EPFL

Thèse n° 10 433

Optimization Methods for Control: From Embedded Programmable Hardware to Data-Driven Process Optimization

Présentée le 19 février 2021

Faculté des sciences et techniques de l'ingénieur
Laboratoire d'automatique 3
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Harsh Ambarishkumar SHUKLA

Acceptée sur proposition du jury

Prof. A. Karimi, président du jury
Prof. C. N. Jones, directeur de thèse
Prof. D. Limón, rapporteur
Dr E. C. Kerrigan, rapporteur
Prof. G. Ferrari Trecate, rapporteur

École polytechnique fédérale de Lausanne

2021

I am not handsome but I can give my hand to someone who needs help...
Because beauty is required in the heart, not in face.
— A. P. J. Abdul Kalam

To my loveliest family and to everyone who has supported and cared...

Abstract

The research community has been making significant progress in hardware implementation, numerical computing and algorithm development for optimization-based control. However, there are two key challenges that still have to be overcome for optimization-based control to be a viable option in the context of advanced industrial applications. First, the large existing gap between algorithm development and its deployment on platforms used by practitioners in industry. Second, from a more theoretical viewpoint, the lack of robustness of certain approaches, which are based on the unreasonable assumption that the model at hand perfectly represents the object under investigation. This thesis addresses the aforementioned challenges by establishing software toolboxes for automatic code generation, and proposing a data-driven methodology to enhance the performance of real-time optimization strategies during operation.

The first part of this thesis focuses on the efficient implementation of Model Predictive Control (MPC) based on first-order operator splitting methods. Because of the cheap numerical operations associated with them, splitting methods are favorable candidates for applications with limited computing power. We first identify the computational bottlenecks and, subsequently, discuss their efficient deployment on processors, Field Programmable Gate Arrays (FPGA), and heterogeneous platforms. For rapid prototyping and deployment, two code generation toolboxes are developed: SPLIT and LAFF. These possess a high-level parsing interface for MATLAB and yield optimized C code that can be directly used in a variety of FPGA platforms. Features such as pipelining, memory partitioning, and parallelization are automatically incorporated, not requiring users to have in-depth knowledge about computer architecture and low-level programming. We then propose a framework to a priori solve the co-design problem arising in splitting method-based MPC to provide trade-offs between resources and latency. We provide analytical expressions that can avoid the daunting and time-consuming task of exploring the design space manually, thus reducing the final application development time.

Abstract

The second part of the thesis deals with learning plant-model mismatch using Gaussian processes (GPs) in Real Time Optimization (RTO) schemes. Inaccurate models, the presence of disturbances, and time-varying conditions typically lead to the suboptimal operation of many plants. We use data-driven global surrogate models in the form of GPs to cope with such problems and show better numerical convergence and handling of noise effectively when compared to standard RTO techniques. We moreover prove that GPs can be certified as probabilistic and deterministic fully linear models, a key property to guarantee global convergence of derivative-free trust region (DFT) methods. We then propose a novel DFT methodology to incorporate noise, which requires less plant evaluations than other alternatives. Finally, we conclude this work by performing experiments on a Solid-Oxide Fuel Cell system.

Keywords: Model predictive control; Splitting methods; FPGAs; Co-design problems; Code-generation; Embedded platforms; Real-time optimization; Gaussian processes; Derivative-free trust region method; Fully linear model; Solid-Oxide Fuel cells

Résumé

La communauté des chercheurs a fait des progrès significatifs dans la mise en œuvre du matériel, du calcul numérique et du développement d'algorithmes pour le contrôle basé sur l'optimisation. Toutefois, il reste deux défis majeurs à relever pour que le contrôle basé sur l'optimisation soit une option viable dans le contexte des applications industrielles avancées. Premièrement, l'écart important qui existe entre le développement d'un algorithme et son déploiement sur les plateformes utilisées par les praticiens dans l'industrie. Deuxièmement, d'un point de vue plus théorique, le manque de robustesse de certaines approches, qui sont basées sur l'hypothèse déraisonnable que le modèle en question représente parfaitement l'objet étudié. Cette thèse aborde les défis susmentionnés en établissant des boîtes à outils logicielles pour la génération automatique de code, et en proposant une méthodologie basée sur les données pour améliorer la performance des stratégies d'optimisation en temps réel pendant l'exploitation.

La première partie de cette thèse se concentre sur la mise en œuvre efficace d'une commande prédictive (MPC) basé sur des méthodes de séparation d'opérateurs de premier ordre. En raison des opérations numériques peu coûteuses qui leur sont associées, les méthodes de séparation sont des candidats idéals pour les applications dont la puissance de calcul est limitée. Nous identifions d'abord les goulets d'étranglement en matière de calcul et, ensuite, nous discutons de leur déploiement efficace sur les processeurs, les circuits logiques programmables (FPGA) et les plates-formes hétérogènes. Pour le prototypage et le déploiement rapides, deux boîtes à outils de génération de code sont développées : SPLIT et LAFF. Celles-ci possèdent une interface d'analyse de haut niveau pour MATLAB et produisent du code C optimisé qui peut être directement utilisé dans diverses plates-formes FPGA. Des fonctionnalités telles que le pipelining, le partitionnement de la mémoire et la parallélisation sont automatiquement incorporées, ne nécessitant pas de connaissances approfondies des utilisateurs en matière d'architecture informatique et de programmation de bas niveau. Nous proposons ensuite un cadre permettant de résoudre à priori le problème de co-création qui se pose lors de l'utilisation de méthodes de

fractionnement MPC, afin de fournir un compromi entre les ressources nécessaires et la latence. Nous fournissons des expressions analytiques qui permettent d'éviter l'exploration manuellement et fastidieuse de l'espace de conception, réduisant ainsi le temps de développement final de l'application.

La deuxième partie de la thèse porte sur l'apprentissage de l'inadéquation système-modèle en utilisant des processus gaussiens (GP) dans des schémas d'optimisation en temps réel (RTO). Des modèles imprécis, la présence de perturbations et des conditions variables dans le temps conduisent généralement à un fonctionnement sous-optimal de nombreuses systèmes. Nous utilisons des modèles de substitution globaux pilotés par les données sous la forme de GP pour faire face à ces problèmes et montrer une meilleure convergence numérique et un traitement efficace du bruit par rapport aux techniques RTO standard. Nous prouvons en outre que les GP peuvent être certifiés comme des modèles probabilistes et déterministes entièrement linéaires, une propriété clé pour garantir la convergence globale des méthodes de régions de confiance sans dérivés (DFT). Nous proposons ensuite une nouvelle méthodologie DFT pour intégrer le bruit, qui nécessite moins d'évaluations des systèmes que les autres alternatives. Enfin, nous concluons ce travail en réalisant des expériences sur un système de piles à combustible à oxyde solide.

Mots-clés : Commande prédictive ; Méthodes de séparation ; Circuits logiques programmables (FPGA) ; Problèmes de co-création ; Génération de code ; Plates-formes embarquées ; Optimisation en temps réel ; Processus gaussiens ; Méthode des régions de confiance sans dérivés ; Modèle entièrement linéaire ; Piles à combustible à oxyde solide

Acknowledgements

Can we survive technology?

John von Neumann

$$\text{Life} := \int_{\text{birth}}^{\text{death}} (\ \text{Present\_moment}\ )\, dt + L_0,$$

where L_0 is the initial condition at birth. For example, for an Indian, it is the past karmas or for an aristocrat, it is the family he is born in. In any case, the undeniable fact is: whatever we achieve, it comes with the contribution of many people whom we have stumbled upon in our lives.

First of all, I want to thank Prof. C. N. Jones. Colin, thank you for giving me the freedom to drive this research the way I wanted. I could not have survived without your guidance, patience, trust, encouragement, and very kind support. Your length and breadth of knowledge on different research topics, constructive thinking, calm attitude, and foresight in posing a research question have profoundly inspired me. You have been not only a great supervisor, but a great mentor as well. I also had the privilege to work with Dr. E. C. Kerrigan, Prof. D. Bonvin, and Prof. T. Faulwasser. I thank Eric for hosting me twice at Imperial College London. Your unique approach to research (there is no free lunch, so where is the trade-off?) has been of tremendous help in attacking scientific problems. Dominique and Timm, thank you for helping me critically introspect the research I was pursuing. It is thanks to your guidance that I learned posing critical questions such as "can I convince an engineer working in the industry to use my research?" I thank you all for also being great mentors. This acknowledgment would be incomplete without extending my sincere thanks to the esteemed members of the jury who dedicated their time to read my thesis and provided me with insightful comments. Thank you, Prof. D. Limon, G. Ferrari-Trecate, E. C. Kerrigan, and A. Karimi, as your feedback improved the quality and delivery of this thesis. I was fortunate enough to collaborate with amazing colleagues. Georgios, thanks for fascinating me with multiple scientific and philosophical discussions; Jean, for the first paper that I

ever wrote. Bulat and Tafarel, I just want to say that this thesis would have a very different title and content without you two! I thank Dr. S. Almer and Dr. J. Ferreau for hosting me at ABB Switzerland. Dr. Salzmann, please add another LA student in your list of students whose studies would not have finished without your help. Thank you to all LA secretaries (Nicole, Ruth, Eva, and Margot) for being extremely cooperative and helpful with administrative work.

I am extremely grateful to all my teachers and professors. Special thanks goes to Dr. J. P. Dave, Prof. V. A. Shah, Prof. S. N. Sharma, and Prof. G. J. Joshi for their constant encouragement. Prof. D. Charkraborty for introducing me the world of research. Prof. J. van Wingerdan, Prof. M. Mazo, and Prof. A. Haber for guiding me during my stay at TU-Delft.

Being an INFJ (I like the pseudo-science of MBTI cognitive functions), I would wander around spending ample time alone, and would interact with strangers. I am thankful to all those strangers who shared with me diverse aspects of their lives; I got to learn a lot from you. Given my humble background, I would not have been able to pursue my studies without publicly funded education. I therefore extend my sincere thanks to Indian and European taxpayers thanks to whom I could afford my studies. I thank everyone who has directly or indirectly contributed to my life and this work.

I would like to thank all my friends from SVNIT, IAESTE Paderborn, IIT Bombay, TU Delft, Imperial College London, and EPFL. I am aware it is grossly unfair that I am not even mentioning your names here given how much you have shaped my life. However, I am constrained by space and I sincerely apologize. Please remember that I often reminisce about the beautiful time we spent together. All the LA members during my stay, thank you for all the research-related discussions and your extremely valuable help, and for making environment at LA fun and bearing with my idiosyncratic behavior at occasions. I thank TEMPO colleagues and have to admit that guys, you rock!

I left my home 14 years ago pursuing studies, and many of my friends—Adi, Paolo, Aayush, Tafarel, Valentine, Shruti, Lucie, Bulat, Pratik, Emilio, Kashyap, Tomasz, Pulkit, Bhanu, Nilay, Carito, Praveen, Cedric, Sophia, and to name a few—have become like my family members. You witnessed me passing through all different phases in my life, and were always there sharing those moments with me. I have no words to express my gratitude to you, so I abstain from futile attempts here.

I am looking forward to more of our adventures to come! Maddalena, you came into my life at a time of great uncertainty, both internal and external. Even with the world bearing the brunt of a pandemic, all I felt around you was a sense of safety and security. Thank you for your love, understanding, and patience. I am truly grateful to have found someone I can not only rely on, but also seek comfort in.

Thank you, papa, for inspiring me to pursue my dreams, and mum, for your unconditional love. Big thanks to my sister for her constant support. I will be eternally indebted to all of you. I thank all my family members without whom this work would not even have happened in my wildest dreams.

As I say,

"Life is an MPC problem",

and I used to be scared about the imprecise model that I have about my life and, more than that, about unexpected disturbances. But with time, I guess, I have started appreciating model-imprecisions and unknown disturbances, and I thank everybody who enabled me appreciating it.

Lausanne, December 4, 2020

Peace and Love,
H. A. S.

Contents

Introduction and Outline

> I was born not knowing and have had only a little time to change that here and there.
>
> Richard Feynman

0.1 Introduction and Outline

With the tremendous increase in computational power and the availability of powerful algorithms, optimization-based control has gained significant attention since the 1980s. As a consequence of the efforts made by the community working on the topic, considerable progress has been made both in hardware implementation and in numerical computing. However, two key limitations still exist, which prevent optimization-based control from becoming an appealing solution in advanced industrial applications. First, a large gap between algorithm development and algorithm deployment, especially on commercial embedded platforms accepted by industry. Second, poor performance of solutions that strongly rely on models due to the unavoidable mismatch between it and the physical plant under study. This thesis aims to address these two challenges and is divided in two parts.

0.1.1 Embedded Optimization on Programmable Hardware

The first two chapters of the thesis focus on efficient implementation of first-order operator splitting methods on embedded platforms. The advantage of splitting methods is that they decompose the original optimization problem into subproblems, which are computationally cheaper to solve than the original one. This feature makes them ideal candidates for deployment on resource-limited embedded platforms. We particularly focus on implementing splitting method-based model predictive control

on embedded platforms. Model predictive control is a control technique that targets the minimization of a predefined cost function satisfying design constraints. We first analyze the computational and memory requirements of numerical operations involved in various splitting methods, and then examine the operations that represent bottlenecks for deployment. Finally, we study different control problems and the sparsity patterns arising from them, which is then exploited to achieve an efficient implementation.

In Chapter 3, a code-generation toolbox called SPLIT is proposed. SPLIT is an open source and free package that implements various splitting methods on different embedded platforms. The toolbox supports code-generation for embedded processors, Field Programmable Gate Arrays (FPGAs) and System on Chips (SOCs). The toolbox enables users to rapidly prototype and deploy splitting methods without getting lost in the numerous architectural details or their target hardware. To that end, a high-level MATLAB interface is provided for parsing the problem. The toolbox generates C-based code that is tailored to the hardware at hand, exploring features such as pipelining, parallelism, and memory management. When handling heterogeneous platforms, SPLIT also automatizes the procedure of sharing computations amongst FPGAs and processors.

In Chapter 4, we present a framework for solving co-design problems *a-priori*. Our approach yields closed-form expressions that accurately predict both latency and consumed resources for FPGA-tailored C code. The *a-priori* aspect of it dispenses with the need of synthesizing the code, one very time-consuming task. The central idea is to break down the general algorithm in various fundamental blocks and analytically compute the latency and resources required to execute each of these operations. Next, we propose another code-generation toolbox: LAFF. As opposed to SPLIT, LAFF is not limited to first-order methods and generates code only to FPGA boards. Again, an easy-to-use high-level interface in MATLAB is provided, enabling rapid prototyping without requiring knowledge about hardware description languages. In contrast to existing tools, we solve a co-design problem that can once more *a priori* estimate execution time and consumed resources, thereby saving time, cost, and energy. The level of trade-off between algorithm execution time and consumed resources can be entirely controlled by the user by varying a parallelization level parameter. To show the efficacy of LAFF, we deploy linear Model Predictive Control (MPC) on Field Programmable Gate Arrays (FPGAs) in different resource scenarios.

0.1.2 Gaussian Processes-Based Constrained Process Optimization

In the context of real-time optimization of process systems, measurement-based approaches allows for dealing with structural plant-model mismatch and parametric uncertainties. Modifier-adaptation schemes are measurement-based and rely on first-order corrections to the model cost and constraint functions so as to achieve plant optimality upon convergence. However, first-order corrections rely crucially on the estimation of plant gradients, which typically require additional plant experiments.

A real-time optimization approach is proposed in Chapter 6 importing non-parametric tools from the machine learning community. Specifically, to both avoid plant-gradient estimation and attenuate measurement noise, we propose to estimate the existing mismatch via recursive Gaussian processes. We use real-time optimization data to build Gaussian-process surrogate functions and, by doing so, higher-order correction terms are possible. The application of the proposed scheme is illustrated via a constrained variant of the Williams-Otto reactor problem.

Chapter 7 considers Gaussian Processes (GP)s as global surrogate models in derivative-free trust-region methods. It is well known that derivative-free trust-region methods converge globally—provided that the surrogate model is probabilistically fully linear. We prove that GPs are indeed probabilistically fully linear, thus resulting in fast (compared to linear or quadratic local surrogate models) and global convergence. We draw upon the optimization of a chemical reactor to demonstrate the efficiency of GP-based trust-region methods.

Chapter 8 has the following two contributions:

- *A generalised and superior version of the derivate-free trust-region method:* We propose an algorithm–DMT– which relaxes the necessary convergence condition of trust-region surrogate models: A fully-linear property. We prove that DMT guarantees convergence to a neighborhood of a local optimum solution. Our algorithm requires a less stringent condition on the surrogate model and needs less certification checks compared with traditional methods. Consequently, it can handle measurement noise better and shows faster convergence in experiments.

- *Certification of machine learning-based surrogate models:* We prove that Gaussian process-based surrogate models can be certified as fully-linear models, a

key property required for convergence of trust-region methods. Since machine learning based models are known to perform better compared with traditional surrogate models in many applications, our work will enable combining them with derivative-free methods while guaranteeing convergence.

Finally, we present in Chapter 9 the results of two experiments carried out on a solid-oxide fuel cell. In the first experiment, we verified the strengths of GPs as global surrogate models in the RTO context. In the second experiment, based on practical observations, we propose and discuss a novel method to compute the modifiers without perturbing the plant. Concluding remarks and possible future investigations are given in Chapter 10.

0.2 Collaborations

Chapter 1 and 2 are based on the paper [1]. The contents of Chapter 1 are the fruit of a collaboration with Dr. G. Stathopoulos. Chapter 3 is based on [2]. Chapters 3 and 4 are based on a joint work with Dr. B. Khusainov and Dr. E. C. Kerrigan. Chapter 5 to 9 are outcomes of a collaboration with Dr. T. de Avila Ferreira, Prof. T. Faulwasser, and Prof. D. Bonvin. Chapter 6 is based on the article [3]. The Solid-Oxide Fuel Cell experiments were performed at the GEM laboratory supervised by Dr. J. Van Herle.

Part I
Embedded Optimization on Programmable Hardware

1 Solving Model Predictive Control Problems with Splitting Methods

> I can't go to a restaurant and order food because I keep looking at the fonts on the menu. Five minutes later I realize that it's also talking about food.
>
> Donald Knuth

The significant progress that has been made in recent years both in hardware implementations and in numerical computing has rendered real-time optimization-based control a viable option when it comes to advanced industrial applications[1]. More recently, the need for control of a process in the presence of a limited amount of hardware resources has triggered research in the direction of embedded optimization-based control. Many efficient high-speed solvers have been developed for both linear and nonlinear control, based on either *first order methods* (FiOrdOs [4], QPgen [5],[6], DuQuad [7], OSQP [8]), *interior point (IP) methods* (FORCES [9], CVXGEN [10]) and *active set methods* (QPOASES [11]).

In this part of the thesis, we focus on systems with linear dynamics, giving rise to convex control problems. We first briefly explore a family of first order methods known as *decomposition schemes* or *operator splitting methods*. The abstract form of the problem at hand is the minimization of the sum of two convex functions subject to linear equality constraints, and can be written as

$$\text{minimize} \quad f(z) + g(Lz) \ , \tag{1.0.1}$$

[1] The material of this chapter was written with Dr. Giorgos Stathopoulos during a collaborative work.

with variables $z \in \mathbb{R}^n$, where f and g are closed, proper convex functions and $L : \mathbb{R}^n \to \mathbb{R}^p$ is a linear map. A splitting method can be applied to the above problem after rewriting it as

$$\begin{aligned} &\text{minimize} && f(z) + g(y) \\ &\text{subject to} && Lz = y \ , \end{aligned} \tag{1.0.2}$$

by alternatively (or simultaneously) minimizing over y and z. Clearly, the solutions of problems (1.0.2) and (1.0.1) are identical. Inequality constraints that might appear are already embedded in one of the two functions in the form of indicator functions, *i.e.*, a membership function for a set $\mathcal{C}$

$$\delta_{\mathcal{C}}(z) = \begin{cases} 0 & z \in \mathcal{C} \\ \infty & \text{otherwise,} \end{cases} \tag{1.0.3}$$

which is the reason why both f and g are considered to be *extended-real-valued functions* (see [12, Section 3.1.2]). Formulations similar to the above have been studied extensively and we can look for their roots in the method of multipliers [13], [14], the Arrow-Hurwicz method [15], Douglas-Rachford splitting [16] and ADMM [17, 18]. Decomposition of the original problem into simpler ones is beneficial when distributed computation tools are available. This potential is already suggested in the classical references [19] and [20]. It was not until recently, though, that decomposition algorithms were indeed applied in modern engineering problems (signal and image processing, big data analysis, machine learning, [21] and [22]), in cases where off-the-shelf interior point solvers simply fail due to the large dimensions involved. The thesis [23] provides a comprehensive description of the connection of several splitting algorithms under a common framework. Finally, the book [24] provides a mathematically rigorous introduction to operator splitting methods in general Hilbert spaces.

The plethora of different approaches for solving problem (1.0.2) is partly a consequence of the problem-dependent behavior of first order methods. This behavior has both its pros and cons; on one hand, sensitivity to the problem's structure and data requires pre-processing and tuning of several parameters, a procedure that can be cumbersome. However, it is exactly this procedure that gives the flexibility to customize the solver to the problem at hand, and, in many cases, outperform general purpose solvers by several orders of magnitude. Consequently, there are numerous approaches, each of which can be less or more pertinent for the specific problem.

Mentioning some of the most important categorizations, we can solve either the *primal* problem, the *dual* problem, or a *primal-dual* formulation. Regarding primal approaches, the most popular one is the primal decomposition method [19, 25], where the original problem is decomposed into a master problem and two subproblems. The two subproblems have both local and shared (complicating) variables, while the master subproblem manipulates only the complicating variables. Primal decomposition works well when the complicating variables for the two subproblems are few.

Dualization plays a crucial role in more complicated problems. It can be performed by means of *Lagrangian relaxations* (dual decomposition [26–29]), *augmented Lagrangian relaxations* [30–32], *alternating minimization (Gauss-Seidel) augmented Lagrangian schemes* (ADMM), mixture of Lagrangian with augmented Lagrangian schemes (AMA [33]), *linearized augmented Lagrangians* or *approximate minimization* schemes [34, 35] and, finally, *mixtures of alternating minimization with partial linearization* (PDHG [36–39] and several similar primal-dual schemes [40–42]).

Although it is well-established that splitting methods are quite beneficial when applied to large-scale problems, their potential in solving small to medium scale embedded optimization problems has not been studied so extensively. It was not until very recently that the first works attempting to apply decomposition methods in control problems started making their appearance [5, 6, 43–45]. Our purpose is to study the behavior of such algorithms as solvers of control-related convex problems of that scale, *i.e.*, from tens to a few hundreds of variables. Our effort focuses on identifying computational characteristics of these problems and how they can be exploited when deployed on embedded platforms. Some of the questions that we attempt to answer are:

1. What is the computational and memory complexity of splitting methods?

2. Given that a control problem has to be solved repeatedly (*e.g.*, MPC), what kind of numerical algebra should be used for efficiently computation?

3. What is the best way to deploy a splitting method on various platform, for example, embedded possessors, Field Programmable Gate Arrays (FPGAs), and System On a Chip (SOC)?

4. How can one automatize the procedure for deploying splitting methods on different embedded platforms?

5. What type of resources and computational units are required for fulfilling real-time constraints?

In what follows we present three well-understood splitting algorithms, the *Alternating Direction Method of Multipliers (ADMM)*, the *Alternating Minimization Algorithm (AMA)* and a *Primal-Dual Algorithm (PDA)*, the most popular representative of several primal-dual schemes that have been recently developed. These three methods come from different sides of the spectrum described above, but also hold very strong similarities. Our choice is motivated from the fact that the methods are analyzed and extended from several communities, and hence their properties are well-understood.

1.1 Problem Formulation

We narrow the general formulation discussed in the previous section to our problems of interest, which can, without loss of generality, be written as

$$\begin{aligned} &\text{minimize} && (1/2)z^\top Qz + c^\top z + \sum_{i=1}^{M} g_i(L_i z + l_i) \\ &\text{subject to} && Az = b \ , \end{aligned} \tag{P}$$

with variable $z \in \mathbb{R}^n$ and data $Q \in \mathbb{S}^n_+$, $L_i \in \mathbb{R}^{p_i \times n}$, $l_i \in \mathbb{R}^{p_i}$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Here, $Q \in \mathbb{S}^n_+$ denotes Q is a positive-definite matrix. Defining $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$, we assume the following:

Assumption 1. *The functions $g_i : \mathbb{R}^{p_i} \to \overline{\mathbb{R}}$ are closed, proper, convex functions.*

Formulation (P) is quite general and can describe any convex optimization problem. The choice of the quadratic part $(1/2)z^\top Qz + c^\top z$ and the equality constraints $Az = b$ being represented in an explicit way is motivated by the standard form of control problems. The constraints are usually expressed through indicator functions g_i.

It is important to mention that the original formulation (1.0.2) involves two functions in the objective, while in (P) we consider *two groups of functions*. The first group

contains two functions expressed as

$$f(z) := h(z) + \delta_{\mathcal{D}}(z) \ , \tag{1.1.1}$$

where $h : \mathbb{R}^n \to \mathbb{R}$ is defined as $h(z) = (1/2)z^\top Qz + c^\top z$ and $f : \mathbb{R}^n \to \overline{\mathbb{R}}$. Note that we use the indicator function

$$\delta_{\mathcal{D}}(z) = \begin{cases} 0 & Az = b \\ \infty & \text{otherwise.} \end{cases}$$

to restrict h to the subspace spanned by the dynamics equation. The second group constitutes of M functions $g_i(y_i)$. By introducing slack variables $y_i = L_i z + l_i,\ i = 1, \ldots, M$, and subsequently concatenating the vectors and matrices associated with the affine terms in the $g_i(\cdot)$ functions as $L = (L_1, \ldots, L_M)$ and $l = (l_1, \ldots, l_M)$, we can recast (P) as

$$\begin{aligned} &\text{minimize} && f(z) + g(y) \\ &\text{subject to} && Lz - y = -l \ , \end{aligned} \tag{1.1.2}$$

where $g(y) = \sum_{i=1}^{M} g_i(y_i)$, $g : \mathbb{R}^{p_1} \times \cdots \times \mathbb{R}^{p_M} \to \overline{\mathbb{R}}$. Thus we end up with the original formulation (1.0.2). Note that it is possible to proceed with such a scheme because the variables are still updated in two sequential turns, since all the y_i updates occur *in parallel*.

The splitting schemes we will discuss provide both a primal and a dual solution to problem (P). However, their construction derives from several reformulations of (P). In the following sections, we derive the dual problem to (P), a saddle function reformulation, and then set the foundations for the derivation of the splitting methods by means of the proximal operator acting on these three different forms.

We note that the *Lagrangian* for (P) can be written as

$$\mathcal{L}(x, y; \lambda) = f(z) + g(y) + \langle \lambda, Lz + l - y \rangle \ , \tag{L}$$

where $\lambda = (\lambda_1, \ldots, \lambda_M)$, $\lambda_i \in \mathbb{R}^{p_i}$ are dual variables associated with the equality constraints introduced above. Furthermore, for a closed, proper, convex function f,

its *proximal operator* $\mathbf{prox}_f : \mathbb{R}^n \to \mathbb{R}^n$ is defined as

$$\mathbf{prox}_{\rho f}(x) := \underset{z}{\operatorname{argmin}} \left\{ f(z) + (1/2\rho)\|z - x\|^2 \right\} \quad . \tag{1.1.3}$$

The proximal operator firstly appear in the seminal work by Moreau [46, 47]. The operator is evaluated at a given point x and looks for a minimizer that makes a compromise between the minimizer of the function f and the point x.

We refer to proximal methods as being a family of abstract algorithmic schemes that find a minimizer of a (sum of) convex function(s) by means of the proximal operator. More details can be found in the recent survey [48]. The course notes [49] also provide a detailed reference to the topic. Next, we present three algorithms that will be deployed on embedded platforms.

