UNIVERSITÉ DE LORRAINE | BIBLIOTHÈQUES UNIVERSITAIRES

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact bibliothèque : ddoc-theses-contact@univ-lorraine.fr
(Cette adresse ne permet pas de contacter les auteurs)

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Optimisation de la performance des systèmes à partir de modèles identifiés: Application au design de projectiles et à la commande feedforward des systèmes Wiener.

THÈSE

présentée et soutenue publiquement le 25 Mars 2025

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention Automatique, Traitement du signal et des images, Génie informatique)

par

Alain Uwadukunze

Composition du jury

Président :	Guillaume MERCERE	Professeur, Université de Poitiers
Rapporteurs :	Stéphane VICTOR	Maître de conférences HdR, Université de Bordeaux
	Jonathan WEBER	Professeur, Université de Haute-Alsace
Examinateur :	Guillaume MERCERE	Professeur, Université de Poitiers
Directeurs de thèse :	Xavier BOMBOIS	Directeur de Recherche CNRS, Laboratoire Ampère
	Marion GILSON	Professeur, Université de Lorraine
Co-encadrante :	Marie ALBISSER	Chargée de recherche, Institut franco-allemand de recherches de Saint-Louis

Université de Lorraine, CRAN, CNRS — UMR 7039

Mis en page avec la classe thesul.

System Performance Optimization based on Identified Models: Application to Projectile Design and Feedforward Control of Wiener Systems

Acknowledgments

At the end of this thesis journey, I would like to express my gratitude to all those who directly or indirectly contributed to the completion of this chapter of my life.

First of all I would like to thank my PhD Supervisors, Pr. Marion Gilson, Pr. Xavier Bombois and Dr. Marie Albisser.

Thank you Marie, for believing in me and giving me the chance to do an internship at ISL which ultimately led to the thesis work. I have appreciated working with you the last 4 years. Your advice, constructive feedback and teaching in the domain of ballistics, have greatly improved my skills.

Like Marie, I started working with Marion during my internship and later on the PhD thesis. Thank you Marion for your moral and scientific support throughout the thesis. Your guidance during the PhD, as well as your advice on writing and organisation, have been invaluable. Thank you also for joining me for runs, even on cold and rainy days in Nancy.

I met Xavier at the beginning of my thesis, and I am very grateful that he was part of this adventure. Thank you Xavier for sharing your knowledge and passion in the fields of control and identification with me. Thank you also for your patience and time taken to help me improve my writing and work overall. I would also like to thank you for your multiple invitations to Lyon, and especially for inviting me to your home with your family and preparing a delicious meal!

I would also like to express my gratitude to the members of my "Comitée de Suivi Interne", Pr. Guillaume Mercère who also accepted to be part of my thesis jury and Dr. Vincent Laurain. Thank you for accepting to follow my thesis and for providing me with constructive advice during the the yearly meetings. I would also like to thank the other members of my thesis jury who accepted to review my work, Pr. Jonathan Weber and Associate Professor Stéphane Victor.

Most of my PhD was spent at ISL in Saint-Louis, within the Aerodynamic and eXterior Ballistics group (ABX). I would thus like to thank my colleagues at ABX who gave me a warm welcome and always made me feel comfortable and part of the group. I would like to thank in particular Dr. Bastien Martinez, head of the group ABX, whose support and numerous scientific discussions helped me advance in this thesis. Thank you also to Dr. Daniel Kharlamov for providing me with the CFD data used in my PhD. Finally, I would like to thank Andreas for sharing an office with me for 3 years, lending me a bike when I first arrived in Saint-Louis and serving as my advisor when I bought my car! The list of people I would like to thank in ABX could go on, but to finish, I would like to say thank you to all the other members for their kindness towards me, for (desperately) trying to teach me German, for donner "donnerstag" and for the interesting discussions at coffee breaks.

Although I did not spend a much time at CRAN at Polytech Nancy, I was always well received there. I would therefore like to thank the members of CRAN, the PhD Students with whom I shared the office for a few days, as well as the researchers (and my former teachers) in Nancy for their warm welcome.

Sports have always been a significant part of my life, and always helped me go through diffi-

cult times. Since it is always better to do sports with others, I would like to thank the amazing people I met and did sports with during the PhD. First of all, thank you to members of ISL volleyball team who introduced me to a sport which became later one of my passions. Thank you to various team-mates and coaches at VBCR Huningue who helped me improve and with whom I shared amazing moments. Finally, I would also like to thank my faithful ISL launch runs colleagues for sharing amazing chill runs and painful, yet enjoyable, interval run exercises with me.

I would like to finish by thanking people, who even from afar always sent me their support. Thank you to my high school friends from Moselle, "a.k.a Guillemettes 2.0", for their infinite support these last 3 years. Finally, but not least I would like to thank my family, my mom, Thierry, my little sister, my brother and his cat Eden (who was kind with me when I was at my brother's place during the times in Nancy) for their encouragements and trust in my abilities since the beginning!

"Le vent se lève,
il faut tenter de vivre."
Paul Valéry

Contents

List of Figures xi

List of Tables xiii

List of Algorithms xv

Notations xvii

Résumé en français 1

Introduction 7

List of Publications 13

Chapter 1
Data-driven models for static system identification 15

1.1 Review of possible surrogate models . . . 15
1.2 Neural Networks for regression . . . 17
1.2.1 Multi Layer Perceptrons (MLP) parametrization . . . 17
1.2.2 Identification of a MLP . . . 19
1.3 Gaussian Processes for regression . . . 21
1.3.1 Single Output Gaussian Processes . . . 21
1.3.2 Multiple correlated Output Gaussian Processes . . . 23
1.4 Performance metrics . . . 24
1.5 Conclusion . . . 24

Chapter 2
Data selection to improve models for optimization 25

2.1 Surrogate based optimization . . . 25
2.2 Bayesian Optimization . . . 28
2.2.1 Acquisition functions . . . 30

2.2.2 Bayesian Optimization for constrained optimization . . . 31
2.3 Neural Network Smart Grid Based Optimization . . . 32
2.3.1 Exploring Multi Layer Perceptrons . . . 33
2.3.2 Acquisition function for the MLP . . . 35
2.3.3 Constrained optimization case . . . 36
2.4 Numerical Illustration . . . 39
2.5 Conclusion . . . 41

Chapter 3
Aerodynamic coefficients prediction **43**

3.1 Aerodynamic Coefficients . . . 44
3.1.1 Aerodynamic Forces and Moments . . . 44
3.1.2 Stability derivatives . . . 47
3.1.3 Aerodynamic coefficients estimation methods . . . 47
3.2 Review of methods used to predict aerodynamic coefficients . . . 49
3.3 Study Case . . . 51
3.3.1 Projectile . . . 51
3.3.2 Dataset . . . 52
3.3.3 Estimating f_{CFD} from f_{aero} or f_{PRODAS} . . . 52
3.3.4 Results . . . 54
3.4 Conclusion . . . 57

Chapter 4
Projectile Shape Optimization **59**

4.1 Aerodynamic design . . . 60
4.2 Aerodynamic design problem . . . 61
4.3 Optimization using Neural Networks as surrogate . . . 65
4.4 Optimization using Bayesian Optimization . . . 66
4.5 Possibility to evaluate only one configuration . . . 67
4.6 Results . . . 68
4.6.1 Setup and initial database . . . 68
4.6.2 Adding one additional configuration . . . 69
4.6.3 Results using Bayesian Optimization . . . 72
4.6.4 Results using the Neural Network as surrogate model . . . 72
4.6.5 Comparison to alternative approaches . . . 76
4.7 Conclusion . . . 77

Chapter 5
Identification for feedforward control of Wiener systems **79**

5.1 Introduction . . . 79
5.2 Framework . . . 81
5.3 Structures and identification criterion . . . 83
5.3.1 Parametrization and identification of $\mathcal{M}$ . . . 83
5.3.2 Parametrization and design of $\mathcal{F}$ . . . 84
5.4 Iterative learning procedure . . . 85
5.5 Numerical Examples . . . 86
5.5.1 Controller synthesis without an iterative procedure . . . 87
5.5.2 Application of the iterative procedure . . . 90
5.5.3 An example with the knowledge of the exact parametrization . . . 94
5.6 Conclusion . . . 96

Chapter 6
General Conclusions and Perspectives **97**

6.1 Conclusions . . . 97
6.2 Perspectives . . . 100

Appendix A
Illustration of Controller Design using NNSGO

Bibliography

Résumé **113**

Abstract **114**

List of Figures

1 Basic Finner, un exemple classique d'un projectile équipé d'ailettes, dans la même famille que les projectiles étudiés dans cette thèse. 2

1 Basic Finner, a classic example of a projectile fitted with fins in the same family of projectiles as the one studied in this thesis 8

1.1 MLP with only one hidden layer 18

2.1 Value of GP σ (represented by the blue stars, note that this value is multiplied by 4 for better visualization) and MLP error (represented by the orange dots) w.r.t δ_{min} (horizontal axis). 34
2.2 Value of $\lambda(X)$ (vertical axis) when the constraints are respected (green) and when they are not respected (red) w.r.t δ_{min}^{scaled} (horizontal axis). 38

3.1 Body and Wind frames 45
3.2 Summary of aerodynamic coefficients estimation methods 49
3.3 Finned projectile characteristics (Table 3.1) 51
3.4 Prediction of C_D coefficient for a projectile configuration $X_{1,..,6}$ not in the training dataset at different Mach numbers X_7, using the models f_{mean} (yellow), $\hat{f}_3$ (violet), $\hat{f}_2$ (blue dots), $\hat{f}_1$ (black) and comparison with the target value f_{CFD} (green) and the lower fidelity models f_{aero} (red) and f_{PRODAS} (blue square) 56
3.5 Prediction of C_L coefficient for a projectile configuration $X_{1,..,6}$ not in the training dataset at different Mach numbers X_7, using the models f_{mean} (yellow), $\hat{f}_3$ (violet), $\hat{f}_2$ (blue dots), $\hat{f}_1$ (black) and comparison with the target value f_{CFD} (green) and the lower fidelity models f_{aero} (red) and f_{PRODAS} (blue square) 56
3.6 Prediction of C_m coefficient for a projectile configuration $X_{1,..,6}$ not in the training dataset at different Mach numbers X_7, using the models f_{mean} (yellow), $\hat{f}_3$ (violet), $\hat{f}_2$ (blue dots), $\hat{f}_1$ (black) and comparison with the target value f_{CFD} (green) and the lower fidelity models f_{aero} (red) and f_{PRODAS} (blue square) 57

4.1 Finner projectile with rectangular fins 62
4.2 Neural Network prediction $\hat{F}_1(X_{new}, M)$ (blue) and $F_1(X_{new}, M)$ evaluated with PRODAS (green) for $X_{new} = (12.6859, 10, 1.34, 1.72, 0.43)^T$ and for characteristic values of $M \in [2\ 5]$. 70
4.3 Neural Network prediction $\hat{F}_2(X_{new}, M)$ (blue) and $F_2(X_{new}, M)$ evaluated with PRODAS (green) for $X_{new} = (12.6859, 10, 1.34, 1.72, 0.43)^T$ and for characteristic values of $M \in [2\ 5]$. The yellow line represents the threshold $C_{m_\alpha}^{max} = -10$. 71

4.4 Neural Network prediction $\hat{F}_3(X_{new}, M)$ (blue) and $F_3(X_{new}, M)$ evaluated with PRODAS (green) for $X_{new} = (12.6859, 10, 1.34, 1.72, 0.43)^T$ and for characteristic values of $M \in [2\ 5]$. 71
4.5 $F_1(X, M)$ (evaluated with PRODAS) for characteristic values of $M \in [2\ 5]$ and for three values of X, *i.e.*, $X = X^*_{BO}$ (blue), $X = X^*_{NN}$ (green) and $X = X^*_{\mathcal{D}_{X,almost}}$ (red). 74
4.6 $F_2(X, M)$ (evaluated with PRODAS) for characteristic values of $M \in [2\ 5]$ and for three values of X, *i.e.*, $X = X^*_{BO}$ (blue), $X = X^*_{NN}$ (green) and $X = X^*_{\mathcal{D}_{X,almost}}$ (red). The yellow line represents the threshold $C^{max}_{m_\alpha} = -10$. 74
4.7 $F_3(X, M)$ (evaluated with PRODAS) for characteristic values of $M \in [2\ 5]$ and for three values of X, *i.e.*, $X = X^*_{BO}$ (blue), $X = X^*_{NN}$ (green) and $X = X^*_{\mathcal{D}_{X,almost}}$ (red). 75

5.1 Control scheme 81
5.2 Model output $\hat{y}_1$ (blue), true system output y_1 (black) and difference between $\hat{y}_1$ and y_1 (red) for a given input $u_1(t)$ 88
5.3 Model output $\hat{y}_a$ (blue), true system output y_a (green), reference signal r_a (black) and difference between r_a and y_a (red) when the input is the control signal provided by the designed controller $\mathcal{F}_1$ 89
5.4 New reference signal to follow r_b (in blue) compared to $y_1(t)$ used to identify the model $\mathcal{M}_1$ (in black) 90
5.5 Model output $\hat{y}_b$ (blue), true system output y_b (green), reference signal r_b (black) and difference between r_b and y_b (red) when the input is the control signal $u_b(t)$ provided by the controller designed using $\mathcal{M}_1$ and r_b 91
5.6 Model output $\hat{y}_b$ (blue), true system output y_b (green), reference signal r_b (black) and difference between r_b and y_b (red) when the input is the control signal $u_b(t)$ provided by the new designed controller $\mathcal{F}_2$ 92
5.7 Static nonlinearity of the model after the first identification $\hat{g}_{\theta 1}$ (red), after the second identification $\hat{g}_{\theta 2}$ (blue) and true system static nonlinearity g_0 (green). . . 93
5.8 Model output $\hat{y}_c$ (blue), true system output y_c (green), reference signal r_c (black) and difference between r_c and y_c (red) when the input is the control signal $u_c(t)$ provided by the controller $\mathcal{F}_2$ 93
5.9 Model output $\hat{y}_b$ (blue), true system output y_b (green), reference signal r_b (black) and difference between r_b and y_b (red) when the input is the control signal $u_b(t)$ provided by the controller $\mathcal{F}_{1,bis}$ 95

A.1 Target step response (in black) compared to the step response based on the controllers found using the NNSGO approach (θ_{NNSGO} in red) and BO (θ_{BO} in blue) 105

List of Tables

2.1 Optimization results with 100 additional points 40

3.1 Main characteristics of a projectile fitted with fins 51
3.2 Neural Network Architectures used and performance metrics for the two methods for the C_D coefficient 54
3.3 Neural Network Architectures used and performance metrics for the two methods for the C_L coefficient 55
3.4 Neural Network Architectures used and performance metrics for the two methods for the C_m coefficient 55

4.1 Main characteristics of a rectangular finned projectile 62
4.2 Definition of the set $\mathcal{X}$ for each characteristic X_i of the geometry (X_1, X_3, X_4 and X_5 are expressed in calibers and X_2 in degrees) 68
4.3 Robustness of Algorithm 2 w.r.t β 76
4.4 Comparison between the different alternatives 77

List of Algorithms

1 Bayesian optimization algorithm 30
2 MLP model improvement for optimization 36
3 Iterative model identification and controller design procedure 86

Notations

Abbreviations

BO	Bayesian Optimization
GP	Gaussian Process
MLP	Multi Layer Perceptron
NN	Neural Network
NNSGO	Neural Network Smart Grid Based Optimization
CFD	Computational Fluid Dynamics
PRODAS	Projectile Rocket Ordnance Design & Analysis System
RMSE	Root Mean Squared Error
RBF	Radial Basis Function
UCB	Upper Confidence Bound
PI	Probability of Improvement
EI	Expected Improvement
w.r.t	with respect to

Aerodynamic moments and Forces

F_A	Axial force
F_D	Drag force
F_N	Normal force
F_L	Lift force
M	Pitching moment

Aerodynamic coefficients and Stability Derivatives

C_A	Axial force coefficient
C_D	Drag coefficient
C_N	Normal force coefficient
C_L	Lift coefficients
C_m	Pitch moment coefficient
C_{A_0},C_{A_2}	First and Second order Axial force stability derivatives
C_{D_0},C_{D_2}	First and Second order drag force stability derivatives
$C_{N\alpha}$	Normal force coefficient slope
$C_{L\alpha}$	Lift force coefficient slope
$C_{m\alpha}$	Pitch moment coefficient slope or overturning coefficient
C_{mq}	Pitch damping coefficient

Résumé en français

Dans de nombreuses applications techniques, l'objectif est de déterminer les performances optimales d'un système. Pour évaluer ces performances, il est nécessaire de mesurer la sortie du système pour une entrée donnée. Cependant, pour de nombreux scénarios concrets, cela représente une tâche chronophage ou coûteuse car elle peut nécessiter des ressources importantes. Par conséquent, si le processus d'optimisation nécessite plusieurs évaluations sur le système pour trouver l'optimum, celui-ci devient également coûteux à réaliser.

Même si un système est cher à évaluer, il est possible d'accéder à des données collectées lors d'expériences réalisées sur ce système ou de recueillir des données supplémentaires. De plus, grâce aux récents progrès en matière de capacités de calcul, nous pouvons gérer, analyser et stocker efficacement de grandes quantités de données. La possibilité de collecter des données, combinée aux ressources informatiques disponibles, permet d'identifier des modèles basés sur les données, remplaçant les systèmes complexes et coûteux à évaluer. Ces modèles sont moins coûteux et plus rapides à évaluer que les systèmes, rendant le processus d'optimisation plus efficace.

Le thème principal de cette thèse concerne l'optimisation des performances d'un système à partir des modèles identifiés.

Selon l'application, les performances d'un système peuvent être exprimées de différentes manières. Cette thèse est réalisée en collaboration avec l'**Institut Franco-Allemand de Recherches de Saint-Louis (ISL)**, au sein du groupe spécialisé dans les domaines de **l' Aérodynamique et Balistique Extérieure (ABX)**. La première application de cette thèse est liée à ces deux domaines. L'aérodynamique concerne l'étude du mouvement de l'air lorsqu'il interagit avec des objets solides en mouvement, tandis que la balistique extérieure se concentre sur le comportement d'un véhicule lorsqu'il est en vol libre, c'est-à-dire, lorsqu'il n'est plus perturbé par l'arme à partir de laquelle il a été lancé (McCoy, 1999). Dans ces domaines, nous nous intéressons fréquemment aux coefficients aérodynamiques (McCoy, 1999; Zipfel, 2000). Ces derniers sont des coefficients statiques associés aux forces et aux moments des véhicules en vol. Ils dépendent de la géométrie du véhicule et de ses conditions de vol. Ces coefficients sont très importants car ils permettent de caractériser le comportement du véhicule pendant son vol afin de savoir s'il est, par exemple, stable. L'un des principaux sujets de recherche dans les domaines de l'aérodynamique et de la balistique est la conception (ou l'optimisation) de véhicules selon des critères associés à ces coefficients aérodynamiques. Ces types d'études sont souvent appelés conception aérodynamique ou optimisation aérodynamique (Martins, 2022). Au sein du groupe **ABX** de l'**ISL**, les types de véhicules étudiés sont des projectiles tirés depuis des lanceurs de laboratoire, et l'un de leurs objectifs est de trouver des conceptions optimales pour ces munitions afin d'accomplir des missions spécifiques. Dans cette thèse, l'un des objectifs sera d'effectuer cette tâche, dans le cas où la munition est un projectile équipé d'ailettes (voir Figure 1).



Figure 1: Basic Finner, un exemple classique d'un projectile équipé d'ailettes, dans la même famille que les projectiles étudiés dans cette thèse.

Dans la conception aérodynamique des projectiles, les performances dépendent de la configuration du projectile et sont fonction des coefficients aérodynamiques. Par conséquent, pour réaliser une telle tâche, il est nécessaire d'avoir accès aux coefficients aérodynamiques associés à différentes configurations de projectiles dans différentes conditions de vol. Cependant, il n'existe pas de modèles physiques permettant d'obtenir ces coefficients directement en fonction de la configuration d'un projectile et des conditions de vol souhaitées. La principale méthode pour les estimer repose sur des essais expérimentaux (Albisser, 2015) ou des simulations numériques, qui sont coûteuses et chronophages. Par consequent, la conception aérodynamique est également généralement coûteuse, car elle nécessite d'évaluer les coefficients pour de multiples configurations. L'un des principaux défis dans le domaine de l'aérodynamique est de développer de nouvelles méthodes pour estimer les coefficients aérodynamiques rapidement et avec précision. Une façon d'y parvenir est d'utiliser des bases de données provenant d'expériences passées, pour construire des modèles basés sur les données, capables d'estimer les coefficients en fonction de la configuration du véhicule et de ses conditions de vol. Ce type d'étude sera également l'un des objectifs de cette thèse dans le cas où les véhicules sont des projectiles équipés d'ailettes comme celui présenté dans la Figure 1. Si un modèle prédictif des coefficients est disponible et est peu coûteux en temps de calcul, alors le problème de conception aérodynamique pourra être résolu plus rapidement et à moindre coût.

L'application décrite ci-dessus concerne des systèmes statiques. Une deuxième application est étudiée dans cette thèse et porte sur des systèmes dynamiques dans le domaine du contrôle de systèmes. Dans de telles applications, un des objectifs est de concevoir des regulateurs qui maximisent ou minimisent un certain critère de performance. Ce dernier dépendra du regulateur conçu et sera exprimé par rapport au système dynamique réel. Cependant, dans certains cas d'applications, le système réel est inconnu ou coûteux à évaluer, rendant difficile l'évaluation des performances d'un regulateur donné et, par conséquent, la conception d'un regulateur adapté. Comme dans la première application, on pourrait utiliser les données collectées sur ce système ou effectuer une expérience pour recueillir des données, puis utiliser ces données pour identifier un modèle du système réel. Ce modèle pourrait ensuite être utilisé à la place du système réel dans l'expression du critère de performance et ainsi permettre de concevoir le regulateur.

Dans les deux applications présentées dans les paragraphes précédents, les performances obtenues dépendent de la qualité du modèle et plus particulièrement des données avec lesquelles il a été identifié. Si le modèle n'a pas été identifié avec des données adéquates, il ne sera pas une fidèle représentation du vrai système et ne permettra pas d'atteindre les performances optimales

souhaitées. Pour obtenir de meilleures performances, le modèle doit être amélioré en le réidentifiant avec de nouvelles données. Étant donné que les systèmes peuvent être coûteux à évaluer, les données choisies pour améliorer les modèles doivent être sélectionnées avec soin.

L'objectif de cette thèse est de développer des procédures pour améliorer un modèle identifié, employé dans l'optimisation des performances d'un système, tout en tenant compte des coûts liés à l'acquisition des données.

Dans des problèmes tels que la conception aérodynamique, où le but est de trouver l'optimum d'une fonction statique, les données utilisées pour améliorer un modèle à des fins d'optimisation sont généralement choisies sur la base d'un compromis entre deux objectifs connus sous les noms d'exploration et d'exploitation. L'objectif d'exploitation vise à identifier les points où le modèle est optimal, tandis que l'objectif d'exploration cherche à trouver des points dans des régions peu couvertes par le jeu de données. L'approche principale permettant de réaliser ce compromis est connue sous le nom d'Optimisation Bayésienne (Frazier, 2018). En Optimisation Bayésienne, la fonction à optimiser, généralement coûteuse à évaluer, est modélisée à l'aide d'un Processus Gaussien (Rasmussen, 2003). Ses prédictions et les incertitudes associées sont ensuite utilisées pour déterminer itérativement les candidats les plus prometteurs à évaluer afin de se rapprocher de l'optimum. En dehors de la conception aérodynamique, l'Optimisation Bayésienne a également été appliquée dans de nombreux domaines, tels que l'automatique pour la conception de regulateurs (Guzman et al., 2020), (Bombois and Forgione, 2022), ou la surveillance environnementale (Marchant and Ramos, 2012), et a prouvé son efficacité pour ces différents problèmes. Une possible limitation de cette approche réside dans le fait que les modèles de Processus Gaussiens ont tendance à mal gérer des quantités de données importantes. En revanche, d'autres modèles d'apprentissage automatique, comme les Réseaux de Neurones, sont mieux adaptés pour traiter de grands jeux de données.

Notre première contribution dans cette thèse consistera à illustrer comment les Réseaux de Neurones peuvent être utilisés dans des procédures similaires à l'Optimisation Bayésienne afin d'effectuer l'optimisation de fonctions coûteuses à évaluer. Plus particulièrement, nous développons une procédure qui peut être utilisée pour sélectionner les candidats les plus prometteurs afin d'améliorer le modèle de Réseau de Neurones dans un but d'optimisation, en se basant sur le compromis entre les objectifs d'exploration et d'exploitation. Il sera montré que la procédure peut fournir des résultats compétitifs par rapport à l'Optimisation Bayésienne et qu'elle peut être bénéfique dans certains cas.

Certaines procédures ont également été proposées pour améliorer un modèle de système dynamique utilisé pour concevoir un regulateur qui maximise un certain critère de performance. La procédure de base est peu coûteuse et consiste à utiliser des données déjà disponibles pour identifier un nouveau modèle sans avoir à effectuer d'expériences supplémentaires sur le système réel. En effet, chaque fois qu'un regulateur est conçu, il est testé sur le système réel pour évaluer ses performances. Même si les performances obtenues ne sont pas suffisantes, cela permet de collecter de nouvelles données. Ces nouvelles données, ainsi que les données passées, peuvent ainsi être utilisées pour identifier un nouveau modèle, puis re-concevoir le regulateur à partir de de ce nouveau modèle. De telles approches sont connues sous le nom d'identification de modèle et de conception de regulateur successive dans la littérature. Ils ont connu un développement considérable pour les systèmes linéaires au cours des dernières années comme proposé dans (Van Den Hof and Schrama, 1995), (Gevers, 2002) et (Forgione et al., 2015).

Notre deuxième contribution dans cette thèse consistera à montrer comment de tels résultats peuvent être étendus aux systèmes dynamiques non linéaires afin de concevoir un regulateur qui fournit l'action de commande optimale pour le suivi d'une famille de signaux de référence. Plus particulièrement, nous élargirons les résultats pour le cas où le regulateur est conçu dans un cadre de commande en boucle ouverte et pour une classe spéciale de systèmes non linéaires connus sous le nom de systèmes de Wiener, consistant en une interconnexion d'un bloc dynamique linéaire suivi par une non-linéarité statique (Giri and Bai, 2010). Il sera montré que de telles procédures d'identification de modèle et de conception de regulateur itérative permettent d'obtenir des performances de suivi efficaces pour les systèmes non linéaires étudiés.

Pour résumer, l'objectif de cette thèse est de répondre à la problématique suivante :

Problématique :

Comment atteindre des performances optimales d'un système à l'aide d'un modèle identifié et comment améliorer ce modèle si nécessaire, tout en tenant compte du coût lié à l'acquisition des données?

Afin de répondre à cette problématique, nous avons identifié deux axes de recherche étroitement liés :

- **Axe de Recherche 1** : Comment améliorer un modèle utilisé pour trouver l'optimum d'une fonction statique coûteuse à évaluer? Cet axe de recherche est appliqué à l'optimisation aérodynamique des projectiles.

- **Axe de Recherche 2** : Comment concevoir une commande prédictive pour un système non linéaire de type Wiener basé sur un modèle identifié de ce dernier, et comment améliorer ce modèle si le regulateur initialement conçu ne donne pas de performances satisfaisantes ?

Les résultats du premier axe de recherche sont détaillés des Chapitres 1 à 4 :

- **Chapitre 1** : Dans ce chapitre, nous discutons de différents modèles d'apprentissage automatique qui peuvent être utilisés pour identifier des systèmes statiques. Une attention particulière est portée sur les Réseaux de Neurones et les Processus Gaussiens, qui sont utilisés dans la suite de la thèse.

- **Chapitre 2** : Ensuite, nous discutons de la manière dont les modèles présentés au Chapitre 1 peuvent être utilisés pour trouver l'optimum d'une fonction coûteuse. Plus particulièrement, nous présentons des approches permettant de sélectionner les données à utiliser pour améliorer leurs performances dans un objectif d'optimisation. La première approche est

l'Optimisation Bayésienne, et la seconde implique une procédure similaire, mais où des Réseaux de Neurones sont utilisés à la place des Processus Gaussiens.

- **Chapitre 3** : Dans ce chapitre, nous présentons les coefficients aérodynamiques et construisons des modèles capables de les estimer. Nous exposons différentes approches pour y parvenir en fonction de la source et de la fidélité des données.

- **Chapitre 4** : Ce chapitre est dédié au problème de conception aérodynamique. Nous utilisons les éléments présentés dans les trois premiers chapitres pour résoudre le problème de recherche des dimensions optimales d'un projectile. Nous proposons également une comparaison de différentes approches pouvant être utilisées pour résoudre un tel problème.

Les résultats du deuxième axe de recherche sont détaillés dans le Chapitre 5 :

- **Chapitre 5** : Ce chapitre présente une procédure itérative d'identification et de conception de regulateurs, visant à concevoir un regulateur pour un système non linéaire à partir d'un modèle identifié du vrai système. Cette procédure est appliquée à un schéma de contrôle en boucle ouverte, où il est supposé que le système réel a une structure de type Wiener.

Introduction

In many engineering applications, the objective is to find the optimal performances of a system. To evaluate the performances, it is necessary to measure the output of the system for a given input. However, for many real life processes accomplishing this is an expensive task, *i.e.*, it requires significant resources or is time-consuming. Therefore, if the optimization process necessitates multiple evaluations of the system to find the optimum, it will be expensive to carry out as well.

Even if a system is expensive to evaluate, it remains possible to access data collected from past experiments on the system or to gather additional data. Moreover, thanks to recent advancements in computational capabilities, we can efficiently manage, analyze, and store vast amounts of that data. The ability to collect data combined with the available computational resources allows identifying data-driven models that replace the complex and expensive to evaluate systems. These models will be cheaper and faster to evaluate than the expensive systems, making it possible to achieve the optimal performances more efficiently.

The main theme of this thesis concerns the optimization of system performances based on identified models.

Depending on the application, the performances of a system can be expressed in different ways. This thesis is conducted in collaboration with the **French-German Research Institute of Saint-Louis (ISL)**, within the group which is specialized in the fields of **Aerodynamic and Exterior Ballistics (ABX)**. The first application in this thesis is related to these two domains. Aerodynamic concerns the study of the motion of air as it interacts with solid objects in motion while Exterior Ballistics focuses on the behaviour of a flying vehicle when it is in free flight, *i.e.*, it is no longer perturbed by the gun it is launched from (McCoy, 1999). In these domains, we are frequently interested in the so-called aerodynamic coefficients (McCoy, 1999; Zipfel, 2000). The latter are static coefficients which are associated to the forces and moments of vehicles in flight. They depend on the geometry of the vehicle and its flight conditions. These coefficients are very essential because they allow characterising the behaviour of the vehicle during its flight to, *e.g.*, know if it is stable. One of the main research topics in the fields of aerodynamic and ballistics, is the design (or optimization) of vehicles based on criteria associated with these aerodynamic coefficients. These types of studies are often called aerodynamic design or aerodynamic optimization (Martins, 2022). In the **ABX** group at **ISL**, the types of vehicles which are studied are fired ammunitions and one of their goals is to find optimal designs for them in order to accomplish specific missions. In this thesis, one of the objectives will be to perform such tasks, for the case where the ammunition is a projectile fitted with fins (see Figure 1).



Figure 1: Basic Finner, a classic example of a projectile fitted with fins in the same family of projectiles as the one studied in this thesis

In aerodynamic design of projectiles, the performance will depend on the configuration of the projectile and will be a function of the aerodynamic coefficients. Therefore, to perform such a task, one needs to have access to the aerodynamic coefficients associated with different configurations of projectiles in different flight conditions. However, there are no readily-available physical models that provide these coefficients based on the configuration of a projectile and the desired flight condition. The main way to estimate them is through expensive and time consuming experiments (Albisser, 2015) or numerical simulations. As a result, performing aerodynamic design is also usually costly as it requires evaluating the coefficients for multiple configurations. One of the main challenges in the field of aerodynamics is in developing new methods to estimate aerodynamic coefficients fast and accurately. A way to achieve this, is to use data bases from past experiments, to build data-driven models which estimate the coefficients based on the configuration of the vehicle and its flight condition. This type of study will also be one of the objectives in this thesis for the case where the vehicles are projectiles fitted with fins as the one described in Figure 1. If an inexpensive and accurate model which predicts the aerodynamic coefficients is available, then the aerodynamic design problem can be solved faster and cheaper.

The application described above concerns static systems. A second application is studied in this thesis and pertains to dynamic systems in the framework of control engineering. In such applications, the interest is on designing controllers which maximize/minimize a certain performance criterion. The latter will depend on the designed controller and expressed with respect to the true dynamic system. However, it is often the case that the true system is unknown or expensive to evaluate making it difficult to assess the performances of a given controller, and therefore to design a suitable one. As in the first application, one could use data collected from this system, or perform an experiment on it to collect data, then use these data to identify a model for the true system. This model can then be employed instead of the true system in the expression of the performance criterion and thus allow designing the controller.

In the two applications presented in the previous paragraphs, the performances obtained will depend on the quality of the model and more particularly with the data it has been identified with. If the model has not been identified with suitable data then it will not be an accurate representation of the true system and may not allow achieving the desired optimal performances. To obtain better performances, the model must be improved by re-identifying it with new data. Since the systems may be expensive to evaluate, the data chosen to increase the accuracy of the models must be chosen with care.

The goal of this thesis is to develop procedures to improve an identified model that is employed

to optimize the system performances, while accounting for the cost related to data acquisition.

In problems such as aerodynamic design, where the goal is to find the optimum of a static function, the data employed to improve a model for optimization purposes is usually chosen based on a trade-off between two objectives known as exploration and exploitation. The exploitation objective seeks to find points where the model is optimum whereas the exploration objective seeks to find points in regions not well covered by the dataset. The main approach which accomplishes the trade-off between these two objectives is known as Bayesian Optimization (Frazier, 2018). In Bayesian Optimization, the to-be optimized and typically expensive to evaluate function is modelled using a Gaussian Process (Rasmussen, 2003). Its predictions and the associated uncertainties are then used to iteratively determine the most promising candidates to evaluate in order to get close to the optimum. Other than aerodynamic design, Bayesian Optimization has also been applied in many domains, such as control engineering for controller design (Guzman et al., 2020), (Bombois and Forgione, 2022), or environmental monitoring (Marchant and Ramos, 2012) and has proven to provide efficient results. A possible limitation of this approach is that the Gaussian Process models tend to scale poorly when the number of samples in the dataset increases. In contrast, other machine learning models, such as Neural Networks, are better suited to handling large datasets.

