draw $U_{n+1} \sim \mathcal{U}(0, 1)$.

4. We put:

$$X_{n+1} = \begin{cases} X'_{n+1} & \text{if } U_{n+1} \leq a(X_n, X'_{n+1}) \\ X_n & \text{if } U_{n+1} > a(X_n, X'_{n+1}) \end{cases} \tag{1.72}$$

We note that we only need to know p up to a multiplicative constant to be able to implement the algorithm according to eq. (1.71). In the case where q is symmetrical $q(\mathbf{x}', \mathbf{x}) = q(\mathbf{x}, \mathbf{x}')$, the acceptance rate is simply $\min(1, \frac{p(X'_{n+1})}{p(X_n)})$. We have a symmetrical q in particular when the exploration law is Gaussian: $(q(\mathbf{x}', \mathbf{x}))_{\mathbf{x}\in R^d}$ is the density of the law $\mathcal{N}(\mathbf{x}', \sigma^2 I)$ with σ^2 to be calibrated in order to have an acceptance rate that is neither too high (which means that we do not explore enough), nor too low (which means that we reject the proposition too often because it is too far). Usually we calibrate σ^2 during the simulation to observe a constant acceptance rate of the order of $\frac{1}{4}$, for more details see (Gelman, Gilks, & Roberts, 1997).

The Hamiltonian Monte Carlo (HMC) algorithm is a special instance of Metropolis-Hastings algorithm. HMC method is special because the exploration law $q(\mathbf{x}', \mathbf{x})$ is determined by a Hamiltonian dynamic, in which the potential energy depends on the target density p. This model is proposed in (Duane, Kennedy, Pendleton, & Roweth, 1987). One algorithm that is efficient for HMC on the No-U-Turn Sampler (NUTS). It is used in this manuscript to sample the posterior distribution of $(\boldsymbol{\theta}, \sigma)$, see (Hoffman & Gelman, 2014).

1.2.2.3 Other black-box methods

We have already seen the BNN and the drop out in the previous sections. There are many other methods with uncertainty quantification for neural networks. In this section we will discuss a few of them.

A good survey of these methods is available in (Kabir et al., 2018). In the numerous methods that generate uncertainty of prediction from a NN, 3 methods have been selected: bootstrap method, conformal prediction and lower upper bound estimation method. These methods will be briefly introduced in the following paragraphs.

Bootstrap method The Bootstrap method is a well-known method in statistics that processes inference from learning points, see (Hastie et al., 2009, p. 249). From a learning set of N points we select a sub-learning set of size N'. The model is fitted using the sub-learning set. With all the different models fitted it is now possible to obtain a sample of the output. The prediction is the empirical mean of all the prediction outputs. The prediction interval is determined by the empirical quantiles depending on the prediction interval we want. This method can be costly if we want to build a complete sample with all possible sub-learning sets. The choice of the size and the number of sub-learning sets is a question in Bootstrap.

This method has been used to quantify output uncertainty for NN. A method close to the general idea has been implemented for a practical application in (M. K. Tiwari & Chatterjee, 2010). (Khosravi, Nahavandi, Srinivasan, & Khosravi, 2015) proposes a method that improves the prediction interval using only the variance for uncertainty prediction.

Conformal prediction Conformal prediction is an online learning method that provides a prediction of a time-series based on previous values and input of the model. The idea is to predict the value of $\mathbf{y}_n$ knowing $\mathbf{x}_n$ and the previous pairs $(\mathbf{x}_k, \mathbf{y}_k)_{k=1,\cdots,n-1}$. More details are available in (Balasubramanian, Ho, & Vovk, 2014). This method is specific to time-series and the prediction assumes that the properties of the time-series are stationary.

Lower upper bound estimation method This method, unlike the other two, requires changes to the structure of the neural network. The general idea is to build a classical network but to change the last layer so that for each output there are actually two outputs. One output corresponds to the upper bound and one to the lower bound. The problem is how to conserve this property in the learning phase. An algorithm is proposed in (Khosravi, Nahavandi, Creighton, & Atiya, 2011). This method is efficient but the fact that the target prediction interval is set strongly in the model makes it very inflexible.

The choice of BNN is based on the comparison given in the figure (Hastie et al., 2009, p. 413), see fig. 1.9. It is shown that the BNN is very competitive compared to other regression methods.

1.3 Criteria for assessing the quality of models

In order to evaluate surrogate models we need to have accurate and reliable metrics. There are two goals that we want to achieve. First, the error of prediction will compare the mean of prediction with the output

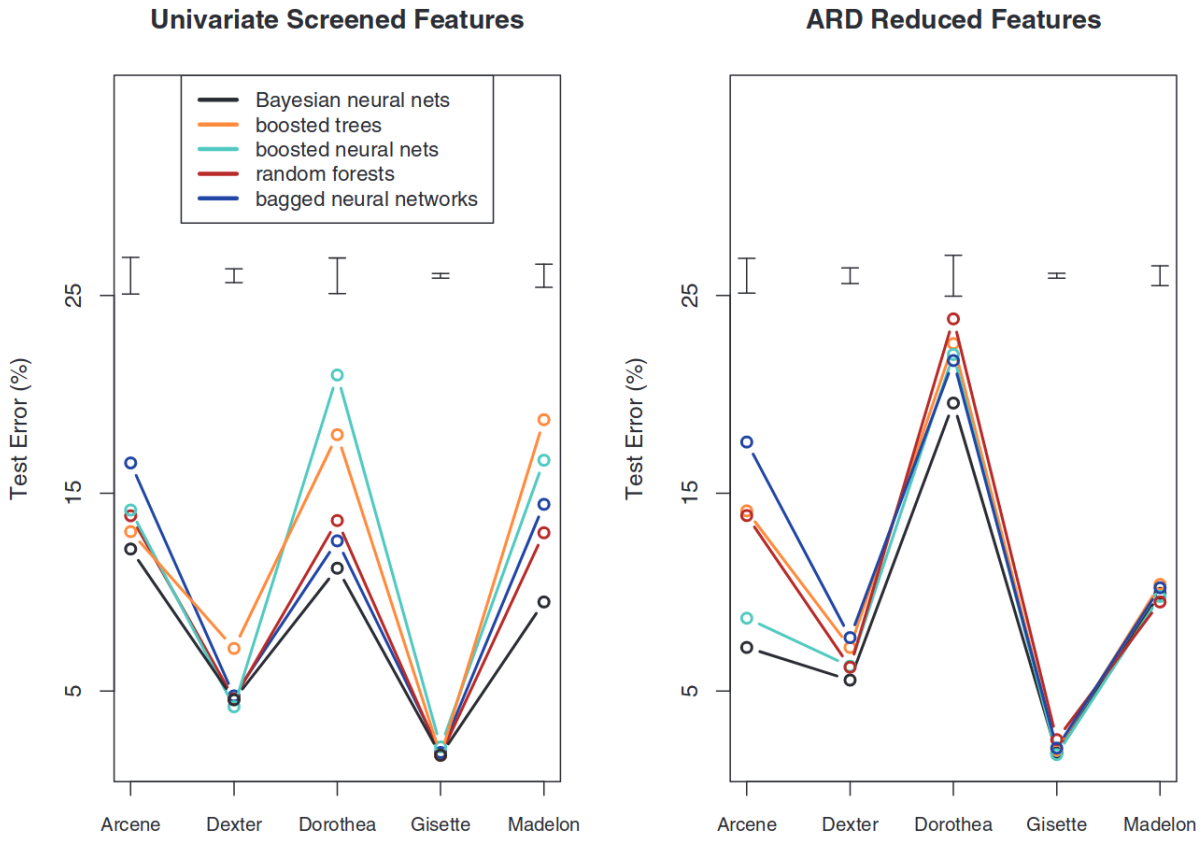


Figure 1.9: Reproduction of (Hastie, Tibshirani, & Friedman, 2009, fig. 11.12): Performance of different learning methods on five problems, using both univariate screening of features (top panel) and a reduced feature set from automatic relevance determination. The error bars at the top of each plot have width equal to one standard error of the difference between two error rates. On most of the problems several competitors are within this error bound.

of the code. Second, the prediction interval must be evaluated. This problem is a trade-off between a large size, too large to give information, that contains certainly the output and a small size that does not contain the output value.

The problem with studying the efficiency of a model is that there is no *best* estimate of the error. It is therefore essential to use estimators that fit our needs. It should be noted that according to the adage: *there is no free lunch in statistics*, there is no method that is superior to all others. This also applies to the evaluation of models.

In this section we will consider a test set: $\left\{\mathbf{x}_{\mathrm{T}}^{(i)}, \mathrm{g}(\mathbf{x}_{\mathrm{T}}^{(i)})\right\}_{i=1,\cdots,N_{\mathrm{T}}}$, with N_{T} the size of the test set. In this section we consider that the surrogate model gives a prediction mean $\tilde{\mu}\left(\mathbf{x}_{\mathrm{T}}^{(i)}\right)$ and a prediction interval $\mathcal{I}_{\alpha}(\mathbf{x})$ the prediction interval at point $\mathbf{x}$ with confidence level α.

1.3.1 Error prediction metrics

Definition 1.3.1. *$\ell_{p,\text{loss}}$ is the error defined with the p-norm, with $p \in [1, +\infty]$. We can express it as:*

$$\ell_{p,\text{loss}} = \left(\frac{1}{N_{\mathrm{T}}}\sum_{i=1}^{N_{\mathrm{T}}} \|\tilde{\mu}\left(\mathbf{x}_{\mathrm{T}}^{(i)}\right) - \mathrm{g}(\mathbf{x}_{\mathrm{T}}^{(i)})\|_p^p\right)^{\frac{1}{p}}, \tag{1.73}$$

with $\|\cdot\|_p$ the l^p norm over $\mathbb{R}^q$.

A highly predictive model will give a $\ell_{p,\text{loss}}$ close to 0, but the value will depend on p. In a more general way, the smaller p is, the more we take into account all the errors in the same way, whereas the larger p is, the more we penalize the fact of making a large error. Generally only the values $p = 1$,$p = 2$ and $p = \infty$ are used.

Definition 1.3.2. *The mean square error, MSE is:*

$$\mathit{MSE} = \frac{1}{N_{\mathrm{T}}}\sum_{i=1}^{N_{\mathrm{T}}} \|\tilde{\mu}\left(\mathbf{x}_{\mathrm{T}}^{(i)}\right) - \mathrm{g}(\mathbf{x}_{\mathrm{T}}^{(i)})\|_2^2 \tag{1.74}$$

MSE is the most commonly-used measure of the quality of a surrogate model, according to (James, Witten, Hastie, & Tibshirani, 2021c, p. 29). This estimator will give a small value to an accurate surrogate model and a bigger one to a less accurate surrogate model. For a description on how to use this estimator in learning see (James et al., 2021c, pp. 29-33).

The error evaluations proposed above are all dependent on the unit of the model output. As a consequence, the error values given will depend on the set normalization. In order to overcome this problem we propose to use indicators that take the form of a proportion.

Definition 1.3.3. *The* Q^2 *error is evaluated by:*

$$\mathrm{Q}^2 = 1 - \frac{\sum_{i=1}^{N_\mathrm{T}} \|\tilde{\mu}\left(\mathbf{x}_\mathrm{T}^{(i)}\right) - \mathrm{g}(\mathbf{x}_\mathrm{T}^{(i)})\|_2^2}{N_\mathrm{T}\mathbb{V}_\mathrm{T}\left(\mathrm{g}\right)}, \tag{1.75}$$

with $\mathbb{V}_\mathrm{T}(\mathrm{g}) = \frac{1}{N_\mathrm{T}}\sum_{i=1}^{N_\mathrm{T}} \| \mathrm{g}(\mathbf{x}_\mathrm{T}^{(i)}) - \frac{1}{N_\mathrm{T}}\sum_{j=1}^{N_\mathrm{T}} \mathrm{g}(\mathbf{x}_\mathrm{T}^{(j)})\|_2^2$.

A highly predictive model gives a Q^2 close to 1 while a less predictive model has a smaller Q^2. We note that the Q^2 can be negative, this is generally not a good sign for the accuracy of the surrogate model. This estimator comes from the Nash-Sutcliffe model efficiency coefficient introduced in (Nash & Sutcliffe, 1970).

There is an estimator of the error that is almost identical to Q^2 which is the R^2. The only difference between these estimators is that the Q^2 is constructed from test data (which was not used for training), whereas the R^2 is constructed on the training data $\left\{\left(\mathbf{x}^{(i)}, \mathrm{g}(\mathbf{x}^{(i)})\right), i = 1, \cdots, N\right\}$. This has a big consequence because the R^2 is between 0 and 1 for linear regression while the Q^2 is less than 1. The R^2 is an estimation of the error that is expressed:

$$\mathrm{R}^2 = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}}, \tag{1.76}$$

with $\mathrm{TSS} = \sum_{i=1}^{N} \| \mathrm{g}(\mathbf{x}^{(i)}) - \frac{1}{N}\sum_{j=1}^{N} \mathrm{g}(\mathbf{x}^{(j)})\|_2^2$ the total sum of square and $\mathrm{RSS} = \sum_{i=1}^{N} \|\tilde{\mu}\left(\mathbf{x}^{(i)}\right) - \mathrm{g}(\mathbf{x}^{(i)})\|_2^2$ the residual sum of square. For more detail on R^2, with specification in linear regression, (James et al., 2021b, pp. 69-70).

A well-known error evaluation is RMSE, Root Mean Square Error. The estimator I use the most is Q^2.

Remark 1.3.4. *When the output of the computer code is a time-series we can evaluate the estimation of the error for each point and get a time-series* $\mathrm{Q}^2(t)$*.*

1.3.2 Error uncertainty quantification metrics

In this paragraph we evaluate the prediction interval of the surrogate model. It is very difficult to have a complete evaluation with only one real of the quality of the prediction interval. The goal is that the interval is as small as possible and that the output of the code is within this interval with a probability close to what is required. To evaluate the surrogate model we choose a value of the confidence level α. Here we will limit ourselves to the case where $q = 1$.

The first indicator presented is simple and simply checks whether the value of α is possible with respect to the prediction interval. This measure was presented in (Hong, Meeker, & McCalley, 2009).

Definition 1.3.5. *The coverage probability* CP_α *is defined as the probability for the actual value of the*

function to be within the prediction interval with confidence level α of the surrogate model:

$$\mathrm{CP}_\alpha = \frac{1}{N_\mathrm{T}} \sum_{i=1}^{N_\mathrm{T}} \mathbf{1}_{g(\mathbf{x}_\mathrm{T}^{(i)}) \in \mathcal{I}_\alpha(\mathbf{x}_\mathrm{T}^{(i)})}, \tag{1.77}$$

with $\mathbf{1}$ the indicator function and $\mathcal{I}_\alpha(\mathbf{x}_T^i)$ the prediction interval at point $\mathbf{x}$ with confidence level α.

The prediction uncertainty of the surrogate model is well characterized when CP_α is close to α.

We then want to see if the prediction interval is as small as it can. To this purpose we use an estimator presented in (Acharki, Bertoncello, & Garnier, 2023).

Definition 1.3.6. *The mean prediction interval width MPIW_α is the average width of the prediction intervals:*

$$\mathrm{MPIW}_\alpha = \frac{1}{N_\mathrm{T}} \sum_{i=1}^{N_\mathrm{T}} \left| \mathcal{I}_\alpha(\mathbf{x}_\mathrm{T}^{(i)}) \right|, \tag{1.78}$$

where $\mathcal{I}_\alpha(\mathbf{x})$ is the prediction interval at point $\mathbf{x}$ with confidence level α and $|\mathcal{I}_\alpha(\mathbf{x})|$ the length of the prediction interval $\mathcal{I}_\alpha(\mathbf{x})$.

The prediction interval of the surrogate model is small when MPIW_α is small.

Chapter 2

Dimension reduction for regression

In Chapter 1, we saw a number of regression methods with quantification of the prediction uncertainty. However, these methods are only effective when the outputs to be approximated are relatively low dimensional. When the regression problem has high-dimensional outputs, such as time-series, then the natural solution is to reduce the dimension of the outputs to return to the previous case.

The problem we are interested in is the following: let us consider a computational code of which outputs are of the form $z(\mathbf{x}, t)$ with $\mathbf{x} \in Q \subset \mathbb{R}^d$ and $t \in [0, 1]$.

We know this code in a limited number of $\mathbf{x}$ but for t discretized on a regular grid. We try to reduce the output $z(\mathbf{x}, t)$ by r coefficients $a_1(\mathbf{x}), \cdots, a_r(\mathbf{x})$. For that we use a dimension reduction noted $\mathcal{F}$. The principle of dimension reduction is illustrated in fig. 2.1. For the regression, the objective is to predict in the space of $z(\mathbf{x}, t)$, it is thus necessary to be able to return to the output space of the code. This is how we look for a pseudo-inverse of $\mathcal{F}$ denoted $\mathcal{F}^{-1}$. Note that $\mathcal{F}^{-1}$ is not an inverse of $\mathcal{F}$ because there would be no reduction of dimension. The only exception would be the case where r is equal to the number of points on the time grid. Thus, we obtain an approximation $\hat{z}(\mathbf{x}, t)$ of $z(\mathbf{x}, t)$. Our objective is to have r as small as possible and to have $\hat{z}(\mathbf{x}, t)$ as close as possible to $z(\mathbf{x}, t)$. In order to solve the regression problem, we will then use the algorithms presented in chapter 1 on the coefficients $a_1(\mathbf{x}), \cdots, a_r(\mathbf{x})$.

In the section 2.1, the most common dimension reduction techniques in a regression framework are presented. These dimension reduction techniques are constructed based on data. We restrict ourselves to methods that allow an inverse reconstruction in the original space (existence of a pseudo inverse introduced before)

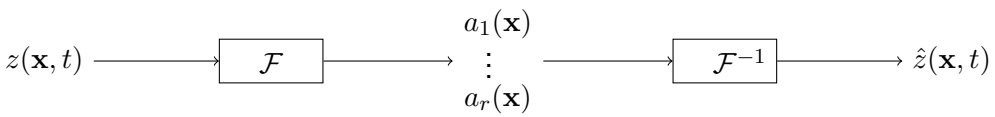


Figure 2.1: Illustration of the dimension reduction for time-series outputs. $\mathcal{F}$ is the dimension reduction function and $\mathcal{F}^{-1}$ its pseudo inverse.

The methods presented are singular value decomposition, independent component analysis, autoencoders and kernel methods.

In the section 2.2, the wavelet transform is presented. Although not strictly speaking a dimension reduction method, it is widely used to compress information. Before introducing the wavelet transform, we will introduce the Fourier transform. Finally, dimension reduction methods based on the wavelet transform are presented.

2.1 Data based methods

Dimension reduction is usually based on the learning data. A data-driven dimension reduction method consists in building a decomposition specific to the data. The challenges are the construction of dimension reduction with a low number of data and the associated uncertainty quantification. This section is divided into a part on singular value decomposition, a part on independent component analysis, a part on kernels PCA and a part on autoencoders.

2.1.1 Singular Value Decomposition

The most commonly used method according to (Hastie et al., 2009) is the Singular Value Decomposition (SVD). Principal component analysis (PCA) is a particular case of SVD. The aim is to achieve a low rank approximation of a matrix $\mathbf{Y}$.

Definition 2.1.1. *The singular value decomposition (SVD) is a factorization of a matrix. It is a generalization of the eigendecomposition but for a non-square matrix. The singular value decomposition of the matrix* $\mathbf{Y} \in \mathcal{M}_{n,m}(\mathbb{R})$ *is the factorization:*

$$\mathbf{Y} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T, \tag{2.1}$$

where $\mathbf{U} \in \mathcal{M}_{n,n}(\mathbb{R})$ *and* $\mathbf{V} \in \mathcal{M}_{m,m}(\mathbb{R})$ *are orthogonal and* $\mathbf{\Lambda}$ *a rectangular diagonal matrix in* $\mathcal{M}_{n,m}(\mathbb{R})$ *with non-negative coefficients. We choose that the coefficients of* $\mathbf{\Lambda}$ *are non-decreasing. The matrix can be expressed as* $\mathbf{U} = [u_1, \cdots, u_n]$, $\mathbf{V} = [v_1, \cdots, v_m]$ *and* $\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \cdots, \lambda_n)$, *where the vectors* $(u_i)_i$ *form an orthogonal basis of* $\mathbb{R}^n$, *the vector* $(v_i)_i$ *form an orthogonal basis of* $\mathbb{R}^m$, $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$, *and*

The interpretation of this decomposition is that the first λ_i and the associated singular vectors u_i,v_i are the most important in the decomposition of the matrix. And each is less important than the next in the reconstruction of the matrix $\mathbf{Y}$. So it is very easy to reduce dimension by truncating the SVD.

To be able to truncate it is necessary to give a reconstruction error. We look for the difference between the matrix $\mathbf{Y}$ and the reconstruction $\hat{\mathbf{Y}}$. To construct $\hat{\mathbf{Y}}$ we truncate the coefficients by taking only the first

r triplets of singular values and vectors. Thus, we can express $\hat{\mathbf{Y}}$ as:

$$\hat{\mathbf{Y}} = \sum_{i=1}^{r} \lambda_i u_i v_i^T, \tag{2.2}$$

with r the chosen size of the reduced space. This reconstructed matrix must be compared to the actual matrix in order to evaluate the error of reconstruction. To calculate the error the Frobenius norm is used. It is defined for the matrix A as:

$$\|A\|_F = \sqrt{\mathrm{trace}(A^T A)}. \tag{2.3}$$

By properties on the Frobenius norm and SVD decomposition we have:

$$\|\mathbf{Y} - \hat{\mathbf{Y}}\|_F^2 = \sum_{i=r+1}^{N} \lambda_i^2, \tag{2.4}$$

with $N = \min\{n, m\}$. This development gives us the following theorem.

Theorem 2.1.2. *Rank theorem The best approximation by a r-rank matrix of a matrix $\mathbf{Y}$ in the sense of the Frobenius norm is the SVD-decomposition of $\mathbf{Y}$ reduced to its r first coefficients. This is expressed as:*

$$\mathrm{argmin}_{\mathrm{rank}\, A=r} \|A - \mathbf{Y}\|_F^2 = \sum_{i=1}^{r} \lambda_i u_i v_i^T. \tag{2.5}$$

We will now apply the SVD decomposition to the $N_t \times N_x$ matrix $\mathbf{Y} = \left\{z(\mathbf{x}^{(i)}, t_u)\right\}_{i,u}$, N_x is the number of elements and N_t is the number of time samples. We have $N = \min\{N_x, N_t\}$. We express our data as the sum of the product of a coefficient depending on $\mathbf{x}$ and a function depending on t. The SVD, eq. (2.1), of $\mathbf{Y}$ gives:

$$z(\mathbf{x}^{(j)}, t_u) = \sum_{i=1}^{N} a_i(\mathbf{x}^{(j)}) \gamma_i(t_u), \tag{2.6}$$

with $a_i(\mathbf{x}^{(j)}) = \lambda_i (u_i)_j$ and $\gamma_i(t_u) = (v_i)_u$. In particular, the vectors γ_i, $i = 1, \cdots, N_t$, form a basis of $\mathbb{R}^{N_t}$ and we define the orthogonal matrix $\gamma = \{\gamma_i\}_i$. This is the SVD expression of the dimension reduction function $\mathcal{F}$, introduced in fig. 2.1.

The stochastic process $Z(\mathbf{x}, t_u)$ with $\mathbf{x} \in \mathbb{R}^d$ and $u = 1, \cdots, N_t$ is used to emulate the function. We consider that there exists a determinist, unknown basis $\mathbf{\Gamma}$ such that:

$$Z(\mathbf{x}, t_u) = \sum_{i=1}^{N} A_i(\mathbf{x}) \Gamma_i(t_u), \tag{2.7}$$

with $A_i(\mathbf{x}) = \sum_{u=1}^{N_t} Z(\mathbf{x}, t_u) \Gamma_i(t_u)$. We assume that $A_i(\mathbf{x})$ are independent Gaussian processes. Then $(Z(\mathbf{x}, t_u))_{\mathbf{x} \in Q, u=1,\cdots,N_t}$ is a GP and because $\mathbf{\Gamma}$ is deterministic then $Z(\mathbf{x}, t_u)$ defined as a linear combination of $(A_i(\mathbf{x}))_{\mathbf{x} \in Q, i=1,\cdots,N}$ is a GP. Knowing the data, we set $\mathbf{\Gamma} = \gamma$. To predict $Z(\mathbf{x}, t_u)$ the GP regression is

used on A_i knowing $A_i(\mathbf{x}^{(j)}) = a_i(\mathbf{x}^{(j)})$ for all i.

We need to truncate in order to reduce the dimension. A discussion on this truncation is proposed on chapter 6. A surrogate model on a SVD basis is used in (Nanty, Helbert, Marrel, Pérot, & Prieur, 2017).

Some limits:

- The basis is based on data. However, the uncertainty on the basis is not taken into account. In the case of using the SVD for regression, the question of uncertainty about the basis comes up.
- The truncation introduces error in the reconstruction. This error depends on the learning data and can change depending on the learning set.

2.1.2 Independent component analysis

We consider $z(\mathbf{x}, t_u)$ the output of the calculation code. It is model by $Z(\mathbf{x}, t_u)$. The aim of the PCA is to build a $\{\mathbf{\Gamma}\}$ basis such that :

$$Z(\mathbf{x}, t_u) = \sum_{i=1}^{N} A_i(\mathbf{x})\Gamma_i(t_u), \tag{2.8}$$

with $A_i(\mathbf{x})$ independent coefficients. In the case where the coefficients A_i are Gaussian, an ICA can be performed by SVD. However, it may be interesting to have independent coefficients $A_i(\mathbf{x})$ for more complex modeling. This is exactly what is proposed by Independent Component Analysis (ICA).

Unlike the SVD method, ICA is a class of algorithm and not an algorithm. The most known method of ICA is Infomax (Bell & Sejnowski, 1995), FastICA (Hyvärinen & Oja, 2000) and Kernel ICA (Bach & Jordan, 2002).

ICA is not a method but more a class of methods. In the following two subsections we present two methods that are related to ICA. As such it is very difficult to conclude on the advantages and disadvantages. However, compared to SVD, the quantification of uncertainty is more complicated to assess for most of the ICA methods. We do not use these methods for our production.

2.1.3 Kernel PCA

In this subsection we want to reduce the dimension of N_t. We want to estimate the function $z(\mathbf{x}, t_u)$ output of the code with $\mathbf{x} \in Q$ and $u = 1, \cdots, N_t$ by a function $Z(\mathbf{x}, t'_j)$ with $j = 1, \cdots, N'_j$ with $N'_j < N_t$.

In the field of multivariate statistics, kernel principal component analysis (Schölkopf, Smola, & Müller, 1998) is an extension of PCA using the techniques of kernel methods. Kernel PCA involves projecting the data through a non-linear application into a high-dimensional space known as the feature space where the linear PCA is applied. This method is based on the kernel trick.

Kernel trick The kernel trick is to replace a scalar product in a high-dimensional space with a kernel function, which is easy to calculate. A linear classifier can easily be transformed into a non-linear classifier with this method. Another advantage of kernel functions is that the transformation to high-dimensional space does not need to be explicit. This method is proposed in (Boser, Guyon, & Vapnik, 1992).

This paragraph was written with the help of (Schölkopf, Burges, & Smola, 1999, chapter 20). The methods start by choosing a kernel k that satisfies the Mercer's conditions, i.e. the kernel k is continuous symmetric and positive semi-definite. We will work on the discretized kernel, as this is the case we are interested in for the implementation. The considered points are t_u, with $u = 1, \cdots, N_t$. Then the kernel matrix $K = \{k(t_u, t_v)\}_{u,v=1}^{N_t}$ is computed using the input variables t. K is symmetric positive semi-definite. We can diagonalize K and get an eigendecomposition. The principal component at a point t is extracted using projection onto the eigenvectors of K in the reduced space. This means that at each step a PCA is made close to the t point. The condition on the kernel gives that this procedure corresponds to the PCA in large dimension.

The major problem with this method is the inability to have an inverse transform. This method is difficult to use for our regression problem. The uncertainty quantification of the reconstructed basis is also an issue for the problem we want to solve. This type of technique is not used in the following, although there are some resinous papers that exploit it, see (T. Zhou & Peng, 2020).

2.1.4 Autoencoders

An autoencoder is an artificial neural network used for unsupervised learning of discriminative features. This method was proposed in (Kramer, 1991).

The purpose of an autoencoder is to learn a representation, or encoding, of a data set, usually with the aim of reducing the dimensionality of that set.

The simplest form of autoencoder is a forward propagating neural network, very similar to the multilayer perceptron. The autoencoder has an input layer $\mathbf{x}$, an output layer $\hat{\mathbf{x}}$ and one or more hidden layers connecting them. The output layer has the same number of nodes as the input layer; in other words, $\mathbf{x}$ and $\hat{\mathbf{x}}$ have the same dimension. The innermost hidden layer is denoted $\tilde{\mathbf{x}}$. $\tilde{\mathbf{x}}$ is called latent variables or latent representation. It represents $\mathbf{x}$ but in a space with reduced dimension. An illustration of an autoencoder is given in fig. 2.2.

There is another form of autoencoder called variational autoencoder, see (Kingma & Welling, 2014). In a variational autoencoder, the input data is sampled from a parameterized distribution. The encoder and decoder are jointly driven so that the output minimizes a reconstruction error between the parametric posterior and the observation.

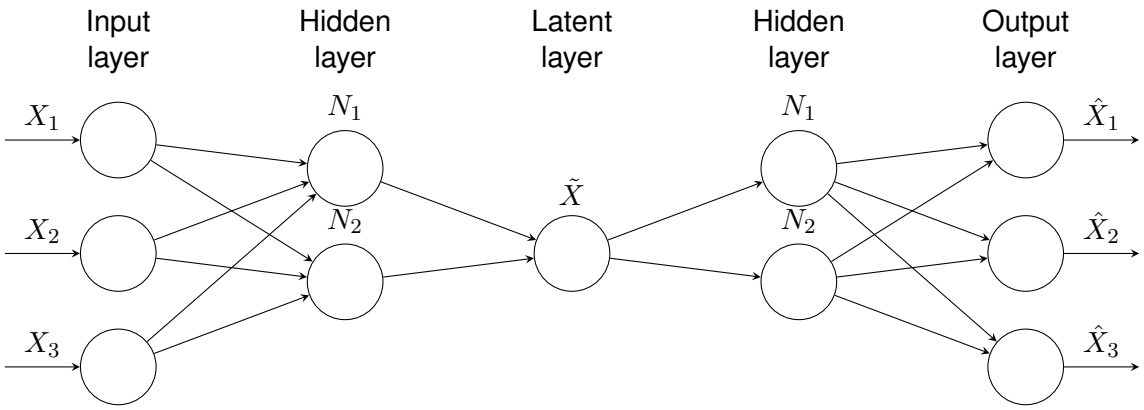


Figure 2.2: Autoencoders with input dimension 3 and laten space 1.

2.2 Fourier and Wavelet transform

The Fourier transform is widely used in many areas of mathematics. In particular, it is used to solve differential equations. Fourier transforms are also used in signal processing and in a large number of numerical methods. This representation is also very important in understanding multiresolution analysis. The wavelet transform should allow us to achieve a different dimension reduction compared to what is proposed by other methods.

In this section we will explain the Fourier transform and its main properties. Then the wavelet transform will be presented. We will focus on the Daubechies wavelets and mainly the Haar case.

2.2.1 Fourier transform

The key idea of Joseph Fourier was to represent periodic functions as series of harmonically related sinusoids. The first application of this groundbreaking representation was to solve the heat equation.

Definition 2.2.1. *The Fourier transform of an integrable function* $\mathrm{g} : \mathbb{R}^n \to \mathbb{C}$ *is the function* $\hat{\mathrm{g}} : \mathbb{R}^n \to \mathbb{C}$*, such that:*

$$\hat{\mathrm{g}}(\xi) = \int_{\mathbb{R}^n} \mathrm{g}(\mathbf{x}) e^{-i<\mathbf{x},\xi>} \mathrm{d}x, \tag{2.9}$$

with $< \cdot, \cdot >$ *the scalar product.*

If the Fourier transform is integrable, an inverse Fourier transform exists:

$$\mathrm{g}(\mathbf{x}) = \frac{1}{(2\pi)^n} \int_{\mathbb{R}^n} \hat{\mathrm{g}}(\xi) e^{i<\mathbf{x},\xi>} \mathrm{d}\xi, \tag{2.10}$$

There are different Fourier transform conventions. We have chosen this convention, called angular frequency, non-unitary.

For the following we consider that the dimension is 1.

Properties 2.2.2. *Let f and g be two integrable functions and denote by $\hat{f}$ and $\hat{g}$ their Fourier transform. Let $a \in \mathbb{R}$ and $b \in \mathbb{R}$. We denote by $\mathcal{F}$ the Fourier transform operator.*

- *The Fourier transform is linear:* $\mathcal{F}\left[af(x) + bg(x)\right](\xi) = a\hat{f}(\xi) + b\hat{g}(\xi)$
- *Shift in the "time" domain gives:* $\mathcal{F}\left[f(x-a)\right] = e^{-ia\xi}\hat{f}(\xi)$
- *Shift in the "frequency" domain gives:* $\mathcal{F}\left[f(x)e^{iax}\right] = \hat{f}(\xi - a)$
- *Scaling in the "time" domain gives:* $\mathcal{F}\left[f(ax)\right] = \frac{1}{|a|}\hat{f}\left(\frac{\xi}{a}\right)$

This is a non-exhaustive list of the properties of the Fourier transform. For a number of common functions there is an analytical expression for their Fourier transform.

The problem with the use of the Fourier transform is that it does not allow the phenomena to be positioned in the so-called temporal space. The so-called temporal space is the space of the original function before the transformation.

One idea is to do the Fourier transform on a reduced window. Thanks to this we can have the instantaneous frequency of the signal. This is called the short term Fourier transform.

Definition 2.2.3. *We define S_f the Short Term Fourier Transform of the function f on the window g :*

$$S_f(u,\xi) = \int f(x)g(x-u)e^{-i\xi x}\mathrm{d}x. \tag{2.11}$$

The spectrogram is defined as $|S_f(u,\xi)|^2$. g is a window function that means that it is a function with compact support on a closed interval.

The spectrogram makes it possible to estimate the frequency of a function over a short period. But the accuracy of the frequency estimation is limited by the Weyl-Heisenberg theorem. There is a trade-off between frequency estimation and temporal estimation.

Theorem 2.2.4. *(Weyl-Heisenberg inequality) Assume that f, $xf(x)$ and $\xi\hat{f}(\xi)$ are square integrable. The inequality is:*

$$\left(\int x^2|f(x)|^2\mathrm{d}x\right)\left(\int \xi^2|\hat{f}(\xi)|^2\mathrm{d}\xi\right) \geq \frac{1}{2}\left(\int |f(x)|^2\mathrm{d}x\right)^2. \tag{2.12}$$

This means that there is a trade-off between frequency and position in the input space.

The Fourier transform is not strictly speaking a dimension reduction. However, it is possible to have a regression done in the Fourier domain. In high dimensions, one method may be to truncate the coefficients. The problem is complicated because it depends on the application.

2.2.2 Wavelet transform

Wavelets are proposed by Jean Morlet and Alex Grossmann and used for applications in (Kronland-Martinet, Morlet, & Grossmann, 1987). An introduction to the wavelet transform is given in (S. G. Mallat, 2009).

With all the methods proposed above, the idea is to express the data on a space (depending on the method) which simplifies the expression of the data, and it is thus possible to reduce the dimension. These dimension reduction methods are known to be effective especially for image compression, see (Calderbank, Daubechies, Sweldens, & Yeo, 1997). For wavelet dimension reduction the idea is different. We try to build several subspaces which will allow us to reduce the dimension.

2.2.2.1 Multi-resolution analysis and wavelet function

We start by introducing multi-resolution analysis.

Definition 2.2.5. *A multi-resolution analysis of $L^2(\mathbb{R})$ is a sequence of nested subspaces:*

$$\{0\} \subset \cdots \subset V_1 \subset V_0 \subset V_{-1} \subset \cdots \subset V_{-n} \subset V_{-(n+1)} \subset \cdots \subset L^2(\mathbb{R}) \tag{2.13}$$

that satisfies self-similarity in space, self-similarity in scale, completeness and regularity relations.

The development of multi-resolution in the context of signal processing has been firstly proposed in (S. Mallat, 1989). The link with wavelet is presented in the same paper.

The properties that should satisfy the multi-resolution, for $k \in \mathbb{Z}$ are:

- Self-similarity in space: each subspace V_k is invariant under shifts by integer multiples of 2^k. That is, for each $f \in V_k,\ m \in \mathbb{Z}$ the function g defined as $g(x) = f(x - m2^k)$ also belongs to V_k.
- Self-similarity in scale: all subspaces $V_k \subset V_l,\ k > l$, are time-scaled versions of each other, with scaling dilation factor 2^{k-l}. i.e., for each $f \in V_k$ there is a $g \in V_l$ with $\forall x \in \mathbb{R} :\ g(x) = f(2^{k-l}x)$.
- The space resolution of a subspace is the length of the time sequence associate. In the sequence of subspaces, for $k > l$ the space resolution 2^l of the l^{th} subspace is higher than the resolution 2^k of the k^{th} subspace.
- Regularity: the model subspaces V_0 is generated as the linear hull of the integer shifts of one or a finite number of generating functions ϕ or $\phi_1, \ldots, \phi_r$.
- Completeness: $\cup_k V_k$ should be dense in $L^2(\mathbb{R})$, and $\cap_k V_k$ should only contain the zero element. Those integer shifts should form a frame for the subspace $V_0 \subset L^2(\mathbb{R})$.

A multiresolution decomposition is characterized by the scaling function ϕ. The function ϕ generates an orthogonal basis of each space V_j. We then can write a scaling equation because $V_1 \subset V_0$, $2^{-1/2}\phi(t/2) \in V_1$ and $\{\phi(t-n)\}_{n\in\mathbb{Z}}$ is an orthogonal basis of V_0. The decomposition is:

$$\frac{1}{\sqrt{2}}\phi(\frac{t}{2}) = \sum_{n\in\mathbb{Z}} h[n]\phi(t-n), \tag{2.14}$$

with h a discrete filter called conjugate mirror filter, and $h[n] = \langle \frac{1}{\sqrt{2}}\phi(\frac{t}{2}), \phi(t-n)\rangle$.

Orthogonal wavelet subspaces carry the details that increase the resolution of a signal. We have that $V_j \subset V_{j-1}$. For the multi-resolution analysis $\{V_j\}$ the orthogonal complement W_j of V_j to V_{j+1}, we write this $V_j = V_{j+1} \bigoplus W_{j+1}$. The orthogonal projection of f on V_{j-1} can be expressed as the sum of two orthogonal projections on V_j and W_j. (S. G. Mallat, 2009, theorem 7.3) proves that an orthogonal basis of W_j can be constructed by scaling and translating a wavelet ψ. $\{\psi_{j,k}\}_{k\in\mathbb{Z}}$ is an orthogonal basis of W_j. As for ϕ and h we can associate a discrete filter to ψ: $g[n] = \langle \frac{1}{\sqrt{2}}\psi(\frac{t}{2}), \phi(t-n)\rangle$. We will use the definition of the family $\{\psi_{j,k}, j \in \mathbb{Z}, k \in \mathbb{Z}\}$ and $\{\phi_{j,k}, j \in \mathbb{Z}, k \in \mathbb{Z}\}$ of translated and dilated functions:

$$\psi_{j,k}(t) = 2^{-\frac{j}{2}}\psi\left(2^{-j}t - k\right), \tag{2.15}$$

and

$$\phi_{j,k}(t) = 2^{-\frac{j}{2}}\phi\left(2^{-j}t - k\right). \tag{2.16}$$

For any square integrable function $f : \mathbb{R} \mapsto \mathbb{R}$, the wavelet coefficients $W^f_{j,k}$ are defined by:

$$W^f_{j,k} = \int f(t)\psi_{j,k}(t)\mathrm{d}t \quad j \geq 0, k \in \mathbb{Z} \tag{2.17}$$

2.2.2.2 Orthogonal wavelet

We introduce the orthogonal wavelet because the orthogonal framework is efficient for regression.

Definition 2.2.6. *Orthogonal wavelet An orthogonal wavelet is a wavelet of which associated wavelet transform is orthogonal. This means that the inverse wavelet transform is the adjoint of the wavelet transform.*

Assuming that the scaling function ϕ is of compact support and with orthogonal shifts this means that ϕ is a refinable function. This means that it satisfies the refinement equation. The refinement equation is:

$$\phi(t) = \sum_{k=-N}^{N} a_k\phi(2t-k), \tag{2.18}$$

with $(a_i)_{i=-N}^{N}$ the scaling sequence. The same sum can be computed for the wavelet function ψ:

$$\psi(t) = \sum_{k=-N}^{N-1} (-1)^k a_{1-k} \phi(2t-k). \tag{2.19}$$

Consequently, $\left\{\psi_{k,j}(t) = 2^{-j/2}\psi(2^{-j}t-k) : k,j \in \mathbb{Z}\right\}$ is a countable complete orthogonal wavelet basis in $L^2(\mathbb{R})$.

Necessary condition for orthogonality: The wavelet ψ is necessarily orthogonal if the scaling sequence $(a_i)_{i=-N}^{N}$ is orthogonal to any shifts of itself by an even number of coefficients. This is translated in equation by:

$$\sum_{n\in\mathbb{Z}} a_n a_{n+2m} = 2\delta_{m,0}, \tag{2.20}$$

where $\delta_{i,j}$ is the Kronecker delta.

Vanishing moments

Definition 2.2.7. *Vanishing moments ψ has p vanishing moments if $0 \leq k < p$:*

$$\int t^k \psi(t) \mathrm{d}t = 0. \tag{2.21}$$

This means that ψ is orthogonal to any polynomial of degree $p-1$.

Theorem 2.2.8. *If ψ is an orthogonal wavelet with p vanishing moments, then it has a support of size larger than or equal to $2p-1$.*

Proof. The proof of this theorem is given in (Daubechies, 1988). The proof uses properties of the discrete wavelet transform and in particular the size of its scale sequence. □

2.2.2.3 Discrete wavelet transform algorithm

As for Fast Fourier Transform we need a way to compute the wavelet transform for a discretized function. There is a continuous function $y(t)$ associated with the discretized function $y[n]$ for $n \in [0, N_t - 1]$. We define $y[n] = \int_n^{(n+1)} y(t)dt$. N_t is the number of sample for the discrete function. For simplicity, we assume that N_t is a power of 2. If this is not the case we can either interpolate with sub-sampling or use the 0 padding method. The Discrete wavelet transform (DWT) is the adaptation of the wavelet transform to the case of a discrete signal $y[n]$. The DWT consists of computing a reduced number of coefficients $W_{j,k}^y$ for specific values of j,k. If a discretized procedure of the previous definitions was used the computation time would be long, then an accelerated procedure has been proposed.