1.2 Algorithms

1.2.1 Alternating Minimization Algorithm (AMA)

Algorithm 1.2.1 Alternating minimization algorithm (AMA)

Require: Initialize $\lambda^0 \in \mathbb{R}^p$ and $0 < \rho < \frac{2\sigma_f}{\|L\|^2}$

loop

1: $z^{k+1} = \underset{z}{\operatorname{argmin}} \quad f(z) + \sum_{i=1}^{M} \langle \lambda_i^k, L_i z \rangle$

2: $y_i^{k+1} = \mathbf{prox}_{\frac{1}{\rho} g_i} \left(L_i z^{k+1} + l_i + \lambda_i^k / \rho \right), \ i = 1, \ldots, M$

3: $\lambda_i^{k+1} = \lambda_i^k + \rho(L_i z^{k+1} + l_i - y_i^{k+1}), \ i = 1, \ldots, M$

end loop

We present AMA in Algorithm 1.2.1 [33]. From the perspective of the Lagrangian function, the first step of AMA is equivalent to the minimization of (L) with respect to the z variable. The second step involves the minimization of the *augmented Lagrangian (AL)*, that can be expressed for problem (P) as

$$\mathcal{L}_\rho(z, y; \lambda) = f(z) + g(y) + \langle \lambda, Lz + l - y \rangle + (\rho/2)\|Lz + l - y\|_2^2 \quad . \tag{AL}$$

Augmented Lagrangian functions have a long history in the optimization literature [32], [30]. Roughly speaking, minimization of the augmented Lagrangian function instead of the classical one results in faster convergence due to better

regularization of the problem through the quadratic term. The augmented Lagrangian minimization problem results in proximal steps that can be implemented in parallel. In the end, a dual multiplier update ensures convergence of the algorithm by enforcing consensus of the sequence of updates $\{Lz^k + l\}$ to $\{y^k\}$. AMA converges for $0 < \rho < \frac{2\sigma_f}{\|L\|^2}$, where σ_f is the strong convexity modulus of f.

1.2.2 Primal-Dual Algorithm (PDA)

In the context of large-scale convex optimization, the evaluation of the minimizer of f, as it appears, *e.g.*, in the first step of AMA, might be undesirably expensive. This is, *e.g.*, the case when f is a quadratic function with a dense Hessian, the minimization of which would require the solution of a linear system of equations. This motivated the development of numerous primal-dual algorithms that comprise a sequence of evaluations of proximal operators, where the gradients and linear operators involved in the steps are called explicitly without inversion.

Early works of this type involved two functions in the objective and could not exploit potential regularity properties such as smoothness of the functions [38, 50]. Later works expanded the previous ones to handle an extra (third) smooth function [39, 40], and even to deal with inexactness in the evaluation of the proximal steps [41]. These versions come under many different names, mostly referred to as the *Vũ-Condat algorithm*. In this work, we adopt the name *Primal-Dual Algorithm (PDA)* to describe a method that is generic enough to encapsulate most of the existing ones, though suitable for our setting (Algorithm 1.2.2). The proposed algorithm is in line with the recent scheme presented in [51]. Note that the function $g^*(x)$ denotes the conjugate of a convex function $g(x)$.

Algorithm 1.2.2 Primal-Dual Algorithm (PDA)

Require: Initialize $\lambda^0 \in \mathbb{R}^p$ and $z^0 \in \mathbb{R}^n$. Choose stepsizes $\tau, \rho > 0$ such that $\sqrt{\tau\rho}\sqrt{\sum_{i=1}^{M} \|L_i\|^2} < 1 - (L_h/2)\tau$

loop

1: $z^{k+1} = \mathbf{prox}_{\tau\delta_{\mathcal{D}}}\left(z^k - \tau(\nabla h(z^k) + L^\top \lambda^k)\right)$

2: $\lambda_i^{k+1} = \mathbf{prox}_{\rho_i g_i^*}\left(\lambda_i^k + \rho_i(L_i(2z^{k+1} - z^k) + l_i)\right),\ i = 1, \ldots, M$

end loop

An important common characteristic of these methods is that they make use of the information that f is the sum of a smooth and a non-smooth term, h and $\delta_{\mathcal{D}}$,

respectively, as presented in (1.1.1). Consequently, a quadratic model is constructed for f, *i.e.*, $\hat{f}(z) = \delta_{\mathcal{D}}(z) + h(z^k) + \langle \nabla h(z^k), z - z^k \rangle + (1/2\tau)\|z - z^k\|_2^2$, and is minimized instead of the original function in the first pass. In contrast to AMA, the cost and the dynamics are not lumped together in this case. The function h being smooth, information about the Lipschitz constant of its gradient is incorporated into the algorithm, typically resulting in faster convergence. The first step of the algorithm involves the projection of the evaluated gradient iteration onto the dynamics' subspace, while the second step is composed of dual variable updates by means of proximal operators that can be performed in parallel, as is the case for AMA.

1.2.3 Alternating Direction Method of Multiplier (ADMM)

ADMM, presented in Algorithm 1.2.3, is probably the most popular of the splitting methods, mostly due to its simplicity and the very few assumptions for convergence in comparison to other splitting schemes [17, 18, 52]. It was rediscovered 30 years later under a new name: *split Bregman method* [53]. In the case of ADMM, the functions f and g need only be convex.

Algorithm 1.2.3 Alternating Direction Method of Multipliers (ADMM)

Require: Initialize $y^0 \in \mathbb{R}^p$, $\lambda^0 \in \mathbb{R}^p$, and $\rho > 0$

loop

1: $z^{k+1} = \underset{z}{\text{argmin}} \quad f(z) + \sum_{i=1}^{M} \langle \lambda_i^k, L_i z \rangle + (\rho/2) \sum_{i=1}^{M} \|L_i z + l_i - y_i^k\|^2$

2: $y_i^{k+1} = \mathbf{prox}_{\frac{1}{\rho} g_i}\left(L_i z^{k+1} + l_i + \lambda_i^k/\rho\right), \ i = 1, \ldots, M$

3: $\lambda_i^{k+1} = \lambda_i^k + \rho(L_i z^{k+1} + l_i - y_i^{k+1}), \ i = 1, \ldots, M$

end loop

Compared to AMA, ADMM only differs in the minimization of the augmented Lagrangian function in the first step. This trait has the advantage that *no stepsize restrictions* occur for ADMM, in contrast to AMA and PDA. On the other hand, *the augmented Lagrangian minimization complicates the first step by the addition of a (possibly dense) quadratic form*, even in the case that the original structure of f allowed for a cheaper evaluation. This is not the case with AMA and PDA, where the first step remains simple. We compare and summarize discussed three algorithms in Table 1.1. It is also possible to enable different features for the listed algorithms: acceleration based on Nesterov's relaxation, preconditioning, termination criterion,

and adaptive restart. We refer to [1] for details.

	Strong convexity	Stepsize restrictions	Decouples variables of linear constraints
ADMM	no	no	no
AMA	yes	yes on f	yes
PDA	yes	no	yes

Table 1.1 – Comparison between three discussed algorithms.

1.3 Introduction to Model Predictive Control

Model Predictive Control (MPC) is a control technique that aims to minimize a predefined cost function satisfying the design constraints. The cost function often reflects the difference between predicted and desired trajectories of the system states and inputs, while constraints capture physical limitations of the system. The mathematical formulation for an MPC problem is:

$$\begin{aligned} \text{minimize } & \sum_{i=0}^{N-1} \left(x_i^\top Q x_i + u_i^\top R u_i\right) + x_N^\top Q_f x_N \\ \text{subject to } & x_{i+1} = Ax_i + Bu_i, \text{ for } i = 0, \dots, N-1 \ , \\ & x_i \in \mathcal{X} \ , \text{ for } i = 0, \dots, N \ , \\ & u_i \in \mathcal{U} \ , \text{ for } i = 0, \dots, N-1 \ , \\ & x_N \in \mathcal{X}_f \ , \\ & x_0 = \hat{x} \ , \end{aligned} \tag{1.3.1}$$

where the vector $x_i \in \mathbb{R}^{n_x}$ represents states, $u_i \in \mathbb{R}^{n_u}$ is the input vector, the matrix $Q \in R^{n_x \times n_x}$, $Q_f \in R^{n_x \times n_x}$, and $R \in R^{n_u \times n_u}$ are penalty matrices on states, the terminal state and inputs. with appropriate dimensions. The matrix A and B are state transition matrix and input matrix. The constraints on the states are ($\mathcal{X}$) and inputs ($\mathcal{U}$). The terminal constraint on states is denoted by $\mathcal{X}_f$. $\hat{x}$ is an initial state and N is a prediction horizon.

An MPC controller requires the solution of an optimization problem (1.3.1) at every sampling instant, which can be computationally intensive. This challenge can

be addressed by using explicit MPC [54] where the solution map is precomputed using multi-parametric programming. However, this approach quickly reaches its limitation for high dimensional problems because it requires the storage of off-line maps, leading to large memory demands. Therefore, for implementing MPC for reasonably large problems, one needs to rely on solving optimization problems online.

Broadly speaking, there are two approaches to solve the optimization problem online, using second-order or first-order methods. The limitation of the former case is the requirement of solving a linear system online, which has a computational complexity of $\mathcal{O}(n^3)$. This can be a significant limitation for embedded control where computational resources are limited. Since the computational compelxity of first-order methods is $\mathcal{O}(n^2)$, they have gained a lot of attention for embedded control applications. Amongst the first-order methods, splitting methods are becoming popular because it decomposes the original optimization problem into subproblems, which are computationally cheaper to solve than the original problem.

1.4 Discussions on Splitting Methods Based Model Predictive Control

From a first look to Algorithms 1.2.1, 1.2.2 and 1.2.3, several differences are already visible. In terms of applicability, ADMM requires minimal assumptions in order to work (convexity). The same holds for PDA (note that h can be set to zero in the case of absence of a smooth term in the objective). AMA requires strong convexity of f, which might seem, in principle, restrictive. However, in the framework of MPC, and assuming a quadratic cost in terms of both states and inputs, *i.e.*, $z = (x, u)$, with x being the concatenated (over a prediction horizon) state vector and u the corresponding vector of the inputs, we can distinguish two plausible formulations for which this holds:

1. The optimization problem is rewritten in terms of the control inputs only, *i.e.*, the states are eliminated. In this case, f becomes strongly convex and the dynamics equation $Az = b$ vanishes. Then $f(z) = z^\top Hz + r^\top z$, for some dense Hessian H, and the stepsize is upper bounded by $\lambda_{\min}(H)/\|L\|^2$.

2. We have that $z^\top Qz > 0$ for $z \neq 0$ and $Az = 0$, *i.e.*, positive definiteness of Q in the nullspace of A (see [6, Proposition 33]). This translates into positive definiteness of the objective (in both x and u), when restricted to

the nullspace of the dynamics. In this case we can write the KKT system

$$\begin{bmatrix} K_{11} & K_{21} \\ K_{21} & K_{22} \end{bmatrix} = \begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix}^{-1} . \tag{1.4.1}$$

The KKT matrix is nonsingular, and hence the first step of AMA can be solved by means of a linear system solve. The stepsize is upper bounded by $\lambda_{\min}(K_{11})/\|L\|^2$.

In conclusion, for a plethora of MPC problems involving regulation or tracking, all three methods are potentially applicable.

Restrictions on the stepsizes hold for both AMA and PDA. There is an obvious downside, but also an upside about this fact. The former regards the small stepsizes that are required for convergence, especially if the matrices Q and L in (P) are badly conditioned. The upside, though, is that the stepsize selection for AMA is made easier in comparison to ADMM, *i.e.*, the stepsize can be selected as the maximum allowable one. The selection is trickier in the case of PDA, since the condition $\sqrt{\tau\rho}\sqrt{\sum_{i=1}^{M}\|L_i\|^2} < 1 - (L_h/2)\tau$ involves two stepsizes affecting one another.

Finally, there are several computational differences among the three algorithms. As we mentioned above, augmented Lagrangian methods like ADMM tend to converge faster than Lagrangian methods due to the extra regularity coming from the quadratic term. This is more visible when the objective function does not involve a quadratic term by construction. On the other hand, the augmentation term contributes with a (possibly) dense quadratic form to an originally (possibly) non-dense objective. Consider, *e.g.*, a quadratic objective of the form $h(x,u) = (1/2)x^\top Qx + (1/2)u^\top Ru$, where Q and R are diagonal. The first step of AMA would require the solution of the KKT system (1.4.1), with K_{11} diagonal, while ADMM would densify the matrix. Solving via the Schur complement would require inversion of K_{11} (see [12, Appendix C, Example C.4]), which becomes costly in the latter case. Regarding PDA, the first step requires a projection onto the dynamics' subspace. Such a projection can be written in closed form as

$$P_{\mathcal{D}}(p) = p + A^\top(AA^\top)^{-1}(b - Ap) \ , \tag{1.4.2}$$

where $p = z^k - \tau(\nabla h(z^k) + L^\top\lambda^k)$, and thus requires the inversion of $AA^\top$, with $A \in \mathbb{R}^{m\times n}$ as defined in (P). This operation is inexpensive if $m << n$. The A matrix for a typical MPC problem is of size $Nn_x \times N(n_x + n_u)$. It thus makes sense

to prefer such an inversion, especially if $n_u > n_x$. The fact that the matrix under inversion is positive (semi)definite, allows for further offline manipulation, as we will see in Chapter 2.

This short discussion reveals that a good choice depends mostly on two factors: 1. Time investment for offline tuning and 2. the computational complexity of the first step, which in all cases involves a linear system solve. We will elaborate more on these aspects in the subsequent chapters.

This part of the thesis focuses on deploying splitting method based MPC on embedded hardware. Chapter 2 analyzes computation and memory complexity of splitting methods. We then compare various ways of deploying splitting methods on processors. In Chapter 3 we introduce SPLIT, a code-generating toolbox, using which we deploy splitting methods on embedded processors, FPGAs, and SOC platforms. Following that, Chapter 4 discusses a priori solving co-design problem for splitting methods based MPC.

2 Deployment of Splitting Methods on Processors

> The trouble with programmers is that you can never tell what a programmer is doing until it's too late.
>
> Seymour Cray

In this chapter we analyze the numerical operations involved in the splitting algorithms. The primary goal is to identify the key operations which can result in computational bottlenecks for the methods of interest. At a first glance, it is easy to conclude that the first step of all the methods amounts to the solution of a linear system, which is commonly the most computationally intensive part of the algorithms. We will discuss in detail the structure of these linear systems and present different tools and techniques to solve them. In addition, matrix-vector product is another common operation which will be analyzed in detail. We propose ways to perform both operations, making use of modern linear algebra packages and support our findings with experimental results.

2.1 Linear System Solver for Splitting Methods

The requirement for solving a linear system arises in the z-minimization step of AMA and ADMM (Algorithms 1.2.1 and 1.2.3). This is due to the fact that the step can be expressed as an equality-constrained QP, thus it gives rise to KKT conditions written in the form of a linear system [12, 55].

AMA: Recalling the Lagrangian definition (L), the first step of Algorithm 1.2.1 reads:

$$\begin{aligned} &\text{minimize} && \mathcal{L}(z;\lambda^k) \\ &\text{subject to} && Az = b \ , \end{aligned}$$

with variable z and λ^k entering as a parameter. The first order optimality conditions give rise to:

$$\begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} z \\ \nu \end{bmatrix} = \begin{bmatrix} -c - \sum_{i=1}^{M} L_i^\top \lambda_i^k \\ b \end{bmatrix} \ . \tag{2.1.1}$$

ADMM: With the only difference from AMA being the minimization of the augmented Lagrangian (AL), the first step of Algorithm 1.2.3 is:

$$\begin{aligned} &\text{minimize} && \mathcal{L}_\rho(z;y^k,\lambda^k) \\ &\text{subject to} && Az = b \ , \end{aligned}$$

expressed as the linear system

$$\begin{bmatrix} \left(Q + \rho \sum_{i=1}^{M} L_i^\top L_i\right) & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} z \\ \nu \end{bmatrix} = \begin{bmatrix} -c - \sum_{i=1}^{M} L_i^\top \lambda_i - \rho \sum_{i=1}^{M} L_i^\top (l_i - y_i^k) \\ b \end{bmatrix} \ . \tag{2.1.2}$$

Clearly, the linear systems (2.1.1) and (2.1.2) have very similar structure, commonly referred to as a *KKT system*. From now on we denote a general KKT system as

$$\begin{bmatrix} K_{11} & K_{21}^\top \\ K_{21} & 0 \end{bmatrix} \begin{bmatrix} z \\ \nu \end{bmatrix} = \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} \ , \tag{2.1.3}$$

where

$$K = \begin{bmatrix} K_{11} & K_{21}^\top \\ K_{21} & 0 \end{bmatrix} \tag{2.1.4}$$

is the *KKT matrix*. For a typical MPC problem with horizon N, n_x states and n_u inputs, (2.1.4) is symmetric and indefinite of dimension $((N+1)2n_x + Nn_u) \times ((N+1)2n_x + Nn_u)$[1]. For the sake of clarity, we consider from now on that (2.1.4) is of

[1] Matrix A is not necessarily the dynamics matrix of the system, but any general equality constraint.

dimension $n \times n$, $z \in \mathbb{R}^{n_1}$, $\nu \in \mathbb{R}^{n_2}$, where $n_1 + n_2 = n$.

We observe the following:

1. In the case of linear time-invariant (LTI) systems, the matrix (2.1.4) does not change over the iterations of AMA. The same holds for ADMM, as long as the penalty parameter ρ remains constant. Thus we can either precompute the inverse or factorize (2.1.3) using an $\mathrm{LDL}^\top$ factorization. Alternatively, block elimination can be used. The resulting matrices can be cached and reused over the iterations. The interested reader can refer to [12, Appendix C] for a quick guide.

2. When an adaptive penalty ρ is used, K_{11} varies over the iterations when ADMM is used. In practice, this means that K_{11} cannot be prefactored. When the dynamics equation has been suppressed ($K_{21} = 0$), one can use a *simultaneous diagonalization* technique [56] to aleviate the complexity. Note that, in AMA, a varying stepsize does not create any issue regarding the linear system solve.

3. The sparsity of (2.1.3) depends, apparently, on the Hessian Q, as well as the dynamics matrix A, in the original problem definition (P). One can expect that Q is always block diagonal and generally sparse, while A has a banded structure, with possible dense bands. In the case of ADMM, the sparsity of the K_{11} block might be lost by the addition of $\sum_{i=1}^{M} L_i^\top L_i$. This is not the case with AMA.

PDA: The need for solving a linear system comes from the first step of the algorithm, where a projection onto the dynamics' subspace has to be performed. We repeat the explicit form of the solution, formerly given in Section 1.4:

$$P_{\mathcal{D}}(p) = p + A^\top (AA^\top)^{-1}(b - Ap) \ . \tag{2.1.5}$$

The matrix $AA^\top \in \mathbb{S}_{++}$ can be treated offline by means of a Cholesky factorization. From Step 1 of Algorithm 1.2.2 one can see that the stepsize does not enter the inversion.

However, since in the majority of the cases considered in control problems this equality constraint will represent the dynamics, we refer to $A = K_{21}$ as 'dynamics matrix' hereafter.

2.2 Numerical Methods for Solving Linear Systems

It is evident that there are two important steps associated to the solution of the linear systems arising in the three algorithms, namely (2.1.3) for AMA, ADMM and (2.1.5) for PDA: Factor and Solve. In this section we discuss various methods to perform these two steps. We consider the problem:

$$Kz = k \ , \tag{2.2.1}$$

where $K \in \mathbb{S}^{n \times n}$. The matrix K refers either to the KKT matrix (2.1.4), or $K = AA^\top$, encompassing both (2.1.3) and (2.1.5). The interpretation will be clear from the context.

Numerical methods to solve (2.2.1) can be categorized in two families: (i) direct solvers and (ii) iterative solvers. We focus on the first category, given the size of our problems of interest.

In what follows we discuss different approaches for solving linear systems arising from ADMM, AMA, PDA and their variants, using different linear algebra libraries written for the programming language C. The purpose is to perform a comprehensive comparison and identify which combination of approach and software package is more suitable for a given problem. The approaches taken are: matrix inversion and matrix-vector multiplication, factorization and forward-backward substitution, block elimination, nullspace method and Riccati recursion. We analyze the computational complexity by means of Floating Point Operations (flops), one flop being equal to one addition, subtraction, multiplication, or division of two floating-point numbers. Memory complexity is measured in terms of the amount of memory used to store floating point numbers.

1. **Precompute inverse:** The computation of the inverse of K is performed offline, with the drawback that, although (2.1.3) might be sparse, once inverting, the sparsity is lost. Thus, the computational and memory complexities of $K^{-1}k$ are $\mathcal{O}(2n^2)$ and $\mathcal{O}(n^2)$, respectively. The computational and memory complexities for computing the inverse of the matrix (performed offline) are $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively.

2. **Factor and solve:** When the matrix K does not change between consecutive solves (as is, *e.g.*, the case where LTI systems are considered), it can be pre-

factored. In this way, only the factors are used in the solve step, rendering the operation much cheaper than inverting the matrix. The following factorizations are possible:

- LU, $\mathrm{LDL}^\top$, and Cholesky factorization: LU factorizes K as $K = LU$, where L is lower triangular and U an upper triangular matrix. The cost for an unstructured matrix is $(2/3)n^3$ flops. It is advantageous to use on banded matrices since it preserves the bandwidth [57, 58]. $\mathrm{LDL}^\top$ is suitable for symmetric invertible matrices. The factorization cost reduces to $(1/3)n^3$ flops. An advantage against LU is that only the storage of a lower triangular matrix L and a diagonal D are required. On the downside, it does not preserve the (possibly) banded structure of K, typically leading to additional fill-in. Finally, Cholesky is a special case of $\mathrm{LDL}^\top$, applicable to positive definite matrices. The matrix is factored as $K = LL^\top$, at the cost of $(1/3)n^3$ flops. It can be applied to factor K when PDA is used.
- Since the factors resulting from the previous step are lower (and) upper triangular matrices, the solve step can be performed using forward and backward substitutions. The cost of a forward-backward operation for an unstructured matrix is $2n^2$.

It is important to exploit sparsity when the factor and solve steps are performed. Among available linear algebra packages for C, CLAPACK [59] performs these operations treating the matrices as dense. Hence, the computational and memory complexities for performing the factorization (offline) are $\mathcal{O}(n^3/3)$ and $\mathcal{O}(n^2)$, respectively, while for the forward-backward substitution (online step) $\mathcal{O}(2n^2)$ and $\mathcal{O}(n^2)$. On the contrary, SuiteSparse [60] uses a Compressed Sparse Row (CSR) format to store the matrix elements, exploiting the advantage of having few nonzero elements. The CSR format represents the matrix by three vectors. The first vector contains integers representing the number of nonzero elements in each row. The second (integer) vector stores the indices at which nonzero elements are present in each row. The last vector stores all the nonzero elements of the matrix. If the matrix is sparse (which is indeed the case with (2.1.3)), then the computational complexity to perform the factorization is much less than $\mathcal{O}(n^3/3)$. The memory complexity is also reduced to $\mathcal{O}(\mathrm{nnz} + n)$, where nnz is the number of nonzeros in L. The forward-backward substitution step ends up having computational and memory complexities of (roughly) $\mathcal{O}(2\mathrm{nnz} + n)$ and $\mathcal{O}(\mathrm{nnz} + n)$, respectively.

3. **Block elimination:** Block elimination is suitable for systems that have the KKT form (2.1.3). The system is solved in two steps, namely $z = K_{11}^{-1}(k_1 - K_{21}\nu)$ and $S\nu = k_2 - K_{21}K_{11}^{-1}k_1$, where S is the *Schur complement* of K_{11} in K, given by $S = -K_{21}K_{11}^{-1}K_{21}^{\top}$, and S can be factored using a Cholesky factorization. However, since (2.1.3) is structured, exploiting this fact can further reduce the complexity. It is interesting to analyze this in more detail, following the same procedure as in [12, Section C.4].

   Since we solve repetitively, there is once the factorization cost of K_{11}, the formulation of $K_{11}^{-1}K_{21}^{\top}$, as well as the factorization of S, which costs $(2/3)n_2^3$ flops. Subsequently, two solves are performed at each iteration with respect to ν and z. Forward-backward substitution for z and ν cost $\mathcal{O}(n_1^2)$ and $\mathcal{O}(n_2^2)$ flops, respectively, hence resulting in quadratic complexity in the horizon length and the number of states and inputs.

   If K_{11} is diagonal (see, *e.g.*, AMA variants with diagonal Hessians in the cost), the factorization cost for K_{11} is zero. Consequently, the total solve cost is dominated by the solution of ν. If K_{11} is block diagonal, with blocks of size n_x and n_u, the factorization can be performed for each block separately, resulting in $(2/3)\sum_{i=1}^{N}(n_x^3 + n_u^3)$ flops. The same holds for the z-solve step which can be carried out in $2\sum_{i=1}^{N}(n_x^2 + n_u^2)$ flops.

4. **Nullspace method:** One significant drawback of block elimination is that it assumes that the K_{11} matrix is invertible which need not always be the case. The nullspace method can be used even in the case when K_{11} is not invertible. We define $H = (1/2)\left(K_{11} + K_{11}^{\top}\right)$. The requirement for using the method is that $\ker(H) \cap \ker(K_{12}) = \{\emptyset\}$ (see [61] for more details). The offline computation steps are:

   (a) Find a particular solution $\hat{z}$ such that $K_{12}\hat{z} = k_2$

   (b) Compute the matrix Z such that $K_{12}Z = 0$, *i.e.*, the range of Z is the null space of K_{12}. This can be computed using, *e.g.*, rank-revealing QR or rank-revealing LU decomposition.

   (c) Factorize $K_{21}K_{21}^{\top}$ and $Z^{\top}K_{11}Z$ for the linear system solves that will follow.

   Once the vector $\hat{z}$ and matrix Z are computed, the rest of the calculations are performed online as follows:

   (a) Solve $Z^{\top}K_{11}Zy = Z^{\top}(k_1 - K_{11}\hat{z})$. The vector $K_{11}\hat{z}$ can be precomputed offline. If the rank of K_{21} is n_2, then the dimension of y is $n_1 - n_2$.

Assuming that the factorization of $Z^\top K_{11} Z$ has been performed offline, the online computations require only forward backward solves, leading to a computational complexity of $\mathcal{O}((n_1 - n_2)^2)$, for the fully dense factorization.

(b) Once y is computed, z of (2.1.3) is calculate using $z = Zy + \hat{z}$. This step involves a matrix-vector multiplication and a vector addition, resulting in a computational complexity of $\mathcal{O}(n_1 n_2)$.

(c) Finally, ν is computed by solving the equation $K_{21}^\top \nu = k_1 - K_{11} z$. Notice that K_{21} is a rectangular matrix in most of the cases, thus one can solve for ν if $K_{21} K_{21}^\top$ is full rank by using the left pseudoinverse $\nu = (K_{21} K_{21}^\top)^{-1} K_{21}(k_1 - K_{11} z)$. Again, the factorization of $K_{21} K_{21}^\top$ can be computed offline since K_{21} is fixed. Subsequently, the computational complexity is $\mathcal{O}\left(n_2^2\right)$ assuming the factorization is fully dense.

5. **Riccati recursion:** Suppose that the KKT system (2.1.3) we have examined results from the minimization of a multistage cost coupled with the system's dynamics, expressed by the matrix A. Under the assumption that f is convex quadratic in states and inputs, this fact allows for an alternative way to perform the z-minimization step in both ADMM, AMA (and the variants), namely to perform a *Riccati recursion*. This approach has been commonly considered in the control literature, mostly due to its computational advantages that arise in several cases (see [62–66] for details). This method has approximate computational complexity $N(6n_x^2 + 8n_x n_u + 2n_u^2)$ and memory complexity $N(2n_x^2 + 3n_x n_u + n_u^2/2)$. In the case of LTI systems the memory complexity can be further reduced.

6. **Custom solver:** Finally, we created a custom solver for the sake of comparison with the aforementioned methods. The approach is based on exhaustive code generation (see, *e.g.*, [10]). The idea is to compute the $\mathrm{LDL}^\top$ factorization in Matlab, and then explicitly write the entries of the matrix in a generated C file. Subsequently, the data is loaded in C and used with a custom forward-backward solver. A reverse-CutHill McKee reordering is utilized to reduce the fill in L [60, 67]. The matrices $L, D, L^\top$ are stored explicitly, using a format similar to CSR, as opposed to SuiteSparse which stores only L and D. Thus, for the forward-backward step the computational and memory complexities are $\mathcal{O}(2\mathrm{nnz} + n)$ and $\mathcal{O}(2\mathrm{nnz} + n)$.