Our first contribution in this thesis will be to illustrate how Neural Networks can be used in procedures similar to Bayesian Optimization in order to perform the optimization of functions which are expensive to evaluate. More particularly, we develop a procedure which can be used to select the most promising candidates in order to improve the Neural Network model for the optimization purpose, based on the trade-off between the exploration and exploitation objectives. It will be shown that the procedure can provide competitive results to Bayesian Optimization and that it can be beneficial in some cases.

Some procedures have also been proposed to improve a model of a dynamic system used to design a controller that maximises a certain performance criterion. The basic procedure is a costless one, involving the use of already available data to identify a new model without having to perform extra experiments on the true system. Indeed, every time a controller is designed, it is tested on the true system to evaluate its performances. Even if the obtained performances are not good enough, it allows collecting some new data. These new data as well as the past data, can thus be used to identify a new model then re-design the controller based on the new model. Such approaches are known as iterative model identification and controller design in the literature and have been well developed for linear systems in the past years, see *e.g.*, (Van Den Hof and Schrama, 1995), (Gevers, 2002) and (Forgione et al., 2015).

Our second contribution in this thesis will be to show how such results can be extended to nonlinear dynamic systems in order to design a controller which provides the optimal control action for the tracking of a given class of reference signals. More particularly, we will extend the results for the case where the controller is designed in a feedforward control framework and for a special class of nonlinear systems known as Wiener, *i.e.*, interconnection of linear dynamic block followed by a static nonlinearity (Giri and Bai, 2010). It will be shown that such iterative model identification and controller design procedures allow obtaining efficient tracking performances for the studied nonlinear systems.

To summarize, the goal of this thesis is to answer to the following problematic:

> **Problematic:**
>
> How to achieve optimal system performance using an identified model and how to improve the model if necessary, while accounting for the cost related to data acquisition.

In order to address this problematic, we have identified two strongly related research topics:

- **Research Topic 1:** How to improve a model being used to find the optimum of an expensive to evaluate static function? This research topic is applied to the aerodynamic optimization of projectiles.

- **Research Topic 2:** How to design a feedforward controller for a nonlinear Wiener system based on an identified model of the latter, and how to improve this model if the initially designed controller does not perform well?

The results of the first research line are detailed from Chapter 1 to Chapter 4.

- **Chapter 1:** In this chapter, we discuss different machine learning models which can be used to identify static systems. A focus is done on Neural Networks and Gaussian Processes which are used in the sequel of the thesis.

- **Chapter 2:** Afterwards, we discuss how the models presented in Chapter 1, can be used to find the optimum of an expensive function. More particularly, we present approaches which can be used to select the data to use in order to improve their performance for the optimization purpose. The first approach is Bayesian Optimization and the second one involves a procedure similar to the latter but where Neural Networks are used instead of Gaussian Processes.

- **Chapter 3:** In this chapter, we present aerodynamic coefficients, and build models which are capable of estimating them. We present different approaches to achieve this depending on the source and fidelity of the data.

- **Chapter 4:** This chapter is dedicated to the aerodynamic design problem. We use the elements presented in the first three chapters, to solve the problem of finding the optimal dimensions of a projectile. We also offer a comparison of different approaches which can be used to solve such a problem.

The results of the second research line are detailed in Chapter 5.

- **Chapter 5:** This chapter presents an iterative identification and controller design procedure, to build a controller for a nonlinear system based on the identified model of the true system. It is applied to an open-loop feedforward control scheme, where it is assumed that the true system has a Wiener structure.

List of Publications

International Conferences

- Uwadukunze, A., Bombois, X., Gilson, M., Albisser, M. (2024, June). Neural networks smart grid based optimization for expensive functions. In *2024 European Control Conference (ECC)* (pp. 2736-2741). IEEE.

  This paper covers the procedure presented in Section 2.3 of Chapter 2.

- Uwadukunze, A., Bombois, X., Forgione, M., Gilson, M., Albisser, M. (2024, September). Identification for Feedforward Control of Wiener Systems. In *4th IFAC Conference of Modelling, Identification and Control of Nonlinear Systems*. IFAC-PapersOnLine, 58(21), 208-213.

  This paper covers the results of the second research topic presented in Chapter 5.

Poster Presentations

- Uwadukunze, A., Bombois, X., Gilson, M., Albisser, M. (2022, September). Smart grid search for black-box optimisation with neural networks as surrogate models. European Research Network on System Identification (ERNSI) - Leuven, Belgium.

- Uwadukunze, A., Bombois, X., Gilson, M., Albisser, M. (2022, September). Artificial intelligence for aerodynamic predictions and conceptual geometry design. ISL Scientific Symposium - Saint-Louis, France.

- Uwadukunze, A., Bombois, X., Gilson, M., Albisser, M. (2023, February). Artificial intelligence for aerodynamic predictions and conceptual geometry design. Journée Scientifique autour de l'IA - Nancy, France.

Oral Presentations

- Uwadukunze, A., Albisser, M., Gilson, M., Bombois, X. (2022, May). Artificial Intelligence Techniques applied to Aerodynamics and Ballistics. CTIdentif SAGIP - Paris, France

- Uwadukunze, A., Albisser, M., Bombois, X., Gilson, M., (2023, September). Aerodynamic coefficients prediction and design optimization for projectiles using artificial intelligence - ISL Scientific Symposium - Saint-Louis, France.

- Uwadukunze, A., Bombois, X., Gilson, M., Albisser, M. (2024, May). Identification for feedforward control of Wiener systems. CTIdentif (SAGIP) - Lyon, France.

- Uwadukunze, A., Albisser, M., Bombois, X., Gilson, M., (2024, September). Prediction of aerodynamic coefficients for projectiles : A multi fidelity learning approach - ISL Scientific Symposium - Saint-Louis, France.

Chapter 1

Data-driven models for static system identification

To effectively use a data-driven model for optimization, it is crucial to select and identify one which accurately represents the to-be optimized function. Numerous types of data-driven models have been proposed in the literature for this purpose. The aim of this chapter is to provide a concise overview of the existing ones, the principles behind them and their advantages. A focus is done on two of the most widely used models in the machine learning community which are Neural Networks and Gaussian Processes. These two models will be essential in the sequel for optimization purposes. The chapter is organised as follows: first a brief review of existing data-driven models is provided in Section 1.1. Secondly, we explain how Neural Networks, and more particularly Multi Layer Perceptrons, function and how they can be identified in Section 1.2. We then explain the idea behind Gaussian Processes, how they provide estimates and how their hyper-parameters are optimized in Section 1.3. Finally, we present two metrics which will be used to evaluate the performances of these models.

Contents

1.1 Review of possible surrogate models 15
1.2 Neural Networks for regression 17
1.2.1 Multi Layer Perceptrons (MLP) parametrization 17
1.2.2 Identification of a MLP 19
1.3 Gaussian Processes for regression 21
1.3.1 Single Output Gaussian Processes 21
1.3.2 Multiple correlated Output Gaussian Processes 23
1.4 Performance metrics 24
1.5 Conclusion 24

1.1 Review of possible surrogate models

We are interested in estimating a data-driven model of a static mapping F between a vector of inputs $X \in \mathbb{R}^{D_X}$ of dimension D_X and the corresponding outputs $Y \in \mathbb{R}^{D_Y}$ of dimension D_Y, from a set of observations $\mathcal{D} = \{X_i, Y_i\}_{i=1}^{N}$. In this chapter, it is considered that this data is noise-free. In the sequel, we give a brief review of existing data-driven models which allow having

estimates of the function F.

The most basic approach to estimate such functions is from the use of a polynomial approximation which can take the following form for, *e.g.*, a single input single output system:

$$y = a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1} + a_n x^n, \tag{1.1}$$

where n is the degree of the polynomial and the coefficients $a_i \in \mathbb{R}$, $\{i = 0, ..., n\}$, are the parameters of the polynomial. Polynomials are universal approximators and can be used to estimate any function by increasing the degree n. However, as the degree of the function increases, they may be prone to over-fitting, *i.e.*, the polynomial model is perfectly adapted to the data set used to identify it but is unable to generalize well to new, unseen data. The machine learning community has provided other data-driven models which can be used to identify static functions. These models can be divided into different categories. In this chapter, we are mainly interested on whether they are probabilistic on non-probabilistic.

Non-probabilistic models

We begin by discussing non-probabilistic models, which are also the most commonly found in the literature. Such models do not provide a direct measure of the estimates uncertainty.
Among non-probabilistic models, we can make a clear distinction between non-parametric models and parametric ones.

In non-parametric models, the predictions are not based on a set of parameters identified from the data $\mathcal{D} = \{X_i, Y_i\}_{i=1}^N$. Rather, they are made by directly using the available data to infer outcomes for new inputs. The simplest illustration is with *k-nearest neighbours* (KNN) (Kramer, 2011) where for a new input, KNN identifies the k training examples closest to the input and assigns the average of their values to the new input.

In the class of non-parametric models, we can also find kernel based regression models such as *Kernel Ridge Regression* (Welling, 2013) or *Support Vector Machines* (Hearst et al., 1998). These methods are based on the "*Kernel Trick*" where kernel functions, which compute the similarity between two points X and X', are used to make predictions. Regardless of the effectiveness of kernel based models, choosing the appropriate kernel for them may be a complex task and kernel functions may be difficult to handle when the number of samples increases. Other examples of non-parametric models include *decision trees* (Loh, 2011) and *random forests* (Biau and Scornet, 2016).

Unlike non-parametric models, parametric ones are based on a finite set of parameters identified from the data $\mathcal{D} = \{X_i, Y_i\}_{i=1}^N$. The simplest example of a parametric model is the polynomial model (1.1). Other commonly used parametric models are Neural Networks. They have been successfully used for many different tasks such as Natural Language Processing (Goldberg, 2016), Pattern recognition (Bishop, 1995), etc. The most basic type of a Neural Network is a Multi Layer Perceptron as described in Section 1.2. However several other types of Neural Networks such as *e.g. Convolutional Neural Networks* (Gu et al., 2018), which are designed to process grid-like data (*e.g.*, images), have been developed for different tasks.

Parametric models are usually easier to interpret than non-parametric models. However if the parametrization is not good enough, then the underlying function will not be well explained by the model. One could choose a strategy of over-parametrization, *i.e.*, choosing parametric models with large degrees of liberty, but this will require a large amount of data to identify each parameter as well as large computational resources. Note that some architectures of Neural Networks can have more than million of parameters, *e.g.*, *Resnet* architecture used for image classification. In such cases, they may be considered as non-parametric as the large number of parameters reduces their interpretation.

Probabilistic models

Few models in the literature are probabilistic and provide as a result uncertainty measures on the estimates. Gaussian Process regression models are a primary example and will be discussed in Section 1.3.1. Neural Networks have also been extended to provide uncertainties using *Bayesian Neural Networks* (Goan and Fookes, 2020). In such networks some or all of the parameters are treated as distributions, providing uncertainty in parameters and predictions. However, it has to be noted that even if Bayesian Neural Networks allow providing an uncertainty measure, learning the distribution of each parameter can largely increase the computational cost compared to learning classic Neural Networks.

In summary, several approaches are available to estimate a static function from data. The choice will depend on the quantity of data available, the model preference (parametric or non-parametric), and the need for uncertainty measures. The following sections will delve deeper into two widely used models which are relevant to the sequel of the thesis: Neural Networks (Multi Layer Perceptrons) and Gaussian Processes.

1.2 Neural Networks for regression

Many different types of Neural Networks have been developed and they remain one of the key elements in machine learning and deep learning. In this thesis, the focus is made on a vanilla type of Neural Network called the Multi Layer Perceptron (MLP), which are known to excel in modelling complex and potentially high dimensional functions. Such models are also known to handle well large data sets.

1.2.1 Multi Layer Perceptrons (MLP) parametrization

Multi layer perceptrons are parametric models, where the to-be identified parameters are known as weights and biases. They are composed of three main parts: the input layer, the hidden layers and the output layer.

Each hidden layer is composed of multiple *neurons*, which can be seen as computational units that apply a transformation to the input they receive and provide an output which is transferred as input to the neurons of the next layers. Each neuron ν_k is associated to a vector of weights w_k and a scaler bias term b_k. The length of the vector w_k is the same as the one of the input vector. For a given input vector $u = [u_1, ..., u_{n_u}]^T$, the output of a neuron ν_k is then expressed as:

$$\nu_k(u) = \Phi\left(b_k + \sum_{i=1}^{n_u} w_{ki} u_i\right), \tag{1.2}$$

where Φ is the so-called activation function. Several types of activation functions have been developed in the literature. Two of them which are frequently used for regression tasks in MLPs are:

- the Rectified Linear Unit $\boldsymbol{ReLU}$: $\Phi(x) = \max(0, x)$;
- the hyperbolic tan function $\boldsymbol{tanh}$: $\Phi(x) = tanh(x)$.

The simplest architecture of a MLP is made of only one hidden layer as depicted in Figure 1.1.

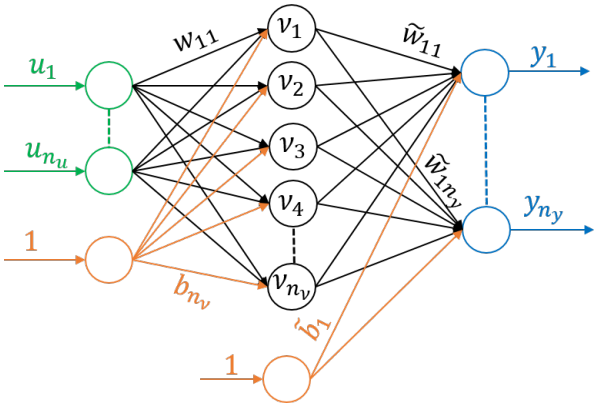


Figure 1.1: MLP with only one hidden layer

Each component y_j of its output vector $y = [y_1, ..., y_{n_y}]^T$ for a given input vector u is expressed as:

$$y_j = \sum_{k=1}^{n_\nu} \tilde{w}_{jk} \nu_k(u) + \tilde{b}_j, \tag{1.3}$$

where n_ν is the number of neurons in the hidden layer and $\tilde{w}_{jk}$ and $\tilde{b}_j$ are the weights vector and bias connected to the outputs.

For a MLP with multiple hidden layers n_l, the output of a neuron k in a layer l, ν_k^l, is given w.r.t the outputs of the neurons in the layer $l-1$ as:

$$\nu_k^l(u) = \Phi\left(b_k^l + \sum_{i=1}^{n_\nu^{l-1}} w_{ki}^l \nu_i^{l-1}(u)\right), \tag{1.4}$$

b_k^l and w_{ki}^l are the biases and weights associated to a neuron k in the layer l of the MLP. The outputs of each neuron are then fed to the neurons of the next layer as in (1.4) until we reach the final layer n_l. The final output of the neuron is then calculated using (1.3) by replacing the input vector u by the outputs of the neurons in the final layer:

$$y_j(u) = \sum_{i=1}^{n_\nu^{n_l}} \tilde{w}_{ji} \nu_i^{n_l}(u) + \tilde{b}_j. \tag{1.5}$$

In the sequel, we consider that all the parameters of the MLP, *i.e.*, the weights and biases are stacked in a parameter vector θ and we discuss the procedure of identifying them.

1.2.2 Identification of a MLP

We are interested in estimating a function $F(X)$ from a dataset of inputs $X \in \mathbb{R}^{D_X}$ associated to outputs $Y \in \mathbb{R}^{D_Y}$, such that $\mathcal{D} = \{X_i, Y_i\}_{i=1}^N$, with N being the number of samples. The identification of a MLP model for the function $F(X)$, requires to perform the following steps:

a. Data preprocessing

b. Model structure and hyper-parameter selection

c. Identification of the model parameters

d. Model validation (see Section 1.4).

a. Data preprocessing

Before identifying a model for $F(X)$, some preprocessing can be performed on the data to improve the performance of the model. Initially, after collecting or measuring the data, it is prepared to address missing values, remove duplicates and correct some inconsistencies. The preprocessing then usually consists in diving the data into different sets and then applying some transformations on it.

First, the dataset is usually divided into 3 sets:

- **Training set** $\simeq 75\%$ of the dataset: This set is used to find the parameters θ which minimize a loss function (defined in the sequel);
- **Validation set** $\simeq 10\%$ of the dataset: This set is used to validate the choice of the structure of the model, see Section 1.2.2;
- **Test set** $\simeq 15\%$ of the dataset: This set is used to quantify the performance of the network on an unseen test set.

When the size N of the dataset is small, the division above is usually not used because there may not be enough data to train the MLP. However, some approaches such as **cross-validation** (Browne, 2000) have been developed and work efficiently for such cases. Ideally, the size of the dataset should be large enough to ensure an efficient identification of the parameters θ, especially when the number of parameters is large, *i.e.*, for MLPs with multiple layers and multiple neurons.

The other important pre-processing to perform on the dataset before identifying the function $F(X)$, is data scaling or normalization. This has proven to improve the performances of several machine learning models including Neural Networks (Sola and Sevilla, 1997). For MLPs, if the different outputs or inputs have different levels of magnitude, the optimization algorithm can converge slowly or get stuck in local minima because it will be dominated by the features with larger ranges.

There are several ways to scale the data. Two approaches which are frequently used are the **Standardization** and the **MinMax scaler**. The **Standardization** transforms the data to have a mean of 0 and a standard deviation of 1. For a vector x: $x = [x_1, ..., x_N]^T$. Each component x_i is normalized as:

$$x'_i = \frac{x_i - \mu(x)}{\sigma(x)}, \tag{1.6}$$

where $\mu(x)$ and $\sigma(x)$ are the mean and standard deviation of the vector x.

The **MinMax scaler** scales the data to a fixed range, usually $[0, 1]$. It is defined as follows:

$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{1.7}$$

b. Model Structure Selection

After the data pre-processing, we need to choose the structure (or hyper-parameters) for the MLP. For such models, the choice is done from, *e.g.*, the number of layers, the number of neurons in each layer or the activation functions. The structure of the MLP is usually chosen as the simplest possible but that allows explaining the data. This can be accomplished using, *e.g.*, grid search or random search (Liashchynskyi and Liashchynskyi, 2019) approaches to find the structure which explains the validation data.

c. Identification of the model parameters

After choosing a structure for the MLP, the next step consists in identifying its parameters using the training set. The identification criterion for MLPs, commonly known as loss function is usually a Mean Squared Error (MSE). Let $F(X_i; \theta)$ denote the predictions of the MLP with a parameter vector θ for the input data X_i. The MSE is then defined as:

$$MSE(\theta) = \frac{1}{N} \sum_{i=1}^{N} \parallel Y_i - F(X_i; \theta) \parallel^2, \tag{1.8}$$

where $\parallel A \parallel = \sqrt{A^T A}$, represents the Euclidean norm of a vector A, and Y_i the target output corresponding to the input X_i. The identification of the Neural Network consists in finding the parameters θ that minimize (1.8):

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \parallel Y_i - F(X_i; \theta) \parallel^2 . \tag{1.9}$$

The parameters $\hat{\theta}$ that minimize the loss function, are usually found using *back-propagation* (Amari, 1993) combined with a gradient-descent algorithm. Two of the most common optimization algorithms used to find the optimal parameters are Stochastic gradient descent (Amari, 1993) and Adam Optimizer (Kingma and Ba, 2014).

Note that the criterion (1.8) can also be used in the model structure selection using the validation set.

1.3 Gaussian Processes for regression

1.3.1 Single Output Gaussian Processes

Gaussian Processes (GP) are non-parametric machine learning tools used for regression and classification tasks. In this thesis, we are interested in the task of regression, where the goal is to build an estimate of a continuous function $f(X) = y$ from a set of observations $\mathcal{D} = \{X_i, y_i\}_{i=1}^{N}$, with N the number of samples. We will first consider the case where $y \in \mathbb{R}$ is a scalar output and $X \in \mathbb{R}^{D_X}$. The Gaussian Process description presented in this section is based on the book of (Rasmussen, 2003).

A **Gaussian Process** is defined as a collection of random variables such that a finite number of those variables have a joint Gaussian distribution (Rasmussen, 2003). In Gaussian Processes, the value of $f(X)$ is considered as a random variable specified by its mean $\mu(X)$ and covariance $k(X, X')$, with:

$$\mu(X) = \mathbb{E}[f(X)] \tag{1.10a}$$
$$k(X, X') = \mathbb{E}[(f(X) - \mu(X))(f(X') - \mu(X'))], \tag{1.10b}$$

with $\mathbb{E}$ the expectation operator. The function $f(X)$ can thus be written as:

$$f(X) \sim \mathcal{GP}(\mu(X), k(X, X')) \tag{1.11}$$

Prior to observing data, the mean $\mu(X)$ of $f(X)$ is usually considered to be 0. This is done for simplicity, however other choices for the mean function have been explored (Rasmussen, 2003).

Prediction using Gaussian Processes

In order to make predictions using Gaussian Processes, we have access to a set of observed point locations $\mathbf{X_o} = \{X_1, X_2, ...X_N\}$ and their corresponding observations $\mathbf{f}(\mathbf{X_o})$. This set can be used to make predictions for an unobserved point X_*, for which we wish to estimate the value of $f(X_*) = y_*$. When using Gaussian Processes, it is assumed that there is a prior joint distribution between $\mathbf{f}(\mathbf{X_o})$ and $f(X_*)$. This distribution is considered to be Gaussian and is given as:

$$\begin{bmatrix} \mathbf{f}(\mathbf{X_o}) \\ f(X_*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X_o}, \mathbf{X_o}) & K(\mathbf{X_o}, X_*) \\ K(X_*, \mathbf{X_o}) & K(X_*, X_*) \end{bmatrix}\right), \tag{1.12}$$

where the term $K(\mathbf{X}', \mathbf{X}'')$, with $\mathbf{X}' = \{X'_1, X'_2, \ldots, X'_{N'}\}$ and $\mathbf{X}'' = \{X''_1, X''_2, \ldots, X''_{N''}\}$, is a covariance matrix of size $N' \times N''$ containing the covariances $k(X'_i, X''_j)$ between all the pairs in

the sets $\mathbf{X}'$ and $\mathbf{X}''$.

Using Gaussian Processes to make predictions, for the unobserved point X_*, involves computing the posterior distribution at X_* from the joint distribution (1.12), given the observed points location $\mathbf{X_o} = \{X_1, X_2, ...X_N\}$ and their corresponding observations $\mathbf{f}(\mathbf{X_o})$. This posterior distribution is given by:

$$f(X_*) \mid \mathbf{X_o}, \mathbf{f}(\mathbf{X_o}), X_* \sim \mathcal{N}(\mu(X_*), \text{cov}(X_*)), \tag{1.13}$$

where the posterior mean $\mu(X_*)$ and the posterior covariance $\text{cov}(X_*)$ are defined as:

$$\mu(X_*) = K(X_*, \mathbf{X_o})K(\mathbf{X_o}, \mathbf{X_o})^{-1}\mathbf{f}(\mathbf{X_o}), \tag{1.14a}$$
$$\text{cov}(X_*) = K(X_*, X_*) - K(X_*, \mathbf{X_o})K(\mathbf{X_o}, \mathbf{X_o})^{-1}K(\mathbf{X_o}, X_*). \tag{1.14b}$$

$\mu(X_*)$ will be the best estimate of $f(X_*)$ which can be derived from the data and $\text{cov}(X_*)$ the variance.

Covariance functions (kernels)

We can see that in the expressions (1.14), the mean and variances strongly depend on the function chosen to describe the covariances between points. These functions are also known as kernels. An example of one of the most widely used covariance function is the squared exponential kernel, also known as Radial Basis Function (**RBF**). It is defined as follows:

$$k_{\text{RBF}}(X, X') = \sigma_f^2 \exp\left(-\frac{\|X - X'\|^2}{2\ell^2}\right), \tag{1.15}$$

A more flexible version of the **RBF** kernel which will be used in the sequel is the Matérn kernel defined by:

$$k_{\text{Matérn}}(X, X') = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{\ell}\|X - X'\|\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{\ell}\|X - X'\|\right), \tag{1.16}$$

where K_ν is known as the *Bessel function*.

In the expressions (1.15) and (1.16), ν, ℓ, σ_f^2 are hyper-parameters which specify how the function values change w.r.t the inputs. These hyper-parameters can be optimized based on the observed data $\mathcal{D}_o = \{\mathbf{X_o}, \mathbf{y_o}\}$, where $\mathbf{y_o} = [f(X_1), f(X_2), .., f(X_N)]^T$. Let us denote by $\theta = [\nu, \ell, \sigma_f]^T$ the vector containing these hyper-parameters. The optimal hyper-parameters θ that allow to best fit the data are obtained by maximising the log marginal likelihood, which quantifies how well the Gaussian Process with given hyper-parameters θ explains the observed data $\mathbf{y_o}$ (Rasmussen, 2003):

$$\log p(\mathbf{y_o} \mid \mathbf{X_o}, \boldsymbol{\theta}) = -\frac{N}{2}\log(2\pi) - \frac{1}{2}\log|K(\mathbf{X_o}, \mathbf{X_o})| - \frac{1}{2}\mathbf{y_0}^\top K(\mathbf{X_o}, \mathbf{X_o})^{-1}\mathbf{y_o}. \tag{1.17}$$

The parameters θ that maximize (1.17) are obtained using, *e.g.*, gradient based optimization algorithm such as the limited memory BFGS (Liu and Nocedal, 1989).

An important aspect to notice in (1.14), is that the variance of the estimate of $f(X_*)$ only depends of the covariance functions of observed and unobserved point locations, *i.e.* $\mathbf{X_o}$ and X_*, and not on the observations $\mathbf{f}(\mathbf{X_o})$. As can be seen in equations (1.15) and (1.16), the covariance functions strongly depend on a distance measure (in this case an Euclidean distance) to express the similarity between points. Therefore, the variances obtained with Gaussian Processes are based on a distance measure, which tells us how far the point to evaluate X_* is from the observed points $\mathbf{X_o}$.

To use Gaussian Processes for regression, we can follow similar steps to the ones discussed in Section 1.2.2 for MLPs. In this case we follow the following procedure:

a. Data set pre-processing (we can here use the same pre-processing as in Section 1.2.2).

b. Kernel function selection.

c. Kernel function hyper-parameters optimization using (1.17).

d. Model validation (see Section 1.4).

1.3.2 Multiple correlated Output Gaussian Processes

Until now we have considered the case where the function to estimate f has a scalar output y. However, in some cases one may face the task of estimating a function F for which the output is a vector $Y \in \mathbb{R}^{D_Y}$, where D_Y is the size of the output vector. In this section we briefly discuss how a model for such functions can be identified using Gaussian Processes.

Two approaches can be used to estimate such a function F:

- Identify one Gaussian Process model for each output y_i in the vector $Y = [y_1, y_2, ..., y_{D_Y}]$;
- Identify a multivariate Gaussian Process model, which provides a vector valued output.

Due to the computational complexity of identifying Gaussian Processes with multiple outputs, the first choice is usually preferred. In such cases, the procedures described in Section 1.3.1 can be used.

However, in cases where the elements of the vector Y are correlated, then it can be interesting to directly identify a multivariate Gaussian Process, which incorporates the correlation between the outputs.

The methods for identifying a Gaussian Process model with multiple outputs follow the same approach as for the single output case. The main difference lies in the way to express the kernel function. A common approach is to construct the kernel as a product of two kernels: one which captures the correlation between the outputs and the second which is applied to the input space. Such cases are not treated in this thesis but more details on how to construct such kernels can be found in (Alvarez et al., 2012).

1.4 Performance metrics

The last step to perform after identifying a model $\hat{F}$ from a training data set is the validation. This is performed on a test set $\mathcal{D}_t = \{X_i, Y_i\}_{i=1}^{N_t}$ which contains samples that have not been used to identify the model. Different metrics can be used to measure the performance of an identified model. In this thesis, the ones which are used are :

- The **Root Mean Squared Error (RMSE)** defined as:

$$RMSE = \sqrt{\frac{1}{N_t}\sum_{i=1}^{N_t} \|\hat{Y}_i - Y_i\|^2}, \tag{1.18}$$

  where $\hat{Y}_i$ represents the predicted output vector from the identified model $\hat{F}$ for a given input X_i, $\hat{F}(X_i) = \hat{Y}_i$. The RMSE quantifies the average magnitude of the prediction error in the same units as the output variable, allowing us to assess the accuracy of the model and its practical relevance.

- The **coefficient of determination** R^2, defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{N_t} \|\hat{Y}_i - Y_i\|^2}{\sum_{i=1}^{N_t} \|\hat{Y}_i - \bar{Y}\|^2}, \tag{1.19}$$

  with $\bar{Y}$ the vector containing the mean values of each component in the vector Y_i. The R^2 provides a normalized measure indicating the goodness of fit.

These two complementary metrics are employed because they offer a robust assessment of both relative and absolute model performance. Note that similar criteria can be employed for the cases where we are evaluating the performance of an identified model of a dynamic system, see (5.15).

1.5 Conclusion

The aim of this chapter was to present some of the data-driven models which can be used for the identification of static systems. Multiple models are available in the literature and the choice depends on the application. Amongst the different models available, two have been presented in this chapter and will be used in the sequel. First a non-probabilistic model belonging to the class of Neural Networks, *i.e.*, a Multi Layer Perceptron was introduced. Some of its main advantages lie in its capacity to approximate complex high-dimensional functions and ability to handle well large data sets. Afterwards, a non-parametric and probabilistic model, *i.e.* a Gaussian Process was also presented. Compared to Multi Layer Perceptrons, the main advantage of Gaussian Processes is that they provide a measure of uncertainty along with the predictions.

In the next chapter we will see how such models can be employed to find the optimum of an expensive to evaluate function, and how we can select the data to improve their performances.

Chapter 2

Data selection to improve models for optimization

Once the data-driven models presented in Chapter 1 have been correctly identified, they can be used to find an optimum of an unknown function when it is expensive, in time or resources, to evaluate. However, if the model has been identified with few data in the vicinity of the optimum of the function, then the performance of this model for optimization will be limited. To enhance this performance, one must improve the accuracy of the model for the optimization purpose. The aim of this chapter is to present procedures which can be used to select the data to use in order to improve the accuracy of a model which is being used for optimization. We first review existing methods in Section 2.1, then delve deep into one of the most frequently used method: Bayesian Optimization, in Section 2.2. We then present a similar procedure to Bayesian Optimization but where the model of the function to optimize is a Neural Network in Section 2.3. We finish by presenting a numerical illustration of these approaches in Section 2.4

Contents

2.1 Surrogate based optimization 25
2.2 Bayesian Optimization 28
2.2.1 Acquisition functions 30
2.2.2 Bayesian Optimization for constrained optimization 31
2.3 Neural Network Smart Grid Based Optimization 32
2.3.1 Exploring Multi Layer Perceptrons 33
2.3.2 Acquisition function for the MLP 35
2.3.3 Constrained optimization case 36
2.4 Numerical Illustration 39
2.5 Conclusion 41

2.1 Surrogate based optimization

In many applications, one seeks to find the optimum of a static function f_{obj}. This function may be unknown and expensive to evaluate, *i.e*, obtaining the value of $f_{obj}(X)$ for a given location X requires a large amount of resources and/or time. The optimization problem could be for example:

$$X_{opt} = \arg \max_{X \in \mathcal{X}} f_{obj}(X), \tag{2.1a}$$

$$\text{s.t. } g_1(X) \leq c_1, \tag{2.1b}$$

where $X \in \mathbb{R}^{D_X}$ is a vector of design variables of dimension D_X and $\mathcal{X}$ is the feasible set. g_1 is a constraint function which can also be considered expensive to evaluate. An optimization algorithm which requires a large number of evaluations of the functions f_{obj} and g_1 will therefore not be the most ideal. Thus, one must find a way to limit the costs associated to the optimization.

Even if f_{obj} is unknown and expensive to evaluate, one may have access to an initial dataset, from, *e.g.*, past experiments, of different observations $f_{obj}(X)$ at different point locations X: $\mathcal{D}_N = \{X_i, f_{obj}(X_i)\}_{i=1}^{N}$, as well as $\mathcal{D}_{\tilde{N}} = \{X_i, g_1(X_i)\}_{i=1}^{\tilde{N}}$. In the sequel, we will consider that the points X which have been observed for f_{obj} are the same which have been evaluated on g_1, thus $N = \tilde{N}$. The first obvious choice in such cases is to look for an optimum in the available dataset:

$$X_{opt}^{\mathcal{D}_X} = \arg \max_{X \in \mathcal{D}_X \cap \mathcal{X}} f_{obj}(X), \tag{2.2a}$$

$$\text{s.t. } g_1(X) \leq c_1, \tag{2.2b}$$

where $\mathcal{D}_X$ is the set containing the points X in the datasets $\mathcal{D}_N$ and $\mathcal{D}_{\tilde{N}}$. The solution to (2.2) may either be satisfying enough, in which case there would be no need for further evaluations of the expensive functions, not good enough or there may be no solution at all. In the last two cases, one must look for a new optimum.

A way to achieve this, is to use the datasets $\mathcal{D}_N$ and $\mathcal{D}_{\tilde{N}}$ along with the identification procedures of data-driven models presented in Chapter 1, to identify models for f_{obj} and g_1. At a given point location X, we would then have estimates $\hat{f}_{obj}(X)$ of $f_{obj}(X)$ and $\hat{g}_1(X)$ of $g_1(X)$. The identified models $\hat{f}_{obj}$ and $\hat{g}_1$ will be cheaper and faster to evaluate than the original expensive functions. Therefore, the solution of the optimization problem (2.1) can be estimated by solving:

$$X_{opt}^{*} = \arg \max_{X \in \mathcal{X}} \hat{f}_{obj}(X), \tag{2.3a}$$

$$\text{s.t. } \hat{g}_1(X) \leq c_1. \tag{2.3b}$$

Since $\hat{f}_{obj}$ and $\hat{g}_1$ are less costly and faster to evaluate, the solution X_{opt}^{*} of (2.3) will also be less costly and faster to obtain than the solution of (2.1). However, the estimated optimum X_{opt}^{*} will depend on the quality of the models $\hat{g}_1$ and $\hat{f}_{obj}$, which itself depends of the datasets $\mathcal{D}_N$ and $\mathcal{D}_{\tilde{N}}$ used to identify the models. If the models have been identified with very few data in the vicinity of X_{opt}, then X_{opt}^{*} may be far from X_{opt} and thus $f_{obj}(X_{opt})$ and $g(X_{opt})$ are far from $\hat{f}_{obj}(X_{opt})$ and $\hat{g}_1(X_{opt})$. To continue using the identified models, one must improve their accuracy by adding new samples to the datasets used to identify them. However, since the functions are expensive to evaluate, only a limited number of evaluations can be performed to collect new data. The new points X to evaluate in order to increase the accuracy of the model must thus be chosen carefully.

In this chapter, we review some approaches which are used to select what data to use in order to improve a model for optimization purposes. We then propose a new one which can be applied when the model of the unknown function is a Neural Network.