The DWT is an iterative procedure. The initialization of the method is that $a_0[n] = y[n]\ \forall n \in [0, N_t - 1]$. At each iteration we build two sequences, the scale sequence $(a_j[n])_n$, and the detail sequence $(d_j[n])_n$ with $j \in [1, M]$ with $2^M = N_t$. The detail coefficients are the wavelet coefficients at the $j + 1$ level. This means that $W_{j,k} = d_j[k]$.

In order to calculate the coefficients a filtering is used which makes the procedure extremely fast. The discrete wavelet decomposition can be done using two filters: a low-pass G filter and a high-pass H filter. They are the discrete Fourier transforms of the conjugate mirror filters g and h.

After applying the filters we use a downsampling operator. Thus, in the case of orthogonal wavelets the dimension is exactly the same in input and output. The downsampling operator is denoted $\downarrow 2$ in fig. 2.3. Half of the points are downsampled.

This decomposition is repeated to increase the size of the detail coefficients and decrease the size of the scale coefficients. The dimension of a_j is decreased until it is no longer possible to downsample it. For a starting sequence of size $N_t = 2^M$ we will have M iterations of the algorithm. At the output of the DWT we have M lists of decreasing size from 2^{M-1} to 1 for the detail functions plus a sequence of size 1 for the scaling function.

The inverse wavelet transform can be done by reversing the procedure. As the wavelet basis is orthogonal then the number of element in the wavelet domain is the same as in the time domain. This procedure is not by itself a dimension reduction because the dimension of the DWT is exactly the dimension of the original signal.

In (S. Mallat, 1989, sec. 7.3), an algorithm is proposed to implement the discrete wavelet transform. This algorithm is filter-based in order to be fast. The illustration of this algorithm is given on fig. 2.3. General framework for implementing wavelet transform efficiently is available in (Rioul & Duhamel, 1992).

To calculate the coefficients $d_{j+1}[n]$ and $a_{j+1}[n]$ we need to convert the wavelet function ψ into two filters h and g. We have $a_j[n] = \langle f, \phi_{j,n} \rangle$ and $d_j[n] = \langle f, \psi_{j,n} \rangle$. Following relations hold true:

$$a_{j+1}[2n] = \langle f, \phi_{j+1,2n} \rangle = a_j \star \bar{h}[4n], \tag{2.22}$$

with $\bar{h}[n] = h[-n]$ and $\star$ the convolution. The same for d_{j+1} gives:

$$d_{j+1}[2n] = \langle f, \psi_{j+1,2n} \rangle = a_j \star \bar{g}[4n]. \tag{2.23}$$

It is then possible to express g and h as functions of ψ and ϕ:

$$g[n - 2p] = \langle \psi_{j+1,p}, \phi_{j,n} \rangle, \tag{2.24}$$

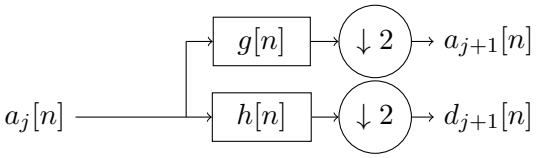


Figure 2.3: Illustration of an iteration of the fast wavelet transform algorithm. $\downarrow$ is the downsampling operator, g is the low-pass filter and h the high-pass filter associated with the discrete wavelet transform.

and

$$h[n-2p] = \langle \phi_{j+1,p}, \phi_{j,n} \rangle, \tag{2.25}$$

with $n \in \mathbb{Z}$ and $p \in \mathbb{Z}$.

The discrete implementation of wavelet can be seen as computing the wavelet coefficients of a wavelet $\psi_{j,k}$. We consider the wavelet coefficient $W^y_{j,k}$. The coefficient j is fixed, i.e. we process the study at a certain scale j. Then we identify the operation,

$$W^y_{j,k} = \int y(t) 2^{-\frac{j}{2}} \psi\left(2^{-j}t - k\right) \mathrm{d}t, \tag{2.26}$$

to a convolution of y. We see the convolution is sampled at points $1, 2^j, 2^{2j}, \cdots, 2^M$. We then use the mirror filter to compute the fast wavelet transform algorithm.

Example: The Haar wavelet can easily be computed with the discretized framework. The Haar function is:

$$\psi(t) = \begin{cases} 1 & \text{if } \ 0 \leq t < 1 \\ -1 & \text{if } \ 1/2 \leq t < 1 \\ 0 & \text{otherwise} \end{cases} . \tag{2.27}$$

Then the associate discrete filters are $h[n] = \frac{\sqrt{2}}{2}[-1, 1]$ and $g[n] = \frac{\sqrt{2}}{2}[1, 1]$.

2.2.2.4 Daubechies Wavelets

Daubechies wavelets are a specific class of wavelets function introduced in (Daubechies, 1988). We need expressions of ϕ and ψ for this class of wavelets.This gi It is interesting that these functions have certain properties. The first of these is that the functions are compactly supported. The Daubechies wavelets maximize the number of vanishing moments of a given support. Having many vanishing moments is interesting for representing functions and in particular polynomials. With m vanishing moments we easily encode polynomials of degree $m-1$. The number of vanishing moments generally gives the name of the Daubechies wavelets. The Daubechies wavelets that has m vanishing moments is called db$m$ with $m \in \mathbb{N}^*$. In fig. 2.4 we present a plot of different Daubechies wavelets.

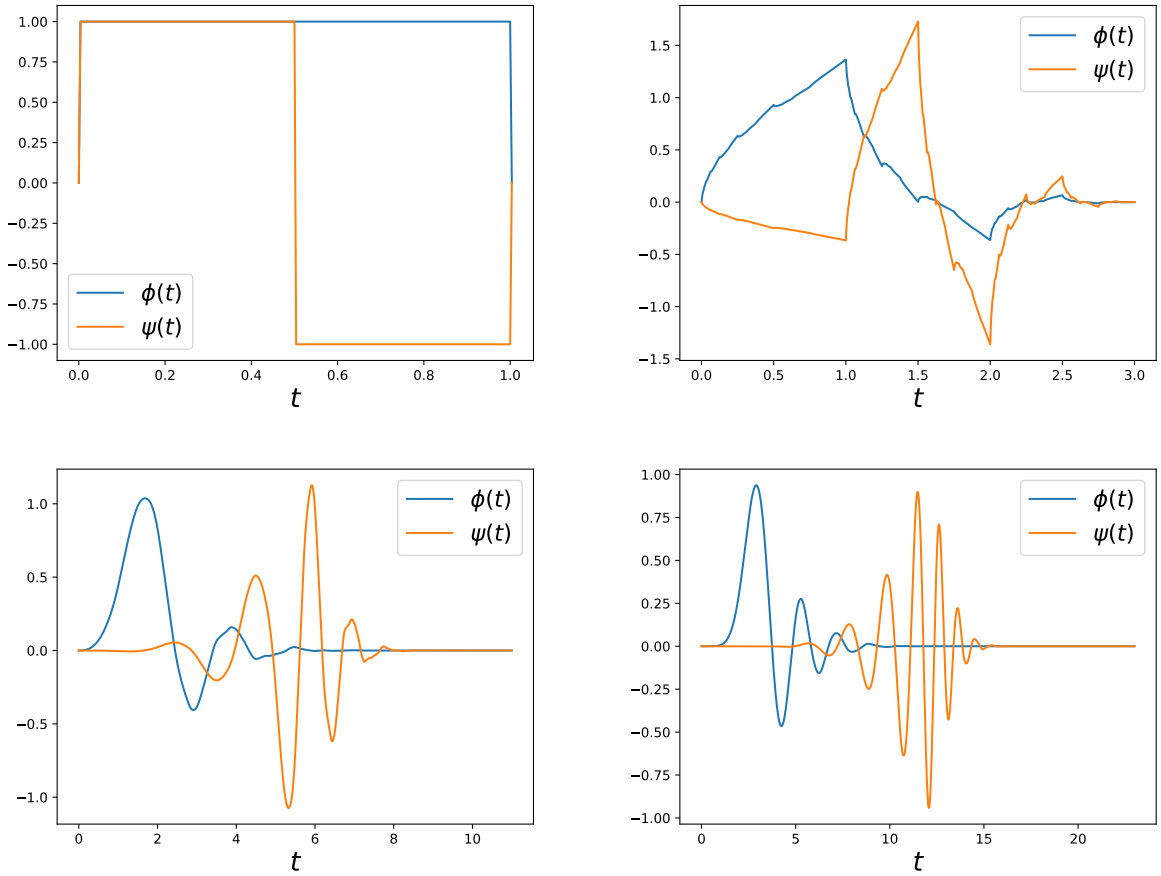


Figure 2.4: Representation of the Daubechies wavelets for $1, 2, 6$ and 12 vanishing moments.

Definition 2.2.9. *(Daubechies wavelet) The Daubechies wavelet of the order m is an orthogonal wavelet defining a discrete wavelet transform and characterized by m vanishing moments and support (number of coefficients) $2m$. It has the minimal support amongst all orthogonal wavelets with m vanishing moments.*

The specific case of the Haar wavelet The Haar basis is easy to understand. The functions of this basis are piecewise-constant. This is why it was introduced first. In this case the Haar basis was used by Alfréd Haar in 1910, even if the formalization came later. The construction of an orthogonal basis with compact support is the key element. In general the Haar basis is defined as a particular Daubechies basis. We will study it because it has two advantages: its Fourier transform has a simple analytical expression and it is the simplest. This simplicity allows us to have a passage from continuous to discrete space easily. In the classification of Daubechies wavelets the Haar wavelet is called db1.

Remark 2.2.10. *We consider the wavelet function db1. The number of vanishing moments is then 1. Then $N = 2$ that means that we will only have 2 coefficients for the scale sequence and the wavelet sequence. The expression of h and g can be then defined as $h[n] = \frac{\sqrt{2}}{2}[-1, 1]$ and $g[n] = \frac{\sqrt{2}}{2}[1, 1]$.*

The Haar basis is the only orthogonal basis of compactly supported wavelets for which the associated averaging function ϕ has a symmetry axis. The Fourier transform is,

$$\hat{\psi}(\xi) = -\frac{i}{\xi}\left[1 - \exp\left(-\frac{i\xi}{2}\right)\right]^2 = ie^{-i\xi/2}\sin(\xi/4)\,\mathrm{sinc}(\xi/4) \tag{2.28}$$

with $\mathrm{sinc}(x) = \frac{\sin(x)}{x}$ for $x \neq 0$ and $\mathrm{sinc}(0) = 1$.

2.2.2.5 Conversion to continuous Wavelets

When we calculate the discrete wavelet, we see that the coefficients are different depending on the sampling. To avoid this, it is possible to use a local filtering method. In this paragraph we present this method. We start with the expression in the continuous case and then the expression in the discrete case. We will focus here on the Haar transform.

Continuous case Let $(f(t))_{t\in[0,2^M]}$ denote the function for which we want to compute the Haar wavelet coefficients. We denote the approximation coefficients at level zero by

$$a_0 = \big(a_0(k)\big)_{k=0}^{2^M-1}, \tag{2.29}$$

which are defined by

$$a_0(k) = \int_k^{k+1} f(t)dt, \quad \text{for} \quad k = 0, 1, \ldots, 2^M - 1. \tag{2.30}$$

Given this level zero representation, the data, we construct successively their wavelet coefficients with respect to the Haar basis as follows. Let

$$a_1(k) = \frac{1}{\sqrt{2}}(a_0(2k) + a_0(2k+1)), \tag{2.31}$$

$$W^f_{1,k} = \frac{1}{\sqrt{2}}(a_0(2k) - a_0(2k+1)), \quad \text{for} \quad k = 0, 1, \ldots, 2^{M-1} - 1, \tag{2.32}$$

be the smoothed signal and its fluctuation, or detail, at the finest scale. This process of averaging and differencing can be continued by defining

$$a_j(k) = \frac{1}{\sqrt{2}}(a_{j-1}(2k) + a_{j-1}(2k+1)), \tag{2.33}$$

$$W^f_{j,k} = \frac{1}{\sqrt{2}}(a_{j-1}(2k) - a_{j-1}(2k+1)), \quad \text{for} \quad k = 0, 1, \ldots, 2^{M-j} - 1, \tag{2.34}$$

for $j = 1, \ldots, M$. The data vector a_0 can then be reconstructed from $\left(a_M, \mathbf{W}^f_M, \mathbf{W}^f_{M-1}, \ldots, \mathbf{W}^f_1\right)$ with $\mathbf{W}^f_j = (W^f_{j,k})_{k=0}^{2^{M-j}-1}$ since from equations (2.31-2.32) we have

$$a_0(2k) = \frac{1}{\sqrt{2}}(a_1(k) + W_{1,k}), \tag{2.35}$$

$$a_0(2k+1) = \frac{1}{\sqrt{2}}(a_1(k) - W_{1,k}), \quad \text{for} \quad k = 0, 1, \ldots, 2^{M-1} - 1, \tag{2.36}$$

and now a_1 can be replaced by sums and differences of a_2 and W_2, etc, so that we have

$$a_j(2k) = \frac{1}{\sqrt{2}}(a_{j+1}(k) + W^f_{j+1,k}), \tag{2.37}$$

$$a_j(2k+1) = \frac{1}{\sqrt{2}}(a_{j+1}(k) - W^f_{j+1,k}), \quad \text{for} \quad k = 0, 1, \ldots, 2^{M-1-j} - 1, \tag{2.38}$$

for $j = M-1, \ldots, 0$.

The detail coefficients at level j can alternatively be expressed as

$$W^f_{j,k} = 2^{-j/2} \int_{-\infty}^{\infty} \psi\big(t2^{-j} - k\big) f(t) dt, \tag{2.39}$$

with the mother wavelet defined by

$$\psi(t) = \begin{cases} 1 & \text{if} \quad 0 \le t < 1 \\ -1 & \text{if} \quad 1/2 \le t < 1 \\ 0 & \text{otherwise} \end{cases} . \tag{2.40}$$

and the average is:

$$a_M = 2^{-M/2} \sum_{k=0}^{2^M-1} a_0(k) = 2^{-M/2} \int_0^{2^M} f(t)dt. \tag{2.41}$$

The detail coefficients correspond to probing the function at different scales and different 'times', with k representing time and j scale. Let us assume that $(f(t))_t$ is a random process. Then the vector $\left(a^M, \mathbf{W}_M^f, \cdots, \mathbf{W}_1^f\right)$ is a random vector. If the process f is stationary, then the vector $\mathbf{W}_j^f = (W_{j,k}^f)_{k=0}^{2^{M-j}-1}$ is stationary for any $j = 1, \cdots, M$. We define the definition set I of points (j,k). From the expression of $\mathbf{W}_j^f$ we have:

$$I = \left\{j,k | j = 1, \cdots, M \ \ \& \ \ k = 0, \cdots, 2^{M-j} - 1\right\}. \tag{2.42}$$

I is a vector of $2^M - 1$ elements, and with the addition of a^M, $\left(a^M, \mathbf{W}_M^f, \cdots, \mathbf{W}_1^f\right)$ is 2^M-dimensional.

Discrete case We now assume that we observe a discretized version of the process $(f(t))_{t\in[0,2^M]}$:

$$f_n^{(\mathrm{d})} = f\big(\frac{n}{2^N}\big), \quad n = 0, 1, \ldots, 2^{M+N} - 1.$$

We introduce the approximation coefficients at level zero:

$$a_0^{(d)} = \big(a_0^{(d)}(k)\big)_{k=0}^{2^M-1}, \tag{2.43}$$

as

$$a_0^{(d)}(k) = 2^{-N} \sum_{n=k2^N}^{(k+1)2^N-1} f_n^{(d)}, \ \text{ for } \ k = 0, 1, \ldots, 2^M - 1. \tag{2.44}$$

Given this level zero representation, the data, we construct successively their wavelet coefficients $\mathbf{W}_j^{(d)}$ with respect to the Haar basis as described in the previous section. We are only interested in the data $a_0^{(d)}$, we will not look for detailed information below this scale.

If the process f is zero-mean Gaussian, then the joint vector $\left(a_M^{(d)}, \mathbf{W}_M^{(d)}, \mathbf{W}_{M-1}^{(d)}, \ldots, \mathbf{W}_1^{(d)}\right)$ is zero-mean Gaussian and its covariance matrix is equal (when N is large enough) to the one obtained in the continuous case (in practice, N does not need to be very large if the correlation 'length' of the process f is of order one or larger, $N = 3$ or 4 should be sufficient).

2.2.2.6 Wavelet and dimension reduction

Wavelet transforms are widely used for data compression. In particular, the 2D wavelet transform is used for image compression, see (Lewis & Knowles, 1992; Usevitch, 2001). The methods presented in these articles are based on quantization, which can be viewed as a process of mapping the input values of a large set to the output values of a smaller set. The quantization in wavelet domain is to neglect the low

values so that a smaller number of points can be used to reconstruct the same image. These methods are used for compression by quantizing the wavelet transform. The whole method is based on the fact that such a transform is less visible than the same quantization but on the image directly.

It should also be noted that there are lossless compression methods as in (Calderbank et al., 1997). These lossless compression methods cannot be used for regression. The compressed output is not identifiable, and it is therefore hard to make a prediction.

There are also regression methods using wavelets such as (Amato, Antoniadis, & De Feis, 2006). These methods are rarely used for regression and are case dependent.

Chapter 3

Multi-fidelity regression

The objective of this chapter is to present the different methods for emulating hierarchical multi-fidelity data, called multi-fidelity regression. Hierarchical multi-fidelity data are data organized in several sets that can be classified according to their computational cost and their accuracy. The most accurate data is naturally the most expensive to obtain. For example, data can be obtained from different versions of computational codes simulating the same phenomenon but with different computational complexities.

In this manuscript, we assume that the data is the output of a computational code, although the methods presented can be used for any hierarchical multi-fidelity data. The objective is then to build a multi-fidelity surrogate model of the output of the most expensive version of the computational code while being able to use the information obtained from the faster versions. Multi-fidelity regression can be performed using different methods.

The historical method, called AR(1) multi-fidelity Gaussian process regression, presented in section 3.1.1, has its origins in the work of Kennedy and O'Hagan in (Kennedy & O'Hagan, 2000). This method is effective when the low-fidelity models can capture the right trends and the outputs of the low- and high-fidelity models have a strong linear correlation. The method ignores low-fidelity data when the relationship between the low- and high-fidelity models is not linear. This means that even if the low-fidelity data do not provide information to build the high-fidelity model, the high-fidelity model will not be degraded because we will be in a single-fidelity Gaussian process regression framework presented in section 1.1 of chapter 1. It is also possible to aggregate multi-fidelity models, as proposed in (De Iozzo, 2013, ch. 6).

With the advent of Deep Gaussian processes (DeepGPs), presented in section 3.1.2, it has become possible to relax the autoregressive condition of the model. Thus, there are successful models even in the case of non-linear relationships between low- and high-fidelity models, see (Cutajar, Pullin, Damianou, Lawrence, & González, 2019; Perdikaris, Raissi, Damianou, Lawrence, & Karniadakis, 2017).

There are also competing methods using neural networks (NN), presented in section 1.2 to solve the same type of problem. These approaches are presented in section 3.2. Although they are not known to be efficient for small data sizes (the number of training points is only slightly larger than the effective

dimension of the problem), the modularity and simplicity of NNs make multi-fidelity NNs methods very attractive.

Multi-fidelity polynomial chaos expansion (MF-PCE) can also be exploited as in (Ng & Eldred, 2012) for multi-fidelity regression. MF-PCEs are mainly used for sensitivity analysis, see (Palar, Zuhal, Shimoyama, & Tsuchiya, 2018).This approach will not be presented, because this manuscript focuses on Gaussian processes.

Before starting this chapter, two caveats should be made. The first is that multi-fidelity regression is a form of transfer learning, but it is not exactly the same thing. In the case of multi-fidelity, it is assumed that data of different fidelities emulate the same phenomenon. This is not the case with transfer learning. One example can be found in (Zhuang et al., 2021). An example of transfer learning is to use a neural network pre-trained on a first set of images to learn a new set of images. The second caveat is that the word "multi-fidelity" is also used in the context of Monte Carlo methods. The problem is then different because it is not a question of building a surrogate model but of accelerating the convergence of the methods. This Monte-Carlo method is generally called multi-level Monte-Carlo method, see (Patsialis & Taflanidis, 2021). These two aspects, transfer learning and Monte Carlo methods, will not be discussed in this manuscript.

In order to use multi-fidelity in real cases, it is often necessary to adapt uncertainty quantification tools to this context. Work on experimental design, (Jakeman et al., 2022), and sensitivity analysis has been done, (Le Gratiet, Marelli, & Sudret, 2015), for example. Multi-fidelity is increasingly used in real applications. For example, (Abdallah, Lataniotis, & Sudret, 2017; Babaee, Bastidas, DeFilippo, Chryssostomidis, & Karniadakis, 2020) use these methods for weather prediction and for wind turbine design.

3.1 Multi-fidelity regression with Gaussian Processes

In this section we present classical multi-fidelity regression methods using Gaussian processes. These methods are the extension of chapter 1 to the multi-fidelity framework. For this purpose we can distinguish two methods, the autoregressive (AR(1)) model and the Deep-Gaussian processes.

3.1.1 GP regression with autoregressive (AR(1)) model

In this section we want to build a surrogate model of a code $\alpha_{\mathsf{H}}(\mathbf{x})$ whose input $\mathbf{x}$ is in $Q \subset \mathbb{R}^d$ and whose scalar output is in $\mathbb{R}$. The construction of a surrogate model for complex computer code is difficult because of the lack of available output data. We consider the situation in which a cheaper and approximate code $\alpha_{\mathsf{L}}(\mathbf{x})$ is available. In this section, we apply the regression method presented by (Kennedy & O'Hagan, 2000), reviewed in (Forrester, Sóbester, & Keane, 2007) and improved in (Le Gratiet & Garnier, 2013).

3.1.1.1 The model

We model the prior knowledge of the code output $(\alpha_{\mathsf{L}}, \alpha_{\mathsf{H}})$ as a Gaussian process $(A_{\mathsf{L}}, A_{\mathsf{H}})$. The vector containing the values of $\alpha_{\mathsf{F}}(\mathbf{x})$ at the points of the experimental design D_{F} is denoted by α^{F}, and $\mathcal{A}^{\mathsf{F}}$ is the Gaussian vector containing $A_{\mathsf{F}}(\mathbf{x})$, $\mathbf{x} \in D_{\mathsf{F}}$.The combination of $\mathcal{A}^{\mathsf{L}}$ and $\mathcal{A}^{\mathsf{H}}$ is $\mathcal{A}$. So is α, the combination of α^{L} and α^{H}. We present the recursive model of multi-fidelity introduced by (Le Gratiet & Garnier, 2013). The experimental design is constructed in such way that $D_{\mathsf{H}} \subset D_{\mathsf{L}}$. We assume the low-fidelity code is computationally cheap, and that we have access to a large experimental design for the low-fidelity code, i.e. $N_{\mathsf{L}} > N_{\mathsf{H}}$ or even $N_{\mathsf{L}} \gg N_{\mathsf{H}}$.

The article (Kennedy & O'Hagan, 2000) presents the Markov property. It can be formalized as: $\forall \mathbf{x} \in Q$, if $A_{\mathsf{L}}(\mathbf{x})$ is known, we can learn nothing more about $A_{\mathsf{H}}(\mathbf{x})$ from $A_{\mathsf{L}}(\mathbf{x}')$ for $\mathbf{x}' \neq \mathbf{x}$. This will lead us to introduce two Gaussian processes $\delta(\mathbf{x})$ and $\tilde{A}_{\mathsf{L}}(\mathbf{x})$ into the model with the $\tilde{A}_{\mathsf{L}}(\mathbf{x}) \perp \delta(\mathbf{x})$ property.

We consider the hierarchical model introduced by (Le Gratiet & Garnier, 2013):

$$\left\{ \begin{array}{rcll} A_{\mathsf{H}}(\mathbf{x}) & = & \rho_{\mathsf{L}}(\mathbf{x})\tilde{A}_{\mathsf{L}}(\mathbf{x}) + \delta(\mathbf{x}) & \\ \tilde{A}_{\mathsf{L}}(\mathbf{x}) & \perp & \delta(\mathbf{x}) & , \\ \rho_{\mathsf{L}}(\mathbf{x}) & = & g_{\mathsf{L}}^T(\mathbf{x})\beta_\rho & \end{array} \right. \tag{3.1}$$

where $\perp$ means independence, ^{T} stands for the transpose,

$$[\delta(\mathbf{x})|\beta_H, \sigma_H] \sim \mathcal{GP}\left(f_{\mathsf{H}}^T(\mathbf{x})\beta_{\mathsf{H}}, \sigma_{\mathsf{H}}^2 r_{\mathsf{H}}(\mathbf{x}, \mathbf{x}')\right), \tag{3.2}$$

and $\tilde{A}_{\mathsf{L}}(\mathbf{x})$ is a Gaussian process conditioned by the values α^{L}. Its distribution is the one of $\left[A_{\mathsf{L}}(\mathbf{x})|\mathcal{A}^{\mathsf{L}} = \alpha^{\mathsf{L}}, \beta_{\mathsf{L}}, \sigma_{\mathsf{L}}\right]$ with

$$[A_{\mathsf{L}}(\mathbf{x})|\beta_L, \sigma_L] \sim \mathcal{GP}\left(f_{\mathsf{L}}^T(\mathbf{x})\beta_{\mathsf{L}}, \sigma_{\mathsf{L}}^2 r_{\mathsf{L}}(\mathbf{x}, \mathbf{x}')\right). \tag{3.3}$$

Therefore, the distribution of $\tilde{A}_{\mathsf{L}}(\mathbf{x})$ is Gaussian, with mean $\mu_{\tilde{A}_{\mathsf{L}}}(\mathbf{x})$ and variance $\sigma^2_{\tilde{A}_{\mathsf{L}}}(\mathbf{x})$:

$$\mu_{\tilde{A}_{\mathsf{L}}}(\mathbf{x}) = f_{\mathsf{L}}^T(\mathbf{x})\beta_{\mathsf{L}} + r_{\mathsf{L}}^T(\mathbf{x})C_{\mathsf{L}}^{-1}\left(\alpha^{\mathsf{L}} - F_{\mathsf{L}}\beta_{\mathsf{L}}\right), \tag{3.4}$$

$$\sigma^2_{\tilde{A}_{\mathsf{L}}}(\mathbf{x}) = \sigma_{\mathsf{L}}^2\big(r_{\mathsf{L}}(\mathbf{x}, \mathbf{x}) - r_{\mathsf{L}}^T(\mathbf{x})C_{\mathsf{L}}^{-1}r_{\mathsf{L}}(\mathbf{x})\big). \tag{3.5}$$

Here:

- $\mathcal{GP}$ means Gaussian process,
- $g_{\mathsf{L}}(\mathbf{x})$ is a vector of q_{L} regression functions,
- $f_{\mathsf{F}}(\mathbf{x})$ are vectors of p_{F} regression functions,
- $r_{\mathsf{F}}(\mathbf{x}, \mathbf{x}')$ are correlation functions,
- β_{F} are p_{F}-dimensional vectors,
- σ^2_{F} are positive real numbers,

- β_ρ is a q-dimensional vector of adjustment parameters,
- $C_{\mathsf{F}} = \left(r_{\mathsf{F}}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})\right)_{i,j=1}^{N_{\mathsf{F}}}$ is the $N_{\mathsf{F}} \times N_{\mathsf{F}}$ correlation matrix of $\mathcal{A}^{\mathsf{F}}$,
- $r_{\mathsf{F}}(\mathbf{x}) = \left(r_{\mathsf{F}}(\mathbf{x}, \mathbf{x}^{(i)})\right)_{i=1}^{N_{\mathsf{F}}}$ is the N_{F}-dimensional vector of correlations between $A_{\mathsf{F}}(\mathbf{x})$ and $\mathcal{A}^{\mathsf{F}}$,
- F_{F} is the $N_{\mathsf{F}} \times p_{\mathsf{F}}$ matrix containing the values of $f_{\mathsf{F}}^T(\mathbf{x})$ for $\mathbf{x} \in D_{\mathsf{F}}$.

For $\mathbf{x} \in Q$, the conditional distribution of $A_{\mathsf{H}}(\mathbf{x})$ is:

$$\left[A_{\mathsf{H}}(\mathbf{x}) | \mathcal{A} = \alpha, \beta, \beta_\rho, \sigma^2\right] \sim \mathcal{N}\left(\mu_{A_{\mathsf{H}}}(\mathbf{x}), \sigma^2_{A_{\mathsf{H}}}(\mathbf{x})\right), \tag{3.6}$$

where $\beta = \left(\beta_{\mathsf{H}}^T, \beta_{\mathsf{L}}^T\right)^T$ is the $p_{\mathsf{H}} + p_{\mathsf{L}}$-dimensional vector of regression parameters, $\sigma^2 = \left(\sigma^2_{\mathsf{L}}, \sigma^2_{\mathsf{H}}\right)$ are the variance parameters,

$$\begin{aligned}\mu_{A_{\mathsf{H}}}(\mathbf{x}) =& g_{\mathsf{L}}^T(\mathbf{x})\beta_\rho \mu_{\tilde{A}_{\mathsf{L}}}(\mathbf{x}) + f_{\mathsf{H}}^T(\mathbf{x})\beta_{\mathsf{H}} \\ &+ r_{\mathsf{H}}^T(\mathbf{x}) C_{\mathsf{H}}^{-1}\left(\alpha^{\mathsf{H}} - \rho^{\mathsf{L}}(D_{\mathsf{H}}) \odot \alpha^{\mathsf{L}}(D_{\mathsf{H}}) - F_{\mathsf{H}}\beta_{\mathsf{H}}\right)\end{aligned} \tag{3.7}$$

and

$$\sigma^2_{A_{\mathsf{H}}}(\mathbf{x}) = \left(g_{\mathsf{L}}^T(\mathbf{x})\beta_\rho\right)^2 \sigma^2_{\tilde{A}_{\mathsf{L}}}(\mathbf{x}) + \sigma^2_{\mathsf{H}}\left(1 - r_{\mathsf{H}}^T(\mathbf{x}) C_{\mathsf{H}}^{-1} r_{\mathsf{H}}(\mathbf{x})\right). \tag{3.8}$$

The notation $\odot$ is the element by element matrix product. $\rho^{\mathsf{L}}(D_{\mathsf{H}})$ is the N_{H}-dimensional vector containing the values of $\rho_{\mathsf{L}}(\mathbf{x})$ for $\mathbf{x} \in D_{\mathsf{H}}$. $\alpha^{\mathsf{L}}(D_{\mathsf{H}})$ is the N_{H}-dimensional vector containing the values of $\alpha_{\mathsf{L}}(\mathbf{x})$ at the points of D_{H}.

The prior distributions of the parameters β and σ are given in section 3.1.1.2. The hyperparameters of the covariance kernels r_{L} and r_{H} can be estimated by maximum likelihood or by leave-one-out cross validation (Bachoc, 2013). The nested property of the experimental design sets $D_{\mathsf{H}} \subset D_{\mathsf{L}}$ is not necessary to build the model, but it is simpler to estimate the parameters with this assumption (Q. Zhou, Wu, Guo, Hu, & Jin, 2020). Moreover, the ranking of codes and the low computer cost of the low-fidelity code allow for a nested design for practical applications.

3.1.1.2 Priors

The goal of a Bayesian prediction is to integrate the uncertainty of the parameter estimation into the predictive distribution as in (Le Gratiet, 2013a). Here the parameters are σ, β and β_ρ. As explained in (Le Gratiet & Garnier, 2013), the result is not Gaussian, but we can obtain expressions of the posterior mean $\mathbb{E}\left[A_{\mathsf{H}}(\mathbf{x}) | \mathcal{A} = \alpha\right]$ and variance $\mathbb{V}\left[A_{\mathsf{H}}(\mathbf{x}) | \mathcal{A} = \alpha\right]$. It is possible to consider informative or non-informative priors for the parameters (Le Gratiet & Garnier, 2013; Ma, 2020). Here we consider informative conjugate

priors:

$$\left[\sigma_{\mathsf{L}}^2\right] \sim \mathcal{IG}\left(m_{\mathsf{L}}, \varsigma_{\mathsf{L}}\right), \tag{3.9}$$

$$\left[\beta_{\mathsf{L}} | \sigma_{\mathsf{L}}^2\right] \sim \mathcal{N}_{p_{\mathsf{L}}}\left(b_{\mathsf{L}}, \sigma_{\mathsf{L}}^2 V_{\mathsf{L}}\right), \tag{3.10}$$

$$\left[\sigma_{\mathsf{H}}^2 | \mathcal{A}^{\mathsf{L}} = \alpha^{\mathsf{L}}, \beta_{\mathsf{L}}, \sigma_{\mathsf{L}}\right] \sim \mathcal{IG}\left(m_{\mathsf{H}}, \varsigma_{\mathsf{H}}\right), \tag{3.11}$$

$$\left[\beta_\rho, \beta_{\mathsf{H}} | \mathcal{A}^{\mathsf{L}} = \alpha^{\mathsf{L}}, \sigma_{\mathsf{H}}^2, \beta_{\mathsf{L}}, \sigma_{\mathsf{L}}\right] \sim \mathcal{N}_{q+p_{\mathsf{H}}}\left(b_{\mathsf{H}} = \begin{pmatrix} b^\rho \\ b_{\mathsf{H}}^\beta \end{pmatrix}, \sigma_{\mathsf{H}}^2 V_{\mathsf{H}} = \sigma_{\mathsf{H}}^2 \begin{pmatrix} V^\rho & 0 \\ 0 & V_{\mathsf{H}}^\beta \end{pmatrix}\right), \tag{3.12}$$

with

- b_{L} a vector of size p_{L},
- b^ρ a vector of size q,
- b_{H}^β a vector of size p_{H},
- V_{H}^β a $p_{\mathsf{H}} \times p_{\mathsf{H}}$ matrix,
- V^ρ a $q \times q$ matrix,
- V_{L} a $p_{\mathsf{L}} \times p_{\mathsf{L}}$ matrix,
- m_{F} and ς_{F} are positive real numbers and $\mathcal{IG}$ stands for the inverse Gamma distribution.

By using these informative conjugate priors we obtain the following a posteriori distributions as in (Le Gratiet & Garnier, 2013) :

$$\left[\sigma_{\mathsf{L}}^2 | \mathcal{A}^{\mathsf{L}} = \alpha^{\mathsf{L}}\right] \sim \mathcal{IG}\left(d_{\mathsf{L}}, Q_{\mathsf{L}}\right), \tag{3.13}$$

$$\left[\beta_{\mathsf{L}} | \mathcal{A}^{\mathsf{L}} = \alpha^{\mathsf{L}}, \sigma_{\mathsf{L}}^2\right] \sim \mathcal{N}_{p_{\mathsf{L}}}\left(\Sigma_{\mathsf{L}} \nu_{\mathsf{L}}, \Sigma_{\mathsf{L}}\right), \tag{3.14}$$

$$\left[\sigma_{\mathsf{H}}^2 | \mathcal{A} = \alpha\right] \sim \mathcal{IG}\left(d_{\mathsf{H}}, Q_{\mathsf{H}}\right), \tag{3.15}$$

$$\left[\beta_{\mathsf{H}}, \beta_\rho | \mathcal{A} = \alpha, \sigma_{\mathsf{H}}^2\right] \sim \mathcal{N}_{p_{\mathsf{H}}+q}\left(\Sigma_{\mathsf{H}} \nu_{\mathsf{H}}, \Sigma_{\mathsf{H}}\right), \tag{3.16}$$

with:

- $d_{\mathsf{F}} = \frac{n_{\mathsf{F}}}{2} + m_{\mathsf{F}}$,
- $\tilde{Q}_{\mathsf{F}} = \left(\alpha^{\mathsf{F}} - H_{\mathsf{F}} \hat{\lambda}_{\mathsf{F}}\right)^T C_{\mathsf{F}}^{-1} \left(\alpha^{\mathsf{F}} - H_{\mathsf{F}} \hat{\lambda}_{\mathsf{F}}\right)$,
- $Q_{\mathsf{F}} = \tilde{Q}_{\mathsf{F}} + \varsigma_{\mathsf{F}} + \left(b_{\mathsf{F}} - \hat{\lambda}_{\mathsf{F}}\right)^T \left(V_{\mathsf{F}} + \left(H_{\mathsf{F}}^T C_{\mathsf{F}}^{-1} H_{\mathsf{F}}\right)\right)^{-1} \left(b_{\mathsf{F}} - \hat{\lambda}_{\mathsf{F}}\right)$,
- $\Sigma_{\mathsf{F}} = \left[H_{\mathsf{F}}^T \frac{C_{\mathsf{F}}^{-1}}{\sigma_{\mathsf{F}}^2} H_{\mathsf{F}} + \frac{V_{\mathsf{F}}^{-1}}{\sigma_{\mathsf{F}}^2}\right]^{-1}$,
- $\nu_{\mathsf{F}} = \left[H_{\mathsf{F}}^T \frac{C_{\mathsf{F}}^{-1}}{\sigma_{\mathsf{F}}^2} \alpha^{\mathsf{F}} + \frac{V_{\mathsf{F}}^{-1}}{\sigma_{\mathsf{F}}^2} b_{\mathsf{F}}\right]$,
- H_{F} is defined by $H_{\mathsf{L}} = F_{\mathsf{L}}$ and $H_{\mathsf{H}} = \left[G^{\mathsf{L}} \odot \left(\alpha^{\mathsf{H}} \mathbf{1}_{q_{\mathsf{L}}}^T\right) \; F_{\mathsf{H}}\right]$,
- G^{L} is the $N_{\mathsf{H}} \times q$ matrix containing the values of $g_{\mathsf{L}}^T(\mathbf{x})$ for $\mathbf{x} \in D_{\mathsf{H}}$,
- $\mathbf{1}_{q_{\mathsf{L}}}$ is a q-dimensional vector containing 1,

- $\hat{\lambda}_{\mathsf{F}} = \left(H_{\mathsf{F}}^T C_{\mathsf{F}}^{-1} H_{\mathsf{F}}\right)^{-1} H_{\mathsf{F}}^T C_{\mathsf{F}}^{-1} \alpha^{\mathsf{F}}$.

Consequently, the posterior distribution of $A_{\mathsf{H}}(\mathbf{x})$ has the following mean and variance:

$$\mathbb{E}\left[A_{\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right] = h_{\mathsf{H}}^T(\mathbf{x})\Sigma_{\mathsf{H}}\nu_{\mathsf{H}} + r_{\mathsf{H}}^T(\mathbf{x})C_{\mathsf{H}}^{-1}\left(\alpha^{\mathsf{H}} - H_{\mathsf{H}}\Sigma_{\mathsf{H}}\nu_{\mathsf{H}}\right), \tag{3.17}$$

$$\begin{aligned}\mathbb{V}\left[A_{\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right] = {} & \left(\hat{\rho}_{\mathsf{L}}^2(\mathbf{x}) + \varepsilon_\rho(\mathbf{x})\right)\sigma^2_{\hat{A}_{\mathsf{L}}}(\mathbf{x}) + \tfrac{Q_{\mathsf{H}}}{2(d_{\mathsf{H}}-1)}(1 - r_{\mathsf{H}}^T(\mathbf{x})C_{\mathsf{H}}^{-1}r_{\mathsf{H}}(\mathbf{x})) \\ & + \left(h_{\mathsf{H}}^T - r_{\mathsf{H}}^T(\mathbf{x})C_{\mathsf{H}}^{-1}H_{\mathsf{H}}\right)\Sigma_{\mathsf{H}}\left(h_{\mathsf{H}}^T - r_{\mathsf{H}}^T(\mathbf{x})C_{\mathsf{H}}^{-1}H_{\mathsf{H}}\right)^T,\end{aligned} \tag{3.18}$$

with $\hat{\rho}_{\mathsf{L}}(\mathbf{x}) = g_{\mathsf{L}}^T(\mathbf{x})\hat{\beta}_\rho$, $\hat{\beta}_\rho = [\Sigma_{\mathsf{H}}\nu_{\mathsf{H}}]_{i=p_{\mathsf{H}}+1,\ldots,p_{\mathsf{H}}+q}$ and $\varepsilon_\rho(\mathbf{x}) = g_{\mathsf{L}}^T(\mathbf{x})\tilde{\Sigma}_{\mathsf{H}}g_{\mathsf{L}}(\mathbf{x})$ with $\tilde{\Sigma}_{\mathsf{H}} = [\Sigma_{\mathsf{H}}]_{i,j=p_{\mathsf{H}}+1,\ldots,p_{\mathsf{H}}+q}$.

The posterior mean $\mathbb{E}\left[A_{\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right]$ is the predictive model of the high-fidelity response and the posterior variance $\mathbb{V}\left[A_{\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right]$ represents the predictive variance of the model.

Interesting work on priors has been done in (Ma, 2020). Objective prior for the AR(1) model are developed in (Ma, 2020, sec. 4). The objective priors proposed in (Ma, 2020) are expressed in such a way that they can be computed efficiently. The prediction is similar in mean and covariance to the AR(1) model with informative prior for large learning set.

3.1.1.3 Illustration

In this section we illustrate the AR(1) multi-fidelity model on a function called Forrester function, see (Forrester et al., 2007). For this example we use the two levels of fidelity function: The low-fidelity function

$$z_L(x) = 0.5(6x-2)^2\sin(12x-4) + 10(x-0.5) - 5,$$

and the high-fidelity function

$$z_H(x) = 2z_L(x) - 20x + 20,$$

with $x \in [0,1]$.

We seek to estimate the value of the function z_H for any value of $x \in [0,1]$. For that we have two training sets $\mathcal{D}_H$ and $\mathcal{D}_L$ respectively for high- and low-fidelity. These two sets are nested, thus $\mathcal{D}_H \subset \mathcal{D}_L$.

In fig. 3.1 we show an example where the number of high-fidelity samples is 4 and the number of low-fidelity samples is varied from 4 to 11. The chosen kernel is Matérn $5/2$ with priors and hyperparameters are selected as in (Le Gratiet, 2012), ($\rho = 1$, $\beta = 0$, and no trends). We can already see that for the case where the two sets are of the same size, the classical GP regression is found. This is due to the Markov property of the AR(1) model. It can also be seen that the prediction improves as the number of low-fidelity points increases. The interest of multi-fidelity is also the reduction of the uncertainty. The 95% prediction interval is reduced close to the low-fidelity known points. This interval can be easily computed thanks to the Gaussian distribution. The prediction standard deviation $\mathbb{V}\left[A_{\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right]^{\frac{1}{2}}$ at this point is multiplied by a coefficient to obtain the half-width of the prediction interval. For 95% we have $1.96\mathbb{V}\left[A_{\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right]^{\frac{1}{2}}$.