Table 2.1 summarizes the above discussion regarding the complexities. Looking at the table, we can roughly state that, if one disregards SuiteSparse and the

exhaustive code generation, the Riccati recursion beats by far the remaining three approaches for moderate to long horizon lengths, since it is the only one scaling linearly with the horizon. The block elimination and nullspace methods are cheaper compared to matrix pre-inversion and CLAPACK because they exploits the block diagonal structure of the KKT system. Finally, the sparsity-exploiting methods (SuiteSparse and custom solve) scale linearly with the horizon, but the computational complexity added by the remaining nonzero elements after the factorization has been performed is, generally, unknown. We can roughly say that the resulting lower triangular L matrix will have an almost banded structure, but the width of the band is not known in advance.

Method	Computational	Memory
Inverse	$\mathcal{O}\left(2n^2\right)$	$\mathcal{O}\left(n^2\right)$
CLAPACK	$\mathcal{O}\left(2n^2\right)$	$\mathcal{O}\left(n^2\right)$
SuiteSparse	$\mathcal{O}\left(2\text{nnz} + n\right)$	$\mathcal{O}\left(\text{nnz} + n\right)$
Block elimination	$\mathcal{O}\left(N^2(n_x + n_u)^2\right)$	$\mathcal{O}\left(N^2(n_x + n_u)^2\right)$
Nullspace method	$\mathcal{O}\left(N^2 n_x(n_x + n_u)\right)$	$\mathcal{O}\left(N^2 n_x(n_x + n_u)\right)$
Riccati	$\mathcal{O}\left(N(6n_x^2 + 8n_x n_u + 2n_u^2)\right)$	$\mathcal{O}\left(N(2n_x^2 + 3n_x n_u + n_u^2/2)\right)$
Custom	$\mathcal{O}\left(2\text{nnz} + n\right)$	$\mathcal{O}\left(2\text{nnz} + n\right)$

Table 2.1 – Computational and memory complexities for online linear system solve. The block elimination and nullspace methods perform on the KKT system (2.1.3), hence the complexity is expressed in terms of n_x, n_u and N. The same holds for the Riccati recursion, which operates in a special way. For the rest of the methods, n can be either $N(2n_x + n_u)$ for (2.1.3), or Nn_x for (2.1.5).

2.2.1 Numerical Results

In this section we perform numerical experiments regarding the linear system solve operation discussed before. The comparison involves only factor and solve methods, namely matrix pre-inversion and matrix-vector multiplication, forward-backward substitution with CLAPACK and SuiteSparse, as well as the custom solver. All the experiments are performed on Mac OS with Intel core i7 2.8 GHz with 16GB RAM. Since the comparisons do not involve any iterative methods, we consider a

randomly generated multi-stage optimal control problem of the form:

$$\begin{aligned} \text{minimize} \quad & (1/2)\sum_{i=1}^{N}\left(x_i^\top Q x_i + u_i^\top R u_i\right) + (1/2)x_{N+1}^\top Q x_{N+1}, \\ \text{subject to} \quad & x_{i+1} = Ax_i + Bu_i \ , i = 0, \ldots, N \\ & u_{\min} \leq u_i \leq u_{\max} \ , i = 0, \ldots, N \\ & \|F_x x_i\|_2 \leq f_x \ , i = 0, \ldots, N+1 \\ & \|F_u u_i\|_2 \leq f_u \ , i = 0, \ldots, N \ . \end{aligned} \tag{2.2.2}$$

The matrices Q and R are set to be the identity and the system is randomly generated. The number of states equals the number of inputs. We vary the size of the inputs (states) and also the horizon length as per Table 2.2.

n_x	4	10	20	30	30	40
n_u	2	5	10	15	15	20
N	4	10	10	15	20	20

Table 2.2 – Problem (2.2.2) instances of varying size.

The problem is created and parsed in Matlab, while the solve step is performed in C. The results are illustrated in Figures 2.1, 2.2 and 2.3 for ADMM, AMA and PDA, respectively. The time depicted is per algorithmic iteration, using four different ways to perform the linear system solve, for increasing number of variables. The upper yellow part of each bar plot is the time needed for the remaining operations (proximal step, dual update etc.). The scale is logarithmic.

It is evident that, irrespective to the problem size, SuiteSparse and the custom solver outperform CLAPACK and the pre-inversion approach. The main reason for this is that the KKT system and its factors are sparse. Especially for the custom solver, the explicit storage of L^T (in contrast to SuiteSparse), allows for sequential access of the memory or spatial locality, which is important for problems for which the size of the data does not fit into the cache. Regarding the pre-inversion, once inverted, the KKT matrix becomes fully populated, with obvious implications.
For the case of PDA, the trends are similar to ADMM and AMA. However, the size of the linear system to solve is significantly smaller than in the other two cases. We also observe that the remaining operations have non-negligible contribution in the

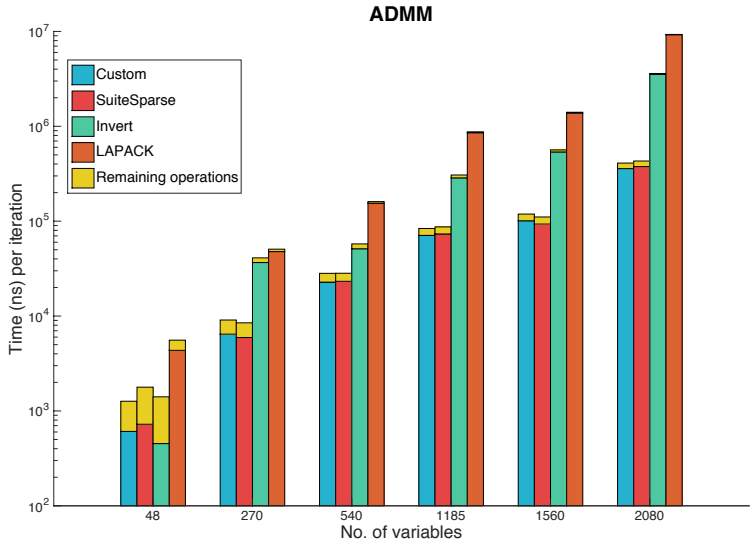


Figure 2.1 – Computational load distribution per iteration of ADMM. The remaining operations in yellow color refer to the remaining algebraic operations of the algorithm.

total iteration time.

2.3 Numerical Methods for Computing Matrix-Vector Multiplication

Matrix-vector multiplication (matvec operation hereafter) is the most common and the second most expensive operation from the computational viewpoint. In this section we compare popular linear algebra packages that compute matvecs, analyse their computational complexity, and illustrate timing results for varying sizes and different sparsity patterns.

More specifically, we consider:

- **forloops:** In this naive approach, the matrix is treated as dense and unstructured. Two nested for-loops are used to compute the matvec. The computational and memory complexities are $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2)$.
- **BLAS:** BLAS stands for Basic Linear Algebra Subprograms. It performs basic linear algebra operations *e.g.*, vector manipulation (addition, multiplication), matrix-vector manipulation and matrix-matrix manipulation. BLAS considers dense matvec operations, with corresponding computational and memory

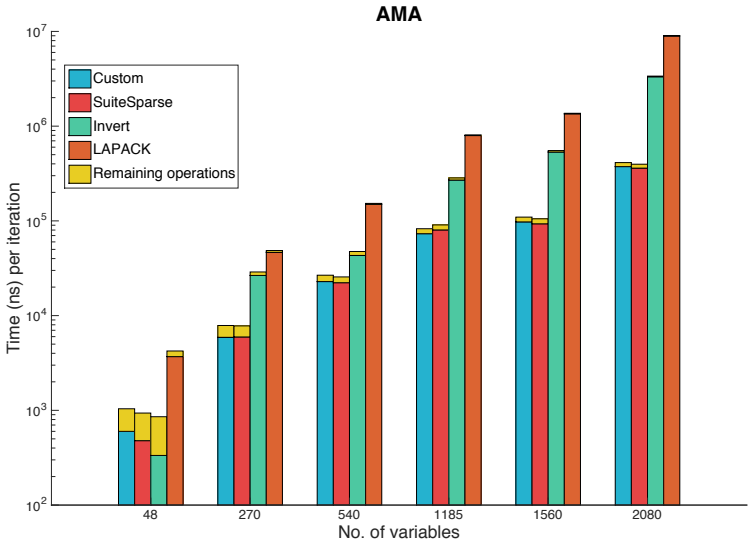


Figure 2.2 – Computational load distribution per iteration of AMA.

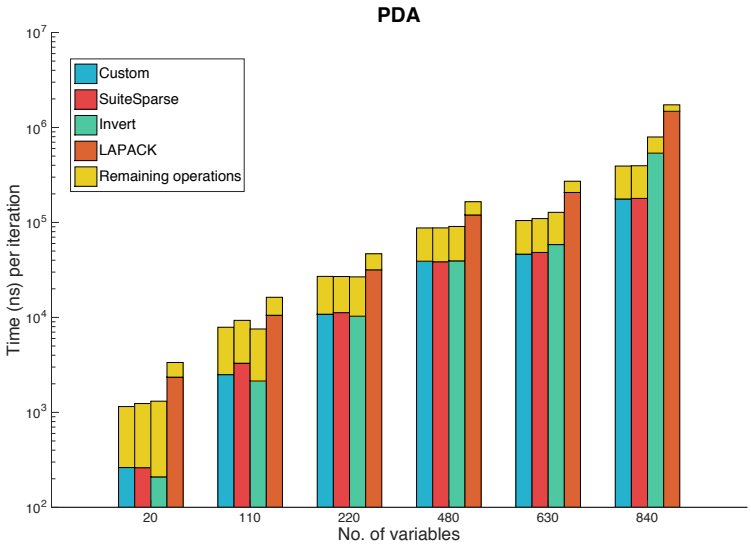


Figure 2.3 – Computational load distribution per iteration of PDA.

complexity of $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2)$.

- **SuiteSparse :** SuiteSparse representes the matrices in CSR format, as we already mentioned in Section 2.2. The matvec operation has computational and memory complexities of $\mathcal{O}(\text{nnz})$ and $\mathcal{O}(\text{nnz})$.
- **Custom methods :** We perform exhaustive code generation for this method [10]. The idea is to explicitly write the entries of the matrix in a generated C file.

2.3.1 Numerical Results

Problem Formulation: Looking at all three algorithms and their variants, one observes two matvec operations that are present in all cases: Lz and $L^\top\lambda$. We thus restrict our analysis to these two operations, since they are the most common and they both involve the same matrix, namely L. Since $L_i,\ i = 1, \ldots, M$ are the linear maps that appear in the g_i functions in (P), they mostly represent constraints on states and inputs. These constraints also tend to be stage-wise, or, less frequently, they couple more than one time stage, however almost always resulting in a structured and sparse matrix L. If a condensed formulation of the problem is considered (*i.e.*, the constraints are expressed in terms of the control inputs only), then possibly existing state constraints will impose a full, lower triangular structure on L. To summarize, among the common cases, L can be a full lower triangular matrix, in the worst case. Following this reasoning, to compare the aforementioned approaches, we compute matvecs on a lower triangular banded matrix with varying size and varying sparsity. To vary sparsity, we start with a diagonal matrix and gradually fill in the matrix by adding bands until it becomes a completely filled lower triangular matrix.

The results are depicted in Figure 2.4. The plotted times for a matvec operation are in ns, using the (optimized) gcc compiler. Various matrix sizes are used, with different sparsity percentages. The plotted curves regard diagonal matrix, 50% fill-in as well as fully populated lower triangular matrix. When sparse matrices are considered, the custom method beats all the others since all the entries of the matrix are explicitly written. As expected, SuiteSparse follows closely, exploiting the sparsity. As the matrix becomes gradually filled, SuiteSparse and the custom method become costlier, while the solve time for BLAS does not change considerably, due to the fact that irrespective of sparsity BLAS always treats the matrix as fully

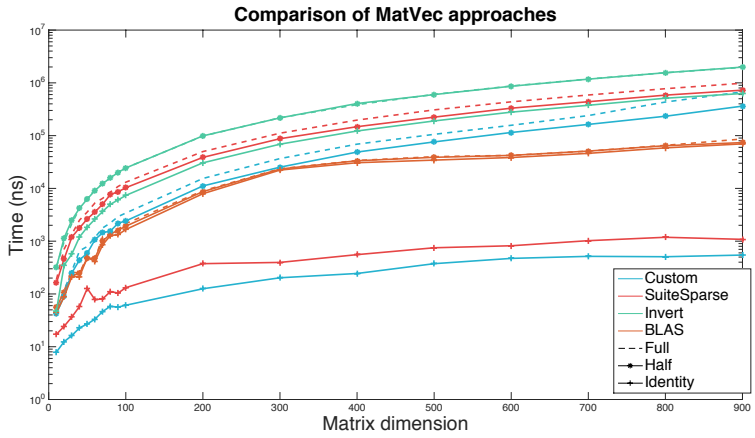


Figure 2.4 – Custom code generation performs well, but at the cost of very long pre-processing periods. Once the matrices become half-full, BLAS (in orange) clearly outperforms the alternatives, keeping an almost constant cost per solve, regardless of the sparsity.

dense. Furthermore, it is worth noting that the code generation for the custom method potentially takes too much time to execute, rendering it impractical for any purpose (see Figure 2.5 and 2.6).

2.4 Conclusions

In this chapter, we analyzed numerical operations involved in implementing splitting methods. We discussed computational bottlenecks for splitting methods. We compared and recommended different algebra packages based on sparsity and problem structure for efficient implementations.

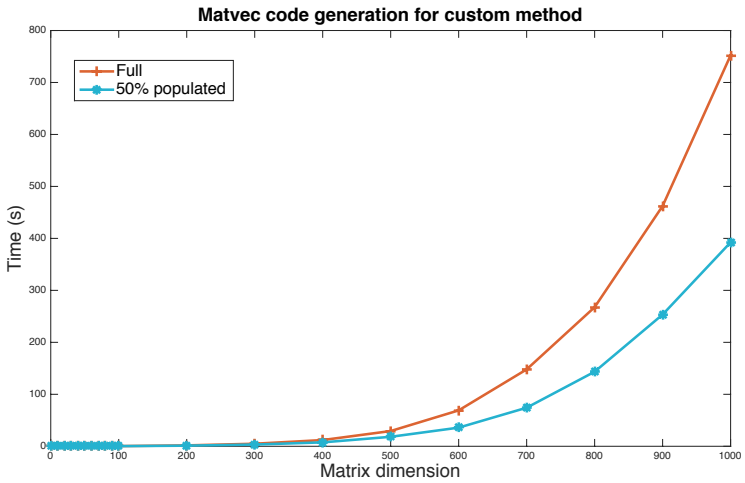


Figure 2.5 – Code generation for matvec operations of varying size (horizontal axis) versus time (vertical axis). The matrices involved are fully populated (red) and 50% populated lower triangular. The generation time increases polynomially.

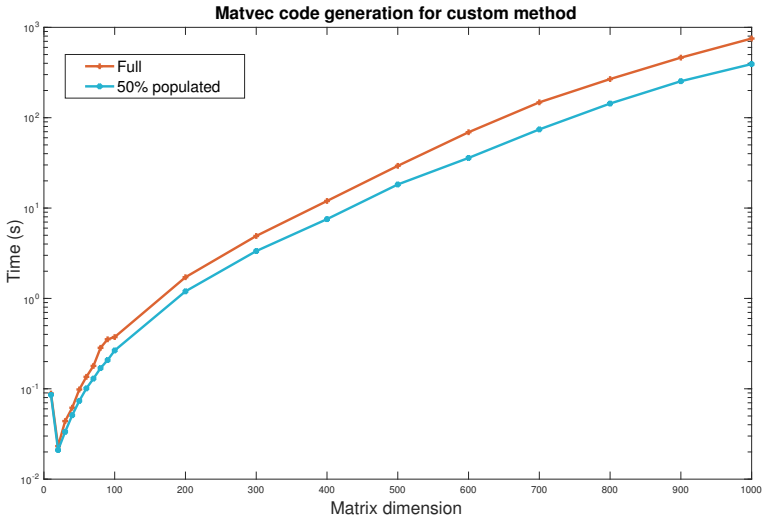


Figure 2.6 – Code generation for matvec operations of varying size (horizontal axis) versus time (log scale vertical axis). The matrices involved are fully populated (red) and 50% populated lower triangular. The generation time increases polynomially.

3 Deployment of Splitting Methods on Programmable Hardware

> The cheapest, fastest, and most reliable components are those that aren't there.
>
> Gordon Bel

Until now, we discussed deployment of splitting method based MPC on processors. In this chapter we deploy splitting methods on FPGAs and System On Chips (SOCs). We also propose a high-level code-generation toolbox—SPLIT—that enables rapid development and deployment of splitting methods on various hardware platforms.

FPGAs are known for their deterministic guarantee on execution time and are therefore a suitable candidate for achieving real-time guarantees. Furthermore, they offer a better computation/power consumption ratio compared to microprocessors or GPUs (Graphics Processing Units). FPGAs have various resources such as Digital Signal Processors (DSPs) , Flip-Flops (FFs), Block Random Access Memories (BRAMs), Look-Up Tables (LUTs), etc. The user interconnects these resources to make Central Processing Unit (CPU) like units. The advantage is that circuit can be tailored to the algorithm. Besides, this enables trading-off resources with exectution time, i.e., less execution time can be achieved by utilizing more resources.

Historically, FPGAs are programmed using Register Transfer Level (RTL) languages such as VHSIC Hardware Description Language (VHDL) or Verilog, but these languages have a steep and cumbersome learning process which acts as a major obstacle between conceptualization and deployment. Furthermore, it is tedious and complex to maintain large projects. In conclusion, the challenge with RTL based languages is that they are not appropriate for rapid prototyping and deployment purposes.

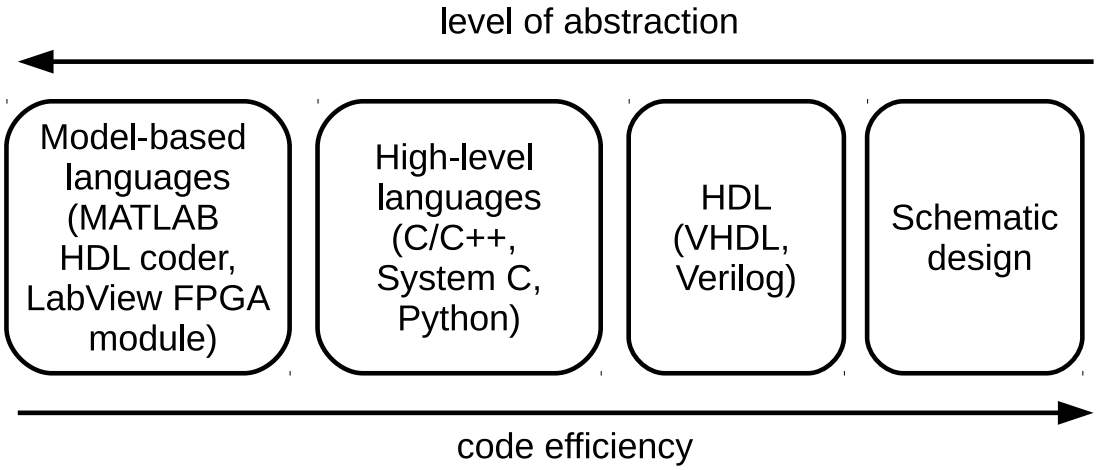


Figure 3.1 – Efficiency versus abstraction for FPGA programming languages [68].

To overcome the limitations of RTL languages, FPGA vendors and the software community have explored the possibility of using high-level languages, e.g., C, C++, OpenCL, MATLAB and LabVIEW because of their popularity and user-friendliness (Figure 3.1). We refer to [69] for a listing and comparison among FPGA circuit design tools. Usually, high-level languages increase the productivity of a programmer and enables a structured approach for the handling of larger projects. Various comparisons in details between high-level languages versus RTL can be found in a recently published survey [70] and can be concluded as: while RTL languages beat high-level languages–with a small margin–in terms of efficient usage of resources, High Level Synthesis (HLS) requires only a third of the development time compared to RTL and increases the productivity four times. Therefore, without a surprise, HLS tools are gaining attention and popularity among hardware developers and there are massive efforts to improve their efficiency.

The challenge with HLS (C/C++) code is that if the program fails incorporating various FPGA features like pipelining, parallelism, memory accesses, etc, then FPGAs lose their compute performance significantly [2]. That is why HLS toolboxes exploit these features [71, 72]; however, they still rely on users for having insight about FPGA's architecture and programming. This is a limitation because not all potential FPGA users have the necessary knowledge–especially in the control community. This demands developing a toolbox for users, without having background knowledge, interested in quickly prototyping and deploying their algorithms while exploiting FPGA features.

Despite several efforts to deploy control algorithms on FPGAs (see survey [73]

and [74–78]), little work is done developing high-level code-generation toolboxes for control algorithms [2, 74]. Next, we will address this challenge with two main contributions:

1. SPLIT, a high-level code generation toolbox, that facilitates rapid development and deployment on embedded prosessors, FPGAs, and heterogeneous platforms.

2. Analysis and novel methods for deploying splitting methods efficiently on reconfigurable platforms.

3.1 Code Generation for Software using SPLIT

The purpose of SPLIT is to provide an easy-to-use open source and free toolbox which saves time and effort for users deploying splitting methods on embedded systems. The toolbox is written in a high-level language (MATLAB) and code generation is tailored to a specific splitting algorithm. In this way the toolbox exploits the problem structure for generating efficient C code. SPLIT can be used to target an embedded general-purpose processor or it can also be used to deploy on pure FPGAs or heterogeneous platforms. In this section, we will focus particularly on deployment on general-purpose embedded processors and Section 3.3 is dedicated for detailed discussions on code generation for FPGAs.

The toolbox supports three splitting methods as discussed in the previous chapters.

- Alternating Direction Method of Multipliers;
- Alternating Minimization Algorithm;
- Primal Dual Algorithm.

Since SPLIT has its own libraries it can be used as a library-free toolbox. The toolbox also provides users an option to select different linear algebra libraries discussed previously, like "SuiteSparse" [79], "BLAS" [80] and "LAPACK" [81]. Thus, depending on an application, a user can use SPLIT as library-free or with a suitable library. Another strength of the toolbox is that it analyses the sparsity pattern of the problem and recommends to the user a particular library or method.

As discussed in Section 2.1, the computationally expensive operations for splitting algorithms are solving linear systems and matrix-vector multiplication. We summarize various ways to solve linear systems and computing matrix vector multiplications using SPLIT in Table 3.1 and Table 3.2 respectively. Note that "Custom_HW_sparse" and "Custom_HW_dense" are tailored for implementation on FPGAs and they are explained in Section 3.3.

Library	Sparsity	Method	Suitable for
Custom	Sparse	LDL	Small sparse matrices
SuiteSparse	Sparse	LDL	Large sparse matrices
CLAPACK	Dense	LDL	Dense matrices

Table 3.1 – Different linear system solves supported by SPLIT

Library	Suitable for
BLAS	Dense matrices
SuiteSparse	Sparse matrices
Custom_software	Small sparse matrices
Custom_HW_sparse	Sparse matrices on FPGAs
Custom_HW_dense	Dense matrices on FPGAs

Table 3.2 – Matrix vector multiplications supported by SPLIT

Before we discuss how SPLIT deploys splitting algorithms on FPGAs, we first introduce reconfigurable and heterogeneous computing in the following section.

3.2 Reconfigurable and Heterogeneous Computing Platforms

An FPGA is an array of relatively simple circuits, namely flip-flops (FFs) and lookup tables (LUTs), that are connected to switch matrices (Figure 3.3 and 3.4). Configuring switch matrices allows creating connections between and within logic blocks so that the desired circuit is obtained. Modern FPGAs provide special purpose units (e.g. dedicated memory blocks) that are more efficient from a silicon usage and signal routing point of view, compared to general purpose FFs and LUTs [82].

The key outstanding feature of reconfigurable platforms is customizability. Unlike fixed architecture CPUs that have fixed logic for performing a predefined set of operations, FPGAs allow synthesizing computational units with respect to a given algorithm. Moreover, computational units can be connected directly to each other to create *data pipelines*. The data storage subsystem is also flexible: memory blocks can be partitioned and placed near the corresponding processing units so that each block is processed independently.

However, the above-mentioned advantages often come at the price of certain limitations. Firstly, an FPGA clock rate is often slower compared to CPU-like architectures. Overcoming this limitation requires introducing a sufficient degree of parallelism. Secondly, some algorithms cannot be efficiently mapped on hardware due to data dependencies and/or resource consuming operations. In such cases it might be useful to employ heterogeneous platforms, known as systems-on-a-chip (SoCs), that incorporate both sequential CPUs and FPGAs (Figure 3.5). With a heterogeneous computing approach, computationally heavy parts of the workload can be accelerated on FPGA, while keeping the rest of the algorithm on CPU to save computational resources. The SPLIT toolbox considers three options of splitting the workload:

- Pure software implementation.
- Heterogeneous implementation: accelerating solving the linear system on the FPGA and computing the rest on the CPU.
- Pure FPGA implementation.

A heterogeneous computing platform can be programmed in several ways. Conventional approaches propose programming each subsystem using a dedicated, often low-level, language and handling data transfer between subsystems manually. Although low-level programming often leads to efficient realizations both from a time and resource usage point of view, these benefits come at the price of high implementation effort and hence long time-to-market. Model-based languages (e.g. the Matlab HDL coder) on the other hand significantly reduce implementation effort, providing flexibility of shifting the workload between different computational subsystems and allowing quick closed-loop performance verification. Unfortunately, the resulting circuit often cannot be considered as efficient. Using C-based integrated development environments (e.g. Xilinx SDSoC) is a compromise between design effort and implementation efficiency: although the whole computing system

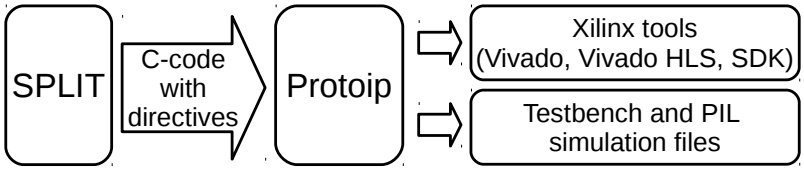


Figure 3.2 – The proposed toolchain flow.

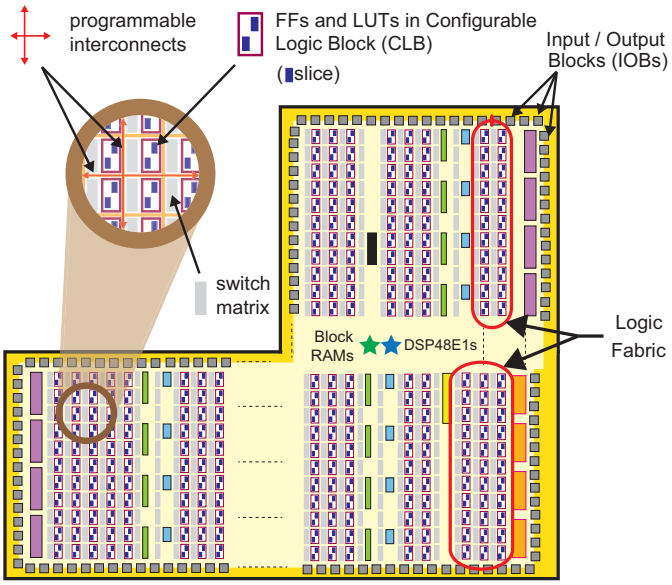


Figure 3.3 – FPGA architecture [84].

is programmed using a unified language, supplying the code with additional compiler directives allows specifying low level details to satisfy given design constraints. The SPLIT toolbox relies on a C-based approach and uses the Protoip tool [83] to manage underlying projects (Figure 3.2). Protoip allows rapid prototyping of optimization algorithms on FPGAs by providing processor-in-the-loop (PIL) test facilities while abstracting low-level implementation details. PIL simulation allows verifying controller performance by running optimization algorithm on FPGA and simulating the plant on a desktop machine.