The most common way to select points X to evaluate in order to improve a model for an optimization purpose relies on a trade-off between two objectives:

- **Exploitation:** Improve the model in regions of the design space that have shown promising results, *i.e*, in regions where the model is optimum.

- **Exploration:** Improve the model in uncertain or untested regions of the design space to discover potentially optimal solutions and understand the global behaviour of the objective function.

The most popular algorithm which performs the trade-off between exploration and exploitation, to select the data to evaluate on the objective function is the so-called Bayesian Optimization. This procedure is described in Section 2.2.

However, there are other existing approaches in the literature that accomplish this trade-off (Vu et al., 2017). For instance, (Jakobsson et al., 2010) uses a Radial Basis function (RBF) interpolation model to identify the model of the objective function. The next point to evaluate is then chosen using a *quality function*, which combines the distance to observed samples (used for uncertainty) to a weight function which takes a low or high value depending on the estimate of the objective function.

Similar to (Jakobsson et al., 2010), (Regis and Shoemaker, 2007) also uses an RBF model to estimate the objective function and provides two approaches to estimate its optimum. In the first procedure, the authors suggest to choose the next point to evaluate as the best local minimizer of the RBF model obtained from multiple initialization points of the optimization algorithm. If the best local minimizer is too close to observed samples (based on a tolerance parameter) or is an already observed sample, then a local minimizer which is farther is chosen instead. If all local minimizers are close to observed samples, random points are generated in the vicinity of one of them and the random point with the "best" value (in an optimum sense) of the RBF model is chosen as the next point to evaluate.

The second approach, see also (Gutmann, 2001) and (Vu et al., 2017), is less in the spirit of the trade-off between exploration and exploitation. Consider a training set $\mathcal{D} = \{X_i, f_{obj}(X_i)\}_{i=1}^N$ and a f_{obj}^* a guess of the global minimum of the expensive function f_{obj}. The next point to evaluate is then chosen as the point X_{N+1} such that the RBF model which learns $\mathcal{D} = \{X_i, f_{obj}(X_i)\}_{i=1}^N$ and the sample $\{X_{N+1}, f_{obj}^*\}$ has minimum *bumpiness* (*i.e.* is the smoothest function). This is based on the idea that among all possible approximations of the RBF model, the smoothest one is the most accurate and should therefore be chosen to find the optimum. Note that this method requires to have a good estimate of the value of the optimum of the objective function: f_{obj}^*, otherwise the obtained optimum location may be far from the real one.

Regardless of their effectiveness in estimating functions, Neural Networks are less used in approaches similar to the ones described above. This is mainly due to the fact that classic Neural Networks do not provide a direct measure of uncertainty which is frequently used for exploration in the other approaches.

In this chapter, we develop an approach to select the data to improve a Neural Network model which is being used for optimization. We show that even without a measure of uncertainty, Neural Networks can also be used in procedures balancing exploration and exploitation to find the optimum of an expensive function. The developed approach can also be employed as an alternative for Bayesian Optimization and in some cases, may provide better performances. Before presenting the approach in Section 2.3, we provide a brief review of Bayesian Optimization in the following section.

2.2 Bayesian Optimization

Bayesian Optimization is a famous machine learning algorithm which can be used to find an optimum of a function. It has been employed in many domains such as control (Guzman et al., 2020),(Bombois and Forgione, 2022), aerodynamic design (see Chapter 4.2) or environmental monitoring (Marchant and Ramos, 2012). We first consider the case where Bayesian Optimization is used to solve an optimization problem without constraints:

$$X_{opt} = \arg\max_{X \in \mathcal{X}} f_{obj}(X), \tag{2.4}$$

where $X \in \mathbb{R}^{D_X}$ is a vector of design variables of dimension D_X, $\mathcal{X}$ is the feasible set and f_{obj} is an objective function for which we want to find the maximum X_{opt}. We here consider a maximisation problem for ease of the presentation (this will become more clear in the sequel). If we wish to find the minimum of f_{obj}, we can simply consider the problem of maximising its negative value.

Bayesian Optimization is mostly interesting when the objective function f_{obj} is unknown and expensive to evaluate, *i.e.*, it requires a large amount of resources or time to evaluate the value of $f_{obj}(X)$ for a given value of X. As mentioned above, for such functions, an optimization algorithm which requires a large number of function evaluations will not be the most appropriate. However, one may have access to an initial data of different observations $f_{obj}(X)$ at different point locations X: $\mathcal{D}_N = \{X_i, f_{obj}(X_i)\}_{i=1}^N$. The key idea behind Bayesian Optimization is to identify a probabilistic model of $f_{obj}(X)$ from the dataset $\mathcal{D}_N$ and use it to find the optimum in a limited number of k evaluations.

Note however that in some cases, there may not be any initial observations from the objective function, *i.e*, $N = 0$. In such cases, few points X are evaluated on f_{obj} in order to have access to a dataset to identify the probabilistic model.

The most commonly used probabilistic model in Bayesian Optimization is a Gaussian Process. However, the use of other models such as Neural Networks to which a Bayesian layer is added on the last layer of the network (Snoek et al., 2015) or Inverse distance weighting and radial basis functions (Bemporad, 2020), has also been explored. In this section, we focus on the use of Gaussian Processes in Bayesian Optimization. As seen in the Section 1.3.1, using a Gaussian Process regression model, $f_{obj}(X)$ is considered as a random variable and is approximated by a normal distribution:

$$f_{obj}(X) \sim \mathcal{N}\left(\mu(X), \sigma^2(X)\right), \tag{2.5}$$

where $\mu(X)$ is the best estimation of $f_{obj}(X)$ and $\sigma^2(X)$ is the variance, see (1.14). Using the information provided by the Gaussian Process, Bayesian Optimization seeks to iteratively find the most promising points X to test on f_{obj}, with the final goal of approximating the optimal point X_{opt}, while working within a limited number of evaluations k. These points are chosen based on a trade-off between the exploration and exploitation objectives, which in this case are defined as follows:

- **Exploitation objective:** Searching for points where the model of the function to evaluate is optimum *i.e.* where $\mu(X)$ is large;

- **Exploration objective:** Searching for points where the model of the function to evaluate is the most uncertain, *i.e.*, where $\sigma(X)$ is large.

It then becomes clear why Gaussian Processes are used in Bayesian Optimization, since they provide a measure of uncertainty which can directly be used for the exploration objective. The latter is very important for two reasons. First of all, it allows testing points in regions where the model is uncertain. Indeed, if the optimum is in these regions and the model estimates them wrongly, without the exploration objective the optimum would never be tested. Furthermore, the point which accomplishes the trade-off between the exploration and exploitation objectives is used to update the model. With the exploration objective, the model will thus be updated in regions where it is the most uncertain.

In order to find the points that accomplish the trade-off between exploration and exploitation, we maximise a so-called "Acquisition function" $A(X)$. These functions use the estimate $\mu(X)$ as well as the uncertainty on the estimate $\sigma(X)$, provided by the Gaussian Processes to accomplish the trade-off. A brief review of different ways to accomplish this trade-off is provided in Section 2.2.1. For now, we just consider that we have an acquisition function $A(X)$ and that its maximum provides the next point X_{N+1} to evaluate:

$$X_{N+1} = \arg\max_{X \in \mathcal{X}} A(X) \tag{2.6}$$

The point X_{N+1}, is then evaluated on f_{obj}, and the new sample $\{X_{N+1}, f_{obj}(X_{N+1})\}$ is added to the dataset $\mathcal{D}$ which now contains $N+1$ samples. With this new data set, the Gaussian Process is updated and provides more accurate estimates of f_{obj}. The acquisition function can then be constructed and maximised once again, to find the next point X_{N+2} to evaluate. The procedure is repeated iteratively for k iterations until we can no longer afford to evaluate the objective function. After performing the k iterations, we have a dataset $\mathcal{D}$ containing $N+k$ samples. The maximum, is then chosen as the point X for which $f_{obj}(X)$ is the largest in the new augmented dataset. The general procedure is described in Algorithm 1.

Algorithm 1 Bayesian optimization algorithm

for $j = 0, ..., k$, with k the maximum number of iterations, **do**

1. Train GP with available set of samples. $\mathcal{D}_{N+j} : \{X_i, f(X_i)\}_{i=1}^{N+j}$
2. Build acquisition function $A_j(X)$ (see Section 2.2.1 for examples).
3. The next point X_{N+j} to evaluate on f_{obj} is

$$X_{N+j} = \arg\max_X A_j(X)$$

4. Evaluate X_{N+j} on f_{obj} and add $(X_{N+j}, f_{obj}(X_{N+j}))$ to the training samples $\mathcal{D}_N$.

end for

return X with the highest value of $f_{obj}(X)$ from $\mathcal{D}_{N+k}$

2.2.1 Acquisition functions

Acquisition functions are one of the main ingredients of the Bayesian Optimization procedures. In this section, we give a brief review of some of the commonly used acquisition functions.

Upper Confidence Bound

The *Upper Confidence Bound* (UCB) is probably the most straightforward acquisition function. It is defined as follows:

$$UCB(X) = \mu(X) + \beta\sigma(X), \tag{2.7}$$

where β is a trade-off parameter. In (2.7), $\mu(X)$ is used for the exploitation objective and $\sigma(X)$ for the exploration objective. The value of the parameter β determines whether to perform more exploration or more exploitation. The larger the value of β is, the more the search will concentrate on the exploration objective and vice-versa. Note that β can also be chosen for other objectives, *e.g.*, to bound some cumulative regret, see (Srinivas et al., 2012).

Probability of Improvement

Another widely used acquisition function is the *Probability of Improvement* (PI). Let X_+ be value of X in the dataset corresponding to the highest value of f_{obj}. For a given new point location X, the improvement I over the current maximum location is:

$$I(X) = \max\left(0, \mu(X) - f_{obj}(X_+)\right). \tag{2.8}$$

The probability of improvement at a given point location X, $PI(X)$, is the probability that the Improvement $I(X)$ is positive. This probability can be obtained as follows:

$$\mathrm{PI}(X) = \Phi\left(\frac{\mu(X) - f_{obj}(X_+)}{\sigma(X)}\right), \tag{2.9}$$

where $\Phi(\cdot)$ is the cumulative distribution function (CDF) of the standard normal distribution.

Expected Improvement

A similar acquisition function to the probability of improvement is the *Expected Improvement* (EI). This function considers the expectation value of the improvement, instead of its probability.

It is defined as follows:

$$\mathrm{EI}(X) = \mathbb{E}[I(X)], \tag{2.10}$$

where $I(X)$ is the improvement defined in (2.8). This leads to the following formulation of the expected improvement (Frazier, 2018):

$$EI(X) = (\mu(X) - f(X_+) - \xi)\Phi(Z) + \sigma(X)\phi(Z), \tag{2.11}$$

where:

$$Z = \frac{\mu(X) - f_{obj}(X_+) - \xi}{\sigma(X)}, \tag{2.12}$$

- $f_{obj}(X_+)$ is the best observed value of the objective function so far;
- $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution;
- $\phi(\cdot)$ is the probability density function of the standard normal distribution;
- ξ is a parameter which can be used to control the degree of exploration. This parameter can also be added in the expression of the probability of improvement (2.9).

2.2.2 Bayesian Optimization for constrained optimization

The procedure described in the sections above can be performed to solve optimization problems without constraints. However, Bayesian Optimization can also be used to solve constrained optimization problems such as:

$$X_{opt} = \arg \max_{X \in \mathcal{X}} f_{obj}(X), \tag{2.13a}$$

$$\text{s.t. } g_1(X) \leq c_1, \tag{2.13b}$$

where $g_1(X)$ and $f_{obj}(X)$ are both considered expensive to evaluate. The basic idea behind Bayesian Optimization described in Algorithm 1 does not change much. The main difference is in the expression of the acquisition function. Many expressions of the acquisition function for Bayesian Optimization for constrained optimization problems have been proposed (see (Gelbart, 2015) for some examples). In this chapter, we consider the approach described in (Gardner et al., 2014) which is the mainly used one.

To solve the optimization problem (2.13), both the objective function f_{obj} and constraint $g_1(X)$ are modelled by two independent Gaussian Processes. Therefore, in addition to the dataset $\mathcal{D}_N = \{X_i, f_{obj}(X_i)\}_{i=1}^N$, we also have access to a dataset $\mathcal{D}_{\tilde{N}} = \{X_i, g_1(X_i)\}_{i=1}^{\tilde{N}}$. We here consider that the point locations X which have been evaluated on f_{obj} are the same which have been used to evaluate g_1, therefore, $N = \tilde{N}$. The estimate of f_{obj} is the same as in (2.5) and the estimate of $g_1(X)$ can also be written as:

$$g_1(X) \sim \mathcal{N}\left(\mu_1(X), \sigma_1^2(X)\right). \tag{2.14}$$

The acquisition function which is used to solve optimization problems under constraints using Bayesian Optimization is the Expected Improvement (2.11) extended to include the constraints.

More precisely, the Expected Improvement (2.11), is multiplied by a probability of feasibility of the constraint $PF(X)$:

$$PF(X) := \Pr[\mu_1(X) \leq c_1]. \tag{2.15}$$

In this case $PF(X)$ is a univariate Gaussian cumulative distribution function. The final acquisition function for constrained optimization is thus:

$$A(X) = PF(X)EI(X), \tag{2.16}$$

where $EI(X)$ is the acquisition function described in Section (2.2.1). In this case, the definition of X_+ in (2.11) is extended to be the currently known maximum location of f_{obj} in the dataset $\mathcal{D}_N = \{X_i, f_{obj}(X_i)\}_{i=1}^N$ that respects the given constraint, *i.e.*, $g_1(X_+) \leq c_1$. The procedure described in Algorithm 1 is then performed with these minor changes:

- In step 1 of the `for` loop, 2 Gaussian Processes models are identified. One for g_1 and the other for f_{obj};
- In step 4 of the `for` loop, the maximum of the acquisition function (2.16) is evaluated on both g_1 and f_{obj};
- Finally, the optimum is chosen as the point X with the highest value of $f_{obj}(X)$ such that $g_1(X) \leq c_1$.

Note that this procedure can also be extended for more than one constraint (Gardner et al., 2014).

Remark: Regardless of the effectiveness of Bayesian Optimization, it may face a limitation associated to the computational cost of identifying a Gaussian Process model. Indeed, in step 1 of the `for` loop in Algorithm 1, one must identify one or multiple Gaussian Process models (depending on the number of constraints). Identifying this model requires tuning the hyperparameters of the covariance function. As shown in Section 1.3.1, this is achieved by maximising (1.17). To compute (1.17), one must invert a covariance matrix of size $N \times N$, where N is the number of observed samples. The computational complexity of performing this operation is $\mathcal{O}(N^3)$. Therefore, if we wish to perform Bayesian Optimization with an initially large dataset or if we need to perform a large number of iterations to get close to the optimum, then Bayesian Optimization may not be the most appropriate approach. Compared to Gaussian Processes, Neural Networks are known to scale better when the number of samples increases. In the next section, we see how such models can be used in optimization procedures similar to Bayesian Optimization.

2.3 Neural Network Smart Grid Based Optimization

As discussed in the previous chapter, Neural Networks excel in modelling complex nonlinear and high dimensional functions. It therefore makes sense to see how they can be used in procedures similar to Bayesian Optimization. Therefore, in this section we present how we can select data to improve a Neural Network model which is being used to find the optimum of a function f_{obj}.

We first consider the case where the optimization problem does not include constraints. Similar to Section 2.2, the goal is to find the maximum of an objective function f_{obj} (see (2.4)) where

$X \in \mathbb{R}^{D_X}$ is a vector of design variables of dimension D_X and $\mathcal{X}$ is the feasible set. In this case also, f_{obj} is an unknown and expensive function to evaluate. We also have access to an initial dataset of different observations $f_{obj}(X)$ at different point locations X: $\mathcal{D}_N = \{X_i, f_{obj}(X_i)\}_{i=1}^N$ and the possibility to perform k experiments on f_{obj} to collect new samples.

The idea behind this approach will be similar to Bayesian Optimization, see Section 2.2. If we have the possibility to iteratively evaluate k times the objective function to obtain the optimum, then the point X to evaluate on f_{obj} must also accomplish the exploitation and exploration objectives for the same reasons stated in the previous sections.

2.3.1 Exploring Multi Layer Perceptrons

In Bayesian Optimization, the exploration is accomplished by testing in regions where the uncertainties provided by the Gaussian Process model are the highest. However, unlike Gaussian Processes, regular MLPs do not provide a measure of uncertainty. In this case, we propose to perform the exploration with the objective of covering regions not already covered by the dataset (and therefore not already observed by the MLP). The rationale behind this proposal is that when a point X to be predicted is far from the observed points, then it is likely that the prediction $\hat{f}_{obj}(X)$ will differ from the target value $f_{obj}(X)$. This idea is used to create a measure which is employed for the exploration objective of the proposed procedure. In this sense, $\delta_{min}(X)$[1] is defined as the Euclidean distance between a point X and the closest point to it from the points in the dataset $\mathcal{D}_X = \{X|(X, f_{obj}(X)) \in \mathcal{D}_N\}$. It is therefore defined as:

$$\delta_{min}(X) = \min_{X_i \in \mathcal{D}_X} \|X - X_i\|. \tag{2.17}$$

This means that $\delta_{min}(X)$ is small when evaluating a point close to the observed points and increases when evaluating a point far from the observed ones. When $\delta_{min}(X)$ is low, it is likely that the estimate of $f_{obj}(X)$, $\hat{f}_{obj}(X)$ is close to the real value $f_{obj}(X)$ and vice-versa. As mentioned in Section 1.3.1, the uncertainties of the Gaussian Process are also more a measure of how far the points to predict are far from the observed samples. Therefore, the measure δ_{min} used for the exploration is similar to the uncertainties provided by the Gaussian Process.

Illustration of the comparison between δ_{min} and the GP uncertainties

The measure δ_{min} can be compared to the uncertainties provided by a GP model. To illustrate this, $N = 5000$ random samples are generated from a function $f(X)$ which corresponds to the 12 dimensional form of the Rosenbrock function defined as:

$$f(X) = \sum_{i=1}^{D_X - 1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2], \tag{2.18}$$

with $X = (x_1, x_2..., x_{D_X}) \in \mathbb{R}^{D_X}$ and $D_X = 12$. The goal of this example is to visualize how the MLP performs and how the Gaussian Process standard deviation behaves when the value of δ_{min} increases. For this purpose, a GP and a MLP model of the 12 dimensional Rosenbrock function, are identified using the dataset $\mathcal{D}_N = \{X_i, f(X_i)\}_{i=1}^N$. A new dataset $\mathcal{D}_l$ containing $l = 500$ new samples $\notin \mathcal{D}_N$ is generated. The GP and MLP models are then used to predict the

[1] This measure is also used in (Jakobsson et al., 2010) as an uncertainty quantification method for Radial Basis Functions.

500 unseen points from $\mathcal{D}_l$. The following elements are then calculated:

- the standard deviation $\sigma(X)$ for $X \in \mathcal{D}_l$;
- the $\delta_{min}(X)$ for $X \in \mathcal{D}_l$ and computed with (2.17) where $\mathcal{D}_X$ is the set containing the $k = 5000$ initial points;
- the error of the MLP for each point $X \in \mathcal{D}_l$ determined as the absolute value of the difference between the estimate of the Rosenbrock function by the MLP $\hat{f}(X)$ and the actual value $f(X)$: error $= |\hat{f}(X) - f(X)|$.

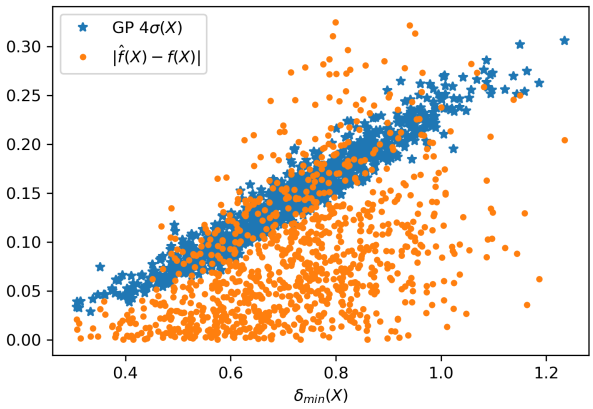


Figure 2.1: Value of GP σ (represented by the blue stars, note that this value is multiplied by 4 for better visualization) and MLP error (represented by the orange dots) w.r.t δ_{min} (horizontal axis).

Figure 2.1 presents the GP standard deviation (σ) and the MLP error w.r.t to δ_{min} for different tested points. On this figure, the value on the vertical axis of each orange dot represents the error on the MLP prediction ($|\hat{f}(X) - f(X)|$) of an unseen tested point X and the corresponding value on the horizontal axis, represents the value of δ_{min} for that point. The value on the vertical axis of each blue star represents the GP standard deviation σ of an unseen tested point X and the corresponding value on the horizontal axis, represents the value of δ_{min} for that point.

We can observe that the points X for which $\delta_{min}(X)$ is low (*i.e*, points in regions well covered by the dataset) are relatively well predicted by the MLP. The GP standard deviation $\sigma(X)$ is also low for these points. However, we see that when $\delta_{min}(X)$ increases, more and more points tend to be poorly predicted by the MLP as the error increases. The GP standard deviation is also greater for points where $\delta_{min}(X)$ is large.

Therefore, by using the measure $\delta_{min}(X)$ for the exploration objective of our procedure, we will test in regions where the model is likely to perform poorly. As mentioned before, this is essential because if the optimum is in those regions and therefore poorly predicted, without the exploration it may never be tested. This will also allow improving the quality of the model in regions not already covered by the dataset.

2.3.2 Acquisition function for the MLP

The measure δ_{min} is therefore used to construct the acquisition function for our procedure. The objective function f_{obj} is identified by a MLP model $\hat{f}(X)$ using the initial samples $\mathcal{D}_N$. We then choose the following expression for the acquisition function $A(X)$ for the presented procedure:

$$A(X) = \beta\delta_{min}(X) + \hat{f}_{obj}(X). \tag{2.19}$$

The acquisition function is very similar to the Upper Confidence Bound (2.7). Compared to the definition of the latter in (2.7), $\hat{f}(X)$ plays the role of the GP $\mu(X)$ and serves for the exploitation purpose, δ_{min} serves the exploration purpose just like $\sigma(X)$ in (2.7) and β is a trade-off hyper-parameter which balances the search between exploration and exploitation. The choice of β is very crucial in this acquisition function also. First of all, β must be chosen in a way that both the exploration and exploitation terms have the same order of magnitude. Afterwards, the value of β can be increased or decreased to permit more exploration or more exploitation, respectively. One could also, *e.g.*, start with a relatively high value of β in the first iterations (to enforce more exploration when the model is less accurate) and decrease β in the subsequent iterations (to enforce more exploitation when the model becomes more accurate).

In our procedure, the first $k-1$ points to evaluate are chosen by achieving the trade-off between the exploration and exploitation objectives, *i.e.*, by maximisation of the acquisition function (2.19). Since the real function can only be evaluated once after the final iteration, performing exploration in the objective of improving the model is no longer useful. The last point to evaluate is thus, chosen solely through the exploitation objective, *i.e.*, by solving:

$$X_k = \arg\max_{X\in\mathcal{X}} \hat{f}_{obj}(X). \tag{2.20}$$

The main steps of the procedure are described in Algorithm 2. Step 1 of the `for` loop in this algorithm which corresponds to the identification of the MLP model can usually be performed faster after each iteration using a "*warm start*" approach. As mentioned in Section 1.2, the goal of the identification of a MLP is to find a parameter vector θ which minimises an identification criterion such as (1.9). The algorithms used to minimize this criterion (*e.g.* Gradient-descent) usually require an initial guess for θ. At iteration $j = 0$ of the `for` loop in Algorithm 2, this initial guess will usually be chosen randomly. However, for iterations $j = 1, ..., k$ we can use the parameter vector θ_{j-1}, which corresponds to the parameters of the MLP identified at iteration $j-1$ with the dataset $\mathcal{D}_{N+j-1}$, as an initial guess for the identification of the parameters θ_j for the MLP model at iteration j with the dataset $\mathcal{D}_{N+j}$. Since the new MLP model will not change too drastically compared to the previous model by learning only one new sample, then the parameters θ_j will not be too different from θ_{j-1}. Therefore, this initialization will allow the optimization algorithm (used to identify the model) to converge faster and therefore fasten the overall procedure.

Algorithm 2 MLP model improvement for optimization

for $j = 0, ..., k-1$ **do**

1. Identify MLP model $\hat{f}_{obj}$ with available set of samples $\mathcal{D}_{N+j} : \{X_i, f_{obj}(X_i)\}_{i=1}^{N+j}$
2. Build acquisition function $A_j(X) = \beta\delta_{min}(X) + \hat{f}_{obj}(X)$
3. The next point X_{N+j} to evaluate on f_{obj} is

$$X_{N+j} = \arg\max_{X \in \mathcal{X}} A_j(X)$$

4. Evaluate X_{N+j} on f_{obj} and add $(X_{N+j}, f(X_{N+j}))$ to the training samples D_{N+j}.

end for

Identify MLP model $\hat{f}_{obj}$ with available set of samples $\mathcal{D}_{N+k-1}$

The final point to evaluate on f_{obj} is:

$$X_{N+k} = \arg\max_{X \in \mathcal{X}} \hat{f}_{obj}(X)$$

Evaluate X_{N+k} on f_{obj} and add $(X_{N+k}, f_{obj}(X_{N+k}))$ to the training samples $\mathcal{D}_{N+k}$.

return X with the highest $f_{obj}(X)$ from D_{N+k}

2.3.3 Constrained optimization case

In this section we consider the problem of finding the solution to the problem (2.13), where $f_{obj}(X)$ and $g_1(X)$ are both expensive to evaluate. In addition to the dataset $\mathcal{D}_N = \{X_i, f_{obj}(X_i)\}_{i=1}^{N}$, we also have access to a dataset $\mathcal{D}_{\tilde{N}} = \{X_i, g_1(X_i)\}_{i=1}^{\tilde{N}}$. We consider here also the case where the point locations X which have been evaluated on f_{obj} are the same which have been used to evaluate g_1, therefore, $N = \tilde{N}$.

To include constraints in the proposed approach, it is necessary to have an idea of whether they are likely to be respected or not, for a given point X. This is achieved by identifying a MLP model for g_1 using the dataset $\mathcal{D}_{\tilde{N}}$. Note that it is also possible to build a MLP model with 2 outputs, one which represents the estimate of f_{obj} and the other the estimate of g_1.

We then build a penalty function $\lambda(X)$ which will be multiplied by the acquisition function (2.19). The penalty function should be high if the constraints are likely to be respected and low if not. First of all, $\lambda(X)$ will depend of the estimate of the constraint function $\hat{g}_1(X)$. Indeed, if $\hat{g}_1(X) \leq c_1$, *i.e*, the model predicts that the constraints are respected, then $\lambda(X)$ should take a large value. If $\hat{g}_1(X) > c_1$, *i.e* the model predicts that the constraints are not respected then $\lambda(X)$ should take a small value. The exact value of $\lambda(X)$ will depend on the value of δ_{min}. Indeed, as mentioned before, if the points which are being evaluated are far from the observed samples then it is likely that their estimate is far from the true value. We therefore consider the following cases:

1. The model predicts that the constraints are respected for a given X:
   (a) If $\delta_{min}(X)$ is small, then it is likely that the constraints are actually respected. Thus, $\lambda(X)$ should be the highest of all possible scenarios;
   (b) If $\delta_{min}(X)$ is large, then it is likely that the constraints are actually not respected. Thus, $\lambda(X)$ should be lower than in the previous case.

2. The model predicts that the constraints are not respected for a given X:

   (a) If $\delta_{min}(X)$ is small, then it is likely that the constraints are actually not respected. Thus, $\lambda(X)$ should be the lowest of all possible scenarios;

   (b) If $\delta_{min}(X)$ is large, then it is likely that the constraints are actually respected. Thus, $\lambda(X)$ should be higher than in the previous case.

Regardless of the value of $\delta_{min}(X)$, the value of $\lambda(X)$ when the model predicts that the constraints are respected will always be higher than its value when it predicts that the constraints are not respected. With the previous considerations, we define the value of $\lambda(X)$ as follows:

$$\lambda(X) = \begin{cases} 2 - \delta_{min}^{scaled}(X) & \text{if } \hat{g}_1(X) \leq c_1 \\ \delta_{min}^{scaled}(X) & \text{otherwise,} \end{cases} \tag{2.21}$$

where

$$\delta_{min}^{scaled}(X) = \frac{\delta_{min}(X)}{\max\limits_{\tilde{X} \in \mathcal{X}} \delta_{min}(\tilde{X})}. \tag{2.22}$$

$\delta_{min}^{scaled}(X)$ is a scaled version of δ_{min} (2.17) in order to have its values range between 0 and 1. $\max\limits_{\tilde{X} \in \mathcal{X}} \delta_{min}(\tilde{X})$ represents the highest possible value of δ_{min} in the search domain $\mathcal{X}$ which corresponds to the distance between the lowest possible coordinates of X and the highest possible. With the definition of (2.21) the values of λ will vary between 0 and 2:

- if the model predicts that the constraints are not respected for a given X, $\lambda(X)$ will vary between 0 and 1. When $\delta_{min}(X)$ is small, we are in the situation (2a) and the value of $\lambda(X)$ will be close to 0 ($\delta_{min}^{scaled}(X)$ is close to 0). As $\delta_{min}(X)$ increases, we get closer and closer to the situation (2b), therefore $\lambda(X)$ gets closer and closer to 1, which is the maximum value it can reach when the constraints are not respected;

- if the model predicts that the constraints are respected for a given X, $\lambda(X)$ will vary between 2 and 1. When $\delta_{min}(X)$ is small, we are in the situation (1a) and the value of $\lambda(X)$ will be close to 2. Note that this value can be increased if we wish to give more importance to the case where the constraints are respected. As $\delta_{min}(X)$ increases, we get closer and closer to the situation (1b), therefore $\lambda(X)$ gets closer and closer to 1 which is the minimum value it can reach when the constraints are respected.

A visual representation of the possible values $\lambda(X)$ can take depending on whether the constraints are respected and the value of $\delta_{min}(X)$ is provided in Figure 2.2.

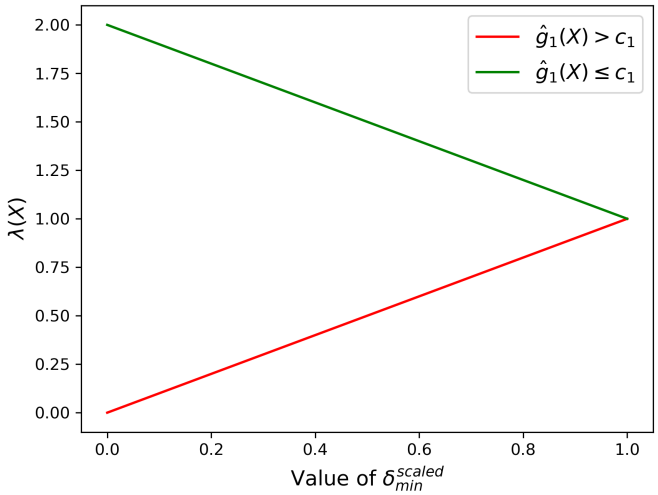


Figure 2.2: Value of $\lambda(X)$ (vertical axis) when the constraints are respected (green) and when they are not respected (red) w.r.t δ_{min}^{scaled} (horizontal axis).

As mentioned above the penalty term $\lambda(X)$ (2.21) is multiplied by the original acquisition function. The new acquisition function is thus:

$$A_{cons}(X) = \lambda(X)[\beta\delta_{min}(X) + \hat{f}_{obj}(X) + \rho], \tag{2.23}$$

where ρ is a scalar offset added to make sure that the term $\beta\delta_{min}(X) + \hat{f}_{obj}(X) + \rho$ is non negative. This term is necessary since we want $A_{cons}(X)$ to be the lowest when $\lambda(X)$ is the lowest *i.e* equal to 0.

Just like for Bayesian Optimization, the general procedure does not change in the case where the optimization problem comes with constraints. Therefore, Algorithm 2 can be performed with these minor changes:

- in step 1, in addition to the model of f_{obj}, a model of g_1 is also identified;
- in step 3 of the for loop, $A(X)$ (2.19) is replaced by A_{cons} (2.23);
- in step 4 of the for loop, the maximum of the acquisition function (2.23) is evaluated on both g_1 and f_{obj};
- finally the optimum is chosen as the point X with highest value of $f_{obj}(X)$ such that $g_1(X) \leq c_1$.

Note that this procedure can be extended to multiple equality or inequality constraints by simply changing the `if` condition in (2.21) to include the other constraints.

2.4 Numerical Illustration

This section follows on from Section 2.3.1: the objective is to perform the optimization of the 12 dimensional form of the Rosenbrock function $f(X)$ (2.18). In this case study, the proposed optimization approach using Neural Networks will be compared to Bayesian Optimization. In the sequel Bayesian Optimization will be referred to as BO and the solution using Neural Networks (see Section 2.3 and Section 2.3.3) as NNSGO (Neural Network Smart Grid based Optimization).

Before performing the procedures, a preliminary study is carried out to find the structures of the NN and of the GP models that best fit the data using, *e.g*, a grid search approach (Liashchynskyi and Liashchynskyi, 2019), see Section 1.2.

The NN and GP models are built using *Scikit-learn* (Buitinck et al., 2013), a famous Python machine learning library which offers the possibility to build simple MLP regression models and GPs. For more advanced NN models, tools such as *PyTorch* (Paszke et al., 2019) or *Keras* (Ketkar and Ketkar, 2017) could also be used.

The optimization of different acquisition functions is achieved using the Limited-memory BFGS (Byrd et al., 1995) method in its *Scipy* (Virtanen et al., 2020) implementation. Simulations are performed on a PC equipped with a 12th Gen Intel CORE i7-12700H processor with 14 cores and 64 GB of RAM.

The Rosenbrock function has one global minimum at $X = (1, 1, ..., 1)$ where $f(X) = 0$. The optimization is performed here in the following search domain[2]:

$$\mathcal{X} = \{X = (x_1, .., x_{12})^T | -2.048 < x_i < 2.048 (i = 1, ..., 12)\}, \tag{2.24}$$

where $X \in \mathbb{R}^{12}$.

For the purpose of this study, it is considered that the function is unknown and expensive (in time and resources) to evaluate. However, we have access to a dataset containing 5000 samples (pairs $\{X^n, f(X^n)\}_{n=1}^k$ with $k = 5000$) and we have the possibility to test 100 additional points X. The dataset is generated randomly in a corner of the search domain which does not include the global minimum to motivate the use of a smart grid based approach. In this initial dataset, the point X_D, for which the value of the Rosenbrock function is the lowest, yields $f(X_D) = 226$.