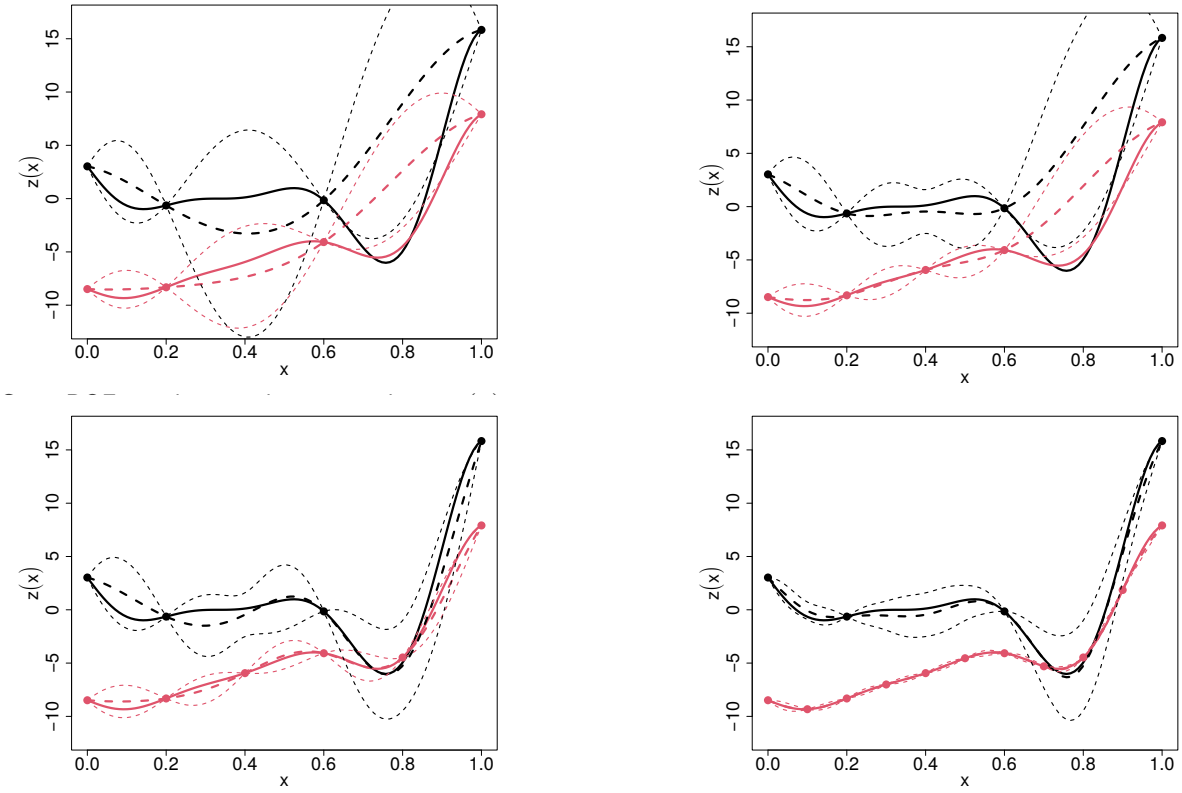


Figure 3.1: Multi-fidelity AR(1) regression for the Forrester function with various learning sets. The low-fidelity curves are represented in red and in black for high-fidelity one. The solid curve is the function, and dashed curve is the surrogate model with 95% prediction interval.

This shows the interest of the multi-fidelity approach.

However, this function is not very representative of what can be seen in multi-fidelity codes and was constructed for the AR(1) model. This explains the very good performance of the AR(1) method on this function. Moreover, this function is already very easy to learn in simple-fidelity even with little data, which limits the significance of the example.

3.1.2 DeepGP for multi-fidelity modeling

GP regression is an effective method for building emulators. However, it fails for a number of high-dimensional problems, such as images. With the increase of computing power and the development of network learning, a new method has emerged. In (Damianou & Lawrence, 2013), the deep GP method is proposed. This method consists in building a Gaussian process network in order to have a more complex model of stochastic processes. Deep GP was then extended to the multi-fidelity framework in (Perdikaris et al., 2017). In this case the deep GP is split into two. One part of the deep GP emulates the low-fidelity (the first layer) while the other one emulates the high-fidelity (the second layer). This method can theoretically be extended to more than two levels of fidelity (layers) but in practice the computational cost

explodes.

3.1.2.1 General definition of DeepGP

Deep Gaussian processes are recent methods introduced in (Damianou & Lawrence, 2013). The main idea is to offer a more flexible model than the classical Gaussian processes. In particular, Deep GPs are an efficient way to generalize non-stationary processes. An example of a metamodel constructed using this technique is (Radaideh & Kozlowski, 2020).

A deep Gaussian process is a stochastic process from $\mathbb{R}^d$ to $\mathbb{R}$ of the form:

$$Z_H \circ Z_{H-1} \circ \cdots \circ Z_1, \tag{3.19}$$

where, for $h = 1, \ldots, H$, Z_h is a Gaussian process from $\mathbb{R}^{d_h}$ to $\mathbb{R}^{d_{h+1}}$. d_h is the dimension of the latent variables, $d_1 = d$ and $d_{H+1} = 1$. In analogy to neural networks, $Z_2, \ldots, Z_{H-1}$ are hidden layer processes and H is the number of layers. Deep GP are based on composing Gaussian processes, their mathematical analysis becomes non-tractablew. A lot of computing power during training can overcome this problem.

Analogy between deep GP and NN is interesting but should not be confused with other work that seeks to find GPs in NN structures. Links have been made between certain types of NN and GP, see (Lee et al., 2018) for deep NN and (Garriga-Alonso, Rasmussen, & Aitchison, 2019) for convolutional NN. However, for Deep GPs the structure of the network is used to construct a complex stochastic process from Gaussian processes.

As what we are interested in here is multi-fidelity, the deep GP regression will not be further detailed here. However, a few works of interest can be detailed. The general framework was introduced in (Damianou & Lawrence, 2013) and with more details in (Damianou, 2015). For optimization, (Hebbal, Brevault, Balesdent, Talbi, & Melab, 2021) proposes an interesting approach and benefits from the non-stationarity of the output process. The general framework of the deep GP and the expression of the posterior is presented in (Bachoc & Lagnoux, 2021). Non-parametric Bayesian estimation of Deep GPs can be considered from this paper.

3.1.2.2 Deep GP for multi-fidelity regression

The Deep GP method introduced in (Perdikaris et al., 2017) makes it possible to adapt the approach to cases in which the relationships between the code levels are nonlinear. An improvement has been further made by adding to the covariance function of the high-fidelity GP proposed by (Perdikaris et al., 2017) a linear kernel in (Cutajar et al., 2019). Multi-fidelity GP regression has been used in several fields as illustrated in (Pilania, Gubernatis, & Lookman, 2017; Song, Chen, & Yue, 2019). In this section we place ourselves in the context of multi-fidelity with two levels of codes. We can extend to more than two levels,

but this becomes very expensive in terms of computation time. The low-fidelity code is emulated by a_L and the high-fidelity one by a_H.

The multi-fidelity deep GP model We generalize the autoregressive multi-fidelity scheme of eq. (3.1) to

$$a_H(\mathbf{x}) = g_L(a_L(\mathbf{x})) + \delta(\mathbf{x}), \tag{3.20}$$

where $g_L(\cdot)$ is a function that maps the low-fidelity model output to the high-fidelity one. In (Perdikaris et al., 2017), a Bayesian non-parametric treatment of g_L is proposed by assigning it a GP prior. a_L in eq. (3.1) is a GP prior. The composition of two GP priors $g_L(a_L(\mathbf{x}))$ gives rise to the deepGP as first put forth in (Damianou, 2015; Damianou & Lawrence, 2013), and, therefore, the posterior distribution of a_H is no longer Gaussian. The independence of g_L and δ allows eq. (3.20) to be simplified to:

$$a_H(\mathbf{x}) = \tilde{g}_L(a_L^\star(\mathbf{x}), \mathbf{x}), \tag{3.21}$$

where $\tilde{g}_L$ is a Gaussian process of mean μ_H and covariance k_H. The idea of this model is that δ is included in the modeling of $\tilde{g}_L$. $a_L^\star$ is the conditional low-fidelity GP. Under the assumption of noise-free data and stationary kernels, this leads to a property equivalent to the Markov property of the AR(1) model. (Zertuche, 2015) proposes an approach to this subject.

The covariance functions In this paragraph we discuss the choice of k_H. There are different possibilities of defining this covariance depending on the integration between codes. We will present the model proposed in (Perdikaris et al., 2017). Then we propose a covariance function we think is the most adapted to many cases: (Cutajar et al., 2019).

The covariance kernel proposed in (Perdikaris et al., 2017) is as follows:

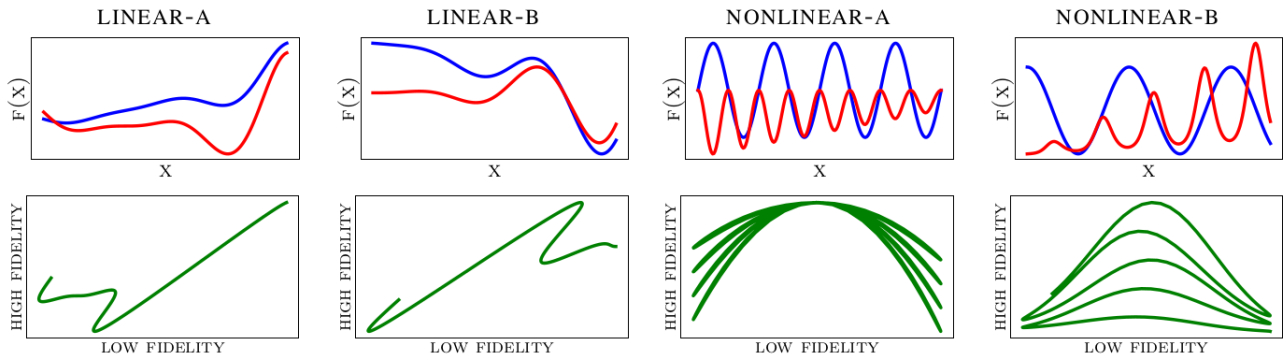
$$k_H(a_L^\star(\mathbf{x}), a_L^\star(\mathbf{x}'), \mathbf{x}, \mathbf{x}') = k^\rho(\mathbf{x}, \mathbf{x}')k^f(a_L^\star(\mathbf{x}), a_L^\star(\mathbf{x}')) + k^\delta(\mathbf{x}, \mathbf{x}'), \tag{3.22}$$

where k^f is the covariance between outputs obtained from the low-fidelity level, k^ρ is a space dependent scaling factor, and k^δ is the kernel modeling the bias at that fidelity level.

The proposed covariance function in (Cutajar et al., 2019) is as follows:

$$k_H(a_L^\star(\mathbf{x}), a_L^\star(\mathbf{x}'), \mathbf{x}, \mathbf{x}') = k^\rho(\mathbf{x}, \mathbf{x}') \left[\sigma^2 a_L^\star(\mathbf{x}) a_L^\star(\mathbf{x}') + k^f(a_L^\star(\mathbf{x}), a_L^\star(\mathbf{x}'))\right] + k^\delta(\mathbf{x}, \mathbf{x}'), \tag{3.23}$$

where σ^2 is the variance associated to the linear part. This term tends to represent the linear part of the integration between codes, which simplifies the optimisation. The two expressions are equivalent in theory but differ in terms of their parameters. This has an influence on the performance of the DeepGP



(a) Linear mapping.

(b) Nonlinear mapping.

Figure 3.2: Capture form (Cutajar, Pullin, Damianou, Lawrence, & González, 2019) of four examples of multi-fidelity functions. High-fidelity is in red and low-fidelity is in blue. The interaction between fidelity is in green.

model.

The parameter estimation is done as for DeepGP but much simpler. In fact we have an additional information: the output of the low-fidelity code. The prediction is therefore done in two steps. First, a classical simple-fidelity regression for the low-fidelity code. Second, a DeepGP regression but without changing the parameters associated with the first GP or low-fidelity layer.

Motivation of DeepGP: The AR(1) model is efficient for the case in which the integration between codes is linear. However, as a large part of the computational codes are derived from complex non-linear problems, it is expected that there will be non-linear interactions between codes. For non-linear interactions the DeepGP model is ideal. In this paragraph we try to show the interaction between two known codes and why DeepGP can have a contribution in certain cases.

In (Cutajar et al., 2019), represented in fig. 3.2, we can see different examples of multi-fidelity functions. We distinguish them in two groups: functions with linear and non-linear interactions. Their interactions can be seen on the lower curve in green. There is a clear difference between the two cases. For the linear case, we imagine that the AR(1) model fits perfectly, which is shown above and in (Cutajar et al., 2019). A straight line can be seen between codes. Even for the part where the line is not perfectly straight this can be solved by regression on δ. On the contrary, it is necessary in the two non-linear cases to take into account this non-linearity under the risk of losing all the advantage of the multi-fidelity regression. There are links between codes in this case but nothing linear, which is a problem for modeling. In particular, for the non-linear A, we can see a quadratic interaction which is relatively clear and which could help in the learning process. It is in non-linear cases that the use of DeepGP is recommended.

3.2 Multi-fidelity with Neural Network

Recent improvements in the implementation of neural networks have motivated research on multi-fidelity neural networks (Li, Xing, Kirby, & Zhe, 2020). In (Meng & Karniadakis, 2020), the authors combine a fully connected neural network (NN) and a linear system for the interactions between codes. The low-fidelity surrogate model is built using a fully connected NN, see (Zhang, Xie, Ji, Zhu, & Zheng, 2021) for a direct application. In order to quantify the prediction deviations and to evaluate the reliability of the prediction, the NNs have been improved to become Bayesian Neural Networks (BNN) (MacKay, 1992). The multi-fidelity model has been improved by using BNN for high-fidelity modeling in (Meng, Babaee, & Karniadakis, 2021). In this manuscript this method will be called MBK. The method in (Meng et al., 2021) offers options for single-fidelity active learning and is more general for multi-fidelity modeling. The disadvantages of methods using NN are the non-prediction of the model uncertainties and the difficult optimisation of the hyper-parameters in a small data context. These disadvantages can be overcome by the use of BNNs. The ability of BNN for uncertainty quantification is explained in (Kabir et al., 2018).

3.2.1 "Fully" connected modeling

Neural networks are a widely used method for regression as proposed in section 1.2. Moreover, it is an extremely flexible tool, and it is therefore possible to build a model as a composition of models. The simplest method considered is to build a fully connected network for low-fidelity. It takes inputs from the low-fidelity code and estimates the low-fidelity outputs. The high-fidelity replacement model has as input the inputs of the high-fidelity code plus the estimated low-fidelity outputs. It predicts the outputs of the high-fidelity code. This method is common and has been proposed in (Li, Kirby, & Zhe, 2020) for example. This method, although very simple, is extremely effective for problems with a large amount of data.

In order to make it even more predictive and to deal with problems where the amount of data is more limited, the BNN framework is used for high-fidelity regression. For this model, it is assumed that there is enough low-fidelity data to have a perfect or almost perfect emulator of the low-fidelity code. However, the high-fidelity is modelled by a BNN. This model has been proposed in (Meng et al., 2021).

The main advantage of these methods is their extreme simplicity. They are easily expandable to larger input or output dimensions. However, multi-fidelity NNs have defaults that prevent us from using them to solve our problem:

- The predictive uncertainty of low-fidelity surrogate model is not taken into account even though we have little low-fidelity data. We address this problem in chapter 4.
- The learning curve is very long even for simple applications. It is a waste to use tools that are very time-consuming when there are less time-consuming and equally efficient tools, such as Gaussian

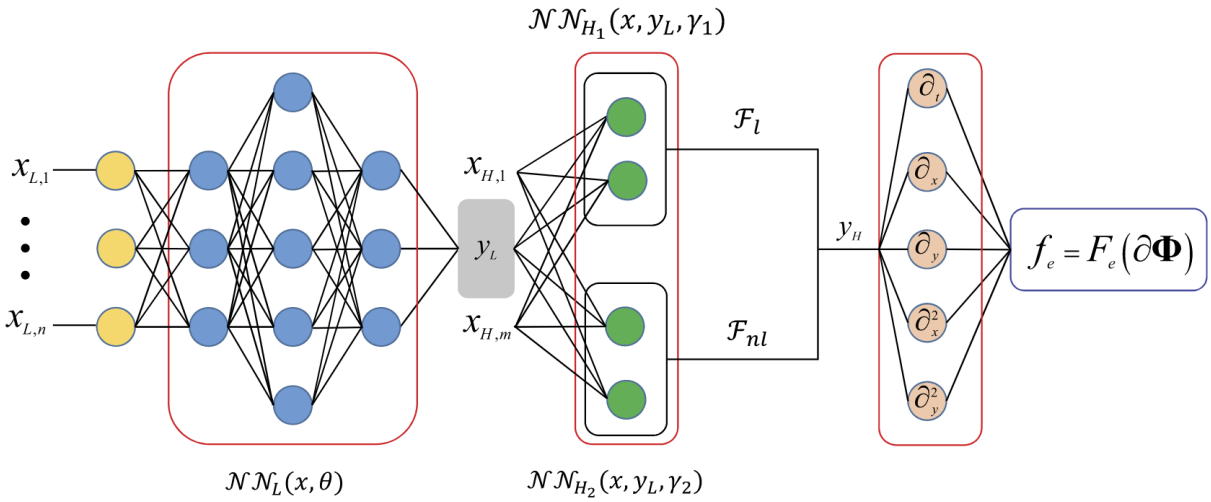


Figure 3.3: Illustration extracted from (Meng & Karniadakis, 2020) that shows the multi-fidelity NN model proposed.

processes.

- The output model is difficult to interpret. Sensitivity analysis is harder than for GP based models.

In this section we have only looked at NNs with simple structures. However, a method with a more complex structure has caught our attention. In the next paragraph we will present this method and its advantages over other multi-fidelity methods.

3.2.2 Physically inspired and AR(1) like NN

In this paragraph we present a method that has been introduced in (Meng & Karniadakis, 2020). Because of the structure we interpreted this as the NN version of the AR(1) model. Therefore, we call this method AR(1) like NN. It should be noted that this method is often linked to physically inspired neural networks, as in (Jagtap, Mitsotakis, & Karniadakis, 2022; Meng & Karniadakis, 2020). An example where the two methods are not related is (Zhang et al., 2021), but this paper only presents application results.

The principle of the method is to emulate the low-fidelity code by a NN as for the other methods. The difference is for high fidelity. As with the previous methods, the high-fidelity emulator takes as input the code inputs plus the low-fidelity prediction. However, the NN is split into two parts. The first part is a NN without the activation function, completely equivalent to a linear regression. The second part is a classical NN. The two parts are put in parallel and summed in output. We find (as for the AR(1) regression) a model with a linear part plus a term which depends on the inputs, hence the name of the method. An illustration of this model is proposed fig. 3.3. It can be noted that in the classical case a physics inspired NN is placed last. We can use the model without the physics-inspired NNs in the case where we have no information on the regularity of our code. The outputs are then less accurate but not necessarily less predictive.

The advantage of this method is that it can be extended to high-dimensional problems. Moreover, its interpretation is easier thanks to the linear part. We have used this method in chapter 6 to compare with the methods we have proposed. The computational cost is also greatly reduced because the model is simpler than for the NN black boxes. However, the uncertainty is not taken into account. Nevertheless, this method seems to us to be essential in the study of the high output dimension, even if theoretical developments are lacking to characterize its efficiency.

3.3 Conclusion on state-of-the-art multi-fidelity

There are many methods for performing multi-fidelity regression. They are adapted to different cases and often have complementary drawbacks. It seems to us that the AR(1) approach can be applied to many problems. The model adapts well to many problem frameworks. However, there is still a lot of work to be done in high dimensions. More recent methods exist but are not very convincing in terms of quantification of inaccuracy, the NN, or in terms of computation time, the DeepGP.

The state-of-the-art methods and the original methods that we proposed in chapter 4 can be compared on test cases, from simple functions to complex numerical codes. Such comparisons are carried out in (Cutajar et al., 2019) and in chapter 4. It is impossible to conclude on the superiority of one method over the others. It can be said that with little data or when the link between codes is clode to linear the AR(1) model is the most efficient. The DeepGP models are the most efficient when the link between codes is known but non-linear. NN methods are more efficient when we have little knowledge about the code, in high dimension and with large sets of data.

Part II

Multi-fidelity models for high-dimension output

Chapter 4

Gaussian Process-Bayesian Neural Network

In this chapter we will present a work on multi-fidelity modeling. Even if we consider the AR(1) multi-fidelity model as very efficient, it is interesting to improve the model and to try to overcome its deficiencies. Gaussian process regression is a proven method. BNNs are more and more present in the literature of uncertainty quantification. It is therefore quite natural that we are interested in combining the two in a multi-fidelity model.

This chapter is based on the work (Kerleguer et al., 2022). In a first part, the general model of Gaussian Process-Bayesian Neural Network is presented. In a second part, three sampling methods to transfer the output information from the GP into inputs of the BNN, are proposed. Finally, in a last part the GP-BNN method is compared to other methods.

4.1 Combining Gaussian processes and Bayesian neural network

From now on we consider a multi-fidelity framework with two levels of code, high f_H and low f_L fidelity, as in (Kennedy & O'Hagan, 2000). The input is $\mathbf{x} \in \mathbb{R}^d$ and the outputs of both computer code levels $f_L(\mathbf{x})$ and $f_H(\mathbf{x})$ are scalar. We have access to N_L low-fidelity points and N_H high-fidelity points, with $N_H < N_L$. In our article we focus on the small data framework where the low-fidelity code is not perfectly known. Under such circumstances it remains uncertainty in the low-fidelity surrogate model. If $N_H \ll N_L$ the situation is different, and we could assume that the low-fidelity emulator is perfect.

We therefore have two surrogate modeling tools: GP regression and BNN. To do multi-fidelity with non-linear interactions the standard methods use combinations of regression methods. With our two methods we can make four combinations: GP-GP also called DeepGP in (Cutajar et al., 2019; Perdikaris et al., 2017), GP-BNN the method proposed in our manuscript, BNN-GP and BNN-BNN. The Deep GP model

will be compared to the proposed method in all examples of our manuscript. The BNN-BNN method would be extremely expensive and very close to the full NN methods by adding the predictive uncertainty. The logic behind our choice of GP-BNN over BNN-GP is as follows: if we assume that the low-fidelity code is simpler than the high-fidelity code, then it must be approximated by a simpler model. GP regression is a surrogate model easier to obtain and it gives a Gaussian output distribution that can be sampled easily. Whereas BNN is more complex to construct and allows for more general output distributions to be emulated.

The code output to estimate is f_H with the help of low- and high-fidelity points. As the low-fidelity code f_L is not completely known a regression method with uncertainty quantification, GP regression, is used, to emulate it, as in section 1.1. The output of the low-fidelity GP is then integrated into the input to a BNN, described in section 1.2.2.2, to predict f_H.

The low-fidelity surrogate model is a GP $Y_L(\mathbf{x})$ built from N_L low-fidelity data points $\left(\mathbf{x}_L^{(i)}, f_L(\mathbf{x}_L^{(i)})\right) \in \mathbb{R}^d \times \mathbb{R}$. The optimisation of the hyper-parameters of the GP is carried out in the construction of the surrogate model. The GP is characterized by a predictive mean $\mu_L(\mathbf{x})$ and a predictive covariance $C_L(\mathbf{x}, \mathbf{x}')$.

To connect the GP with the BNN the simplest way is to concatenate $\mathbf{x}$ and $\mu_L(\mathbf{x})$ (the best low-fidelity predictor) as input to the BNN. However, this does not take into account the predictive uncertainty. Consequently, we may want to add $C_L(\mathbf{x}, \mathbf{x})$ or $\sqrt{C_L(\mathbf{x}, \mathbf{x})}$ to the input vector of the BNN. The idea is that the BNN could learn from the low-fidelity GP more than from its predictive mean only, in order to give reliable predictions of the high-fidelity code with quantified uncertainties. We will show that the idea is fruitful, and it can be pushed even further.

We have investigated three methods to combine the two surrogate models and to transfer the posterior distribution of the low-fidelity emulator to the high-fidelity one. We demonstrate in section 4.3 that the best solution is the so-called Gauss-Hermite method.

4.2 Transfer methods

The two learning sets are $\mathcal{D}^L = \{(\mathbf{x}_L^{(i)}, f_L(\mathbf{x}_L^{(i)})), i = 1, \ldots, N_L\}$ and $\mathcal{D}^H = \{(\mathbf{x}_H^{(i)}, f_H(\mathbf{x}_H^{(i)})), i = 1, \ldots, N_H\}$ with typically $N_H < N_L$ and we do not need to assume that the sets $\{\mathbf{x}_L^{(i)}, i = 1, \ldots, N_L\}$ and $\{\mathbf{x}_H^{(i)}, i = 1, \ldots, N_H\}$ are nested.

The low-fidelity model is emulated using GP regression, as a consequence the result is formulated as a posterior distribution given $\mathcal{D}^L$ that has the form of a Gaussian law.

Property 4.2.1. *The posterior distribution of $Y_L(\mathbf{x})$ knowing $\mathcal{D}^L$ is the Gaussian distribution with mean $\mu_L(\mathbf{x})$ and variance $\sigma_L^2(\mathbf{x})$ of the form (1.21-1.22). We denote its pdf by $p(y_L|\mathcal{D}_L, \mathbf{x})$.*

Proof. The proof is given in (Rasmussen & Williams, 2006, ch. 2.2) (prediction with noise free observations) and in section 1.1. □

The posterior distribution of the high-fidelity code given the low-fidelity learning set $\mathcal{D}^L$ and the high-fidelity learning set $\mathcal{D}^H$ may have different forms depending on the input of the BNN.

4.2.1 Mean-Standard deviation method

In the Mean-Standard deviation method, called Mean-Std method, we give as input to the BNN the point $\mathbf{x}$ and the information usually available at the output of a GP regression, i.e. the predictive mean and standard deviation of the low-fidelity emulator at the point $\mathbf{x}$.

In this method, the input of the BNN whose output gives the prediction of the high-fidelity code at $\mathbf{x}$ is $\mathbf{x}^{\text{BNN}} = (\mathbf{x}, \mu_L, \sigma_L)$. The idea is that the BNN input consists of the input $\mathbf{x}$ of the code and of the mean and standard deviation of the posterior distribution of the low-fidelity emulator at $\mathbf{x}$. The high-fidelity emulator is modelled as:

$$Y_H(\mathbf{x}) = BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x})) + \sigma\epsilon, \tag{4.1}$$

with $\epsilon \sim \mathcal{N}(0,1)$. We use the learning set $\mathcal{D}^H_{MS} = \big\{\big(\mathbf{x}_H^{(i)}, \mu_L(\mathbf{x}_H^{(i)}), \sigma_L(\mathbf{x}_H^{(i)}), f_H(\mathbf{x}_H^{(i)})\big), i = 1, \ldots, N_H\big\}$ to train the BNN and get the posterior distribution of $(\boldsymbol{\theta}, \sigma)$, see section 1.2.2.2. Note that $\mathcal{D}^H_{MS}$ can be deduced from $\mathcal{D}^L$ and $\mathcal{D}^H$.

Property 4.2.2. *The posterior distribution of $Y_H(\mathbf{x})$ knowing $\mathcal{D}^L$ and $\mathcal{D}^H_{MS}$ has pdf*

$$p\big(y_H|\mathbf{x}, \mathcal{D}^H_{MS}, \mathcal{D}^L\big) = \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{(y_H - BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x})))^2}{2\sigma^2}\Big) p(\boldsymbol{\theta}, \sigma|\mathcal{D}^H_{MS}) d\sigma d\boldsymbol{\theta}, \tag{4.2}$$

with $p(\boldsymbol{\theta}, \sigma|\mathcal{D}^H_{MS})$ the posterior pdf of the hyper-parameters of the BNN.

Proof. Let $p(y_H|\boldsymbol{\theta}, \sigma, \mathbf{x}, \mathcal{D}^L)$ be the probability density function (pdf) of $Y_H(\mathbf{x})$ given by eq. (4.1). This pdf can be written:

$$p(y_H|\boldsymbol{\theta}, \sigma, \mathbf{x}, \mathcal{D}^L) \;=\; \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{(y_H - BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x})))^2}{2\sigma^2}\Big)$$

The law of total probability gives that:

$$\begin{aligned} p\big(y_H|\mathbf{x}, \mathcal{D}^H_{MS}, \mathcal{D}^L\big) &= \iint p(y_H|\boldsymbol{\theta}, \sigma, \mathbf{x}, \mathcal{D}^L) p(\boldsymbol{\theta}, \sigma|\mathcal{D}^H_{MS}) d\sigma d\boldsymbol{\theta} \\ &= \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{(y_H - BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x})))^2}{2\sigma^2}\Big) p(\boldsymbol{\theta}, \sigma|\mathcal{D}^H_{MS}) d\sigma d\boldsymbol{\theta}. \end{aligned}$$

□

Corollary 4.2.2.1. *The mean and variance of the posterior distribution of $Y_H(\mathbf{x})$ knowing $\mathcal{D}^H_{MS}$ and $\mathcal{D}^L$ is:*

$$\mu_H(\mathbf{x}) = \iint BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x})) p(\boldsymbol{\theta}, \sigma|\mathcal{D}^H_{MS}) d\sigma d\boldsymbol{\theta}, \tag{4.3}$$

and

$$C_H(\mathbf{x}) = \iint \left(BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x}))^2 + \sigma^2\right) p(\boldsymbol{\theta}, \sigma | \mathcal{D}_{MS}^H) d\sigma d\boldsymbol{\theta}, \tag{4.4}$$

Proof. By the definition of the mean and variance we get:

$$\begin{aligned} \mu_H(\mathbf{x}) &= \int y_H p\big(y_H | \mathbf{x}, \mathcal{D}_{MS}^H, \mathcal{D}^L\big) dy_H, \\ C_H(\mathbf{x}) &= \int (y_H - \mu_H(\mathbf{x}))^2 p\big(y_H | \mathbf{x}, \mathcal{D}_{MS}^H, \mathcal{D}^L\big) dy_H. \end{aligned}$$

We replace by the expressions given in property 4.2.2.

$$\mu_H(\mathbf{x}) = \int y_H \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(- \frac{(y_H - BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x})))^2}{2\sigma^2}\Big) p(\boldsymbol{\theta}, \sigma | \mathcal{D}_{MS}^H) d\sigma d\boldsymbol{\theta} dy_H,$$

which gives eq. (4.3). For the variance:

$$C_H(\mathbf{x}) = \int (y_H - \mu_H(\mathbf{x}))^2 \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(- \frac{(y_H - BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x})))^2}{2\sigma^2}\Big) p(\boldsymbol{\theta}, \sigma | \mathcal{D}_{MS}^H) d\sigma d\boldsymbol{\theta} dy_H,$$

which gives eq. (4.4). □

Corollary 4.2.2.2. *Given $\mathcal{D}^L$ and $\mathcal{D}_{MS}^H$, given a sample $(\theta_i, \sigma_i)_{i=1}^{N_v}$ of the posterior distribution of $(\boldsymbol{\theta}, \sigma)$:*

$$\tilde{\mu}_H(\mathbf{x}) = \frac{1}{N_v} \sum_{i=1}^{N_v} BNN_{\boldsymbol{\theta}_i}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x})), \tag{4.5}$$

and

$$\tilde{C}_H(\mathbf{x}) = \frac{1}{N_v} \sum_{i=1}^{N_v} \left(BNN_{\boldsymbol{\theta}_i}(\mathbf{x}, \mu_L(\mathbf{x}), \sigma_L(\mathbf{x}))^2 + \sigma_i^2\right) - \tilde{\mu}_H(\mathbf{x})^2, \tag{4.6}$$

are consistent estimators of $\mu_H(\mathbf{x})$ and $C_H(\mathbf{x})$.

The estimators need samples of the posterior distribution of $(\boldsymbol{\theta}, \sigma)$. By the HMC method (NUTS), we get a sample of the hyperparameters $(\boldsymbol{\theta}_j, \sigma_j)$ with the posterior distribution of the high-fidelity model.

4.2.2 Quantiles method

The Quantiles method consists of giving the mean and two quantiles of the low-fidelity GP emulator as input to the BNN. Assuming we want to have the high-fidelity output uncertainty at level $\alpha\%$ we take the

$\alpha/2\%$ and the $(1-\alpha/2)\%$ quantiles. The high-fidelity emulator is modelled by $Y_H(\mathbf{x})$:

$$Y_H(\mathbf{x}) = BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), Q_{L,\alpha/2}(\mathbf{x}), Q_{L,(1-\alpha/2)}(\mathbf{x})) + \sigma\epsilon, \tag{4.7}$$

with $\epsilon \sim \mathcal{N}(0,1)$. The expression of the BNN input vector is $\mathbf{x}^{\text{BNN}} = (\mathbf{x}, \mu_L, Q_{L,\alpha/2}, Q_{L,(1-\alpha/2)})$ and the high-fidelity learning set is $\mathcal{D}_Q^H : \left\{ \left((\mathbf{x}_H^{(i)}, \mu_L(\mathbf{x}_L^{(i)}), Q_{L,\alpha/2}(\mathbf{x}_H^{(i)}), Q_{L,(1-\alpha/2)}(\mathbf{x}_H^{(i)})), f_H(\mathbf{x}_H^{(i)}) \right), i = 1, \cdots, N_H \right\}$.

Property 4.2.3. *The posterior distribution of $Y_H(\mathbf{x})$ knowing $\mathcal{D}^L$ and $\mathcal{D}_Q^H$ has pdf*

$$p(y_H|\mathbf{x}, \mathcal{D}_Q^H, \mathcal{D}^L) = \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{(y_H - BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), Q_{L,\alpha/2}(\mathbf{x}), Q_{L,(1-\alpha/2)}(\mathbf{x})))^2}{2\sigma^2} \Big) p(\boldsymbol{\theta}, \sigma|\mathcal{D}_Q^H) d\sigma d\boldsymbol{\theta}, \tag{4.8}$$

with $p(\boldsymbol{\theta}, \sigma|\mathcal{D}_Q^H)$ the posterior pdf of the hyper-parameters of the BNN.

Corollary 4.2.3.1. *The mean and variance of the posterior distribution of $Y_H(\mathbf{x})$ knowing $\mathcal{D}_Q^H$ and $\mathcal{D}^L$ is:*

$$\mu_H(\mathbf{x}) = \iint BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), Q_{L,\alpha/2}(\mathbf{x}), Q_{L,(1-\alpha/2)}(\mathbf{x})) p(\boldsymbol{\theta}, \sigma|\mathcal{D}_Q^H) d\sigma d\boldsymbol{\theta}, \tag{4.9}$$

and

$$C_H(\mathbf{x}) = \iint \left(BNN_{\boldsymbol{\theta}}(\mathbf{x}, \mu_L(\mathbf{x}), Q_{L,\alpha/2}(\mathbf{x}), Q_{L,(1-\alpha/2)}(\mathbf{x}))^2 + \sigma^2 \right) p(\boldsymbol{\theta}, \sigma|\mathcal{D}_Q^H) d\sigma d\boldsymbol{\theta}, \tag{4.10}$$

Corollary 4.2.3.2. *Given $\mathcal{D}^L$ and $\mathcal{D}_Q^H$, given a sample $(\theta_i, \sigma_i)_{i=1}^{N_v}$ of the posterior distribution of $(\boldsymbol{\theta}, \sigma)$.*

$$\tilde{\mu}_H(\mathbf{x}) = \frac{1}{N_v} \sum_{i=1}^{N_v} BNN_{\boldsymbol{\theta}_i}(\mathbf{x}, \mu_L(\mathbf{x}), Q_{L,\alpha/2}(\mathbf{x}), Q_{L,(1-\alpha/2)}(\mathbf{x})), \tag{4.11}$$

and

$$\tilde{C}_H(\mathbf{x}) = \frac{1}{N_v} \sum_{i=1}^{N_v} \left(BNN_{\boldsymbol{\theta}_i}(\mathbf{x}, \mu_L(\mathbf{x}), Q_{L,\alpha/2}(\mathbf{x}), Q_{L,(1-\alpha/2)}(\mathbf{x}))^2 + \sigma_i^2 \right) - \tilde{\mu}_H(\mathbf{x})^2, \tag{4.12}$$

are consistent estimators of the mean and variance in corollary 4.2.3.1.

This method is very similar to the Mean-Std method and only the BNN input changes. This is why the estimators have the same form. It can be noted that the dimension of the BNN inputs is larger than for the Mean-Std method. The estimators need samples of the posterior distribution of $(\boldsymbol{\theta}, \sigma)$. By the HMC method (NUTS), we get a sample of the posterior law of the hyperparameters $(\boldsymbol{\theta}_j, \sigma_j)$ with the posterior distribution of the high-fidelity model.

4.2.3 Gauss-Hermite quadrature

In this section we want to transfer the information about the posterior distribution of the low-fidelity emulator by a sampling method. The GP posterior distribution is one-dimensional and Gaussian for each value of $\mathbf{x}$. Therefore, a deterministic method for sampling is preferable in order to limit the number of calls to the BNN. In the following this method is called GPBNN. We propose to represent the Gaussian distribution by S nodes of the Gauss-Hermite quadrature (Gautschi, 2012, ch. 3). This method has been chosen because it estimates the Gaussian distribution efficiently. The samples $\tilde{f}_{L,j}(\mathbf{x})$, with $j = 1, \dots, S$, of the GP posterior distribution are constructed using the roots $z_{S,j}$ of the physicists' version of the Hermite polynomials $H_S(x) = (-1)^S e^{x^2} \partial_x^S e^{-x^2}$, $S \in \mathbb{N}$. For each input $\mathbf{x}$ the GP posterior law has mean $\mu_L(\mathbf{x})$ and variance $C_L(\mathbf{x}, \mathbf{x})$. Therefore, the jth realisation in the Gauss-Hermite quadrature formula is:

$$\tilde{f}_{L,j}(\mathbf{x}) = \mu_L(\mathbf{x}) + z_{S,j}\sqrt{2}\sqrt{C_L(\mathbf{x}, \mathbf{x})}, \tag{4.13}$$

and the associated weight is $p_{S,j} = \frac{2^{S-1} S!}{S^2 H_{S-1}^2(z_{S,j})}$, for $j = 1, \dots, S$. The learning set of the BNN is $\mathcal{D}_{GH}^H$: $\left\{ \left((\mathbf{x}_H^{(i)}, \mu_L(\mathbf{x}_H^{(i)}), \sigma_L(\mathbf{x}_H^{(i)})), f_H(\mathbf{x}_H^{(i)}) \right), i = 1, \cdots, N_H \right\}$. The high-fidelity emulator is modelled as:

$$Y_H(\mathbf{x}) = \sum_{j=1}^{S} p_{S,j} BNN_{\boldsymbol{\theta}}(\mathbf{x}, \tilde{f}_{L,j}(\mathbf{x})) + \sigma\epsilon, \tag{4.14}$$

with $\epsilon \sim \mathcal{N}(0, 1)$ and $\tilde{f}_{L,j}(\mathbf{x})$ is given by eq. (4.13). The Gauss-Hermite method for multi-fidelity is illustrated at fig. 4.1.

Property 4.2.4. *The posterior distribution of $Y_H(\mathbf{x})$ knowing $\mathcal{D}_{GH}^H$ and $\mathcal{D}^L$ is*

$$p\left(y_H | \mathbf{x}, \mathcal{D}_{GH}^H, \mathcal{D}^L\right) = \iint \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2} \left(y_H - \sum_{j=1}^{S} p_{S,j} BNN_{\boldsymbol{\theta}}(\mathbf{x}, \tilde{f}_{L,j}) \right)^2 \right) p(\boldsymbol{\theta}, \sigma | \mathcal{D}_{GH}^H) d\sigma d\boldsymbol{\theta}, \tag{4.15}$$

with $p(\boldsymbol{\theta}, \sigma | \mathcal{D}_{GH}^H)$ the posterior distribution of the hyper-parameters of the BNN. $\tilde{f}_{L,j}$ is given by eq. (4.13)*.*

Corollary 4.2.4.1. *The posterior mean of $Y_H(\mathbf{x})$ is:*

$$\mu_H(\mathbf{x}) = \iint \left(\sum_{j=1}^{S} p_{S,j} BNN_{\boldsymbol{\theta}}(\mathbf{x}, \tilde{f}_{L,j}) \right) p(\boldsymbol{\theta}, \sigma | \mathcal{D}_{GH}^H) d\sigma d\boldsymbol{\theta}. \tag{4.16}$$

The posterior variance of $Y_H(\mathbf{x})$ is:

$$C_H(\mathbf{x}) = \iint \left(\left(\sum_{j=1}^{S} p_{S,j} BNN_{\boldsymbol{\theta}}(\mathbf{x}, \tilde{f}_{L,j}) \right)^2 + \sigma^2 \right) p(\boldsymbol{\theta}, \sigma | \mathcal{D}_{GH}^H) d\sigma d\boldsymbol{\theta} - \mu_H^2(\mathbf{x}). \tag{4.17}$$

The expression of $\tilde{f}_{L,j}$ is given by eq. (4.13)*.*

Corollary 4.2.4.2. *Given $\mathcal{D}^L$ and $\mathcal{D}^H_{GH}$, given a sample $(\boldsymbol{\theta}_i, \sigma_i)_{i=1}^{N_v}$ of the posterior distribution of $(\boldsymbol{\theta}, \sigma)$,*

$$\tilde{\mu}_H(\mathbf{x}) = \frac{1}{N_v} \sum_{i=1}^{N_v} \sum_{j=1}^{S} p_{S,j} BNN_{\boldsymbol{\theta}_i}(\mathbf{x}, \tilde{f}_{L,j}(\mathbf{x})), \tag{4.18}$$

and

$$\tilde{C}_H(\mathbf{x}) = \frac{1}{N_v} \sum_{i=1}^{N_v} \left(\sum_{j=1}^{S} p_{S,j} BNN_{\boldsymbol{\theta}_i}(\mathbf{x}, \tilde{f}_{L,j}(\mathbf{x})) \right)^2 + \frac{1}{N_v} \sum_{i=1}^{N_v} \sigma_i^2 - \tilde{\mu}_H^2(\mathbf{x}), \tag{4.19}$$

are consistent estimators of the mean and variance in corollary 4.2.4.1.

Note that this sampling method is different from the Quantiles method (even for $S = 3$ and $\alpha \approx 0.110$). Indeed, in the Quantiles method the input of the BNN contains the mean and quantiles of the low-fidelity code predictor whereas the Gauss-Hermite method is a sampling method in which the input of the BNN contains only one sample of the low-fidelity code predictor and the predictor is a weighted average of the BNN.

The sample $(\boldsymbol{\theta}_i, \sigma_i)$ of the posterior distribution of $(\boldsymbol{\theta}, \sigma)$ can be obtained by the HMC method (NUTS).

The choice of S is a trade-off between a large value that is computationally costly and a small value that does not propagate the uncertainty appropriately. $S = 2$ is the smallest admissible value regarding the information that should be transferred. At first glance, a large value of S could be expected to be the best choice in the point of view of the predictive accuracy. However, a too large value of S degrades the accuracy of the predictive mean estimation. This is due to large variations of $\tilde{f}_{L,i}(\mathbf{x})$ for large values of S. Interesting values turn out to be between 3 and 10 depending on the quality of the low-fidelity emulator, as discussed in section 4.3.