3.3 Code Generation for Hardware using SPLIT

We proceed to explain how SPLIT allows users to directly deploy an MPC controller on FPGAs or on heterogeneous platforms. The flow for code generation for FPGAs is illustrated in Figure 3.2. At the first stage of the design flow a user defines an MPC formulation, as illustrated in Figure 3.6.

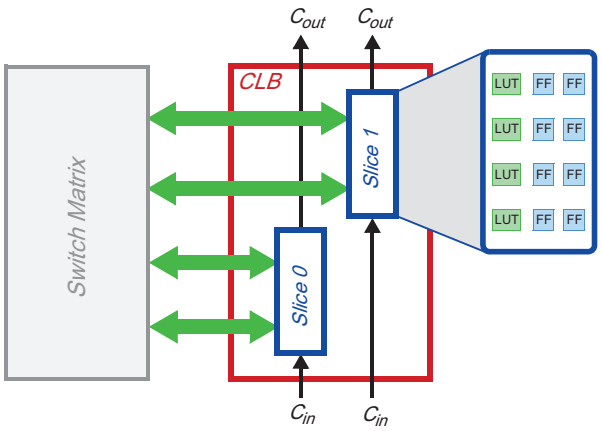


Figure 3.4 – Configurable logic block [84].

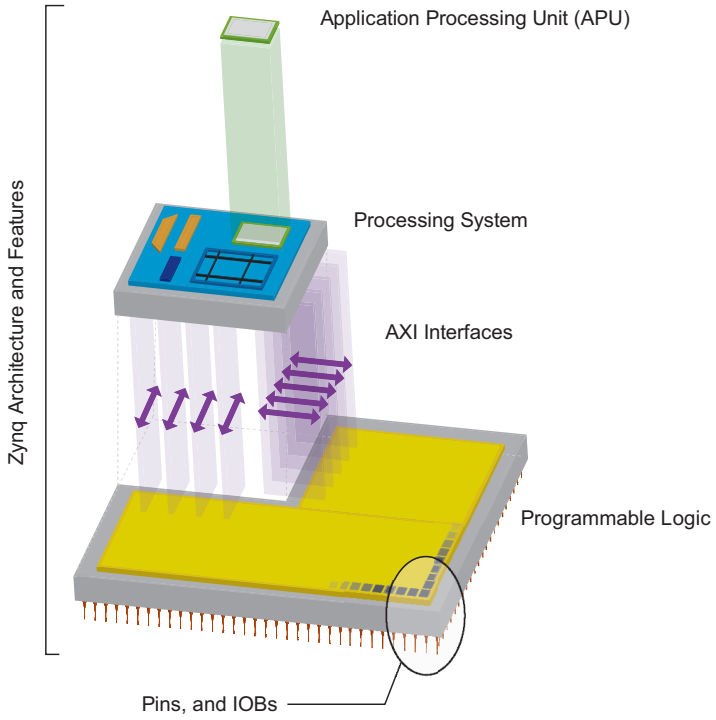


Figure 3.5 – Programmable Logic (FPGA) and Processing system (Embedded possessors) On SOC [84].

Following this SPLIT generates hardware-oriented synthesizable code with synthesis directives (e.g. pipelining, parallelization) that allow efficient mapping on hardware. We classify all underlying computations for splitting algorithms into: scalar, vector-vector and matrix-vector operations.

```
splitProb.clearProblem;
x = splitvar(nx,N+1);
u = splitvar(nu,N);
x_par = parameter(nx,1);
obj = 0;
%% Define cost
for j = 1:N
    obj = obj + 0.5*x(:,j)'*Q*x(:,j)+0.5*u(:,j)'*R*u(:,j)
end
obj = obj + 0.5*x(:,N+1)'*Q*x(:,N+1);
x(:,1) == x_par;
%% define constraints
for j = 1:N
    x(:,j+1) == A*x(:,j) + B*u(:,j);
    u(:,j) <= u_upper;
    u(:,j) >= u_lower;
end
minimize(obj);
prob = splitProb.genProblem;
```

Figure 3.6 – An MPC problem defined using SPLIT in MATLAB

Scalar operations, e.g. computing Nesterov's relaxation, do not require acceleration, since computational complexity for these operations does not scale with the problem size.

Vector-vector operations often can be accelerated by pipelining[1] the loop that iterates over vectors elements. For the simplest case, when consecutive iterations do not depend on each other (e.g. element-wise addition), this is implemented by supplying the source code with a pipelining directive. However, for some vector operations pipelining cannot be implemented in a straightforward manner, which is a consequence of read-write data dependencies. Consider the example of computing a sum of vector's elements (Figure 3.7 and 3.8).

With a software-oriented approach the next iteration of the main loop cannot be started before finishing the previous iteration due to data reading and writing dependencies, which applies restrictions on pipelining. Hardware-oriented code computes partial sums of vector's elements independently, which allows avoiding undesirable data dependencies and creating an efficient pipeline. After the partial accumulation is finished, the final loop accumulates the partial sums. According

[1]We will discuss pipelining and parallelism in details in Section 4.1.

to the report from the synthesis tool (Vivado HLS), the FPGA implementation of `vec_sum_hw()` is 4.75x faster compared to `vec_sum_sw()`. Note that the same memory access pattern with the considered example holds for calculating vector norms, vector-vector and matrix-vector multiplications, which is exploited by SPLIT.

```
1   // SW oriented code
2   float vec_sum_sw(float vec_in[1000])
3   {
4   int i;
5   float sum 0;
6   for(i = 0; i < 1000; i++)
7   {
8   sum += vec_in[i];
9   }
10  return sum;
11  }
```

Figure 3.7 – Software oriented C-code for calculating sum of vector elements.

```
1   // HW oriented code
2   float vec_sum_hw(float vec_in[1000])
3   {
4   int i,j;
5   int mask[2] = {0, ~((int) 0)};
6   float sum_p[8] = {0};
7   for(i = 0, j = 0; i < 1000; i++) // partial accumulation
8   {
9   #pragma HLS DEPENDENCE variable=sum_p array inter distance=8 true
10  #pragma HLS PIPELINE
11  sum_p[j] += vec_in[i];
12  j = (j+1) & mask[(j+1) != 8];
13  }
14  for(i = 1; i < 8; i++) // final accumulation
15  {
16  #pragma HLS UNROLL
17  sum_p[0] += sum_p[i];
18  }
19  return sum_p[0];
```

Figure 3.8 – Hardware oriented C-code for calculating sum of vector elements.

Matrix-vector computations required for splitting methods can be classified into dense matrix-vector multiplication, sparse matrix-vector multiplication and sparse

forward/backward substitution. Dense matrix-vector multiplication essentially represents a set of vector-vector multiplications, which can be computed in parallel. SPLIT allows trading off computation time against FPGA resource usage by changing the degree of parallelism, i.e. the number of rows processed in parallel. Regarding sparse matrix-vector computations, SPLIT handles non-zero elements in a certain order that allows avoiding data dependencies and hence opens possibility of pipelining. This is achieved by scheduling, i.e. determining the order of processing non-zero elements, offline, at code generation stage.

Once tailored C code with synthesis directives is generated, SPLIT uses the Protoip toolbox for synthesizing the code and deploying the controller on FPGA. Using Protoip on the underlying level allows quick prototyping of the algorithms on hardware and closed-loop performance verification with processor-in-the-loop tests. Note that the entire design flow is fully automated and no prior FPGA knowledge is required from a user.

In this way, by just defining an MPC-based formulation in a high-level language like MATLAB, users can deploy a controller on FPGAs or heterogeneous platforms. This is the first free and open source toolbox that provides code generation and deployment of an MPC-based controller on FPGAs or heterogeneous platforms.

3.4 Processor-in-the-Loop Experimental Results

In this work we use a Xilinx Zynq-7000 XC7Z020 SoC with dual-core ARM Cortex-A9 and FPGA logic, which contains: 53200 LUTs, 106400 FFs, 220 DSP blocks and 140 block RAMs with total capacity 4.9 Mb. A high throughput-oriented AXI bus provides fast communication between the CPU and FPGA and hence allows heterogeneous implementations of computationally intensive algorithms. We consider the following optimal control problem for the rest of the section:

$$\begin{aligned} \text{minimize } & \sum_{i=0}^{N-1} \left(x_i^\top Q x_i + u_i^\top R u_i \right) + x_N^\top Q_N x_N \\ \text{subject to } & x_{i+1} = A x_i + B u_i, \text{ for } i = 0, \ldots, N-1 \ , \\ & x_i \in \mathcal{X} \ , \text{ for } i = 0, \ldots, N \ , \\ & u_i \in \mathcal{U} \ , \text{ for } i = 0, \ldots, N-1 \ , \\ & x_0 = \hat{x} \ , \end{aligned} \tag{3.4.1}$$

where the vector $x_i \in \mathbb{R}^{n_x}$ represents states, $u_i \in \mathbb{R}^{n_u}$ is the input vector, Q and R are penalty matrices on states and inputs with appropriate dimensions. $A \in \mathbb{R}^{n_x \times n_x}$ is state transition matrix, $B \in R^{n_x \times n_u}$ is input matrix. We consider $\mathcal{X}$, $\mathcal{U}$ are convex sets for constraints on the states and inputs respectively. $\hat{x}$ is an initial state and N is a prediction horizon.

3.4.1 Software, Heterogeneous and Hardware Implementations

In this example, we randomly create predictive control problems of the form (1.3.1) while varying number of states, inputs and horizon length as provided in Table 3.3.

	n_u	n_x	N	Dimensions
Problem 1	2	4	4	48
Problem 2	4	8	7	156
Problem 3	6	12	12	384

Table 3.3 – Generated problems with varying size

Parameter "Dimensions" in Table 3.3 represents the matrix size when solving a linear system. We incorporate box constraints on inputs. The optimization problem is solved using an accelerated version of AMA with single precision floating point arithmetic. The goal is to compare and study the latency and resource usage for implementation on an FPGA, heterogeneous platform and on a general-purpose embedded processor. We consider the following five scenarios:

1. **Implementation on FPGA with synthesis directives:** In this case, all the numerical operations are performed on an FPGA. The code generated by SPLIT with synthesis directive is used. The latency is illustrated in Figure 3.9 as PL_directive.

2. **Implementation on FPGA without synthesis directive:** In this case, all synthesis directives are commented to compare and study efficiency of generated C code with directives by SPLIT. The latency performance is illustrated in Figure 3.9 as PL.

3. **Heterogeneous implementation:** In this experiment, all the operations, except solving linear systems, are computed on the embedded processor and

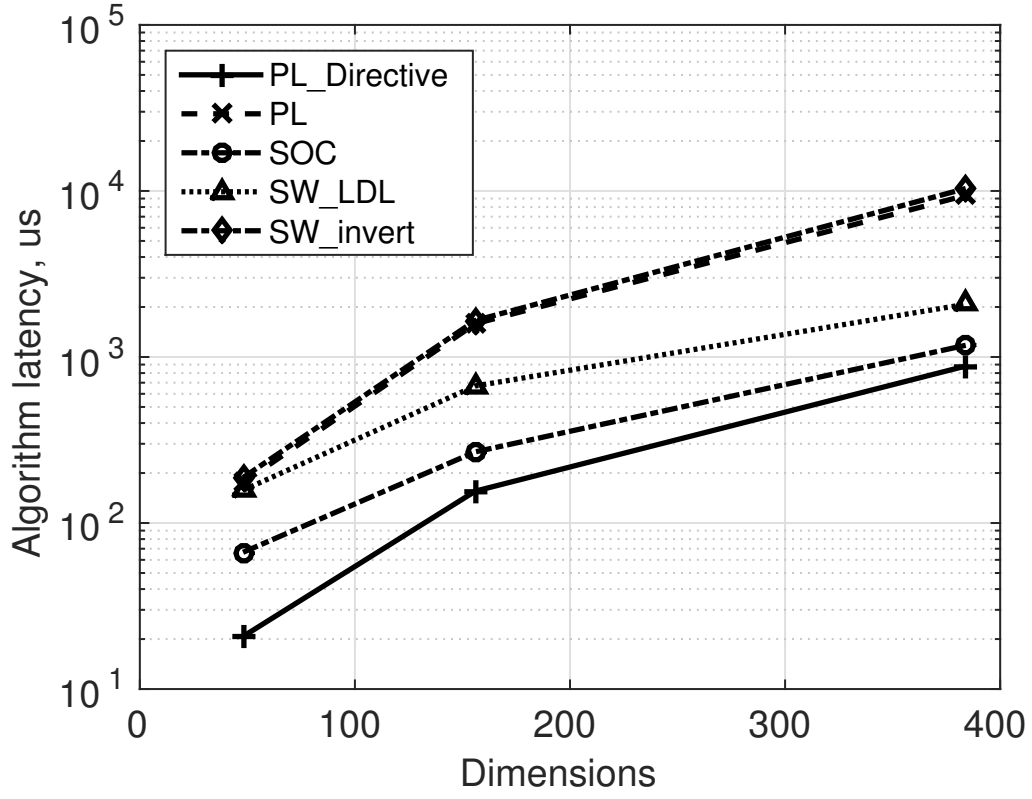


Figure 3.9 – Time per iteration with varying size of Problem log scale

the linear system is solved on an FPGA. This is because the computational bottleneck for splitting methods is the solution of a linear system. Since the matrix involved in solving the linear system does not change over iterations, it is preloaded on the FPGA at circuit synthesis stage. Thus there is no online memory communication for the matrix between the embedded processor and FPGA. Heterogeneous implementation is also inherently supported by SPLIT. The latency of this approach is illustrated in Figure 3.9 as SOC.

4. **Software-based implementation of LDL:** In this approach, all the numerical calculations are performed using the onboard dual-core ARM Cortex-A9 processor. The linear system is solved using LDL factorization. In Figure 3.9, this is denoted by SW_LDL.

5. **Software-based implementation of mat-vec:** To compare the performance of LDL factorization, we precompute the inverse of the matrix in the linear solve and perform matrix-vector multiplication online. All the numerical operations are computed using onboard embedded processor. For latency, see Figure 3.9 with label SW_invert.

For all considered scenarios the CPU clock frequency was set to 667 MHz, while the FPGA was clocked at 100 MHz. We do not parallelize any operations on the FPGA for this example to have a fair comparison with a pure software-based implementation on an embedded processor. As illustrated in Figure 3.9, implementation on an FPGA with synthesis directives has the least latency, followed by a heterogeneous platform and a software implementation with LDL factorization. This shows the trade-off between a pure FPGA-based implementation and a pure software-based implementation. It is important to note here that an FPGA-based implementation without synthesis directives has a higher latency than an LDL-based pure software implementation. Since SPLIT is tailored to an algorithm, it performs synthesis directives during code generation and thus is ready to deploy for end users. It is also interesting to observe from Figure 3.9 that solving the linear system based on precomputing the inverse in a software-based implementation has the worst performance. We also note here that to make a library-free implementation, we apply a custom LDL factorization, which exploits sparsity. Since the linear system solve step has a sparse matrix, the factorization is sparse as well. Thus, the time taken for a factorization-based solver is better than computing the inverse and performing matrix-vector multiplication. However, due to lack of definite pattern in-fill, the forward backward solve is not suitable for implementing on FPGAs. A heterogenous implementation is a good trade-off between a pure FPGA implementation and

	Problem 1	Problem 2	Problem 3
PL_directive	11.82	13.64	13.64
PL	11.82	12.73	12.73
SOC	2.27	2.27	2.27

Table 3.4 – DSP usage in %

	Problem 1	Problem 2	Problem 3
PL_directive	19.70	22.42	25.72
PL	19.67	21.49	24.74
SOC	3.51	3.27	3.34

Table 3.5 – LUT usage in %

implementation on an embedded processor. This is due to the fact the main computational bottleneck is performed on the FPGA.

We summarize resources used in Tables 3.4, 3.5, 3.6 and 3.7. It is worth noting that a synthesis directive-based implementation uses similar amounts of resources as one without a synthesis directive, while offering much better latency. A heterogeneous implementation uses less resources compared to a pure FPGA-based implementation, while suffering in latency.

	Problem 1	Problem 2	Problem 3
PL_directive	7.54	8.86	11.76
PL	7.66	8.70	11.62
SOC	1.44	1.33	1.34

Table 3.6 – FF usage in %

	Problem 1	Problem 2	Problem 3
PL_directive	2.68	9.82	50.53
PL	2.15	9.46	50.18
SOC	1.60	6.60	46.61

Table 3.7 – BRAM usage in %

3.4.2 Trade-off : Latency Versus Resources

In this example, we illustrate the trade-off between resource usage and latency. We randomly generate an MPC problem of form (3.4.1) with 4 states, 2 inputs and horizon length of 5 with box constraints on inputs. We solve the optimization problem using an accelerated version of the alternating minimization algorithm and implement on an FPGA. Varying the number of parallel processors for solving the linear system allows trading off FPGA logic usage against latency (Figure 3.10). As we increase parallelization, the latency improves (3x faster) at the cost of using more resources. Selecting the number of parallel processors for a particular application one has to keep in mind Ahmdal's law, which states that overall algorithm parallelization speedup is limited by the sequential part of the algorithm. This explains why after reaching a certain point parallelizing computations does not improve performance.

3.5 Conclusion

This chapter presented a code generation tool for software, hardware and heterogeneous implementations of predictive control algorithms using operator splitting methods. Experimental results confirmed that generating synthesizable hardware tailored C-code allows achieving 3x to 11x speedup with hardware realizations compared to pure software implementations. Moreover, it was shown that splitting the workload between software and hardware allows achieving a compromise between latency and computational resource utilization.

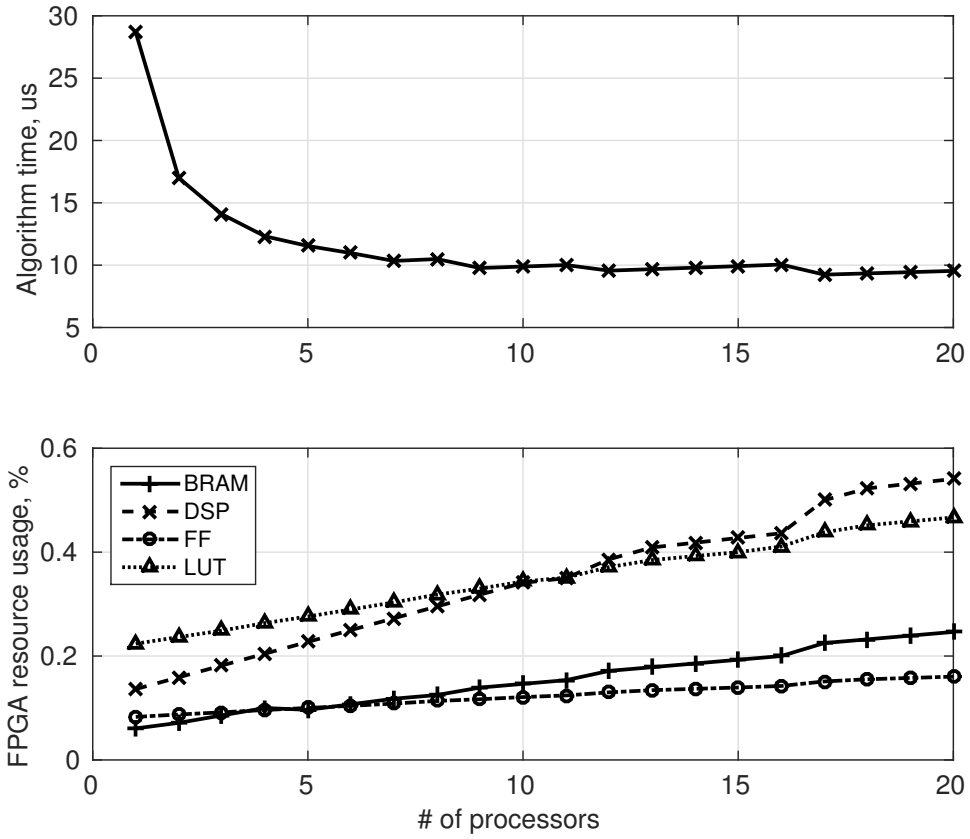


Figure 3.10 – Latency vs Resources trade-off

4 Solving Control Co-Design Problems for Splitting Methods

> There are two ways to write error-free programs; only the third one works.
>
> Alan J. Perlis

In this chapter we solve *a priori* the co-design problem, i.e., a priori finding a trade-off between execution time and utilized resources. For solving a co-design problem, engineers usually rely on a heuristic approach [85, 86], meaning, recompiling the program for different levels of parallelism. In the previous chapter, we solve a co-design problem in a heuristic way (See Section 3.4.2 and Figure 3.10). However, the challenge (besides requiring insight about FPGAs) is slow compilation time of FPGA programs. Therefore, the heuristic approach is time-consuming and laborious. We address this issue by providing closed-form analytical solutions that estimate consumed latency and resources *a priori*, i.e., without performing any kind of FPGA program compilation. In conclusion, the proposed framework helps an FPGA user solving a co-design problem a priori, without the tedious procedure of the heuristic approach. Furthermore, we propose the FPGA-oriented code-generation toolbox-LAFF. The main difference between SPLIT and LAFF is that LAFF is a general purpose tool while SPLIT is restricted to splitting methods. On the other hand, LAFF only supports code generation for FPGAs, while SPLIT supports different embedded platforms as discussed in the previous chapter.

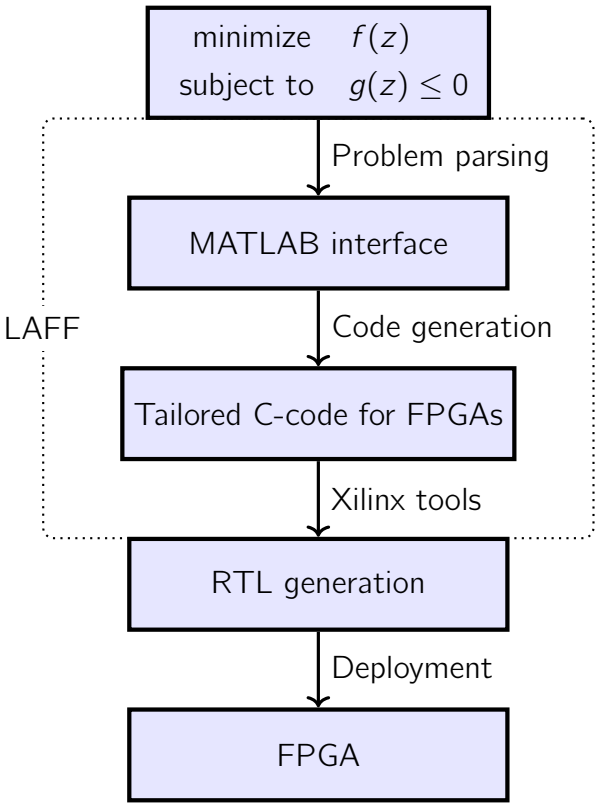


Figure 4.1 – Workflow for algorithm deployment on FPGA using LAFF.

4.1 LAFF: a Code Generation Toolbox

Overview of the toolbox

LAFF is a code generation toolbox that enables users to define a target algorithm in a high-level language, MATLAB. The MATLAB program is then parsed and a hardware-oriented C-program is generated for optimal performance. The generated C-program is tailored for the High Level Synthesis (HLS) tool provided by Xilinx. This concept is illustrated in Figure 4.1 for an optimization algorithm. A code snippet for parsing the gradient projection algorithm is provided in Figure 4.2. Note the difference in parsing between SPLIT (Figure 3.6) and LAFF (Figure 4.2).

The generated C-code from LAFF is customized for hardware implementation because it exploits the concepts of pipelining, parallelism, efficient memory access, loop-unrolling, loop-flattening and fixed-point as well as floating-point arithmetic.

Moreover, like SPLIT, the proposed toolbox supports the ProtoIP toolbox, which

```
settings.datatype = 'fixed';
settings.integ_bits = 5;
settings.frac_bits = 12;

%% initial setup
laff_init(inputs, outputs, settings);
%% Write variables
laff_write_data('I_H', I_H, 'real');
%% choose parallelism level
PAR_requested = 5;

% begin for gradient projection
laff_for_loop_begin('itr', 'main_loop');
% copy vector
laff_copy_vector('z_prev', 'z', n);
% matrix vector
laff_MV_MAC(I_H, PAR_requested, 'y', 'z', []);
% vector subtraction
laff_vector_scale_add('t', 'z', 'lf', n, 1, -1 );
% projection on box
laff_box_clipping('z', 't', n, lmin, umax);
% vector scaling and subtraction
laff_vector_scale_add('y', 'z', ...
... 'z_prev', n, (1+beta), -beta );
%end for algorithm
laff_for_loop_end;

laff_end;
```

Figure 4.2 – Gradient projection parser of LAFF in MATLAB

facilitates the deployment of the generated code automatically on FPGAs without studying Xilinx tools. As LAFF supports communication between MATLAB and FPGAs, users can access and analyze results obtained from FPGAs in MATLAB. In what follows, we explain how Matrix-Vector (MV) and vector manipulations are implemented in LAFF as well as SPLIT for exploiting various FPGA features.

Vector manipulations

First we analyze operations with $\mathcal{O}(n)$ complexity namely vector addition and product.

Element-wise vector multiplication and addition: Consider the following operation,

$$z = x \odot y + w \tag{4.1.1}$$

where $x, y, z, w \in \mathbb{R}^n$ and $\odot$ represents element-wise multiplication. In C programming, this is implemented using a loop with n iterations. Each iteration of

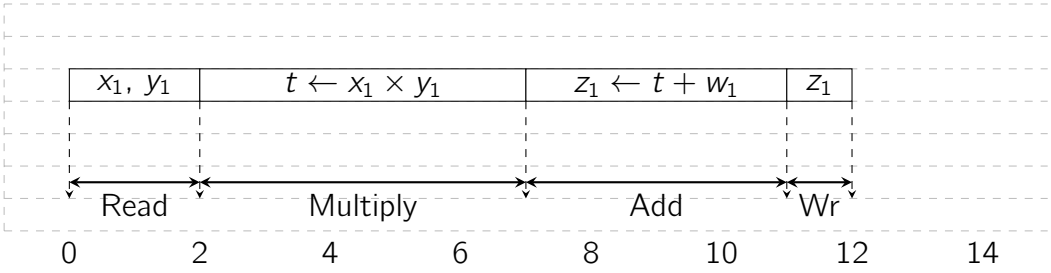
the loop involves three data-read operations, one scalar multiplication, one scalar addition and a data-write operation, as illustrated in Figure 4.3a. The horizontal axis represents the number of clock cycles required to execute these operations, which are 2, 5, 4 and 1, respectively. The Loop Iteration Latency (LIL) is defined as the number of clock cycles needed to execute one iteration of the loop, and in this case is 12. The total number of clock cycles required to multiply and add n elements is $12n$, known as Loop Latency (LL). The overall latency can be reduced by exploiting the concepts of pipelining and parallelism introduced in the following paragraphs.

Pipelining enables loops to execute in a concurrent manner, meaning that the execution of the next iteration starts before the previous one ends. The number of clock cycles required to begin the next iteration before finishing the previous iteration is called the Initiation Interval (II). In Figure 4.3b, we illustrate pipelining with the II equal to one clock cycle. This is the best-case scenario and, as we shall discuss later, whether it is possible to achieve this best-case depends on the type of operation involved and the writing style of the program. For the operation in (4.1.1), the number of clock cycles required to execute all iterations is $12 + n - 1$, lower than without pipelining - $12n$. We note that extra resources needed for pipelining do not scale with the number of iterations.