The goal of this section is to find the optimum X_{opt} of the following optimization problem:

$$X_{opt} = \arg\max_{X \in \mathcal{X}} -f(X). \tag{2.25}$$

To solve the optimization problem (2.25)[3], the NNSGO is performed using Algorithm 2 with the following considerations:

- the number of iterations is set to 100;

[2] The search domain corresponds to the one usually used in the literature to evaluate optimization algorithms on the Rosenbrock function

[3] Note here that the problem is formulated as a maximization problem to match with the concepts presented in this chapter.

- the NN model used is a MLP with 1 hidden layer with 256 neurons and a *ReLU* activation function on each neuron;
- the maximum of the acquisition function provides the next point X to evaluate and is given as:
$$A(X) = -\hat{f}(X) + \beta\delta_{min}(X)$$
where $\hat{f}(X)$, is the estimate of $f(X)$ by the MLP and $\beta = 1$.

The NNSGO is compared to BO which is performed using Algorithm 1 with the following parameters:

- the number of iterations is set to 100;
- a GP with a Matern kernel (1.16) is used;
- the acquisition function is here chosen as the UCB (2.2.1) where $\mu(X)$ is the GP estimate of $-f(X)$, $\sigma(X)$ the standard deviation provided by the GP model and $\beta = 2$.

The values of β in Algorithm 1 (BO) and Algorithm 2 (NNSGO) are chosen to have the same order of magnitude between the exploration and exploitation terms.

Table 2.1: Optimization results with 100 additional points

Method used	Optimization time	Value of optimum
NNSGO	40 seconds	$f(X_{NNSGO}) = 40$
BO	1 hour	$f(X_{BO}) = 38$

Table 2.1 presents the values of f for the optimum points found using the NNSGO procedure, noted X_{NNSGO}, and BO, noted X_{BO}, as well as the time needed to run 100 iterations of both procedures. As shown in this table, both methods perform similarly in terms of optima found. By increasing the dataset with 100 additional points using the two smart grid based approaches (BO and NNSGO), we are able to find optimal points X for which the value of $f(X)$ is 84% smaller than the initially smallest value in the dataset ($f(X_D) = 226$).

Even if the optima found with both approaches (BO and NNSGO) have close values of $f(X)$, the NNSGO has a clear advantage in terms of the computational time needed to perform 100 iterations in this case study. This is because Neural Networks scale better in training time than regular GPs when the size of the dataset is large. In this particular case, the NNs takes 8 seconds to train whereas the GPs take 30 seconds. Furthermore, after each iteration of the NNSGO, the previously trained model parameters are re-used as an initialization for the new model. This further decreases the training time of the updated NN models. This is achieved by using the *warm start* option in *scikit-learn* multi layer perceptron regressor implementation. Without using this option, the optimization when using the NNSGO procedure takes 12 minutes, which is still 5 times lesser than when using the regular BO to perform 100 iterations. (Note that the

warm-start option is not available for GP regression models in *scikit-learn*).

An example of how BO and NNSGO procedures can be applied in other fields such as Controller Design is provided in Appendix A.

2.5 Conclusion

The goal of this chapter was to present how we can select the data to use in order to improve models, which are being used for optimization, in key regions. In this sense, we presented how two of the most widely used identification methods, *i.e.*, Gaussian Processes and Neural Networks, can be used for this purpose. First, we showed how Bayesian Optimization selects the data to use in order to improve a Gaussian Process model for optimization purposes. Then, an alternative method to Bayesian optimization for the optimization of costly unknown functions is also proposed. It is based on the use of Neural Networks instead of Gaussian Processes. This method uses the distance between points, instead of the usual uncertainties provided by Gaussian Processes, to search for a potential optimum to be tested next on the unknown function. The proposed procedure provides competitive results to regular Bayesian Optimization. It is also illustrated that when the initial dataset contains a large number of samples, the optimization procedure using the proposed approach will require less time to perform a certain number of iterations than the classic Bayesian optimization algorithm and can thus be more interesting in such cases.

In the next sections, the methods presented in Chapter 1 and 2 are applied to solve aerodynamic and ballistic problems. First in Chapter 3, the study focuses on using the models presented in Chapter 1 to predict aerodynamic coefficients. Then in Chapter 4, it is shown how such models can be used to find optimal configurations of projectiles using the optimization procedures presented in this chapter.

Chapter 3

Aerodynamic coefficients prediction

In the previous chapters, methods which can be used to identify models for static systems were presented and it was shown how such models can be employed for optimization purposes. One of the key areas where such methods can be useful is in the aerodynamic and ballistics domain. In these studies, we are usually interested in finding optimal designs for vehicles based on their aerodynamic characteristics. Due to the lack of directly available and easy to evaluate physical models that allow characterizing aerodynamically the vehicles, it is usually necessary to identify a model which estimates these characteristics. Finding the latter usually involves estimating the so-called aerodynamic coefficients. Such coefficients are essential in any study related to aerodynamics and ballistics but are expensive and time-consuming to obtain. Therefore, having a model which estimates them fast and accurately is essential to finding optimal designs for vehicles (see Chapter 4.2). The aim of this chapter is thus to see how the approaches presented in Chapter 1 can be used to identify models which predict aerodynamic coefficients of a projectile. The chapter is structured as follows: we first present aerodynamic coefficients, how they can be measured and the methods which have been used in literature to estimate them in Section 3.1 and Section 3.2 respectively. We then present the prediction of aerodynamic coefficients for a specific type of finned projectiles in Section 3.3. Two approaches to estimate the coefficients are presented based on the sources of the data.

Contents

3.1 Aerodynamic Coefficients 44

3.1.1 Aerodynamic Forces and Moments 44

3.1.2 Stability derivatives 47

3.1.3 Aerodynamic coefficients estimation methods 47

3.2 Review of methods used to predict aerodynamic coefficients 49

3.3 Study Case 51

3.3.1 Projectile 51

3.3.2 Dataset 52

3.3.3 Estimating f_{CFD} from f_{aero} or f_{PRODAS} 52

3.3.4 Results 54

3.4 Conclusion 57

3.1 Aerodynamic Coefficients

Aerodynamic coefficients are dimensionless quantities associated to the forces and moments acting on a vehicle. In this thesis, the vehicle in question corresponds to a flying projectile. These coefficients are essential in understanding the behaviour of the projectile in flight under various conditions. They can be used to, *e.g.*, design the projectile in order to optimize its performances and stability, or to control it.

To understand aerodynamic coefficients, one must first of all take a look at the forces and moments acting on the projectile.

3.1.1 Aerodynamic Forces and Moments

The forces and moments acting on a projectile are usually expressed in two coordinate systems: the wind frame and the body frame.

- **Body frame**: The body frame is a coordinate system that is fixed to the body. Its coordinates x_b, y_b, z_b and origin are described as follows:
  - the origin is located at the Center of Gravity (C.G.) of the projectile;
  - the x-axis x_b points forward along the longitudinal axis of the projectile;
  - the y-axis y_b points to the right of the projectile;
  - the z-axis z_b points downward, *i.e.*, toward the bottom of the projectile.

- **Wind frame**: The wind frame is also known as aerodynamic frame. It is a coordinate system that is aligned with the trajectory of the projectile through the air. Its coordinates x_w, y_w, z_w and origin are described as follows:
  - the origin is located at the center of gravity (C.G.) of the projectile;
  - the x-axis x_w points forward along the velocity vector $\vec{V}$ of the projectile;
  - the y-axis y_w points perpendicular to the x_w axis and lies in the horizontal plane of symmetry of the projectile;
  - the z-axis z_w points downward and perpendicular to both x_w and y_w.

These two reference frames are illustrated in Figure 3.1. More details can be found in (Zipfel, 2000),(Morelli and Klein, 2016) and (Cook, 2012).

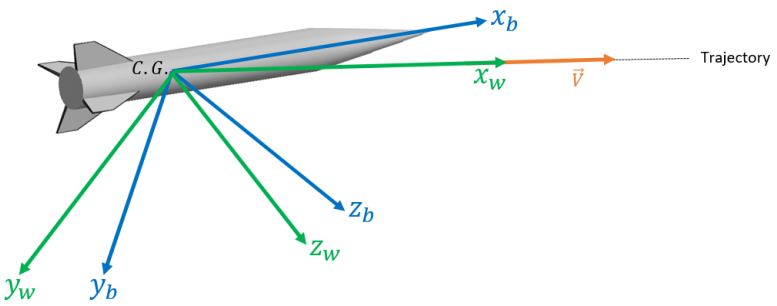


Figure 3.1: Body and Wind frames

Depending on the application (control, aerodynamic analysis,...) engineers will prefer to use the wind or body frame to express the forces and moments. However, it remains possible to translate from one frame to another (Albisser, 2015).

The aerodynamic forces acting on the projectile expressed in the body frame are the Axial Force (F_A), the Normal force (F_N) and the Side force (F_Y). Three moments act on the projectile during its flight. The roll moment L, the pitch moment M and the yaw moment N.

In this thesis, the ones which are studied are the Axial Force, the Normal Force and the Pitch Moment. The Axial force is the one acting in the x_b axis of the body. It provides to the projectile its forward motion. The Normal force acts perpendicular to the surface of the body in the direction of the z_b axis. It allows the projectile to rise and stay airborne. The pitching moment M, is applied along the y_b axis and causes the nose of the vehicle to move up and down.

In the wind frame (or aerodynamic frame) which is also frequently used, the Normal force and the Axial force are replaced by the Lift force F_L and Drag force F_D. The Drag force is the force opposing the forward velocity of the projectile. It is in the opposite direction of the x_w axis of the wind frame. It quantifies how much the projectile resists to the fluid it is travelling into. The aerodynamic lift force, is the force perpendicular to the trajectory, *i.e.*, along the z_w axis of the wind frame. It tends to pull the projectile upwards or downwards depending on its orientation.

The axial force F_A, the normal force F_N, the drag force F_D, the lift force F_L and the pitching

moment M are expressed as follows:

$$F_A = \bar{q}\, S\, C_A \tag{3.1a}$$
$$F_N = \bar{q}\, S\, C_N \tag{3.1b}$$
$$F_D = \bar{q}\, S\, C_D \tag{3.1c}$$
$$F_L = \bar{q}\, S\, C_L \tag{3.1d}$$
$$M = \bar{q}\, S\, d\, C_m, \tag{3.1e}$$
$$\text{with } \bar{q} = \frac{1}{2}\rho V^2, \tag{3.1f}$$
$$\text{and } S = \frac{(\pi d^2)}{4}). \tag{3.1g}$$

In equations (3.1), $\bar{q}$ is the dynamic pressure, ρ represents the density of the fluid and V the velocity of the projectile. The constant d is the reference diameter of the projectile and S the reference surface area.

The quantities C_A, C_N, C_D, C_L, C_m are the **aerodynamic coefficients** related to **the Axial force, Normal force, Drag force, Lift force, pitching moment** respectively (McCoy, 1999). Note that the Axial and Normal force aerodynamic coefficients are directly related to the Lift and Drag force aerodynamic coefficients. Indeed, we have:

$$F_D = F_N\, sin\, \alpha_t \; - \; F_A\, cos\, \alpha_t \tag{3.2a}$$
$$F_L = F_N\, cos\, \alpha_t \; + \; F_A\, sin\, \alpha_t. \tag{3.2b}$$

From (3.1) and (3.2), we derive that:

$$C_D = C_N\, sin\, \alpha_t \; - \; C_A\, cos\, \alpha_t \tag{3.3a}$$
$$C_L = C_N\, cos\, \alpha_t \; + \; C_A\, sin\, \alpha_t. \tag{3.3b}$$

In equations (3.2) and (3.3), the angle α_t expressed in radians, is known as the **total angle of attack**. It can be approximated by:

$$\alpha_t = \sqrt{\alpha^2 + \beta^2}, \tag{3.4}$$

where α is the **angle of attack**, *i.e*, the angle between the longitudinal axis of the body (x_b) and the axis of the projectile direction (x_w). β is known as the **sideslip angle** and represents the deviation of the projectile flight path from its intended direction in the horizontal plane.

These aerodynamic coefficients are very essential because they allow characterising the behaviour of the projectile in flight. They depend of several factors:

- the shape of the projectile[4];
- the velocity of the projectile. In aerodynamics, we usually refer to the **Mach Number**, which represents the ratio between the velocity of the projectile and the speed of sound in the fluid the projectile is travelling into;
- the total angle of attack of the projectile.

3.1.2 Stability derivatives

The aerodynamic coefficients described above exhibit nonlinear behaviours w.r.t to quantities such as the total angle of attack α_t and angular velocities, *i.e.*, the rate at which the projectiles rotates or spins around fixed axis. This allows decomposing them into components which express how they change w.r.t these quantities. These components are called **stability derivatives**. The aerodynamic coefficients described in the section above are approximated by:

$$C_A = C_{A_0} + C_{A_2} \, sin^2 \, \alpha_t \tag{3.5a}$$
$$C_D = C_{D_0} + C_{D_2} \, sin^2 \, \alpha_t \tag{3.5b}$$
$$C_L = C_{L_\alpha} \, sin \, \alpha_t \tag{3.5c}$$
$$C_N = C_{N_\alpha} \, sin \, \alpha_t \tag{3.5d}$$
$$C_m = C_{m_\alpha} \, sin \, \alpha_t + C_{m_q} \frac{d}{2V} \, q, \tag{3.5e}$$

where q is the pitch rate, *i.e.*, the rate at which the projectile nose moves up or down about its lateral axis y_b. The quantities $C_{A_0}, C_{A_2}, C_{D_0}, C_{D_2}, C_{L_\alpha}, C_{N_\alpha}, C_{m_\alpha}$ and C_{m_q} are the **stability derivatives**. They will depend on both the geometry of the projectile and the velocity it is travelling at, *i.e.*, the Mach number. Some stability derivatives are crucial in the analysis of the projectile. For example, C_{m_α} and C_{m_q} allow analysing the static and dynamic stability of the projectile, see (Albisser, 2015).

3.1.3 Aerodynamic coefficients estimation methods

There are no readily available physical models which provide aerodynamic coefficients or stability derivatives based on the shape and Mach number of the projectile. However, they can be obtained in different ways.

The first possibility, which is also considered as the baseline, corresponds to experimental tests. They allow measuring the coefficients by putting the projectile in the desired flight conditions. There are two possibilities to measure them:

- **Free flight experiments**: In these experiments, the projectile is launched with desired flight conditions, *i.e.*, at a certain initial velocity and angle of attack. Then using different measurement techniques (sensors, cameras,..), a certain amount of data is collected which allows estimating the forces, velocity, trajectories... The coefficients are then calculated

[4]The material composing the projectile is considered to be uniformaly distributed, therefore quantities such as the center of gravity, inertia, etc. directly depend of the shape of the projectile.

from the estimated quantities. (Albisser, 2015)

- **Wind Tunnel experiments**: Wind tunnels are experimental facilities which allow studying the effects of air moving over and around projectiles. They are composed of a tunnel from which the wind is blown at a controlled speed. This then allows analysing the behaviour of the projectile and calculate its forces and moments and therefore obtain the aerodynamic coefficients.

The main advantage of experimental tests is associated to the quality of the coefficients obtained. Since the projectile is put into its real flight conditions, the measured coefficients are the ones which allow describing the behaviour of the projectile the best. There may be, nevertheless, some measurement noise affecting the quality of the obtained coefficients.

Experimental tests come with a considerable cost associated to the time and resources required to carry them out. Indeed, they require a lot of material resources, to build and launch projectiles, human resources and usually require a large amount of time. This means that only a few quantity of data of aerodynamic coefficients can be obtained from experimental tests due to the costs associated to their realisation.

The second, and one of the most widely used possibility to obtain aerodynamic coefficients, is **Computational Fluid Dynamics** (**CFD**) (Anderson and Wendt, 1995). In CFD simulations, the physics between the projectiles and their interaction with air are simulated using computers. Different solvers and numerical methods are then used to solve equations (Navier Stokes equations (Anderson and Wendt, 1995)) which describe the motion of fluid substances. Solving these equations provide the pressure distribution over the surfaces of the projectile which can be used to calculate aerodynamic forces and moments. The latter can then be used to calculate aerodynamic coefficients.

Compared to experimental tests, CFD simulations will require less resources. However, the computational cost of running simulations is still quite high and may require expensive computational resources. Moreover, the simulations often take a large amount of time. Nevertheless, the coefficients obtained from CFD simulations are usually close to the ones obtained from experiments. In a more general case, even if the ones obtained from experiments will still be considered as the baseline and will still be preferred to the ones from CFD, many engineers use these simulations to obtain accurate coefficients since, in some cases, there is no possibility to perform experiments. Computational Fluid Dynamics simulations are also usually performed before testing the projectiles via the experimental approaches to have an idea of what will be obtained.

Experimental tests and CFD simulations are the main methods which are used to obtain the coefficients. There exist some other tools which are often sold commercially and that allow estimating aerodynamic coefficients. These tools usually consist of semi-empirical codes built over databases of experimental or CFD simulation data. However, in most cases, there is neither the possibility to have access to the database which was used nor the models behind them. Therefore, they can be considered as black box models. The main advantage of these tools is that they allow having fast estimations of aerodynamic coefficients. Indeed, one can evaluate multiple configurations in a very short time. However, compared to CFD simulations, the coef-

ficients obtained from these commercial codes are usually less close to the experimental ones.

Figure 3.2 summarizes the fidelity, the cost and the amount of data collected, for the different aerodynamic coefficients estimation methods discussed in the previous paragraphs.



Figure 3.2: Summary of aerodynamic coefficients estimation methods

To conclude, aerodynamic coefficients (and stability derivatives) are very essential in the analysis of a projectile behaviour. They allow analysing its stability, design its shape to increase its performances and control it. However, obtaining them is a very difficult task. Indeed, the methods which provide accurate coefficients are usually expensive to carry out. On the contrary, the coefficients provided by methods which are faster to use, usually lack of accuracy. One of the main growing area of research in aerodynamics concerns the application of artificial intelligence approaches to estimate these coefficients from available data. In the next section, we review some of the work that has been performed in this area.

3.2 Review of methods used to predict aerodynamic coefficients

Due to the complexity of obtaining precise aerodynamic coefficients, there has been a growing interest in the use of machine learning to build models capable of predicting them. Most of the work done in this domain is on airfoils, which are shapes designed to augment lift of an object. The prediction of the aerodynamic coefficients of such shapes using machine learning models has been studied in the aerospace domain as suggested in (Moin et al., 2022) and (Chen et al., 2020) for aircrafts. This study has also been performed for other objects than aircrafts. For example, in (Oh, 2020), the authors are interested in predicting the lift to drag ratio (*i.e.* the ratio between the lift and drag aerodynamic coefficients) of a wind turbine. In the military domain, the prediction of aerodynamic coefficients of missiles has also been investigated, see, *e.g.*, (Lee et al., 2021).

The main models used to predict aerodynamic coefficients are Neural Networks. The choice of the Neural Network type depends of the quantity of available data, the type of data or the dimensions of the features and outputs. For example, Convolutional Neural Networks (CNN) are used in (Chen et al., 2020) to predict the coefficients. The use of CNN is due to the fact that the authors use image data as input, where CNNs are known to excel for predictions. In (Tumse et al., 2022), Feedforward Neural Networks and Elman Neural Networks are used to estimate the drag and lift coefficients of a delta wing. In (Secco and Mattos, 2017), a more vanilla type of Neural Network is used to predict aerodynamic coefficients of a transport airplane.

Gaussian Processes are also frequently employed for such tasks. In (Lee et al., 2021) they are used to predict the coefficients of the missile. Their application can also be found in (Akian

et al., 2022) to predict the ones of an airfoil and in (Valayer et al., 2024) to predict the drag, the lift and the pitch moment aerodynamic coefficients.

Other models have also been considered to accomplish the predictions. In (Yetkin et al., 2024), the authors provide examples of the use of models such as XGB, Catboost and LightGBM for such tasks and provide the interest of each model.

In the studies presented above, the considered main sources of data are CFD simulations and Wind Tunnel experiments. Moreover, the authors use only one source at a time to identify their data-driven models. However, in some cases, you may have access to datasets from different sources. These datasets may have different levels of fidelity depending on the method used to obtain them. For example, as discussed in the previous Section 3.1, aerodynamic coefficients obtained from CFD simulations will be more viable than those obtained from semi-empirical tools.

In many cases, the dataset from the source with the highest level of fidelity, contains less samples than the ones with lower levels of fidelity. This is due to the fact that obtaining more accurate measurements is usually more expensive (in time and resources) than obtaining less accurate measurements. In these scenarios, the model should be designed to predict the measurements that would come from the most reliable source. However, if the number of samples in the most reliable source is limited, identifying a model may be a challenging task.

The main idea in such cases is to use the knowledge from lower fidelity data to improve the predictions of a model built with higher fidelity data. This idea is known in the literature, as **multi fidelity learning** (Fernández-Godino, 2016) (Kennedy and O'Hagan, 2000) (Le Gratiet, 2013). The idea behind these approaches is that a higher fidelity model y_h can be approximated from a lower fidelity model y_l combined with some additive or multiplicative correction terms learnt using data-driven models. In the case where the lower fidelity model is sufficiently close to the higher fidelity model, these correction terms may be easier to learn than the whole high fidelity model where only few samples are available. This idea has been applied in the aerodynamic literature. In (Arenzana et al., 2021), it is used to combine data from CFD simulations and wind tunnel experiments in order to predict aerodynamic coefficients. This is accomplished using an autoregressive Gaussian Process model (Fernández-Godino, 2016) (Kennedy and O'Hagan, 2000). More applications of this idea can be found in *e.g.* (Kuya et al., 2011) and (Kou and Zhang, 2019).

To summarize, multiple models have been employed to predict aerodynamic coefficients. The main two models which are used are Neural Networks and Gaussian Processes even if the use of other models has also been explored. The choice mainly depends on the type of dataset and its size. For large dimension datasets and for a large number of samples, Neural Networks may be preferred to Gaussian Processes. However, when a measure of uncertainty is required, Gaussian Processes may be preferred to Neural Networks. In the next sections, we see how the methods presented in Chapter 1 can be used to identify a model which estimates aerodynamic coefficients of a projectile fitted with fins. We also present a multi-fidelity learning approach using only a single Neural Network. Such models are crucial in studies related to aerodynamic especially in aerodynamic design, see Chapter 4.2, due to the cost associated to measuring aerodynamic coefficients.

3.3 Study Case

3.3.1 Projectile

In this study, we are mainly interested in estimating the drag, the lift and the pitching moment coefficients of different shapes of projectiles fitted with fins. Such projectiles can be characterized for different values of parameters described in Table 3.1. In this table all the parameters are in calibre (1 calibre = 28 mm) except for X_2 which is in degrees. A visual representation of these parameters for a 2D model of the projectile is provided in Figure 3.3

Table 3.1: Main characteristics of a projectile fitted with fins

Characteristic X	Design Parameter	Unit
X_1	Total Length	calibre
X_2	Nose Angle	degrees
X_3	Fins Tip Length	calibre
X_4	Fins Chord Length	calibre
X_5	Wing span	calibre
X_6	Rear Fin Tip Shift	calibre

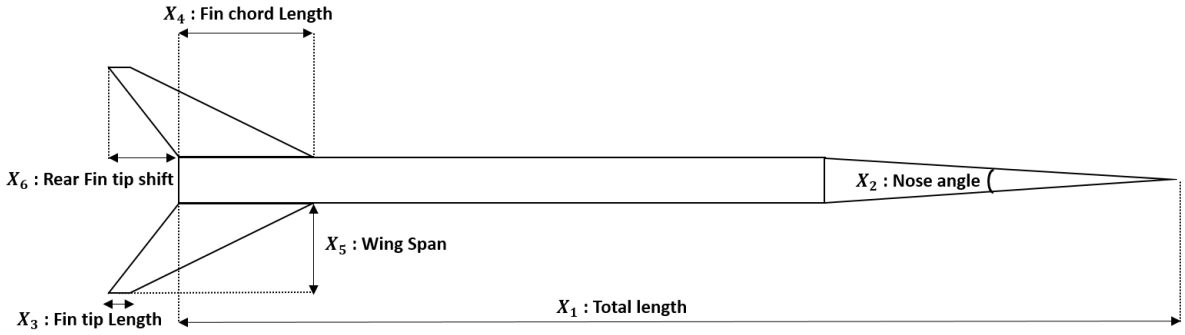


Figure 3.3: Finned projectile characteristics (Table 3.1)

Changing the values of these parameters will affect the overall shape of the projectile and therefore change its aerodynamic coefficients. Another projectile parameter which affects the values of aerodynamic coefficients is the number of fins. The projectiles studied in this case have 4 fins. As mentioned in Section 3.1.1, the angle of attack and the Mach number also influence the values of aerodynamic coefficients. The angle of attack is fixed to 1[5] degree and the value of the Mach number will be used as input to predict the coefficients. Overall, the input vector $X = [X_1, X_2, ..., X_7]^T$, which is used to estimate the projectile coefficients, contains Table 3.1 (parameters $X_i, i = 1, ..., 6$) along with the Mach number (parameter X_7).

[5]The dataset used to build the models only contains measurements at an angle of attack of 1°.

3.3.2 Dataset

To construct the model which predicts aerodynamic coefficients, we have access to a dataset $\mathcal{D}_{CFD}$ obtained from CFD simulations. This dataset contains 364 configurations of projectiles, *i.e.*, different combination values of $X_i, i = 1, ..., 6$: $X_1, X_2, ..., X_6$, associated to different values of Mach numbers X_7. In total there are 2074 different values of X. Each value of X is associated to a vector Y_{CFD} containing the drag, lift and pitch moment coefficients obtained from CFD simulations, $Y_{CFD} = [C_{D_{CFD}}, C_{L_{CFD}}, C_{m_{CFD}}]^T$. In the sequel, the coefficients will be predicted independently from one another, *i.e.*, a model will be built for each of the coefficients in Y_{CFD}. This is done to reduce the computational burden of identifying models with multiple outputs. For each of the three coefficients in Y_{CFD}, we have thus access to a dataset $\mathcal{D}^N_{CFD}$ such that $\mathcal{D}^N_{CFD} : \{X^i, y^i_{CFD}\}_{i=1}^N$, where $N = 2074$ represents the number of samples and y_{CFD} is one the 3 coefficients in Y_{CFD}. We define a function f_{CFD} which, for a given input vector X, provides the aerodynamic coefficient y_{CFD}.

$$y_{CFD} = f_{CFD}(X) \tag{3.6}$$

In addition to data from CFD, we have access to two black box models which allow estimating aerodynamic coefficients. These models, that we name f_{aero} and f_{PRODAS} (Obtained from a simulation code, named *PRODAS*, *i.e.*, *Projectile Rocket Ordnance Design & Analysis System*, see *www.prodas.com* for further informations), allow computing the value of an aerodynamic coefficient for a given input vector X:

$$y_{aero} = f_{aero}(X), \tag{3.7}$$

$$y_{PRODAS} = f_{PRODAS}(X), \tag{3.8}$$

where y_{aero} and y_{PRODAS} represent an aerodynamic coefficient estimated by f_{aero} and f_{PRODAS} respectively. The fidelity of these models is known to be lower than the one of CFD. However, they allow estimating the aerodynamic coefficients of the projectiles rapidly and without a large computational cost.

Since CFD has a higher fidelity than the black box models, our goal will be to identify a model which estimates f_{CFD}. With the data $\mathcal{D}^N_{CFD}$ and the models f_{aero} and f_{PRODAS}, we have two possibilities to accomplish this:

1. **Case 1**: Identify a model directly from the data $\mathcal{D}^N_{CFD}$.
2. **Case 2**: Use a multi-fidelity approach to estimate f_{CFD} from f_{aero} or f_{PRODAS}.

Case 1 can directly be accomplished using the procedures described in Chapter 1. We here choose to use a Neural Network model. Their parameters can thus be identified using the identification criterion (1.8) using the dataset $\mathcal{D}^N_{CFD}$. The results are presented in Section 3.3.4. In the next section, we explain how we can estimate f_{CFD} from a lower fidelity model, *i.e.*, Case 2.

3.3.3 Estimating f_{CFD} from f_{aero} or f_{PRODAS}

In this section, we discuss how f_{CFD} can be estimated using a lower fidelity model, in our case f_{aero} or f_{PRODAS}. The procedure to estimate f_{CFD} from f_{aero} or f_{PRODAS} will be the same. Therefore, in this section we consider the problem of estimating f_{CFD} from a lower fidelity model

that we name f_{LF}. In the numerical illustration in Section 3.3.4, f_{LF} will be replaced by f_{aero} or f_{PRODAS} depending on the source of the data. There are different possibilities which can be used to express a high fidelity model in terms of a low fidelity one, a summary can be found in (Fernández-Godino, 2016). In our case we choose to express f_{CFD} as:

$$f_{CFD}(X) = \rho(X) f_{LF}(X) + \Delta(X). \tag{3.9}$$

The expression (3.9) means that f_{CFD} can be expressed as f_{LF} multiplied by a function $\rho(X)$ plus a discrepancy term $\Delta(X)$ that captures the differences between the two functions for a given input X. Since, f_{LF} and f_{CFD} are both unknown, $\Delta(X)$ and $\rho(X)$ are also not known and must be estimated from some data.

In order to estimate $\Delta(X)$ and $\rho(X)$, we follow the following steps:

- **Initial dataset setup**: First we use the $N = 2074$ values of X in $\mathcal{D}^N_{CFD}$, to compute the corresponding aerodynamic coefficient $y_{LF} = f_{LF}(X)$. We thus have a dataset $\mathcal{D}^N_{LF} : \{X^i, y^i_{LF}\}^N_{i=1}$, where $N = 2074$.

- **Model structure selection**: To model $\rho(X)$ and $\Delta(X)$, we opt to use a Neural Network with two outputs:

  - the first output, $\hat{\rho}(X)$, provides an estimate of $\rho(X)$;
  - the second output, $\hat{\Delta}(X)$, provides an estimate of $\Delta(X)$.

- **Model parameter identification**: We denote by θ the parameters of the Neural Network model. Using the datasets $\mathcal{D}^N_{LF} : \{X^i, y^i_{LF}\}^N_{i=1}$ and $\mathcal{D}^N_{CFD}$, the identification of the model will consist in finding the parameters $\hat{\theta}$ which minimize the following criterion:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{N} \left[(\rho(X_i, \theta) f_{LF}(X_i) + \Delta(X_i, \theta)) - f_{CFD}(X_i) \right]^2 . \tag{3.10}$$

The parameter vector θ that minimizes (3.10) can be found using optimization routines which are typically used for the identification of Neural Network. Its structure (number of layers, number of neurons and activation functions) is found using the structure selection approaches using a validation set as discussed in Section 1.2.

As mentioned before, multi-fidelity learning approaches as the one presented in this section, are particularly employed when the number of samples collected from higher fidelity models, *e.g.* f_{CFD} in our case, are not enough to identify a good model. In such cases learning the discrepancy terms, $\rho(X)$ and $\Delta(X)$, between the lower and higher fidelity models, from fewer data may be easier than learning f_{CFD}. Note however, that this does not necessarily mean that the models for $\rho(X)$ and $\Delta(X)$ will be accurate since they have been identified from few samples. However they may provide better performances than a model of f_{CFD} in such cases. In this study, the dataset $\mathcal{D}^N_{CFD} : \{X^i, y^i_{CFD}\}^N_{i=1}$ contains $N = 2704$ observations, which can be considered as

large enough to identify a good model of f_{CFD} without the use of f_{LF}. Nevertheless, it remains interesting to identify $\hat{f}_{CFD}$ using the approach presented in this section. This will provide us with two different estimates which can both be used to approximate f_{CFD}.

3.3.4 Results

In this section we present the results obtained when identifying a model which estimates f_{CFD} using the dataset $\mathcal{D}^N_{CFD}$ only and when we use the multi-fidelity approach presented in Section 3.3.3. The results are presented for 3 coefficients, *i.e.*, the drag coefficient C_D, the lift coefficient C_L and the pitching moment coefficient C_m.

As explained in the Section 1.2, the dataset is divided into three sets: the training set (used to identify the models), the validation set (used to find the best structures and hyper-parameters), and the test set. The performance results which are presented in the sequel are based on the test set. The metrics used to measure the performances are the RMSE (1.18) and the R^2 score (1.19).

To avoid confusion, we will use the following notations:

- $\hat{f}_1$: the model identified from the dataset $\mathcal{D}^N_{CFD}$ only, *i.e.* using the method 1;
- $\hat{f}_2$: the model identified using the dataset $\mathcal{D}^N_{CFD}$ as well as f_{aero}, using the procedure described in 3.3.3.;
- $\hat{f}_3$: the model identified using the dataset $\mathcal{D}^N_{CFD}$ as well as f_{PRODAS}, using the procedure described in 3.3.3.

To identify these models, *i.e.*, to solve the criteria (3.10) and (1.8), the python machine learning library known as *PyTorch* (Paszke et al., 2019) is used.

Coefficients / Method	C_D		
	NN architecture	**RMSE**	**R^2**
Method 1 $\hat{f}_1$	1 hidden layer 128 neurons ϕ: tanh (1.2)	0.0132	0.99
Method 2 $\hat{f}_2$	1 hidden layer 128 neurons ϕ: tanh (1.2)	0.0127	0.99
Method 2 $\hat{f}_3$	1 hidden layer 128 neurons ϕ: tanh (1.2)	0.0133	0.99
Mean of all models	/	0.0124	0.99

Table 3.2: Neural Network Architectures used and performance metrics for the two methods for the C_D coefficient

Coefficients / Method	C_L		
	NN architecture	**RMSE**	$\mathbf{R^2}$
Method 1 $\hat{f}_1$	1 hidden layer 64 neurons ϕ: tanh (1.2)	5.2×10^{-3}	0.99
Method 2 $\hat{f}_2$	1 hidden layer 128 neurons ϕ: tanh (1.2)	5.7×10^{-3}	0.99
Method 2 $\hat{f}_3$	1 hidden layer 128 neurons ϕ: tanh (1.2)	5.7×10^{-3}	0.99
Mean of all models	/	4.7×10^{-3}	0.99

Table 3.3: Neural Network Architectures used and performance metrics for the two methods for the C_L coefficient

Coefficients / Method	C_m		
	NN architecture	**RMSE**	$\mathbf{R^2}$
Method 1 $\hat{f}_1$	1 hidden layer 128 neurons ϕ: tanh (1.2)	0.025	0.99
Method 2 $\hat{f}_2$	1 hidden layer 128 neurons ϕ: tanh (1.2)	0.024	0.99
Method 2 $\hat{f}_3$	1 hidden layer 128 neurons ϕ: tanh (1.2)	0.026	0.99
Mean of all models	/	0.020	0.99

Table 3.4: Neural Network Architectures used and performance metrics for the two methods for the C_m coefficient

The Tables 3.2, 3.3 and 3.4, present the results obtained from the different approaches. The three different estimations of the aerodynamic coefficients provide similar performances on the test data, with an overall good accuracy. Among the lower fidelity models, the one which provides the best results is the one built using data from f_{aero}, *i.e.*, $\hat{f}_2$. For the C_D and C_m coefficients, this model also outperforms $\hat{f}_1$, *i.e.*, the model built using data from f_{CFD} only.