The value of N_v is chosen large enough so that the estimators in eqs. (4.18) and (4.19) have converged. As shown in the analysis of section 4.3 $N_v = 500$ is sufficient.

We could expect the computational cost of the GPBNN method to be expensive. The number of operations needed to compute an iteration of the HMC optimisation is proportional to $S \times N_v \times N_H$. Because S and N_H are small in our context the computational cost of one realisation of the BNN is actually low. Thus optimisation of the hyperparameters is feasible for $N_H \lesssim 100$.

4.3 Experiments/Comparison with other methods

In this section we present two analytic examples and a simulated one. The first one is a 1D function, and we consider that the low-fidelity function may be unknown in a certain subdomain. The second one is a

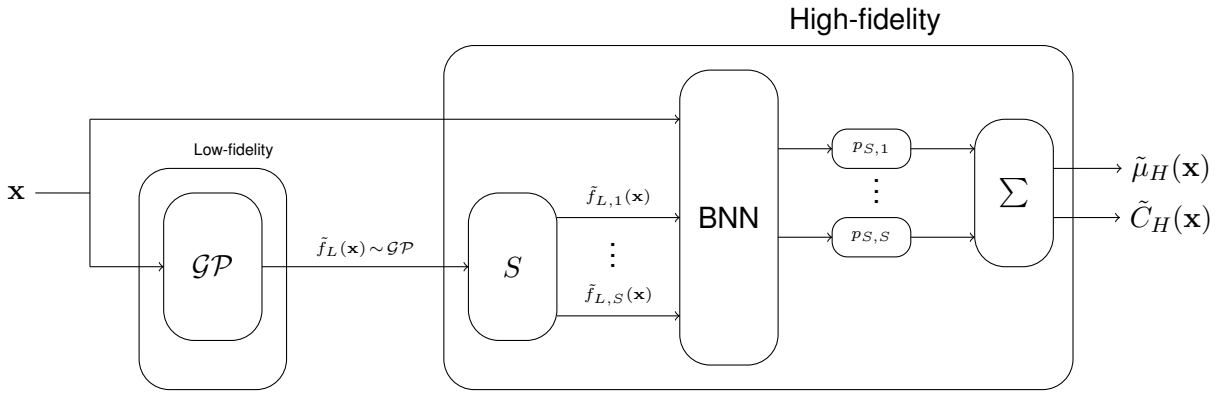


Figure 4.1: Schematic of the multi-fidelity Gauss-Hermite model. The input is a point $\mathbf{x}$. The output consists of a predictive mean $\tilde{\mu}_H(\mathbf{x})$ and a predictive variance $\tilde{C}_H(\mathbf{x})$.

2D function with noise. Finally, we test the strategy on a more complex double pendulum system. All the numerical experiments are carried out on a laptop (of 2017, dell precision with intel core i7) using only CPU and the running time never exceeds 2 hours.

4.3.1 1D function approximation

The low- and high-fidelity functions are:

$$f_L(x) = \sin 8\pi x, \qquad f_H(x) = \big(x - \sqrt{2}\big) f_L^2(x), \tag{4.20}$$

with $x \in [0,1]$, where f_H is the high-fidelity function (code) and f_L the low-fidelity function (code). These functions have been introduced in (Perdikaris et al., 2017) and are well estimated with a DeepGP and a quadratic form of the covariance. In this example we assume that we have access to a lot of low-fidelity data, $N_L = 100$, while the high-fidelity data is small, $N_H = 20$. We also consider situations in which there is a segment $\bar{I} \subset [0,1]$ where we do not have access to $f_L(x)$. The learning set for the high-fidelity code is obtained by partitioning $[0,1]$ into N_H segments with equal lengths and then by choosing independently one point randomly on each segment with uniform distribution. The learning set for the low-fidelity code is obtained by partitioning $[0,1]\backslash\bar{I}$ into N_L segments with equal length and then by choosing independently one point randomly on each segment with uniform distribution. The test set is composed of $N_{\mathrm{T}} = 1000$ independent points following a random uniform law on $[0,1]$.

We adapt the errors available in section 1.3 to the multi-fidelity context. We denote by $\big(\mathbf{x}_{\mathrm{T}}^{(i)}, f_H(\mathbf{x}_{\mathrm{T}}^{(i)})\big)_{i=1,\dots,N_{\mathrm{T}}}$ the test set. The error is evaluated by:

$$Q^2_{\mathrm{T}} = 1 - \frac{\sum_{i=1}^{N_{\mathrm{T}}} \big[\tilde{\mu}_H(\mathbf{x}_{\mathrm{T}}^{(i)}) - f_H(\mathbf{x}_{\mathrm{T}}^{(i)})\big]^2}{N_{\mathrm{T}} \mathbb{V}_{\mathrm{T}}\left(f_H\right)}, \tag{4.21}$$

with $\mathbb{V}_{\mathrm{T}}(f_H) = \frac{1}{N_{\mathrm{T}}} \sum_{i=1}^{N_{\mathrm{T}}} \big[f_H(\mathbf{x}_{\mathrm{T}}^{(i)}) - \frac{1}{N_{\mathrm{T}}} \sum_{j=1}^{N_{\mathrm{T}}} f_H(\mathbf{x}_{\mathrm{T}}^{(j)})\big]^2$. A highly predictive model gives a Q^2_{T} close

to 1 while a less predictive model has a smaller Q^2_{T}. The coverage probability CP_α is defined as the probability for the actual value of the function to be within the prediction interval with confidence level α of the surrogate model:

$$\mathrm{CP}_\alpha = \frac{1}{N_{\mathrm{T}}} \sum_{i=1}^{N_{\mathrm{T}}} \mathbf{1}_{f_H(\mathbf{x}_{\mathrm{T}}^{(i)}) \in \mathcal{I}_\alpha(\mathbf{x}_{\mathrm{T}}^{(i)})}, \tag{4.22}$$

with $\mathbf{1}$ the indicator function and $\mathcal{I}_\alpha(\mathbf{x})$ the prediction interval at point $\mathbf{x}$ with confidence level α. The mean prediction interval width MPIW_α is the average width of the prediction intervals:

$$\mathrm{MPIW}_\alpha = \frac{1}{N_{\mathrm{T}}} \sum_{i=1}^{N_{\mathrm{T}}} \left|\mathcal{I}_\alpha(\mathbf{x}_{\mathrm{T}}^{(i)})\right|, \tag{4.23}$$

where $\mathcal{I}_\alpha(\mathbf{x})$ is the prediction interval at point $\mathbf{x}$ with confidence level α and $|\mathcal{I}_\alpha(\mathbf{x})|$ the length of the prediction interval $\mathcal{I}_\alpha(\mathbf{x})$. The prediction uncertainty of the surrogate model is well characterized when CP_α is close to α. The prediction uncertainty of the surrogate model is small when MPIW_α is small.

For the GPBNN method we obtain the interval $\mathcal{I}_\alpha(\mathbf{x})$ by sampling N_v realisations of the posterior distribution of the random process $Y_H(\mathbf{x})$ described in section 4.2. The interval $\mathcal{I}_\alpha(\mathbf{x})$ is the smallest interval that contains the fraction α of the realisations $(y_j)_{j=1}^{N_v}$ of $Y_H(\mathbf{x})$. The performances for the different methods are compared in table 4.1. For the GP 1F model and the AR(1) model the prediction interval is centered at the predictive mean and its half-length is $q_{1-\frac{\alpha}{2}}$ times the predictive standard deviation, where $q_{1-\frac{\alpha}{2}}$ is the $1-\frac{\alpha}{2}$-quantile of the standard Gaussian law, because the posterior distributions are Gaussian. For the Deep GP model the prediction interval is obtained by Monte-Carlo sampling of the posterior distribution (with 1000 samples).

We use GP regression with zero mean function and tensorized Matérn $5/2$ covariance function for the low-fidelity GP regression. The implementation we use is from (GPy, 2012). The optimisation for GP regression gives a correlation length of 0.108. For this example we choose $N_n = 30$ neurons, we use the ReLU function as activation function, and we use the BNN implementation proposed in (Bingham et al., 2019). The sample size of the posterior distribution of the BNN parameter $(\boldsymbol{\theta}, \sigma)$ is $N_v = 500$ (see fig. 4.3).

Low-fidelity surrogate models of different accuracies are considered to understand how our strategy behaves under low-fidelity uncertainty. This is done by considering that low-fidelity data points are only accessible in $[0,1]\backslash\bar{I}$. We have thus chosen to study three cases, a very good low-fidelity emulator with $\bar{I} = \emptyset$ (for which the $Q^2_{l\to l}$ of the low-fidelity emulator is 0.99), a good emulator with $\bar{I} = [\frac{1}{3}, \frac{2}{3}]$ ($Q^2_{l\to l} = 0.98$) and a poor emulator with $\bar{I} = [\frac{3}{4}, 1]$ ($Q^2_{l\to l} = 0.84$).

Table 4.1 compares for these examples the different techniques, proposed in section 4.2, for $\bar{I} = \emptyset$. All methods have the same efficiency in terms of Q^2_{T}. The uncertainties of the predictions are overestimated for all methods. However, the Gauss-Hermite method has the best CP_α and the best prediction

Table 4.1: Error Q_T^2, coverage probability CP_α and mean prediction interval width MPIW_α for $\alpha = 80\%$ and for different methods of sampling. Here $\bar{I} = \emptyset$.

	Q_T^2	CP_α	MPIW_α
Gauss-Hermite $S = 5$	**0.99**	**0.88**	**0.083**
Mean-Std	**0.99**	0.97	0.095
Quantiles	**0.99**	0.90	0.105

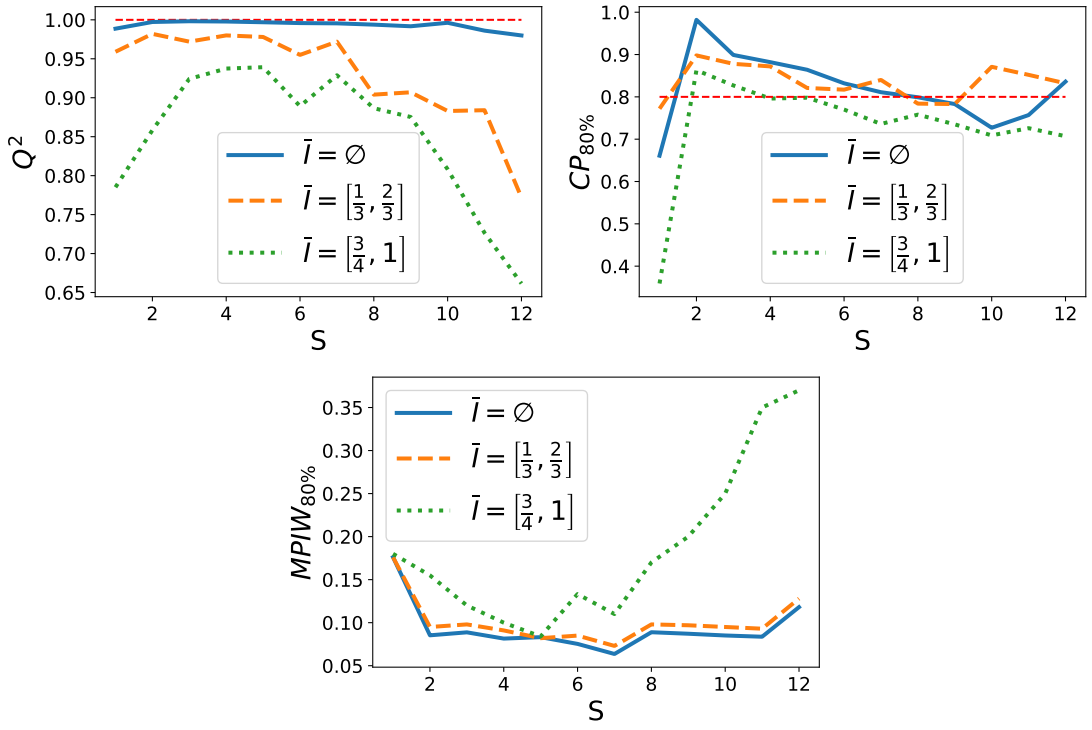


Figure 4.2: Error Q_{T}^2, coverage probability at 80% and $\mathrm{MPIW}_{80\%}$ as functions of S.

interval (i.e., the smallest mean prediction interval width MPIW_α). The quantiles method and the Mean-Std method also have reasonable CP_α, but their prediction intervals are larger. This leads us to use the Gauss-Hermite method. However, we note that all methods over-estimated the prediction interval. We believe this is due to the high regularity of the function to be predicted.

In fig. 4.2 we report the performances of the GPBNN method as functions of S between 1 and 12 for different $\bar{I}$. For $S = 1$ the uncertainty is underestimated and the accuracy of the prediction is not optimal, which shows that it is important to exploit the prediction interval predicted by the low-fidelity model. For $2 \leq S \leq 5$ the prediction is good, the error is constant and the Q_{T}^2 is maximal as seen in fig. 4.2(a). Figure 4.2(b) shows that the coverage probability is acceptable for $2 \leq S \leq 7$. Finally, the $\mathrm{MPIW}_{80\%}$ on fig. 4.2(c) is minimal for $3 \leq S \leq 7$. The best value of S is in $[2, 5]$ depending on the accuracy of the

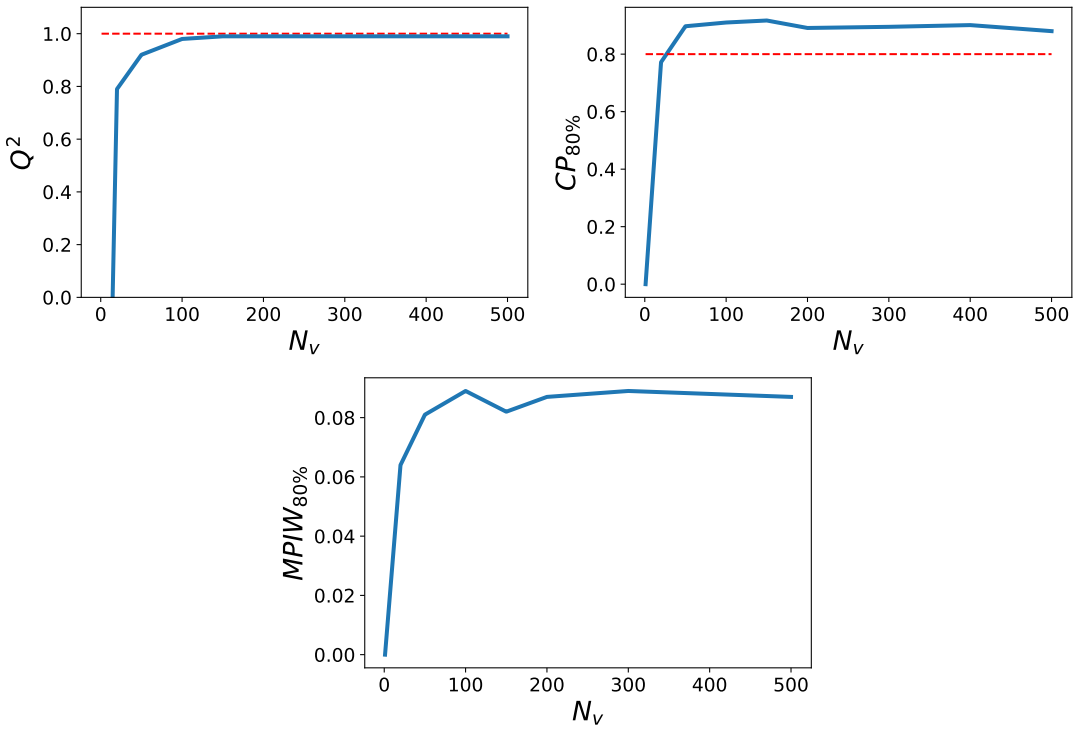


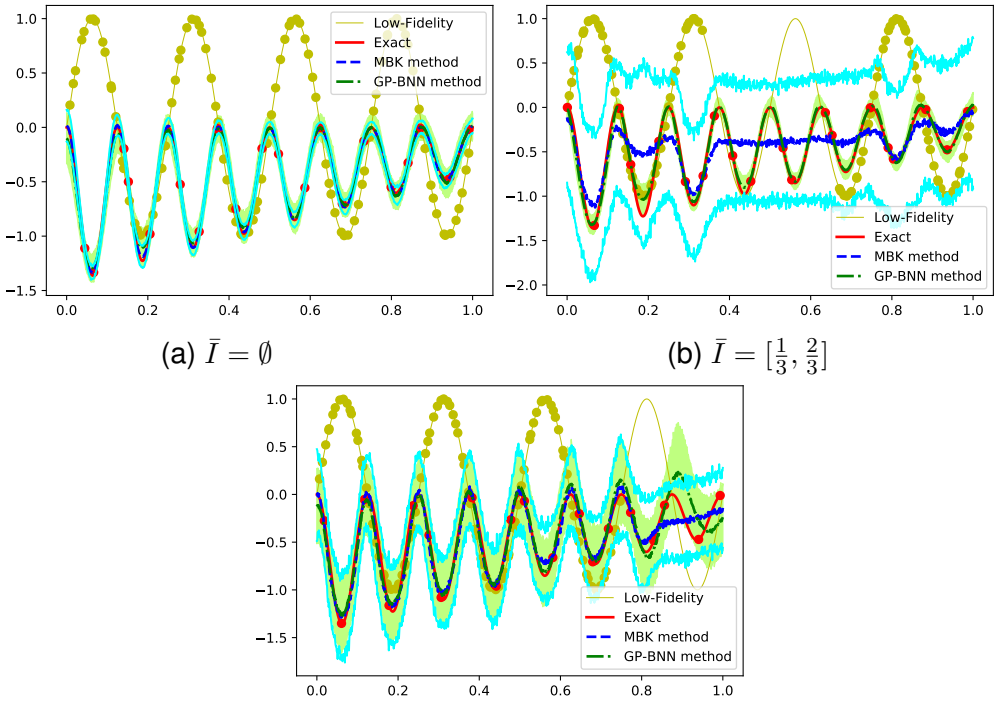
Figure 4.3: Error Q^2_{T}, coverage probability at 80% and $\mathrm{MPIW}_{80\%}$ as functions of N_v for $\bar{I} = \emptyset$.

low-fidelity emulator.

We have carried out a study on the best value of N_v. We thought that the best value would be the largest possible. Therefore, we tested for values of N_v ranging from 1 to 1000 for $\bar{I} = \emptyset$. For values of N_v greater than 200 the performance is identical as a function of N_v. For values below 200 a greater variability was found. We choose to use $N_v = 500$ to have a sufficient margin. In Figure 4.3 we have averaged the estimators for 5 independent training sets.

We now want to compare our GPBNN method with other state-of-the-art methods. The single-fidelity GP method used to emulate the high-fidelity code from the N_H high-fidelity points is called GP 1F. We use the implementation in (GPy, 2012). The multi-fidelity GP regression with autoregressive form introduced by (Kennedy & O'Hagan, 2000) and improved by (Le Gratiet & Garnier, 2013) is called autoregressive model AR(1). The method proposed in both (Cutajar et al., 2019; Perdikaris et al., 2017) is called DeepGP, we use the implementation from (Perdikaris et al., 2017) and the covariance given in (Cutajar et al., 2019) equation (11). The method from (Meng et al., 2021) is called MBK method. The MBK method is the combination of a fully connected NN for low-fidelity regression and BNN for high-fidelity. We implemented it using (Bingham et al., 2019). The methods that require the minimal assumptions on the function f_L and f_H are the GPBNN and MBK methods. This is the reason why they are the two methods that are compared in fig. 4.4.

First, the output laws for the MBK method and for the GPBNN presented in fig. 4.4 are different. For



(a) $\bar{I} = \emptyset$ (b) $\bar{I} = [\frac{1}{3}, \frac{2}{3}]$

(c) $\bar{I} = [\frac{3}{4}, 1]$

Figure 4.4: Comparison between the MBK method (in blue) and the GPBNN method with $S = 5$ (in green) for the estimation of the function f_H eq. (4.20) (in red). The high-fidelity data points are in red. The light colored lines (blue and green) plot the prediction intervals $\mathcal{I}_{80\%}(\mathbf{x})$.

the MBK method the posterior law is associated to the high-fidelity BNN knowing both the high-fidelity data and the low-fidelity model. Unlike the MBK method, the GBBNN method's output law represents the posterior law knowing high-fidelity points and the posterior distribution of the low-fidelity model built from the N_L low-fidelity points. In fig. 4.4 $S = 5$ because, as discussed above, this value seems to be the best value when the low-fidelity code is not so accurate.

The presented techniques for multi-fidelity regression are compared in tables 4.2 to 4.4. The GP 1F model and the multi-fidelity AR(1) model do not have good predictive properties. For the GP 1F model it is due to the lack of high-fidelity data and for the AR(1) model it is due to the strongly nonlinear relationship between the code levels. The three other methods perform almost perfectly when $\bar{I} = \emptyset$. The DeepGP is outstanding when $\bar{I} = \emptyset$ but the interaction between the codes (a quadratic form) is given exactly in the covariance structure which is a strong assumption in DeepGP that is hard to verify in practical applications. When $\bar{I}$ is non-empty the MBK method gives less accurate predictions because the method assumes strong knowledge of the low-fidelity code level. However, its prediction interval seems realistic for this example although too large. The DeepGP has reasonable errors but Tables 4.3 and 4.4 show that the predictive uncertainty of the DeepGP method does not fit the actual uncertainty of the prediction (it has poor coverage probability, either too large or too small). The GPBNN method has the smallest error (best Q^2_{T}) and it is predicting its accuracy precisely (it has good and nominal coverage probability; here $S = 5$) and the predictive variance and prediction interval width are small compared to the other methods: The GP 1F and AR(1) models have reasonable coverage probabilities but large mean prediction interval widths. For this simple illustrative example the GPBNN method seems to be the most suitable method.

Table 4.2: Q^2_{T} for different methods and segments $\bar{I}$ of missing low-fidelity values. Here $S = 5$.

$\bar{I}$	GP 1F	AR(1)	DeepGP	MBK	GPBNN
$\emptyset$	0.12	-0.29	$\mathbf{0.99}$	$\mathbf{0.99}$	$\mathbf{0.99}$
$\left[\frac{1}{3}, \frac{2}{3}\right]$	0.13	-0.34	$\mathbf{0.98}$	0.90	$\mathbf{0.98}$
$\left[\frac{3}{4}, 1\right]$	0.12	-0.29	0.90	0.51	$\mathbf{0.93}$

Table 4.3: Coverage probability CP_α for $\alpha = 80\%$ and for different methods and segments $\bar{I}$ of missing low-fidelity values. Here $S = 5$.

$\bar{I}$	GP 1F	AR(1)	DeepGP	MBK	GPBNN
$\emptyset$	**0.82**	**0.82**	0.99	0.76	**0.88**
$\left[\frac{1}{3}, \frac{2}{3}\right]$	**0.78**	**0.79**	0.60	0.84	**0.83**
$\left[\frac{3}{4}, 1\right]$	**0.78**	**0.82**	0.62	0.86	**0.78**

Table 4.4: Mean prediction interval width MPIW_α for $\alpha = 80\%$ and for different methods and segments $\bar{I}$ of missing low-fidelity values. Here $S = 5$.

$\bar{I}$	GP 1F	AR(1)	DeepGP	MBK	GPBNN
$\emptyset$	0.44	0.55	**0.002**	0.037	**0.083**
$\left[\frac{1}{3}, \frac{2}{3}\right]$	0.42	0.45	**0.010**	0.36	**0.082**
$\left[\frac{3}{4}, 1\right]$	0.44	0.45	0.097	0.31	**0.084**

4.3.2 2D function approximation

The CURRIN function is a two-dimensional function, with $\mathbf{x} \in [0,1]^2$. This function is commonly used to simulate computer experiments (Cutajar et al., 2019). The high- and low-fidelity functions are:

$$f_H(\mathbf{x}) = \left[1 - \exp\left(-\frac{1}{2x_2}\right)\right] \frac{2300x_1^3 + 1900x_1^2 + 2092x_1 + 60}{100x_1^3 + 500x_1^2 + 4x_1 + 20}, \tag{4.24}$$

$$\begin{aligned} f_L(\mathbf{x}) =& \frac{1}{4}\left[f_H(x_1 + \delta, x_2 + \delta) + f_H(x_1 + \delta, \max(0, x_2 - \delta))\right] \\ &+ \frac{1}{4}\left[f_H(x_1 - \delta, x_2 + \delta) + f_H(x_1 - \delta, \max(0, x_2 - \delta))\right], \end{aligned} \tag{4.25}$$

with $\mathbf{x} = [x_1, x_2]$ and δ the filter parameter. In (Cutajar et al., 2019) we have $\delta = 0.05$ and this gives very small differences between the two functions and the prediction of the high-fidelity function by the low-fidelity one has $Q^2_{l\to h} = 0.98$. In the following we set $\delta = 0.1$, and then $Q^2_{l\to h} = 0.87$. An additive Gaussian noise is added to the low-fidelity code. The noise has a zero mean and a variance equal to the empirical variance of the signal 0.08.

In this example we also consider that the low-fidelity code is costly and we only have a small number of low-fidelity points: $N_L = 25$ and $N_H = 15$. The high- and low-fidelity points are chosen by maximin Latin Hypercube Sampling (LHS), see (Helton & Davis, 2003). LHS is a space-filling design of experiment. The test set is composed of 1000 independent points following a random uniform law on $[0,1]^2$.

The kernel used for GP regression low-fidelity is a Matérn $5/2$ covariance function. The predictive error for the GP regressor of the low-fidelity model is $Q^2 = 0.91$ using a nugget effect in the Gaussian

Table 4.5: Comparison of the multi-fidelity methods on the CURRIN function via Q^2_{T}, $\mathrm{CP}_{80\%}$ and $\mathrm{MPIW}_{80\%}$.

	GP 1F	AR(1)	DeepGP	MBK	GPBNN
Q^2_{T}	0.73	0.80	0.29	0.27	0.88
$\mathrm{CP}_{80\%}$	0.68	0.80	0.62	0.57	0.80
$\mathrm{MPIW}_{80\%}$	0.5	1.0	0.13	1.9	0.51

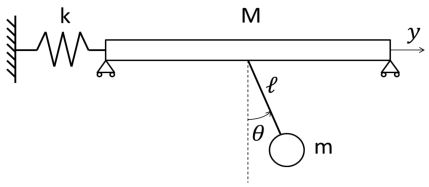


Figure 4.5: Illustration of the double pendulum.

process regression. The BNN is defined with $N_l = 40$ neurons and the mean and variance are evaluated by eqs. (4.18) and (4.19) with $N_v = 500$. N_l could be chosen arbitrarily but the fact that we are in a small data context leads us to choose a small value. It is possible to use cross-validation to choose N_l, but the computer cost would be here prohibitive. The previous example discussed in section 4.3.1 suggests choosing S between 3 and 5 for the low-fidelity surrogate model sampling. In this example, due to the large low-fidelity error the model needs a large value of S to account appropriately for the uncertainty and we choose $S = 5$.

All methods have been compared in table 4.5. The GPBNN model in this example seems to be accurate in terms of Q^2_{T} and in uncertainty quantification. It is much better than the GP 1F model (single-fidelity GP model built with the high-fidelity data). We presume that it is due to the lack of high-fidelity data. AR(1) model gives better but not satisfying results. This is expected due to the non-linearity between codes. The results of the DeepGP method and the MBK method are worse than the one of the GP 1F in error and in uncertainty. For the DeepGP this can be understood by the fact that the covariance is not well adapted, see (Cutajar et al., 2019). And for the MBK the lack of points leads to a very poor optimisation of the hyper-parameters.

4.3.3 Double pendulum

The double pendulum system can be seen as a dual-oscillator cluster, see Figure 4.5. The system is presented in (Perrin, 2020). The inputs of the system are of dimension 5, including $(k, M, \theta_0, \dot{\theta}_0, y_0)$. The output is of dimension 1, it is the maximum along the axis y of the position of the mass m in the first 10 seconds. We have two codes: the high-fidelity code numerically solves Newton's equation. The low-fidelity code simplifies the equation, by linearisation for small angles of the pendulum motion, and

Table 4.6: Comparison of the multi-fidelity methods on the pendulum example via Q^2_{T}, $\mathrm{CP}_{80\%}$ and $\mathrm{MPIW}_{80\%}$.

	GP 1F	AR(1)	DeepGP	MBK	GPBNN
Q^2_{T}	0.93	0.94	$\mathbf{0.95}$	0.54	$\mathbf{0.95}$
$\mathrm{CP}_{80\%}$	$\mathbf{0.81}$	$\mathbf{0.78}$	0.62	0.88	$\mathbf{0.80}$
$\mathrm{MPIW}_{80\%}$	0.154	0.146	0.069	0.859	$\mathbf{0.101}$

solves the system. Our goal is to build a surrogate model of the high-fidelity code using $N_L = 100$ and $N_H = 20$, with maximin LHS sampling. The input parameters are the mass $M \in [10, 12]$, the spring stiffness $k \in [1, 1.4]$, the initial angle of the pendulum $\theta_0 \in [\frac{\pi}{4}, \frac{\pi}{3}]$, the initial derivative $\dot{\theta_0} \in [0, \frac{1}{10}]$ and the initial position of the mass $y_0 \in [0, 0.2]$. The fixed parameters are $\dot{y_0} = 0$, the gravitational acceleration $g = 9.81$, the length of the pendulum $l = 2$ and its mass $m = 0.5$. The output is the maximum in time of the amplitude of the mass m. The error is computed on a test set, different for each learning set, of 64 samples uniformly distributed on the input space. To evaluate each model we use 5 independent learning and test sets.

The $Q^2_{l\to l}$ for the low-fidelity surrogate model with a Matérn $5/2$ as kernel for the GP is 0.98. The BNN is defined with $N_l = 30$ and $N_v = 500$, and we use the GPBNN strategy with $S = 5$. The results are presented in Table 4.6. The prediction of the MBK method is not accurate compared to all the other methods. We think that this is due to the small data set regarding the dimension of the BNN's input. However, the uncertainty of prediction is still accurate even if the prediction interval is large compared to the other methods (i.e. the coverage probability is close to the target value). This result is very surprising for us in regard with the poor quality of the low-fidelity model, that has a $Q^2_{l\to l}$ of 0.7. The DeepGP model shares the best predictive error with GPBNN. The single fidelity and the AR(1) models display slightly larger errors. The $\mathrm{CP}_{80\%}$ values are in the target area for GP1F and AR(1) but they are associated with large prediction intervals. The DeepGP clearly underestimates the uncertainty of its predictions. $\mathrm{CP}_{80\%}$ value is good for the GPBNN and close to the target value. Moreover, the prediction interval is the smallest of all methods. On this real life example the GPBNN is competitive compared to other state-of-the-art methods.

4.4 Conclusion

Our main focus in this chapter is to give the Gaussian Process regression posterior distribution of a low-fidelity model as input to a Bayesian neural network for multi-fidelity regression. Different methods are proposed and studied to transfer the uncertain predictions of the low-fidelity emulator to the high-fidelity

one, which is crucial to obtain minimal predictive errors and accurate predictive uncertainty quantification. The Gauss-Hermite quadrature method is shown to significantly improve the predictive properties of the BNN. The conducted experiments show that the GPBNN method is able to process noisy and real life problems. Moreover, the comparison with some stat-of-the-art methods for multi-fidelity surrogate model highlight the precision in prediction and in uncertainty quantification.

It is possible to extend this method to more than two levels of codes. This is done by considering the output of the BNN as an input to the higher fidelity model. Two different models can therefore be considered. One uses the output of the BNN directly and by sampling gives it as input to the higher fidelity BNN. We can also generate realisations of our surrogate model and consider them as realisations of the low-fidelity code and thus apply the GPBNN method.

The interest of combining regression method for multi-fidelity surrogate modeling is not to be proved, but this manuscript adds the heterogeneity of models for multi-fidelity modeling. To be able to combine heterogeneous models into one model and to consider the uncertainty between them is one of the keys to adapt the multi-fidelity surrogate model to a real-life regression problem.

It exists several methods that allow large learning sets for GP regression. This approach could be used for the GP part of the GPBNN. With more time in the training the BNN part will be able to deal with many points, even if this ability is less critical because $N_L \gg N_H$. Consequently, the GPBNN can be extended in order to tackle larger data sets.

Existing autoregressive models and Deep GP can only be used for low-dimensional outputs. We wish to extend the method to high-dimensional outputs. Dimension reduction techniques have already been used as principal component analysis or autoencoder, as well as tensorized covariance methods (Perrin, 2020) and section 1.1.3 that remain to be extended to the multi-fidelity context. However, neural networks are known to adapt to high-dimensional outputs. We should further investigate how to build multi-fidelity surrogate models with functional input/output in the context of small data. The ability to construct models that are tractable for high-dimensional input and output is key for further research.

Chapter 5

Multi-fidelity wavelet Gaussian process regression

The methods for multi-fidelity regression presented in chapters 3 and 4 are well suited to the case of regression of one-dimensional outputs with multi-dimensional inputs. However, both methods based on Gaussian processes and neural networks are limited in the number of training points or uncertainty estimation. Moreover, with the improvement of numerical computation capacities, the output of the codes can be of higher dimension and in particular time-series. This leads to an increase in the number of training points. In this chapter we propose a method to perform multi-fidelity regression with time-series output but limiting the number of training points.

Fourier transforms and wavelet transforms are commonly used to compress information from natural signals. These methods allow working in known projection spaces which have the advantage to concentrate the information on a few coefficients. Moreover, these methods are invertible, which is essential for regression. We have chosen not to study the Fourier transform because we are interested in signals that are not periodic, but this method is efficient mainly in this context. We will therefore decompose our output signals on a wavelet basis.

In this chapter we define an object called Wavelet Gaussian process equivalent to a Gaussian process but in wavelet space. The interest is to concentrate the information, so that the training can be carried out on the points with the most information. As the information is very concentrated we can dispense with the information on the less informative points. The key is the construction of a covariance kernel in the wavelet space. Using this kernel the selection of the training points is made with the addition of the data expressed in the wavelet space.

Although this work is original, there is some work that has been done on Gaussian process regression and wavelets. It is even interesting to note that work has already been done on multi-fidelity and wavelets. In the manuscript (Zertuche, 2015) a framework for a multi-fidelity model is proposed but unlike this chap-

ter the work focuses mainly on wavelet decomposition. We can also note the work of (De Iozzo, 2013) in which multi-fidelity and wavelets are considered together, even if in this case multi-fidelity is not considered under the same perspective as in this manuscript. Indeed, this study is based on a high-fidelity code and several non-hierarchical low-fidelity codes. The models are aggregated using Gaussian processes. Haar wavelet transformations are also found in (De Iozzo, 2013) for heteroscedastic regression. In this study different models are linked using Haar wavelets. A simple-fidelity regression model in a wavelet basis is presented in (Balafas, Kiremidjian, & Rajagopal, 2018). This paper proposes a model with independence of coefficients, which is not the one favoured here. For the modeling of non-stationary processes we can quote (Hu & Wang, 2015).

In this chapter the Wavelet Gaussian process is presented. Then the specific case of Wavelet Gaussian process with Haar wavelet and Matérn kernel is presented. Finally, the multi-fidelity AR(1) model is presented adapted to Wavelet Gaussian processes.

5.1 Wavelet Gaussian process

Let us assume that our prior knowlege of the code $z(\mathbf{x}, t)$ has the form of a Gaussian process $Z(\mathbf{x}, t)$ with tensorized covariance. We express Z as:

$$Z(\mathbf{x}, t) \sim \mathcal{GP}\left(\mu(\mathbf{x}, t), k(\mathbf{x}, t, \mathbf{x}', t')\right), \tag{5.1}$$

with $t, t' \in \mathbb{R}$, the mean function is μ and the kernel function can be expressed as $k(\mathbf{x}, t, \mathbf{x}', t') = k_x(\mathbf{x}, \mathbf{x}')k_t(t, t')$. k_t is stationary, it means that $k_t(t, t') = F(t - t')$. The spectral density of Z for a fixed $\mathbf{x}$ is $k_x(\mathbf{x}, \mathbf{x})\hat{F}$, where $\hat{F}$ is the Fourier transform of F. This is a tensorized framework for covariance kernel. The kernels k_x and F can be chosen independently. The process Z is stationary if we assume that the chosen kernels are stationary. The Gaussian process Z can be expressed as linear combination of wavelet function and scale function. We call this the Wavelet Gaussian process (WGP). In the following we will present this object.

In the following we will assume that the covariance is tensorized so that we will be able to eliminate the $\mathbf{x}$. Moreover, for the calculation of the covariance we assume a continuous time t. However, we know that for the application the time is on a regular grid. Consequently, the model is defined for the case of time on a grid. So we have a discretized model in t, but the covariance formula is expressed for any value of t which allows to obtain it on all the points of the grid. In fact, the considered process is discretized, but we have an expression for its continuous Fourier transform. This is what motivates the use of continuous notations.

Definition 5.1.1. *(Wavelet Gaussian process) Let ψ and ϕ be respectively a wavelet function and the scale function. Let Z be a Gaussian process with tensorized covariance $k = k_x F$, we construct an associated*

Gaussian process A such that the process Z can be expressed as:

$$Z(\mathbf{x},t) = A(\mathbf{x},0,0)\phi(t) + \sum_{j,k\in I} A(\mathbf{x},j,k)\psi_{j,k}(t), \tag{5.2}$$

where I is defined in eq. (2.42)*, $I = \bigcup_{M\to\infty}\left\{j,k | j = 1,\cdots,M \ \ \& \ \ k = 0,\cdots,2^{M-j}-1\right\}$. A is called the (ϕ,ψ) WGP of Z.*

Theorem 5.1.2. *If Z is a Gaussian process with covariance $k_x(\mathbf{x},\mathbf{x}')F(t-t')$, then the ($\phi$,$\psi$) WGP A of Z is a Gaussian process with covariance $k_A(\mathbf{x},\mathbf{x}',j,k,j',k') = k_x(\mathbf{x},\mathbf{x}')k_{WGP}(j,k,j',k')$:*

$$k_{WGP}(j,k,j',k') = \begin{cases} \frac{1}{2\pi}\int \hat{\phi}(\xi)\overline{\hat{\phi}(\xi)}\hat{F}(\xi)\mathrm{d}\xi & j = j' = 0 \\ \frac{\sqrt{2^j}}{2\pi}\int e^{-i\xi 2^j k}\hat{\psi}(2^j\xi)\overline{\hat{\phi}(\xi)}\hat{F}(\xi)\mathrm{d}\xi & j = 0 \ \textit{and} \ j' \neq 0 \\ \frac{\sqrt{2^j}}{2\pi}\int e^{-i\xi 2^{j'}k'}\hat{\psi}(2^{j'}\xi)\overline{\hat{\phi}(\xi)}\hat{F}(\xi)\mathrm{d}\xi & j \neq 0 \ \textit{and} \ j' = 0 \\ \frac{\sqrt{2^j 2^{j'}}}{2\pi}\int e^{-i\xi(2^j k - 2^{j'}k')}\hat{\psi}(2^j\xi)\overline{\hat{\psi}(2^{j'}\xi)}\hat{F}(\xi)\mathrm{d}\xi & \textit{otherwise} \end{cases} \tag{5.3}$$

Proof. In the proof we limit ourselves to the case $j \neq 0$ et $j' \neq 0$. The other cases are variations of this one.

This is done in order to separate the calculation of the covariance in $\mathbf{x}$ which is identical to the classical case here and the covariance in t which is specific for the WGP. For a fixed $\mathbf{x}$ let us consider the Gaussian process $Z_x(t) = \frac{Z(t,\mathbf{x})}{\sqrt{k_x(\mathbf{x},\mathbf{x})}}$ of stationary covariance function F and spectral density $\hat{F}$:

$$\hat{F}(\xi) = \int F(t)e^{-i\xi t}\mathrm{d}t, \tag{5.4}$$

and:

$$F(t) = \frac{1}{2\pi}\int \hat{F}(\xi)e^{i\xi t}\mathrm{d}\xi. \tag{5.5}$$

The spectral representation of $Z_x(t)$ can be expressed as:

$$Z_x(t) = \frac{1}{2\pi}\int \sqrt{\hat{F}(\xi)}e^{i\xi t}\mathrm{d}B_\xi, \tag{5.6}$$

with B_ξ a complex Wiener process. We have:

$$\mathbb{E}\left[\mathrm{d}B_\xi \overline{\mathrm{d}B'_\xi}\right] = 2\pi\delta(\xi-\xi')\mathrm{d}\xi\mathrm{d}\xi', \tag{5.7}$$

with δ the Dirac delta function.

If we combine eqs. (2.17) and (5.6) we can get an expression wavelet coefficients.

$$\begin{aligned} W_{j,k}^{Z} &= \int Z_x(t)\psi_{j,k}(t)dt && (5.8)\\ &= \frac{1}{2\pi}\iint \sqrt{\hat{F}(\xi)}e^{i\xi t}\psi_{j,k}(t)\mathrm{d}t\mathrm{d}B_\xi && (5.9)\\ &= \frac{1}{2\pi}\int \sqrt{\hat{F}(\xi)}\overline{\hat{\psi}_{j,k}}(\xi)\mathrm{d}B_\xi && (5.10) \end{aligned}$$

We consider that the mean of the Gaussian process is null. The covariance function k is then:

$$k_{WGP}(j,k,j',k') = \mathrm{Cov}\left(W_{j,k}^{Z}, W_{j',k'}^{Z}\right) = \mathbb{E}\left[W_{j,k}^{Z}\overline{W_{j',k'}^{Z}}\right]. \tag{5.11}$$

Consequently, eq. (5.11) gives:

$$k_{WGP}(j,k,j',k') = \frac{1}{2\pi}\int \hat{\psi}_{j,k}(\xi)\overline{\hat{\psi}_{j',k'}(\xi)}\hat{F}(\xi)\mathrm{d}\xi \tag{5.12}$$

with $\overline{\hat{\psi}_{j,k}(\xi)}$ complex conjugate of $\hat{\psi}_{j,k}(\xi)$. Then the covariance can be expressed as a function of the Gaussian process spectrum and the wavelet function:

$$k_{WGP}(j,k,j',k') = \frac{\sqrt{2^j 2^{j'}}}{2\pi}\int e^{-i\xi\left(2^j k - 2^{j'}k'\right)}\hat{\psi}(2^j\xi)\overline{\hat{\psi}(2^{j'}\xi)}\hat{F}(\xi)\mathrm{d}\xi. \tag{5.13}$$

Note that eq. (5.13) applies for any covariance kernel and any orthogonal wavelet basis. □

5.1.1 The examples of Daubechies wavelet transform of GP with Matérn kernel

In this subsection we will consider that the covariance kernel is a Matérn 5/2 function and that the wavelet basis considered is Haar. The Matérn 5/2 is presented in chapter 1. The expression of its spectral density is important in this context. The Haar basis is presented in section 2.2.2. This transformation was presented because it is a classic wavelet transformation. Its Fourier transform has a simple analytical expression.