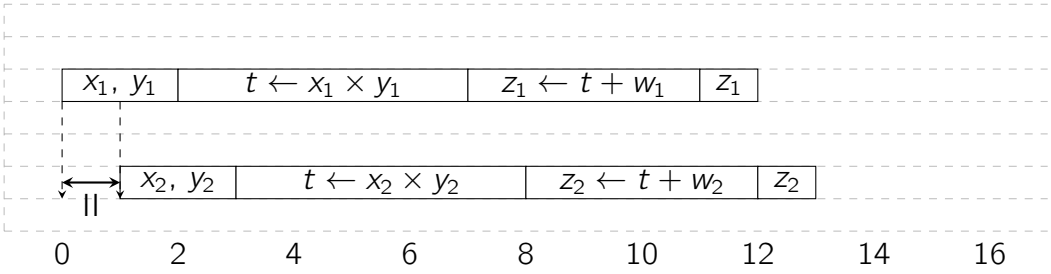
The execution time of the operation in (4.1.1) can be significantly reduced by computing each iteration in *parallel* as illustrated in Figure 4.3c. The bottleneck with parallelism is accessing more than two vector-elements simultaneously because a maximum of two elements can be accessed from a single Random Access Memory (RAM). However, more than two elements can be accessed if the vector is partitioned and its elements are stored in different BRAM on an FPGA (For example Xilinx's Zedboard has 280 BRAMs of size 18Kb). When all the iterations are computed in parallel, the latency is equal to the LIL, i.e., the number of clock cycles required for executing one iteration. However, the better performance in terms of latency comes with a drawback that the parallel execution needs more computational units, and therefore more resources.

Vector inner product: Consider an inner product operation:

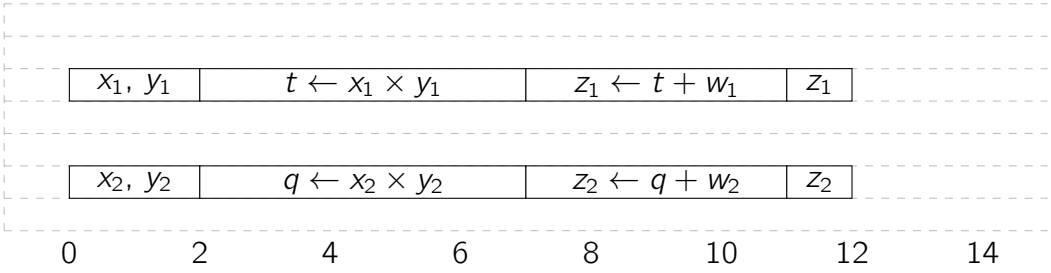
$$a = x^\top y, \tag{4.1.2}$$



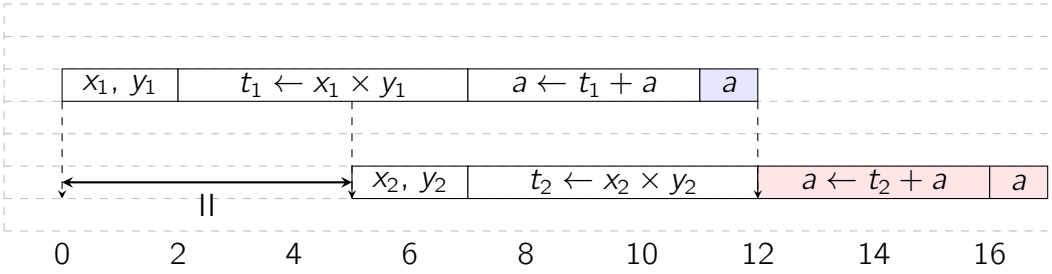
(a) Sequential Computation of operation given $z = x \odot y + w$ (equation (4.1.1)).



(b) Reducing latency using pipelining for $z = x \odot y + w$ (equation (4.1.1)) .



(c) Reducing latency using parallelism for $z = x \odot y + w$ (equation (4.1.1)).



(d) Illustrating read-write dependency for $a = x^\top y$ (equation (4.1.2)).

Figure 4.3 – Different scenarios for computations on FPGAs.

where $x, y \in \mathbb{R}^n$ and a is a scalar. In C programming, this is computed using a loop of n iterations with the i-th iteration computing $a = a + x_i * y_i$. Figure 4.3d illustrates that the first iteration finishes computing a at the end of the 11-th clock cycle which

makes the second iteration wait[1] until the 11-th clock cycle before reading and writing scalar a, forcing the Initiation Interval equal to 5. This phenomena is known as *read-write dependency*. The total number of clock cycles for computing pipelined vector inner-products is $(n-1)\mathrm{II} + \mathrm{LIL}$. LAFF/SPLIT detects and overcomes the *read-write dependency* problem by storing the results in buffers, and for the case in Figure 4.3d, the proposed toolboxes use five scalar buffers to remove read-write dependency giving a latency of $\mathrm{LIL} + n - 1$ clock cycles.

Remark 4.1.1. *LAFF/SPLIT supports various other $\mathcal{O}(n)$ complex operations, e.g., evaluating different indicator functions, computing norms, etc. We will not go into detail about its efficient implementation because it uses the same concepts discussed in this subsection.* □

Matrix-Vector multiplication

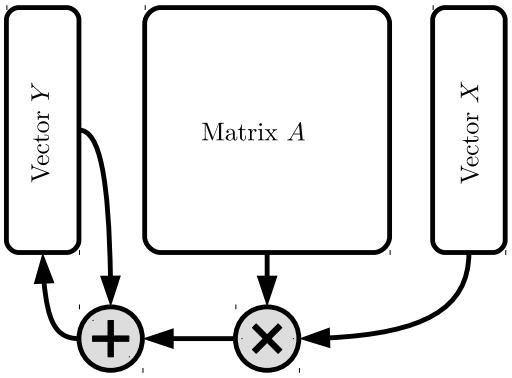
The Matrix-Vector (MV) multiplication has $\mathcal{O}(n^2)$ computational and memory complexity. Next, we explain how the proposed toolboxes trade-off between latency and resources by varying the level of parallelism. Due to this feature, users are not restricted to the minimum latency or minimum resource consumption solutions.

The MV multiplication can be computed as a sequence of vector inner products (Figure 4.4). In the previous subsection, we discussed the data *read-write* challenges of a vector inner product. Note that, unlike row-wise access, the column wise-access of matrix entries does not have read-write dependencies and needs buffer-vectors only when the number of rows is larger than the Initiation Interval. Therefore we process MV column-wise. Furthermore, it improves the execution time using parallelism.

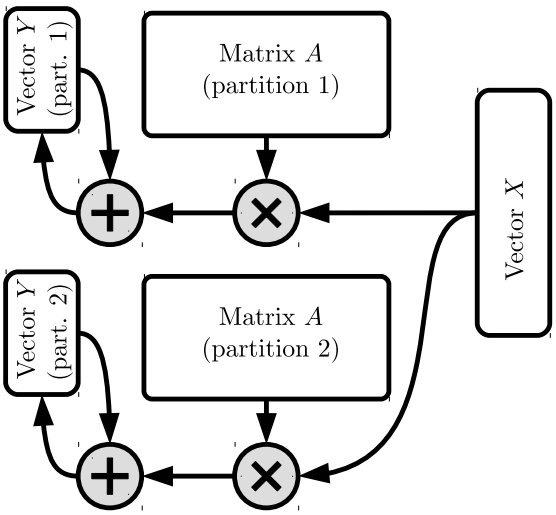
LAFF/SPLIT improves the MV latency by exploiting parallelism. First, the matrix is partitioned row-wise (Figure 4.4b and 4.5) and each partition is stored in different BRAMs. For parallel execution, one element of each partition gets multiplied with a corresponding vector-element in the same clock cycle. For the example given in Figure 4.5, the first iteration computes $a_{11} * x_1$ and $a_{31} * x_1$ simultaneously and then the second iteration computes $a_{21} * x_1$ and $a_{41} * x_1$ simultaneously.

The first advantage of the proposed procedure is that matrix and vector elements are accessed from memory only once. For example, once x_1 is accessed, x_2 is not retrieved until all multiplication related to x_1 is finished–resulting in memory efficient

[1] Note that the result is erroneous if the second iteration reads scalar a before 12-th clock cycle.



(a) Sequential Matrix-Vector multiplication.



(b) Parallel Matrix-Vector multiplication.

Figure 4.4 – Sequential versus parallel Matrix-Vector multiplication [68].

access. The second advantage is that there is no *read-write dependency* as long as the number of rows of each partition is larger than the Initiation Interval. In case the number of rows in each partition is less than the Initiation Interval, buffer vectors (denoted as y^1 and y^2 in Figure 4.5) are introduced and the final result is computed by adding them. Thus, MV in proposed toolboxes is computed in four steps, as listed in Algorithm 4.1.1 and illustrated in Figure 4.4.

Remark 4.1.2. *If partitions cannot be done in equal size then all but the last partition has the same dimensions.* □

Remark 4.1.3. *Varying the number of partitions of each matrix allows obtaining the Pareto-optimal curve for trading-off between the execution time and utilized resources.* □

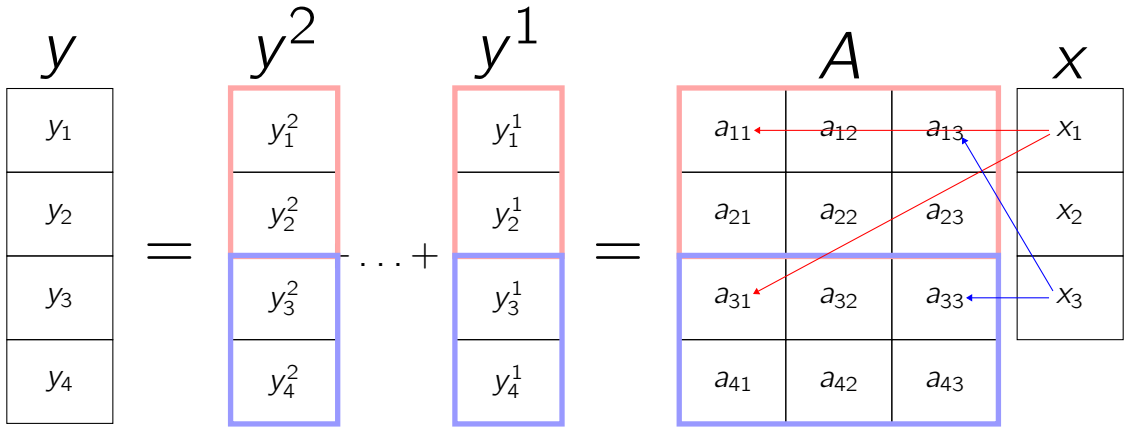


Figure 4.5 – A parallel MV implementation in LAFF and SPLIT.

Algorithm 4.1.1 Matrix Vector multiplication steps in LAFF

Step 1: Partition the matrix and create buffer vectors

Step 2: Set buffer vectors to zeros

Step 3: Compute partition-wise multiplication

Step 4: If buffer vectors are used then add them to compute the final product

4.2 Estimating Latency and Resource Consumption

In this section we provide a framework to *a priori* estimate the latency and the consumed resources for operations involved in splitting methods. Therefore, saving efforts of HLS programmers/users from performing laborious and time-consuming HLS synthesis. One of the advantages of the proposed framework is that it can be easily generalized and applied to other algorithms besides splitting methods. In what follows, we explain a proof of concept about the proposed framework. Based on this proof of concept, we provide closed-form expressions for estimating latency and resources for matrix-vector product and vector manipulations in Appendix 4.5.1 and Appendix 4.5.2 respectively.

Estimating Latency

We utilize the following four ingredients to estimate the latency:

1. **Deterministic execution time:** An FPGA needs a fixed number of clock cycles to perform various operations, e.g., scalar addition and multiplication, executing conditional statements, copying, entering and exiting a loop, etc. We find clock-cycles for these various operations and utilize them as a building-blocks to estimate the latency required for a complex operations.

2. **Latency required by a single loop execution:** A pipelined single loop with n iterations needs $(n-1)$II+LIL clock cycles where II is the loop Initiation Interval and LIL is the Loop Iteration Latency as explained in Section 4.1.1. If the loop is completely parallelized, then it needs LIL clock cycles to compute the task.

3. **Latency required by nested loops execution:** Nested loop latency is estimated using individual loop latency recursively.

4. **Latency required by sequential and parallel function executions:** For a sequential execution of functions, the total latency is the sum of the individual functions/operations. For a parallel execution of functions, the total latency is the latency of the function with maximum latency.

Using the above ingredients, the latency required for any algorithm can be calculated as illustrated in Appendix 4.5.

Estimating DSP usage

DSPs are used to compute addition and multiplication. The Number of DSPs required to compute a scalar addition and multiplication are 2 and 3, respectively. Pipeling the loop does not lead to any extra DSP usage, while parallelizing with factor m leads to m times more usage of DSP units. Therefore, we estimate the DSP usage of an algorithm based on a number of scalar additions and multiplications incorporating the level of parallelism.

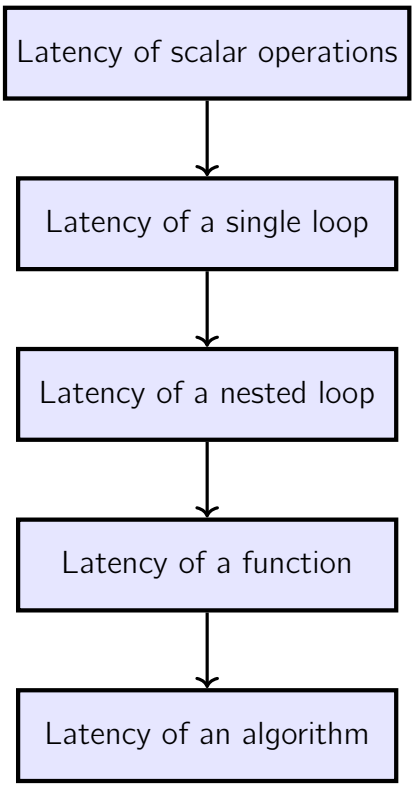


Figure 4.6 – Building blocks for hierarchical computation of latency.

Estimating BRAM usage

BRAM is used as a memory to store the variables. As discussed earlier, FPGAs have a set of small RAM, called as BRAM. The number of BRAMs required to store an array with n elements and m bits precision is

$$R^{bram} = \text{ceil}\left(\frac{n * m}{\text{BRAM size}}\right), \tag{4.2.1}$$

where "ceil$(\cdots)$" is the ceiling operation. The size of a single BRAM depends on the FPGA manufacturer and model of the FPGA. When an array is partitioned, each partition is stored in different BRAMs and the number of BRAMs required by each partition is calculated using equation (4.2.1).

Estimating FF and LUT usage

FFs and LUTs are used for storing all the temporary variables, iteration counters, and operations like compare, increment and conditional statements. Consequently,

for the same C program, it is usual to get a different utilization of FFs and LUTs depending on the HLS tools used, the version of the tool and the type of a FPGA. Therefore, accurately predicting their usage is challenging. LAFF estimates conservatively utilization of FFs and LUTs by incorporating the fact that the majority of FF and LUT consumption comes from storing temporary variables and for creating addition/multiplication units.

4.3 A Co-Design Problem for Splitting Methods

As we discussed in the previous chapters, the numerical operations involved for splitting methods are matrix-vector multiplications, vector clipping and vector-addition respectively. We provide closed-form formulas for these operations in Appendix 4.5 using which we address the following three questions:

1. **Finding minimum latency for a given resources:** What is the least achievable latency given limited resources and how to achieve that? This question arises when MPC problem needs to be solved on a given hardware as quickly as possible.

2. **Finding required resources for a given latency:** What resources are needed for achieving the targeted latency? This problem is relevant for a real-time application and helps designers to choose the type of FPGA needed for their applications.

3. **Finding the minimum achievable latency:** What resources are needed to achieve the least possible latency? Since this problem gives the best possible latency, a designer can decide whether the algorithm is capable of achieving the targeted latency. We remind the reader that it is not possible to achieve arbitrarily small latency due to Amdahl's law [87]

Finding minimum latency for a given resources

$$
\begin{aligned}
\underset{P_i}{\text{minimize}} \quad & \sum_i L_i\left(P_i\right) \\
\text{subject to} \quad & \sum_i R_i^{dsp}\left(P_i\right) \leq \text{DSPs available}, \\
& \sum_i R_i^{bram}\left(P_i\right) \leq \text{BRAMs available}, \\
& \sum_i R_i^{ff}\left(P_i\right) \leq \text{FFs available}, \\
& \sum_i R_i^{lut}\left(P_i\right) \leq \text{LUTs available}.
\end{aligned}
\tag{4.3.1}
$$

Let the predicted latency and resources for the i-th numerical operation for splitting methods be $L_i\left(P_i\right)$, $R_i^{bram}\left(P_i\right)$, $R_i^{dsp}\left(P_i\right)$ $R_i^{ff}\left(P_i\right)$ and $R_i^{lut}\left(P_i\right)$ where P_i denotes the opted parallelism level. The solution of the integer optimization problem (4.3.1) determines the parallelism level of each operation to achieve the least possible latency given the resource constraints. Note that the solution can also be used for finding the number of maximum possible iterations to execute an algorithm, given constraints on resources and sampling time. We note that in the scenario where more than one algorithm needs to be deployed and run in parallel on the same FPGA, problem (4.3.1) takes the form of problem (4.3.2), where superscript j is used to denote the algorithm index. Formulation (4.3.2) is useful for applications where resources are shared amongst different applications and algorithms. One example is surveillance based drones where a controller and image-processing units potentially share the same resources. In such cases, we are interested in solving the following problem:

$$
\begin{aligned}
\underset{P_i^j}{\text{minimize}} \quad & \sum_j \sum_i L_i^j\left(P_i^j\right) \\
\text{subject to} \quad & \sum_j \sum_i R_i^{j^{dsp}}\left(P_i\right) \leq \text{DSPs available}, \\
& \sum_j \sum_i R_i^{j^{bram}}\left(P_i\right) \leq \text{BRAMs available}, \\
& \sum_j \sum_i R_i^{j^{ff}}\left(P_i\right) \leq \text{FFs available}, \\
& \sum_j \sum_i R_i^{j^{lut}}\left(P_i\right) \leq \text{LUTs available}.
\end{aligned}
\tag{4.3.2}
$$

Finding required resources for a given latency

For real-time embedded applications, where latency as well resources have paramount importance, we are interested in finding the least amount of resources required for achieving the targeted latency. In such applications, resources are limited and can not be overused due to various constraints, e.g., available power onboard or shared resources. The following formulations can be used to solve such problems:

$$\begin{aligned} \underset{P_i}{\text{minimize}} \quad & \sum_i \left(R_i^{dsp}(P_i) + R_i^{bram}(P_i) + R_i^{lut}(P_i) + R_i^{ff}(P_i) \right) \\ \text{subject to} \quad & \sum_i L_i(P_i) \leq \text{Target Latency}. \end{aligned} \tag{4.3.3}$$

Finding the minimum achievable latency

Next, we consider a problem to achieve the least latency assuming no constraints on resources, i.e., solving problem (4.3.1) without constraints (see problem (4.3.4)). When control decisions are needed to be taken as fast as possible, Solving problem (4.3.4) helps a designer finding the least latency and resources required

$$\underset{P_i}{\text{minimize}} \quad \sum_i L_i(P_i) \tag{4.3.4}$$

Remark 4.3.1. *The techniques proposed in Section 4.2 and 4.3 are not restricted to splitting methods and can be applied to any identification or control problem targeting FPGAs.* □

4.4 Numerical Examples: Solving an MPC Co-Design Problem for a Ball-Plate System using ADMM

In this subsection, we consider controlling a ball and plate system using MPC of structure given in equation (4.4.1). The detailed problem description and parameters used can be found in [88]. The system has two states, one input and the prediction horizon equal to 15. Box-constraints are imposed on the states and input. This

leads to 45 optimization variables. The MPC problem is solved using Alternating Direction Method of Multipliers (ADMM) described as Algorithm 1.2.3. Note that the first step involves solving a linear system of equations with dimension 75. This is achieved by precomputing the matrix inversion. The second step of Algorithm 1.2.3 is an indicator function on a box-constraints. The third step is computed using vector multiplication and additions. In total, one matrix-multiplication, one box-clipping and four vector multiplications and additions are involved.

$$\begin{aligned} \underset{x_i, u_i}{\text{minimize}} \ & \sum_{i=0}^{N-1} \left(x_i^\top Q x_i + u_i^\top R u_i \right) \\ \text{subject to } & x_{i+1} = A x_i + B u_i, \ \text{for } i = 0, \ldots, N-1 \ , \\ & x_i \in \mathcal{X} \ , \ \text{for } i = 0, \ldots, N \ , \\ & u_i \in \mathcal{U} \ , \ \text{for } i = 0, \ldots, N-1 \ , \\ & x_0 = \hat{x} \ , \end{aligned} \tag{4.4.1}$$

where the vector $x_i \in \mathbb{R}^{n_x}$ represents states, $u_i \in \mathbb{R}^{n_u}$ is the input vector, the matrix $Q = \begin{bmatrix} 100 & 0 \\ 0 & 10 \end{bmatrix}$ and $R = 1$ are penalty matrices on states and inputs with appropriate dimensions. $A = \begin{bmatrix} 1 & 0.01 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{n_x \times n_x}$ is state transition matrix, $B = \begin{bmatrix} -0.0004 \\ -0.0701 \end{bmatrix} \in R^{n_x \times n_u}$ is input matrix. The constraints on the states ($\mathcal{X}$) and inputs ($\mathcal{U}$) are box constraints. They are defined as $\mathcal{X} := [-0.2, 0.01] \times [-0.1, 0.1]$ and $\mathcal{U} := [-0.0524, 0.0524]$. $\hat{x}$ is an initial state and N is a prediction horizon.

We consider three Xilinx FPGAs as reported in Table 4.1. All the experiments are performed using Vivado-HLS 2017.4 with target clock frequency of 100 MHz. The aim is to find the minimum latency, while ensuring the algorithm can fit on the given FPGA board by solving the optimization problem given in equation (4.3.1). We first *a priori* estimate latency for all operations involved with pipelining enabled and without parallelization (see Figure 4.7). It is clear that the matrix-vector multiplication, the first step of Algorithm 1.2.3, is the computational bottleneck. Thus, the goal is to find a priori, an allowed parallelization level for matrix-vector product. We report the suggested level of parallelization found a priori in Table 4.1. We want to remind the reader, that this was achieved without any sort of deployment, thus avoiding time-consuming synthesizing procedure. The suggested parallelization

level was then used for generating C-code using LAFF. The results obtained in terms of resources and latency are reported in Table 4.1 and Figure 4.8. It is evident that the estimates obtained a priori are very close to given by Vivado-HLS. We also conclude that for Artix XC7A12 and XC7A35, the limiting factor is the number of DSPs and for Spartan XC7S2S, limited LUTs availability prohibits from further parallelization.

Resources	Artix-7 XC7A12	Spartan-7 XC7S25	Artix-7 XC7A35
Suggested Parallel-level	1	8	15
Latency (clock cycles)	6172	1052	826
DSPs-Used	17	52	87
DSPs-Available	20	80	90
DSPs-%	85%	65%	96%
BRAMs-Used	23	24	22
BRAMs-Available	40	90	100
BRAMs-%	46%	26%	22%
FFs-Used	2870	7952	12014
FFs-Available	20000	29200	41600
FFs-%	14%	27%	28%
LUTs-Used	4856	6949	16603
LUTs-Available	10000	14600	20800
LUTs-%	48%	47%	79 %

Table 4.1 – Suggested parallelization level for different Xilinx FPGAs and consumed resources by Vivado-HLS for the synthesized program.

4.5 Appendix: Closed-Form Formulas for Estimating Latency and Resources

In this section, we provide closed-form formulas involving latency and resource estimations for linear algebra operations involved in the numerical examples.

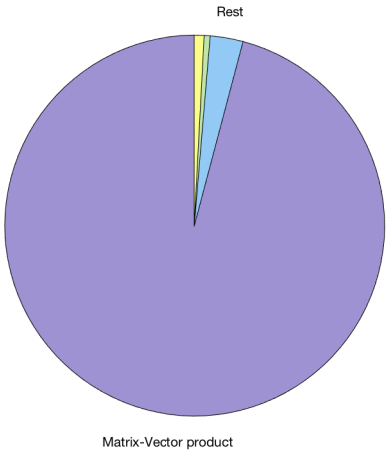


Figure 4.7 – A priori finding latency bottle-neck using the proposed framework.

4.5.1 Formulas for Matrix-Vector Product

Latency

The number of clock cycles for the matrix-vector product is computed as follows.

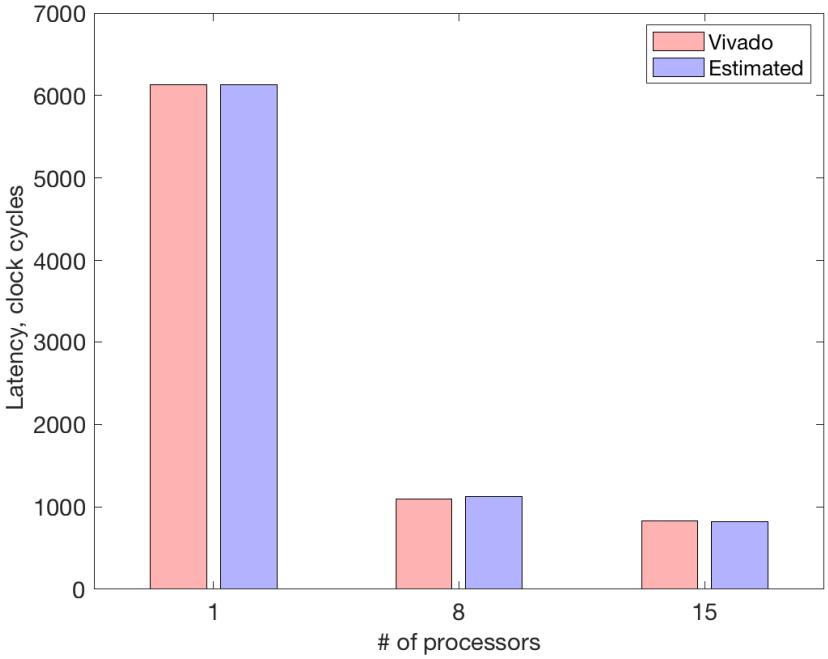
$$\begin{aligned} L_1 &= \text{Column\_Size} \times \text{Part\_Size} + \text{LIL}, \\ L_2 &= \text{Acc\_Size} \times \text{Part\_Size} * \text{PAR} + \text{LIL}, \\ L_3 &= \text{ACC\_Size} \times \text{Part\_Size} + \text{LIL}, \\ L &= L_1 + L_2 + L_3, \end{aligned}$$

where Column_Size, PAR, Part_Size, LII is the number of columns, the number of partitions, the number of rows in each partitions and the Loop Iteration Latency respectively. The ACC_Size is determined based on the number of buffer vectors.

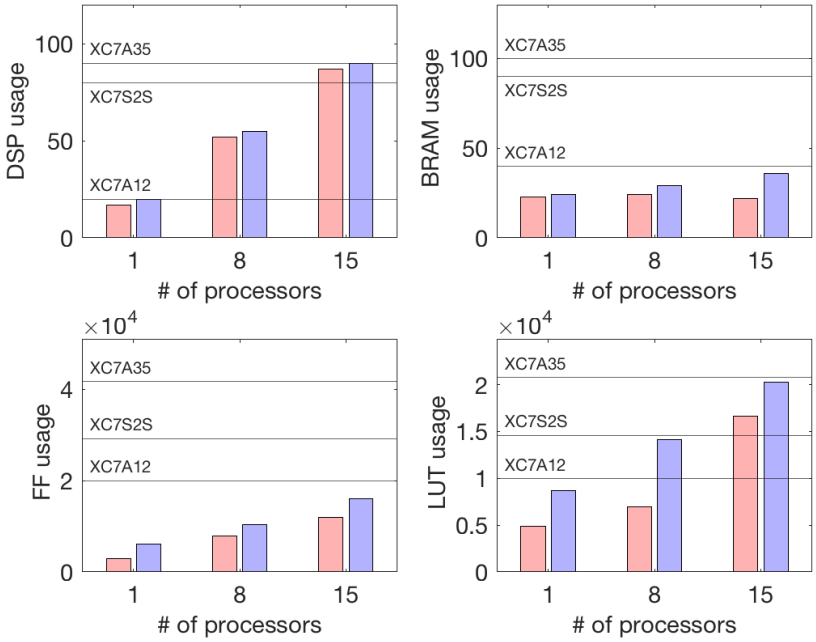
BRAM consumption

The BRAM consumption for matrix-vector product represented by N_bits is the following:

$$R^{bram} = \text{PAR} \times \text{ceil}\left(\frac{\text{Column\_Size} \times \text{Part\_Size} \times \text{N\_bits}}{\text{BRAM\_size}}\right).$$



(a) Comparision of latency given by the proposed framework with Vivado-HLS.