In most cases, a Neural Network structure with 1 hidden layer and 64 or 128 neurons, see the Tables 3.2, 3.3 and 3.4 for more details, found using the validation set, is sufficient to identify an accurate model, for either f_1, f_2 or f_3.

The last row of these tables present the performances when f_{CFD} is estimated using the mean from all the estimates $\hat{f}_1$,$\hat{f}_2$ and $\hat{f}_3$. This estimate is denoted as f_{mean} and given by:

$$f_{mean}(X) = \frac{\hat{f}_1(X) + \hat{f}_2(X) + \hat{f}_3(X)}{3} \tag{3.11}$$

As shown in Tables 3.2, 3.3 and 3.4, taking the average value of the three estimates, $\hat{f}_1$,$\hat{f}_2$ and $\hat{f}_3$, to estimate f_{CFD} allows increasing slightly the performances. For the C_D coefficient the performance ($RMSE$ on the test data) is increased by 3% compared to the performance of the best estimate. For the C_L it is increased by 10% and for the C_m by 8%.

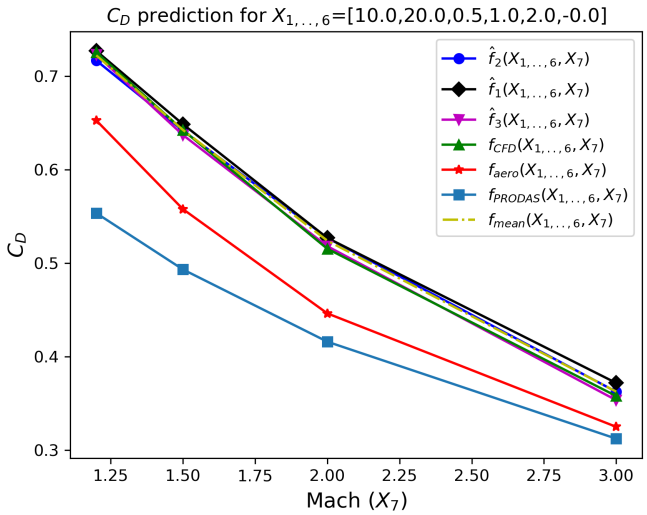


Figure 3.4: Prediction of C_D coefficient for a projectile configuration $X_{1,..,6}$ not in the training dataset at different Mach numbers X_7, using the models f_{mean} (yellow), $\hat{f}_3$ (violet), $\hat{f}_2$ (blue dots), $\hat{f}_1$ (black) and comparison with the target value f_{CFD} (green) and the lower fidelity models f_{aero} (red) and f_{PRODAS} (blue square)

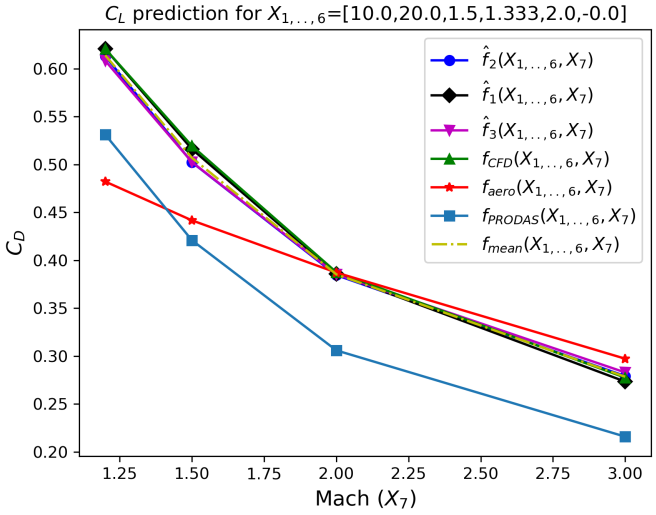


Figure 3.5: Prediction of C_L coefficient for a projectile configuration $X_{1,..,6}$ not in the training dataset at different Mach numbers X_7, using the models f_{mean} (yellow), $\hat{f}_3$ (violet), $\hat{f}_2$ (blue dots), $\hat{f}_1$ (black) and comparison with the target value f_{CFD} (green) and the lower fidelity models f_{aero} (red) and f_{PRODAS} (blue square)

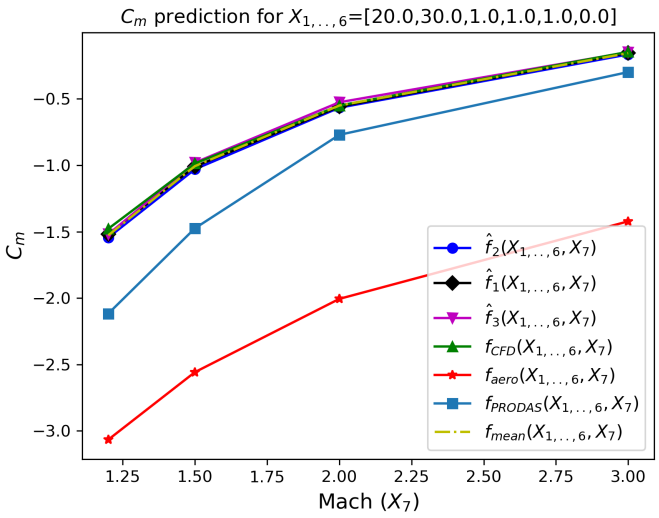


Figure 3.6: Prediction of C_m coefficient for a projectile configuration $X_{1,..,6}$ not in the training dataset at different Mach numbers X_7, using the models f_{mean} (yellow), $\hat{f}_3$ (violet), $\hat{f}_2$ (blue dots), $\hat{f}_1$ (black) and comparison with the target value f_{CFD} (green) and the lower fidelity models f_{aero} (red) and f_{PRODAS} (blue square)

Figures 3.4, 3.5 and 3.6 provide a visual representation of some of the predictions using $\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$ and f_{mean} for a given projectile configuration $X_{1,..,6}$ at different values of Mach numbers X_7. The predictions are compared to the target f_{CFD} as well as f_{aero} and f_{PRODAS} to see the improvement brought by $\hat{f}_2$ and $\hat{f}_1$ compared to f_{aero} and f_{PRODAS} respectively. As shown on these figures, the coefficients obtained from f_{PRODAS} or f_{aero} are far from the target f_{CFD}. However, using the multi-fidelity approach allows estimating efficiently f_{CFD} from f_{PRODAS} or f_{aero}.

3.4 Conclusion

The goal in this chapter was to build models capable to estimate aerodynamic coefficients., in order to reduce the cost associated to their acquisition. We presented two approaches which can be used to estimate them depending on the available dataset. The first approach learns only from the data of the highest fidelity source whereas the second one combines data from higher and lower fidelity sources to predict the coefficients. Both approaches show competitive performances and using the average of all the estimates provides better performances than any individual estimate. The approach can be improved by, *e.g.*, combining the models smarter than a simple average, for example based on the uncertainty of each estimate. It would also be interesting to see how the multi fidelity approach performs on a smaller dataset from experiments for example.

In the next chapter we will see how and why having a model which estimates aerodynamic coefficients can be crucial for the optimization of a projectile.

Chapter 4

Projectile Shape Optimization

The models identified in the previous chapter are not only used to predict aerodynamic coefficients but can also be employed to find the optimal geometrical configuration of a projectile based on criteria related to these coefficients. Such problems are known as aerodynamic design in the literature. As mentioned in the previous chapter, obtaining aerodynamic coefficients is an expensive task, which thus makes the optimization problem expensive to solve. However, by leveraging the models presented in the previous chapter, and the optimization procedures outlined in Chapter 2, the optimization problem can be solved more efficiently. The aim of this chapter is thus to see how this can be used to find the optimal configuration of a finned projectile for a given flight scenario. First, we begin with an overview on aerodynamic design and how such problems are solved in the literature in Section 4.1. Secondly, the aerodynamic design problem in this thesis, is presented in Section 4.2. Then we propose two methods to solve this problem : the Neural Network Smart grid based approach (see Section 2.3 of Chapter 2) in Section 4.3 and Bayesian Optimization in Section 4.4. We then present a special case where we can only perform one experiment in order to find the optimal solution in Section 4.5. The final section, Section 4.6, is dedicated to the application of the procedures to the aerodynamic design problem.

Contents

4.1 Aerodynamic design 60

4.2 Aerodynamic design problem 61

4.3 Optimization using Neural Networks as surrogate 65

4.4 Optimization using Bayesian Optimization 66

4.5 Possibility to evaluate only one configuration 67

4.6 Results 68

4.6.1 Setup and initial database 68

4.6.2 Adding one additional configuration 69

4.6.3 Results using Bayesian Optimization 72

4.6.4 Results using the Neural Network as surrogate model 72

4.6.5 Comparison to alternative approaches 76

4.7 Conclusion 77

4.1 Aerodynamic design

In many aerospace applications, the flight behaviour highly depends on the shape, the dimension, the materials,... of the considered flying vehicles. Consequently, the flight behaviour can be optimized by adjusting these quantities. In the aerospace literature, such a problem is known as aerodynamic design or aerodynamic optimization (see, *e.g.*, Driver and Zingg (2007)).

In this chapter, we will perform such task for (gun-launched) projectiles. In particular, the geometrical configuration of a rectangular finner which is a special case of the projectile studied in Section 3.3 of Chapter 3, see also (Dupuis and Hathaway, 1997), will be optimized to obtain an optimal flight behaviour for a flat fire trajectory, *i.e.*, for the case where the finner is launched with an initial total angle of attack equal to zero. The flight behaviour will be here deemed optimal when the projectile reaches its target in a minimum time.

As will be shown in the sequel, this aerodynamic optimization problem can be formulated as a constrained optimization problem having as decision variable the geometrical configuration X of the projectile and involving a number of the so-called stability derivatives of the projectile, which are functions of the geometry X of the projectile and of its Mach number M (see Chapter 3). Our goal in this chapter will be to solve this problem using the identification and optimization procedures described in the previous chapters.

This constrained optimization problem is a nonlinear optimization problem involving the static nonlinear function $F(X, M)$ relating X and M on the one hand and the stability derivatives on the other hand. As mentioned in Chapter 3, two approaches are generally used to determine the stability derivatives for a given configuration, *i.e.*, to evaluate $F(X, M)$. The first one is an experimental one involving wind-tunnel experiments or free flight tests and the second one is to perform high-fidelity simulations. Evaluating $F(X, M)$ for a given configuration using these approaches is thus either financially costly or time-consuming. In other words, the absence of simple and accurate models for the complex function $F(X, M)$ makes it extremely difficult to address the nonlinear optimization problem discussed in the previous paragraph using, *e.g.*, a classic gradient-based algorithm.

However as seen, in the Chapter 3, if we have access to a dataset containing different measurements of stability derivatives associated to different configurations X at different Mach numbers M, we can use it to derive a simple mathematical model $\hat{F}(X, M)$ of the static function $F(X, M)$. This simple mathematical model, which is also called *surrogate model* in the literature (Forrester et al., 2008), allows predicting $F(X, M)$ for any values of X and M in a fast and efficient way. Therefore, it can be used to solve the considered optimization problem via, *e.g.*, gradient-descent.

As discussed in Chapter 2, if the database does not cover the space of possible configurations X in a sufficient manner, the model derived from this database may be inaccurate and the optimal geometrical configuration deduced based on that model may not have the desired properties. Nevertheless, if we have the possibility of determining, via an experiment or via a simulation software, the values of $F(X, M)$ for a number N_{sup} of additional configurations X, we can use it to improve the accuracy of the model $\hat{F}(X, M)$ in regions which are interesting for the optimization.

As seen in Chapter 2, this is usually achieved using approaches balancing the exploration and exploitation objectives. In Chapter 2 two approaches which can be used to achieve this trade-off

were presented. First, Bayesian Optimization can be employed to determine which configurations to evaluate in order to find the optimal solution. This method has already been applied to address aerodynamic optimization problems for other flying vehicles than gun-launched projectiles as presented in (Jeong et al., 2005; Renganathan et al., 2021; Arnoult et al., 2020)). In (Jeong et al., 2005), Bayesian Optimization is applied to find the optimal design of an airfoil which maximises the lift to drag ratio, using the expected improvement acquisition function (2.11). The optimal shape of airfoil which minimises the drag subject to an equality constraint on the lift coefficients, is also searched using Bayesian Optimization in (Renganathan et al., 2021). They propose to increase the accuracy of the surrogate model by using a trained Neural Network as prior to the Gaussian Process model. Finally, in (Arnoult et al., 2020), the optimal geometry of control surfaces of a missile to reach a desired impact point are searched using Bayesian Optimization. Note that in the paper (Arnoult et al., 2020), a Neural Network is used to predict the aerodynamic coefficients of the missile which allows calculating the trajectories. However, the objective function that Bayesian Optimization solves is modelled by a Gaussian Process.

Bayesian Optimization can thus be used to solve the problem above. However, as shown in Chapter 2, Neural Networks can also be used in procedures similar to Bayesian Optimization. Moreover, as shown in Chapter 3 Neural Networks can excel in modelling aerodynamic coefficients which allows using an accurate model for the optimization procedure.

We therefore propose to solve this aerodynamic design problem using two approaches: Bayesian Optimization and the NNSGO procedure described in Section 2.3 of Chapter 2, involving the identification of a Neural Network. Before applying these procedures, we begin by presenting the aerodynamic design problem in detail in the following section.

4.2 Aerodynamic design problem

In this chapter we apply the optimization procedures described in chapter 2 to solve an *aerodynamic design* problem where we have to find the optimal dimensions of a projectile. We begin by presenting the problem in this section.

We here consider a so-called rectangular finner of caliber 28 mm with four rectangular fins which is a special case of the projectile studied in Section 3.3 of Chapter 3. Compared to the configuration studied in Section 3.3 of Chapter 3, the fins of the rectangular finner can only have rectangular shapes. This means that the Fins Tip Length and the Fins Chord Length, see Table 3.1 and Figure 3.3, will always be equal. The other difference is that the edge of the fins can be placed anywhere on the body of the projectile (excluding the nose), and does not necessary start at the trail edge of the body as in the study case in Section 3.3 of Chapter 3. Such a projectile can be built with different geometrical configurations, *i.e.*, for different values of the five parameters[6] X_1, X_2, ..., X_5 defined in Table[7] 4.1 and represented in Figure 4.1. X_1, X_3, X_4, X_5 will be expressed in calibers (1 calibre = 28 mm) and X_2 in degrees.

[6]Note that we will suppose that the building material of the finner is homogeneous. Its center of gravity is thus entirely determined by the five parameters in Table 4.1. We can also adjust its weight by changing the used material

[7]The values of X_1, X_2, ..., X_5 in this table are different from the ones presented in Table 3.1.

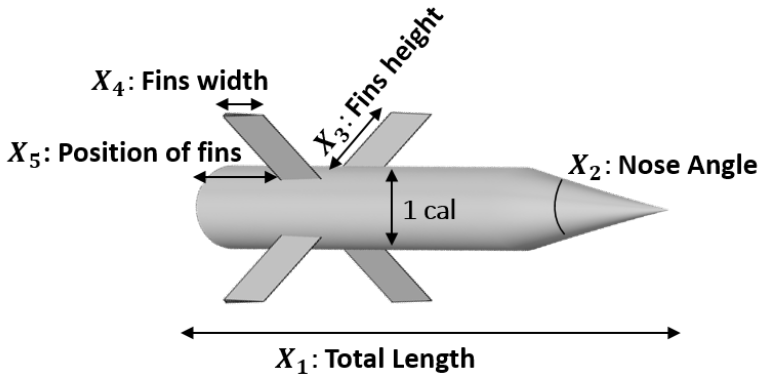


Figure 4.1: Finner projectile with rectangular fins

Table 4.1: Main characteristics of a rectangular finned projectile

Characteristic X	Design Parameter	Unit
X_1	Total Length	calibre
X_2	Nose Angle	degrees
X_3	Fins Height	calibre
X_4	Fins Width	calibre
X_5	Position of fins	calibre

We wish to determine the geometrical configuration of the finner in such a way that it reaches its target in a minimum time when the finner is fired with an initial velocity M_{max}, *e.g.*, Mach 5, and then flies towards its target with a total angle of attack of zero.

To address this problem, we have to consider the stability derivatives of the finner which are coefficients associated to the forces and moments acting on the projectile during its flight. As mentioned in Section 3.1.2 of Chapter 3, they depend on both the geometrical configuration of the finner, determined by the vector $X = [X_1, X_2, \ldots, X_5]^T$ and its velocity, *i.e.*, the Mach number M. In particular, to minimize the time to reach the target, it is well known that one of the solutions is to minimize the drag coefficient of the projectile (McCoy, 1999). Since we suppose that the finner flies with a total angle of attack equal to zero°, the drag coefficient is equal to the *axial force coefficient first order stability derivative at zero yaw* C_{A_0}, one of the stability derivatives, see Section 3.1.2 in Chapter 3. Since C_{A_0} not only depends on the geometrical configuration X, but also on the Mach number M, we will in fact minimize the average value of C_{A_0} over a velocity range $[M_{min}\ M_{max}]$ with M_{min} and M_{max} chosen in such a way that the interval $[M_{min}\ M_{max}]$ covers the possible velocities of the finner during its flight (*e.g.*, $M_{min} = 2$ and $M_{max} = 5$).

The stability derivative C_{A_0} is not the only stability derivative that is important for an optimal flight. We will also wish to guarantee strong flight stability by making sure that the projectile will be dynamically and statically stable (McCoy, 1999), (Albisser, 2015).

- To ensure static stability, the *pitch moment coefficient slope* C_{m_α}, see Section 3.1.2 in Chapter 3, will have to be smaller than $C_{m_\alpha}^{max}$;

- To ensure dynamic stability the *pitch damping moment coefficient* C_{m_q}, see Section 3.1.2 in Chapter 3, will have to remain smaller than $C_{m_q}^{max}$.

These two conditions have to be respected for all velocities in the interval $[M_{min}\ M_{max}]$.

Our objective is therefore to determine the geometrical configuration of the finner, *i.e.*, the value of the five parameters in Table 4.1 that leads to the lowest average value of the drag coefficient C_{A_0} while guaranteeing that, at all times, C_{m_α} and C_{m_q} remains smaller than $C_{m_\alpha}^{max}$ and $C_{m_q}^{max}$, respectively.[8]

Let us formulate this objective in a mathematical form. For this purpose, we denote by :

$$Y = (C_{A_0}, C_{m_\alpha}, C_{m_q})^T, \tag{4.1}$$

the vector containing the stability derivatives considered in this study. Since Y is a function of X and M, we can therefore use the following notation:

$$Y = F(X, M), \tag{4.2}$$

where F is a static function taking as inputs the vector X and the scalar M and giving as output the vector Y. For the ith entry of Y $(i = 1, ..., 3)$, the notation $Y_i = F_i(X, M)$ will also be used.

For a given geometrical configuration X, the average value $J(X)$ of C_{A_0} over the interval $[M_{min}\ M_{max}]$ can thus be expressed as:

$$J(X) = \frac{1}{M_{max} - M_{min}} \int_{M_{min}}^{M_{max}} F_1(X, M) dM. \tag{4.3}$$

Likewise, the stability constraints can be expressed by:

$$F_2(X, M_{max}) \leq C_{m_\alpha}^{max} \text{ and } F_3(X, M_{max}) \leq C_{m_q}^{max}. \tag{4.4}$$

We here use the fact that, for rectangular finners in supersonic regime, both $C_{m_\alpha} = Y_2$ and $C_{m_q} = Y_3$ always reach their maximum at the highest velocity M_{max} which is also the initial velocity of the flight.

The optimization problem considered in this chapter can therefore be formulated as follows:

$$X^* = \arg \max_{X \in \mathcal{X}} -J(X) \tag{4.5a}$$

$$\text{s.t. } F_2(X, M_{max}) \leq C_{m_\alpha}^{max} \text{ and } F_3(X, M_{max}) \leq C_{m_q}^{max} \tag{4.5b}$$

[8] The quantities C_{A_0}, C_{m_α}, C_{m_q} and M are dimensionless.

Note that we here formulate the optimization problem (4.5) as a maximization to facilitate the presentation in the sequel. Minimizing $J(X)$ is indeed equivalent to maximizing $-J(X)$.

In (4.5), $\mathcal{X}$ represents the allowed search space: the set $\mathcal{X}$ can, *e.g.*, contain all geometrical configurations X that are achievable in practice or, alternatively, the set $\mathcal{X}$ can also be a subset of the latter if, for some reasons, we wish to restrict the possible geometrical configurations.

Solving the optimization problem (4.5) is not straightforward. Indeed, the static function F is unknown, *i.e.*, there is no readily available physical model for this function (at least not one which does not require computationally intensive simulations). However, there exist databases containing values of this unknown static function, *i.e.*, a set of data giving, for a number of geometry configurations, the corresponding vector Y for different Mach numbers. As mentioned in Section 3.1, these databases are generally determined using experimental campaigns or computationally intensive simulations.

We assume here that we have access to such a database. It contains N_X different geometrical configurations for which the stability derivatives Y have been computed for N_M different Mach numbers in the interval $[M_{min}\ M_{max}]$ [9]. In other words, the database contains the value of the vector $Y = F(X^k, M^l)$ for all pairs (X^k, M^l) with M^l $(l = 1, \ldots, N_M)$ different values of M in the interval $[M_{min}\ M_{max}]$ and with X^k $(k = 1, \ldots, N_X)$ different geometrical configurations. Let us denote by $\mathcal{D}_X$ the set containing these N_X geometrical configurations for which we know Y at N_M different Mach numbers, *i.e.*, $\mathcal{D}_X = \{X^k | k = 1, \ldots, N_X\}$.

Using this database, $J(X)$ can be approximated for all $X \in \mathcal{D}_X$ using:

$$J(X) = \frac{1}{N_M} \sum_{l=1}^{N_M} F_1(X, M^l). \tag{4.6}$$

From now onwards, the expression (4.6) for $J(X)$ will be used and we will assume that M_{max} is a velocity for which Y is given in the database. Then, by restricting attention to the geometrical configurations $X \in \mathcal{X}$ that are present in $\mathcal{D}_X$, we can derive an estimate $X^*_{\mathcal{D}_X}$ of the solution X^* of the optimization problem (4.5):

$$X^*_{\mathcal{D}_X} = \arg \max_{X \in \mathcal{D}_X \cap \mathcal{X}} -J(X) \tag{4.7a}$$

$$\text{s.t. } F_2(X, M_{max}) \leq C^{max}_{m_\alpha} \text{ and } F_3(X, M_{max}) \leq C^{max}_{m_q} \tag{4.7b}$$

It is to be noted that $\mathcal{D}_X \cap \mathcal{X} = \mathcal{D}_X$ if $\mathcal{X}$ is the set of all achievable geometrical configurations. The solution $X^*_{\mathcal{D}_X}$ of the optimization problem (4.7) is only an approximation of X^*. However, it is the best approximation that can be derived given the information in the database. It is also clear that the larger the database, the better the approximation $X^*_{\mathcal{D}_X}$ of X^*. In this study, the objective is to improve $X^*_{\mathcal{D}_X}$ by extending the set $\mathcal{D}_X$, *i.e.*, the set of geometrical configurations for which the value of Y, at different Mach numbers, is known. Extending the set $\mathcal{D}_X$ comes with a cost. Indeed, if the objective is to know the value of Y at different Mach numbers for a

[9] The database may of course also contain the value of the stability derivatives for velocities outside this interval.

geometrical configuration $X \notin \mathcal{D}_X$, a flight experiment has to be carried out with a projectile having that particular geometrical configuration or, alternatively, a computationally intensive simulation has to be performed (see Section 3.1 of Chapter 3). Consequently, we wish to extend the set $\mathcal{D}_X$ in a smart way, *i.e.*, intelligently determining which configuration(s) will be added to $\mathcal{D}_X$ and for which a flight experiment or a simulation has to be performed in order to find the optimal configuration.

If we have the possibility to evaluate multiple configurations in order to improve the dataset, we can use the approaches presented in Chapter 2, to choose which configurations to evaluate. To achieve this, we can derive a model for $F(X, M)$ using the available dataset and use it to estimate $J(X)$, $F_2(X, M_{max})$ and $F_3(X, M_{max})$. Based on these estimates, one could then use a procedure balancing the exploitation and exploration objectives to decide which configurations to evaluate. In the following sections we will see how the procedures presented in Chapter 2 can be used to achieve this and thus solve the aerodynamic design problem. First in Section 4.3, we see how the approach involving the use of a Neural Network described in Section 2.3.3 of Chapter 2 can be employed. Then in Section 4.4 we will see how Bayesian Optimization can also be used to solve this problem. Lastly, we will consider a special case in Section 4.5 where we can only evaluate one configuration to find the optimum to see if the other two methods are essential.

4.3 Optimization using Neural Networks as surrogate

In this section we describe how the procedure described in Section 2.3 of Chapter 2 can be used to choose the geometrical configurations to evaluate. To achieve this, the database is used to derive a Neural Network model, see Section 1.2 of Chapter 1, $\hat{F}$ of the unknown function F (see (4.2)):

$$\hat{Y} = \hat{F}(X, M). \tag{4.8}$$

Using this model $\hat{F}$, the value of Y can now be predicted for configurations X that are not in the database. In other words, for a configuration $X \notin \mathcal{D}_X$, we can estimate $F_2(X, M_{max})$ and $F_3(X, M_{max})$ by, respectively, the second and third entries of $\hat{F}(X, M_{max})$, *i.e.*, $\hat{F}_2(X, M_{max})$ and $\hat{F}_3(X, M_{max})$, and we can estimate $J(X)$ (see (4.6)) by :

$$\hat{J}(X) = \frac{1}{N_M} \sum_{l=1}^{N_M} \hat{F}_1(X, M^l) \tag{4.9}$$

We then define the acquisition function $A(X)$ (see Section 2.3.3 in Chapter 2) for this aerodynamic design problem as:

$$A(X) = \lambda(X)\ A_{obj}(X), \tag{4.10}$$

where

$$A_{obj}(X) = -\hat{J}(X) + \beta\ \delta_{min}(X) + \rho, \tag{4.11}$$

with

$$\delta_{min}(X) = \min_{X_{DB} \in \mathcal{D}_X} \|X - X_{DB}\|, \tag{4.12}$$

where X_{DB} are the configurations X in the dataset $\mathcal{D}_X$.

$$\lambda(X) = \begin{cases} 2 - \delta_{min}^{scaled}(X) & \text{if } \hat{F}_2(X, M_{max}) \leq C_{m_\alpha}^{max} \text{ and } \hat{F}_3(X, M_{max}) \leq C_{m_q}^{max} \\ \delta_{min}^{scaled}(X) & \text{otherwise.} \end{cases} \tag{4.13}$$

The procedure described in Algorithm 2 can thus be used to decide which geometrical configurations X to evaluate, with the acquisition function (4.10) and the considerations in Section 2.3.3 of Chapter 2.

4.4 Optimization using Bayesian Optimization

In this section we apply the procedure described in Section 2.2.2 of Chapter 2 to solve the optimization problem (4.5). Unlike in Section 4.3, where a Neural Network is employed to identify $F(X, M)$ and used to reconstruct the objective function $J(X)$ and the constraints $F_2(X, M_{max})$ and $F_3(X, M_{max})$, we here have to identify a separate model for the objective function and the constraints. This is done to have access to the uncertainties associated to the design variable X and not both X and M. In a more general case, when performing Bayesian Optimization for constrained Optimization problems, the objective function and the constraints are usually identified with separate Gaussian Process models to reduce the costs associated to identifying multivariate Gaussian Processes (see Section 1.3.2 of Chapter 1).

To solve this aerodynamic design problem using Bayesian Optimization, the first step thus consists in modelling the objective function and constraints with Gaussian Processes. In this sense, three Gaussian Process models are identified for:

1. the objective function corresponding to the average value of the drag coefficient for different Mach numbers, see (4.6);
2. the value of the C_{m_α} coefficient at Mach M_{max}, *i.e* $F_2(X, M_{max})$;
3. the value of the C_{m_q} coefficient at Mach M_{max}, *i.e* $F_3(X, M_{max})$.

For the three Gaussian Process models, we use the Matern kernel described in (1.16) which usually allow estimating many functions. Its hyper-parameters are tuned using the log-marginal (1.17). The three Gaussian Process models thus provide us with an estimate of $J(X)$:

$$J(X) \sim \mathcal{N}\left(\mu_J(X), \sigma_J^2(X)\right), \tag{4.14}$$

and an estimate of $F_2(X, M_{Max})$ and $F_3(X, M_{Max})$:

$$F_2(X, M_{max}) \sim \mathcal{N}\left(\mu_{F_2}(X), \sigma_{F_2}^2(X)\right), \tag{4.15a}$$
$$F_3(X, M_{max}) \sim \mathcal{N}\left(\mu_{F_3}(X), \sigma_{F_3}^2(X)\right). \tag{4.15b}$$

With the estimate of J_X (4.14), we construct the expected improvement acquisition function (2.11) as:

$$EI(X) = (-\mu_J(X) - (-J(X_+)) - \xi)\Phi(Z) + \sigma_J(X)\phi(Z), \tag{4.16}$$

where:

$$Z = \frac{-\mu_J(X) - (-J(X_+)) - \xi}{\sigma_J(X)}, \tag{4.17}$$

and $J(X_+)$ is the current known lowest value of the average drag in the dataset which respects the constraints. Using the estimates of $F_2(X, M_{Max})$ and $F_3(X, M_{Max})$, we can determine the probabilities that the constraints in (4.5) are respected:

$$P_{F_2}(X) := \Pr[\mu_{F_2}(X) \leq C_{m_\alpha}^{max}], \tag{4.18a}$$
$$P_{F_3}(X) := \Pr[\mu_{F_3}(X) \leq C_{m_q}^{max}]. \tag{4.18b}$$

As mentioned in Section 2.2.2 in Chapter 2, $P_{F_2}(X)$ and $P_{F_3}(X)$ will both be univariate Gaussian cumulative distribution functions. $F_2(X, M_{max})$ and $F_3(X, M_{max})$ are considered to be independent thus the probability that both constraints are respected is $P_{F_2}(X) \times P_{F_3}(X)$. The final acquisition function is thus:

$$A(X) = P_{F_2}(X)P_{F_3}(X)EI(X), \tag{4.19}$$

with $EI(X)$ as in (4.16). The points to evaluate can thus be found by following Algorithm 1, using the acquisition function (4.19) and the considerations in Section 2.2.2 of Chapter 2.

4.5 Possibility to evaluate only one configuration

The procedures described in Section 4.3 and Section 4.4 are particularly useful when we have the possibility to perform multiple experiments or simulations in order to determine the coefficients. In the case where we only have the possibility to perform only one experiment, we follow the procedure described in the Section 2.1 in Chapter 2. As in the previous sections, we use the available dataset to derive estimates for $J(X)$, $F_2(X, M_{max})$ and $F_3(X, M_{max})$ using data-driven models.

Given the estimates $\hat{J}(X)$, $\hat{F}_2(X, M_{max})$ and $\hat{F}_3(X, M_{max})$, the most promising configuration X_{new} to evaluate is:

$$X_{new} = \arg \max_{X \in \mathcal{X}} -\hat{J}(X), \tag{4.20a}$$
$$\text{s.t. } \hat{F}_2(X, M_{max}) \leq C_{m_\alpha}^{max} \text{ and } \hat{F}_3(X, M_{max}) \leq C_{m_q}^{max}. \tag{4.20b}$$

In other words, X_{new} is chosen as the geometrical configuration that would be equal to X^* (4.5) if the model would be perfect, *i.e.*, $\hat{F}(X, M) = F(X, M)$ for all X and M. If the optimization problem (4.20) leads to a configuration $X_{new} \notin \mathcal{D}_X$, *i.e.*, a configuration for which the model $\hat{F}$ predicts that X_{new} outperforms $X^*_{\mathcal{D}_X}$ (4.7), this new configuration X_{new} should be tested via

an experiment or a simulation to obtain the actual values of $F(X_{new}, M^l)$ for the velocities M^l $(l = 1, \ldots, N_M)$. Afterwards, if we have that:

$$F_2(X_{new}, M_{max}) \leq C_{m_\alpha}^{max} \text{ and } F_3(X_{new}, M_{max}) \leq C_{m_q}^{max}, \tag{4.21a}$$

$$J(X_{new}) < J(X^*_{\mathcal{D}_X}), \tag{4.21b}$$

then X_{new} is the new optimal configuration. However, it may happen that one of the conditions in (4.21) is not satisfied. This occurs when the estimated models $\hat{J}$, $\hat{F}_2$ and $\hat{F}_3$ are far from J, F_2 and F_3 in the vicinity of X_{new} respectively. In this case, the procedures in Section 4.3 and Section 4.4 should be considered.

Remark: Note that this procedure, is a special case of Algorithm 2, where the number of evaluations is $k = 1$. In this case, we only perform an exploitation of the model. However, any model other than a Neural Network, could be used to identify the objective function and the constraints.

4.6 Results

4.6.1 Setup and initial database

In this section, we present the setup and initial database used to determine the optimal geometrical configuration X of a rectangular finner (see Figure 4.1) for the flight scenario described in Section 4.2 with $M_{min} = 2$ and $M_{max} = 5$. The search set $\mathcal{X}$ is here limited to enable that the projectile can be fired by a specific field gun (see Table 4.2 for the formal definition of the set $\mathcal{X}$).

Table 4.2: Definition of the set $\mathcal{X}$ for each characteristic X_i of the geometry (X_1, X_3, X_4 and X_5 are expressed in calibers and X_2 in degrees)

Dimension X	Minimum	Maximum
X_1 : Total Length	10	25
X_2 : Nose Angle	10	34.5
X_3 : Fins Height	0.5	2.5
X_4 : Fins Width	0.5	1.72
X_5 : Position of fins	0	1

For the stability constraints in (4.5), the upper bounds are fixed to the following realistic values:

- $C_{m_\alpha}^{max}$ is fixed at -10;
- $C_{m_q}^{max}$ is fixed at -100.

During the thesis, there was no possibility to conduct many Computational Fluid Dynamics (CFD) evaluations or experiments in order to determine aerodynamic coefficients for a given configuration due to various constraints. Consequently, the optimization procedures in Section 4.3, Section 4.4 and Section 4.5 were tested using a simulation environment as ground truth. The simulation code, named *PRODAS*, *i.e.*, *Projectile Rocket Ordnance Design & Analysis System* (See *www.prodas.com* for further informations), corresponds to the lower fidelity model which provides f_{PRODAS} (3.8) used in Section 3.3.2 of Chapter 3. It allows computing the values of the coefficients for a given geometrical configuration in a cheaper and faster manner than CFD simulations and experiments. However, the results from this procedure will be validated with CFD simulations in the future to ensure accuracy and effectiveness.