Theorem 5.1.3. *If Z a Gaussian process with covariance $k_x(\mathbf{x},\mathbf{x}')F(t-t')$, then the Haar WGP of Z is A. If F be a Matern 5/2 kernel, then the covariance of A is $k_A(\mathbf{x},\mathbf{x}',j,k,j',k') = k_x(\mathbf{x},\mathbf{x}')k_{WGP}(j,k,j',k')$:*

$$k_{WGP}(j,k,j',k') = \frac{200\sqrt{5}\sigma^2}{3\pi\sqrt{2^j 2^{j'}}}\left[-\sum_{i\in N_{j,j'}}\frac{\alpha_i\pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}}{1000}\left(3\sqrt{5}l^2 + 7|\gamma_i|l + \sqrt{5}\gamma_i^2\right) - \sum_{i\in N_{j,j'}}\frac{\pi\alpha_i|\gamma_i|l}{125}\right], \tag{5.14}$$

with values of α_i, γ_i and $N_{j,j'}$ depending on j,k,j',k' given in tables 5.1 to 5.4.

Proof. The first part of the proof is to show that the covariance function can be written as:

$$k_{WGP}(j,k,j',k') = \frac{200l\sqrt{5}\sigma^2}{3\pi\sqrt{2^j 2^{j'}}}\int_0^\infty \frac{\sum_i \alpha_i \cos(\xi\gamma_i)}{\xi^2\left(l^2\xi^2+5\right)^3}\mathrm{d}\xi. \tag{5.15}$$

The second part of the proof is to determine the value of the integral:

$$D = 2l\int_0^\infty \frac{\sum_i \alpha_i \cos(\xi\gamma_i)}{\xi^2\left(l^2\xi^2+5\right)^3}\mathrm{d}\xi.$$

In order to simplify the expressions we focus only on the case $j \neq 0$ and $j' \neq 0$. The Matérn 5/3 kernel is:

$$F(\tau) = \left(1+\frac{\sqrt{5}|\tau|}{\ell}+\frac{5\tau^2}{3\ell^2}\right)\exp\left(-\frac{\sqrt{5}|\tau|}{\ell}\right), \quad \hat{F}_{\frac{5}{2}}(\xi) = \frac{400\sqrt{5}}{\ell^5}\left(\frac{3}{\ell^2}+\xi^2\right)^{-3}. \tag{5.16}$$

We compute eq. (5.13) for the kernel (Matérn 5/2). Equation (5.13) becomes:

$$k_{WGP}(j,k,j',k') = \frac{200l}{3\pi}\sqrt{5}\sigma^2\sqrt{2^j 2^{j'}}\int e^{-i\xi\left(2^j k-2^{j'}k'\right)}\hat{\psi}(2^j\xi)\overline{\hat{\psi}(2^{j'}\xi)}\frac{\mathrm{d}\xi}{\left(\xi^2 l^2+5\right)^3}. \tag{5.17}$$

The Haar wavelet basis in eq. (5.13) is introduced. This gives :

$$k_{WGP}(j,k,j',k') = \frac{3200l\sqrt{5}\sigma^2}{3\pi\sqrt{2^j 2^{j'}}}\int \sin^2\left(\frac{2^j\xi}{4}\right)\sin^2\left(\frac{2^{j'}\xi}{4}\right)\frac{e^{-i\xi\left(2^j(k+\frac{1}{2})-2^{j'}(k'+\frac{1}{2})\right)}}{\xi^2\left(l^2\xi^2+5\right)^3}\mathrm{d}\xi \tag{5.18}$$

We can note that:$\Im\left(\exp -i\xi(2^j k-2^{j'}k')\right) = -\Im\left(\exp i\xi(2^j k-2^{j'}k')\right)$, with $\Im$ a function that returns only the imaginary part of the input element. Consequently, the imaginary part of $k_{WGP}(j,k,j',k')$ is null and $k_{WGP}(j,k,j',k')$ can therefore be written as:

$$k_{WGP}(j,k,j',k') = \frac{3200l\sqrt{5}\sigma^2}{3\pi\sqrt{2^j 2^{j'}}}2\int_0^\infty \sin^2\left(\frac{2^j\xi}{4}\right)\sin^2\left(\frac{2^{j'}\xi}{4}\right)\frac{\cos\left(\xi\left(2^j(k+\frac{1}{2})-2^{j'}(k'+\frac{1}{2})\right)\right)}{\xi^2\left(l^2\xi^2+5\right)^3}\mathrm{d}\xi \tag{5.19}$$

In order to simplify the integration, we try to have an expression of α_i and γ_i such that:

$$\sin^2\left(\frac{2^j\xi}{4}\right)\sin^2\left(\frac{2^{j'}\xi}{4}\right)\cos\left(\xi\left(2^j(k+\frac{1}{2})-2^{j'}(k'+\frac{1}{2})\right)\right) = \sum_i \alpha_i\cos(\xi\gamma_i). \tag{5.20}$$

The full computation is given in appendix C.2. The same can be proceeded for ($j = j' = 0$), ($j \neq 0$ and $j' = 0$) and ($j = 0$ and $j' \neq 0$). This gives tables 5.1 to 5.4. We have completed the first part of the proof.

Now we compute the second part of the integral:

$$D = 2l\int_0^\infty \frac{\sum_i \alpha_i \cos(\xi\gamma_i)}{\xi^2\left(l^2\xi^2+5\right)^3}\mathrm{d}\xi. \tag{5.21}$$

In order to output the value of the integral it is necessary to ensure that the product of the cosine with a rational fraction is integrable. To do this we decompose our rational fraction into a sum of 4 fractions:

$$\frac{1}{\xi^2\left(\xi^2 l^2+5\right)^3} = -\frac{l^2}{125\left(l^2\xi^2+5\right)} - \frac{l^2}{25\left(l^2\xi^2+5\right)^2} - \frac{l^2}{5\left(l^2\xi^2+5\right)^3} + \frac{1}{125\xi^2} \tag{5.22}$$

With using eqs. (5.21) and (5.22) we are able to simplify the integral to compute. To be able to compute the value of the integral, the 4th term in cosine is modified by the property $\sum_i \alpha_i = 0$ we have $\sum_i \alpha_i cos(\gamma_i\xi) = \sum_i \alpha_i \left(cos(\gamma_i\xi) - 1\right)$. Then:

$$D = 2l^3 \int_0^\infty \sum_i \left(-\frac{\alpha_i \cos(\xi\gamma_i)}{125\left(l^2\xi^2+5\right)} - \frac{\alpha_i \cos(\xi\gamma_i)}{25\left(l^2\xi^2+5\right)^2} - \frac{\alpha_i \cos(\xi\gamma_i)}{5\left(l^2\xi^2+5\right)^3} - \frac{\alpha_i \sin^2(\frac{\xi\gamma_i}{2})}{125\xi^2 l^2} \right) \mathrm{d}\xi. \tag{5.23}$$

Since $l > 0$ and $\gamma_i \in \mathbb{R}$, we compute the 4 integrals:

$$\int_0^\infty \frac{\cos(\xi\gamma_i)}{l^2\xi^2+5}\mathrm{d}\xi = \frac{\pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}}{2\sqrt{5}l}, \tag{5.24}$$

$$\int_0^\infty \frac{\cos(\xi\gamma_i)}{(l^2\xi^2+5)^2}\mathrm{d}\xi = \frac{\pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}(5|\gamma_i| + \sqrt{5}l)}{100l^2}, \tag{5.25}$$

$$\int_0^\infty \frac{\cos(\xi\gamma_i)}{(l^2\xi^2+5)^3}\mathrm{d}\xi = \frac{\pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}(\sqrt{5}|\gamma_i|(5\gamma_i^2+3l^2) + 15\gamma_i^2 l)}{2000|\gamma_i|l^3}, \tag{5.26}$$

$$\int_0^\infty \frac{\sin^2(\xi\gamma_i/2)}{\xi^2}\mathrm{d}\xi = \frac{\pi|\gamma_i|}{4}. \tag{5.27}$$

From the previous results we have:

$$\begin{aligned} D = &-\sum_i \frac{\alpha_i l^2 \pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}}{125\sqrt{5}} - \sum_i \frac{\alpha_i l \pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}(5|\gamma_i| + \sqrt{5}l)}{1250} \\ &- \sum_i \frac{\alpha_i \pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}(\sqrt{5}|\gamma_i|(5\gamma_i^2+3l^2) + 15\gamma_i^2 l)}{5000|\gamma_i|} - \sum_i \frac{\pi\alpha_i|\gamma_i|l}{125}. \end{aligned} \tag{5.28}$$

Straightforward computations give:

$$D = -\sum_i \left(\alpha_i \pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}\right) \left[\frac{60|\gamma_i|l^2 + 35\sqrt{5}\gamma_i^2 l + 5|\gamma_i|(5\gamma_i^2+3l^2)}{5000\sqrt{5}|\gamma_i|}\right] - \sum_i \frac{\pi\alpha_i|\gamma_i|}{125}, \tag{5.29}$$

$$D = -\sum_i \frac{\alpha_i \pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}}{1000}\left(3\sqrt{5}l^2 + 7|\gamma_i|l + \sqrt{5}\gamma_i^2\right) - \sum_i \frac{\pi\alpha_i|\gamma_i|l}{125}, \tag{5.30}$$

which completes the proof. □

i	1	2
α_i	$\frac{1}{2}$	$-\frac{1}{2}$
γ_i	0	1

Table 5.1: Values of γ_i and α_i for $j = j' = 0$. $N_{0,0} = \{1, 2\}$.

i	1	2	3	4	5	6
α_i	2	-2	-1	-1	1	1
γ_i	$-2^{j'}k' - \frac{1}{2}$	$-2^{j'}k' + \frac{1}{2}$	$2^{j'}\left(\frac{1}{2} - k'\right) - \frac{1}{2}$	$2^{j'}\left(\frac{1}{2} + k'\right) + \frac{1}{2}$	$2^{j'}\left(\frac{1}{2} - k'\right) + \frac{1}{2}$	$2^{j'}\left(\frac{1}{2} + k'\right) - \frac{1}{2}$

Table 5.2: Values of γ_i and α_i for $j = 0$ and $j' \neq 0$. $N_{0,j'} = \{1, 2, 3, 4, 5, 6\}$.

The computations of tables 5.1 to 5.4 are available in appendix C.2.

It should be noted that theorem 5.1.3 is necessary for the generation of WGP. This provides the covariance matrix in the wavelet space of a given Gaussian process. It can be useful for some applications, such as optimization or hyperparameters selection, to get the derivative of the covariance. The derivative of the covariance can be found in appendix C.1. However, for the regression it is necessary to make some further small adaptations.

Generate wavelet Gaussian process The purpose of this paragraph is to show the difference between the generation of a GP and a WGP. In this paragraph we fix $\mathbf{x}$. The GP is generated in the time domain. The WGP is generated in the wavelet domain but thanks to the inverse wavelet transform we are able to have realizations of the WGP in the time domain. The two GP are equivalent, as a consequence the covariance in the time domain should be the same for the GP and the WGP. The comparison in the time domain of the empirical covariance function obtained from a GP and a WGP with Matèrn kernel is given at fig. 5.1. The two models are absolutely identical, it is just the empirical estimate of the covariance matrix that is different from the theoretical matrix. It can be seen that the covariance of GP and WPG in the time domain are equal, within to numerical errors.

The Goal is to generate a GP in the time domain with the framework of a WGP in the wavelet domain. This means that instead of computing the randomness on $Z(\mathbf{x}, t)$ we compute the randomness on

i	1	2	3	4	5	6
α_i	2	-2	-1	-1	1	1
γ_i	$-2^{j}k - \frac{1}{2}$	$-2^{j}k + \frac{1}{2}$	$2^{j}\left(\frac{1}{2} - k\right) - \frac{1}{2}$	$2^{j}\left(\frac{1}{2} + k\right) + \frac{1}{2}$	$2^{j}\left(\frac{1}{2} - k\right) + \frac{1}{2}$	$2^{j}\left(\frac{1}{2} + k\right) - \frac{1}{2}$

Table 5.3: Values of γ_i and α_i for $j \neq 0$ and $j' = 0$, $N_{j,0} = \{1, 2, 3, 4, 5, 6\}$.

i	1	2	3	4	5	6	7	8	9
α_i	4	-4	-2	-2	-2	1	1	1	1
$\gamma_i - (2^j k - 2^{j'} k')$	0	$\frac{2^j}{2}$	$-\frac{2^j}{2}$	$-\frac{2^{j'}}{2}$	$\frac{2^{j'}}{2}$	$\frac{2^j}{2} - \frac{2^{j'}}{2}$	$-\frac{2^j}{2} + \frac{2^{j'}}{2}$	$\frac{2^j}{2} + \frac{2^{j'}}{2}$	$-\frac{2^j}{2} - \frac{2^{j'}}{2}$

Table 5.4: Values of γ_i and α_i for $j \neq 0$ and $j' \neq 0$.$N_{j,j'} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

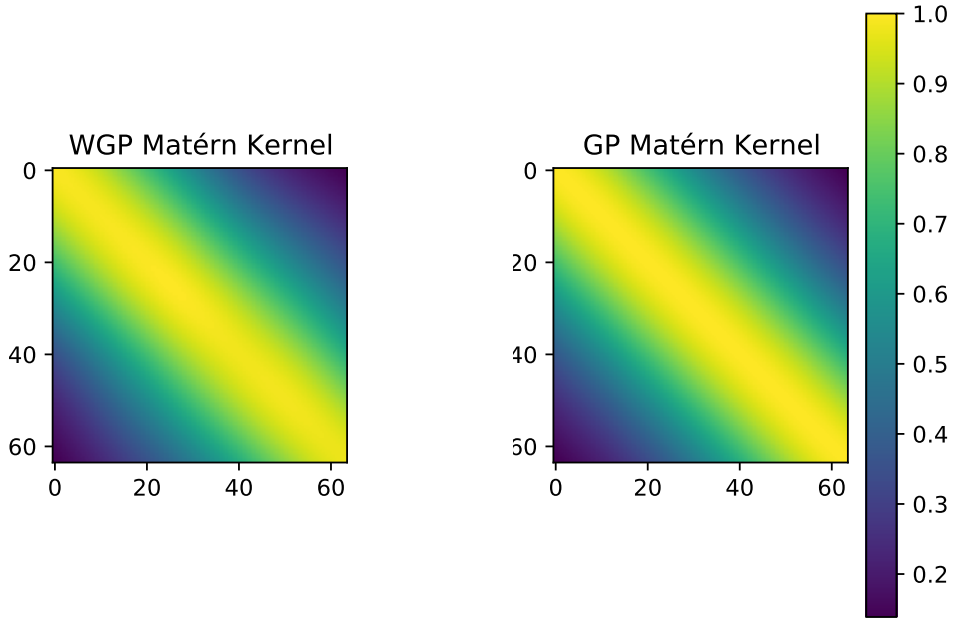


Figure 5.1: Comparison in the time-domain of the empirical covariance functions obtained from a WGP and from a GP. We plot the covariance of the Gaussian vectors on a given time grid obtained from a WGP (left) or form a GP. The time grid is $t = 1, \cdots, 64$ and $t' = 1, \cdots, 64$, the correlation length is $\ell = \frac{1}{2} * 64$, the variance is $\sigma = 1$. 1000 realizations are used to compute empirical covariance matrices.

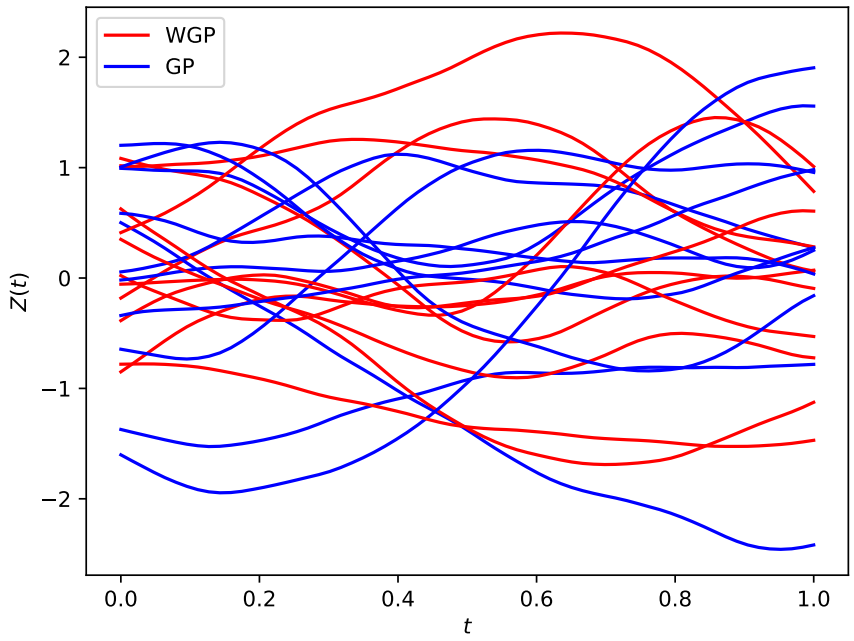


Figure 5.2: Realizations of a WGP and a GP with same covariance in the time domain. The time grid is $t = 0, \frac{1}{63}, \cdots, 1$, the correlation length is $\ell = \frac{1}{2}$, the variance is $\sigma = 1$

$A(\mathbf{x}, j, k)$. Figure 5.2 presents the same GP in the time domain, but one is generated in the time domain and the other one, the WGP, is generated in the wavelet domain. We see that both are close from each other. This allows us to consider the regression by WGP.

5.1.2 Regression of Wavelet Gaussian process in large dimension

We place ourselves in the framework of simple-fidelity regression. The aim of this section is to present the tools for running the regression. We consider A is the WGP associated to Z. As the outputs of the code are time-series, N_t data points $(\mathbf{x}, t_u)_{u=1}^{N_t}$ are available for all points in the training set $\mathbf{x}$. For simplicity, we assume that $N_t = 2^M$ with $M \in \mathbb{N}$. At the observed data $z(\mathbf{x}, t_u)$ of the GP $Z(\mathbf{x}, t_u)$ we associate $a(\mathbf{x}, j, k)$ with $(j, k) \in I$, associated to the WGP $A(\mathbf{x}, j, k)$ with $(j, k) \in I$ because $\mathrm{card}(I) = N_t$. This means that we have $N_t \times N_x$ learning points $(\mathbf{x}, j, k)$. The expressions of the mean and variance, in section 1.1.2.1, of the prediction limits the number of points that we can use for learning. This is due to the needed inversion of $\mathbf{R}$. We call N_{Krig} the maximum number of points that can be considered for GP regression. Because $N_t \times N_x > N_{\text{Krig}}$ the selection of point is the key to the regression.

One could say that we have the same problem as in the time domain. The difference is that the data are more correlated in the wavelet domain, which allows us to condition on fewer points for the same result. The question is: which points should be chosen to do the regression efficiently?

Selection of points: In order to preserve the advantages of a surrogate model we limit ourselves to a training set $\tilde{\mathcal{D}}$ of N_{Krig} points $(\mathbf{x}, j, k)$. For that it is necessary to eliminate points. We will thus classify the points according to a criterion. For our criterion we have imagined two ideas. The first idea is to look at the values of the coefficients. The objective is to take into account only the points which have the largest $a(\mathbf{x}, j, k)$ observed values. We want to build the set $\tilde{\mathcal{D}} = \{(\mathbf{x}_i, j_i, k_i)\}_{i=1}^{N_{\text{Krig}}}$. The learning set $\tilde{\mathcal{D}}$ must maximize the criterion:

$$\tilde{\mathcal{D}} = \text{argmax}_{\mathcal{D} \text{ st } \text{card}(\mathcal{D})=N_{\text{Krig}}} \sum_{(\mathbf{x},j,k)\in\mathcal{D}} a(\mathbf{x}, j, k)^2. \tag{5.31}$$

This idea is inspired by what is done in wavelet-based image compression. The problem is that we can only take into account points with a large amplitude but not necessarily an amplitude characteristic of the data. We therefore arrive at the second idea which consists in taking into account the variance of each coefficient. Therefore, from the given covariance theorem 5.1.3 it is possible to see which points should have the largest and therefore play the largest role in the WGP. This can be written:

$$\tilde{\mathcal{D}} = \text{argmax}_{\mathcal{D} \text{ st } \text{card}(\mathcal{D})=N_{\text{Krig}}} \sum_{(\mathbf{x},j,k)\in\mathcal{D}} \mathbb{V}\left[A(\mathbf{x}, j, k)\right]. \tag{5.32}$$

The disadvantage of this method is that it does not take into account the data. So we tried to combine the two approaches to have both advantages: using the data but also the points that have the most influence on the WGP. We write the criterion:

$$\tilde{\mathcal{D}} = \text{argmax}_{\mathcal{D} \text{ st } \text{card}(\mathcal{D})=N_{\text{Krig}}} \sum_{(\mathbf{x},j,k)\in\mathcal{D}} \mathbb{V}\left[A(\mathbf{x}, j, k)\right] + \alpha a(\mathbf{x}, j, k)^2, \tag{5.33}$$

with α the coefficient between the two criteria. This criterion seems to be efficient but depends on the parameter α. In the following we take $\alpha = 1$.

We did not investigate point selection further. However, it seems to us that a criterion taking into account the reconstruction error in the time domain would be relevant. Work in this direction is in progress.

Regression: We use the same GP regression method as described in chapter 1. We call this GP regression in wavelet domain or WGP regression. This regression is equivalent to the regression in the time domain. The only difference is the covariance kernel. In order to make predictions it is necessary to go back into the time domain. This is done with the discrete wavelet transform.

The optimization of the hyperparameters is initialized with the correlation length of a trajectory in time domain. We fix a $\mathbf{x}$ in the training set and perform a GP regression on $Z(\mathbf{x}, t)$. By maximum likelihood estimation we obtain the initialization of the hyperparameters. For this we use the GP regression presented in chapter 1. Then, we place ourselves in the wavelet domain, i.e. we perform the wavelet transform on the data. We select the most interesting points $(\mathbf{x}, j, k)$ with the previously calculated correlation length,

i.e. we construct $\tilde{\mathcal{D}}$. This is done using eq. (5.33). This method is effective when the representation of the data in a wavelet domain is sparce.

5.1.3 Illustration on an example

In this illustration we want to emulate a function g of 2 variables t and x. The expression of g is:

$$\mathrm{g}(x,t) = \sin(4\pi(\frac{x}{4}+1)t + \frac{x}{10}). \tag{5.34}$$

The input variable is $x \in [0,1]$. The so-called time variable is $t \in [0,1]$. For this example we have $N_t = 32$, $N_{\mathbf{x}} = 13$ for the learning set. The goal of this illustration is to show the point selection and the regression efficiency. In order to perform the point selection we first construct the WGP covariance matrix associated with the data. This covariance matrix, shown in fig. 5.3, is typical of a data concentrating transformation. It can be seen that only certain coefficients have a significant variance. The idea of point selection is to assume that we are able to estimate all the coefficients from carefully selected points. We have tested the 3 selections procedure described in the previous subsection. We have observed that the first procedure gives unreliable result. The second one gives unstable results. The last one gives results that we describe below.

5.1.3.1 Point selection:

On fig. 5.3, we can see that a significant proportion of the points $(\mathbf{x}, j, k)$ play a small role in the wavelet Gaussian process, i.e. setting many $A(\mathbf{x}, j, k)$ coefficients to zero has little effect on $Z(\mathbf{x}, t)$. This motivates the fact that we neglect to add the least interesting points to the learning set. The criterion 5.33 was chosen. The ratio of selected points $(\mathbf{x}, j, k)$ is 1 in 5 in the training set.

5.1.3.2 Results:

The prediction of the WGP surrogate model is accurate compared to the actual code. As illustrated in fig. 5.4 the results are accurate.

We can see that visually the prediction is accurate for each of the trajectories. However, the $Q^2(t)$ is not completely satisfactory at certain points. This is due to the example having a very low dispersion close to $t = 0.1$. So the prediction has to be very good to get a good $Q^2(t)$. But as t increases prediction quantity improves, and we end up with a value of $Q^2(t)$ close to 1 for t close to 1.

It can also be noted that a decrease in the quality of the prediction is present periodically. This is particularly visible on the $Q^2(t)$. However, these poorly predicted points are the local maxima of the function. We have therefore made the hypothesis that this is due to the sampling of the sine function. The effect fades away as the sine function shifts in phase and there is no longer a cluster effect of maximums.

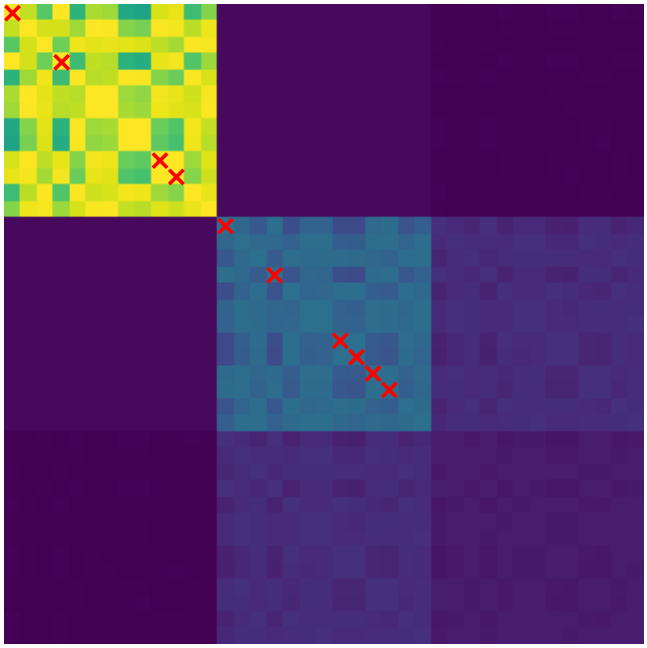


Figure 5.3: Covariance matrix for the learning points $(\mathbf{x}, j, k)$. Each axis encodes the triple $(\mathbf{x}, j, k)$. The grid of the matrix is $\{(\mathbf{x}_i, j = 0, k = 0); (\mathbf{x}_i, j = 1, k = 0); (\mathbf{x}_i, j = 1, k = 1), i = 1, \cdots, N_x\}$. The crossed points are the ones selected for the learning part. We represent the covariance for the values of $\mathbf{x}, j, k$ with the highest amplitudes. As such most of the coefficients have not been presented.

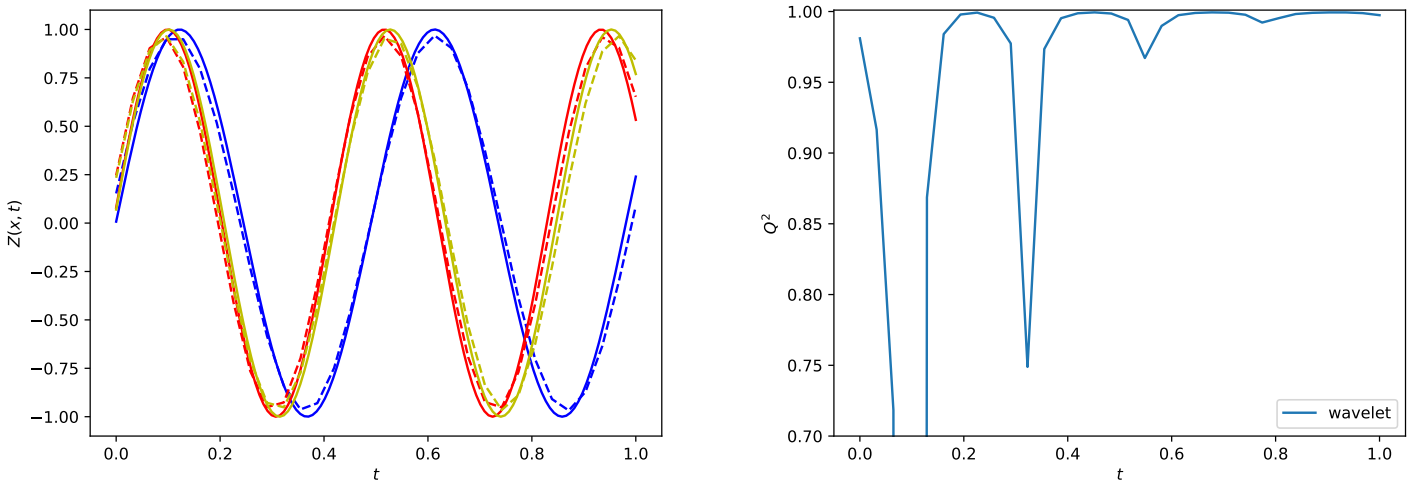


Figure 5.4: Result of regression. On left three examples of trajectories $\mathbf{x} = 0.20, 0.29, 0.97$, solid lines are the function and dashed lines are the prediction. On right, we see the $Q^2(t)$ of the WGP regression. For this example we have $N_t = 32$, $N_{\mathbf{x}} = 13$ and $\mathbf{x} \in [0, 1]$ on a uniform grid in t and $\mathbf{x}$.

This problem is particularly difficult because the sinus has a different frequency for each value of $\mathbf{x}$. This type of configuration is particularly difficult to manage for dimension reduction methods such as SVD. It is with this type of problem in mind that we have devised this method.

5.2 Towards multi-fidelity Wavelet regression

In this section we explain how to modify the WGP framework for multi-fidelity using the AR(1) model.

AR(1) modeling for multi-fidelity is accurate in many contexts, see chapter 3. An attempt has been made in this section to adapt the WGP to AR(1) multi-fidelity. The objective is to build a model with the same properties as the AR(1) model but in the wavelet domain.

Let assume that (Z_H, Z_L) is a Gaussian process of the form:

$$Z_L(\mathbf{x}, t) = A_L(\mathbf{x}, 0, 0)\phi(t) + \sum_{j,k} A_L(\mathbf{x}, j, k)\psi_{j,k}(t),$$

$$Z_H(\mathbf{x}, t) = A_H(\mathbf{x}, 0)\phi(t) + \sum_{j,k} A_H(\mathbf{x}, j, k)\psi_{j,k}(t),$$

then the AR(1) hypothesis is done on (Z_H, Z_L):

$$Z_H(\mathbf{x}, t) = \tilde{\rho}(\mathbf{x})Z_L(\mathbf{x}, t) + \tilde{\delta}(\mathbf{x}, t),$$

with $\tilde{\delta}$ and Z_L two independent GP. The hypothesis is also true for (A_H, A_L):

$$A_H(\mathbf{x}, j, k) = \rho(\mathbf{x})A_L(\mathbf{x}, j, k) + \delta(\mathbf{x}, j, k),$$

with δ and A_L two independent GP. We assume that the covariance of (Z_H, Z_L) is of parametric form. Then the covariance of (A_H, A_L) is of parametric form. This model is proposed in chapter 3. A_H and A_L are correlated. This method is considered to solve the problem of multi-fidelity with large output dimension.

Selection of points The selection of points for multi-fidelity poses an additional problem. If we use the AR(1) framework with the implementation of (Le Gratiet & Garnier, 2013), it is convenient to have nested designs between high and low-fidelity. However, with the criterion proposed in section 5.1.3.1, nesting is not guaranteed. It can be noted that the nesting criterion can be relaxed, see (Zertuche, 2015), but this is done at the cost of an increase in the computational cost. The nesting constraint can also be seen as a constrained optimisation problem, which is the method we envisage for future work.

Multi-fidelity is for the moment an avenue for the use of WGP. Results for solving high-dimensional output multi-fidelity using WGPs are still to come.

5.3 Conclusion

This chapter is the basis for our work on WGPs. The aim is to show the interest of such a framework. It has been shown to be of interest for simple-fidelity regression with high dimensional output. Thanks to an illustration we have seen that WGP can be adapted to particular cases. Focusing on Haar wavelets and Matérn kernels we were able to describe analytically the covariance of WGPs.

Then we presented the property that interests us for the transition to multi-fidelity. It is thanks to this modeling that we hope to be able to implement the WGP in multi-fidelity.

However, there are still a number of difficulties to overcome. The first of these is point selection. We have proposed a simple and efficient method for our illustration but it seems difficult to go further into the implementation. The second difficulty will be the transition to multi-fidelity with several training sets. In particular, the question of nested designs, which does not pose too many problems in applications, will be an issue. There are however tools as proposed in (Le Gratiet, 2013b, ap. B).

Chapter 6

Dimension reduction for multi-fidelity functional output

We have described in section 1.1.3 a simple-fidelity regression method for time-series outputs. Moreover, the AR(1) multi-fidelity model for scalar outputs is described in section 3.1.1. We therefore seek to combine the two methods. In this chapter two approaches are presented. The first approach consists of a modification of the covariance tensorization method. However, this method cannot be used numerically. For this reason, two methods have been introduced based on covariance tensorization method and the AR(1) model. The first method is based on dimension reduction. The second method adds an orthogonal part to the first. As it is independent, GP regression with tensorized covariance can be used. The positive results presented in this chapter are also available in (Kerleguer, 2021).

6.1 Limitation of the tensorized covariance method for multi-fidelity

In this section we use covariance tensorization to perform a multi-fidelity regression. However, when confronted with the limitations of this method, we find that it is not feasible in practice

6.1.1 Regression problem

To carry out a single-fidelity regression for a code whose output is a time-series we can use the method presented in section 1.1.3 or use output dimension reduction. This work is inspired by: (Rougier, 2008) which shows a separation of the covariance matrix, (Conti, Gosling, Oakley, & O'Hagan, 2009) which presents its use in Gaussian processes and (Perrin, 2020) which presents a regression model. (Marque-Pucheu, 2018) gives the structure we will use in this manuscript.

We are interested in the problem presented in section section 1.1.3. We start with some reminders on the covariance function and numerical methods of resolution in the simple-fidelity framework.

Definition 6.1.1. *Let C be the covariance function of the GP regression proposed in section 1.1.3. We assume that there exists two covariance functions C_t and C_x such that:*

$$C\left(t_i, t_j, \mathbf{x}, \mathbf{x}'\right) = C_t\left(t_i, t_j\right) C_x\left(\mathbf{x}, \mathbf{x}'\right), 1 \leq i, j \leq N_t, \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d \tag{6.1}$$

Consequently, the matrices R_t and R_x that determine the covariance matrix of the data are defined as:

$$\left(R_t\right)_{ij} = C_t\left(t_i, t_j\right), \quad \left(R_x\right)_{kl} := C_x\left(\mathbf{x}^{(k)}, \mathbf{x}^{(l)}\right). \tag{6.2}$$

Note: For simple-fidelity GP regression we are able to estimate the matrix R_t. We use (Perrin, 2020, Proposition 3) and get:

$$\hat{R}_t := \frac{1}{N_x}\left(Y_{\text{obs}} - \hat{M}\right) {R_x}^{-1} \left(Y_{\text{obs}} - \hat{M}\right)^T, \tag{6.3}$$

with Y_{obs} the outputs $N_t \times N_x$ matrix of the code and $\hat{M}$ the mean of the outputs for "Simple Kriging" and a linear combination of functions for "Universal Kriging".

As reported in (Perrin, 2020), the matrix R_x is typically ill-conditioned. The chosen regularization is:

$$\widehat{{R_x}^{-1}} = \left(R_x + \varepsilon \mathbb{I}_{Nx}\right)^{-1} \tag{6.4}$$

This regularization seems better for GP regression with tensorized covariance than the following regularization:

$$\widehat{{R_x}^{-1}} = \left({R_x}^T R_x + \varepsilon^2 \mathbb{I}_{Nx}\right)^{-1} {R_x} \tag{6.5}$$

To compute eq. (6.3) the value of ε must be large enough and the regularization proposed in eq. (6.4) is used.

We now seek to extend the method proposed in section 1.1.3 to multi-fidelity. To do this, we will start by setting up a model. Then a resolution method is presented with the estimates presented in the model. For the multi-fidelity regression we assume that:

$$Z_H(\mathbf{x}, t) = \rho Z_L(\mathbf{x}, t) + \delta_H(\mathbf{x}, t), \tag{6.6}$$

with $Z_L(\mathbf{x}, t)$ and $\delta_H(\mathbf{x}, t)$ are independent Gaussian processes. $Z_L(\mathbf{x}, t)$ is of mean μ_L and tensorized covariance C_L. By the model of eq. (6.1) we set:

$$C_L = C_{tL} C_{xL}. \tag{6.7}$$

of the form given in eq. (6.1). $\delta_H(\mathbf{x}, t)$ is of mean μ_δ and tensorized covariance C_δ. Equation (6.1) gives

the expression of the covariance:

$$C_\delta = C_{t\delta} C_{x\delta}. \tag{6.8}$$

In order to estimate the outputs of two codes: high-fidelity (H) and low-fidelity(L). The problem can be express as in property 6.1.2.

Property 6.1.2. *In the multi-fidelity model 6.6 the joint distribution of* $Z_H^{(\mathbf{x})} = \{Z_H(\mathbf{x}, t_u), u = 1, \cdots, N_t\}$, $Z_L^{(D_H)} = \left\{Z_H(\mathbf{x}^{(i)}, t_u), u = 1, \cdots, N_t; i = 1, \cdots, N_H\right\}$ *and* $Z_L^{(D_L)} = \left\{Z_H(\mathbf{x}^{(i)}, t_u), u = 1, \cdots, N_t; i = 1, \cdots, N_L\right\}$, *knowing* μ_L, μ_δ, ρ *and the hyperparameters of* C_L *and* C_δ, *is:*

$$\begin{pmatrix} Z_H^{(\mathbf{x})} \\ Z_H^{(D_H)} \\ Z_L^{(D_L)} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \rho\mu_L(\mathbf{x}) + \mu_\delta(\mathbf{x}) \\ \rho\mu_L^{(D_H)} + \mu_\delta^{(D_H)} \\ \mu_L^{(D_L)} \end{pmatrix}, \begin{pmatrix} \rho^2 C_L^{(\mathbf{x})} + C_\delta^{(\mathbf{x})} & r_x(\mathbf{x})^T \\ r_x(\mathbf{x}) & B \end{pmatrix} \right), \tag{6.9}$$

with the covariances of the outputs $C_L^{(\mathbf{x})} = \{C_L(\mathbf{x}, \mathbf{x}, t_u, t_v), u = 1, \cdots, N_t; v = 1, \cdots, N_t\}$, *same for* $C_H^{(\mathbf{x})}$, $r_x(\mathbf{x}) = \left(\rho^2 C_L^{(D_H, \mathbf{x})} + C_\delta^{(D_H, \mathbf{x})}, \rho C_L^{(D_L, \mathbf{x})}\right)^T$ *and*

$$B = \begin{pmatrix} \rho^2 R_L^{(D_H)} + R_\delta^{(D_H)} & \rho C_L^{(D_H, D_L)} \\ \rho C_L^{(D_L, D_H)} & R_L^{(D_L)} \end{pmatrix},$$

with $R_L^{(D_H)}$ *and* $R_\delta^{(D_H)}$ *are* $N_t N_H \times N_t N_H$ *covariance matrix and* $R_L^{(D_H)}$ *is a* $N_t N_L \times N_t N_L$ *covariance matrix.* $C_L^{(D_L, D_H)}$ *is a* $N_t N_L \times N_t N_H$ *covariance matrix.*

The distribution $Z_H^{(\mathbf{x})} | Z_H^{(D_H)} = z_H^{(D_H)}, Z_L^{(D_L)} = z_L^{(D_L)}, \mu_L, \mu_\delta, \rho, C_L, C_H$ *is Gaussian with mean* $\mu_H^\star(\mathbf{x})$ *and variance* $v_H^\star(\mathbf{x})$*:*

$$\mu_H^\star(\mathbf{x}) = \rho\mu_L(\mathbf{x}) + \mu_\delta(\mathbf{x}) + \mathbf{r}_x^T(\mathbf{x}) B^{-1}(\mathbf{z} - \mathbf{F}), \tag{6.10}$$

with $\mathbf{z} = \left(z_H^{(D_H)}, z_L^{(D_L)}\right)$, $\mathbf{F} = \left(\rho\mu_L^{(D_H)} + \mu_\delta^{(D_H)}, \mu_L^{(D_L)}\right)$, *and*

$$v_H^\star(\mathbf{x}) = \rho^2 C_L^{(\mathbf{x})} + C_\delta^{(\mathbf{x})} - \mathbf{r}_x^T(\mathbf{x}) B^{-1} \mathbf{r}_x(\mathbf{x}). \tag{6.11}$$

In order to perform the GP regression it is necessary to perform an inversion of the $N_t(N_H + N_L) \times N_t(N_H + N_L)$ matrix B. For this purpose a generalized eigenvalue problem is introduced.

Definition 6.1.3. *This definition is extracted form (Harville, 1998, Section 21.14) and (Van Loan & Golub, 1983, Section 8.7).*
Given a symmetric matrix $A \in \mathbb{R}^{n \times n}$ *and a symmetric positive definite* $B \in \mathbb{R}^{n \times n}$, *the generalized eigenvalue problem is finding the not-null vectors* $\mathbf{x}$ *and scalar* λ *that satisfy*

$$A\mathbf{x} = \lambda B\mathbf{x} \tag{6.12}$$

Theorem 6.1.4. *((Van Loan & Golub, 1983, Corollary 8.7.2)) If we assume that A and B satisfy the conditions of definition 6.1.3 then we can write the matrices A and B in the form:*

$$\begin{cases} A=\mathbf{V}\Lambda\mathbf{V}^T \\ B=\mathbf{V}\mathbf{V}^T \end{cases}, \tag{6.13}$$

where $\mathbf{V}=(x_1,\cdots,x_n)$ and $\Lambda=\operatorname{diag}\lambda_1,\cdots,\lambda_n$.

This corollary gives that the matrices A and B can be expressed in the same basis as diagonal matrices. In addition, the diagonal ratio gives the generalized eigenvalues. By a basis change we can therefore reduce ourselves to the case of theorem 6.1.4.

Remark 6.1.5. *However, the matrices whose system we want to solve are ill-conditioned. It is therefore necessary to carry out a regularization. (Ghojogh, Karray, & Crowley, 2019) presents two methods. The solution we used is the one called the rigorous solution.*

The algorithm is used in (Ghojogh et al., 2019, Algorithm 1).