(b) Comparision of resources given by the proposed framework with Vivado-HLS. The maximum available resources represented by horizontal line.

Figure 4.8 – Solving co-design problem *a priori* for the ball and plate system.

DSP Resource consumption

$$R^{dsp} = \text{PAR} \times (\text{add} + \text{mul}) \,,$$

where *add* and *mul* is required DSP for a scalar addition and a scalar multiplication which is two and three respectively for floating point arithmetic. It gets multiplied by *PAR* because we unroll the loop.

FF and LUT consumption

As discussed in the Section 4.2, it is challenging to precisely estimate these two resources because all the temporary variables, iteration counters, operations like compare, increment and logic gates are created using FFs and LUTs. Moreover, the resource utilization can vary depending on the tools and type of an FPGA. Here, we present closed-form formula which overestimates consumption of these resources.

$$R^{ff} = 700\,(\text{PAR} + 1)\,.$$
$$R^{lut} = 900\,(\text{PAR} + 1)\,.$$

4.5.2 Vector-Vector Operations

The main three vector operations involved for the numerical example are (i) copying a vector, (ii) scaling and addition of two vectors, and (iii) an indicator function on box-constraints. We list closed-form estimation formulas for these operations in Table 4.2. Resources are estimated by multiplying the length of vector by factors reported in Table 4.3.

Operation	Definition	Latency (clock cycles) length + LIL
Copy Vector	$y = x$	$L = \text{Length} + 2$
Vector Scale Add	$z = \alpha x + \beta y$	$L = \text{Length} + 13$
Projection on Box	$y \in [lb, ub]$	$L = \text{Length} + 6$

Table 4.2 – Definition and latency for different Vector operations.

Operation	BRAM	DSP	FF	LUT
Copy Vector	0	0	30	100
Vector Scale Add	0	5	700	900
Projection on Box	0	0	370	700

Table 4.3 – Multiplying factors for resource estimation of different Vector operations.

4.6 Conclusion

This chapter presented LAFF, a code generating tool that enables obtaining hardware-oriented C-code directly from MATLAB. This optimized C-code can then be readily synthesized using Vivado-HLS, helping control engineers to quickly deploy their algorithms on FPGAs. Low-level knowledge is not required to achieve an efficient implementation. Moreover, a framework to estimate latency and consumed resources *a priori* for High Level Synthesis (HLS) is presented. The proposed framework can save time and energy as it does not require to synthesize the HLS program. In our realistic numerical case study, it is illustrated that the estimation is accurate. We find the parallelization level that achieves minimum latency while fitting the problem into different FPGAs.

Part II

Gaussian Process Based Constrained Process Optimization

5 Gaussian Processes Based Data-Driven Optimization

> Noli, obsecro, istum disturbare (Do not, I entreat you, disturb that (sand)).
>
> Archimedes

5.1 Introduction and Outline

Increased computational power, ubiquitous availability of computational resources and improved algorithms have driven steady research interest in Real-Time Optimization (RTO) of uncertain processes. The core objective of RTO is to ensure system operation, while meeting quality specifications and guaranteeing safe operation. Competitiveness can be seen as the ability to reach this objective with a minimal time and investment's cost. Due to plant-model mismatch, purely model-based optimization is often incapable of reaching plant optimality. Even with accurate models, external disturbances may shift the plant optimum, which may result in infeasibility and/or suboptimality. Hence, in order to ensure optimal operation, RTO methods adapt the model-based optimization problem using process measurements.

The task of optimizing the performance of uncertain processes in a run-to-run or periodic fashion appears in several application contexts such as repeated robotic motion tasks [89], airborne wind energy systems [90, 91], or batch-process optimization [92, 93]. Interestingly, run-to-run performance optimization is roughly of the same complexity as optimizing the steady-state performance under plant-model mismatch, modulo the larger number of decision variables in the former problem. Hence it is not surprising that several methods have been proposed to tackle different problem variants—this ranges from iterative learning in predictive control [94, 95], robust

and stochastic optimization [96], and extremum seeking [97, 98], and modifier adaptation (MA) of process systems via RTO [99, 100] to Derivative-Free Optimization (DFO) [101, 102], data-driven control [103] and machine learning [104]. Common to all the methods mentioned is that they deal with plant-model mismatch and that they rely on feedback (or on sampling of unknown functions in the case of DFO and machine learning).

This part of the thesis combines three main elements: (i) RTO (i.e. using process feedback to optimize system performance despite plant-model mismatch), (ii) a derivative-free trust region framework (trading off exploration and performance improvement), and (iii) Gaussian Processes (GPs) as approximators of unknown plant-model mismatch functions. In what follows, we briefly review the state-of-the-art literature related to these three elements in the context of process optimization.

RTO for process optimization

The most commonly used RTO method in industry is the two-step approach, which consists of repeated parameter estimation and optimization [105–108]. However, in the presence of structural plant-model mismatch, this approach tends not to converge to the plant optimum [109, 110].

Modifier adaptation (MA) is an RTO method that uses measurements to correct the cost and constraint functions of the optimization problem [109, 111]. The main advantage of MA is that, under suitable assumptions, it reaches optimality upon convergence. Its drawback is that it typically requires estimates of the plant gradients. Several MA precursors and variants are documented in the literature, including early works [112, 113] and more recent results [109, 111, 114, 115]. Moreover, the link between so-called modifier adaptation schemes and trust region methods has been explored in [116]. A detailed overview of the state of the art is given in [115].

While the basic MA scheme relies on first-order corrections, the use of second-order modifiers has also been proposed [117]. However, accurate Hessian estimation from noisy data is rather difficult in practice. Recently, it has been proposed to locally fit a quadratic function to plant data, with the plant gradients being obtained from this local fit [118, 119]. However, the method depends heavily on the quality of the first-order modifiers, that is, it still requires gradient estimation.

Introduction to DFO based trust-region method

The set of DFO trust-region methods comprises established tools to optimize unknown—or expensive to evaluate—objectives [102]. The pivotal idea is the use of a local surrogate model, built at each iteration by evaluating the objective at a number of sample points within the trust region. Probabilistic derivative-free trust-region methods rely on randomized surrogate models [120, 121]. The key advantage of using a probabilistic method is its ability to capture uncertainties efficiently. This is indeed useful for noisy objectives and/or inaccurate models. However, the key bottleneck of deterministic and probabilistic derivative-free trust-region methods alike is twofold: (i) ensuring the quality of the surrogate model, and (ii) guaranteeing a sufficiently large domain of validity. The former can be achieved via complicated procedures for sample-set maintenance [102], while the latter calls for global surrogate models.

The convergence of trust-region methods relies on the accuracy of the surrogate model within the trust region. Intuitively speaking, the convergence mechanism increases the sampling of the unknown function and decreases the trust-region radius until the local surrogate model is sufficiently accurate in zeroth- and first-order compared to the unknown function. This accuracy, which is defined as "full linearity", helps move in a decent direction. Global convergence of derivative-free trust-region methods for deterministic and stochastic version is described in [102] and [120], respectively. For a constrained-optimization case, global convergence is provided in [122] and is proved by convexifying constraints. In this context, the main challenge is the construction of a surrogate model by performing as few plant evaluations as possible. Hence, if full-linearity can be certified, a global surrogate model is usually preferred over local ones.

Introduction to Machine learning based surrogate modeling

At the same time, there is a recent and steadily growing interest in machine learning techniques in computer science as well as in systems and control. This spans {supervised, reinforcement} learning and data-driven function approximations by deep neural networks [123] and GP [124, 125]. There exists a body of literature on using machine learning for optimization, e.g., [125, 126]. In some cases, it is possible to guarantee convergence to the global minimum of unknown functions. In the machine learning community, Gaussian-process (GP) regression is a popular tool for estimating unknown functions [125, 127]. A GP is a probabilistic, non-parametric

modeling technique that can be interpreted as the extension of multivariate normal distribution to infinitely many random variables. The main strength of GP regression is its ability, using very few parameters, to capture complex unknown functions. Due to its simplicity and effectiveness, GP regression is gaining attention in the field of control, optimization and dynamical systems. For selected application examples, the reader is referred to [128] and references therein.

Since GPs are excellent candidates to be used as global surrogate models, it is natural to combine them with derivative-free trust-region methods. The idea, which dates back to Conn's book [102], was analyzed empirically in [129]. However, to the best of our knowledge, it is yet to be shown whether GP can be certified to be fully linear, which is key for guaranteeing global convergence of derivative-free trust-region methods.

Outline and contributions

Chapter 6 proposes to use machine learning via GP in the context of RTO. More specifically, process measurements are used to recursively estimate the plant-model mismatch via GP. Put differently, we propose a variant of MA, whereby the usual modifiers are replaced by high-order regression functions. Note that deep machine learning has been used in the context of MA before. In [130], a feedforward decision maker was designed to anticipate the effect of disturbances. The feedforward terms, which are constructed from historical data and deep machine learning, can improve the performance of the MA scheme by providing a better initial point for the iterative scheme. In contrast, this work proposes to use machine learning via GP to estimate the plant-model mismatch, that is, the complete modifier terms. Our first contribution includes: (i) a simple way of introducing GP regression in the MA framework, (ii) an illustration that high-order corrections can help reach plant optimality despite the presence of measurement noise.

In Chapter 7 we prove that GPs can satisfy the probabilistic fully-linearity property for a derivative-free trust region framework and present a necessary procedure for this certification. Furthermore, using a GP as a global surrogate model leads to fewer trust-region iterations, implying fewer plant evaluations. This has a clear advantage over local surrogate models and other empirical local model correction methods as illustrated in the numerical section.

In Chapter 8 by proving that a GP can also satisfy the deterministic fully-linearity

property, we extend the results. Next, we propose a novel derivative-free trust-region method to address limitations of the standard DFO based trust-region method. The first limitation of the standard DFO trust region methods is that they require to build a local surrogate model and check its accuracy multiple times in each iteration, which is not desirable due to the fact that it leads to additional function/plant evaluations to estimate gradients. Furthermore, the traditional DFO trust-region approaches require more accurate gradients with shrinking of the trust region radius, which is difficult to obtain in the presence of noise. In contrast to this, we will explicitly avoid building and performing multiple checks of accuracy of a surrogate model in each trust region iteration by defining suitable criteria for adaptive model improvement. Moreover, we assume the process-measurements are corrupted by noise. Thus, the proposed algorithm reaches to a neighborhood of an optimal solution comparatively in a smaller number of iterations. Finally, we present results of experiments on a Solid Oxide Fuel Cell (SOFC) system in Chapter 9.

Next we introduce Gaussian Processes, which are used as surrogate model in this work.

5.2 Using Gaussian Processes for Global Surrogate Modeling

Unlike parametric identification techniques, where data are discarded after constructing the model, GPs are kernel-based methods that use all available data (or a subset thereof) to learn a map between input and output data. We will briefly introduce GPs and refer to [125, 131] for further details.

Here, we are interested in approximating the unknown function $f : \mathbb{R}^n \to \mathbb{R}$ using a GP. In the RTO context, the function $f(\cdot)$ can represent the plant steady-state map, the constraints or the cost. In this study, given a set of plant inputs and outputs, we are interested in approximating the mismatch between the true plant map and its available model.

Consider the unknown function $f : \mathbb{R}^n \to \mathbb{R}$, $z = f(u) + \nu$, where $\nu \sim \mathcal{N}(0, \sigma^2)$. Using p available input-output pairs, the input-output data generated by $f(\cdot)$ are $\bar{U} = [u_1, u_2, \ldots, u_p] \in \mathbb{R}^{n \times p}$ and $\bar{z} = [z_1, z_2, \ldots, z_p]^T \in \mathbb{R}^{p \times 1}$. We use GP regression to establish a relationship between $\bar{U}$ and $\bar{z}$ and obtain a corresponding conditional

distribution of the output z for a new query input point u, that is,

$$z|\bar{U}, \bar{z}, u \sim \mathcal{N}\left(z_m(u), z_v(u)\right), \tag{5.2.1}$$

where the mean and the variance of z are:

$$z_m(u) = c^T \left(\bar{C} + \sigma^2 I\right)^{-1} \bar{z}, \tag{5.2.2}$$

$$z_v(u) = \kappa(u) - c^T \left(\bar{C} + \sigma^2 I\right)^{-1} c. \tag{5.2.3}$$

Here, $\bar{C} \in \mathbb{R}^{p\times p}$ is a covariance matrix with the elements $\bar{C}_{ij} = c(u_i, u_j)$, where $c(u) = [c(u, u_1), c(u, u_2), \ldots, c(u, u_p)]^T \in \mathbb{R}^{p\times 1}$, $\kappa(u) = c(u, u)$, where $c(\cdot, \cdot)$ is a covariance function labeled *kernel*. One example of a kernel is the auto relevance determination squared exponential covariance function defined as

$$c(u_i, u_j) = \sigma_f^2 \exp\left(-\frac{(u_i - u_j)^T \Lambda (u_i - u_j)}{2}\right), \tag{5.2.4}$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, The parameters $\theta = [\sigma_f, \lambda_{1:n}] \in \mathbb{R}^{n+1}$ are the hyperparameters that need to be learned/estimated from the data $\{\bar{U}, \bar{z}\}$ during the training phase. Since the covariance matrix $\bar{C}$ and the covariance vector c depend on the hyperparameters θ and the inputs $\bar{U}$, one can also write $\bar{C}$ as $\bar{C}(\theta, \bar{U})$. To this end, consider $M(\theta, \bar{U}) := \bar{C}(\theta, \bar{U}) + \sigma^2 I$ and the log-marginal likelihood

$$\mathcal{L}(\theta, \bar{U}, \bar{z}) = -\frac{1}{2}\bar{z}^T M(\theta, \bar{U})^{-1}\bar{z} - \frac{1}{2}\log|M(\theta, \bar{U})| - \frac{n}{2}\log 2\pi.$$

Given $\bar{U}$ and $\bar{z}$, the parameters are learned by maximizing the log-marginal likelihood,

$$\theta^* = \underset{\theta}{\text{argmax}}\ \mathcal{L}(\theta, \bar{U}, \bar{z}). \tag{5.2.5}$$

The above maximization problem is nonlinear and nonconvex in nature, for which deterministic as well as stochastic solution methods can be used. Thorough discussions on these methods can be found in Chapter 2.4.2 in [128] for details. Moreover, we refer to Chapter 7 in [125] and Chapter 2 in [128] for details regarding convergence properties and rate of convergence.

In summary, a GP provides z that has a normal distribution for a query point u. The mean and covariance function indicate how similar u is with the training data using

the hyperparmeters θ^*. The GP can be used to compute the output distribution of z with more weight on the nearest inputs. This way, the predicted output z is influenced more by the nearby input-output pairs obtained from the training data set. This flexibility is the main advantage of GP compared to fixed structure input-output relationship based on parametric methods. Hence, GP models are able to capture complex nonlinear input-output relationships through the use of only a few parameters.

A first advantage of using GPs, as discussed in the previous paragraph, is that GP models are able to capture complex nonlinear input-output relationships through the use of only a few parameters. A second advantage is that, generally, they offer an interesting trade-off between exploration and exploitation [125]. A third advantage, particularly in a DFO setting, is the fact that GP constitute global surrogate models. Finally, the mismatch between an unknown function and a GP can be bounded (in probabilistic as well as deterministic sense) in terms of the number of samples—a key instrumental property that we will rely on for proving global convergence of the derivative-free trust-region framework.

The main drawback of GP is that the computational complexity grows as a cubic function of the number of data points N, that is, the computational complexity is $\mathcal{O}(N^3)$ for computing $z_v(u)$ in (5.2.3). However, in the context of RTO of process systems, this is not a major challenge as most systems have a large settling time.

6 Gaussian Processes Combined with Modifier Adaptation for Real-time Optimization

An equation means nothing to me unless it expresses a thought of God.

Srinivasan Ramanujan

The optimization of the plant steady-state can be stated by the following NLP[1]:

$$\min_{u} \Phi_p(u) := \phi\Big(u, y_p(u)\Big) \tag{6.0.1a}$$

subject to

$$G_{p,i}(u) := g_i\Big(u, y_p(u)\Big) \leq 0 \quad i = 1, ..., n_g, \tag{6.0.1b}$$

$$u \in \mathcal{U}. \tag{6.0.1c}$$

Here $u \in \mathbb{R}^{n_u}$ are the decision (or input) variables; $y_p \in \mathbb{R}^{n_y}$ are the measured output variables; ϕ: $\mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R}$ is the cost function to be minimized; g_i: $\mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R}$, $i = 1, ..., n_g$, is the set of process-dependent inequality constraint functions; and $\mathcal{U}$ is typically determined by lower and upper bounds on the input variables, $\mathcal{U} = \{u \in \mathbb{R}^{n_u} : u^L \leq u \leq u^U\}$. The subscript $(\cdot)_p$ indicates a quantity related to the plant.

Usually, since the steady-state input-output map of the plant, $u \in \mathbb{R}^{n_u} \mapsto y_p \in \mathbb{R}^{n_y}$, is not precisely known, one relies on an approximation given by an available model. That is, instead of tackling Problem (6.0.1) directly, one solves the following

[1] The material of this chapter was written with Dr. Tafarel de Avila Ferreira during a collaborative work.

model-based optimization problem:

$$\min_u \Phi(u) := \phi\Big(u, y(u)\Big) \tag{6.0.2a}$$

subject to

$$G_i(u) := g_i\Big(u, y(u)\Big) \leq 0 \quad i = 1, ..., n_g, \tag{6.0.2b}$$

$$u \in \mathcal{U}\,. \tag{6.0.2c}$$

Due to plant-model mismatch and disturbances, the solutions to Problems (6.0.1) and (6.0.2) are usually different. RTO aims at reaching plant optimality by iteratively updating the model using plant measurements.

Remark 6.0.1 (RTO for dynamic processes)**.** *At first glance, the problem setting outline above might look restrictive as it focuses on optimizing steady-state performance. However, whenever one aims to optimize the performance of a repeated (batch) process or of a periodic process,*

$$\dot{x} = f_p(x, v), \quad x(0) = x_0$$

one will typically start with an optimal control problem, which after applying direct discretization techniques can be cast in a mathematically equivalent form to (6.0.1) *and* (6.0.2)*. In this process inputs $v(t)$ will usually be described by a finite number of parameters $p_1, \ldots, p_{n_p}$ using suitable basis functions ($v(t) = \sum_{i=1}^{n_p} p_i \mu_i(t)$). Now, consider some appropriate discretization of the ODE and set $u \leftarrow p_1, \ldots, p_{n_p}$, we see that also repeated/periodic dynamic problems can be cast in the setting sketched above. In other words, the problem setting sketched above is quite generic.* □

6.0.1 Modifier Adaptation

MA uses first-order corrections in order to match the necessary conditions of optimality of the plant upon convergence [109]. Input-affine terms are added to the cost and constraint functions of Problem (6.0.2). The optimal inputs are computed

by solving the following modified optimization problem [109]:

$$u_{k+1}^{\star} = \underset{u}{\operatorname{argmin}} \quad \Phi(u) + (\lambda_k^{\Phi})^{\intercal} u \tag{6.0.3a}$$

subject to

$$\mathsf{G}_i(u) + \varepsilon_{i,k} + (\lambda_k^{\mathsf{G}_i})^{\intercal}(u - u_k) \leq 0 \qquad i = 1, ..., n_g, \tag{6.0.3b}$$

$$u \in \mathcal{U}, \tag{6.0.3c}$$

with

$$\varepsilon_{i,k} = \mathsf{G}_{p,i}(u_k) - \mathsf{G}_i(u_k), \tag{6.0.3d}$$

$$(\lambda_k^{\Phi})^{\intercal} = \frac{\partial \Phi_p}{\partial u}(u_k) - \frac{\partial \Phi}{\partial u}(u_k), \tag{6.0.3e}$$

$$(\lambda_k^{\mathsf{G}_i})^{\intercal} = \frac{\partial \mathsf{G}_{p,i}}{\partial u}(u_k) - \frac{\partial \mathsf{G}_i}{\partial u}(u_k). \tag{6.0.3f}$$

The RTO iteration is denoted by the subscript $(\cdot)_k$. The zeroth-order term $\varepsilon_{i,k}$ represents the differences between the plant values and the predicted values of the constraints at u_k, while the first-order modifiers $\boldsymbol{\lambda}_k^{\Phi}$ and $\boldsymbol{\lambda}_k^{\mathsf{G}_i}$ correspond to the differences between the plant gradients and the gradients predicted by the model at u_k. The optimal input $u_{k+1}^{\star}$ may be filtered, as proposed by [109]:

$$u_{k+1} = u_k + K(u_{k+1}^{\star} - u_k), \tag{6.0.4}$$

where $K = \operatorname{diag}(k_1, ..., k_{n_u}) \in \mathbb{R}^{n_u}$, $k_i \in (0, 1]$, $i = 1, ..., n_u$.

The main advantage of modifier adaptation lies in its ability to reach a KKT point of Problem (6.0.1) upon convergence. However, the estimation of the plant gradients $\frac{\partial \Phi_p}{\partial u}(u_k)$ and $\frac{\partial \mathsf{G}_{p,i}}{\partial u}(u_k)$ at each RTO iteration is quite challenging.

6.1 Using Gaussian Processes for RTO

The main limitation of standard MA stems from the need to estimate plant gradients. Typically, finite-difference approximations are used to compute plant gradients, that is, the gradients are estimated based on additional plant runs with imposed perturbations. Note that this becomes impractical when the input dimension is large. Moreover, standard MA uses only first-order corrections to update the cost and constraint functions. Here, we propose an MA scheme that uses GP to overcome these difficulties.

The main idea consists in replacing the zeroth- and first-order corrections used in Problem (6.0.3) by a description of the plant-model mismatch obtained from GP regression. Subsequently, we will rely on recursively adapted GP. Hence, for the sake of compact notation, given some unknown function f, we write

$$z_m = (\mathcal{GP})^f \left(u, \bar{U}, \bar{z}\right)$$

to denote the GP mean z_m (see equation 5.2.2) obtained for the input u using a GP regression of f based on solving equation 5.2.5 subject to the input-output data $\bar{U} \in \mathbb{R}^{n \times p}$ and $\bar{z} \in \mathbb{R}^{p \times 1}$. Since the output data $\bar{z}$ is an implicit function of the input data $\bar{U}$, that is $\bar{z}(\bar{U})$, one can write $z_m = (\mathcal{GP})^f \left(u, \bar{U}, \bar{z}(\bar{U})\right)$. For simplicity of notation, we can drop the set of output data $\bar{z}$ as an argument of $(\mathcal{GP})^f$ and simply write:

$$z_m = (\mathcal{GP})^f \left(u, \bar{U}\right). \tag{6.1.1}$$

While the evaluation $z_m = (\mathcal{GP})^f \left(u, \bar{U}\right)$ is computationally cheap for fixed $\bar{U}$ and $\bar{z}$, one has to repeatedly solve Problem (5.2.5) when the data $\bar{U}$ and $\bar{z}$ change through acquisition of more data.

To this end, and considering equation 6.1.1, we use

$$(\mathcal{GP})^{(j_p - j)} \left(u, \bar{U}\right), \quad j \in \{\Phi, G_1, \ldots, G_{n_g}\}, \tag{6.1.2}$$

to denote the GP approximation to the plant-model mismatch of the cost and constraints, that is, the approximation to $\Phi_p - \Phi$ and $G_{p,i} - G_i$, $i = 1, ..., n_g$, based on the data $\bar{U}$ and $\bar{\mathbf{z}}$ (whereby the later argument is again dropped for simplicity). The superscript $(j_p - j)$ serves to identify the unknown function.

6.1.1 Proposed RTO Scheme

We suggest an RTO scheme based on solving the following NLP:

$$u^\star_{k+1} = \underset{u}{\text{argmin}} \;\; \Phi(u) + (\mathcal{GP})_k^{(\Phi_p - \Phi)}(u, \bar{U}_k) \tag{6.1.3a}$$

subject to

$$G_i(u) + (\mathcal{GP})_{i,k}^{(G_p - G)}(u, \bar{U}_k) \leq 0, \quad i = 1, ..., n_g, \tag{6.1.3b}$$

$$u \in \mathcal{U}, \tag{6.1.3c}$$

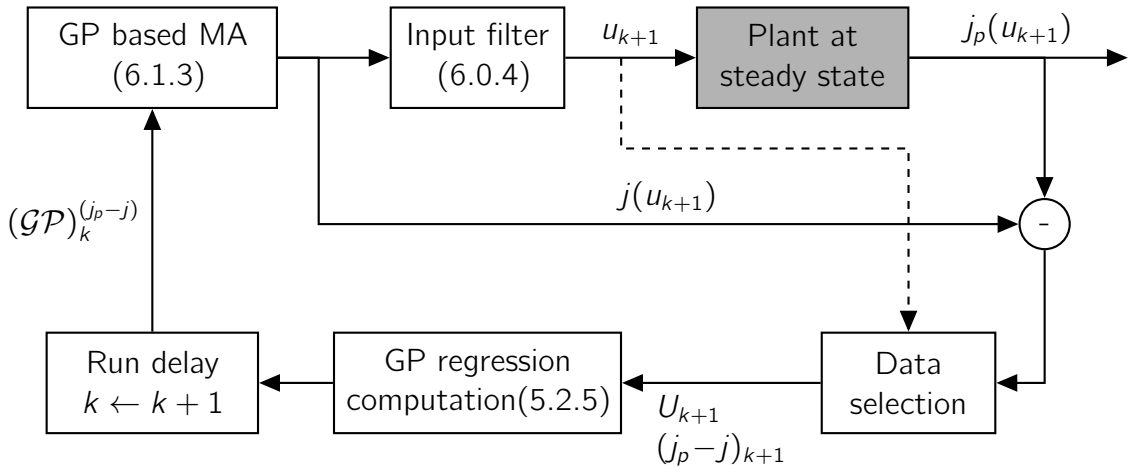


Figure 6.1 – GP-based MA at the k^{th} iteration to estimate the plant-model mismatch, with $j \in \{\Phi, G_1, \ldots, G_{n_g}\}$.

where the difference between the plant and the model of the cost and constraint functions are modeled by $(\mathcal{GP})_k^{(\Phi_p-\Phi)} \in \mathbb{R}$ and $(\mathcal{GP})_{i,k}^{(G_p-G)} \in \mathbb{R}$, $i = 1, ..., n_g$, respectively. The superscript $(\Phi_p - \Phi)$ indicates that the difference is between the plant and the model of the cost functions. Similarly, the superscript $(G_p - G)$ points to the difference between the plant and the model of the constraint functions. $\bar{U}_k$ is the available input set at the k^{th} iteration.

We may also filter the optimal input $u_{k+1}^\star$ as in Eq. (6.0.4). Fig. 6.1 depicts the MA scheme that uses GP to estimate the plant-model mismatch.

It is important to note that, in contrast to standard MA that uses zeroth- and first-order correction terms to update the optimization problem, the optimization problem is modified here by adding GP regression functions to the cost and constraint functions such that the cost and constraint functions of the modified optimization Problem (6.1.3) locally match those of the plant. Furthermore, we use a smooth squared exponential kernel so that, if Problem (6.0.2) admits an optimal solution, so does Problem (6.1.3). Hence, upon convergence, if the GP regression functions locally approximate the plant-model mismatch well, Problem (6.1.3) will converge to a KKT point of the plant. In addition, in order to avoid overfitting, we take into account a finite number of points within a certain radius from the current operating point (see the data selection block in Fig. 6.1). This way, the range of dimension of the covariance matrix used for building the GP functions does not grow with the

size of the data and is comparatively very small as we shall see in the next section. Note that the marginal likelihood optimization problems (5.2.5) corresponding to each GP regression can be computed in parallel.