For the type of finner projectiles studied and the flight scenario considered, we have a database giving the vector Y for $N_X = 324$ different configurations X and for $N_M = 8$ different Mach numbers in the interval [2 5]. This database has been generated using the simulation code *PRODAS*.

Note that out of the $N_X = 324$ different configurations in $\mathcal{D}_X$, only 144 are in the restricted set $\mathcal{X}$. Among these configurations $X \in \mathcal{X} \cap \mathcal{D}_X$, none has a value of $F_2(X, M) \leq -10$. Consequently, $X^*_{\mathcal{D}_X}$ does not exist. However, among the configurations $X \in \mathcal{X} \cap \mathcal{D}_X$ that almost respect the constraints, the one leading to the smallest value of $J(X)$ is the configuration :

$$X^*_{\mathcal{D}_{X,almost}} = (20, 20, 1.5, 1.5, 0)^T. \tag{4.22}$$

More specifically for $X = X^*_{\mathcal{D}_{X,almost}}$, we have:

$$J(X) = 0.45 \tag{4.23a}$$
$$F_2(X, M = 5) = -8 \tag{4.23b}$$
$$F_3(X, M = 5) = -1400, \tag{4.23c}$$

which shows that C_{m_α} is only slightly too high.

In the next sections we present the results, when we apply the optimization procedures described in Section 4.3, Section 4.4 and Section 4.5, to see if we can obtain better configurations than $X^*_{\mathcal{D}_{X,almost}}$. We begin with the case where only one configuration can be evaluated.

4.6.2 Adding one additional configuration

To present the results when only one configuration can be evaluated, we choose to use a Neural Network (Multi Layer Perceptron) as surrogate model. We here choose to identify only one model for the constraints and objective function as explained in Section 4.3. As a first step, we split the data in $\mathcal{D}_X$ to determine a suitable structure for the Neural Network (see the Section 1.2 of Chapter 1). This leads to a Neural Network with 1 hidden layer, 128 neurons and a *reLu* activation function on each neuron output. The parameters of this Neural Network model are identified using the procedures described in Chapter 1, Section 1.2. The different identification criteria are solved using the machine learning library *Scikit-learn* (Buitinck et al., 2013).

The Neural Network model provides an estimate $\hat{F}(X, M)$, which can be used to reconstruct $\hat{J}(X)$, $\hat{F}_2(X, M = 5)$ and $\hat{F}_3(X, M = 5)$. These estimates can now be used in the procedure of Section 4.5. The new configuration X_{new} is thus determined using the optimization problem (4.20) that we here solve using the *Sequential Least Square Programming* algorithm given in (Virtanen et al., 2020). This leads to:

$$X_{new} = (12.6859, 10, 1.34, 1.72, 0.43)^T. \tag{4.24}$$

This solution X_{new} is such that the model $\hat{F}(X, M)$ predicts a value of 0.32 for $J(X_{new})$, *i.e.*, $\hat{J}(X_{new}) = 0.32$ (see (4.9)). By construction, we have also that $\hat{F}_2(X_{new}, M = 5) \leq -10$ and $\hat{F}_3(X_{new}, M = 5) \leq -100$. More precisely:

$$\hat{F}_2(X_{new}, M = 5) = -10 \tag{4.25a}$$
$$\hat{F}_3(X_{new}, M = 5) = -419. \tag{4.25b}$$

See the blue curve in Figures 4.2, 4.3 and 4.4 for other characteristic values of $\hat{F}_i(X_{new}, M^l)$ $(i = 1, \dots 3, l = 1, \dots, N_M)$.

Let us now verify whether the predictions of $\hat{F}$ about X_{new} are confirmed in reality. For this purpose, we use *PRODAS* to compute $Y^{new} = F(X_{new}, M)$ at different Mach numbers M (see the green curve in Figures 4.2, 4.3 and 4.4 for characteristic values).

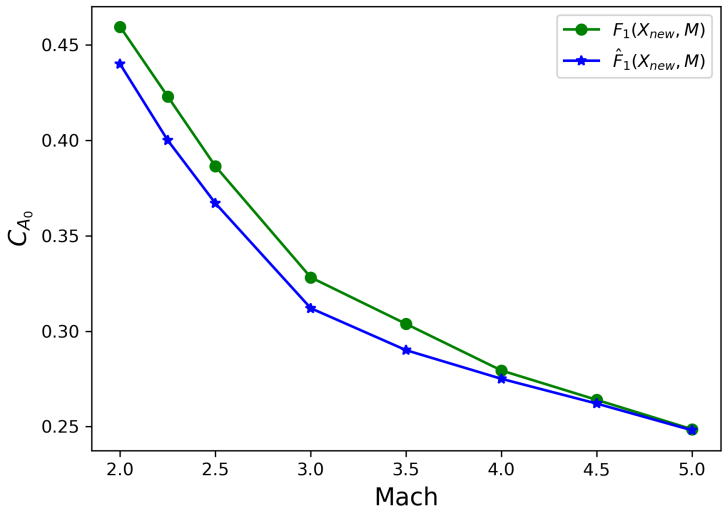


Figure 4.2: Neural Network prediction $\hat{F}_1(X_{new}, M)$ (blue) and $F_1(X_{new}, M)$ evaluated with PRODAS (green) for $X_{new} = (12.6859, 10, 1.34, 1.72, 0.43)^T$ and for characteristic values of $M \in [2\ 5]$.

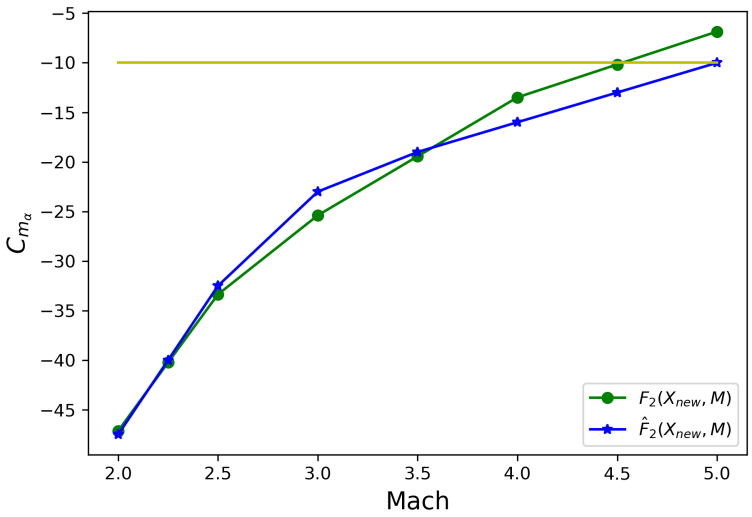


Figure 4.3: Neural Network prediction $\hat{F}_2(X_{new}, M)$ (blue) and $F_2(X_{new}, M)$ evaluated with PRODAS (green) for $X_{new} = (12.6859, 10, 1.34, 1.72, 0.43)^T$ and for characteristic values of $M \in [2\ 5]$. The yellow line represents the threshold $C_{m_\alpha}^{max} = -10$.

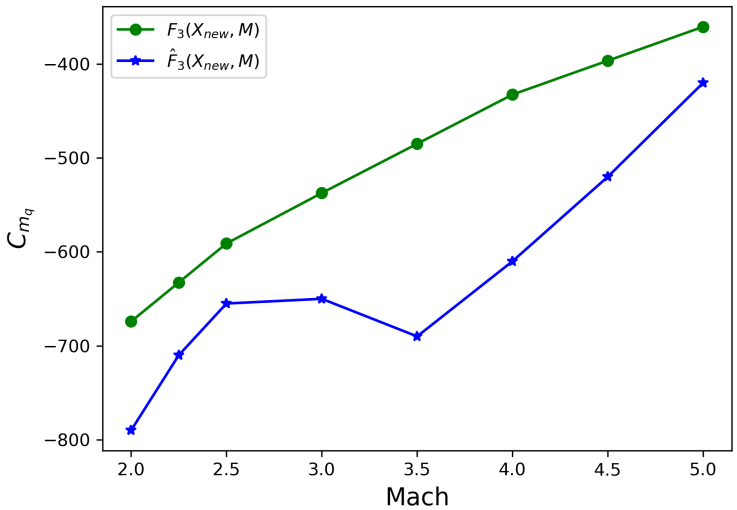


Figure 4.4: Neural Network prediction $\hat{F}_3(X_{new}, M)$ (blue) and $F_3(X_{new}, M)$ evaluated with PRODAS (green) for $X_{new} = (12.6859, 10, 1.34, 1.72, 0.43)^T$ and for characteristic values of $M \in [2\ 5]$.

This allows observing that:

$$J(X_{new}) = 0.33 \text{ (see (4.6))} \tag{4.26a}$$
$$F_2(X_{new}, M = 5) = -6 \tag{4.26b}$$
$$F_3(X_{new}, M = 5) = -360 \tag{4.26c}$$

In other words, as opposed to what was predicted by the model $\hat{F}$, the constraint on C_{m_α} is not respected for the configuration X_{new}. This can certainly be explained by the fact that the model $\hat{F}$ has been trained with too few data in the vicinity of X_{new}. We must thus follow the procedures described in Section 4.4 and 4.3, to search for which points to test in order to increase the accuracy of the model and get close to the optimum.

4.6.3 Results using Bayesian Optimization

In this section, we present the results when we follow the procedure described in Section 4.4. The Gaussian Process models and the optimization of the parameters of their kernels are performed using the *Scikit-learn* (Buitinck et al., 2013) machine learning library. The acquisition function is optimized using the *differential evolution* (Storn and Price, 1997) algorithm in its *Scipy* implementation (Virtanen et al., 2020). Since there are no configuration which respect the constraints in the initial data set, the value of $J(X_+)$ in (4.16) is chosen as the value of $J(X^*_{\mathcal{D}_{X,almost}})$ which is the configuration with the lowest drag that almost respects the constraints in the initial dataset.

The number of configurations to evaluate is fixed to 20. After performing Bayesian Optimization for 20 iterations, the dataset now contains $N_X + 20$, 344 configurations X, for which the value of $F(X, M)$ has been computed. The extended database using Bayesian Optimization, $\mathcal{D}_{X,BO}$ allows determining the optimum X^*_{BO} as the configuration with the lowest drag that respects the constraints in the new extended dataset. The configuration X^*_{BO} is:

$$X^*_{BO} = (12.6, 10.4, 1.3, 1.72, 0)^T \tag{4.27}$$

This configuration X^*_{BO} is such that :

$$J(X^*_{BO}) = 0.34 \tag{4.28a}$$
$$F_2(X^*_{BO}, M = 5) = -10 \tag{4.28b}$$
$$F_3(X^*_{BO}, M = 5) = -410 \tag{4.28c}$$

Therefore, using Bayesian Optimization to choose which 20 configurations to evaluate we are able to find a configuration which respects the constraints and that has a lower drag than the best configuration in the initial dataset (4.22). This configuration is found after 10 iterations. Among the other tested configurations 8 respect the constraints. The first configuration which respects the constraints is found at the 3rd iteration.

4.6.4 Results using the Neural Network as surrogate model

In the previous section we saw that Bayesian Optimization, and more particularly Gaussian Processes, can be used to select which points to evaluate to get close to the optimum. In this

section the results obtained when a Neural Network is used for this aerodynamic design problem following the procedure described in Section 4.3, are presented. The optimization of the acquisition function $A(X)$ (see (4.10)) in Step 1 of the `repeat loop` of Algorithm 2 is performed using the *dual annealing* algorithm (Xiang et al., 1997) in its *Scipy* implementation.

To apply Algorithm 2, two parameters in the definition of the acquisition function $A(X)$ have to be specified: the offset ρ and the parameter β balancing the exploitation and exploration objectives. While the choice of the offset ρ is not crucial, we here choose $\rho = 1$, the parameter β has to be chosen with care. Following the philosophy introduced in Section 2.3 of Chapter 2, β must be chosen in such a way that the term $\hat{J}(X)$ and $\beta\ \delta_{min}(X)$ have the same order of magnitude for all $X \in \mathcal{X}$. In our case, this means that β must be chosen in the interval [0.5 4]. As proposed in Section 2.3 of Chapter 2, we therefore choose to start with $\beta = 4$ in the first iteration of the *repeat loop* of Algorithm 2 and linearly decrease this value at each iteration in such a way that, at the last iteration, $\beta = 0.5$.

With this choice for ρ and β, Algorithm 2 delivers an extended database containing the values of Y for $N_X + 20 = 344$ configurations gathered in the set denoted as $X^*_{\mathcal{D}_{X,NN}}$. The extended set using the NNSGO procedure, $X^*_{\mathcal{D}_{X,NN}}$, allows determining the optimum X^*_{NN} as the configuration with the lowest drag that respects the constraints in the new extended dataset. This configuration is:

$$X^*_{NN} = (10.8, 13.42, 1.45, 1.72, 0)^T. \tag{4.29}$$

The configuration X^*_{NN} is such that :

$$J(X^*_{NN}) = 0.34 \tag{4.30a}$$
$$F_2(X^*_{NN}, M = 5) = -10 \tag{4.30b}$$
$$F_3(X^*_{NN}, M = 5) = -313 \tag{4.30c}$$

Using a Neural Network, we are therefore also able to find a configuration which is better than the initial configurations in the dataset. This configuration has the same average drag as the one found using Bayesian Optimization in Section 4.6.3. However, the optimal configuration X^*_{NN} is found after evaluating 20 configurations, compared to Bayesian Optimization where the optimal configuration was found after evaluating 11 configurations. In this case Bayesian Optimization *converges* faster than NNSGO. However, with the NNSGO procedure, the first configuration X which is evaluated is a feasible point, *i.e.*, a configuration which respects the constraints. In Bayesian Optimization such a configuration was found after three iterations. Moreover, among the 20 additional configurations that are added to the database using the NNSGO procedure, 14 configurations respect the stability constraints compared to 8 with Bayesian Optimization. Therefore in this example, both methods are able to find optima with the same value of the objective function. Bayesian Optimization converges faster to that value whereas the NNSGO procedure allows us to test more configurations which respect the constraints.

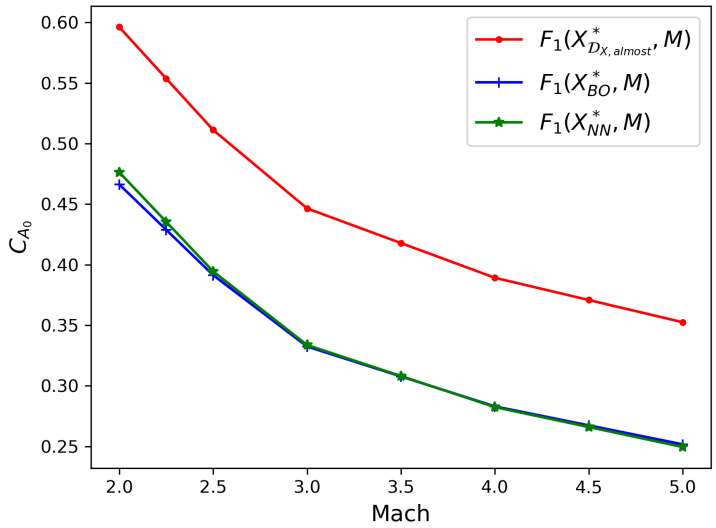


Figure 4.5: $F_1(X, M)$ (evaluated with PRODAS) for characteristic values of $M \in [2\ 5]$ and for three values of X, *i.e.*, $X = X^*_{BO}$ (blue), $X = X^*_{NN}$ (green) and $X = X^*_{\mathcal{D}_{X,almost}}$ (red).

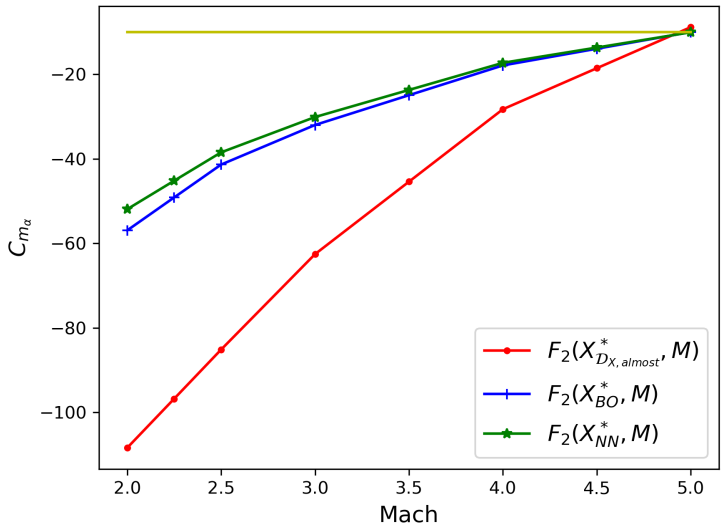


Figure 4.6: $F_2(X, M)$ (evaluated with PRODAS) for characteristic values of $M \in [2\ 5]$ and for three values of X, *i.e.*, $X = X^*_{BO}$ (blue), $X = X^*_{NN}$ (green) and $X = X^*_{\mathcal{D}_{X,almost}}$ (red). The yellow line represents the threshold $C^{max}_{m_\alpha} = -10$.

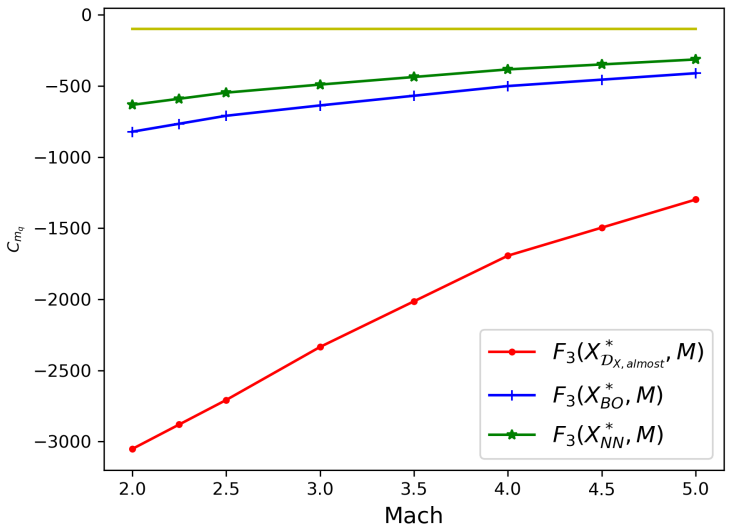


Figure 4.7: $F_3(X, M)$ (evaluated with PRODAS) for characteristic values of $M \in [2\ 5]$ and for three values of X, *i.e.*, $X = X^*_{BO}$ (blue), $X = X^*_{NN}$ (green) and $X = X^*_{\mathcal{D}_{X,almost}}$ (red).

Figures 4.5, 4.6 and 4.7 present the values of the aerodynamic stability derivatives w.r.t to the value of the Mach number for three different configurations X, $X = X^*_{BO}$ (blue), $X = X^*_{NN}$ (green) and $X = X^*_{\mathcal{D}_{X,almost}}$, respectively. As shown on Figure 4.5, the optimal configurations found using Bayesian Optimization and NNSGO, *i.e* X^*_{BO} and X^*_{NN} respectively, both have a better overall drag than the initially best configuration in the dataset $X^*_{\mathcal{D}_{X,almost}}$. Moreover, these two configurations, respect the stability constraint on C_{m_α}, see Figure 4.6, that was not originally respected with $X^*_{\mathcal{D}_{X,almost}}$.

If we compare the value of $F_1(X, M)$, *i.e.,* the value of the drag coefficient for a given Mach number M, for $X = X^*_{BO}$ and $X = X^*_{NN}$, we see that both configurations have approximately the same value of $F_1(X, M)$. For lower values of M, $F_1(X^*_{NN}, M)$ is larger than $F_1(X^*_{BO}, M)$ and for higher values of M, $F_1(X^*_{NN}, M)$ is slightly smaller than $F_1(X^*_{BO}, M)$. Therefore, both Bayesian Optimization and NNSGO approaches, provide configurations with similar values of the average drag $J(X)$, but which are obtained with different behaviours of $F_1(X, M)$ at different values Mach numbers M.

Robustness of Algorithm 2 w.r.t β

To analyse the robustness of the NNSGO procedure w.r.t the choice of the parameter β in Algorithm 2, we have run this algorithm with other choices of β in the interval [0.5 4]. More specifically, it is run for 20 iterations with different constant values of β at each iteration in the `repeat loop`. Table 4.3 presents the values of β considered, the values of the average drag J obtained with each value of β and the number of configurations which respect the constraints among the ones tested for each value of β.

Table 4.3: Robustness of Algorithm 2 w.r.t β

β	Average drag J	Number of configurations which respect the constraints
0.5	0.35	8
1	0.37	12
2	0.38	17
4	0.36	18

As shown in Table 4.3, when Algorithm 2 is performed with different choices of β, we can observe that they all lead to an optimal configuration respecting the stability constraints with a good average drag J. Moreover, among the 20 additional configurations that are added to the database via Algorithm 2, a relatively large number respects the stability constraints.

4.6.5 Comparison to alternative approaches

As shown above, the procedures introduced in Section 4.4 and Section 4.3 and implementing a trade-off between exploitation and exploration to generate 20 additional configurations for the database, deliver efficient results. To determine how efficient these results really are, let us compare the results obtained with these procedures with alternative approaches to generate the 20 additional configurations. Recall for comparison purpose that, using Bayesian Optimization and NNSGO to evaluate 20 additional configurations, 40% and 70% additional configurations respect the stability constraints respectively. Among these configurations respecting the constraints, we have projectiles for which the average drag is equal to 0.34.

Alternative approach 1 (Random generation). In this first alternative approach, we choose to generate, in a random manner, 60 new configurations $X \in \mathcal{X}$, *i.e.*, three times more than the number used in the Bayesian Optimization and NNSGO. Only 20% of the randomly generated configurations respect the constraints which is much smaller than in BO and NNSGO. Among these configurations respecting the constraints, the smallest J is 0.42, which is 20% larger than what is obtained with BO and NNSGO.

Alternative approach 2 (Exploration only). Instead of balancing exploitation and exploration, we choose in this second alternative approach to uniquely favour the exploration objective. This can, *e.g.*, be achieved by performing Algorithm 2 with an alternative definition for the acquisition function $A(X)$, *i.e.*, $A(X) = \delta_{min}(X)$. When we follow this approach to evaluate 20 additional configurations, the extended database does not improve the initial one. Indeed, none of the additional configurations respects the stability constraints.

Alternative approach 3 (Exploitation only). Instead of balancing exploitation and exploration, we choose in this third alternative approach to uniquely favour the exploitation objective. This can be achieved by performing a modified Algorithm 2 where, in step 1 of the `repeat loop`, X_{new} is determined using optimization problem (4.20). When we follow this approach for 20 iterations, we obtain 20 additional configurations of which only one respects the constraints. This configuration $X = (13.23, 10, 1.34, 1.72, 0)^T$ also achieves an average drag of 0.34. Since this particular configuration (the only one respecting the constraints) is obtained at iteration 19 of

the `repeat loop`, this means that, as opposed to the initially proposed approaches, this third alternative would not lead to any configuration respecting the constraints if we performed less than 19 iterations.

Table 4.4: Comparison between the different alternatives

Approach	$J(X)$ at the optimum	% of tested configurations which respect the constraints
NNSGO	0.34	70%
Alternative approach 1 (Random generation)	0.42	20%
Alternative approach 2 (Exploration only)	-	0%
Alternative approach 3 (Exploitation only)	0.34	5%
Bayesian Optimization	0.34	40%

Table 4.4 summarizes the results obtained from the different alternatives. As we can see from this table and from the discussions above, it is clear that the proposed approaches balancing exploitation and exploration have the best overall results for this particular geometrical configuration. Indeed, BO and NNSGO are the approaches which provide the configuration with the lowest average drag $J(X)$ and where there are the most tested configurations which respect the constraints.

4.7 Conclusion

The aim of this chapter is to solve an aerodynamic design problem where the goal is to find the optimal dimensions of a projectile fitted with fins. To solve this problem we propose to use a data-driven model to estimate the relationship between the dimensions of the projectile and the objective function and constraints, since there is no readily-available physical model between the two. We can thus use the procedures presented in Chapter 2, to find the data to select in order to improve the models and get close to the optimum. In this sense, we perform the Optimization using Bayesian Optimization and the Neural Network Smart Grid based Optimization procedures. The optimal configurations found by both approaches have the same value of the objective function. Therefore both approaches can be used to solve such an aerodynamic design problem. Moreover, these two approaches, which consists in a trade-off between exploration and exploitation, perform better than alternative approaches where we perform random search, exploration or exploitation only to select the configurations to evaluate.

Chapter 5

Identification for feedforward control of Wiener systems

In the previous chapters we saw how identified data-driven models for static systems can be used to find the optimum of certain functions. When the system is dynamic, similar problems are formulated in control engineering: determining a control action which will lead to the maximum of a performance criterion. We will here consider a particular type of dynamic systems: the Wiener systems and a particular type of control objective: the tracking of a given reference using feedforward control. We begin by discussing how such problems have been solved in the literature in Section 5.1. Then in Section 5.2, we present the controller design problem and motivate our approach. In Section 5.3, we discuss the identification criterion of the model and the design criterion of the controller. We also present the parametrization of both the controller and the model. In Section 5.4, we present a procedure which can be used to improve the accuracy of the model if necessary. We finish by illustrating the procedures through numerical examples in Section 5.5.

Contents

5.1 Introduction **79**
5.2 Framework **81**
5.3 Structures and identification criterion **83**
5.3.1 Parametrization and identification of $\mathcal{M}$ 83
5.3.2 Parametrization and design of $\mathcal{F}$ 84
5.4 Iterative learning procedure **85**
5.5 Numerical Examples **86**
5.5.1 Controller synthesis without an iterative procedure 87
5.5.2 Application of the iterative procedure 90
5.5.3 An example with the knowledge of the exact parametrization 94
5.6 Conclusion **96**

5.1 Introduction

From mechanical, electronic to aerospace systems, numerous real life applications exhibit non-linear characteristics, posing significant challenges in the design of controllers. In this chapter,

we are particularly interested in a specific type of control: feedforward control.

The primary goal of this chapter is to present an iterative identification procedure for the design of a feedforward controller for nonlinear systems which can be represented by Wiener structures, *i.e.*, a cascade of a linear dynamic block and static nonlinearity.

Feedforward control is generally used to achieve the tracking of a given reference signal $r(t)$. This is generally achieved by determining the input signal that has to be applied to the true system using optimal control (Vinter and Vinter, 2010) or iterative learning control (ILC) (Bristow et al., 2006). In optimal control, the optimal input is determined based on a model of the true system while in ILC this input is determined using multiple experiments on the system (model-free approach). In both cases, the optimal input is determined for a given realization of the reference signal. This means that, for a reference having the same frequency content, but a different time realization, the whole procedure has to be repeated. This drawback in fact only pertains to the case of nonlinear systems. Indeed, in the linear case, we can, *e.g.*, use H_2 control to design a feedforward controller $\mathcal{F}$ (whose expression depends on the frequency content of the reference signal). Once the controller $\mathcal{F}$ has been designed for a given type of reference, it can therefore be used to generate the optimal control input for different realizations of the reference (and for references having similar frequency content). It would therefore be interesting to be able to design such feedforward controllers also for nonlinear systems. This is what is proposed in, *e.g.*, (Aarnoudse et al., 2021), (Van Hulst et al., 2022) and (Poot et al., 2023). In these papers, iterative learning control is used to determine the optimal input for a representative realization of the reference signal and a nonlinear feedforward controller which allows reproducing the optimal input when fed with the reference signal is then determined.

The ILC based approaches in (Aarnoudse et al., 2021), (Van Hulst et al., 2022) and (Poot et al., 2023) are particularly interesting when we do not have a precise idea of the structure of the true system. In this chapter, we consider the situation where this structure is known. In particular, we will suppose that the true system is a Wiener system. In this case, one can use a single experiment on the true system to derive input-output data that can then be used to identify a model of the true system. Based on this identified model and a representative realization of the reference signal, we propose an approach to determine the feedforward controller using an approach close to an identification criterion. The structure of the feedforward controller is chosen as a Hammerstein system, *i.e.*, a static nonlinearity followed by a linear transfer function. This choice is motivated by the fact that feedforward control can be seen as a way to somehow invert the true system (see also (Van Hulst et al., 2022) and (Poot et al., 2023)).

While iterative learning control generally requires multiple experiments on the true system, the identification approach proposed in the previous paragraph has the advantage that it requires in theory only one well-designed experiment on the true system in order to identify the model of the true system. It is clear though that, if this experiment is not well designed, the identified model will not be an accurate representation of the true system and the resulting feedforward controller will not be able to track the reference in a satisfactory manner. In this situation, the input-output data obtained by applying the feedforward controller on the true system can be used to improve the accuracy of the model and to update the feedforward controller. This procedure can be repeated until a suitable controller is synthetized. Note that this approach can be considered as costless since the additional data are obtained in a feedforward control configuration. Note also that this iterative procedure can be considered as an extension to the case of feedforward

controller design for nonlinear (Wiener) systems of the classic iterative identification-control design procedure presented in (Van Den Hof and Schrama, 1995), (Gevers, 2002) and (Forgione et al., 2015) for the linear case.

As mentioned above, in this chapter, the structure of the true system is assumed to be a Wiener system and the feedforward controller will have a Hammerstein structure. The Wiener and Hammerstein structures are classic examples of block-oriented models (Giri and Bai, 2010). These types of models allow representing nonlinear systems by dividing them into interconnected static nonlinear and linear dynamic blocks. Due to their flexibility and easy-understanding, block-oriented models have found applications in the identification of numerous systems (Firouz et al., 2020),(Muhammad et al., 2021), (Naitali et al., 2008). They have also been used as models for to-be-controlled systems in, *e.g.*, (Rollins et al., 2016), (Dolanc and Strmenik, 2008), and (Kalafatis et al., 2005). As indicated above, we will identify a Wiener model of the Wiener true system and we will design the Hammerstein feedforward controller using an approach close to an identification criterion. For this purpose, we can leverage the numerous identification approaches that are available for such model structures (see, *e.g.*, (Schoukens et al., 2015) and (Schoukens et al., 2012)). In this chapter, we will in particular take advantage of the recently introduced "*dynoNet*" nonlinear identification tool proposed in (Forgione and Piga, 2021).

To sum up, our goal in this chapter will be to:

- show how a feedforward controller can be designed from an identified block-oriented Wiener model of the to-be controlled system.

- present a costless iterative model identification and controller design approach in a case where an initially designed controller does not perform well on the true system.

5.2 Framework

In this chapter, we are interested in a feedforward open-loop control scheme for a nonlinear system which can be represented by a Wiener [10] structure (*i.e.*, a cascade of a linear dynamic block and static nonlinearity).

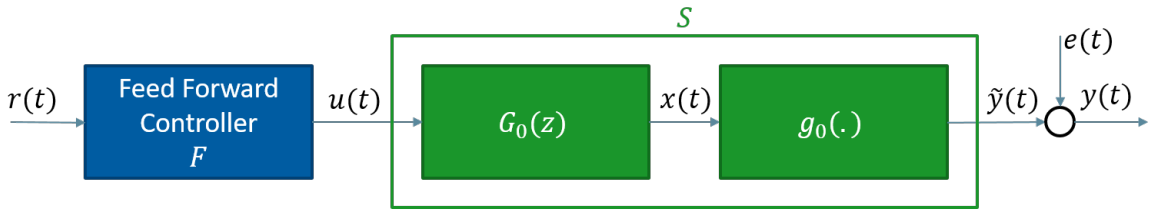


Figure 5.1: Control scheme

[10] This study is presented in the case where the system can be represented by a Wiener structure. However, the same study can be performed with a system which can be represented with other block oriented models with serial structures

This open-loop controller design problem is depicted in Figure 5.1. In this figure, $r(t)$ is the reference that the true system has to track. Like in the linear case, it is important to define the class of reference signals that we wish to track. To illustrate the concepts in this chapter, we will suppose that the frequency content of $r(t)$ can be described by a known transfer function $H(z)$. In other words, we will suppose that $r(t)$ is given by:

$$r(t) = H(z)\tilde{e}(t), \tag{5.1}$$

where t represents the sample number and $\tilde{e}(t)$ is a white Gaussian noise whose variance defines the amplitude range of $r(t)$.

The signal $u(t)$ on Figure 5.1 is the input provided to the system and $y(t)$ is the measured output of the system.

The true system $\mathcal{S}$ is considered to be a Wiener system. These systems are composed of a linear time dynamic block followed by a static nonlinearity. In this chapter, the linear block, noted $G_0(z)$, is considered to be stable. The static nonlinearity is noted $g_0(.)$.

In Wiener systems, the input $u(t)$ is filtered through the linear dynamic block $G_0(z)$, producing an intermediate signal $x(t) = G_0(z)u(t)$. This signal is then passed through the static nonlinearity $g_0(.)$, yielding the output signal $\tilde{y}(t) = g_0(x(t))$. In the remainder of the chapter, we will use the following expression for $\tilde{y}(t)$:

$$\tilde{y}(t) = g_0(G_0(z)u(t)). \tag{5.2}$$

The output of the Wiener system $\tilde{y}(t)$ is corrupted with a zero mean white Gaussian noise e: $\mathcal{N}(0, \sigma^2)$. For a given input $u(t)$, the measured output $y(t)$ can thus be expressed as follows:

$$y(t) = g_0(G_0(z)u(t)) + e(t). \tag{5.3}$$

The goal in this chapter is to design the feedforward controller $\mathcal{F}$ for the system $\mathcal{S}$. For a given reference $r(t)$, the controller should provide a control signal $u(t)$ to pass to the system, allowing it to track the reference. The controller $\mathcal{F}$ should be designed in order to minimize the following criterion:

$$\mathbb{E}(r(t) - \tilde{y}(t))^2 + \lambda \mathbb{E}u^2(t), \tag{5.4}$$

where $\mathbb{E}$ represents the expectation operator and $\lambda > 0$ is a user chosen constant which defines the penalty on the control efforts.