Property 6.1.6. *Using the tensorization definition 6.1.1 we get eq.* (6.14)*.*

$$B=\begin{pmatrix} \rho^2 R_{L,t}\otimes R_{L,x}^{(D_H)}+R_{\delta,t}\otimes R_{\delta,x}^{(D_H)} & \rho R_{L,t}\otimes C_{L,x}^{(D_H,D_L)} \\ \rho R_{L,t}\otimes C_{L,x}^{(D_L,D_H)} & R_{L,t}\otimes R_{L,x}^{(D_L)} \end{pmatrix} \tag{6.14}$$

Since the temporal grid is common to all samples the matrix $R_{L,t}$ is estimated with all samples from the low-fidelity code and eq. (6.3) *and the matrix $R_{H,t}$ from high-fidelity samples using eq.* (6.3) *with $Y_{obs}=\{z_H(\mathbf{x},t_u)-\rho z_L(\mathbf{x},t_u), u=1,\cdots,N_t$ and $\mathbf{x}\in\mathcal{D}_H\}$. We assume that $N_t>N_H$ in order to ensure that $R_{H,t}$ is invertible.*

If we use remark 6.1.5 $R_{L,t}$ and $R_{H,x}$ can be expressed as:

$$\begin{cases} R_{L,t}=\mathbf{V}\Lambda\mathbf{V}^T \\ R_{H,t}=\mathbf{V}\mathbf{V}^T \end{cases}. \tag{6.15}$$

Remark 6.1.7. *Note that the diagonal terms of Λ are non-negative because $R_{L,t}$ is symmetric positive semi-definite. The expression eq.* (6.15) *leads to a simplification in B:*

$$B=(\mathbf{V}\otimes\mathbb{I}_{N_L+N_H})\,C\left(\mathbf{V}^T\otimes\mathbb{I}_{N_L+N_H}\right), \tag{6.16}$$

with

$$C=\begin{pmatrix} \rho^2\Lambda\otimes R_{L,x}^{(D_H)}+\mathbb{I}_{N_t}\otimes R_{H,x}^{(D_H)} & \rho\Lambda\otimes C_{L,x}^{(D_H,D_L)} \\ \rho\Lambda\otimes C_{L,x}^{(D_L,D_H)} & \Lambda\otimes R_{L,x}^{(D_L)} \end{pmatrix}. \tag{6.17}$$

We want to invert the B matrix. To do this, we calculate the inverse of the C matrix using Schur's formula ((Gallier, 2010), (Harville, 1998, Theorem 8.5.11)). This gives us eq. (6.18).

$$C^{-1} = \begin{pmatrix} Q^{-1} & -Q^{-1}U^T \\ -UQ^{-1} & \Lambda^{-1} \otimes {R_{L,x}^{(D_L)}}^{-1} + UQ^{-1}U^T \end{pmatrix} \tag{6.18}$$

with,

$$Q = \mathbb{I}_{N_t} \otimes R_{H,x}^{(D_H)} + \rho^2 \Lambda \otimes \left(R_{L,x}^{(D_H)} + C_{L,x}^{(D_H,D_L)} {R_{L,x}^{(D_L)}}^{-1} C_{L,x}^{(D_L,D_H)} \right) \tag{6.19}$$

and

$$U = \rho \mathbb{I}_{N_t} \otimes \left({R_{L,x}^{(D_L)}}^{-1} C_{L,x}^{(D_L,D_H)} \right) \tag{6.20}$$

For the computation of $\Lambda^{-1} \otimes {R_{L,x}^{(D_L)}}^{-1}$ eq. (6.18) we use the Moore-Penrose inverse for Λ.

Remark 6.1.8. *Instead of computing 6.19 and to invert this formula, a better solution is to consider* $D = R_{H,x}^{(D_H)}$ *and* $E = \rho^2 \left(R_{L,x}^{(D_H)} + C_{L,x}^{(D_H,D_L)} {R_{L,x}^{(D_L)}}^{-1} C_{L,x}^{(D_L,D_H)} \right)$ *and to compute:*

$$ {Q_i}^{-1} = (D + \lambda_i E)^{-1} \tag{6.21}$$

with Q_i *the* i^{th} *diagonal block of the matrix* Q*. This reduces the computation cost of the inversion from* $(N_t N_H)^2$ *to* $N_t N_H^2$*.*

6.1.2 Fault line

The general problem we are trying to solve to be able to invert the B matrix is to find a $\mathbf{V}$ matrix such that we can write the $R_{t,L}$ and $R_{t,H}$ matrices in the form:

$$\begin{cases} R_{t,L} = \mathbf{V} \Lambda \mathbf{V}^T \\ R_{t,H} = \mathbf{V}\mathbf{V}^T \end{cases} . \tag{6.22}$$

In other words: we are trying to express two matrices without links and whose only known property is to be semi-positive definite. However, the two matrices are very ill-conditioned. With a strong regularization we thought we could solve the problem. Thanks to a strong regularization we can write the two matrices in the same reduced basis.

The problem we encountered is that $\mathbf{V}$ then contains large coefficients which in the end explode the outputs of our metamodel. Indeed, very small eigenvalues are multiplied by very large coefficients, which come from $\mathbf{V}$. Thus, it can be said that it is generally not possible to use this method for regression. Therefore, we introduce in section 6.2 other methods that solve a similar problem to this one.

6.2 AR(1) multi-fidelity model with projection

For dimension reduction, as in (Nanty et al., 2017), a basis is chosen. The functional output is expanded onto this basis, the expansion is truncated, and a surrogate model is built for each scalar-valued coefficient of the truncated expansion. In our case, we deal with both multi-fidelity and time-series outputs.

One solution could be to use covariance tensorization in the multi-fidelity framework. This method requires the inversion of large covariance matrices. However, the inversion methods that are possible in a single-fidelity framework become impossible in a multi-fidelity framework because the matrices are then too ill-conditioned to be inverted.

This leads us to introduce new methods. The most naive method consists in starting again from the dimension reduction technique and to carry out the projection of the outputs of the two codes onto the same appropriate basis. It is therefore possible to use the model eq. (3.1). The problem is that it is not possible to define a basis that is optimal for both the high- and low-fidelity codes. A basis estimated from the low-fidelity data is preferred as it is more robust thanks to the larger number of data.

6.2.1 Model

We recall that $(Z_{\mathsf{L}}, Z_{\mathsf{H}})$ is a stochastic process. The temporal grid is $\{t_u\}_{u\in\{1,\dots,N_t\}}$. We assume a hierarchical model. First, we define a model for the basis, determined by an orthogonal matrix $\mathbf{\Gamma}$, in the space $\mathbb{R}^{N_t}$ of the time-series outputs. Second, given the basis we propose a model for the $\mathbf{x}$-dependent coefficients of the decomposition of $(Z_{\mathsf{L}}, Z_{\mathsf{H}})$ onto this basis.

Without prior knowledge the matrix $\mathbf{\Gamma}$ is assumed to have a non-informative distribution in the orthogonal group O_{N_t}, the group of $N_t \times N_t$ orthogonal matrices. This distribution is the Haar measure on the orthogonal group O_{N_t} also called the Uniform Orthogonal Matrix Law. To generate a random matrix from the Haar measure over O_{N_t}, one can first generate a $N_t \times N_t$ matrix with independent and identically distributed coefficients with the reduced normal distribution, then, apply the Gram-Schmidt process onto the matrix. As shown in (Diaconis, 2005), this generator produces a random orthogonal matrix with the uniform orthogonal matrix law. This method completely ignores the available data and is not appropriate in our framework.

Let $\mathbf{\Gamma}$ be an orthogonal $N_t \times N_t$ matrix. The columns of $\mathbf{\Gamma}$, Γ_i, form an orthonormal basis of $\mathbb{R}^{N_t}$. We assume that, given $\mathbf{\Gamma}$, $(Z_{\mathsf{L}}, Z_{\mathsf{H}})$ is a Gaussian process with a covariance function which is such that, for any $\mathbf{x}, \mathbf{x}'$, the matrix $(C(\mathbf{x}, \mathbf{x}', t_u, t_{u'}))_{u,u'=1}^{N_t}$ can be diagonalized on the basis formed by the columns of $\mathbf{\Gamma}$.

The processes Z_{F} can then be expanded as:

$$Z_{\mathsf{L}}(\mathbf{x}, t_u) = \sum_{i=1}^{N_t} A_{i,\mathsf{L}}(\mathbf{x})\Gamma_i(t_u), \tag{6.23}$$

$$Z_{\mathsf{H}}(\mathbf{x}, t_u) = \sum_{i=1}^{N_t} A_{i,\mathsf{H}}(\mathbf{x})\Gamma_i(t_u), \tag{6.24}$$

where $\left(A_{i,\mathsf{L}}(\mathbf{x}), A_{i,\mathsf{H}}(\mathbf{x})\right)$ are Gaussian processes which are independent with respect to i, given $\mathbf{\Gamma}$.

N_{F} observations are available for different values of $\mathbf{x}$ at the fidelity F. Let $\left(\alpha_{i,\mathsf{F}}(\mathbf{x})\right)_{i=1}^{N_t}$ be the $\mathbb{R}^{N_t}$-valued function:

$$\alpha_{i,\mathsf{F}}(\mathbf{x}) = \sum_{u=1}^{N_t} z_{\mathsf{F}}(\mathbf{x}, t_u)\Gamma_i(t_u). \tag{6.25}$$

We denote by α_i^{F} the $1 \times N_F$ row vector $\left(\alpha_{i,\mathsf{F}}(\mathbf{x}^{(j)})\right)_{j=1}^{N_F}$ that contains the available data. The full data set is $\alpha = \left(\alpha^{\mathsf{L}}, \alpha^{\mathsf{H}}\right)$.

Consequently, we can use the model presented in section 3.1.1 given $\mathbf{\Gamma}$. This leads us to the model presented in Equation (6.26). Given $\mathbf{\Gamma}$, $\forall i \in \{1, \ldots, N_t\}$,

$$\begin{cases} A_{i,\mathsf{H}}(\mathbf{x}) &= \rho_{i,\mathsf{L}}(\mathbf{x})\tilde{A}_{i,\mathsf{L}}(\mathbf{x}) + \delta_i(\mathbf{x}) \\ \tilde{A}_{i,\mathsf{L}}(\mathbf{x}) &\perp \delta_i(\mathbf{x}) \\ \rho_{i,\mathsf{L}}(\mathbf{x}) &= g_i^T(\mathbf{x})\beta_{\rho_{\mathsf{L}},i} \end{cases}, \tag{6.26}$$

where:

$$[\delta_i(\mathbf{x})|\mathbf{\Gamma}, \sigma_{i,H}, \beta_{i,H}] \sim \mathcal{GP}\left(f_{i,\mathsf{H}}^T(\mathbf{x})\beta_{i,\mathsf{H}}, \sigma_{i,\mathsf{H}}^2 r_{i,\mathsf{H}}(\mathbf{x}, \mathbf{x}')\right),$$

and $\tilde{A}_{i,\mathsf{L}}(\mathbf{x})$ a Gaussian process conditioned by the values α^{L}. The distribution of $\tilde{A}_{i,\mathsf{L}}(\mathbf{x})$ is the one of $\left[A_{i,\mathsf{L}}(\mathbf{x})|\mathbf{\Gamma}, \mathcal{A}^{\mathsf{L}} = \alpha^{\mathsf{L}}, \beta_{i,\mathsf{L}}, \sigma_{i,\mathsf{L}}\right]$ where:

$$\left[A_{i,\mathsf{L}}(\mathbf{x})|\mathbf{\Gamma}, \sigma_{i,L}, \beta_{i,L}\right] \sim \mathcal{GP}\left(f_{i,\mathsf{L}}^T(\mathbf{x})\beta_{i,\mathsf{L}}, \sigma_{i,\mathsf{L}}^2 r_{i,\mathsf{L}}(\mathbf{x}, \mathbf{x}')\right). \tag{6.27}$$

$g_i(\mathbf{x})$ are vectors of q regression functions, $f_{i,\mathsf{F}}(\mathbf{x})$ are vectors of p_{F} regression functions, $r_{i,\mathsf{F}}(\mathbf{x}, \mathbf{x}')$ are correlation functions, $\beta_{i,\mathsf{F}}$ are p_{F}-dimensional vectors, $\beta_{\rho_{\mathsf{L}},i}$ are q-dimensional vectors and $\sigma_{i,\mathsf{F}}^2$ are positive real numbers. For simplicity the regression functions g_i and $f_{i,\mathsf{F}}$ do not depend on i.

6.2.2 The posterior distribution of the basis

In the model presented in section 6.2.1 we assume that basis follows a uniform distribution. We are looking for the posterior distribution of the basis. For this purpose, two approaches are proposed in this subsection.If a lot of information based on the output of the code is available, then we can use a Dirac distribution concentrated on one orthogonal matrix γ (it is a form of plug-in method). As shown in

section 6.2.2.1, the matrix γ can be estimated from the SVD of the low-fidelity data. If the uncertainty on the basis is to be taken into account because the amount of information is not large enough, then an empirical distribution based on cross-validation procedure can be used as shown in section 6.2.2.2. In order to make the best use of the available information, i.e. the known results of the code, an empirical law can be used (it is a form of full Bayesian method).

6.2.2.1 Dirac distribution

We can choose the distribution of the random matrix $\mathbf{\Gamma}$ as a Dirac distribution concentrated on a well chosen orthogonal matrix γ. This matrix is chosen when the basis is known a priori or if the basis can be efficiently estimated from the observed code outputs. Motivated by the remark below Equation (6.24), the matrix γ can be computed using the singular value decomposition (SVD) of the code outputs.

The general idea is to choose subsets $\tilde{D}_{\mathsf{F}} \subset D_{\mathsf{F}}$ of size $\tilde{N}_{\mathsf{F}}$ and to apply a SVD on the $N_t \times \left(\tilde{N}_{\mathsf{H}} + \tilde{N}_{\mathsf{L}}\right)$ matrix $\tilde{\mathbf{Z}}_{\mathsf{obs}}$ that contains the observed values $(z_{\mathsf{H}}(\mathbf{x}, t_u))_{\substack{u=1,\dots,N_t \\ \mathbf{x}\in\tilde{D}_{\mathsf{H}}}}$ and $(z_{\mathsf{L}}(\mathbf{x}, t_u))_{\substack{u=1,\dots,N_t \\ \mathbf{x}\in\tilde{D}_{\mathsf{L}}}}$. The SVD gives:

$$\tilde{\mathbf{Z}}_{\mathsf{obs}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{V}}^T. \tag{6.28}$$

The choice of γ is $\tilde{\mathbf{U}}$.

The first idea is to mix all available data, high- and low-fidelity: $\tilde{D}_{\mathsf{H}} = D_{\mathsf{H}}$ and $\tilde{D}_{\mathsf{L}} = D_{\mathsf{L}}$. However, we typically have that $N_{\mathsf{L}} \gg N_{\mathsf{H}}$, so the basis is mainly built from the low-fidelity data. In addition, the small differences in the data between high- and low-fidelity code outputs that would be useful to build the basis have negligible impact because they are overwhelmed by the low-fidelity data. This method is not appropriate in our framework.

We have to choose between high- and low-fidelity. High-fidelity has the advantage of being closer to the desired result. However, it is also almost impossible to validate the chosen γ because the high-fidelity data size N_{H} is small. The low-fidelity data set is larger, hence the estimation of γ is more robust. In order to choose γ, we therefore suggest to use the low-fidelity data and to calculate the SVD with $\tilde{D}_{\mathsf{H}} = \emptyset$ and $\tilde{D}_{\mathsf{L}} = D_{\mathsf{L}}$.

6.2.2.2 Cross-Validation Based (CVB) distribution

The downside of the Dirac distribution is that the uncertainties on the basis estimation are not taken into account. A cross-validation based (CVB) method to assess the uncertainty estimation is therefore considered.

The proposed method uses only the low-fidelity data because it is assumed that there are too few high-fidelity data to implement this method, so $\tilde{D}_{\mathsf{H}} = \emptyset$. For the construction of the basis we try to have different sets to evaluate the basis in order to have empirical estimates of the moments of the basis vectors. Let k

be a fixed integer in $\{1,\dots,N_{\mathsf{L}}\}$. Let $I=\{J_1,\dots,J_k\}$ be a random set of k elements in $\{1,\dots,N_{\mathsf{L}}\}$, with uniform distribution over the subsets of k elements in $\{1,\dots,N_{\mathsf{L}}\}$. The empirical distribution for the matrix $\mathbf{\Gamma}$ is defined as follows: for any bounded function $f: O_{N_t}\to\mathbb{R}$,

$$\mathbb{E}\left[f(\mathbf{\Gamma})\right]=\frac{1}{\binom{N_{\mathsf{L}}}{k}}\sum_{\{J_1,\dots,J_k\}\subset\{1,\dots,N_{\mathsf{L}}\}} f(\tilde{\mathbf{U}}_{[J_1,\dots,J_k]}), \tag{6.29}$$

where $\tilde{\mathbf{U}}_{[J_1,\dots,J_k]}$ is the matrix of the left singular vectors of the SVD of $\left(z_{\mathsf{L}}(\mathbf{x}^{(i)},t_u)\right)_{\substack{u\in\{1,\dots,N_t\}\\ i\in\{1,\dots,N_{\mathsf{L}}\}\setminus\{J_1,\dots,J_k\}}}$. This distribution depends on the choice of k.

6.2.3 Predictive mean and variance of Z_{H}

The goal of this section is to calculate the posterior distribution of $Z_{\mathsf{H}}(\mathbf{x},t_u)$. The problem can be split into two parts: the multi-fidelity regression of the basis coefficients knowing $\mathbf{\Gamma}$ and the integration with respect to the distribution of $\mathbf{\Gamma}$. The Dirac and CVB distributions described in section 6.2.2 can be used to define the law of $\mathbf{\Gamma}$.

Multi-fidelity surrogate modeling of the coefficients By applying the model proposed in section 3.1.1 we can therefore deduce the prediction mean and variance. Their expressions are given in appendix B.2.

6.2.3.1 Dirac law of $\mathbf{\Gamma}$

Here we assume that the law of $\mathbf{\Gamma}$ is Dirac at γ. Consequently, the posterior distribution of $Z_{\mathsf{H}}(\mathbf{x},t)$ is Gaussian. In order to characterize the law of $Z_{\mathsf{H}}(\mathbf{x},t_u)$ it is necessary and sufficient to compute its mean and variance.

Mean: The posterior mean is:

$$\mathbb{E}\left[Z_{\mathsf{H}}(\mathbf{x},t_u)|\mathcal{A}=\alpha\right]=\sum_{i=1}^{N_t}\gamma_i(t_u)\mathbb{E}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right], \tag{6.30}$$

where the expectation $\mathbb{E}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right]$ is given by eq. (B.3).

Variance: The posterior variance:

$$\mathbb{V}\left[Z_{\mathsf{H}}(\mathbf{x},t_u)|\mathcal{A}=\alpha\right]=\sum_{i=1}^{N_t}\gamma_i^2(t_u)\mathbb{V}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right], \tag{6.31}$$

where the variance $\mathbb{V}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathcal{A}=\alpha\right]$ is given by eq. (B.4).

6.2.3.2 CVB law of $\mathbf{\Gamma}$

Because the law is different from Dirac the posterior distribution of $Z_{\mathsf{H}}(\mathbf{x}, t)$ is not Gaussian anymore. However, we can characterize the posterior mean and the variance of $Z_{\mathsf{H}}(\mathbf{x}, t_u)$.

We denote $\mathbb{E}_\alpha[\cdot] = \mathbb{E}[\cdot|\mathcal{A} = \alpha]$, $\mathbb{V}_\alpha[\cdot] = \mathbb{V}[\cdot|\mathcal{A} = \alpha]$, $\mathbb{E}_{\mathbf{Z}_{\text{obs}}}[\cdot] = \mathbb{E}[\cdot|\mathcal{Z} = \mathbf{Z}_{\text{obs}}]$ and $\mathbb{V}_{\mathbf{Z}_{\text{obs}}}[\cdot] = \mathbb{V}[\cdot|\mathcal{Z} = \mathbf{Z}_{\text{obs}}]$, with $\mathbf{Z}_{\text{obs}}$ the observed data. $\mathbf{Z}_{\text{obs}} = \left(z_H(\mathbf{x}^{(i)}, t_u), z_L(\mathbf{x}^{(j)}, t_u)\right)_{\substack{u=1,\dots,N_t \\ i=1,\dots,N_H \\ j=1,\dots,N_L}}$

Mean The linearity of the expectation and the law of total expectation give:

$$\mathbb{E}_\alpha[Z_{\mathsf{H}}(\mathbf{x}, t_u)] = \sum_{i=1}^{N_t} \mathbb{E}_\alpha\left[\Gamma_i(t_u)\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right], \tag{6.32}$$

where the expectation $\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]$ is given by Equation (B.3).

Variance The law of total variance gives :

$$\mathbb{V}_\alpha[Z_{\mathsf{H}}(\mathbf{x}, t_u)] = \mathbb{V}_\alpha[\mathbb{E}_\alpha[Z_{\mathsf{H}}(\mathbf{x}, t_u)|\mathbf{\Gamma}]] + \mathbb{E}_\alpha[\mathbb{V}_\alpha[Z_{\mathsf{H}}(\mathbf{x}, t_u)|\mathbf{\Gamma}]] \tag{6.33}$$

By appendix B.3 we get:

$$\begin{array}{c} \mathbb{V}_\alpha[Z_{\mathsf{H}}(\mathbf{x}, t_u)] = \sum_{i=1}^{N_t} \mathbb{V}_\alpha\left[\Gamma_i(t_u)\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ + \sum_{i,j=1, i\neq j}^{N_t} \mathrm{Cov}_\alpha\left(\Gamma_i(t_u)\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right], \Gamma_j(t_u)\mathbb{E}_\alpha\left[A_{j,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right) . \\ + \sum_{i=1}^{N_t} \mathbb{E}_\alpha\left[\Gamma_i^2(t_u)\mathbb{V}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \end{array} \tag{6.34}$$

Equations (6.32) and (6.34) are combinations of expectations of explicit functions of $\mathbf{\Gamma}$. We can compute the result using our knowledge on the law of $\mathbf{\Gamma}$. The expectation of a function of $\mathbf{\Gamma}$ is given by Equation (6.29).

6.2.4 Truncation

There is a problem with the surrogate modeling of the coefficients of the decomposition with indices larger than N_{L}. Indeed, we typically have $N_{\mathsf{L}} < N_t$ so the vectors Γ_i with indices larger than N_{L} of the basis are randomly constructed, which is not suitable for building surrogate models. To solve this problem, it is possible to truncate the sum. Only the first N coefficients, with $N \leq N_{\mathsf{L}}$ are calculated. This would be reasonable if the contributions of the terms $A_{i,\mathsf{H}}(\mathbf{x})\Gamma_i(t_u)$ for $i > N$ were negligible. However, it turns out that these terms are often not collectively negligible and the truncation method does not achieve a good bias-variance trade-off even when optimizing with respect to N (by a cross validation procedure for instance). The high- and low-fidelity outputs do not necessarily have the same forms. Thus, it is possible

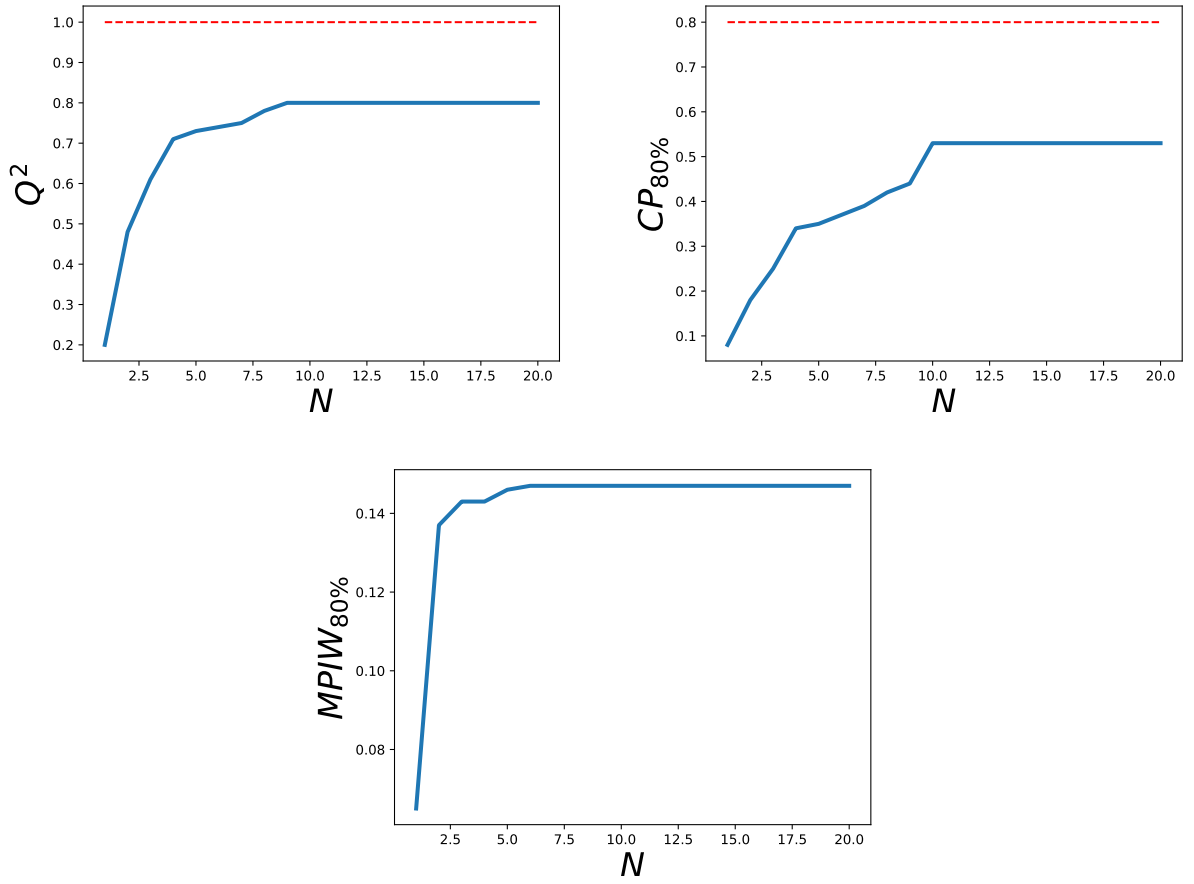


Figure 6.1: Error and prediction interval evaluation in the prediction of the double pendulum, see section 6.4, for different values of N.

that an important part of the high-fidelity code is neglected because it is not taken into account by the sub-space spanned by $\{\Gamma_i\}_{i\leq N}$. We will therefore propose in the next section an original method to tackle this problem.

In order to choose the best value of truncation N we study the Q^2, $\mathrm{CP}_{80\%}$ and $\mathrm{MPIW}_{80\%}$ averaged over all as functions of N. Figure 6.1 shows that the estimation error Q^2 saturates to a value of the order of 0.8 for $N = 10$. It can be seen that the quality of the prediction interval, represented by $\mathrm{CP}_{80\%}$ and $\mathrm{MPIW}_{80\%}$, evolves as the Q^2. This shows that it is not possible to build good predictive models for the components $A_{i,H}(\mathbf{x})\Gamma_i(t_u)$ for $i \geq 10$. We will see in section 6.3 that it is possible to build a surrogate model for $Z_H(\mathbf{x}, t_u)$ with better predictive properties than the truncated model presented here. This can be achieved by modeling and estimating the orthogonal part consisting of the sum of the last components in a collective way.

6.3 AR(1) Multi-fidelity model with projection and orthogonal part

The naive method presented in section 6.2 has many flaws. It leaves part of the output untreated by regression and the variance is underestimated. The major problem with the solution we propose in section 6.2, is that we typically have $N_{\mathsf{L}} < N_t$. Consequently, for $i > N_{\mathsf{L}}$ the vectors Γ_i of the basis do not represent the typical variations of Z_{H}. We should find an appropriate way to predict the law of the projection of Z_{H} on $\operatorname{span}\{\Gamma_i, i > N_{\mathsf{L}}\}$.

One interesting approach is to apply the covariance tensorization method to the orthogonal part. This idea is to address the last terms of the expression collectively through a GP model with tensorized covariance structure. This allows us to split the problem into two parts. The first part is to compute (as presented in the previous section) the first N terms of the expansion of Z_{H} onto the basis by using a co-Kriging approach. The second part is to compute the projection of Z_{H} onto the orthogonal space by using a Kriging approach with tensorized covariance. The choice of the optimal N will be carried out by a K-fold cross-validation method.

6.3.1 Decomposition

We recall that $(Z_{\mathsf{L}}, Z_{\mathsf{H}})$ is a stochastic process. The temporal grid is $\{t_u\}_{u\in\{1,\dots,N_t\}}$. We assume a hierarchical model. First, we define a model for the basis, determined by an orthogonal matrix $\mathbf{\Gamma}$, in the space $\mathbb{R}^{N_t}$ of the time-series outputs. Second, given the basis we propose a model for the $\mathbf{x}$-dependent coefficients of the decomposition of $(Z_{\mathsf{L}}, Z_{\mathsf{H}})$ onto this basis.

Without prior knowledge the matrix $\mathbf{\Gamma}$ is assumed to have a non-informative distribution in the orthogonal group O_{N_t}, the group of $N_t \times N_t$ orthogonal matrices. The proposed method is based on the decomposition of the outputs, as presented in eq. (6.23).

Projection Let $\mathbf{\Gamma}$ be an orthogonal matrix. Let N be an integer, smaller than the time dimension N_t. As discussed in section 6.2.4 the full computation of all N_t surrogate models may not give good results. This leads to the idea of the introduction of the orthogonal subspaces $S_N^{\|}$ and $S_N^{\perp}$, where $S_N^{\|} = \operatorname{span}\{\Gamma_1, \dots, \Gamma_N\}$ and $S_N^{\perp} = \operatorname{span}\{\Gamma_{N+1}, \dots, \Gamma_{N_t}\}$.

Given a basis $\mathbf{\Gamma}$ it is possible to decompose the code outputs. The decomposition over the basis $\mathbf{\Gamma}$ gives us coefficients. We decompose Z_{H} and Z_{L} over the subspace $S_N^{\|}$. The rests are denoted $Z_{\mathsf{H}}^{\perp}$ and $Z_{\mathsf{L}}^{\perp}$. Consequently, we get:

$$Z_{\mathsf{L}}(\mathbf{x}, t_u) = Z_{\mathsf{L}}^{\|}(\mathbf{x}, t_u) + Z_{\mathsf{L}}^{\perp}(\mathbf{x}, t_u) = \sum_{i=1}^{N} A_{i,\mathsf{L}}(\mathbf{x})\Gamma_i(t_u) + Z_{\mathsf{L}}^{\perp}(\mathbf{x}, t_u) \tag{6.35}$$

and

$$Z_{\mathsf{H}}(\mathbf{x}, t_u) = Z_{\mathsf{H}}^{\parallel}(\mathbf{x}, t_u) + Z_{\mathsf{H}}^{\perp}(\mathbf{x}, t_u) = \sum_{i=1}^{N} A_{i,\mathsf{H}}(\mathbf{x})\Gamma_i(t_u) + Z_{\mathsf{H}}^{\perp}(\mathbf{x}, t_u). \tag{6.36}$$

We are able to describe the code outputs with the basis $\mathbf{\Gamma}_N = \{\Gamma_i\}_{i=1,\dots,N}$ of S_N, the coefficients $A_{i,\mathsf{H}}$ and $A_{i,\mathsf{L}}$, and the orthogonal parts $Z_{\mathsf{L}}^{\perp}$ and $Z_{\mathsf{H}}^{\perp}$. We denote by $\alpha_{i,\mathsf{H}}$ and $\alpha_{i,\mathsf{L}}$ the available data sets, the full set is called α. The expression of $\alpha_{i,\mathsf{F}}$ is given by eq. (6.25).

We will use the method presented in section 3.1.1 for all $i \leq N$. Given $\mathbf{\Gamma}$,

$$\left\{ \begin{array}{rcll} A_{i,\mathsf{H}}(\mathbf{x}) & = & \rho_{i,\mathsf{L}}(\mathbf{x})\tilde{A}_{i,\mathsf{L}}(\mathbf{x}) + \delta_i(\mathbf{x}) & \\ \tilde{A}_{i,\mathsf{L}}(\mathbf{x}) & \perp & \delta_i(\mathbf{x}) & , \\ \rho_{i,\mathsf{L}}(\mathbf{x}) & = & g_i^T(\mathbf{x})\beta_{i,\rho_{\mathsf{L}}} & \end{array} \right. \tag{6.37}$$

where:

$$[\delta_i(\mathbf{x})|\mathbf{\Gamma}, \beta_{i,H}, \sigma_{i,H}] \sim \mathcal{GP}\left(f_{i,\mathsf{H}}^T(\mathbf{x})\beta_{i,\mathsf{H}}, \sigma_{i,\mathsf{H}}^2 r_{i,\mathsf{H}}(\mathbf{x}, \mathbf{x}')\right),$$

and $\tilde{A}_{i,\mathsf{L}}(\mathbf{x})$ is a Gaussian process conditioned by α^{L}. Its distribution is the one of $\left[A_{i,\mathsf{L}}(\mathbf{x})|\mathbf{\Gamma}, \mathcal{A}^{\mathsf{L}} = \alpha^{\mathsf{L}}, \beta_{\mathsf{L}}, \sigma_{\mathsf{L}}\right]$ where the law of $\left[A_{i,\mathsf{L}}(\mathbf{x})|\mathbf{\Gamma}, \beta_{i,L}, \sigma_{i,L}\right]$ is of the form eq. (6.27):

$$\left[A_{i,\mathsf{L}}(\mathbf{x})|\mathbf{\Gamma}, \beta_{i,L}, \sigma_{i,L}\right] \sim \mathcal{GP}\left(f_{i,\mathsf{L}}^T(\mathbf{x})\beta_{i,\mathsf{L}}, \sigma_{i,\mathsf{L}}^2 r_{i,\mathsf{L}}(\mathbf{x}, \mathbf{x}')\right),$$

where, g_i are vectors of q regression functions, $f_{i,\mathsf{F}}(\mathbf{x})$ are vectors of p_{F} regression functions, $r_{i,\mathsf{F}}(\mathbf{x}, \mathbf{x}')$ are correlation functions, $\beta_{i,\mathsf{F}}$ are p_{F}-dimensional vectors, $\beta_{i,\rho_{\mathsf{L}}}$ are q-dimensional vectors and $\sigma_{i,\mathsf{F}}^2$ are positive real numbers.

For the orthogonal part projected onto $S_N^{\perp}$ the method is different. The hypothesis is that the projection $Z_{\mathsf{L}}^{\perp}(\mathbf{x}, t_u)$ of $Z_{\mathsf{L}}(\mathbf{x}, t_u)$ has a negligible influence on the projection $Z_{\mathsf{H}}^{\perp}(\mathbf{x}, t_u)$ of $Z_{\mathsf{H}}(\mathbf{x}, t_u)$. Our assumption is that $Z_{\mathsf{H}}^{\perp}(\mathbf{x}, t_u)$ is a Gaussian process with a tensorized covariance. The method we will use on $Z_{\mathsf{H}}^{\perp}(\mathbf{x}, t_u)$ is described in section 1.1.3.

Note that the value $N = 0$ corresponds to full single-fidelity, in this case we use only GP regression with covariance tensorization method as in section 1.1.3. For $N = N_{\mathsf{L}}$ the dimension reduction is minimal and co-Kriging is applied to all pairs $\left(A_{i,\mathsf{L}}, A_{i,\mathsf{H}}\right)$ for $i \leq N_{\mathsf{L}}$. We will see that the optimal N is in fact positive but smaller than N_{L}.

6.3.2 Predictive mean and covariance

In this section we first make a quick reminder of the methods presented in the state of the art part. Moreover, with different assumptions about the law of $\mathbf{\Gamma}$ we present the regression using the model and the data.

Multi-fidelity of coefficients As in section 6.2.3 we compute the N multi-fidelity models of the first N coefficients of the expansion of the code output given $\mathbf{\Gamma}$. If we apply the method proposed in section 3.1.1 we can therefore deduce the prediction mean and variance, as in section 6.2.3.

Tensorized covariance regression The orthogonal part of the regression is computed using the method presented in section 1.1.3. The adaptation is that the regression must be carried out in subspace $S_N^{\perp}$ given $\mathbf{\Gamma}$, and the data set has the form:

$$\mathbf{Z}_{\mathsf{obs}}^{\perp} = \mathbf{Z}_{\mathsf{obs}} - \left(\sum_{i=1}^{N} \alpha_{i,\mathsf{H}}(\mathbf{x}) \Gamma_i(t_u) \right)_{\substack{u=1,\cdots,N_t \\ \mathbf{x} \in D_{\mathsf{H}}}}, \tag{6.38}$$

where $\alpha_{i,\mathsf{H}}$ is given by Equation (6.25) and $\mathbf{Z}_{\mathsf{obs}} = \left\{ z_H(\mathbf{x}^{(i)}, t_u) \right\}_{i=1,\cdots,N_H}^{u=1,\cdots,N_t}$, with $\mathbf{x}^{(i)} \in \mathcal{D}_H$.

This does not have any consequence on the $\mathbf{x}$ part but only on the t part. Contrarily to $Z_{\mathsf{H}}^{\parallel}(\mathbf{x}, t_u)$ only one surrogate model is needed for $Z_{\mathsf{H}}^{\perp}(\mathbf{x}, t_u)$. The detail of how we can deal with $Z_{\mathsf{H}}^{\perp}(\mathbf{x}, t_u)$ and $Z_{\mathsf{H}}^{\parallel}(\mathbf{x}, t_u)$ is explained in appendix B.1.

$$\left(Z_H^{\perp}(\mathbf{x}, t_u) \right)_{u=1,\ldots,N_t} | R_t, C_x, \mathbf{Z}_{\mathsf{obs}}^{\perp} \sim \mathcal{GP}(\mu_\star(\mathbf{x}), R_\star(\mathbf{x}, \mathbf{x}') R_t). \tag{6.39}$$

with the N_t-dimensional posterior mean:

$$\mu_\star(\mathbf{x}) = \mathbf{Z}_{\mathsf{obs}}^{\perp} R_x^{-1} r_x(\mathbf{x}) + B_\star u(\mathbf{x}) \tag{6.40}$$

where $r_x(\mathbf{x})$ is the N_x-dimensional vector $\left(C_x(\mathbf{x}, \mathbf{x}^{(j)}) \right)_{j=1,\ldots,N_x}$. R_t is estimated by maximum likelihood in appendix B.1. The posterior covariance function $R_\star(\mathbf{x}, \mathbf{x}')$ is :

$$R_\star(\mathbf{x}, \mathbf{x}') = c_\star(\mathbf{x}, \mathbf{x}') \left(1 + v_\star(\mathbf{x}, \mathbf{x}') \right). \tag{6.41}$$

The functions that are used in the regression are:

$$\begin{cases} u(\mathbf{x}) = f(\mathbf{x}) - F^T R_x^{-1} r_x(\mathbf{x}) \\ c_\star(\mathbf{x}, \mathbf{x}') = C_x(\mathbf{x}, \mathbf{x}') - r_x(\mathbf{x})^T R_x^{-1} r_x(\mathbf{x}') \\ v_\star(\mathbf{x}, \mathbf{x}') = u(\mathbf{x})^T (F^T R_x^{-1} F)^{-1} u(\mathbf{x}') c_\star^{-1}(\mathbf{x}, \mathbf{x}') \end{cases}, \tag{6.42}$$

and

$$B_\star = \mathbf{Z}_{\mathsf{obs}}^{\perp} R_x^{-1} F (F^T R_x^{-1} F)^{-1}. \tag{6.43}$$

The a priori $\mathbb{R}^{N_t}$-valued mean function is assumed to be of the form: $\mu(\mathbf{x}) = B f(\mathbf{x})$, where $f(\mathbf{x})$ is a given $\mathbb{R}^M$-valued function and $B \in \mathcal{M}_{N_t \times M}(\mathbb{R})$ is estimated at eq. (6.43). We define by F the $N_x \times M$ matrix

$[f^T(\mathbf{x}^{(i)})]_{i=1,\dots,N_x}$. Hereafter we note: $\mathbb{E}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}},\mathbf{\Gamma}\right]=\mu_\star(\mathbf{x},t_u)$ and $\mathbb{V}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}},\mathbf{\Gamma}\right]=R_\star(\mathbf{x},\mathbf{x})R_t(t_u,t_u)$.

6.3.2.1 Dirac law of $\mathbf{\Gamma}$

Here we assume that $\mathbf{\Gamma}$ is known and its distribution is Dirac at γ. Consequently, as in section 3.1.1, $Z_{\mathsf{H}}(\mathbf{x},t)$ is a Gaussian process by linear combination of independent Gaussian processes. Its posterior distribution is completely determined if we can evaluate its mean and covariance.

Mean The $\Gamma_i(t_u)$' s are constant and equal to $\gamma_i(t_u)$. Consequently:

$$\mathbb{E}_\alpha\Big[Z_{\mathsf{H}}^{\parallel}(\mathbf{x},t_u)|N\Big]=\sum_{i=1}^{N}\mathbb{E}_\alpha\big[A_{i,\mathsf{H}}(\mathbf{x})\big]\gamma_i(t_u), \tag{6.44}$$

and

$$\mathbb{E}_{\mathbf{Z}_{\text{obs}}}[Z_{\mathsf{H}}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}}]=\textstyle\sum_{i=1}^{N}\mathbb{E}_\alpha\big[A_{i,\mathsf{H}}(\mathbf{x})\big]\Gamma_i(t_u)+\mathbb{E}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}}\right]\ , \tag{6.45}$$

where $\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})\right]$ is given by (B.3) and $\mathbb{E}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}}\right]$ by (1.38).

Variance The formula of the variance is:

$$\mathbb{V}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})\Gamma_i(t)\right]=\mathbb{V}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})\right]\gamma_i(t)^2. \tag{6.46}$$

The uncorrelation of the coefficients $A_{i,\mathsf{F}}(\mathbf{x})$ gives $\mathrm{Cov}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x}),A_{j,\mathsf{H}}(\mathbf{x})\right]=0$, for $i\neq j$ and $\mathrm{Cov}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})\gamma_i(t_u),Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)\right]=0$. The expression of the variance becomes simple:

$$\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}}\right]=\textstyle\sum_{i=1}^{N}\mathbb{V}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})\right]\gamma_i(t_u)^2+\mathbb{V}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}}\right], \tag{6.47}$$

where $\mathbb{V}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})\right]$ is given by (B.4) and $\mathbb{V}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}}\right]$ is given by (1.37) on the orthogonal part.

6.3.2.2 CVB law of $\mathbf{\Gamma}$

The posterior distribution of $Z_{\mathsf{H}}(\mathbf{x},t)$ is not Gaussian anymore. However to predict the output of the high-fidelity code and to quantify the prediction uncertainty we are able to compute the posterior mean and variance of $Z_{\mathsf{H}}(\mathbf{x},t)$.

Mean We can decompose the process into two parts:

$$Z_{\mathsf{H}}(\mathbf{x},t_u)=Z_{\mathsf{H}}^{\parallel}(\mathbf{x},t_u)+Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u). \tag{6.48}$$

The linearity of the expectation gives us:

$$\mathbb{E}_{\mathbf{Z}_{\text{obs}}}[Z_{\mathsf{H}}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}}] = \sum_{i=1}^{N} \mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[A_{i,\mathsf{H}}(\mathbf{x})\Gamma_i(t_u)\right] + \mathbb{E}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t)|N,\ell_{\mathbf{x}}\right]. \tag{6.49}$$

The theorem of total expectation gives us:

$$\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}^{\parallel}(\mathbf{x},t_u)|N\right] = \sum_{i=1}^{N} \mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right], \tag{6.50}$$

and therefore,

$$\begin{aligned}\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}}\right] = &\sum_{i=1}^{N} \mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ &+ \mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{E}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|N,\ell_{\mathbf{x}},\mathbf{\Gamma}\right]|N,\ell_{\mathbf{x}}\right].\end{aligned} \tag{6.51}$$

where $\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]$ is given by Equation (B.3) and $\mathbb{V}_{\alpha}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x})|\mathbf{\Gamma}\right]$ is given by Equation (1.38). Equation (6.51) is a combination of expectations of explicit functions of $\mathbf{\Gamma}$, which can be computed by Equation (6.29).