6.2 Case Study: Williams-Otto Reactor

In this section, we apply both standard MA and GP based MA to the Williams-Otto reactor [132]. This reactor is an ideal continuous stirred-tank reactor with the three reactions:

$$\mathrm{A} + \mathrm{B} \xrightarrow{k_1} \mathrm{C}, \tag{6.2.1}$$

$$\mathrm{C} + \mathrm{B} \xrightarrow{k_2} \mathrm{P} + \mathrm{E}, \tag{6.2.2}$$

$$\mathrm{P} + \mathrm{C} \xrightarrow{k_3} \mathrm{G}. \tag{6.2.3}$$

The reactants A and B are fed with the mass flowrates F_A and F_B, respectively. The desired products are P and E, whereas G is an undesired byproduct. The intermediate product C is also produced. Since it is assumed that the reaction scheme is not well understood, the following two reactions have been proposed for the model [133]:

$$\mathrm{A} + 2\mathrm{B} \xrightarrow{k_1^*} \mathrm{P} + \mathrm{E}, \tag{6.2.4}$$

$$\mathrm{A} + \mathrm{B} + \mathrm{P} \xrightarrow{k_2^*} \mathrm{P} + \mathrm{E}. \tag{6.2.5}$$

The material balance equations for the plant and the model are given in [134]. The objective consists in maximizing the steady-state profit, while considering constraints on the concentrations of the reactant A and of the byproduct G [135]. The optimization problem is expressed mathematically as follows:

$$\begin{aligned} \max_{F_B, T_R} \quad & J = P_P X_P F + P_E X_E F - P_A F_A - P_B F_B + 200, \\ \text{s.t.} \quad & \mathrm{g}_1 = X_A - 0.12 \leq 0, \\ & \mathrm{g}_2 = X_G - 0.08 \leq 0, \\ & F_B \in [4, 7], \\ & T_R \in [70, 100], \end{aligned} \tag{6.2.6}$$

Table 6.1 – Species prices for different scenarios.

Prices	P_P	P_E	P_A	P_B
Scenario I	1043.38	20.92	79.23	118.34
Scenario II	1073.25	25.92	94.18	95

where F is the sum of the reactant mass flowrates, $F = F_A + F_B$. X_i is the concentrations of Species i. F_B and T_R are the decision variables. The feed flowrate of Component A is kept constant at $F_A = 1.82$ kg/s.

We solve the aforementioned optimization problem for the two price scenarios given in Table 6.1. In Scenario I, both composition constraints are active at the plant optimum, whereas in Scenario II only the constraint on composition G is active at the plant optimum. We start with Scenario I and switch to Scenario II after 55 iterations. Two optimization schemes are compared :

- *Standard MA* as per Eq. (6.0.3). We consider 5 initial operating points that are used to estimate the initial values of the first-order modifiers via linear interpolation. The plant and model gradients are estimated via forward finite differences and used in Eqs. (6.0.3e) and (6.0.3f).
- *GP based MA* as per Eq. (6.1.3). We consider the same initial operating points as with standard MA. These points are used to find the hyperparameters θ^* of the GP regression. At each RTO iteration, the newly available data are used to update the mean, the covariance, and the hyperparameters.

We assume that the plant cost Φ_p and the concentrations X_A and X_G are subject to noise with zero mean and standard deviations $\sigma_\Phi = 0.5$ and $\sigma_{X_A} = \sigma_{X_G} = 0.0005$ as proposed by [109] and [135]. We choose the rather low filter gain of 0.4 for all diagonal matrix elements so as to easily enforce convergence. In GP based MA, in order to avoid overfitting, we reject the data to compute the GP regression functions if more than 10 points lie in a radius of 1 kg/s for F_B and 10°C for T_R. Hence, the dimension of the covariance matrix computed ranges from 20×20 to 25×25. Throughout this chapter, we use ARD squared exponential kernel and GPML toolbox [125].

Starting the RTO from the initial conservative feasible point $u_0 = [6.9, 86]^\intercal$, simulations are performed for 120 iterations. Figs. 6.2 and 6.3 show the performance of standard MA, while Figs. 6.4 to 6.5 show that of GP based MA. The first 5 points

in light blue correspond to the 5 initial operating points. The remaining points in dark blue are the plant evaluations needed for both standard MA and GP based MA. The dashed green line represents the plant optimal values for Scenarios I and II.

Upon comparing Figs. 6.2 to 6.5, for the same level of noise, one can see that standard MA oscillates around the optimum, whereas GP based MA converges to the plant optimum of Scenario I in 7 iterations, and in about 4 iterations from Scenario I to Scenario II. The lower plot of Fig. 6.3 shows that the constrained concentration X_G is violated several times. Similarly, the cost function and inputs are significantly less noisy for GP based MA than for standard MA. This behaviour can be explained by the fact that GP based MA deals with noisy measurements better than standard MA. In standard MA, the noise is handled by the choice of the step length perturbation for the finite-difference approach, whereas in GP based MA the noise is absorbed by the GP regression computed at each RTO iteration.

Figure 6.6 compares the performance of standard MA and GP based MA in the input space for both Scenarios I and II. The pink dot corresponds to the plant optimum for Scenario I, whereas the green dot indicates the plant optimum for Scenario II. The solid red lines represent the constraints $X_A = 0.12$ and $X_G = 0.08$. The dashed red lines are the contour lines of the plant profit for Scenario I, while the black lines represent the plant profit for Scenario II. Although it takes about 4 to 7 iterations to reach the plant optimum with standard MA for both scenarios, the constrained concentration X_G is constantly violated because of noisy measurements. In contrast, GP based MA takes about 3 to 5 iterations to reach the plant optimum, with the constraints being hardly violated. Furthermore, the inputs are significantly less noisy with GP based MA than with standard MA.

6.3 Conclusion

This chapter proposed a RTO scheme that combines MA and machine learning via GP. The approach, which estimates the plant-model mismatch using GP regression functions, has been illustrated by means of the real-time optimization of the Williams-Otto reactor. Simulations have shown that the proposed approach performs well despite the fair amount of noise added to the measurements. A comparison between standard MA and GP based MA indicated that the latter clearly outperforms the former in terms of noise attenuation.

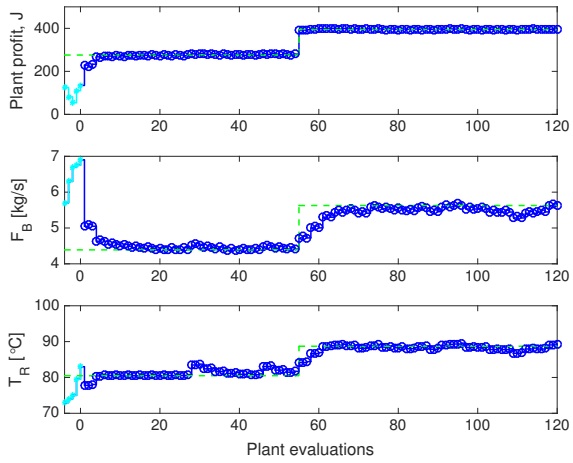


Figure 6.2 – Standard MA applied to the Williams-Otto reactor. Evolution of the cost and of the inputs F_B and T_R for the plant. Dashed green line: plant optimal values. Light blue line: initial operating points. Blue line: evolution of the cost and inputs.

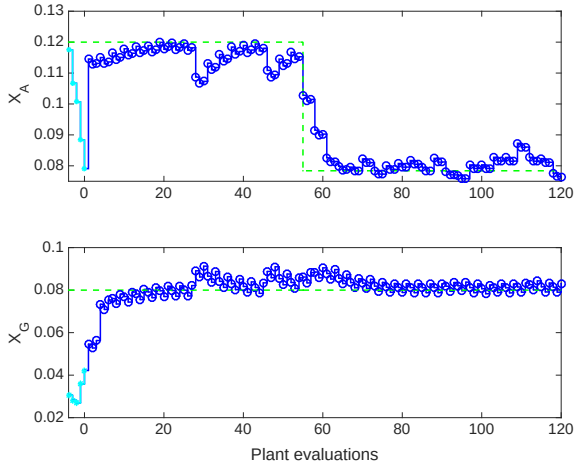


Figure 6.3 – Standard MA. Evolution of the constrained concentrations X_A and X_G for the plant. Dashed green line: plant optimal values. Light blue line: initial operating points. Blue line: evolution of the concentrations.

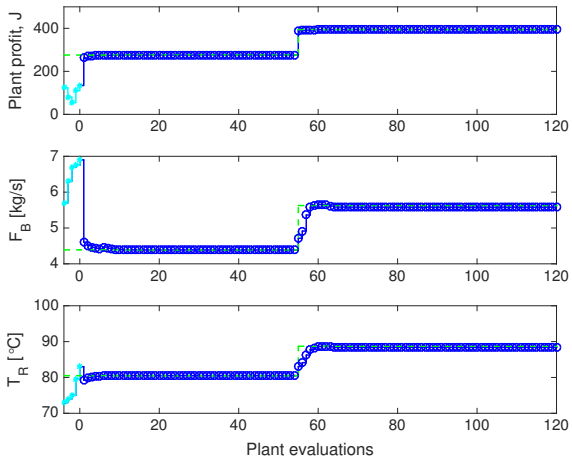


Figure 6.4 – GP based MA applied to the Williams-Otto reactor. Evolution of the cost and of the inputs F_B and T_R for the plant. Dashed green line: plant optimal values. Light blue line: initial operating points. Blue line: evolution of the cost and inputs.

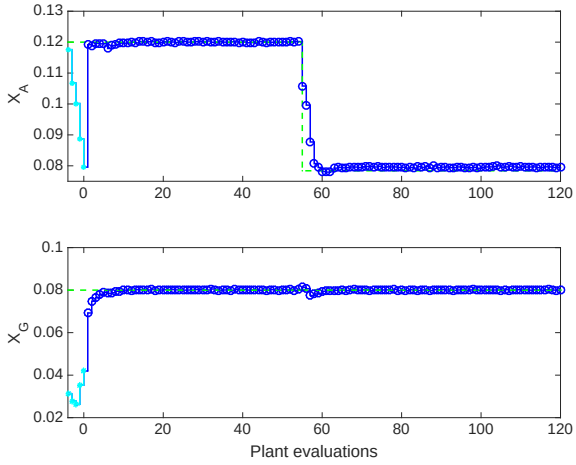


Figure 6.5 – GP based MA. Evolution of the constrained concentrations X_A and X_G for the plant. Dashed green line: plant optimal values. Light blue line: initial operating points. Blue line: evolution of the concentrations.

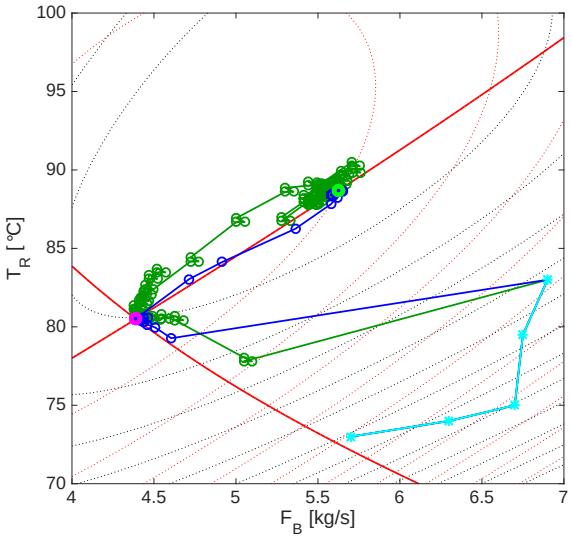


Figure 6.6 – Comparison of input trajectories with standard MA and GP based MA. Pink dot: plant optimum of Scenario I. Light green dot: plant optimum of Scenario II. Blue line: evolution of standard MA. Green line: evolution of GP based MA. Red line: constrained composition bounds. Light blue line: initial operating points.

7 Probabilistic Derivative-Free Trust Method using Gaussian Processes

> Information: the negative reciprocal value of probability.
>
> Calude Shannon

In the previous chapter, we proposed a variant of Modifier Adaptation (MA) and showed its superiority over a traditional MA scheme. However, our proposed approach was heuristic. In this chapter we show that our proposed heuristic variant combined with a probabilistic derivative-free trust region method has a global convergence property. To that end, we show Gaussian processes (GPs) provide a property of probabilistic fully-linear models.

7.1 From Real-Time Optimization to Derivative Free Optimization

In the previous chapter we showed that while Gaussian Processes (GPs) inspired modifier adaptation is more robust to noise and requires fewer iterations for convergence, our proposed method was heuristic. In this chapter, we prove global convergence of GP based surrogate models using a Derivative-Free trust-region framework.

To the end of leveraging Derivative-Free Optimization (DFO) based trust region methods, we convert the constrained optimization problems (6.0.1) and (6.0.2) into unconstrained ones using penalty functions; see [55] for a general discussion

and [136] for Real-Time Optimization (RTO) applications. Consider

$$f(u) = \Phi_p(u) + \sum_{i=1}^{n_g} \psi_i(G_i(u)) + \psi_{\mathcal{U}}(u),$$

where ψ_j are appropriately chosen penalty functions. To account for Remark 6.0.1 in our later developments, we will henceforth adapt the notation and consider the decision variable $x \in \mathbb{R}^{n_x}$ instead of $u \in \mathbb{R}^{n_u}$. Hence, rewriting f from above in terms of $x \in \mathbb{R}^{n_x}$, and without significant loss of generality, we replace (6.0.1) by

$$\min_{x \in \mathbb{R}^{n_x}} f(x), \tag{7.1.1}$$

with the unknown objective $f : \mathbb{R}^{n_x} \to \mathbb{R}$ and the decision variables $x \in \mathbb{R}^{n_x}$. A solution to (7.1.1) can be computed using DFO by sampling the unknown function f and building a surrogate model. The samples are subject to additive noise and therefore their distribution can be written as:

$$\mathsf{z} = f(x) + \nu \quad \text{where } \nu \sim \mathcal{N}\left(0, \sigma^2\right). \tag{7.1.2}$$

Surrogate models usually depend (implicitly or explicitly) on a—yet to be specified—number of past data points,

$$\mathbb{D}_k = \{(x_{k-l-1}, z_{k-l-1}), \ldots, (x_k, z_k)\}, \tag{7.1.3}$$

where $z_{(k)}$ is a realization of the random variable z at time instant k. Hence, by building the surrogate model $m : \mathbb{R}^{n_x} \times \mathbb{R}^{(n_x+1)\times l} \to \mathbb{R}$, the solution to problem (7.1.1) becomes

$$x_{k+1} = \arg\min_{x \in \mathbb{R}^{n_x}} m_k(x), \tag{7.1.4}$$

where the shorthand $m_k(x) := m(x, \mathbb{D}_k)$ is used.

Let $\mathsf{x} \in L^2(\mathcal{D}, \mathbb{P}; \mathbb{R}^{n_x})$ denote random variables and $x := \mathsf{x}(\omega) \in \mathbb{R}^{n_x}$ their realizations, where $L^2(\mathcal{D}, \mathbb{P}; \mathbb{R}^{n_x})$ is the underlying Hilbert space of random variables with finite variance.

In what follows, at each iteration k, we use the GP mean as a surrogate model, that is,

$$m_k(x) := z_m\left(x, \mathbb{D}_k\right), \tag{7.1.5}$$

where the notation $z_m\left(x, \mathbb{D}_k\right)$ highlights that, for fixed hyperparameters, z_m from

(5.2.2) takes x as argument—via $\bar{c}$—and depends on the data set $\mathbb{D}_k$–via $\bar{c}$ and $\bar{C}$.

If the considered function samples obtained via (7.1.2) are indeed subject to additive noise, the data $\mathbb{D}_k$ will contain l samples that correspond to realizations of random variables. Hence, the uncertainty surrounding the data $\mathbb{D}_k$ induces the probabilistic nature of the surrogate model and, consequently, the model available at iteration k can be regarded as $\mathsf{m}_k : \mathbb{R}^{n_x} \to L^2(\mathcal{D}, \mathbb{P}; \mathbb{R})$. Conceptually, its realization can be denoted as $m_k := \mathsf{m}_k(\omega)$. This point of view leads naturally to probabilistic DFO methods.

7.2 Derivative-Free Probabilistic Trust-Region Methods

A standard version of probabilistic derivative-free trust-region method is summarized in Algorithm 7.2.1, cf. [120, 121]. The main idea is to approximate the unknown function via $m_k(x)$ within a certain neighborhood of x_k (a.k.a. the trust region). Whenever the surrogate model fails approximating the original problem, then the trust region is shrunk and the process repeated. Next, we recall the main points of the convergence analysis given in [121].

Assumption 7.2.1 (Differentiability of f [121])**.** *The unknown function f has bounded level sets and the gradient ∇f is Lipschitz continuous with constant L_g.* □

Assumption 7.2.2 (Noise with finite variance [121])**.** *The additive noise ν observed while measuring f is drawn from a normal distribution with zero mean and finite variance.* □

For the remainder, we define $B(x; \Delta)$ as the ball of radius Δ centered at $x \in \mathbb{R}^n$. Furthermore, $\mathcal{C}^k$ denotes the set of functions on $\mathbb{R}^n$ with k continuous derivatives and $\mathcal{LC}^k$ denotes the set of functions in $\mathcal{C}^k$ such that the kth derivative is Lipschitz continuous.

Definition 7.2.1 (κ fully-linear model [121])**.** *Consider f satisfying Assumption 7.2.1. Let $\kappa = (\kappa_{ef}, \kappa_{eg}, \nu_1^m)$ be a given vector of constants and let $\bar{\Delta} > 0$ be given. A model $m \in \mathcal{LC}^1$ with Lipschitz constant ν_1^m is a κ fully-linear model of f on $B(x; \Delta)$ if for all $\Delta < \bar{\Delta}$ and $s \in B(0; \Delta)$,*

$$|f(x+s) - m(x+s)| \leq \kappa_{ef}\Delta^2, \text{ and} \tag{7.2.1a}$$

$$\|\nabla f(x+s) - \nabla m(x+s)\| \leq \kappa_{eg}\Delta. \tag{7.2.1b}$$

□

The above definition is key in the convergence analysis for the case of probabilistic surrogate models. The main idea is to show that these models have good accuracy with sufficiently high probability [120]. Since derivative-free trust-region algorithms sample and collect data at each iteration, let F_{k-1}^{M} denote the realization of events during the first $k-1$ iterations of the algorithm. Now, we are ready to define a probabilistic κ fully-linear surrogate model.

Definition 7.2.2 (κ fully-linear model with probability α [121])**.** *A sequence of random models* $\{\mathsf{m}_k\}$ *is* κ *fully linear with probability* α *on* $\{B(x_k, \Delta_k)\}$ *if the events*

$$S_k = \{m_k \text{ is a } \kappa \text{ fully-linear model of } f \text{ on } B(x_k, \Delta_k)\}$$

satisfy the condition $\mathbb{P}\left(S_k | F_{k-1}^{M}\right) \geq \alpha$ *for all* k *sufficiently large.* □

Next, we introduce Algorithm 7.2.1. The main idea is to build a surrogate model within the trust-region radius and use it to compute a minimizer. As long as the objective decreases sufficiently, accept the step and increase the trust-region radius, otherwise decrease the radius and reject the step. The challenge stems from the probabilistic nature of the surrogate model, in particular from the fact that the confidence in the model is probabilistic. This hinders increasing the trust-region radius significantly. Hence, it is important to have a relationship between the probability α (confidence in the surrogate model) and $\gamma_{inc}/\gamma_{dec}$ (increment/decrement of the radius). This relationship reads [121]:

$$\alpha \geq \max \left\{ \frac{1}{2}, 1 - \frac{\frac{\gamma_{inc}-1}{\gamma_{inc}}}{4\left[\frac{\gamma_{inc}-1}{2\gamma_{inc}} + \frac{1-\gamma_{dec}}{\gamma_{dec}}\right]}, 1 - \frac{1-\gamma_{dec}}{2\left(\gamma_{inc}^2 - \gamma_{dec}\right)} \right\}. \tag{7.2.2}$$

Remark 7.2.1. *A careful look at Step 1 of Algorithm 7.2.1 reveals that we need to build a* κ *fully-linear model only for sufficiently large* k*. This allows having a relatively inaccurate model at the beginning, thereby avoiding unnecessary sampling as long as there is sufficient improvement.*

Theorem 7.2.1 (Global convergence [121])**.** *If Assumptions 7.2.1-7.2.2 are satisfied, and* α *is chosen to satisfy* (7.2.2)*, then* $\{\|\nabla f(x_k)\|\}$ *converges in probability to zero. That is, for all* $\epsilon > 0$*,* $\lim_{k\to\infty} \mathbb{P}\left[\|\nabla f(x_k)\| > \epsilon\right] = 0$*.* □

Algorithm 7.2.1 Derivative-Free Trust-Region Method [121]

Data: Initial model m_0, initial point x_0, and constants $0 < \gamma_{dec} < 1 < \gamma_{inc}$, $0 < \eta < \beta < 1$, $0 < \Delta_0$
and $\alpha \in (0, 1)$ satisfying (7.2.2). Set k =0.

1. **Model building:** Build m_k, a κ fully-linear model with probability α_k on $B(x_k; \Delta_k)$, for some $\alpha_k \in (0, 1)$ such that $\alpha_k \geq \alpha$ for sufficiently large k.

2. **Step calculation:**

$$s_k := \underset{s:\|s\| \leq \Delta_k}{\arg\min}\, m_k(x_k + s) \tag{7.2.3}$$

3. **Compute model decrement:**

   (a) If $m_k(x_k) - m_k(x_k + s_k) < \beta \min\{\Delta_k, \Delta_k^2\}$ then $x_{k+1} = x_k$; $\Delta_{k+1} = \gamma_{dec}\Delta_k$ and go to Step 6.

   (b) Else go to Step 4).

4. **Estimate improvement after plant evaluation:** Evaluate

$$\rho_k = \frac{F_k^0 - F_k^{s_k}}{m_k(x_k) - m_k(x_k + s_k)}. \tag{7.2.4}$$

5. **Trust region and step update:**

   - If $\rho_k \geq \eta$, then $x_{k+1} = x_k + s_k$ and $\Delta_{k+1} = \gamma_{inc}\Delta_k$.
   - If $\rho_k < \eta$, then $x_{k+1} = x_k$ and $\Delta_{k+1} = \gamma_{dec}\Delta_k$.

6. **Setting index:** k = k+1 and go to Step 1.

At this point a pivotal question arises: how to build a κ fully-linear surrogate model with probability α? Details of building and certifying a probabilistic local surrogate model at each iteration—mainly via linear and nonlinear interpolation/regression—are given in [120, 121]. Here, we aim at reducing the number of expensive plant evaluations by constructing a global instead of a local surrogate model. For that, we will use a GP as the surrogate model. We will also show how to certify a GP as a probabilistic fully-linear model. This is still an open question, although GP have been used in a derivative-free trust-region framework [129, 137].

7.3 Certifying Gaussian Processes as Probabilistic-Fully Linear Model

We certify that GPs are probabilistic fully-linear models. We remind the reader that we use the GP mean as the surrogate model, that is, $m(x) := z_m(x)$.

Definition 7.3.1 (Reproducing kernel Hilbert Space [125]). *Let $\mathcal{H}$ be the Hilbert space of real functions f defined on the index set X. Then, $\mathcal{H}$ is called a reproducing kernel Hilbert space (*RKHS*) endowed with an inner product $\langle \cdot, \cdot \rangle$ (and norm $\|f\|_{\mathcal{H}} = \sqrt{\langle f, f \rangle_{\mathcal{H}}}$) if there exists a function $c : X \times X \to \mathbb{R}$ with the following properties:*

1. *for every x, $c(x, x')$ as a function of x' belongs to $\mathcal{H}$, and*
2. *c has the reproducing property $\langle f(\cdot), c(\cdot, x) \rangle_{\mathcal{H}} = f(x)$.*

□

The aim is to show that (7.2.1a) and (7.2.1b) hold with probability at least α when the GP mean is used as a surrogate model. For this, the following two properties are assumed about the mismatch function $h(x) := f(x) - m(x)$.

Assumption 7.3.1 (Bounded RKHS norm [138]).
The unknown function $f(x)$ has a known bounded RKHS *norm ζ under a known kernel c, that is, $\|f\|_c \leq \zeta < \infty$.* □

Assumption 7.3.2 (Lipschitzness of the mismatch function). *The mismatch function $h(x) := f(x) - m(x)$ has Lipschitz continuous gradient with constant γ_{lh}. Furthermore, the sequence x_k generated by applying Algorithm 7.2.1 satisfies $\|\nabla^2 h(x_k)\| \leq \kappa_{bhh} < \infty$, that is, the mismatch function has a bounded Hessian.* □

Assumption 7.3.2 is not very strong and is a consequence of Assumption 7.2.1: the unknown function $f(\cdot)$ has Lipschitz continuous gradient with bounded Hessian.

Note that most of the practically used kernels (e.g. Matern, squared exponential) have Lipschitz continuous gradients [125]. Before deriving the main result, we first state that the distance between an unknown function and the mean is bounded by the GP variance with some probability $1-\delta$.

Lemma 7.3.1 (Probabilistic bound on a mismatch function [138])**.** *Let Assumption 7.3.1 holds, $\delta \in (0,1)$, and assume measurements are corrupted by uniformly bounded noise. It follows that $\mathbb{P}\left\{|m(x)-f(x)| \leq \sqrt{\beta(N,\delta)}\,\sigma_z(x,N)\right\} \geq 1-\delta$.* □

Here, $\sqrt{\beta(N,\delta)}$ depends on the number of samples N, the probability δ and the RKHS norm $\|f\|_c$, see [138] for details. If the unknown function f is sampled from a GP, one can compute β in closed form [139]. We note that for highlighting the dependence of β and σ_z on the number of samples and the probability δ, we simply write them as $\beta(N,\delta)$ and $\sigma_z(x,N)$.

Assumption 7.3.3 (Decrease of $\sqrt{\beta(N,\delta)}\,\sigma_z(x,N)$)**.** *For a given value of $\delta \in (0,1)$ and $N \in \mathbb{N}$, it holds that $\lim_{N\to\infty} \sqrt{\beta(N,\delta)}\,\sigma_z(x,N) = 0$.* □

Remark 7.3.1 (Complexity analysis for $\sigma_z(x,N)$)**.** *An analysis of how fast $\sigma_z(x,N)$ should decrease when f is sampled from GP is given in [139]. The same reference also provides a complexity analysis when f belongs to RKHS and squared-exponential kernels are used.* □

Theorem 7.3.1 (GP is κ fully linear with probability α)**.** *Let Assumptions 7.3.1-7.3.3 hold. If $0 < \Delta < \frac{6}{\gamma_{lh}}(\kappa_{eg} - 2\kappa_{ef} - \kappa_{bhh})$, then there exists a positive integer $N < \infty$ such that, after N sampling steps, a* GP *can be certified κ, fully linear with probability α.*

Proof. Following Definition 7.2.2, the goal is to prove that equations (7.2.1a) and (7.2.1b) hold with probability at least α.

Let us start with equation (7.2.1a) and consider any point within the trust region, that is, $x \in B(x_k, \Delta_k)$. Increased sampling will validate the probability bound in Lemma 7.3.1. Upon performing N plant evaluations and applying Algorithm 1 in [138] with $\alpha \leq 1-\delta$, the following holds with probability α for a given κ_{ef} and Δ:

$$|h(x)| = |m(x) - f(x)| \leq \sqrt{\beta(N,\delta)}\,\sigma_z(x,N) \leq \kappa_{ef}\Delta^2, \tag{7.3.1}$$

which certifies equation (7.2.1a) with probability α.