As indicated in the introduction, the controller $\mathcal{F}$ must somehow achieve an inversion of the true system. Since this true system has a Wiener structure, it therefore makes sense to choose a Hammerstein structure for the controller $\mathcal{F}$, *i.e.*, a static nonlinearity followed by a linear dynamic block. The static nonlinearity of the feedforward controller will be denoted by $f(.)$ and the linear dynamic block by $F(z)$. In Hammerstein systems the input $r(t)$ passes through the static nonlinearity $f(.)$ providing an intermediate signal $\tilde{z}(t) = f(r(t))$. This signal is then filtered through the linear dynamic block $F(z)$ producing the output of the Hammerstein system $u(t) = F(z)\tilde{z}(t)$. In the remainder of the chapter, we will therefore use the following expression for the control input $u(t)$ generated by the reference $r(t)$:

$$u(t) = F(z)f(r(t)). \tag{5.5}$$

In this chapter, it is considered that an initial dataset $\mathcal{D}_1 = \{u_1(t), y_1(t)\}_{t=1}^{N}$, obtained from an open-loop experiment on $\mathcal{S}$ is available. Since we know the structure of the true system, we can use this dataset to identify a model $\mathcal{M}$ of $\mathcal{S}$ and then design the controller based on that model. The true system $\mathcal{S}$ belongs to the class of Wiener systems, therefore, the model $\mathcal{M}$ will also have the same structure. The model $\mathcal{M}$ will be identified using the data $\mathcal{D}_1$ in a model structure parametrized by a parameter vector θ. This vector θ contains the parameters of both the linear block $G_\theta(z)$ and the static nonlinearity $g_\theta(.)$

In the next section, we will show how the model of the true system can be identified and how the feedforward controller can be determined based on this identified model and a representative realization of the reference signal (5.1). The latter will be done using a criterion which is very close to an identification criterion.

5.3 Structures and identification criterion

In this section, we present the identification criterion for the model $\mathcal{M}$ and the design criterion for $\mathcal{F}$. As will be shown in the sequel, the design criterion for $\mathcal{F}$ will be close to an identification criterion. Therefore, we will talk about "identification of the controller" in the sequel.

In this chapter we choose a parametric identification approach for the Wiener and Hammerstein structures. As mentioned above, we will in particular take advantage of the recently introduced "*dynoNet*" nonlinear identification tool proposed in (Forgione and Piga, 2021). The identification procedures for $\mathcal{M}$ and $\mathcal{F}$, as well as their parametrization are discussed in the following sections.

5.3.1 Parametrization and identification of $\mathcal{M}$

As mentioned above the model will have the same structure as the true system, *i.e.*, a Wiener structure. Its blocks are parametrized as follows:

$$G_\theta(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_{n_b} z^{-n_b}}{1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_{n_a} z^{-n_a}} \tag{5.6a}$$

$$g_\theta(.) = \tilde{g}_\theta(., \theta_g), \tag{5.6b}$$

where $\tilde{g}_\theta(., \theta_g)$, is a class of non linear static function described by its parameter vector $\theta_g \in \mathbb{R}^{n_g}$. We note by $\theta_a \in \mathbb{R}^{n_a}$ the vector containing the n_a coefficients of the denominator of the linear block of $\mathcal{M}$ and $\theta_b \in \mathbb{R}^{n_b}$ the vector containing the $n_b + 1$ coefficients of numerator: $\theta_a = [a_1, a_2, ..., a_{n_a}]^T$ and $\theta_b = [b_0, b_1, b_2, ..., b_{n_b}]^T$. The parameter vector $\theta \in \mathbb{R}^{n_a+n_b+1+n_g}$ thus contains the parameters in θ_a,θ_b and θ_g *i.e.*, $\theta = [\theta_a^T, \theta_b^T, \theta_g^T]^T$.

The goal of the identification of $\mathcal{M}$ is to find the parameters θ of $\mathcal{M}$ which allow representing accurately the behaviour of the true system. Given the initial dataset $\mathcal{D}_1 = \{u_1(t), y_1(t)\}_{t=1}^{N}$, this can be achieved by minimizing the following cost function W:

$$W(\theta) = \frac{1}{N}\sum_{t=1}^{N}\left(g_\theta(G_\theta(z)u_1(t)) - y_1(t)\right)^2, \tag{5.7}$$

where N is the number of collected samples. The parameter vector $\hat{\theta}$ which allows representing best the model given a dataset $\mathcal{D}_1 = \{u_1(t), y_1(t)\}_{t=1}^{N}$, is thus provided by:

$$\hat{\theta} = \arg\min_{\theta} W(\theta). \tag{5.8}$$

In the sequel, the blocks of the identified model $\mathcal{M}$ (*i.e.*, the one parametrized with $\hat{\theta}$) will be denoted by $\hat{G}_\theta(z)$ and $\hat{g}_\theta(.)$. For a given input $u(t)$, the output $\hat{y}$ of $\mathcal{M}$ is therefore given by:

$$\hat{y}(t) = \hat{g}_\theta(\hat{G}_\theta(z)u(t)). \tag{5.9}$$

5.3.2 Parametrization and design of $\mathcal{F}$

The feedforward controller $\mathcal{F}$ will have a Hammerstein structure described by a parameter vector ϕ. Its static non linearity, denoted by $f_\phi(.)$, and linear dynamic layer, denoted by $F_\phi(z)$, are parametrized as follows:

$$F_\phi(z) = \frac{d_0 + d_1 z^{-1} + d_2 z^{-2} + ... + d_{n_d} z^{-n_d}}{1 + c_1 z^{-1} + c_2 z^{-2} + ... + c_{n_c} z^{-n_c}} \tag{5.10a}$$

$$f_\phi(.) = \tilde{f}_\phi(., \phi_f), \tag{5.10b}$$

where $\tilde{f}_\phi$, is a class of non linear static function described by its parameter vector $\phi_f \in \mathbb{R}^{n_f}$. We denote by $\phi_c \in \mathbb{R}^{n_c}$ the vector containing the n_c coefficients of the denominator of the linear block and $\phi_d \in \mathbb{R}^{n_d}$ the vector containing the n_d+1 coefficients of the numerator: $\phi_c = [c_1, c_2, ..., c_{n_c}]^T$ and $\phi_d = [d_0, d_1, d_2, ..., d_{n_d}]^T$. We denote $\phi \in \mathbb{R}^{n_c+n_d+1+n_f}$ the vector containing all the parameters of the controller: $\phi = [\phi_c^T, \phi_d^T, \phi_f^T]^T$.

We will suppose that we have a representative[11] realization of duration N_r of the reference signal (5.1) and we will approximate the control criterion (5.4) using this realization and the identified model $\hat{G}_\theta(z), \hat{g}_\theta(.)$ of the true system:

$$V(\phi, \hat{\theta}) = \frac{1}{N_r}\sum_{t=1}^{N_r}\left(\hat{g}_\theta(\hat{G}_\theta(z)u_\phi(t)) - r(t)\right)^2 + \lambda u_\phi^2(t), \tag{5.11}$$

where $u_\phi(t) = F_\phi(z)f_\phi(r(t))$.

Given the identified model $\hat{G}_\theta(z), \hat{g}_\theta(.)$ the optimal parameter vector $\hat{\phi}$ is thus:

$$\hat{\phi} = \arg\min_{\phi} V(\phi, \hat{\theta}). \tag{5.12}$$

Note the similarity between the criterion (5.11)-(5.12) and the identification criterion (5.7)-(5.8). Similar optimization algorithms can therefore be used to determine both the model of the true

[11] *i.e.*, a sufficiently long realization of (5.1)

system and the feedforward controller. This is also why we say in this chapter that the feedforwad controller is "identified".

The optimal controller which minimizes criterion (5.11) with parameter vector $\hat{\phi}$ will be denoted by $\hat{F}_{\phi}(z)\hat{f}_{\phi}(.)$.

Remark 1 In this study, we will suppose that the only information we have about the true system $\mathcal{S}$ is that it is a Wiener system. In other words, we suppose that the order of $G_0(z)$ is unknown and that the exact structure of the nonlinearity $g_0(.)$ is also unknown. We will however suppose that the parametrization $\mathcal{M}$ is chosen sufficiently large to describe the true system. In particular, the model $g_\theta(.)$ for the static nonlinearity will be chosen as a neural network of sufficient complexity (see Section 1.2 of Chapter 1). We can verify whether the model structure $\mathcal{M}$ is sufficiently rich to describe the true system by, *e.g.*, evaluating the fit of the identified model on validation data. In the numerical illustration, we will also show the advantage that can be gained when the structure of $\mathcal{S}$ is perfectly known.

Remark 2 Nonlinear optimization tools are used to solve the optimization problems (5.7) and (5.11). To perform the optimization, we need to provide initial guesses for the parameter vectors θ and ϕ. In the numerical illustrations in the sequel, these initial guesses will be chosen randomly. Some more adequate choices of initialization during the identification of block-oriented models have been proposed in, *e.g.*, (Schoukens, 2015).

5.4 Iterative learning procedure

Using the procedure presented in the previous section, a feedforward controller is available and this feedforward controller (say $\mathcal{F}_1$) can be used to design the control input of the true system using the reference signal $r(t)$. If the performance of the controller is not satisfactory (*i.e.*, the output $y(t)$ is not sufficiently close to $r(t)$), the data generated during the application of the feedforward controller $\mathcal{F}_1$ can be used to improve the model. This data will be denoted by $\mathcal{D}_2 = \{u_2(t), y_2(t)\}_{t=1}^{N}$ in the sequel[12]. A new and better model $\mathcal{M}_2$ can be identified using this new dataset $\mathcal{D}_2$ as well as the previous dataset $\mathcal{D}_1$. The identification criterion for the model $\mathcal{M}_2$ will be:

$$\begin{aligned} W_2(\theta) = &\frac{1}{N}\sum_{t=1}^{N}\left(g_\theta(G_\theta(z)u_1(t)) - y_1(t)\right)^2 \\ &+ \frac{1}{N}\sum_{t=1}^{N}\left(g_\theta(G_\theta(z)u_2(t)) - y_2(t)\right)^2. \end{aligned} \tag{5.13}$$

The new parameter vector, $\hat{\theta}_2$, which allows representing best the true system $\mathcal{S}$, given the dataset $\mathcal{D}_1$ and $\mathcal{D}_2$ is thus given by:

$$\hat{\theta}_2 = \arg\min_{\theta} W_2(\theta). \tag{5.14}$$

When a new model $\mathcal{M}_2$ has been identified, then a new controller $\mathcal{F}_2$ can also be designed based on $\mathcal{M}_2$ and using the identification criterion (5.11). The sequence consisting in first identifying

[12] For the simplicity of notations, we will suppose that $\mathcal{D}_1$ and $\mathcal{D}_2$ contain the same amount of data

a new model and then redesigning the controller based on that model can be repeated iteratively until a controller which performs well on the true system is found. Note that this approach can be considered as costless since the additional data are obtained in a feedforward control configuration.

The procedure is described in Algorithm 3.

Algorithm 3 Iterative model identification and controller design procedure

Initialize: Available dataset $\mathcal{D}_1 = \{u_1(t), y_1(t)\}_{t=1}^{N}$ and a representative realization of length N_r of the to-be-tracked reference $r(t)$.
for $i = 1, ...i_{max}$
1. Identify a model $\mathcal{M}_i$ using the data contained in the set $\mathcal{D} = \{\mathcal{D}_1 = (u_1(t), y_1(t)), \mathcal{D}_2 = (u_2(t), y_2(t))...., \mathcal{D}_i = (u_i(t), y_i(t)))\}$ and the criterion:

$$W_i(\theta) = \frac{1}{N} \sum_{i=1}^{i_{max}} \sum_{t=1}^{N} \left(g_\theta(G_\theta(z)u_i(t)) - y_i(t)\right)^2.$$

2. Design the feedforward controller $\mathcal{F}_i$ using the identified model $\mathcal{M}_i$ and the design criterion:

$$V(\phi, \hat{\theta}_i) = \frac{1}{N_r} \sum_{t=1}^{N_r} \left(\hat{g}_{\theta_i}(\hat{G}_{\theta_i}(z)u_\phi(t)) - r(t)\right)^2 + \lambda u_\phi^2(t).$$

3. Perform an experiment on the true system $\mathcal{S}$ using the designed controller $\mathcal{F}_i$, collect the data $\mathcal{D}_{i+1} = \{u_{i+1}(t), y_{i+1}(t)\}_{t=1}^{N}$ and add them to the set $\mathcal{D}$.
End For

5.5 Numerical Examples

In this section the procedures described above will be illustrated. To evaluate the goodness of the identification of the model and of the tracking of the reference by the controller, we use the "*fit*" metric. It is calculated as follows:

$$fit_M = 100\left(1 - \frac{\sqrt{\sum_{t=1}^{\tilde{N}}(\hat{y}(t) - y(t))^2}}{\sqrt{\sum_{t=1}^{\tilde{N}}(\hat{y}(t) - \overline{y}(t))^2}}\right)\% \tag{5.15a}$$

$$fit_F = 100\left(1 - \frac{\sqrt{\sum_{t=1}^{\tilde{N}_r}(y(t) - r(t))^2}}{\sqrt{\sum_{t=1}^{\tilde{N}_r}(y(t) - \overline{r}(t))^2}}\right)\%, \tag{5.15b}$$

where $\overline{a}$ represents the mean value of the vector containing the N observations of a. fit_M and fit_F are the fit on the identification of the model (computed using validation data, *i.e.*, data that have not been used for the identification) and the fit on the tracking of the reference by the true system, respectively.

Note that, since this is a simulation example, we could have used $\tilde{y}(t)$ instead of $y(t)$ in (5.15). The choice to use $y(t)$ instead of $\tilde{y}(t)$ is nevertheless made since $\tilde{y}(t)$ is unavailable in practice. However, since the variance of the noise will be relatively small (5.10^{-4} in the sequel), $\tilde{y}(t)$ will be very close to $y(t)$.

To illustrate the effectiveness of the procedure, different cases will be considered:

- In the first one, we will consider a case where there is no need for an iterative procedure to design the controller.
- In the second one, we will perform the iterative procedure (see Section 5.4).
- In the third example, we illustrate the advantage of knowing the exact parametrization of the true system.

For the cases considered, we will use the same true system $\mathcal{S}$. As mentioned before, the true system is a Wiener system, *i.e.*, it is composed of a linear dynamic block $G_0(z)$ followed by a static nonlinearity $f_0(x)$.

The linear dynamic bloc $G_0(z)$ of the true system S is:

$$G_0(z) = \frac{1}{1 - 0.1412z^{-1} - 0.1z^{-2}}. \tag{5.16}$$

The static nonlinearity of the true system S is:

$$g_0(x) = 5\tan^{-1}(2x). \tag{5.17}$$

5.5.1 Controller synthesis without an iterative procedure

We will first show that, if the open-loop experiment leading to $\mathcal{D}_1$ is well designed (*i.e.*, u_1 is sufficiently broadband and sufficiently powerful), the identified model $\mathcal{M}_1$ is sufficient to design a satisfactory controller $\mathcal{F}_1$ (no iterative procedure is required).

We assume here, that the following experiment has been conducted on the true system $\mathcal{S}$ in order to collect a dataset $\mathcal{D}_1$:

- the system is excited with a multisine input $u_1(t)$ with a maximum frequency of 2000 Hz, containing enough frequency components to be sufficiently rich for identification;
- 20000 samples from the output of the system $y_1(t)$ are collected.
- the output is corrupted with a zero mean white Gaussian noise of standard deviation $\sigma_e = 5.10^{-4}$.

This dataset is divided into 2 sets of 10000 samples each. The first set is used to identify the model, the second one for validation.

In this example, it is considered that the exact structure of the true system is not known. As mentioned before, we will however choose a model structure $\mathcal{M}$ allowing to describe the true system in a sufficient manner (see **Remark 1**). In this sense, the structure of the model is chosen as the one described in (5.6), where $n_a = 3$,$n_b = 4$ and $\tilde{g}_\theta$ is a Neural Network with 1 hidden layer (see (1.2)) with 16 neurons and a *tanh* activation function (See Section 1.2 of Chapter 1). The parameter vector has dimension 42, *i.e.* 7 parameters for the linear dynamic block and 35 (weights and biases) for the NN model. The optimal parameter vector $\hat{\theta}$ can be identified using the criteria (5.7)-(5.8). The computational time for this identification is of approximately 2 minutes.

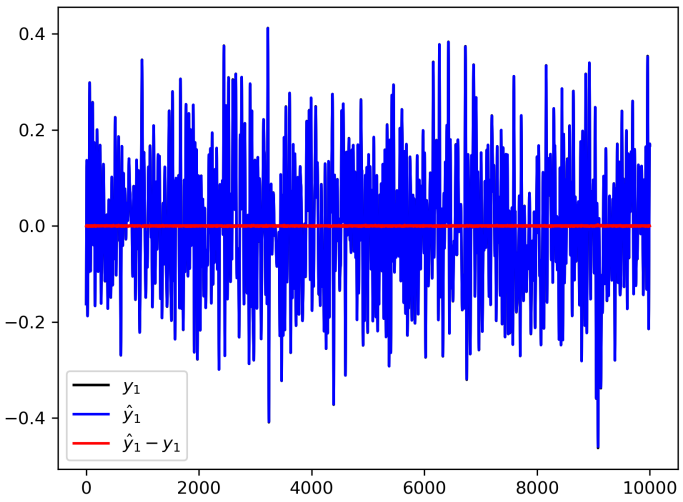


Figure 5.2: Model output $\hat{y}_1$ (blue), true system output y_1 (black) and difference between $\hat{y}_1$ and y_1 (red) for a given input $u_1(t)$

Figure 5.2 presents the difference between the predicted output from the identified model $\mathcal{M}$ compared to the output of the true system $\mathcal{S}$, on the validation set. As can be seen from this figure, the blue signal ($\hat{y}_1$) and the black signal (y_1) are overlapping and the red signal ($\hat{y}_1$-y_1) is very small. The $fit_M = 99\%$. This shows that we are able to identify a parameter vector $\hat{\theta}$ that allows representing well the model with the given data.

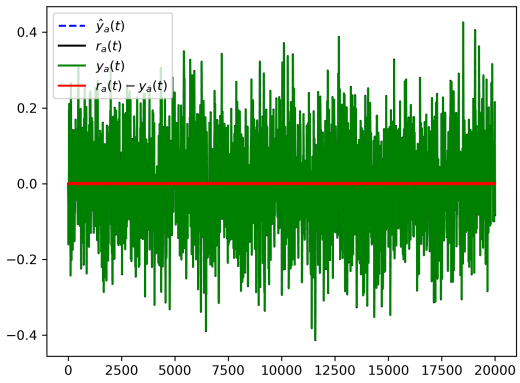
Now that a model has been identified from the data, a controller $\mathcal{F}_1$ can be designed from the model. In this case, we want to design our controller to allow the model to follow a reference signal $r_a(t)$ which corresponds to a filtered white Gaussian noise (5.1) where $H(z)$ is a lowpass filter, with a cutoff frequency at 2000Hz (the sampling frequency is here equal to 51200 Hz). It is given by:

$$H(z) = \frac{0.0014 + 0.004z^{-1} + 0.004z^{-2} + 0.001z^{-3}}{1 - 2.51z^{-1} + 2.13z^{-2} - 0.61z^{-3}}. \tag{5.18}$$

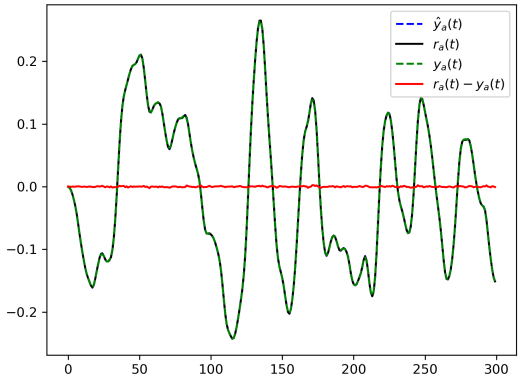
The variance of $\tilde{e}(t)$ has been chosen in such a way that the maximal amplitude of $r_a(t)$ is close to the one of $y_1(t)$ in $\mathcal{D}_1$. The structure of the controller is the one described in (5.10), where $n_d = 4$,$n_c = 4$ and $\tilde{f}_\phi$ is a Neural Network with 1 hidden layer, 16 neurons and a *tanh* activation

function. In total there are 44 parameters stored in the parameter vector ϕ to identify. These parameters are identified using the criteria (5.11)-(5.12) with $\lambda = 0$. The computational time for the controller design is of approximately 3 minutes.

Once the controller has been designed from the model, it can be tested on the true system as well.



(a) Full length signals



(b) Zoomed-in portion of the signals

Figure 5.3: Model output $\hat{y}_a$ (blue), true system output y_a (green), reference signal r_a (black) and difference between r_a and y_a (red) when the input is the control signal provided by the designed controller $\mathcal{F}_1$

Figure 5.3 shows the outputs $y_a(t)$ and $\hat{y}_a(t)$ obtained when we test the identified controller on the true system and the model, respectively.

$$\hat{y}_a(t) = \hat{g}_\theta(\hat{G}_\theta(z)u_\phi(t)) \tag{5.19a}$$
$$y_a(t) = g_0(G_0(z)u_\phi(t)) + e(t), \tag{5.19b}$$

where $u_\phi(t) = \hat{F}_\phi(z)\hat{f}_\phi(r_a(t))$. As can be seen in Figure 5.3, $r_a(t)$, $y_a(t)$ and $\hat{y}_a(t)$ are all overlapping. This means that the designed controller not only performs well on the model but also on the true system. This is even more highlighted on Figure 5.3 where the difference between $r_a(t)$ and $y_a(t)$ (in red on Figure 5.3) is very close to 0. We have the $fit_F = 99\%$.

By choosing a filtered white Gaussian noise as reference signal, the designed controller $\mathcal{F}_1$ will not only allow tracking the specific signal $r_a(t)$ but also other signals with the same frequency content and amplitude. To highlight this, we test the controller for 2 signals different from $r_a(t)$. The first one, $r_{a1}(t)$, is a sinusoid of amplitude 0.4 and of frequency 2000 Hz. The second one, $r_{a2}(t)$, is such as r_a given by (5.1) with $H(z)$ in (5.18), but generated with a different realization of the white noise $\tilde{e}(t)$. With the first signal $r_{a1}(t)$ we have $fit_F = 99\%$. With the second signal $r_{a2}(t)$ we have $fit_F = 99\%$. This shows that the designed controller can be used with other reference signals.

5.5.2 Application of the iterative procedure

In the previous section, the reference r_a that must be tracked and that is used to identify the feedforward controller $\mathcal{F}_1$ will generate, when applied to $\mathcal{F}_1$, signals $u_a(t)$ and $y_a(t)$ that are close (in amplitude) to the ones collected in $\mathcal{D}_1$.

In this section, we will suppose that the to-be-tracked reference (5.1) is generated with a white noise $\tilde{e}$ with higher variance. This reference r_b (having higher amplitude) is compared to y_1 in Figure 5.4. This figure shows that the amplitude of r_b is approximatively ten times higher than y_1.

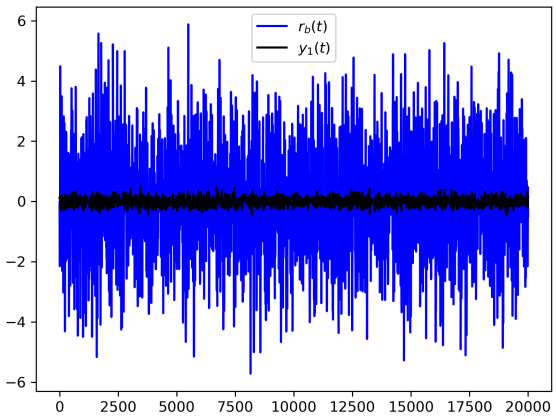
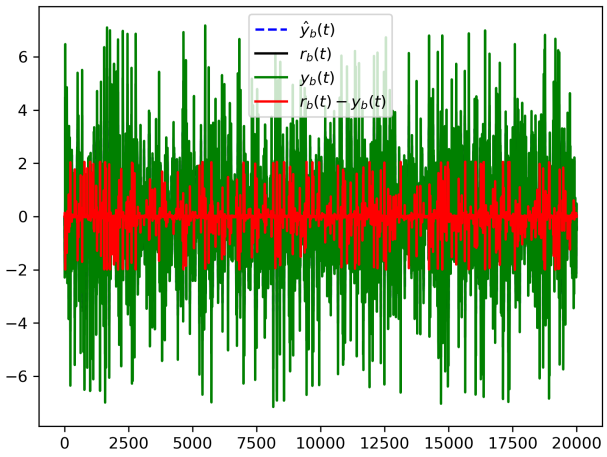
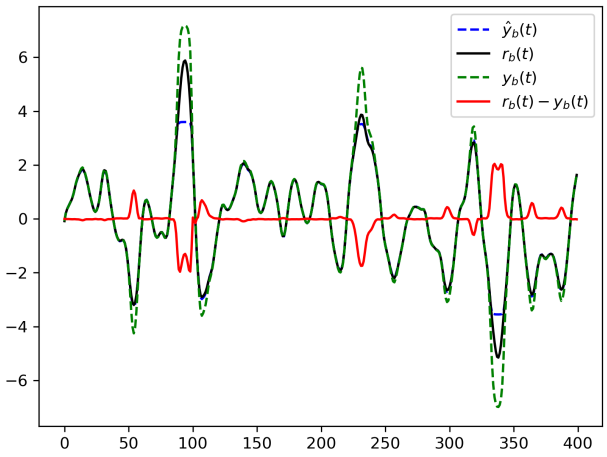


Figure 5.4: New reference signal to follow r_b (in blue) compared to $y_1(t)$ used to identify the model $\mathcal{M}_1$ (in black)



(a) Full length signals

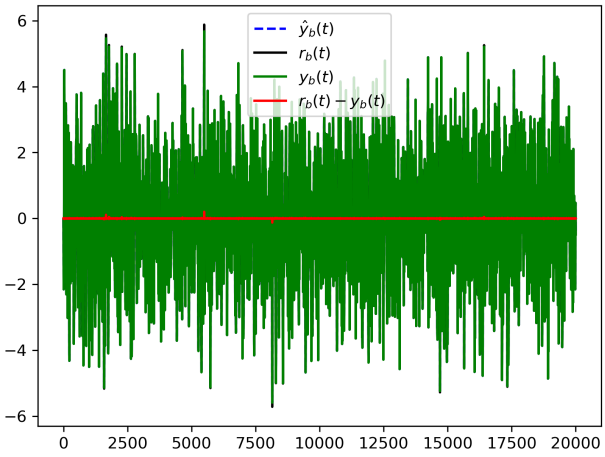


(b) Zoomed-in portion of the signals

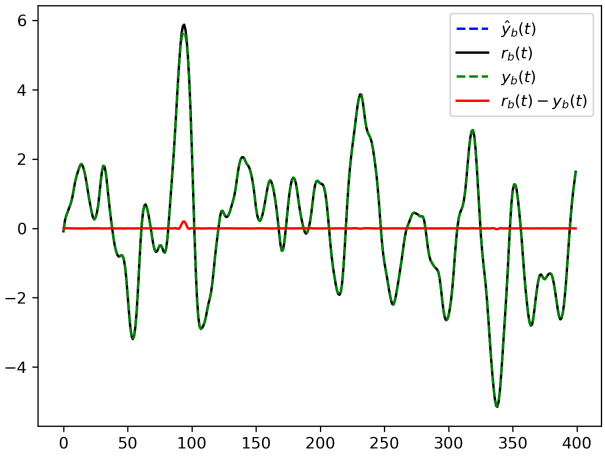
Figure 5.5: Model output $\hat{y}_b$ (blue), true system output y_b (green), reference signal r_b (black) and difference between r_b and y_b (red) when the input is the control signal $u_b(t)$ provided by the controller designed using $\mathcal{M}_1$ and r_b

When (5.11)-(5.12) are used to design the feedforward controller using the model identified with $\mathcal{D}_1$ and with this reference r_b, the tracking performance on the true system is much poorer as shown on Figure 5.5. In this case, we have $fit_F = 78\%$.

Even though the tracking performance observed in Figure 5.5 is not perfect, the application of the controller allows obtaining a certain tracking of r_b which, as indicated in Section 5.4, allows collecting a second dataset $\mathcal{D}_2$. This dataset is used to identify a new model $\mathcal{M}_2$, using the criteria (5.13)-(5.14) and the same structure as $\mathcal{M}_1$. $\mathcal{M}_2$ and r_b are then used to identify a new and better feedforward controller $\mathcal{F}_2$, using the criterion (5.11)-(5.12). The structure of the controller $\mathcal{F}_2$ is the same as the one of $\mathcal{F}_1$.



(a) Full length signals



(b) Zoomed-in portion of the signals

Figure 5.6: Model output $\hat{y}_b$ (blue), true system output y_b (green), reference signal r_b (black) and difference between r_b and y_b (red) when the input is the control signal $u_b(t)$ provided by the new designed controller $\mathcal{F}_2$

By comparing Figures 5.5 and 5.6, it is clear that the controller $\mathcal{F}_2$ identified with $\mathcal{M}_2$ and r_b achieves a much better tracking of r_b than the controller identified with $\mathcal{M}_1$ and r_b. Indeed, we obtain $fit_F = 99\%$.

The improved performance can be related to the fact that the model $\mathcal{M}_2$ identified with both $\mathcal{D}_1$ and $\mathcal{D}_2$ models the nonlinearity g_0 over a larger range than the model identified with only $\mathcal{D}_1$ (see Figure 5.7).

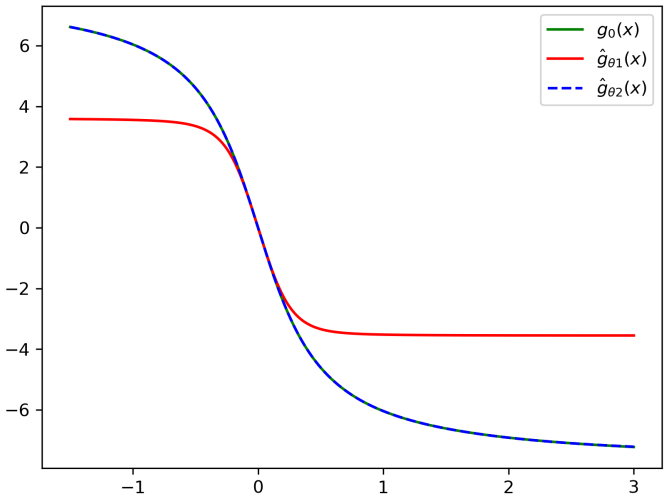


Figure 5.7: Static nonlinearity of the model after the first identification $\hat{g}_{\theta 1}$ (red), after the second identification $\hat{g}_{\theta 2}$ (blue) and true system static nonlinearity g_0 (green).

Note that this controller $\mathcal{F}_2$ also allows tracking efficiently other references with similar frequency content and amplitude as r_b, as well as the initial (smaller in amplitude) reference $r_a(t)$. To highlight this, we choose as reference signal $r_c(t)$: a sinusoid of amplitude 5 and frequency 2000 Hz and feed this sinusoid to $\mathcal{F}_2$.

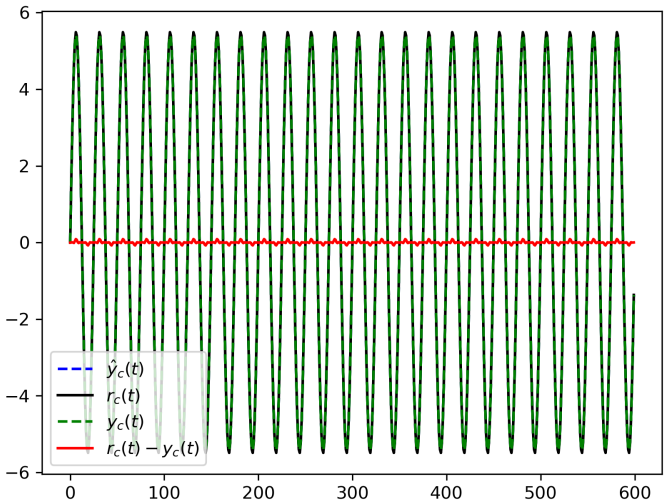


Figure 5.8: Model output $\hat{y}_c$ (blue), true system output y_c (green), reference signal r_c (black) and difference between r_c and y_c (red) when the input is the control signal $u_c(t)$ provided by the controller $\mathcal{F}_2$

As can be observed on Figure 5.8, the identified controller $\mathcal{F}_2$ also allows tracking signals with the same frequency content as $r_b(t)$. With the signal $r_c(t)$, we have a fit_F of 99%. This controller also allows tracking efficiently the initial reference signal $r_a(t)$. Indeed, when we use the controller $\mathcal{F}_2$ to track $r_a(t)$, we have a fit fit_F of 99%.

5.5.3 An example with the knowledge of the exact parametrization

We will see in this subsection that the exact knowledge of the structure of $\mathcal{S}$ will allow improving the results observed in the previous subsection. For this purpose, we consider the same data set $\mathcal{D}_1$ and the same reference r_b as in the previous subsection. However, the model structure $\mathcal{M}$ is now chosen as an exact representation of (5.16)-(5.17):

- The linear dynamic bloc $G_\theta(z)$ of the model $\mathcal{M}$ is parametrized as:

$$G_\theta(z) = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}}. \tag{5.20}$$

- The static nonlinearity $g_\theta(x)$ of the model is parametrized as:

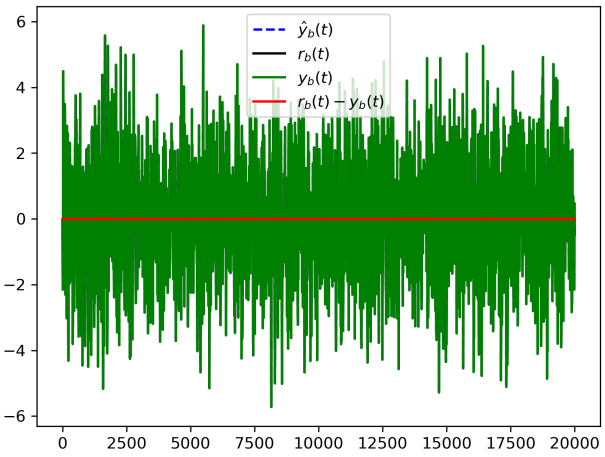
$$g_\theta(x) = \theta_{g,1} tan^{-1}(\theta_{g,2} x). \tag{5.21}$$

We thus have that $\theta = [a_1, a_2, \theta_{g,1}, \theta_{g,2}]^T$. When we use this model structure and the data set $\mathcal{D}_1$, the identification criterion (5.7) yields $\hat{\theta} = [-0.1396, -0.1011, 5.0005, 2.0020]^T$. A close similarity between this identified parameter vector $\hat{\theta}$ and the true parameter vector $\theta_0 = [-0.1414, -0.1, 5, 2]^T$ (see (5.16-5.17)) can be noted. The parameter vector $\hat{\theta}$ allows obtaining a fit_M of 99% on the validation data.