Variance The theorem of the total variance gives us:

$$\begin{aligned}\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|N,\ell_{\mathbf{x}}\right] = &\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ &+\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right]\end{aligned}. \tag{6.52}$$

By appendix B.4 we get:

$$\begin{aligned}\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|N,\ell_{\mathbf{x}}\right] = &\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{E}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ &+\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{V}_{\alpha}\left[Z_{\mathsf{H}}^{\perp}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ &+\textstyle\sum_{i=1}^{N}\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ &+\textstyle\sum_{i=1}^{N}\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)^2\mathbb{V}_{\alpha}\left[A_{i,\mathsf{H}}\left(\mathbf{x}\right)|\mathbf{\Gamma}\right]\right] \\ &+\textstyle\sum_{i,j=1;i\neq j}^{N}\mathsf{Cov}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right],\Gamma_j(t_u)\mathbb{E}_{\alpha}\left[A_{j,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ &+2\textstyle\sum_{i=1}^{N}\mathsf{Cov}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right],\mathbb{E}_{\mathbf{Z}_{\text{obs}}^{\perp}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right]\end{aligned}. \tag{6.53}$$

where $\mathbb{V}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]$ is given by eq. (B.4) and $\mathbb{V}_{\alpha}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x})|\mathbf{\Gamma}\right]$ is given by eq. (1.39). Equation (6.53) is a combination of expectations and variances of explicit functions of $\mathbf{\Gamma}$, which can be computed by eq. (6.29).

6.3.3 Effective dimension

For the formulas in section 6.3.2 to be valid, N must be fixed. We may choose N by knowledge on the physical system or on the code, but it is impossible in most cases due to the high/low-fidelity differences. The best solution is generally to determine N by a K-fold cross validation procedure.

The criterium that we choose to maximize is:

$$Q_N^2(t_u) = 1 - \frac{\sum_{k=1}^{N_\mathsf{H}} \left(z_\mathsf{H}(\mathbf{x}^{(k)}, t_u) - \mathbb{E}\left[Z_\mathsf{H}(\mathbf{x}^{(k)}, t_u) | \mathbf{\Gamma}, N, \ell_\mathbf{x}, \mathbf{Z}_\mathsf{obs}^{(-k)} \right] \right)^2}{N_\mathsf{H} \mathbb{V}\left[z_\mathsf{H}(D_\mathsf{H}, t_u) \right]}, \tag{6.54}$$

where $\mathbb{V}\left[z_\mathsf{H}(D_\mathsf{H}, t_u) \right]$ is the empirical variance of the observed values:

$$\mathbb{V}\left[z_\mathsf{H}(D_\mathsf{H}, t_u) \right] = \frac{1}{N_\mathsf{H}} \sum_{k=1}^{N_\mathsf{H}} z_\mathsf{H}(\mathbf{x}^{(k)}, t_u)^2 - \Big(\frac{1}{N_\mathsf{H}} \sum_{k=1}^{N_\mathsf{H}} z_\mathsf{H}(\mathbf{x}^{(k)}, t_u) \Big)^2.$$

$x^{(k)}$ denotes the points of D_H and $Z_\mathsf{obs}^{(-k)}$ denotes the observations at the points of D_L and the points $\{(x^{(j)}, t_u), j = 1, \cdots, N_\mathsf{H}; u = 1, \cdots, N_t\}$.

The procedure we propose starts with the dimension 0. For the case $N = 0$ the surrogate model depends only on high-fidelity regression.

- We compute the surrogate model for all $N = 0, \ldots, N_\mathsf{L}$

- We calculate the mean in t_u of $Q_N^2(t_u)$:

$$\hat{Q}_N^2 = \frac{1}{N_t} \sum_{u=1}^{N_t} Q_N^2(t_u)$$

We compare the $\hat{Q}_N^2$ values and the value N with the largest $\hat{Q}_N^2$ is chosen. In order to evaluate the surrogate model in the next section, we compute $\hat{Q}^2 = \max_N \hat{Q}_N^2$.

Algorithm We present the algorithm of the full regression in Algorithm 1. This algorithm presents the method we use to choose the dimension N and how we compute the error indicator. Note that the optimization of the hyperparameters is done for each N. It is however not very expensive because we only add the model A_N and $Z_\mathsf{H}^\perp$. The other coefficients are taken from previous optimizations thanks to

the decorrelation of the A_i coefficients.

Result: optimal value of N, parameters surrogate model

Compute the basis $\mathbf{\Gamma}$;

for $\overline{N}$ *in* $1 \cdots N_H$ **do**

Projection of codes on $S_{\overline{N}}^{\perp}$;

The coefficients $\alpha_{i,\mathsf{H}}$ and $\alpha_{i,\mathsf{H}}$ are evaluated ;

Multi-fidelity Kriging on α coefficients ;

Regression of $Z_{\mathsf{H}}^{\perp}$ with tensorized covariance as in section 1.1.3;

Combine the two regressions;

Computation of the leave-one-out $\hat{Q}_{\overline{N}}^2$ with optimisation of the hyperparameters for $N = \overline{N}$;

end

The best value of $\hat{Q}_{\overline{N}}^2$ gives N;

Parameters of surrogate model are optimized using the chosen value of N;

Algorithm 1: Realisation of the regression

6.4 Illustration: double pendulum simulator

6.4.1 Presentation of the double pendulum

The system can be seen as a dual-oscillator cluster. The first oscillator is a spring-mass system whose axis is perpendicular to the gravitational axis. The parameters of this system are the mass of the system M_S and the spring stiffness k. The initial position of the mass is denoted y_0, its initial velocity is 0. The second oscillator is a pendulum. A schematic representation of the system is presented in Figure 6.2. The parameters are the mass m and the length of the pendulum ℓ, which are fixed. The initial value of the angle is θ_0 and its derivative is $\dot{\theta_0}$. By Newton's law of motion, the dynamics is governed by a system of two coupled ordinary differential equations (ODEs). However, we do not have a closed form expression that gives the solution of the system. This forces us to use computer codes. The output signal is the position of the mass m at time $t \in \{t_1, \ldots, t_{N_t}\}$ with $N_t = 101$. The input vector is $\mathbf{x} = \left\{M_S, k, y_0, \theta_0, \dot{\theta_0}\right\}$. The input variables are assumed to be independent and identically distributed with uniform distributions as described in Table 6.1.

The two different code levels We propose two codes. The high-fidelity code numerically solves the coupled system of ODEs by an Euler's derivation of the position y and the angle θ for each t_u. This gives functions $\theta(t_u)$ and $y(t_u)$. The low-fidelity code assumes that the angle of the θ pendulum is small so that the linearization of $\sin(\theta)$ makes it possible to get a simpler form of the expression of the two coupled ODEs and a faster resolution.

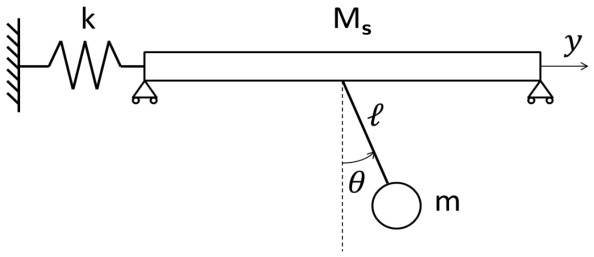


Figure 6.2: The double pendulum system with its parameters.

Table 6.1: Distributions of the input variables.

M_S	k	θ_0	$\dot{\theta_0}$	y_0
$\mathcal{U}(10, 12)$	$\mathcal{U}(1; 1.4)$	$\mathcal{U}(\frac{\pi}{4}; \frac{\pi}{3})$	$\mathcal{U}(0; \frac{1}{10})$	$\mathcal{U}(0; 0.2)$

Code Analysis A sensitivity analysis is carried out for information purposes, but it is not used in the forthcoming surrogate modeling. The sensitivity analysis makes it possible to determine the effective dimension of our problem. We compare outputs of the high- and low-fidelity codes and the associated Sobol indices on Figure 6.3. We estimate Sobol indices by the method described in (Saltelli et al., 2010) and implemented in the R library (Iooss et al., 2020) by using a Monte Carlo sample of size 10^5 for each code. No surrogate model was used to estimate the indices in Figure 6.3. The main result is that the two codes depend on the same input variables. The four most important input variables are y_0, k, M and θ_0.

The experimental designs used to compare the methods are presented in (Le Gratiet, 2013b). They are constructed from two independent maximim LHS designs with $N_{\mathsf{H}} = 10$ and $N_{\mathsf{L}} = 100$ points. The low-fidelity design is then modified so that the designs are nested. Only the points of the low-fidelity design closest to the points of the high-fidelity design are moved. To generate these designs the R packages (Dupuy, Helbert, & Franco, 2015; Le Gratiet, 2012) are used. A random uniform nested design can also be used, but we choose a more effective design for GP regression. The test design is composed of 4000 points randomly chosen in the hypercube determined by the supports of the uniform distributions described in table 6.1.

In this section, we want to demonstrate the interest of the method presented in section 6.3. For this we will compare several methods:

- the multi-fidelity method presented in section 6.2 with a Dirac distribution of $\mathbf{\Gamma}$, called the SVD method.

- the multi-fidelity method that uses GP regression of the orthogonal part with covariance-tensorization and the distribution of $\mathbf{\Gamma}$ is Dirac at γ the matrix of the SVD of the observed low-fidelity code outputs. Its prediction is computed as is section 6.3 and called Dirac method.

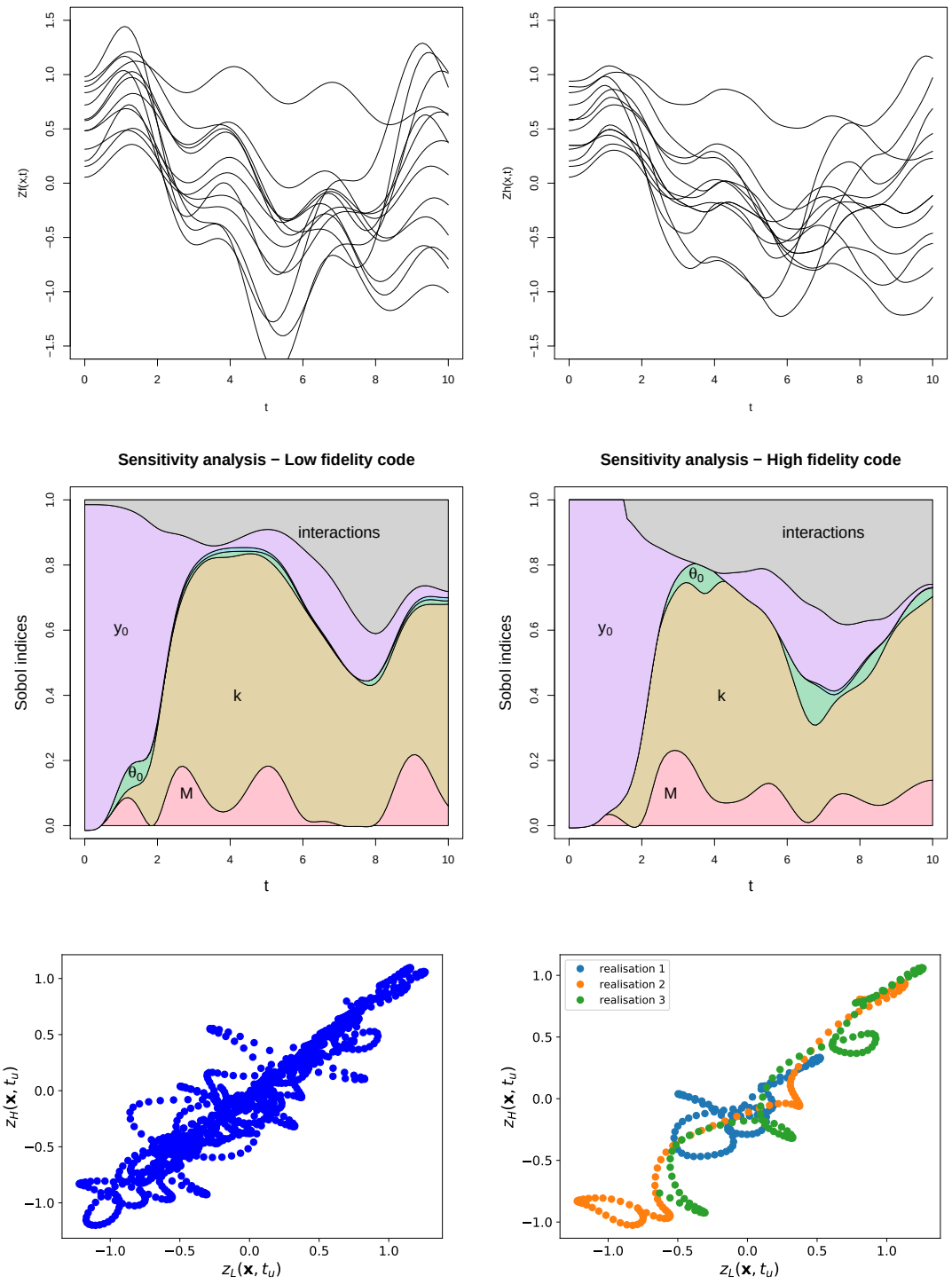


Figure 6.3: Comparison between low-fidelity (top left) and high-fidelity (top right) code outputs. Sobol indices for high- and low-fidelity codes (center left: low-fidelity, center right: high-fidelity). For each time t in the time grid, we report the first-order Sobol indices and "interactions" stands for the sum of the Sobol indices of order larger than 2. Finally, the bottom plots represent the interactions between codes (plots of $(z_L(x,t_u), z_H(x,t_u))_{u=1}^{N_t}$ for different x). The bottom left graph is for 10 and the bottom right graph is for 3 values of x.

- the multi-fidelity method presented in section 6.3 with the CVB distribution, called CVB method.
- the neural network (NN) method presented in (Meng & Karniadakis, 2020). We extend this method for time-series outputs by considering N_t-dimensional outputs for the low- and high-fidelity neural networks and by removing the physical inspired NN part. We used the parameters proposed in the article, i.e. 2 hidden layers for each network with 20 neurons per layer. We also tested the NN method up to 100 neurons per layer, but the best results were obtained with 20 neurons approximately. This method is presented in section 3.2.2.

The method we would like to highlight is the CVB method.

CVB basis The law of $\mathbf{\Gamma}$ needs to be determined in order to compute or estimate the moments (6.32),(6.34), (6.51), and (6.53). The distribution of $\mathbf{\Gamma}$ is the CVB distribution described in section 6.2.2.2. As shown by eq. (6.29) it depends on the size k of the random subset I. Here we choose $k = 4$. Because it is too expensive to compute the sum over all $\binom{N_\mathsf{L}}{k}$ different subsets $\{j_1, \dots, j_k\}$, we estimate the expectation (6.29) by an empirical average over $n = 64$ realizations I_j of the random subset I:

$$\mathbb{E}\left[f(\mathbf{\Gamma})\right] \simeq \frac{1}{n}\sum_{j=1}^{n} f(\tilde{\mathbf{U}}_{I_j}), \tag{6.55}$$

where $\tilde{\mathbf{U}}_{I_j}$ is the matrix of the left singular vectors of the SVD of $\left(z_\mathsf{L}(\mathbf{x}^{(i)}, t_u)\right)_{\substack{u\in\{1,\dots,N_t\} \\ i\in\{1,\dots,N_\mathsf{L}\}\setminus I_j}}$. We have checked that the stability with respect to k is conserved if $1 < k < N_\mathsf{L} - N_\mathsf{H}$ and the stability with regard to n is valid if $n > \max(k, 50)$. We have tested the construction of the CVB basis for all k values in this range and found that changes in k do not influence the basis significantly.

The computational cost of calculating the basis is very important in particular because it is impossible for us to calculate it for all subsets. A method to compute the basis with only a cost of $O(N_t^2)$ is given in (Mertens, Fearn, & Thompson, 1995) whereas we compute it with $O(N_\mathsf{L}^2 N_t)$ by our method. The gain is however very small especially if $N_\mathsf{L} < N_t$ which is our case. We have therefore not implemented this method in the results presented in this manuscript.

Prediction of the orthogonal part A simple model for the a priori mean function is chosen $M = 1$ and $f(\mathbf{x}) = 1$. Consequently, $F^T R_x^{-1} F = \sum_{i,j} \left\{R_x^{-1}\right\}_{i,j}$.

Multi-fidelity regression of the coefficients Our implementation of the multi-fidelity regression is based on (Le Gratiet, 2012). We use an informative prior for the regression of the coefficients. For more information we refer to (Le Gratiet, 2013b, sec. 3.4.4). In this example the size of the priors are $q = p_\mathsf{L} = p_\mathsf{H} = 1$. Considering the relation between the two codes we choose $b^\rho = 1$. The trend is supposed to be null

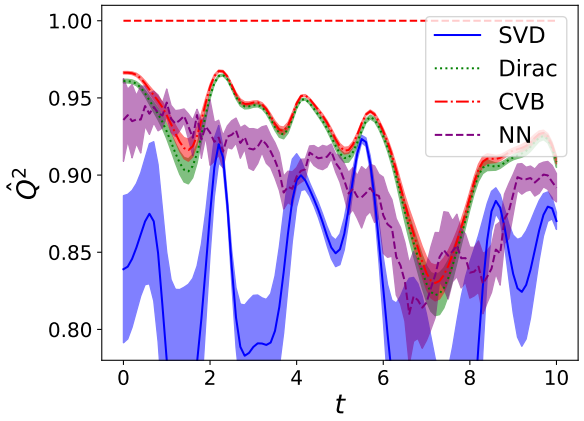


Figure 6.4: Comparison between the methods in terms of time-dependent $\hat{Q}^2$. Averages over 40 random experimental designs are computed. The colored fields represent the confidence intervals determined by ± 1.96 empirical standard deviation. Here $N_{\mathsf{H}} = 10$, $N_{\mathsf{L}} = 100$ and $N_t = 101$.

consequently, $b_{\mathsf{H}}^{\beta} = b_{\mathsf{L}} = 0$. The variances are $\sigma_{\mathsf{L}} = 0.5$ and $\sigma_{\mathsf{H}} = 0.5$ with $V_{\mathsf{H}}^{\beta} = 2$ and $V_{\mathsf{L}} = 2$. The parameters for the inverse Gamma distribution $m_{\mathsf{L}} = m_{\mathsf{H}} = 0.2$ and $\varsigma_{\mathsf{L}} = \varsigma_{\mathsf{H}} = 1.5$. We have checked the robustness of the results with respect to the hyper-parameters of the prior distributions. Alternatively, the article (Ma, 2020) presents non-informative priors for the autoregressive co-Kriging.

Prediction In order to estimate the errors of the surrogate models, we calculate their $\hat{Q}^2$'s and report them in fig. 6.4. To compute $\hat{Q}^2$ for a model, we calculate the difference between the validation set of size 4000 and the predictions of the model. We have averaged the estimates of the $\hat{Q}^2$ over 40 different experimental designs.

The SVD method gives a very interesting result because the $\hat{Q}^2$ is almost always higher than 0.8. However, in fig. 6.5 we can see that it does not capture the form of the times series. The $\hat{Q}^2$ of the Dirac and CVB methods are larger than the ones of the other methods. The error is also less variable as a function of t. And the variance is much lower for both methods. However, even if there is a difference between the Dirac and CVB methods, it is not possible to say that the CVB method is better in this application. The difference between the Dirac and CVB methods is small, in our example.

The variance of the prediction is very important for the quantification of prediction uncertainty. All formulas are given in the previous sections and we illustrate the results in fig. 6.5. We can see that the variance of the SVD method is not accurate and overestimates the quality of the prediction. This method is not acceptable for prediction. The Dirac method and the CVB method have almost the same variance. If we compare to the variance of the SVD method, it means that most of the uncertainty relates to the orthogonal part. This leads to the conclusion that this part is important in the regression. In section 6.4.1 we compare the variance explained by the dimension reduction and by the orthogonal part. It can therefore be seen that the variance is of the same order of magnitude for both parts. This justifies

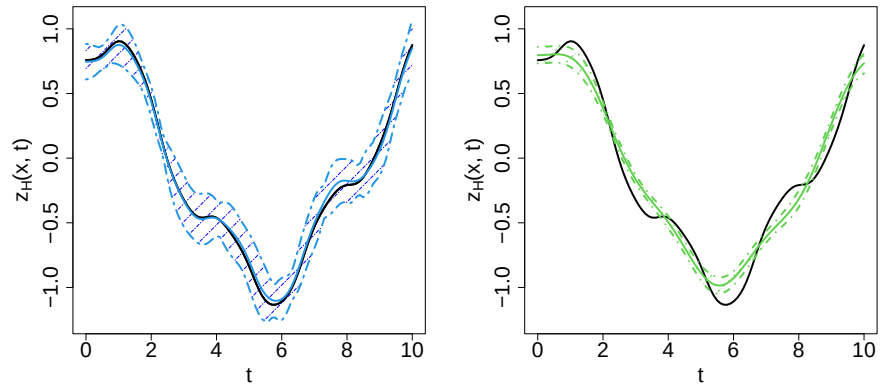


Figure 6.5: Comparison between the predictions of the CVB method (left) and the SVD method (right). The black solid line is the exact high-fidelity time-series, the colored solid line is the prediction mean and the dashed lines are the prediction intervals. In this example the value of N obtained by cross validation is 8.

keeping a quantification of uncertainty for both parts together.

6.5 Conclusion

In this chapter we propose to adapt the covariance tensorization method for GP regression to multi-fidelity. The first approach we have devised is not usable because of the numerical conditioning problem of the empirical covariance matrices. Then we proposed a method using dimension reduction and independent GP regression. This method defines a subspace from SVD on the low-fidelity data and projects the data set onto it. The data low- and high-fidelity are processed in this space using the AR(1) model. For each coefficient the model is independent of the others. Then for the high-fidelity orthogonal space we perform a GP regression with tensorization of the single-fidelity covariance. The balance between the number of items considered in the dimension reduction and the single-fidelity regression is achieved by means of cross-validation. In order to know the output uncertainty perfectly, a method was constructed to give the uncertainty of the SVD decomposition.

The regression results of this method are very good. In particular, they exceed the state of the art and allow uncertainty to be taken into account. We thus have an efficient method for solving multi-fidelity regression problems with time-series output in the context of uncertainty quantification. This method could be improved by using objective priors both for the orthogonal part and in the AR(1) models. It is also possible to consider extending this method to a larger input dimension by using dimension reduction methods.

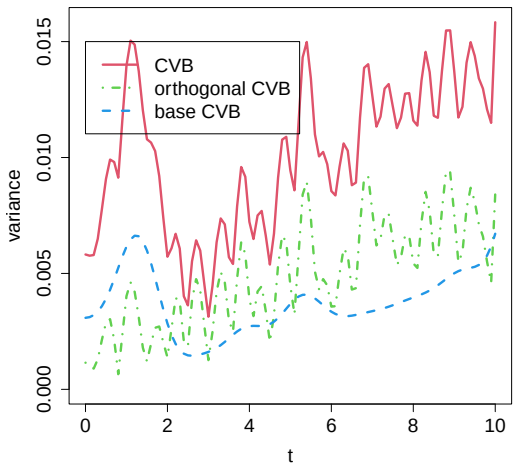


Figure 6.6: Estimation of the different time-dependent prediction variance terms for the empirical method.

Conclusion

The general framework of the thesis is the construction of a surrogate model of the response of a complex and expensive computer code. The objective is to approximate the input/output relationship of the computational code by a function, fast to evaluate from a limited number of executions of the code. Such an approximation is called a surrogate model. From this surrogate model, uncertainty quantification can be performed. For practical applications, the use of a surrogate model is often necessary because complex computer codes are usually very time-consuming, and the analysis (optimisation, sensitivity analysis, ...) requires many calls to the code. Strategies for building a surrogate model depend on the original objective. For optimisation problems, the observations of the computational code should be concentrated, where the expectation of improvement is high. For prediction problems, the observations will generally be space filling over the whole input parameter space. An important point is that a surrogate model is not used in extrapolation, i.e. it will only be valid in the domain determined by the set of experimental designs.

In this manuscript, we are interested in simulators that also have less precise versions but which are also less costly in terms of computing time. These less accurate versions allow the same phenomenon to be simulated but may, for example, use less complex modeling or less accurate resolution methods. The aim is to improve the approximation of the output of a computer code by also using these low-fidelity versions. In this manuscript we propose three methods.

The first method combines a Gaussian process with a Bayesian neural network, see chapter 4. We mainly focus on the modeling of the relationship between the different code levels and on the transfer of information from the low-fidelity emulator output to the high-fidelity emulator input. To achieve an efficient transfer, we have proposed different methods, the most efficient being the Gauss-Hermite quadrature. This model is called GP-BNN and has many advantages over the standard models in the literature. GP-BNNs are very efficient in predicting scalar outputs, even when the interactions between codes are complex. The quantification of prediction uncertainty is particularly effective, even for small data sets. The main drawback of GP-BNN is that we have not been able to make them work in large output dimensions.

The second approach is adapted to the problem of large output dimensions. The method proposed in chapter 5 is based on the classical first-order autoregressive model of multi-fidelity Gaussian process regression. We start by assuming that the emulator is the realization of a Gaussian process with tensorized

covariance. We then assume that the integration between the codes is modeled with AR(1). However, the size of the data becomes too large and makes it impossible to solve the problem numerically. We therefore proposed to "compress" the data. To do this, we placed the data in a wavelet domain. We called this a Gaussian wavelet process. The advantage of this method is that one can afford not to take into account all the available data while keeping a satisfactory regression performance. The choice of point selection is a crucial point and may depend on the application. The major advantage of this method is that it has the same properties as the usual Gaussian processes. We have proposed illustrations of this method in different contexts in order to motivate further research in this area.

The third approach method is based on dimension reduction. This method was motivated by the impossibility of extending the time-series regression method by covariance tensorization to the multi-fidelity setting. We therefore carried out a study on a method, in chapter 6, combining dimension reduction by SVD and multi-fidelity AR(1) regression. Several methods of constructing the projection basis have been tested in order to take into account the uncertainties. In particular, to construct the dimension reduction we focused on low-fidelity data. This induces a risk of neglecting a quantity on the high-fidelity which is not negligible. Moreover, the uncertainty of the prediction is not fully taken into account. Therefore, a method has been introduced where the neglected part is taken into account using a time-series regression method with covariance tensorization. For our example, this method of dimension reduction and AR(1) model combined with time-series regression showed the best results amongst the tested methods, both in terms of prediction performance and uncertainty quantification.

Eventually, in this manuscript, we have proposed a convincing method for multi-fidelity regression of functional output of a computational code. We have proposed openings in BNN or wavelet decomposition. As an extension of the work presented in chapter 4 we imagine using GPBNN for time-series output. This would allow us to have a model that can address the problem of multi-fidelity regression with large output dimensions. We considered using the GP regression method with tensorized covariance for low-fidelity GP, as an (Perrin, 2020). The performance in low-fidelity regression is good, but the transition to high-fidelity has not been possible. We plan to use autoencoders in the GPBNN method in order to adapt it to time-series outputs. This model seems more suitable for GPBNN than a classical SVD method. Indeed, an SVD would not make sense in this context because it leads to a linear integration of the code, and in this case it would be more interesting to use the AR(1) model.

The extension of the WGP to the multi-fidelity framework is not yet fully completed, there is still room for improvement to have an efficient multi-fidelity regression method. Especially for the AR(1) model, it is expected that the implementation will not present any problems for prediction. However, the selection of training points is a problem, especially with multi-fidelity nested designs. Another challenge of WGP is the selection of points taking into account the prediction uncertainty. Indeed, if we are able to predict the prediction error by removing a point from the training set, this will give an efficient point selection criterion.

It may also be possible to construct priors for the WGP parameters.

The work proposed in chapter 6 is more advanced in terms of methodology than the other methods presented in this manuscript. The example presented to show the performance of the method is also more advanced than in the other chapters, however it could be interesting to present an even more complex application. It would be interesting to adapt this method to what is proposed in (Babaee et al., 2020). We could use this example with spatial and temporal variations. The dataset is particularly large, which poses a major challenge for AR(1) models but also for the determination of the projection basis. One can also consider extending this method to the high-dimension inputs. In this case, care must be taken to ensure that sufficient information is available to perform the regression. In addition, the experimental design will be essential in this case. The improvement of the current methodology can also be considered. For the learning of the orthogonal part, which is treated by Gaussian process regression with the covariance tensorization, we consider constructing objective priors for the temporal part. This would allow us to have a non-parametric Bayesian framework on the non-stationary part and would improve the flexibility of this model.

Furthermore, the use of the AR(1) model is not adapted to all types of problems. With the emergence of DeepGP for multi-fidelity regression, it is possible to consider combining DeepGP with some approaches proposed in this manuscript. We can take inspiration from methods such as those proposed in (Ming, Williamson, & Guillas, 2021).

Finally, one point has not been addressed at all in this manuscript, it is sequential learning for multi-fidelity. There is a lot of work on this problem that we have not considered, see (Ehara & Guillas, 2022; Li, Kirby, & Zhe, 2020). It might be possible to extend these types of methods to the models we propose in this manuscript.

Appendix A

LOO formula and discussion

In this appendix the computation of the LOO for GP regression with tensorized covariance with time-series output is studied. This refers to section 1.1.3. This computation is also used in chapter 6, in particular for the prediction of the orthogonal part in section 6.3.

LOO without loop In order to quickly minimize the LOO error with respect to the vector of correlation lengths $\ell_{\mathbf{x}}$ there exist formulas to evaluate $\varepsilon^2(\ell_{\mathbf{x}})$, section 1.1.3 and eq. (1.44), with matrix products (Bachoc, 2013, sec. 3; Dubrule, 1983). The LOO optimization problem is equivalent to minimize a function $f_{\mathbf{CV}}(\ell_{\mathbf{x}})$ given by:

$$f_{\mathbf{CV}}(\ell_{\mathbf{x}}) = z^T R_{\ell_{\mathbf{x}}}^{-1} \operatorname{diag}\left(R_{\ell_{\mathbf{x}}}^{-1}\right)^{-2} R_{\ell_{\mathbf{x}}}^{-1} z, \tag{A.1}$$

where z is a vector that collect all the data.

Considering Equation (1.36) and the mixed-product property, the inverse of a Kronecker product and the formula $\operatorname{diag}(A \otimes B) = \operatorname{diag} A \otimes \operatorname{diag} B$ the cost function can be expressed as:

$$f_{\mathbf{CV}}(\ell_{\mathbf{x}}) = z^T \left(R_t^{-1} \operatorname{diag}\left(R_t^{-1}\right)^{-2} R_t^{-1}\right) \otimes \left(R_x^{-1} \operatorname{diag}\left(R_x^{-1}\right)^{-2} R_x^{-1}\right) z. \tag{A.2}$$

However, the term in R_t is impossible to calculate because R_t is not invertible. $R_t^{-1} \operatorname{diag}\left(R_t^{-1}\right)^{-2} R_t^{-1}$ can be approximated by I_{N_t} in order to have a tractable problem. This assertion is equivalent to the hypothesis:

$$R_t^2 = \operatorname{diag}(R_t^{-1})^{-2}. \tag{A.3}$$

This assumption can be seen as the fact that the error is estimated by taking into account only the spatial distribution of the covariance. Indeed, to calculate the error only the matrix R_x is used, even if the value of R_t is calculated by maximum likelihood later in the GP regression method.

Thus, the minimization described in Equation (1.44) makes it possible to calculate the correlation lengths by minimizing:

$$f_{\mathbf{CV}}(\ell_{\mathbf{x}}) \simeq z^T I_{N_t} \otimes \left(R_x^{-1} \operatorname{diag}\left(R_x^{-1}\right)^{-2} R_x^{-1}\right) z, \tag{A.4}$$

where I_{N_t} is the $N_t \times N_t$ identity matrix. The main interest of this method is to give an approximate value of the error and to make the optimization much faster.

Optimization with hypothesis (A.3) Efficient minimization algorithms require to have the derivative of the function so that it does not have to be calculated by finite differences. Thanks to the simplification (A.4) it is possible to calculate the derivative of the LOO error (Bachoc, 2013):

$$\frac{\partial}{\partial \ell_{x_k}} f_{\mathbf{CV}}(\ell_{\mathbf{x}}) = 2z^T I_{N_t} \otimes R_x^{-1} \operatorname{diag}\left(R_x^{-1}\right)^{-2} \left(R_x^{-1} \frac{\partial R_x}{\partial \ell_{x_k}} R_x^{-1}\right) \operatorname{diag}\left(R_x^{-1}\right)^{-1} R_x^{-1} z \\ -2z^T I_{N_t} \otimes R_x^{-1} \operatorname{diag}\left(R_x^{-1}\right)^{-2} R_x^{-1} \frac{\partial R_x}{\partial \ell_{x_k}} R_x^{-1} z \quad , \tag{A.5}$$

with

$$\left(\frac{\partial R_x}{\partial \ell_{x_k}}\right)_{i,j} = \frac{\ell_{x_k}(x_{k,j} - x_{k,i})^2}{|x_{k,j} - x_{k,i}|} h'_{\frac{5}{2}}\left(\frac{|x_{k,j} - x_{k,i}|}{\ell_{x_k}}\right) \tag{A.6}$$

and

$$h'_{\frac{5}{2}}(x) = -\frac{5}{3} x \left(1 + \sqrt{5}x\right) \exp\left(-\sqrt{5}x\right). \tag{A.7}$$

The method used to calculate the value of $\ell_{\mathbf{x}}$ is the Nelder-Mead method with only one starting point, because starting from more points will be more costly and the function $f_{\mathbf{CV}}$ is close to quadratic consequently does not need multiple starting points.

Optimization without hypothesis (A.3): When hypothesis (A.3) does not hold, a way must be found to calculate $\ell_{\mathbf{x}}$ without this assumption. By a regularization of the matrix R_t it is possible to calculate $f_{\mathbf{CV}}(\ell_{\mathbf{x}})$ and its derivative by Equations (1.44), (A.2) and (A.8). However, the solution will be a regularized solution and not an exact solution.

There are different types of regularization that allow matrices to be inverted. Two methods have been investigated here. The first one is standard (Tilkonov regularization):

$$\widehat{R_t^{-1}} = \left(R_t^T R_t + \varepsilon^2 I_{N_t}\right)^{-1} R_t^T. \tag{A.8}$$

The second one is:

$$\widehat{R_t^{-1}} = \left(R_t + \varepsilon I_{N_t}\right)^{-1}. \tag{A.9}$$

It has the disadvantage of being more sensitive to ε than the first one, which is why it will not be used.

However, in the calculation of the determinant, this adjustment may have advantages. Denoting by $\widehat{R_t^{-1}} = V\Sigma^{-1}U^T$ the SVD of R_t, with $\Sigma^{-1} = \operatorname{diag}\frac{1}{\sigma_i+\varepsilon}$ whereas the same decomposition gives for Equation (A.8) $\Sigma^{-1} = \operatorname{diag}\frac{\sigma_i}{\sigma_i^2+\varepsilon^2}$. This is the reason why the two adjustments presented are not used in the same case. Indeed $\frac{1}{\sigma_i+\varepsilon}$ is less efficient for the calculation of a determinant but more efficient for the calculation of the inverse of $\widehat{R_t}$.

$$\frac{\partial}{\partial \ell_{x_k}} f_{\mathbf{CV}}(\ell_{\mathbf{x}}) = 2z^T \left(\widehat{R_t^{-1}} \operatorname{diag}\left(\widehat{R_t^{-1}}\right)^{-2} \widehat{R_t^{-1}}\right) \\ \otimes R_x^{-1} \operatorname{diag}\left(R_x^{-1}\right)^{-2} \left(R_x^{-1} \frac{\partial R_x}{\partial \ell_{x_k}} R_x^{-1}\right) \operatorname{diag}\left(R_x^{-1}\right)^{-1} R_x^{-1} z. \\ -2z^T \left(\widehat{R_t^{-1}} \operatorname{diag}\left(\widehat{R_t^{-1}}\right)^{-2} \widehat{R_t^{-1}}\right) \otimes R_x^{-1} \operatorname{diag}\left(R_x^{-1}\right)^{-2} R_x^{-1} \frac{\partial R_x}{\partial \ell_{x_k}} R_x^{-1} z \tag{A.10}$$

Equation (A.6) and Equation (A.7) are still valid.

This complete approach was compared to the LOO calculation using a loop. However, the calculation time of Kronecker products is too long compared to the calculation of the simple error with a loop. Moreover, the differences in the errors of the different methods are negligible. Thus this solution is only recommended when the calculation of Equations (A.2) and (A.10) is optimized.

Table A.1 shows that the gain in calculation time by the simplified method is significant even though the error difference is very small. The extremely long time for the complete LOO is mainly due to an implementation of the Kronecker product that is not very effective in our implementation.

	Loop LOO	Full LOO	Simplified LOO
Q^2	$7.06\ 10^{-4}$	$8.72\ 10^{-4}$	$8.14\ 10^{-4}$
time	18.41 s	3.47 min	0.17 s

Table A.1: Benchmark of the different LOO optimization techniques for GP regression with tensorized covariance of the function $f(\mathbf{x}, t) = \cos(4\pi(x_2 + 1)t)\sin(3\pi x_1 t)$. Loop LOO processes the error by computing the approximation, Full LOO processes the regularized analytic expression and Simplified LOO processes the simplified one given by Equation (A.4).

Appendix B

Expressions of some expectations and variances

In this section we show different computations useful for chapter 6.

B.1 Tensorized covariance of the orthogonal part

In this section we adapt the model proposed in section 1.1.3 for the model of the orthogonal part of section 6.3. Here we replace $\mathbf{Z}$ with $\mathbf{Z}^{\perp}$ and $\mathbf{Z}_{\text{obs}}$ with $\mathbf{Z}^{\perp}_{\text{obs}}$.

For R_x we assume that C_x is chosen in the Matérn-5/2 class of functions. The function only depends on the correlation length vector $\ell_{\mathbf{x}}$. The matrix R_t is estimated as described in Section 1.1.3 by the matrix $\widehat{R_t}$ given by:

$$\widehat{R_t} = \frac{1}{N_x}\left(\mathbf{Z}^{\perp}_{\text{obs}} - \hat{\mathbf{Z}}^{\perp}\right) R_x^{-1} \left(\mathbf{Z}^{\perp}_{\text{obs}} - \hat{\mathbf{Z}}^{\perp}\right)^T,$$

where $\hat{\mathbf{Z}}^{\perp}$ is the $N_t \times N_H$ matrix of empirical means $\hat{Z}^{\perp}_{u,i} = \frac{1}{N_H}\sum_{j=1}^{N_x}\left(\mathbf{Z}^{\perp}_{\text{obs}}\right)_{u,j}$, $\forall i = 1,\dots,N_H$ and $u = 1,\dots,N_t$. Its range is indeed in $S_N^{\perp}$.

The prediction mean is the sum of two terms, $\mathbf{Z}^{\perp}_{\text{obs}} R_x^{-1} r_x(\mathbf{x})$ which is $S_N^{\perp}$-valued and $B_\star u(\mathbf{x})$ also $S_N^{\perp}$-valued because:

$$B_\star = {\mathbf{Z}_{\text{obs}}}^{\perp} R_x^{-1} F \left(F^T R_x^{-1} F\right)^{-1}, \tag{B.1}$$

with F the $N_{\mathsf{F}} \times M$ matrix $\left[f^T(\mathbf{x}^{(i)})\right]_{i=1,\dots,N_{\mathsf{F}}}$. Consequently, we have:

$$Z^{\perp}_{\mathsf{H}}(\mathbf{x}, t_u) | \ell_{\mathbf{x}}, \mathbf{\Gamma}, N, \mathbf{Z}^{\perp}_{\text{obs}} \sim \mathcal{GP}(\mu_\star(\mathbf{x}), R_\star(\mathbf{x}, \mathbf{x}')) \tag{B.2}$$

where the mean is given by (1.38) and the covariance by (1.39) with $\mathbf{Z}^{\perp}_{\text{obs}}$ as the observed inputs. LOO estimation of the vector of correlation lengths $\ell_{\mathbf{x}}$ given $\mathbf{\Gamma}$ and N is carried out by the method presented in Appendix A.

B.2 Predictive mean and variance for multi-fidelity coefficients

In this section we compute the posterior prediction of the multi-fidelity coefficient $A_{i,\mathsf{H}}$ proposed in section 6.2 and section 6.3 knowing the basis $\mathbf{\Gamma}$. The model is proposed at eq. (6.26) and eq. (6.37).

Given $\mathbf{\Gamma}$, $\left(A_{i,\mathsf{H}}(\mathbf{x}, t_u), A_{i,\mathsf{L}}(\mathbf{x}, t_u)\right)_{\substack{\mathbf{x}\in Q \\ u=1,\dots,N_t}}$ are independent with respect to i. This independence makes it possible to consider N_t independent surrogate models, with mean and variance given by section 3.1.1.2:

$$\mathbb{E}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma},\mathcal{A}=\alpha\right] = h_{i,\mathsf{H}}^T(\mathbf{x})\Sigma_{i,\mathsf{H}}\nu_{i,\mathsf{H}} + r_{i,\mathsf{H}}^T(\mathbf{x})C_{i,\mathsf{H}}^{-1}\left(\alpha_i^{\mathsf{H}} - H_{i,\mathsf{H}}\Sigma_{i,\mathsf{H}}\nu_{i,\mathsf{H}}\right), \tag{B.3}$$

$$\begin{aligned}\mathbb{V}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma},\mathcal{A}=\alpha\right] &= \left(\hat{\rho}_{i,\mathsf{L}}^2(\mathbf{x}) + \varepsilon_{i,\rho}(\mathbf{x})\right)\sigma^2_{\hat{A}_{\mathsf{L},i}}(\mathbf{x}) \\ &\quad + \frac{Q_{i,\mathsf{H}}}{2(d_{i,\mathsf{H}}-1)}(1 - r_{i,\mathsf{H}}^T(\mathbf{x})C_{i,\mathsf{H}}^{-1}r_{i,\mathsf{H}}(\mathbf{x})) \\ &\quad + \left(h_{i,\mathsf{H}}^T(\mathbf{x}) - r_{i,\mathsf{H}}^T(\mathbf{x})C_{i,\mathsf{H}}^{-1}H_{i,\mathsf{H}}\right)\Sigma_{i,\mathsf{H}}\left(h_{i,\mathsf{H}}^T(\mathbf{x}) - r_{i,\mathsf{H}}^T(\mathbf{x})C_{i,\mathsf{H}}^{-1}H_{i,\mathsf{H}}\right)^T.\end{aligned} \tag{B.4}$$

B.3 Predictive variance for AR(1) multi-fidelity model with projection

In this section we compute the variance of the model proposed in section 6.2 for a CVB estimation of $\mathbf{\Gamma}$. This computation is used in section 6.2.3.2.

The law of total variance gives :

$$\mathbb{V}_\alpha\left[Z_{\mathsf{H}}(\mathbf{x},t_u)\right] = \mathbb{V}_\alpha\left[\mathbb{E}_\alpha\left[Z_{\mathsf{H}}(\mathbf{x},t_u)|\mathbf{\Gamma}\right]\right] + \mathbb{E}_\alpha\left[\mathbb{V}_\alpha\left[Z_{\mathsf{H}}(\mathbf{x},t_u)|\mathbf{\Gamma}\right]\right] \tag{B.5}$$

The variance term can be expressed as follows :

$$\begin{aligned}\mathbb{V}_\alpha\left[\mathbb{E}_\alpha\left[Z_{\mathsf{H}}(\mathbf{x},t_u)|\mathbf{\Gamma}\right]\right] &= \mathbb{V}_\alpha\left[\textstyle\sum_{i=1}^{N_t}\Gamma_i(t_u)\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ &= \textstyle\sum_{i=1}^{N_t}\mathbb{V}_\alpha\left[\Gamma_i(t_u)\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ &+ \textstyle\sum_{i,j=1,i\neq j}^{N_t}\mathrm{Cov}_\alpha(\Gamma_i(t_u)\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right], \Gamma_j(t_u)\mathbb{E}_\alpha\left[A_{j,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right])\end{aligned}, \tag{B.6}$$

where $\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]$ is given by eq. (B.3). The expectation term can be expressed as :

$$\begin{aligned}&\mathbb{E}_\alpha\left[\mathbb{V}_\alpha\left[Z_{\mathsf{H}}(\mathbf{x},t_u)|\mathbf{\Gamma}\right]\right] = \\ &\mathbb{E}_\alpha\left[\textstyle\sum_{i=1}^{N_t}\mathbb{V}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})\Gamma_i(t_u)|\mathbf{\Gamma}\right] + \sum_{i,j=1,i\neq j}^{N_t}\mathrm{Cov}_\alpha\left(A_{i,\mathsf{H}}(\mathbf{x})\Gamma_i(t_u), A_{j,\mathsf{H}}(\mathbf{x})\Gamma_j(t_u)|\mathbf{\Gamma}\right)\right] \\ &= \textstyle\sum_{i=1}^{N_t}\mathbb{E}_\alpha\left[\Gamma_i^2(t_u)\mathbb{V}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ &+ \textstyle\sum_{i,j=1,i\neq j}^{N_t}\mathbb{E}_\alpha\left[\Gamma_i(t_u)\Gamma_j(t_u)\,\mathrm{Cov}_\alpha\left(A_{i,\mathsf{H}}(\mathbf{x}), A_{j,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right)\right]\end{aligned}, \tag{B.7}$$

where $\mathbb{V}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})\Gamma_i(t_u)|\mathbf{\Gamma}\right]$ is given in eq. (B.4) and
$\mathrm{Cov}_\alpha(A_{i,\mathsf{H}}(\mathbf{x}), A_{j,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}) = 0$ if $i \neq j$. Consequently:

$$\begin{aligned}\mathbb{V}_\alpha\left[Z_{\mathsf{H}}(\mathbf{x},t_u)\right] &= \textstyle\sum_{i=1}^{N_t}\mathbb{V}_\alpha\left[\Gamma_i(t_u)\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ &+ \textstyle\sum_{i,j=1,i\neq j}^{N_t}\mathrm{Cov}_\alpha(\Gamma_i(t_u)\mathbb{E}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right], \Gamma_j(t_u)\mathbb{E}_\alpha\left[A_{j,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]) \\ &+ \textstyle\sum_{i=1}^{N_t}\mathbb{E}_\alpha\left[\Gamma_i^2(t_u)\mathbb{V}_\alpha\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right]\end{aligned}. \tag{B.8}$$

B.4 Predictive variance for AR(1) multi-fidelity model with projection and orthogonal part

In this section we compute the variance of the model proposed in section 6.3 for a CVB estimation of $\mathbf{\Gamma}$. This computation is used in section 6.3.2.2.

The theorem of the total variance gives us:

$$\begin{aligned}\mathbb{V}_{\mathbf{Z}_{\mathsf{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|N,\ell_{\mathbf{x}}\right] &= \mathbb{V}_{\mathbf{Z}_{\mathsf{obs}}}\left[\mathbb{E}_{\mathbf{Z}_{\mathsf{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ &+ \mathbb{E}_{\mathbf{Z}_{\mathsf{obs}}}\left[\mathbb{V}_{\mathbf{Z}_{\mathsf{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right]\end{aligned}. \tag{B.9}$$

The two terms of eq. (6.52) are:

$$\begin{array}{c}\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right]= \\ \mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}^{\perp}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]+\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}^{\parallel}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N\right]|N\right]\end{array}, \tag{B.10}$$

and

$$\begin{array}{c}\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right]= \\ \mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}^{\perp}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ +\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[2\mathsf{Cov}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}^{\parallel}\left(\mathbf{x},t_u\right),Z_{\mathsf{H}}^{\perp}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ +\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{V}_{\alpha}\left[Z_{\mathsf{H}}^{\parallel}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N\right]|N\right].\end{array}. \tag{B.11}$$

The uncorrelation of the $A_{i,\mathsf{H}}(\mathbf{x},t_u)$ coefficients given $\mathbf{\Gamma}$ gives $\mathrm{Cov}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x}),A_{j,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]=0$ for $i\neq j$ and $\mathrm{Cov}_{\mathbf{Z}_{\text{obs}}}\left[A_{i,\mathsf{H}}(\mathbf{x})\Gamma_i(t_u),Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|\mathbf{\Gamma}\right]=0$. This leads us to simplify eqs. (B.10) and (B.11) into:

$$\begin{array}{c}\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}(\mathbf{x},t_u)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]\right]=\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{E}_{\alpha}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ +\sum_{i=1}^{N}\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ +\sum_{i,j=1;i\neq j}^{N}\mathsf{Cov}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right],\Gamma_j(t_u)\mathbb{E}_{\alpha}\left[A_{j,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ +2\sum_{i=1}^{N}\mathsf{Cov}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right],\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\mathbf{l}_{\mathbf{x}}\right]\end{array}, \tag{B.12}$$

and

$$\begin{array}{c}\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]\right]=\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}^{\perp}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ +\sum_{i=1}^{N}\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)^2\mathbb{V}_{\alpha}\left[A_{i,\mathsf{H}}\left(\mathbf{x}\right)|\mathbf{\Gamma}\right]|N,\ell_{\mathbf{x}}\right]\end{array}. \tag{B.13}$$

The full formula of the variance can be expressed as :

$$\begin{array}{c}\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}\left(\mathbf{x},t_u\right)|N,\ell_{\mathbf{x}}\right]=\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ +\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\mathbb{V}_{\alpha}\left[Z_{\mathsf{H}}^{\perp}\left(\mathbf{x},t_u\right)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right] \\ +\sum_{i=1}^{N}\mathbb{V}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ +\sum_{i=1}^{N}\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)^2\mathbb{V}_{\alpha}\left[A_{i,\mathsf{H}}\left(\mathbf{x}\right)|\mathbf{\Gamma}\right]\right] \\ +\sum_{i,j=1;i\neq j}^{N}\mathsf{Cov}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right],\Gamma_j(t_u)\mathbb{E}_{\alpha}\left[A_{j,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]\right] \\ +2\sum_{i=1}^{N}\mathsf{Cov}_{\mathbf{Z}_{\text{obs}}}\left[\Gamma_i(t_u)\mathbb{E}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right],\mathbb{E}_{\mathbf{Z}_{\text{obs}}}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x},t_u)|\mathbf{\Gamma},N,\ell_{\mathbf{x}}\right]|N,\ell_{\mathbf{x}}\right]\end{array}. \tag{B.14}$$

where $\mathbb{V}_{\alpha}\left[A_{i,\mathsf{H}}(\mathbf{x})|\mathbf{\Gamma}\right]$ is given by eq. (B.4) and $\mathbb{V}_{\alpha}\left[Z_{\mathsf{H}}^{\perp}(\mathbf{x})|\mathbf{\Gamma}\right]$ is given by eq. (1.39).

Appendix C

Computation for Wavelet Gaussian process

In this appendix we focus on computations of the covariance for wavelet Gaussian process. This is related to chapter 5. In this appendix we focus of the case $j \neq 0$ and $j' \neq 0$.

C.1 Derivative of the covariance

In this section we compute the derivative of the covariance of the Haar Matérn 5/2 WGP. The derivative is computed in σ^2 and l.

$$\frac{\partial k_{MH}}{\partial \sigma^2}(j,k,j',k') = \frac{6400\sqrt{5}}{3\pi\sqrt{2^j 2^{j'}}}\left[-\sum_i \frac{\alpha_i \pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}}{1000}\left[3\sqrt{5}l^2 + 7|\gamma_i|l + \sqrt{5}\gamma_i^2\right] - \sum_i \frac{\pi\alpha_i|\gamma_i|l}{125}\right] \tag{C.1}$$

and

$$\begin{aligned}\frac{\partial k_{MH}}{\partial l}(j,k,j',k') = &-\frac{3200\sigma^2\sqrt{5}}{3\pi\sqrt{2^j 2^{j'}}}\sum_i \frac{\alpha_i \pi e^{-\frac{\sqrt{5}|\gamma_i|}{l}}}{1000}\left(6\sqrt{5}l + 7|\gamma_i|\right) \\ &-\frac{3200\sigma^2\sqrt{5}}{3\pi\sqrt{2^j 2^{j'}}}\sum_i \alpha_i \frac{\sqrt{5}\pi|\gamma_i|}{1000l^2} e^{-\frac{\sqrt{5}|\gamma_i|}{l}}\left(3\sqrt{5}l^2 + 7|\gamma_i|l + \sqrt{5}\gamma_i^2\right) \\ &-\frac{3200\sigma^2\sqrt{5}}{3\pi\sqrt{2^j 2^{j'}}}\sum_i \frac{\pi\alpha_i|\gamma_i|}{125}\end{aligned} \tag{C.2}$$

C.2 Computation of Tables 5.1 to 5.4

We express the values of α_i and γ_i such as in eq. (5.20):

$$\sum_i \alpha_i \cos(\xi\gamma_i) = \sin^2\left(\frac{2^j\xi}{4}\right)\sin^2\left(\frac{2^{j'}\xi}{4}\right)\cos\left(\xi\nu_{j,k,j',k'}\right), \tag{C.3}$$

with $\nu_{j,k,j',k'} = 2^j(k+\frac{1}{2}) - 2^{j'}(k'+\frac{1}{2})$. The trigonometry formula $\sin^2 a = \frac{1-\cos 2a}{2}$ gives us:

$$\sum_i \alpha_i \cos(\xi\gamma_i) = \frac{1}{4}\left(1-\cos\left(\frac{2^j\xi}{2}\right)\right)\left(1-\cos\left(\frac{2^{j'}\xi}{2}\right)\right)\cos\left(\xi\nu_{j,k,j',k'}\right). \tag{C.4}$$

We expand the equation:

$$\sum_i \alpha_i \cos(\xi\gamma_i) = \frac{1}{4}\left(1 - \cos\left(\frac{2^j\xi}{2}\right) - \cos\left(\frac{2^{j'}\xi}{2}\right) + \cos\left(\frac{2^j\xi}{2}\right)\cos\left(\frac{2^{j'}\xi}{2}\right)\right)\cos\left(\xi\nu_{j,k,j',k'}\right). \tag{C.5}$$

By the formula $\cos a \cos b = \frac{1}{2}(\cos(a-b) + \cos(a+b))$ we have:

$$\begin{aligned}\sum_i \alpha_i \cos(\xi\gamma_i) =& \frac{1}{4}\cos\left(\xi\nu_{j,k,j',k'}\right) - \frac{1}{4}\cos\left(\frac{2^j\xi}{2}\right)\cos\left(\xi\nu_{j,k,j',k'}\right) - \frac{1}{4}\cos\left(\frac{2^{j'}\xi}{2}\right)\cos\left(\xi\nu_{j,k,j',k'}\right) \\ &+ \frac{1}{8}\cos\left(\frac{2^j\xi}{2} - \frac{2^{j'}\xi}{2}\right)\cos\left(\xi\nu_{j,k,j',k'}\right) + \frac{1}{8}\cos\left(\frac{2^{j'}\xi}{2} + \frac{2^{j'}\xi}{2}\right)\cos\left(\xi\nu_{j,k,j',k'}\right).\end{aligned} \tag{C.6}$$

And then,

$$\begin{aligned}\sum_i \alpha_i \cos(\xi\gamma_i) =& \frac{1}{4}\cos\left(\xi\nu_{j,k,j',k'}\right) - \frac{1}{8}\cos\left(\xi\nu_{j,k,j',k'} + \frac{2^j\xi}{2}\right) - \frac{1}{8}\cos\left(\xi\nu_{j,k,j',k'} - \frac{2^j\xi}{2}\right) \\ &- \frac{1}{8}\cos\left(\xi\nu_{j,k,j',k'} + \frac{2^{j'}\xi}{2}\right) - \frac{1}{8}\cos\left(\xi\nu_{j,k,j',k'} + \frac{2^{j'}\xi}{2}\right) \\ &+ \frac{1}{16}\cos\left(\xi\nu_{j,k,j',k'} + \frac{2^j\xi}{2} - \frac{2^{j'}\xi}{2}\right) + \frac{1}{16}\cos\left(\xi\nu_{j,k,j',k'} - \frac{2^j\xi}{2} + \frac{2^{j'}\xi}{2}\right) \\ &+ \frac{1}{16}\cos\left(\xi\nu_{j,k,j',k'} + \frac{2^{j'}\xi}{2} + \frac{2^{j'}\xi}{2}\right) + \frac{1}{16}\cos\left(\xi\nu_{j,k,j',k'} - \frac{2^{j'}\xi}{2} - \frac{2^{j'}\xi}{2}\right).\end{aligned} \tag{C.7}$$

This equation gives table 5.4.

Bibliography

Abdallah, I., Lataniotis, C., & Sudret, B. (2017). Hierarchical Kriging for multi-fidelity aero-servo-elastic simulators - Application to extreme loads on wind turbines. arXiv. doi:10.48550/arXiv.1709.07637

Acharki, N., Bertoncello, A., & Garnier, J. (2023). Robust prediction interval estimation for Gaussian processes by cross-validation method. *Computational Statistics & Data Analysis*, *178*, 107597. doi:10.1016/j.csda.2022.107597

Althoff, D., Rodrigues, L. N., & Bazame, H. C. (2021). Uncertainty quantification for hydrological models based on neural networks: The dropout ensemble. *Stochastic Environmental Research and Risk Assessment*, *35*(5), 1051–1067. doi:10.1007/s00477-021-01980-8

Amato, U., Antoniadis, A., & De Feis, I. (2006). Dimension reduction in functional regression with applications. *Computational Statistics & Data Analysis*. Statistical Signal Extraction and Filtering, *50*(9), 2422–2446. doi:10.1016/j.csda.2004.12.007

Babaee, H., Bastidas, C., DeFilippo, M., Chryssostomidis, C., & Karniadakis, G. E. (2020). A Multifidelity Framework and Uncertainty Quantification for Sea Surface Temperature in the Massachusetts and Cape Cod Bays. *Earth and Space Science*, *7*(2). doi:10.1029/2019EA000954

Bach, F. R. & Jordan, M. I. (2002). Kernel Independent Component Analysis. *Journal ofMachine Learning Research*, 48.

Bachoc, F. (2013). Cross validation and maximum likelihood estimations of hyper-parameters of Gaussian processes with model misspecification. *Computational Statistics & Data Analysis*, *66*, 55–69.

Bachoc, F. & Lagnoux, A. (2021). Posterior contraction rates for constrained deep Gaussian processes in density estimation and classication. arXiv. doi:10.48550/arXiv.2112.07280

Balafas, K., Kiremidjian, A. S., & Rajagopal, R. (2018). The wavelet transform as a Gaussian process for damage detection. *Structural Control and Health Monitoring*, *25*(2), e2087. doi:10.1002/stc.2087

Balasubramanian, V., Ho, S.-S., & Vovk, V. (2014). *Conformal Prediction for Reliable Machine Learning: Theory, Adaptations and Applications*. Newnes.

Bell, A. J. & Sejnowski, T. J. (1995). An Information-Maximization Approach to Blind Separation and Blind Deconvolution. *Neural Computation*, *7*(6), 1129–1159. doi:10.1162/neco.1995.7.6.1129

Berger, J. O., Bernardo, J. M., & Sun, D. (2009). The formal definition of reference priors. *The Annals of Statistics*, *37*(2), 905–938. doi:10.1214/07-AOS587

Biau, G. & Scornet, E. (2016). A random forest guided tour. *TEST*, *25*(2), 197–227. doi:10.1007/s11749-016-0481-7

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., ... Goodman, N. D. (2019). Pyro: Deep Universal Probabilistic Programming. *The Journal of Machine Learning Research*, *20*(1), 973–978.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory* (pp. 144–152). doi:10.1145/130385.130401

Calderbank, A., Daubechies, I., Sweldens, W., & Yeo, B.-L. (1997). Lossless image compression using integer to integer wavelet transforms. In *Proceedings of International Conference on Image Processing* (Vol. 1, 596–599 vol.1). doi:10.1109/ICIP.1997.647983

Cigizoglu, H. K. & Alp, M. (2006). Generalized regression neural network in modelling river sediment yield. *Advances in Engineering Software*, *37*(2), 63–68. doi:10.1016/j.advengsoft.2005.05.002

Cohen, N., Sharir, O., & Shashua, A. (2016, June 6). On the Expressive Power of Deep Learning: A Tensor Analysis. In *Conference on Learning Theory* (pp. 698–728). Conference on Learning Theory. PMLR. Retrieved December 19, 2022, from https://proceedings.mlr.press/v49/cohen16.html

Conti, S., Gosling, J. P., Oakley, J. E., & O'Hagan, A. (2009). Gaussian process emulation of dynamic computer codes. *Biometrika*, *96*(3), 663–676.

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative Adversarial Networks: An Overview. *IEEE Signal Processing Magazine*, *35*(1), 53–65. doi:10.1109/MSP.2017.2765202

Curry, H. B. (1944). The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, *2*(3), 258–261. JSTOR: 43633461

Cutajar, K., Pullin, M., Damianou, A., Lawrence, N., & González, J. (2019). *Deep gaussian processes for multi-fidelity modeling*. arXiv: 1903.07320v1

Damianou, A. (2015). *Deep Gaussian Processes and Variational Propagation of Uncertainty* (Doctoral dissertation, University of Sheffield). Retrieved from https://etheses.whiterose.ac.uk/9968/

Damianou, A. & Lawrence, N. D. (2013). Deep Gaussian Processes. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics* (pp. 207–215). PMLR.

Daubechies, I. (1988). Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, *41*(7), 909–996. doi:10.1002/cpa.3160410705

De lozzo, M. (2013). *Modèles de substitution spatio-temporels et multifidélité : Application à l'ingénierie thermique*. Retrieved from https://www.theses.fr/2013ISAT0027

Diaconis, P. (2005). What is a random matrix. *Notices of the AMS*, *52*(11), 1348–1349.

Duane, S., Kennedy, A. D., Pendleton, B. J., & Roweth, D. (1987). Hybrid Monte Carlo. *Physics Letters B*, *195*(2), 216–222. doi:10.1016/0370-2693(87)91197-X

Dubrule, O. (1983). Cross validation of kriging in a unique neighborhood. *Journal of the International Association for Mathematical Geology*, *15*(6), 687–699.

Dupuy, D., Helbert, C., & Franco, J. (2015). DiceDesign and DiceEval: Two R Packages for Design and Analysis of Computer Experiments. *Journal of Statistical Software*, *65*(11), 1–38.

Ehara, A. & Guillas, S. (2022). An adaptive strategy for sequential designs of multilevel computer experiments. arXiv. doi:10.48550/arXiv.2104.02037. arXiv: 2104.02037 [stat]

Fisher, R. (1955). Statistical Methods and Scientific Induction. *Journal of the Royal Statistical Society: Series B (Methodological)*, *17*(1), 69–78. doi:10.1111/j.2517-6161.1955.tb00180.x

Forrester, A. I., Sóbester, A., & Keane, A. J. (2007). Multi-fidelity optimization via surrogate modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *463*(2088), 3251–3269. doi:10.1098/rspa.2007.1900

Frasconi, P., Gori, M., Member, S., & Sperduti, R. (1998). A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, *9*, 768–786.

Gal, Y. & Ghahramani, Z. (2016). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of The 33rd International Conference on Machine Learning* (pp. 1050–1059). PMLR.

Gallier, J. (2010). The Schur complement and symmetric positive semidefinite (and definite) matrices. *Penn Engineering*.

Garriga-Alonso, A., Rasmussen, C. E., & Aitchison, L. (2019). Deep Convolutional Networks as shallow Gaussian Processes. arXiv. doi:10.48550/arXiv.1808.05587. arXiv: 1808.05587 [cs, stat]

Gautschi, W. (2012). Numerical differentiation and integration. In *Numerical Analysis* (pp. 159–251). Springer.

Gelman, A. [A.], Gilks, W. R., & Roberts, G. O. (1997). Weak convergence and optimal scaling of random walk Metropolis algorithms. *The Annals of Applied Probability*, *7*(1), 110–120. doi:10.1214/aoap/1034625254

Ghojogh, B., Karray, F., & Crowley, M. (2019). *Eigenvalue and generalized eigenvalue problems: Tutorial*. arXiv: 1903.11240

Gihman, I. I. & Skorokhod, A. V. (2004). *The Theory of Stochastic Processes I* (Springer Berlin, Heidelberg).

GPy. (2012). GPy: A Gaussian process framework in python. Retrieved from http://github.com/SheffieldML/GPy

Grimmett, G. & Stirzaker, D. (2020). *Probability and Random Processes*. Oxford: Oxford University Press.

Gu, M. & Berger, J. O. (2016). Parallel partial Gaussian process emulation for computer models with massive output. *The Annals of Applied Statistics*, *10*(3), 1317–1347. doi:10.1214/16-AOAS934

Gu, M., Wang, X., & Berger, J. O. (2018). Robust Gaussian stochastic process emulation. *The Annals of Statistics*, *46*, 3038–3066. JSTOR: 26542892

Harville, D. A. (1998). Matrix algebra from a statistician's perspective. Taylor & Francis Group.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY: Springer New York. doi:10.1007/978-0-387-84858-7

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. (pp. 1026–1034). Proceedings of the IEEE International Conference on Computer Vision.

Hebbal, A., Brevault, L., Balesdent, M., Talbi, E.-G., & Melab, N. (2021). Bayesian optimization using deep Gaussian processes with applications to aerospace system design. *Optimization and Engineering*, *22*(1), 321–361. doi:10.1007/s11081-020-09517-8

Helton, J. C. & Davis, F. J. (2003). Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. *Reliability Engineering & System Safety*, *81*(1), 23–69. doi:10.1016/S0951-8320(03)00058-9

Hoffman, M. D. & Gelman, A. [Andrew]. (2014). The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res. 15*(1), 1593–1623.

Hong, Y., Meeker, W. Q., & McCalley, J. D. (2009). Prediction of Remaining Life of Power Transformers Based on Left Truncated and Right Censored Lifetime Data. *The Annals of Applied Statistics*, *3*(2), 857–879. JSTOR: 30244268

Hu, J. & Wang, J. (2015). Short-term wind speed prediction using empirical wavelet transform and Gaussian process regression. *Energy*, *93*, 1456–1466. doi:10.1016/j.energy.2015.10.041

Hyvärinen, A. & Oja, E. (2000). Independent component analysis: Algorithms and applications. *Neural Networks*, *13*(4), 411–430. doi:10.1016/S0893-6080(00)00026-5

Iooss, B., Veiga, S. D., Janon, A., Pujol, G., Broto, w. c. f. B., Boumhaout, K., ... Weber, F. (2020). *Sensitivity: Global Sensitivity Analysis of Model Outputs*. Retrieved from https://CRAN.R-project.org/package=sensitivity

Jagtap, A. D., Mitsotakis, D., & Karniadakis, G. E. (2022). Deep learning of inverse water waves problems using multi-fidelity data: Application to SerreGreenNaghdi equations. *Ocean Engineering*, *248*, 110775. doi:10.1016/j.oceaneng.2022.110775

Jakeman, J. D., Friedman, S., Eldred, M. S., Tamellini, L., Gorodetsky, A. A., & Allaire, D. (2022). Adaptive experimental design for multi-fidelity surrogate modeling of multi-disciplinary systems. *International Journal for Numerical Methods in Engineering*. doi:10.1002/nme.6958

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021a). Deep Learning. In G. James, D. Witten, T. Hastie, & R. Tibshirani (Eds.), *An Introduction to Statistical Learning: With Applications in R* (pp. 403–460). New York, NY: Springer US. doi:10.1007/978-1-0716-1418-1_10

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021b). Linear Regression. In G. James, D. Witten, T. Hastie, & R. Tibshirani (Eds.), *An Introduction to Statistical Learning: With Applications in R* (pp. 59–128). New York, NY: Springer US. doi:10.1007/978-1-0716-1418-1_3

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021c). Statistical Learning. In G. James, D. Witten, T. Hastie, & R. Tibshirani (Eds.), *An Introduction to Statistical Learning: With Applications in R* (pp. 15–57). New York, NY: Springer US. doi:10.1007/978-1-0716-1418-1_2

Jospin, L. V., Buntine, W., Boussaid, F., Laga, H., & Bennamoun, M. (2020). *Hands-on Bayesian Neural Networksa Tutorial for Deep Learning Users*. arXiv: 2007.06823

Kabir, H. M. D., Khosravi, A., Hosen, M. A., & Nahavandi, S. (2018). Neural Network-Based Uncertainty Quantification: A Survey of Methodologies and Applications. *IEEE Access*, *6*, 36218–36234. doi:10.1109/ACCESS.2018.2836917

Karaletsos, T. & Bui, T. D. (2020). Hierarchical Gaussian Process Priors for Bayesian Neural Network Weights. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 17141–17152). Curran Associates, Inc.

Karatzas, I. & Shreve, S. E. (1998). *Brownian Motion and Stochastic Calculus*. Graduate Texts in Mathematics. New York, NY: Springer New York. doi:10.1007/978-1-4612-0949-2

Kennedy, M. & O'Hagan, A. (2000). Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, *87*(1), 1–13. doi:10.1093/biomet/87.1.1

Kerleguer, B. (2021). *Multi-Fidelity Surrogate Modeling for Time-Series Outputs*. arXiv: 2109.11374 [math, stat]

Kerleguer, B., Cannamela, C., & Garnier, J. (2022). *A Bayesian neural network approach to Multi-fidelity surrogate modelling*. Retrieved from https://hal.archives-ouvertes.fr/hal-03608580

Khosravi, A., Nahavandi, S., Creighton, D., & Atiya, A. F. (2011). Lower Upper Bound Estimation Method for Construction of Neural Network-Based Prediction Intervals. *IEEE Transactions on Neural Networks*, *22*(3), 337–346. doi:10.1109/TNN.2010.2096824

Khosravi, A., Nahavandi, S., Srinivasan, D., & Khosravi, R. (2015). Constructing Optimal Prediction Intervals by Using Neural Networks and Bootstrap Method. *IEEE Transactions on Neural Networks and Learning Systems*, *26*(8), 1810–1815. doi:10.1109/TNNLS.2014.2354418

Kimeldorf, G. & Wahba, G. (1971). Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, *33*(1), 82–95. doi:10.1016/0022-247X(71)90184-3

Kingma, D. P. & Welling, M. (2014). Auto-Encoding Variational Bayes. arXiv. doi:10.48550/arXiv.1312.6114. arXiv: 1312.6114 [cs, stat]

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, *37*(2), 233–243. doi:10.1002/aic.690370209

Krige, D. G. (1951). A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy*, *52*(6), 119–139.

Kronland-Martinet, R., Morlet, J., & Grossmann, A. (1987). Analysis of sound patterns through wavelet transforms. *International Journal of Pattern Recognition and Artificial Intelligence*, *01*(02), 273–302. doi:10.1142/S0218001487000205

Le Gratiet, L. (2012). *MuFiCokriging: Multi-Fidelity Cokriging models*. Retrieved from https://CRAN.R-project.org/package=MuFiCokriging

Le Gratiet, L. (2013a). Bayesian analysis of hierarchical multifidelity codes. *SIAM/ASA Journal on Uncertainty Quantification*, *1*(1), 244–269.

Le Gratiet, L. (2013b). *Multi-fidelity Gaussian process regression for computer experiments* (Doctoral dissertation, Université Paris-Diderot - Paris VII). Retrieved April 6, 2020, from https://tel.archives-ouvertes.fr/tel-00866770

Le Gratiet, L. & Garnier, J. (2013). *Recursive co-kriging model for Design of Computer experiments with multiple levels of fidelity with an application to hydrodynamic*. arXiv: 1210.0686 [math, stat]

Le Gratiet, L., Marelli, S., & Sudret, B. (2015). *Metamodel-based sensitivity analysis: Polynomial chaos expansions and Gaussian processes*. arXiv: 1606.04273 [stat]

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541–551. doi:10.1162/neco.1989.1.4.541

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., & Sohl-Dickstein, J. (2018). Deep Neural Networks as Gaussian Processes. arXiv. doi:10.48550/arXiv.1711.00165

Lewis, A. & Knowles, G. (1992). Image compression using the 2-D wavelet transform. *IEEE Transactions on Image Processing*, *1*(2), 244–250. doi:10.1109/83.136601

Li, S., Kirby, R. M., & Zhe, S. (2020). *Deep Multi-Fidelity Active Learning of High-Dimensional Outputs*. arXiv: 2012.00901

Li, S., Xing, W., Kirby, R., & Zhe, S. (2020). Multi-Fidelity Bayesian Optimization via Deep Neural Networks. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 8521–8531). Curran Associates, Inc.

Liu, S. & Deng, W. (2015). Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)* (pp. 730–734). doi:10.1109/ACPR.2015.7486599

Ma, P. (2020). Objective Bayesian Analysis of a Cokriging Model for Hierarchical Multifidelity Codes. *SIAM/ASA Journal on Uncertainty Quantification*, *8*(4), 1358–1382. doi:10.1137/19M1289893. arXiv: 1910.10225

MacKay, D. J. (1992). A practical Bayesian framework for backpropagation networks. *Neural computation*, *4*(3), 448–472.

Mallat, S. G. (2009). *A wavelet tour of signal processing: The sparse way* (3rd ed). Amsterdam ; Boston: Elsevier/Academic Press.

Mallat, S. (1989). A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *11*(7), 674–693. doi:10.1109/34.192463

Marque-Pucheu, S. (2018). *Gaussian process regression of two nested computer codes* (Doctoral dissertation, Université Paris-Diderot - Paris VII). Retrieved from https://hal.archives-ouvertes.fr/tel-02092072

Matsubara, T., Oates, C. J., & Briol, F.-X. (2020). The Ridgelet Prior: A Covariance Function Approach to Prior Specification for Bayesian Neural Networks.

Meng, X., Babaee, H., & Karniadakis, G. E. (2021). Multi-fidelity Bayesian Neural Networks: Algorithms and Applications. *Journal of Computational Physics*, *438*, 110361. doi:10.1016/j.jcp.2021.110361. arXiv: 2012.13294

Meng, X. & Karniadakis, G. E. (2020). A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. *Journal of Computational Physics*, *401*, 109020. doi:10.1016/j.jcp.2019.109020

Mertens, B., Fearn, T., & Thompson, M. (1995). The efficient cross-validation of principal components applied to principal component regression. *Statistics and Computing*, *5*(3), 227–235.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953, June). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. doi:10.1063/1.1699114

Michel, B. & Nouy, A. (2022, May). Learning with tree tensor networks: Complexity estimates and model selection. *Bernoulli*, *28*(2), 910–936. doi:10.3150/21-BEJ1371

Ming, D., Williamson, D., & Guillas, S. (2021). Deep Gaussian Process Emulation using Stochastic Imputation. arXiv. doi:10.48550/arXiv.2107.01590

Nanty, S., Helbert, C., Marrel, A., Pérot, N., & Prieur, C. (2017). Uncertainty quantification for functional dependent random variables. *Computational Statistics*, *32*(2), 559–583.

Nash, J. E. & Sutcliffe, J. V. (1970). River flow forecasting through conceptual models part I A discussion of principles. *Journal of Hydrology*, *10*(3), 282–290. doi:10.1016/0022-1694(70)90255-6

Ng, L. W.-T. & Eldred, M. (2012). Multifidelity Uncertainty Quantification Using Non-Intrusive Polynomial Chaos and Stochastic Collocation. In *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference; 20th AIAA/ASME/AHS Adaptive Structures Conference;14th AIAA*. American Institute of Aeronautics and Astronautics. doi:10.2514/6.2012-1852

Nguyen, D. & Widrow, B. (1990). Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *1990 IJCNN International Joint Conference on Neural Networks* (21–26 vol.3). doi:10.1109/IJCNN.1990.137819

Olver, F. W., Lozier, D. W., Boisvert, R. F., & Clark, C. W. (2010). *NIST Handbook of Mathematical Functions*. Cambridge University Press.

Palar, P. S., Zuhal, L. R., Shimoyama, K., & Tsuchiya, T. (2018). Global sensitivity analysis via multi-fidelity polynomial chaos expansion. *Reliability Engineering & System Safety*, *170*, 175–190. doi:10.1016/j.ress.2017.10.013

Pasini, A. (2015). Artificial neural networks for small dataset analysis. *Journal of Thoracic Disease*, *7*(5), 953–960. doi:10.3978/j.issn.2072-1439.2015.04.61

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems* (Vol. 32). Curran Associates, Inc.

Patsialis, D. & Taflanidis, A. A. (2021). Multi-fidelity Monte Carlo for seismic risk assessment applications. *Structural Safety*, *93*, 102129. doi:10.1016/j.strusafe.2021.102129

Perdikaris, P., Raissi, M., Damianou, A., Lawrence, N. D., & Karniadakis, G. E. (2017). Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *473*(2198), 20160751. doi:10.1098/rspa.2016.0751

Perrin, G. (2020). Adaptive calibration of a computer code with time-series output. *Reliability Engineering and System Safety*, *196*, 106728.

Picheny, V. & Ginsbourger, D. (2014). Noisy kriging-based optimization methods: A unified implementation within the DiceOptim package. *Computational Statistics & Data Analysis*, *71*, 1035–1053. doi:10.1016/j.csda.2013.03.018

Pilania, G., Gubernatis, J., & Lookman, T. (2017). Multi-fidelity machine learning models for accurate bandgap predictions of solids. *Computational Materials Science*, *129*, 156–163. doi:10.1016/j.commatsci.2016.12.004

Radaideh, M. I. & Kozlowski, T. (2020). Surrogate modeling of advanced computer simulations using deep Gaussian processes. *Reliability Engineering & System Safety*, *195*, 106731. doi:10.1016/j.ress.2019.106731

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). *Searching for Activation Functions*. arXiv: 1710.05941 [cs]

Rasamoelina, A. D., Adjailia, F., & Sinák, P. (2020, January). A Review of Activation Function for Artificial Neural Network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)* (pp. 281–286). 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI). doi:10.1109/SAMI48414.2020.9108717

Rasmussen, C. E. & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press.

Rioul, O. & Duhamel, P. (1992). Fast algorithms for discrete and continuous wavelet transforms. *IEEE Transactions on Information Theory*, *38*(2), 569–586. doi:10.1109/18.119724

Robert, C. P. (2004). *The Bayesian Choice* (Springer New York, NY). Retrieved September 12, 2022, from https://link.springer.com/book/10.1007/0-387-71599-1

Robinson, G. K. (1991). That BLUP is a Good Thing: The Estimation of Random Effects. *Statistical Science*, *6*(1), 15–32. doi:10.1214/ss/1177011926

Rougier, J. (2008). Efficient emulators for multivariate deterministic functions. *Journal of Computational and Graphical Statistics*, *17*(4), 827–843.

Ruder, S. (2017). *An overview of gradient descent optimization algorithms*. arXiv: 1609.04747 [cs]

Rullière, D., Durrande, N., Bachoc, F., & Chevalier, C. (2018). Nested Kriging predictions for datasets with a large number of observations. *Statistics and Computing*, *28*(4), 849–867. doi:10.1007/s11222-017-9766-2

Saltelli, A., Annoni, P., Azzini, I., Campolongo, F., Ratto, M., & Tarantola, S. (2010). Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. *Computer physics communications*, *181*(2), 259–270.

Santner, T. J., Williams, B. J., & Notz, W. (2003). *The design and analysis of computer experiments*. Springer, New York, NY.

Schölkopf, B., Burges, C. J., & Smola, A. (1999). *Advances in Kernel Methods: Support Vector Learning*. MIT Press. Google Books: _NYamXKkNM8C

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, *10*(5), 1299–1319. doi:10.1162/089976698300017467

Song, J., Chen, Y., & Yue, Y. (2019). A general framework for multi-fidelity bayesian optimization with gaussian processes. In *The 22nd International Conference on Artificial Intelligence and Statistics* (pp. 3158–3167). PMLR.

Stein, M. L. (1999). *Interpolation of Spatial Data: Some Theory for Kriging*. Chicago: Springer Science & Business Media.

Tiwari, K. & Chong, N. Y. (2019). *Multi-robot exploration for environmental monitoring* (Academic Press). doi:10.1016/B978-0-12-817607-8.00019-8

Tiwari, M. K. & Chatterjee, C. (2010). Uncertainty assessment and ensemble flood forecasting using bootstrap based artificial neural networks (BANNs). *Journal of Hydrology*, *382*(1), 20–33. doi:10.1016/j.jhydrol.2009.12.013

Tripathy, R. K. & Bilionis, I. (2018). Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, *375*, 565–588. doi:10.1016/j.jcp.2018.08.036

Trottier, L., Giguere, P., & Chaib-draa, B. (2017). Parametric Exponential Linear Unit for Deep Convolutional Neural Networks. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 207–214). doi:10.1109/ICMLA.2017.00038

Usevitch, B. (2001). A tutorial on modern lossy wavelet image compression: Foundations of JPEG 2000. *IEEE Signal Processing Magazine*, *18*(5), 22–35. doi:10.1109/79.952803

Van Loan, C. F. & Golub, G. H. (1983). *Matrix computations*. Johns Hopkins University Press, Baltimore.

Williams, C. (1996). Computing with Infinite Networks. In *Advances in Neural Information Processing Systems* (Vol. 9). MIT Press.

Xiu, D. & Karniadakis, G. E. (2002). The Wiener–Askey Polynomial Chaos for Stochastic Differential Equations. *SIAM Journal on Scientific Computing*, *24*(2), 619–644. doi:10.1137/S1064827501387826

Zertuche, F. (2015). *Assessment of uncertainty in computer experiments when working with multifidelity simulators.* (Doctoral dissertation, Université Grenoble Alpes). Retrieved from https://tel.archives-ouvertes.fr/tel-01240812

Zhang, X., Xie, F., Ji, T., Zhu, Z., & Zheng, Y. (2021). Multi-fidelity deep neural network surrogate model for aerodynamic shape optimization. *Computer Methods in Applied Mechanics and Engineering*, *373*, 113485. doi:10.1016/j.cma.2020.113485

Zhou, Q., Wu, Y., Guo, Z., Hu, J., & Jin, P. (2020). A generalized hierarchical co-Kriging model for multi-fidelity data fusion. *Structural and Multidisciplinary Optimization*, *62*(4), 1885–1904. doi:10.1007/s00158-020-02583-7

Zhou, T. & Peng, Y. (2020). Kernel principal component analysis-based Gaussian process regression modelling for high-dimensional reliability analysis. *Computers & Structures*, *241*, 106358. doi:10.1016/j.compstruc.2020.106358

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., ... He, Q. (2021). A Comprehensive Survey on Transfer Learning. *Proceedings of the IEEE*, *109*(1), 43–76. doi:10.1109/JPROC.2020.3004555

Contents

ECOLE
DOCTORALE
DE MATHEMATIQUES
HADAMARD

Titre: Métamodèle multifidélité adaptés aux sorties fonctionnelles pour la quantification de l'incertitude de modèles complexes

Mots clés: Quantification d'incertitude, métamodel, multi-fidélité

Résumé: Cette thèse porte sur l'approximation de la sortie d'un code de calcul complexe dans un cadre multi-fidélité, c'est-à-dire que le code peut être exécuté à différents niveaux de précision avec différents coûts de calcul. Ainsi, les qualités de prédictions du métamodèle de la sortie d'un code complexe peuvent être améliorées en utilisant en plus des simulations moins précises mais plus nombreuses (car moins coûteuses) .
Ce travail a pour objectif d'étendre les méthodes de métamodélisation multi-fidélité lorsque les sorties du code sont fonctionnelles. Tout d'abord, une première approche permettant de combiner les réseaux de neurones bayésiens et les processus gaussiens est proposée. Ce modèle est adapté lorsque les relations entre les codes basse et haute fidélité est non linéaire, mais n'est pas encore adapté aux sorties fonctionnelles.
Dans un second temps, une approche utilisant la décomposition en ondelettes et les processus gaussien est proposée. Cette approche permet de développer un processus gaussien dans l'espace des ondelettes qui est équivalent à un processus gaussien dans l'espace temporel. Cette méthode est naturellement adaptée aux sorties fonctionnelles.
Enfin, la troisième méthode proposée associe une méthode de réduction de dimension de la sortie avec une méthode de régression par processus gaussien à covariance tensorisée. Cette approche est développée dans le cadre de sortie de type série-temporelle. Les expressions analytiques de la moyenne et de la variance de prédiction sont également introduites.

Title: Multi-fidelity surrogate modeling adapted to functional outputs for uncertainty quantification of complex models

Keywords: Uncertainty Quantification, surrogate modeling, Multi-fidelity

Abstract: This thesis focuses on approximating the output of a complex computational code in a multi-fidelity framework, i.e. the code can be run at different levels of accuracy with different computational costs. Thus, the predictive qualities of the surrogate model of the output of a complex code can be improved by using in addition less accurate but more numerous (because less expensive) simulations.
This work aims to extend multi-fidelity surrogate modeling methods when the code outputs are functional. First, an approach allowing to combine Bayesian neural networks and Gaussian processes is proposed. This model is suitable when the relationship between the low- and high-fidelity codes is non-linear, but is not yet suitable for functional outputs.
In a second step, an approach using wavelet decomposition and Gaussian processes is proposed. This approach allows to develop a Gaussian process in the wavelet space which is equivalent to a Gaussian process in the time space. This method is naturally adapted to functional outputs.
Finally, the third proposed method combines an output dimension reduction method with a covariance tensorised Gaussian process regression method. This approach is developed in the context of time-series output. The analytical expressions for the predictive mean and variance are also introduced.

Institut Polytechnique de Paris
91120 Palaiseau, France