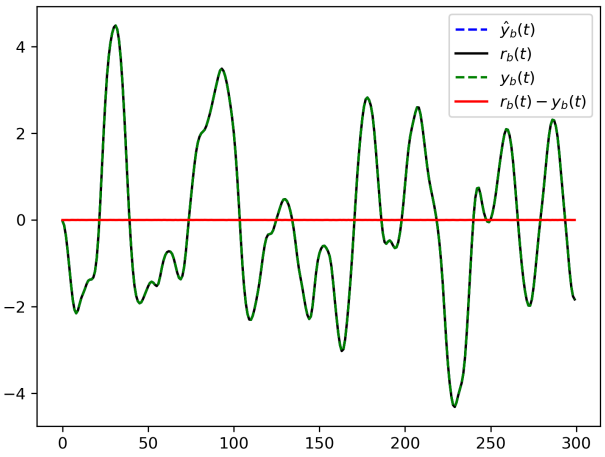
We will now proceed to the determination of the feedforward controller using this identified model and the reference r_b. Note that, since $g_\theta(.)$ is invertible, one can fix the nonlinear part of the feedforward controller to:

$$\hat{f}_\phi(x) = \frac{tan(\frac{x}{\hat{\theta}_{g,1}})}{\hat{\theta}_{g,2}} = \frac{tan(\frac{x}{5.0005})}{2.0020}, \tag{5.22}$$

and only determine the linear part $F_\phi(z)$ in the design criterion (5.11)-(5.12). For this linear part, we choose a transfer function $F_\phi(z)$ with $n_c = 4$ and $n_d = 4$. This leads to a controller $\mathcal{F}_{1,bis}$.



(a) Full length signals



(b) Zoomed-in portion of the signals

Figure 5.9: Model output $\hat{y}_b$ (blue), true system output y_b (green), reference signal r_b (black) and difference between r_b and y_b (red) when the input is the control signal $u_b(t)$ provided by the controller $\mathcal{F}_{1,bis}$

The performance of this controller on r_b is given in Figure 5.9 where fit_F is 99%. This result shows the advantage of having the exact structure of the true system compared to the previous example where a fit_F of 79% was obtained when the controller was designed using the model identified with the dataset $\mathcal{D}_1$ and the reference signal r_b.

Excellent control results are also obtained with other time realizations of r_b and with sinus signals. Indeed, when the controller is tested using these signals, a $fit_F = 99\%$ is obtained for each one of them.

5.6 Conclusion

The objective of this chapter was to see how we can extend the ideas seen in the previous chapters but for dynamic systems. We therefore chose to tackle a problem of designing a feedforward controller for a nonlinear system for which we know the structure. We showed how a data-driven model of such systems could be used to design the controller which provides the control signal that allows tracking a given reference signal. An iterative procedure which consists of re-using the data acquired when testing controllers to re-identify a new model and then design a new controller based on it, in the case where the initially designed model does not allow designing a good controller, is also presented. Results on numerical applications show that the presented method is able to design feedforward controllers which allow following different types of reference signals.

Chapter 6

General Conclusions and Perspectives

6.1 Conclusions

The aim of this thesis work, was to demonstrate how to improve a model identified to optimize the performance of a system while accounting for the cost related to data acquisition.

Two research topics were targeted to answer to this problematic:

- **Research Topic 1**: How to improve a model being used to find the optimum of an expensive to evaluate static function. This research topic is applied to the aerodynamic optimization of projectiles.

- **Research Topic 2**: How to design a feedforward controller for a nonlinear Wiener system based on an identified model of the latter, and how to improve this model if the initially designed controller does not perform well.

The results of the first research topic are presented from Chapter 1 to Chapter 4. First, models which can be used to identify static systems are presented in Chapter 1. Two of the most commonly used models, which are utilized in the thesis, are then discussed. Non-parametric and probabilistic models, Gaussian Processes, and parametric and non-probabilistic models, Neural Networks. The main advantage of Gaussian Processes lies in their ability to provide an uncertainty measure for their estimates of a function, which is not inherent for classic Neural Networks models. However, Neural Networks are better suited for handling large datasets, whereas Gaussian Processes tend to scale poorly as the size of the dataset increases.

Chapter 2, then discusses how the models presented in the first chapter are used to find the optimum of a function when it is expensive to evaluate. More specifically, approaches used in the literature to strategically choose the data to use in order to improve the models for optimization purposes were presented. The main approach for selecting such data involves balancing the exploration and exploitation objectives. The most commonly used procedure which accomplishes this trade-off is Bayesian Optimization. In this method, Gaussian Process models are used to identify a model of the objective function and of the constraints. The estimates provided by the Gaussian Process models and their associated uncertainties are then used to build an acquisition

function which balances the exploration and exploitation objectives. The points to evaluate, in order to improve the model and search for the optimum, are then iteratively chosen by maximising the acquisition function. It was noted that one possible limitations of Bayesian Optimization is that Gaussian Processes scale poorly as the number of samples increases. Therefore if Bayesian Optimization needs to be performed with an initially large dataset it may loose efficiency.

First contribution of the thesis : The first contribution of this thesis is thus to explore the use of other machine learning models, specifically those that handle large datasets well, in procedures similar to Bayesian Optimization. In this sense, Neural Networks are proposed for such task. Unlike Gaussian Processes, traditional Neural Networks do not provide a measure of uncertainty which is typically used for the exploration objective. We thus propose to use the minimum distance (in an Euclidean norm) between a point in the search domain and the closest point to it in the dataset, used to identify the model, as a measure for the exploration objective. It is illustrated that this measure behaves similarly to the uncertainties provided by the Gaussian Process models. Indeed, both increase when evaluated at points which are far away from the data in the data set used to identify the model, which are the points likely to be predicted incorrectly. An acquisition function when the model of the objective function is a Neural Network is then built using this measure and the estimates provided by the model. The procedure, called Neural Networks Smart Grid based Optimization (NNSGO), was tested on multiple examples and its performances were compared to those of Bayesian Optimization. The main conclusions are the following:

- the NNSGO procedure offers competitive performances to Bayesian Optimization, *i.e.*, it allows finding similar optima as Bayesian Optimization;

- it was also observed that when the initial dataset is large, NNSGO is faster, in terms of computational time, to perform than Bayesian Optimization. Indeed, the identification of a Neural Network model is faster than the identification of a Gaussian Process model in this case, which allows fastening the overall procedure.

Chapter 3 and 4 are dedicated to the application of the concepts presented in the first two chapters to the fields of aerodynamics and ballistics. The main goal is to explore how these procedures can be applied to design the optimal dimensions of a projectile fitted with fins for a given flight scenario. To achieve this, one must have access to coefficients associated with the forces and moments applied to the projectile during its flight, known as aerodynamic coefficients. In Chapter 3, these coefficients are presented and the different methods for estimating them are discussed. The most reliable sources of aerodynamic coefficients are experimental tests and Computational Fluid Dynamics simulations. These two approaches are however extremely time-consuming and expensive to carry out. Having a model which estimates them accurately and fast is thus very crucial. In Chapter 3, we have presented how such models could be identified to predict the coefficients based on the configurations of projectiles and the Mach number. Two approaches have been presented to accomplish this depending on the source of data. The first is a classic approach were we use the data from the highest fidelity source to directly identify a model. The second one consists in learning the discrepancy between the higher fidelity source and the lower fidelity source, using data from both source in a multi-fidelity learning approach. It is shown that both approaches provide accurate estimates and that combining them allows

improving even more the accuracy. These models can, therefore, be used to rapidly and relatively accurately estimate the aerodynamic coefficients of the family of projectiles studied instead of relying on expensive computer simulations or experiments.

In Chapter 4, the aerodynamic design problem is addressed. The goal is to find the configuration with the least drag that allows reaching a target in a minimum time, while also ensuring some stability criteria. The problem is formulated as a nonlinear constrained optimization problem which requires evaluating aerodynamic coefficients to solve it. As mentioned above evaluating these coefficients for a given configuration is an expensive task. However, an initial data set containing 344 projectile configurations associated to different aerodynamic coefficients for different Mach numbers is available. Using this data set, and the procedures described in Chapter 2, *i.e.* Bayesian Optimization and NNSGO, are used to select the most promising geometrical configurations of projectiles to evaluate in a limited budget of 20 configurations to evaluate. Both approaches are able to find optimal configurations with approximately the same drag and that are stable. These configurations outperform the best configuration in the initial data set which had a higher drag and was not stable. In this study case, the optimal configuration using Bayesian Optimization is achieved faster (*i.e.* in fewer iterations) than with NNSGO. However, NNSGO allows testing more stable configurations. It is also shown that the procedures involving a trade-off between the exploration and exploitation objectives offer a better compromise in finding the optimal dimensions compared to other approaches such as, searching the configurations randomly, performing the exploitation or exploration of the models only.

The second research topic is discussed in Chapter 5. In this chapter the focus shifts on dynamic systems and control engineering. The goal is to explore how data-driven models of dynamic systems can be used to design controllers, or find optimal control actions, for dynamic systems and how the models can be improved if the initial one does not allow designing a controller which provides good performances. The main approach used in such cases is to identify a new and improved model then design a new controller based on that model. Such approaches are commonly employed in iterative identification and controller design procedures for linear systems.

Second contribution of the thesis : The second contribution in this thesis is to extend such procedures for the case where the systems are nonlinear and can be represented by serial block-oriented structures. More particularly, this idea is applied to the case where the true system can be represented by a Wiener structure and in a feedforward control open-loop framework. First, it was shown how an identified model of the true system can be used to design the controller which has a Hammerstein structure. Secondly, an approach which consists in using the data obtained from testing the controller on the true system and the past data, to identify a new model and then re-design the controller based on the latter is presented. Results on numerical examples show that the controller design procedure allows designing controllers which allow tracking efficiently different types of reference signals. The iterative identification approach then allows improving the designed controller in the case where the initial controller does not allow obtaining efficient results. Finally, it is illustrated that when the exact structure of the true system is known, *i.e.*, the order of the linear dynamic part and the type of nonlinearity, even better performances can be obtained from the procedure.

6.2 Perspectives

Different elements can be considered to extend the work accomplished in this thesis. First, in the procedures related to the first research topic, the data used is assumed to be noise-free. However, most data collected from real-life processes contains noise. It would therefore be interesting to explore how the iterative optimization procedure developed in this thesis, *i.e.*, NNSGO, performs with such noisy data and whether any adaptations are necessary for these cases.

In the NNSGO procedure the exploration measure δ_{min} depends on the distance between a point in the search domain and the closest one to it in the dataset used to identify the Neural Networks. However, one could argue that the closest point alone may not be sufficient to determine whether a point is well predicted or to assess if there are enough points in a given region of the dataset. A possible improvement of this exploration approach would be to use a given number n of closest points in the dataset to the one which is being evaluated, to create the exploration measure. Several research questions arise from this idea: How is the number of points n chosen? How are the distances between the evaluated point and each of the n closest points in the dataset combined? For example, could a weighted mean based on the distance to each point be used?

Another area worth exploring for the NNSGO procedure, is in the approach to account for constraints. Instead of identifying a Multi Layer Perceptron (MLP) model which estimates the value of the constraints, one could identify a MLP classifier (Murtagh, 1991) which determines if the constraints are respected or not. Such models provide as output a measure between 0 and 1 which indicates the proximity to one class or another (*e.g.*, in this case, a value close to 1 would mean that the constraints are likely to be respected, and vice versa). This measure reflects the model confidence on the prediction based on the parameters it has learnt from the data. It could thus be interesting to investigate how this measure can be incorporated in the acquisition function (2.23) and whether it can replace the measure λ (2.21).

From an application perspective, *i.e.*, in aerodynamic design, the main outlook is to extend the design parameters to a broader class of projectiles. Indeed, in the problem solved in this thesis, the input design parameters allow optimizing only a certain family of projectiles. However, many other types of projectiles cannot be described by the chosen parametrization, *e.g.*, if there are multiple locations of fins on the body. Therefore, the study can be extended by choosing a larger parametrization of the projectiles to include more families of projectiles such as spin stabilized projectiles. Another possible approach would be to move away from using a small, finite number of parameters and instead employ a 3D description of the entire projectile to account for its full geometry. This would significantly increase the computational resources needed for the prediction and optimization of design parameters but could potentially allow predicting and designing any type of flying object with only one model.

In the second topic the main perspective is to add an experiment design aspect to the procedure. In the presented procedure, only the knowledge on the model is employed to design the controller and the data acquired from testing the controller is used to increase the performance of the controller. This may require in some cases multiple iterations before designing a suitable controller if the initially identified model is not an accurate description of the true system. An improvement to this, is to design a "disturbance" signal to add to the control signal provided by the controller in order to improve the informativity in the collected data. This can be performed

iteratively at each iteration of the model identification and controller design sequence. Such ideas have been applied for linear systems see *e.g.* (Hjalmarsson, 2009) and (Bombois and Scorletti, 2012), and it would be interesting to extend them for nonlinear systems.

Appendix A

Illustration of Controller Design using NNSGO

In this section, the NNSGO procedure is applied to design a controller for an unknown system. We consider the same study case as in (Bombois and Forgione, 2022) but without including the safety constraints.

The unknown system is considered to be a linear system described by the following transfer function :

$$G(s) = \frac{10}{(s+10)(s+1)}. \tag{A.1}$$

The goal of this study is to find an optimal controller $K(s,\theta)$, parametrized with a parameter vector $\theta[k, p_1, p_2]^T$, in a closed-loop framework. The controller $K(s,\theta)$ has the following transfer function :

$$K(s,\theta) = \frac{k(p_1+s)(p_2+s)}{s(s+4.2)}, \tag{A.2}$$

and is designed to track a reference noted y_{ref}. This reference corresponds to the unit step response of the model M_{ref} defined as :

$$M_{ref}(s) = \frac{9}{s^2+4.2s+9}. \tag{A.3}$$

Since the transfer function of the system $G(s)$ is considered to be unknown, the optimal controller parameters cannot be accessed directly but have to be obtained through an optimization algorithm. In the following, the NNSGO algorithm is performed to directly estimate the controller parameters. A BO approach will also be carried out to compare the results with NNSGO.

To illustrate the methods presented in this paper the following experiment is conducted :

- A unit step response of 5 seconds is applied to the closed-loop system.
- From this step response, $n = 200$ samples are collected at a sampling rate of $T_s = 0.025s$.

The criterion to be maximised by the NNSGO and BO, in order to estimate the optimal controller, is defined as:

$$V(\theta) = \sum_{n=1}^{200} (y_{ref}(n) - y(\theta, n))^2, \tag{A.4}$$

where $y_{ref}(n)$ is the desired output at sample point n, and $y(\theta, n)$ is the obtained output at sample point n with a given parameter vector θ. The optimal controller parameters are thus estimated by solving the following Optimization problem:

$$\theta_{opt} = \arg\max_{\theta} -V(\theta). \tag{A.5}$$

Unlike in BO where the Optimization can be performed with only one initial sample, neural networks need to be provided with a sufficient number of samples in order to efficiently estimate the value of a function. To solve this optimization problem using the NNSGO, 50 controller parameters θ are chosen randomly to initialize the NN model. Among these controllers, the parameters θ_D which perform the best (*i.e.* for which $V(\theta)$ is the lowest) are for $V(\theta_D) = 0.7$. The procedure is then performed using Algorithm 2 with the following considerations:

- The number of iterations is set to 250.
- The NN model used is a MLP with 1 hidden layer with 32 neurons and a *reLu* activation function on each neuron.
- The maximum of the acquisition function provides the next controller to evaluate and is given as :

$$A(\theta) = -\hat{V}(\theta) + \beta\delta_{min}(\theta), \tag{A.6}$$

  where $\hat{V}(\theta)$, is the MLP estimate of $V(\theta)$ and $\beta = 1$.

The NNSGO is compared to BO which is performed using Algorithm 1 with the following considerations :

- The first controller to evaluate is chosen randomly
- The number of iterations is set to 300. The 300 iterations are chosen here to have the exact same number of observed points as with the NNSGO procedure (50 initial samples + 250 tested).
- A GP model with an RBF kernel (1.15) is used.
- The maximum of the acquisition function which provides the next controller to evaluate corresponds to the UCB (2.2.1) where $\mu(\theta)$ is the GP estimate of $-V(\theta)$, $\sigma(\theta)$ the standard deviation provided by the GP model and $\beta = 1$.

The great advantage of these two approaches (NNSGO and BO) is that the optimal controller parameters are estimated without requiring the identification of a model for the process to be controlled. Indeed, the MLP and GP both model the static relationship between the parameter vector θ and the performance of the controller on the model: $V(\theta)$.

After performing 250 iterations of NNSGO using Algorithm 2, the best controller parameters among the tested ones are $\theta_{NNSGO} = [1, 8, 1.1]^T$ with $V(\theta_{NNSGO}) = 0.02$. By using the NNSGO procedure, we are able to find a controller θ_{NNSGO} for which the value of V is 98% lower than the V of θ_D (the controller with the lowest value of V in the initial dataset). The best controller found with BO after performing Algorithm 1 for 300 iterations is $\theta_{BO} = [1, 8.4, 0.98]^T$ with $V(\theta_{BO}) = 0.01$.

The time to perform the optimization for both approaches is of approximately 5 seconds each (Note that this time can slightly vary depending on the time that *Scipy* takes to calculate the response of a particular tested controller). In this case study, BO performs better than the NNSGO procedure since it finds a controller with a smaller value of V than the one found using the NNSGO procedure in the same optimization time. However, the controller provided by the NNSGO procedure remains close to the optimal controller parameters which are $\theta_{opt} = [0.9, 10, 1]^T$.

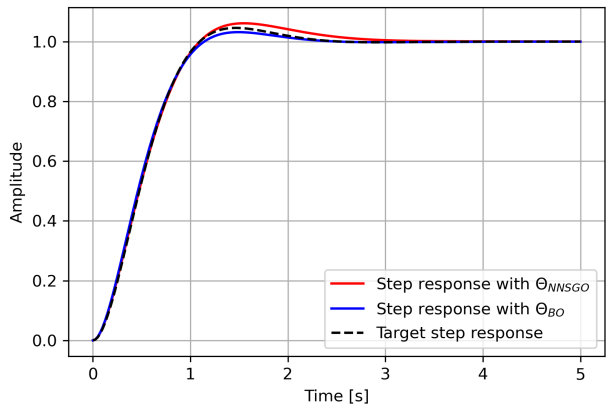


Figure A.1: Target step response (in black) compared to the step response based on the controllers found using the NNSGO approach (θ_{NNSGO} in red) and BO (θ_{BO} in blue)

Figure A.1 presents the obtained response with the best controllers found using NNSGO (θ_{NNSGO}) and using BO (θ_{BO}), compared to the target response associated to M_{ref} (A.3). As it can be seen, the response of the system when θ_{NNSGO} is used as the controller parameters is also close to the target response. Therefore, the procedure can be used as an alternative to BO for this controller design problem.

Bibliography

Aarnoudse, L., Ohnishi, W., Poot, M., Tacx, P., Strijbosch, N., and Oomen, T. (2021). Control-relevant neural networks for intelligent motion feedforward. In *2021 IEEE International Conference on Mechatronics (ICM)*, pages 1–6. IEEE.

Akian, J.-L., Bonnet, L., Owhadi, H., and Savin, É. (2022). Learning "best" kernels from data in gaussian process regression. with application to aerodynamics. *Journal of Computational Physics*, 470:111595.

Albisser, M. (2015). *Identification of the aerodynamic coefficients from free flight data.* PhD thesis, University of Lorraine, ISL Report 118/2015.

Alvarez, M. A., Rosasco, L., Lawrence, N. D., et al. (2012). Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266.

Amari, S.-i. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196.

Anderson, J. D. and Wendt, J. (1995). *Computational fluid dynamics*, volume 206. Springer.

Arenzana, R. C., López-Lopera, A. F., Mouton, S., Bartoli, N., and Lefebvre, T. (2021). Multi-fidelity gaussian process model for cfd and wind tunnel data fusion. In *AeroBest 2021*.

Arnoult, G., Zeidler, M., and Garnier, E. (2020). Control surface geometry surrogate-based optimization for spin-stabilized projectile course correction. *AIAA journal*, 58(2):550–560.

Bemporad, A. (2020). Global optimization via inverse distance weighting and radial basis functions. *Computational Optimization and Applications*, 77(2):571–595.

Biau, G. and Scornet, E. (2016). A random forest guided tour. *Test*, 25:197–227.

Bishop, C. M. (1995). *Neural networks for pattern recognition.* Oxford university press.

Bombois, X. and Forgione, M. (2022). Control design via bayesian optimization with safety constraints. In *2022 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1421–1426. IEEE.

Bombois, X. and Scorletti, G. (2012). Design of least costly identification experiments: The main philosophy accompanied by illustrative examples. *Journal Européen des Systèmes Automatisés (JESA)*, 46(6-7):587–610.

Bristow, D. A., Tharayil, M., and Alleyne, A. G. (2006). A survey of iterative learning control. *IEEE control systems magazine*, 26(3):96–114.

Browne, M. W. (2000). Cross-validation methods. *Journal of mathematical psychology*, 44(1):108–132.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208.

Chen, H., He, L., Qian, W., and Wang, S. (2020). Multiple aerodynamic coefficient prediction of airfoils using a convolutional neural network. *Symmetry*, 12(4):544.

Cook, M. V. (2012). *Flight dynamics principles: a linear systems approach to aircraft stability and control*. Butterworth-Heinemann.

Dolanc, G. and Strmenik, S. (2008). Design of a nonlinear controller based on a piecewise-linear hammerstein model. In *2008 IEEE International Conference on Control Applications*, pages 1007–1012. IEEE.

Driver, J. and Zingg, D. W. (2007). Numerical aerodynamic optimization incorporating laminar-turbulent transition prediction. *AIAA journal*, 45(8):1810–1818.

Dupuis, A. D. and Hathaway, W. (1997). Aeroballistic range tests of the basic finner reference projectile at supersonic velocities. Technical report, Defence Research Establishment Valcatier (Quebec).

Fernández-Godino, M. G. (2016). Review of multi-fidelity models. *arXiv preprint arXiv:1609.07196*.

Firouz, Y., Goutam, S., Soult, M. C., Mohammadi, A., Van Mierlo, J., and Van den Bossche, P. (2020). Block-oriented system identification for nonlinear modeling of all-solid-state li-ion battery technology. *Journal of Energy Storage*, 28:101184.

Forgione, M., Bombois, X., and Van den Hof, P. M. (2015). Data-driven model improvement for model-based control. *Automatica*, 52:118–124.

Forgione, M. and Piga, D. (2021). dynonet: A neural network architecture for learning dynamical systems. *International Journal of Adaptive Control and Signal Processing*, 35(4):612–626.

Forrester, A., Sobester, A., and Keane, A. (2008). *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons.

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Gardner, J. R., Kusner, M. J., Xu, Z. E., Weinberger, K. Q., and Cunningham, J. P. (2014). Bayesian optimization with inequality constraints. In *ICML*, volume 2014, pages 937–945.

Gelbart, M. A. (2015). *Constrained Bayesian optimization and applications*. PhD thesis.

Gevers, M. (2002). A decade of progress in iterative process control design: from theory to practice. *Journal of process control*, 12(4):519–531.

Giri, F. and Bai, E.-W. (2010). *Block-oriented nonlinear system identification*, volume 1. Springer.

Goan, E. and Fookes, C. (2020). Bayesian neural networks: An introduction and survey. *Case Studies in Applied Bayesian Data Science: CIRM Jean-Morlet Chair, Fall 2018*, pages 45–87.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. (2018). Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377.

Gutmann, H.-M. (2001). A radial basis function method for global optimization. *Journal of global optimization*, 19(3):201–227.

Guzman, R., Oliveira, R., and Ramos, F. (2020). Heteroscedastic bayesian optimisation for stochastic model predictive control. *IEEE Robotics and Automation Letters*, 6(1):56–63.

Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., and Scholkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28.

Hjalmarsson, H. (2009). System identification of complex and structured systems. *European journal of control*, 15(3-4):275–310.

Jakobsson, S., Patriksson, M., Rudholm, J., and Wojciechowski, A. (2010). A method for simulation based optimization using radial basis functions. *Optimization and Engineering*, 11:501–532.

Jeong, S., Murayama, M., and Yamamoto, K. (2005). Efficient optimization design method using kriging model. *Journal of aircraft*, 42(2):413–420.

Kalafatis, A. D., Wang, L., and Cluett, W. R. (2005). Linearizing feedforward–feedback control of ph processes based on the wiener model. *Journal of Process Control*, 15(1):103–112.

Kennedy, M. C. and O'Hagan, A. (2000). Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13.

Ketkar, N. and Ketkar, N. (2017). Introduction to keras. *Deep learning with python: a hands-on introduction*, pages 97–111.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kou, J. and Zhang, W. (2019). Multi-fidelity modeling framework for nonlinear unsteady aerodynamics of airfoils. *Applied Mathematical Modelling*, 76:832–855.

Kramer, O. (2011). Unsupervised k-nearest neighbor regression. *arXiv preprint arXiv:1107.3600*.

Kuya, Y., Takeda, K., Zhang, X., and Forrester, A. I. (2011). Multifidelity surrogate modeling of experimental and computational aerodynamic data sets. *AIAA journal*, 49(2):289–298.

Le Gratiet, L. (2013). *Multi-fidelity Gaussian process regression for computer experiments*. PhD thesis, Université Paris-Diderot-Paris VII.

Lee, S., Kang, S., and Lee, K. (2021). Development of a gaussian process modeling application for the prediction of missile aerodynamic coefficients. *Journal of Mechanical Science and Technology*, 35:987–997.

Liashchynskyi, P. and Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv preprint arXiv:1912.06059*.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528.

Loh, W.-Y. (2011). Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23.

Marchant, R. and Ramos, F. (2012). Bayesian optimisation for intelligent environmental monitoring. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 2242–2249. IEEE.

Martins, J. R. (2022). Aerodynamic design optimization: Challenges and perspectives. *Computers & Fluids*, 239:105391.

McCoy, R. (1999). *Modern exterior ballistics: the launch and flight dynamics of symmetric projectiles*. Schiffer.

Moin, H., Khan, H. Z. I., Mobeen, S., and Riaz, J. (2022). Airfoil's aerodynamic coefficients prediction using artificial neural network. In *2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 175–182. IEEE.

Morelli, E. A. and Klein, V. (2016). *Aircraft system identification: theory and practice*, volume 2. Sunflyte Enterprises Williamsburg, VA.

Muhammad, D., Ahmad, Z., and Aziz, N. (2021). Modeling of low density polyethylene tubular reactor using nonlinear block-oriented model. *Materials Today: Proceedings*, 42:39–44.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197.

Naitali, A., Giri, F., Elayan, E., and Haloua, M. (2008). Biosocial culture inspired hierarchical algorithm for miso block oriented nonlinear system identification: application to ozone modelling. *IFAC Proceedings Volumes*, 41(2):7433–7438.

Oh, S. (2020). Comparison of a response surface method and artificial neural network in predicting the aerodynamic performance of a wind turbine airfoil and its optimization. *Applied Sciences*, 10(18):6277.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Poot, M., van Hulst, J., Yan, K. W., Kostić, D., Portegies, J., and Oomen, T. (2023). Feedforward control in the presence of input nonlinearities: With application to a wirebonder. *IFAC-PapersOnLine*, 56(2):1895–1900.

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer.

Regis, R. G. and Shoemaker, C. A. (2007). A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19(4):497–509.

Renganathan, S. A., Maulik, R., and Ahuja, J. (2021). Enhanced data efficiency using deep neural networks and gaussian processes for aerodynamic design optimization. *Aerospace Science and Technology*, 111:106522.

Rollins, D. K., Mei, Y., Loveland, S. D., and Bhandari, N. (2016). Block-oriented feedforward control with demonstration to nonlinear parameterized wiener modeling. *Chemical Engineering Research and Design*, 109:397–404.

Schoukens, M. (2015). Identification of parallel block-oriented models starting from the best linear approximation. *Brussels, Belgium: Vrije Universiteit Brussel (VUB)*.

Schoukens, M., Bai, E.-W., and Rolain, Y. (2012). Identification of hammerstein-wiener systems. *IFAC Proceedings Volumes*, 45(16):274–279.

Schoukens, M., Marconato, A., Pintelon, R., Vandersteen, G., and Rolain, Y. (2015). Parametric identification of parallel wiener–hammerstein systems. *Automatica*, 51:111–122.

Secco, N. R. and Mattos, B. S. d. (2017). Artificial neural networks to predict aerodynamic coefficients of transport airplanes. *Aircraft Engineering and Aerospace Technology*, 89(2):211–230.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015). Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR.

Sola, J. and Sevilla, J. (1997). Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on nuclear science*, 44(3):1464–1468.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. W. (2012). Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE transactions on information theory*, 58(5):3250–3265.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359.

Tumse, S., Bilgili, M., and Sahin, B. (2022). Estimation of aerodynamic coefficients of a non-slender delta wing under ground effect using artificial intelligence techniques. *Neural Computing and Applications*, 34(13):10823–10844.

Valayer, H., Bartoli, N., Castaño-Aguirre, M., Lafage, R., Lefebvre, T., López-Lopera, A. F., and Mouton, S. (2024). A python toolbox for data-driven aerodynamic modeling using sparse gaussian processes. *Aerospace*, 11(4):260.

Van Den Hof, P. M. and Schrama, R. J. (1995). Identification and control—closed-loop issues. *Automatica*, 31(12):1751–1770.

Van Hulst, J., Poot, M., Kostić, D., Yan, K. W., Portegies, J., and Oomen, T. (2022). Feedforward control in the presence of input nonlinearities: a learning-based approach. *IFAC-PapersOnLine*, 55(37):235–240.

Vinter, R. B. and Vinter, R. (2010). *Optimal control*, volume 2. Springer.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272.

Vu, K. K., d'Ambrosio, C., Hamadi, Y., and Liberti, L. (2017). Surrogate-based methods for black-box optimization. *International Transactions in Operational Research*, 24(3):393–424.

Welling, M. (2013). Kernel ridge regression. *Max Welling's classnotes in machine learning*, pages 1–3.

Xiang, Y., Sun, D., Fan, W., and Gong, X. (1997). Generalized simulated annealing algorithm and its application to the thomson model. *Physics Letters A*, 233(3):216–220.

Yetkin, S., Abuhanieh, S., and Yigit, S. (2024). Investigation on the abilities of different artificial intelligence methods to predict the aerodynamic coefficients. *Expert Systems with Applications*, 237:121324.

Zipfel, P. (2000). *Modeling and simulation of aerospace vehicle dynamics*. American Institute of Aeronautics and Astronautics.

Résumé

Dans de nombreuses applications techniques, l'objectif est d'optimiser les performances d'un système. Pour ce faire, il est nécessaire d'évaluer la sortie du système afin d'analyser la performance obtenue. Cependant, dans plusieurs scénarios concrets, les systèmes sont souvent coûteux à évaluer, rendant la tâche d'optimisation complexe. Pour remédier à ce problème, des modèles basés sur les données sont souvent identifiés afin d'estimer les fonctions objectives coûteuses, associées aux systèmes, puis sont utilisés pour estimer l'optimum. Si de mauvaises performances sont obtenues avec ces modèles, ces derniers doivent être améliorés en les ré-identifiant avec des nouvelles données. En revanche, comme l'évaluation des systèmes est coûteuse, ces données doivent être sélectionnées avec soin. L'objectif de cette thèse est de développer des approches qui peuvent être utilisées pour améliorer les modèles identifiés employés dans l'optimisation de la performance des systèmes. Ces approches sont utilisées dans deux applications différentes.
La première concerne la conception aérodynamique, où le but est de déterminer les dimensions optimales d'un projectile sur la base de critères associés aux coefficients aérodynamiques. Ces coefficients sont coûteux à obtenir, ainsi, les configurations géométriques du projectile à évaluer, afin de trouver l'optimum, exigent une sélection attentive. Pour ce faire, des approches telles que l'optimisation bayésienne sont généralement utilisées, où un modèle de processus gaussien est utilisé pour représenter la relation statique entre la configuration du projectile et la fonction objective. Dans cette thèse, une procédure similaire à l'optimisation bayésienne, mais où des réseaux de neurones sont utilisés comme modèles à la place des processus gaussiens, est développée, afin de mieux gérer les ensembles de données plus volumineux. Ces deux approches sont utilisées pour résoudre le problème de conception aérodynamique, et il est démontré qu'elles permettent de réduire les coûts associés à l'optimisation.
La seconde application concerne le domaine de contrôle et, plus particulièrement, l'identification pour le contrôle. L'accent est mis sur la conception de regulateurs feed-forward pour des systèmes non linéaires pouvant être représentés par des structures de type Wiener. Plus précisément, il est montré comment un modèle de ces systèmes peut être utilisé pour concevoir le regulateur. Une procédure d'amélioration itérative du modèle et de re-conception du regulateur est également introduite dans le cas où le regulateur initialement conçu ne permet pas d'obtenir des performances optimales.
Globalement, les approches développées offrent des solutions efficaces pour minimiser les coûts d'évaluation des systèmes lors des tâches d'optimisation dans diverses applications d'ingénierie.

Mots-clés: Identification, Optimisation, Apprentissage Machine, Conception Aérodynamique, Contrôle Feed-Forward, Systèmes Wiener.

Abstract

In many engineering applications, the objective is to find the optima performances of a system. To evaluate the performances, it is necessary to measure the output of the system for given inputs. However, in several real-life scenarios, systems are often expensive to evaluate making it difficult to perform the optimization tasks. To address this issue, data-driven models are often identified to estimate the expensive objective functions, associated with the systems, and are employed to approximate their optimum. If poor performances are obtained using these models, they must be improved by re-identifying them with new data. However, since the systems are expensive to evaluate, the data must be chosen carefully. The aim of this thesis is to develop approaches which can be used to improve identified models employed in system performance optimization.

These approaches are applied in two different applications. The first one is the aerodynamic design where the goal is to find the optimum dimensions of a projectile based on criteria associated with aerodynamic coefficients. These coefficients are costly to acquire, hence the projectile geometry configurations to evaluate, to find the optimum, must be selected with care. This is usually achieved using approaches such as Bayesian Optimization where a Gaussian Process model is employed to model the static relationship between the projectile configuration and the objective function. In this thesis, a procedure similar to Bayesian Optimization but where Neural Networks are employed as data-driven models instead of Gaussian Processes is developed, to enable scalability for larger datasets. Both approaches are used to solve the aerodynamic design problem, and it is shown that they allow reducing the costs associated to aerodynamic optimization.

The second application concerns control engineering and more particularly the framework of identification for control. The focus is on feedforward controller design for nonlinear systems which can be represented by Wiener structures. More particularly, it is shown how a model of such systems can be used to design the controller. A procedure to iteratively improve the model and re-design the controller is also introduced in the case where the initially designed one does not allow obtaining optimal performances.

Overall, the developed approaches provide effective solutions to minimize system evaluation costs during optimization tasks in diverse engineering fields.

Keywords: Identification, Optimization, Machine Learning, Aerodynamic design, Feed-Forward control, Wiener systems.