Next we turn to equation (7.2.1b) and take any $x, x_s \in B(x_k, \Delta_k)$ such that $x_s = x + s$. Taylor's expansions give:

$$\begin{aligned} h(x+s) =& h(x) + s^\top \nabla h(x) + s^\top \nabla^2 h(x) s + \mathcal{O}(s^3) \\ |s^\top \nabla h(x)| =& |h(x+s) - h(x) - s^\top \nabla^2 h(x) s - \mathcal{O}(s^3)| \\ \leq& |h(x+s)| + |h(x)| + |s^\top \nabla^2 h(x) s| + |\mathcal{O}(s^3)| \\ \leq& |h(x+s)| + |h(x)| + |s^\top \nabla^2 h(x) s| + \frac{\gamma_{lh}}{6}\|s\|^3 \ , \end{aligned}$$

where the first inequality comes from norm properties and the second using Lemma 4.1.14 in [140]. Substituting $s := \frac{\nabla h(x)\Delta}{\|\nabla h(x)\|}$ by following Lemma 4.7 in [141] gives,

$$\begin{aligned} \Delta\|\nabla h(x)\| \leq& |h(x+s)| + |h(x)| + \Delta^2\|\nabla^2 h(x)\| + \frac{\gamma_{lh}}{6}\Delta^3 \\ \Delta\|\nabla h(x)\| \leq& |h(x+s)| + |h(x)| + \kappa_{bhh}\Delta^2 + \frac{\gamma_{lh}}{6}\Delta^3. \end{aligned}$$

Here, the last inequality arises because of Assumption 7.3.2. As shown in the first part of this proof, one can guarantee that $|h(x+s)|, |h(x)| \leq \kappa_{ef}\Delta^2$ with at least probability $1-\delta$. Hence, the following holds with with probability at least $(1-\delta)^2$:

$$\begin{aligned} \Delta\|\nabla h(x)\| \leq& 2\kappa_{ef}\Delta^2 + \Delta^2\kappa_{bhh} + \frac{\gamma_{lh}}{6}\Delta^3 \\ \|\nabla h(x)\| \leq& 2\kappa_{ef}\Delta + \Delta\kappa_{bhh} + \frac{\gamma_{lh}}{6}\Delta^2. \end{aligned}$$

Choosing δ such that $\alpha \leq (1-\delta)^2$ and combining the above with Definition 7.2.2 and (7.2.1b), it remains to show that, for a given κ_{eg}, the following criterion can be satisfied:

$$2\kappa_{ef}\Delta + \Delta\kappa_{bhh} + \frac{\gamma_{lh}}{6}\Delta^2 \leq \kappa_{eg}\Delta.$$

Since $0 < \Delta < \frac{6}{\gamma_{lh}}(\kappa_{eg} - 2\kappa_{ef} - \kappa_{bhh})$, the above inequality is satisfied. Hence, equation (7.2.1b) holds with probability at least α, which concludes the proof. □

Remark 7.3.2 (Computing β and finding maximum $\sigma_z(x)$)**.** *$\sqrt{\beta(N,\delta)}$ is not a function of x. However, we need to determine the maximum of $\sigma_z(x, N)$ over x within the trust region. This problem has been tackled rigorously in the machine learning community, see [138] for details. However, for our application, one need not explicitly compute these quantities. Another way to look at it is that one can always choose arbitrarily large κ_{ef} and κ_{eg} such that* (7.2.1a) *and* (7.2.1b) *are*

satisfied with probability at least α. □

Remark 7.3.3 (Condition on Δ in Theorem 7.3.1)**.**
The condition on the trust-region radius Δ in Theorem 7.3.1 does not limit/restrict the algorithm significantly. The reason is that one can choose arbitrarily large values of κ. Moreover, the trust-region radius almost surely goes to zero [Lemma 4 [121]]. Hence, for any positive κ, the condition on the trust region is almost surely satisfied. □

Using GP in the framework of Algorithm 7.2.1 yields almost sure convergence. Moreover, it has two main advantages: (i) since GP approximates unknown functions globally, one does not need to sample after each trust-region iterations as opposed to standard trust-region approaches, where n and $\frac{(n+1)^2}{2}$ data points are required for linear interpolation and nonlinear polynomial-based regression, respectively. This saves a significant number of plant evaluations; (ii) from an implementation point of view, there is no need to build a model at each trust-region iteration. This is due to the fact that the GP mean converges to the exact function in the limit, as per Lemma 7.3.1 and for $\alpha_k > \alpha$ for sufficiently large k. In fact, to implement the algorithm after a failed iteration, one simply needs to sample (not necessarily in the trust-region radius) a few points. Hence, the computation of $\beta(N, \delta)$ and of the maximal variance $\sigma_z(x, N)$ mentioned in the proof of Theorem 7.3.1 can be avoided. This obviously reduces the computational burden.

7.4 Case study: Diketene-Pyrrole Reactor

We consider the run-to-run optimization of a semi-batch reactor with 4 reactions: $A + B \xrightarrow{k_1} C$, $2B \xrightarrow{k_2} D$, $B \xrightarrow{k_3} E$, and $B + C \xrightarrow{k_4} F$. The involved species are A: pyrrole, B: diketene, C: 2-acetoacetyl pyyrole, D, dehyroacetic acid, E: oligomers, F: undesired by-product. The material balance equations for the plant read [142]:

$$\begin{aligned}
\dot{c}_A &= -k_1 c_A c_B - \frac{F}{V} c_A, \\
\dot{c}_B &= -k_1 c_A c_B - 2k_2 c_B^2 - k_3 c_B - k_4 c_B c_c + \frac{F}{V}\left(c_B^{in} - c_B\right), \\
\dot{c}_C &= \quad k_1 c_A c_B - k_4 c_B c_C - \frac{F}{V} c_C, \\
\dot{c}_D &= \quad k_2 c_B^2 - \frac{F}{V} c_D, \\
\dot{V} &= F.
\end{aligned} \tag{7.4.1}$$

Parameter values and initial conditions are given in [142]. To mimic plant-model mismatch, it is assumed that the last two reactions are not known to exist, thereby leading to the following reaction model: $\mathrm{A} + \mathrm{B} \xrightarrow{k_1} \mathrm{C}$, $2\mathrm{B} \xrightarrow{k_2} D$.

We are interested in computing the feed profile of species B so as to maximize the amount of the desired product C at final time, while maintaining the final concentrations of B and D below specified values. The dynamic optimization problem can be written mathematically as:

$$\begin{aligned} \max_{F(t)} \; & J := c_C(t_f) V(t_f) \\ \text{s.t. } & \text{model equations (7.4.1) with } k_3 = k_4 = 0, \\ & c_B(t_f) \leq c_B^{max} \;\; , \quad c_D(t_f) \leq c_D^{max} \;\; , \quad 0 \leq F(t) \leq F^{max} \;\; , \end{aligned} \tag{7.4.2}$$

with the final time $t_f = 250$ min, $F^{max} = 2 \times 10^{-3}$ L min^{-1}, $c_B^{max} = 0.025$ mol L^{-1} and $c_D^{max} = 0.15$ mol L^{-1}.

As illustrated in [142], the optimal feed rate has three parts, namely, a first arc with $F(t) = F^{max}$, a singular arc with $0 \leq F(t) \leq F^{max}$, and a third arc with $F(t) = 0$. Accordingly, the input can be parameterized using three decision variables, namely, $\pi := [t_m, t_s, \bar{F}]^T$, where t_m represents the switching time between the first and second arcs, t_s the switching time between the second and third arcs, and $\bar{F}$ the assumed constant feeding rate during the second arc. The terminal constraints are active in the optimal solution [142]. Note that solving the dynamic optimization problem using the plant model (i.e., with $k_3 = k_4 = 0$) gives qualitatively the same optimal solution (3 arcs, 2 active terminal constraints). However, open-loop application of this solution to the plant, although feasible, yields a much worse performance, namely, $J_{ol} \approx 0.3874$ mol compared to $J_{opt} \approx 0.5079$ mol. In this study, RTO will be used to drive the plant from its model optimum J_{ol} to the true plant optimum J_{opt} without knowledge of the plant model and using as few measurements as possible (each measurement corresponds to running a separate batch). Furthermore, the plant measurements are assumed to be corrupted with additive zero-mean Gaussian noise with 5% standard deviation.

Problem (7.4.2) can be reformulated as an unconstrained NLP as per equation 7.1.1 by (i) reformulating the dynamic optimization problem as a static optimization problem as illustrated in Appendix A in [143], and (ii) incorporating the constraints as a penalty term in the cost function. The parameters of Algorithm 7.2.1 are $\eta = 0.5$, $\gamma_{dec} = 0.5$, $\theta = 5 \times 10^{-5}$, $\gamma_{inc} = 1.2$, and $\Delta_0 = 15$. GP learns the

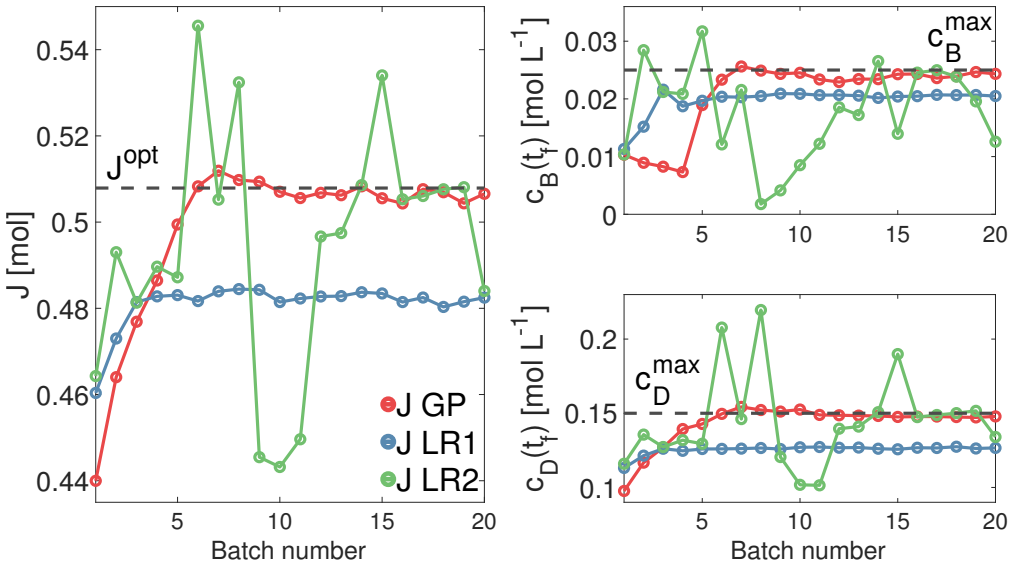


Figure 7.1 – RTO using surrogate models to represent plant-model mismatch. With GP, plant optimality is reached within 5 iterations. Linear regression using global data (LR1) does not reach plant optimality, while the linear regression using local data (LR2) exhibits oscillatory behavior and violates constraints. The plant optimum is denoted as J^{opt}.

mismatch between the plant and model costs. It is trained using 5 random points around the model optimum. For comparison with GP, we use a linear surrogate model which is proven to be fully linear [102, Chapter 2]. Note that a quadratic surrogate model would require at least 10 data points for this case study. We implement linear regression in two ways: (i) by training the regression model with all available data (denoted as LR1 in Figure 7.1), and (ii) by training the regression model with the last 5 data points (denoted as LR2 in Figure 7.1). The algorithm runs for 20 batches.

Figure 7.1 shows the performance of learning and optimization using GP and linear regressors, starting at the model optimum. GP converges to plant optimality within 5 batches. In contrast, LR1 does not reach plant optimality because of tradeoff between the different operating conditions that are included in the training set. On the other hand, LR2 reaches the neighborhood of the optimum, but it violates constraints and exhibits oscillatory behavior. This case study shows that a local surrogate model needs more data points compared to GP and can only capture efficiently local behavior. On the contrary, GP can deal with different operating conditions and is robust to noise. GP reaches plant optimality using all together 10 data points (5 training points and 5 experimental runs). Note that building a local

quadratic surrogate model would require 10 data points. Furthermore, we refer to [142] for a variant of this case study considering various RTO-based techniques.

7.5 Conclusion

In this chapter, we presented a convergence certificate for stochastic derivative-free trust-region methods based on Gaussian Processes. This work is the first to show that GP are indeed probabilistic fully-linear models. This in turn allows inferring global convergence of trust-region methods in an almost surely sense. We have demonstrated the efficacy of GP as surrogate models, drawing upon repeated open-loop optimal control of a chemical batch reaction process.

8 A Novel Deterministic Derivative-Free Trust-Region using Gaussian Processes

> I believe that mathematical reality lies outside us, that our function is to discover or observe it, and that the theorems which we prove, and which we describe grandiloquently as our "creations," are simply the notes of our observations.
>
> G. H. Hardy

Until now, we have shown that Gaussian Processes (GPs) can be certified as a probabilistic fully-linear model and therefore, global convergence proofs of probabilistic derivative-free trust-region methods hold true for GPs. In this chapter, we first certify that GPs also satisfy a deterministic fully linear property. Consequently, global convergence of various deterministic derivative-free trust-region methods can also be applied to GPs. We then propose a novel deterministic derivative-free trust-region method that addresses limitations of the standard trust-region methods. We prove the proposed algorithm's convergence to a neighborhood of an optimum.

8.1 Derivative-Free Trust-Region Methods

Our goal is to attain a local minimizer of (7.1.1) by iteratively solving and updating the surrogate problem (7.1.4) as discussed in the previous chapter. The conceptual idea of the surrogate model m_k is to *learn* the unknown function f locally or globally depending on the type of surrogate model used. Next, we present a derivative-free trust region method whose standard version is summarized in Algorithm 8.1.1, cf. [102, Chap. 10]. Like its probabilistic counterpart, a deterministic trust-region method approximates the unknown function via $m_k(x)$ within a certain neighborhood of x_k (a.k.a. the trust region). Whenever the surrogate model fails to approximate

Algorithm 8.1.1 Derivative Free Trust Region Method [102]

Data: Initital model m_0, initial point x_0, and constants $0 \leq \eta_0 \leq \eta_1, 0 < \gamma_{dec} < 1 < \gamma_{inc}, 0 < \mu$, ϵ_c.

1. **Critical step: if** $\|\nabla m_k(x_k)\| < \epsilon_c$ *and either*
   - *the model* m_k *is not fully linear; or*
   - *the trust region radius satisfies* $\Delta_k > \mu\|\nabla m_k(x_k)\|$,

   then
   call model improvement algorithm to make it fully linear and $\Delta_k \leq \mu\|g_k\|$
   end

2. **Step calculation:**

$$s_k := \underset{s:\|s\|\leq\Delta_k}{\arg\min}\, m_k(x_k + s) \tag{8.1.1}$$

3. **Estimate improvement after plant evaluation:** Evaluate

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \tag{8.1.2}$$

4. **Model improvement:** If $\rho_k < \eta_1$, then either certify that and the model $m_k(x)$ is fully linear or then call model improvement algorithm so that the model $m(x)$ is a fully linear model.

5. **Trust region update:**
   - **if** $\rho_k \geq \eta_1$ **then**
     $\Delta_{k+1} = [\Delta_k, \min\{\gamma_{inc}\Delta_k, \Delta_{max}\}]$ and $x_{k+1} = x_k + s_k$
     end
   - **if** $\rho_k < \eta_1$ *and* m_k *is fully-linear* **then**
     $\Delta_{k+1} = \gamma_{dec}\Delta_k$ and $x_{k+1} = x_k + s_k$
     end
   - **if** $\rho_k < \eta_1$ *and* m_k *is not fully-linear* **then**
     $\Delta_{k+1} = \Delta_k$ and $x_{k+1} = x_k$
     end

the original problem, then the approximation is improved (i.e., additional data points $(x_k, f(x_k))$ are collected and the model is updated) before the process is repeated. We emphasize that deterministic derivative-free trust-region works with a fully-linear

model (Definition 7.2.1) instead of probabilistic fully linear model (Definition 7.2.2).

Next we recall the main points of the convergence analysis given in [102]. Given a set $\mathcal{L}(x_0) = \{x \in \mathbb{R}^n_x : f(x) \leq f(x_0)\}$ and a closed ball $B(x; \Delta_{max})$ with radius Δ_{max} centered at x, the set $\tilde{\mathcal{L}}(x_0)$ is defined as

$$\tilde{\mathcal{L}}(x_0) = \bigcup B(x; \Delta_{max}); \text{ where } x \in \mathcal{L}(x_0). \tag{8.1.3}$$

Assumption 8.1.1 (Differentiability of f [102])**.** *The unknown function $f : \mathbb{R}^{n_x} \to \mathbb{R}$ is continuously differentiable with Lipschitz continuous gradient in an open domain containing the set $\tilde{\mathcal{L}}(x_0)$ for all $x_0 \in \mathbb{R}^{n_x}$.* □

Assumption 8.1.2 (Fully linear model [102])**.** *Let $\mathcal{M}$ denote a set of models*

$$\mathcal{M} = \{m : \mathbb{R}^{n_x} \to \mathbb{R} \mid m \in \mathcal{LC}^1\}$$

There exists a model function m in $\mathcal{M}$, with Lipschitz continuous gradient and corresponding Lipschitz constant ν_1^m such that m can be certified as a fully linear model. □

Assumption 8.1.3 (Model improvement algorithm [102])**.** *Given the model class $\mathcal{M}$ there exists a* model-improvement *algorithm, i.e. an algorithm that in a finite, uniformly bounded (with respect to x and Δ) number of iterations either*

- *establishes that a given model $m \in \mathcal{M}$ is fully linear on $B(x; \Delta)$; or that*
- *constructs a model $\tilde{m} \in \mathcal{M}$ fully linear on $B(x; \Delta)$.* □

Assumption 8.1.4 (f bounded from below [102])**.** *Assume that the unknown function f is bounded from below on $\mathcal{L}(x_0)$; that is, there exists a constant κ_* such that, for all $x \in \mathcal{L}(x_0)$, $f(x) \geq \kappa_*$.* □

Assumption 8.1.5 (Bounded Hessian of m [102])**.** *There exists a constant κ_{bhm} such that, for all x generated by the respectively considered algorithm $\|\nabla^2 m(x)\| \leq \kappa_{bhm}$ holds.* □

Theorem 8.1.1 (Global convergence [102])**.** *Let Assumptions 8.1.1–8.1.5 hold. The application of Algorithm 8.1.1 implies*

$$\lim_{k \to \infty} \|\nabla f(x_k)\| = 0$$

□

8.2 Certifying Gaussian Processes as Fully-Linear Models

In this Section, we present a certificate for GP as a fully linear model. We remind the reader that throughout this chapter we use the mean of GP as a surrogate model:

$$m(x) := \mathbb{E}\left(z(x)\right). \tag{8.2.1}$$

The following lemma, similar to Lemma 7.3.1, bounds the mismatch between the mean of GPs and the unknown function in a deterministic manner.

Lemma 8.2.1. *[Deterministic bound on mismatch between mean and an unknown function [144]] Given Assumption 7.3.1 and bounded measurement noise, the following holds*

$$|z_m(x) - f(x)| \leq \beta^{0.5} z_v(x),$$

where $z_m(x)$ and $z_v(x)$ are defined in (5.2.2) *and* (5.2.3)*, respectively.* □

Here, $\beta^{0.5}$ depends on the number of samples, RKHS norm $\|f\|_c$. See [144] for details.

Remark 8.2.1. *Lemma 7.3.1 and Lemma 8.2.1 bind the mismatch in a probabilistic and a deterministic way respectively. For Lemma 7.3.1, the unknown function $f(x)$ belongs to a distribution of functions, while for Lemma 8.2.1, $f(x)$ is a function lying in a set whose elements have known bounded norm. The set of functions fitting in the data is reduced in size with each new data point.* □

Theorem 8.2.1. *Let Assumptions 7.3.1-7.3.3 hold. If $0 < \Delta < \frac{6}{\gamma_{lh}}(\kappa_{eg} - 2\kappa_{ef} - \kappa_{bhh})$, then there exists a positive integer $N < \infty$ such that, after N sampling steps, a* GP *can be certified to be a fully linear model.*

Proof. This is straight forward adaptation of Proof for Theorem 7.3.1 when combined with Lemma 8.2.1. □

Remark 8.2.2. *Although we certify* GP *as a fully linear model, our proof mechanism can be extended to other machine-learning regression tools as well. In doing so, there are two ingredients required:*

- *Assumption 7.3.2.*

- *Bounded and known mismatch between an unknown function and a surrogate models. This is already proved for RBF [145] and NN [146].*

Therefore, our work can also be used to certify other machine learning based surrogate models as fully linear model. □

8.2.1 Challenges with Derivative-Free Trust-Region Methods

In what follows, we discuss two challenges with Algorithm 8.1.1 because they potentially lead to numerous evaluations of the unknown function f. In the context of RTO, a high number of plant evaluations are undesired and expensive, if not prohibitive.

C1 *Noise sensitivity:* The more the trust-region radius shrinks, the more accurate zeroth and first-order evaluations are required by the fully-linear model. For noisy measurements, this scenario is not realistic. Furthermore, in an RTO setting, where each function evaluation means an application of x_k to a physical system, the first-order sensitivities are difficult to obtain from a rather small number of usually noisy measurements.[1]

We will address this issue by proposing the notion of an ϵ-linear model. The advantage of an ϵ-linear model is that it allows the gradients to be noisy. Contrary to the requirement of fully linear models, we will utilize the fact that as long as our current model $m_k(x)$ provides a descent direction of the unknown function $f(x)$, there is no need for a precise gradient estimation, avoiding an excessive number of plant evaluations.

C2 *Checking full linearity of the model frequently:* Observe that Steps 1), 4) and 5) of Algorithm 8.1.1 needs either fully-linear certification of the current model $m_k(x)$ or a construction of a fully linear model. This requires additional plant evaluations, which in the context of RTO implies experiments/input applications on a real process.

We will address this challenge by proposing a novel algorithm that avoids frequent certification of full linearity.

[1] The gradient of the plant can be estimated from measurements using data-driven techniques [100].

8.3 Proposed Algorithm: Derivative-Free Machine-Learning based Trust-Region Method (DMT)

8.3.1 ϵ-Linear Models

We propose a class of ϵ-linear surrogate model for incorporating noisy gradients.

Definition 8.3.1 (ϵ-linear model)**.** *Consider a function $f : \mathbb{R}^n \to \mathbb{R}$ satisfying Assumption 8.1.1. A model $m \in \mathcal{M}$ is called ϵ-linear on $B(x;\Delta)$ if there exists positive constants $\epsilon_1, \epsilon_2, \kappa_{ef}, \kappa_{eg}$ and ν_1^m such that for any $x \in \mathcal{L}(x_0)$ and $\Delta \in (0, \Delta_{max}]$ the following holds for all $s \in B(0;\Delta)$*

$$|f(x+s) - m(x+s)| \leq \epsilon_1 \Delta + \kappa_{ef}\Delta^2, \tag{8.3.1a}$$
$$\|\nabla f(x+s) - \nabla m(x+s)\| \leq \epsilon_2 + \kappa_{eg}\Delta. \tag{8.3.1b}$$

□

Comparing Definition 7.2.1 to Definition 8.3.1, notice that the latter is more general as it allows error in gradient and function value evaluations via ϵ_1 and ϵ_2, which refers to Challenge C1 discussed in Section 8.2.1.

Assumption 8.3.1 (Model improvement algorithm)**.** *Given the model class $\mathcal{M}$ there exists a* model-improvement *algorithm, i.e. an algorithm that in a finite, uniformly bounded (with respect to x and Δ) number of steps*

- *either establishes that a given model $m \in \mathcal{M}$ is ϵ-linear on $B(x;\Delta)$; or that*
- *computes a model $\tilde{m} \in \mathcal{M}$ ϵ-linear on $B(x;\Delta)$.*

□

Fundamentally, one has to ask at this point, which approaches to learning of surrogate models comply with Definition 8.3.1 and Assumption 8.3.1. We will discuss this later in Section 8.3.3, while next we turn towards a variant of Algorithm relying on ϵ-linear models.

8.3.2 Proposed Algorithm

We now turn to the proposed method (Algorithms 8.3.1a, 8.3.1b and 8.3.1c).

Algorithm 8.3.1a Derivative-free Machine learning based Trust region algorithm (DMT)

Data: Initital model m_0, initial point x_0, and constants $0 < \eta_1, \beta, \epsilon_{1_{max}}, \epsilon_{2_{max}}, \Delta_{threshold}, \epsilon_{grad\_magnitude}, 0 < \gamma_{dec}, \gamma_{\epsilon_{dec}} < 1 < \gamma_{inc}, \gamma_{\epsilon_{inc}}$, and $\epsilon_{accuracy} \in (0, \frac{(1-\eta_1)\epsilon_{grad\_magnitude}}{4})$.
Result: Find a point x_{sol} such that $\nabla \|f(x_{sol})\| \leq \beta$

1. **Critical step:**

   (a) **if** $\|\nabla m_k(x_k)\| \leq \epsilon_{grad\_magnitude}$ **then**
   - set $\epsilon_{k_i} = \epsilon_{accuracy}, i = 1, 2$
   - Improve the model by sampling

   end

   (b) **if** $\Delta_k < \Delta_{threshold}$ **then**
   - $\epsilon_{k_i} = \max\{\gamma_{\epsilon dec}\epsilon_{k_i}, \epsilon_{accuracy}\}, i = 1, 2$
   - Improve the model by sampling

   end

   (c) **if** $\|\nabla m_k(x_k)\| < 2\nu_1^m \Delta_k \epsilon_{k_2}$ *or* $\|\nabla m_k(x_k)\|(1-\eta_1) - 4\epsilon_{k_1} \leq 0$ **then**
   call Algorithm 8.3.1b

   end

2. **Step calculation:** $s_k := \underset{s:\|s\| \leq \Delta_k}{\arg\min}\, m(x_k + s)$

3. **Estimate improvement after plant evaluation:** $\rho_k = \frac{f(x_k+s_k)-f(x_k)}{m_k(x_k+s_k)-m_k(x_k)}$

4. **Incorporate the plant evaluation:** update the model m_k with a new point $x_k + s_k$

5. **Trust region update:** Apply Algorithm 8.3.1c for computing x_{k+1} and Δ_{k+1}

6. **Termination:**
   if $\epsilon_{k_i} \leq \epsilon_{accuracy}, i = 1, 2$ *and* $\|\nabla m_k(x_k)\| \leq \epsilon_{grad\_magnitude}$ **then**
   $x_{sol} = x$ and terminate

   else
   $k = k + 1$ and go to Step 1

   end

Algorithm 8.3.1b Model improvement for small gradients.

Result:
Improved model $m_k(x_k)$ such that either of the following holds true:

- $\|\nabla m_k(x_k)\| \geq 2\nu_1^m \Delta_k \epsilon_{k_2}$ and $\|\nabla m_k(x_k)\|(1-\eta_1) - 4\epsilon_{k_1} > 0$
- $\epsilon_{k_i} \leq \epsilon_{accuracy}, i = 1, 2$ and $\|\nabla m_k(x_k)\| \leq \epsilon_{grad\_magnitude}$

1. **Improve the accuracy:**
   $\epsilon_{k_i} = \max\{\gamma_{\epsilon dec}\epsilon_{k_i}, \epsilon_{accuracy}\}, i = 1, 2$

2. **Reduce the trust region radius:**
   if $\|\nabla m_k\| < 2\nu_1^m \Delta_k \epsilon_{k_2}$ **then**
   $\Delta_k = \gamma_{dec}\Delta_k$
   end

3. Improve the model by sampling

4. **Termination condition:**
   if *either of the following holds true:*
   - $\|\nabla m_k(x_k)\| \geq 2\nu_1^m \Delta_k \epsilon_{k_2}$ *and* $\|\nabla m_k(x_k)\|(1-\eta_1) - 4\epsilon_{k_1} > 0$ *or*
   - $\epsilon_{k_i} \leq \epsilon_{accuracy}, i = 1, 2$ *and* $\|\nabla m_k(x_k)\| \leq \epsilon_{grad\_magnitude}$

   then
   return $m_k(x_k)$ and terminate
   else
   go to Step-1
   end

Algorithm 8.3.1c Trust region radius update.

Result:
Updated radius Δ_{k+1} and new-point x_{k+1} by accepting or rejecting the step.

- **Case(i)- accepted model step:**
  if $\rho_k \geq \eta_1$ **then**

    1. $x_{k+1} = x_k + s_k$
    2. $\Delta_{k+1} = [\Delta_k, \min\{\gamma_{inc}\Delta_k, \Delta_{max}\}]$
    3. **if** $\Delta_{k+1} \geq \Delta_{threshold}$ **then**
       $\epsilon_{k_i} = \epsilon_{i_{max}}, i = 1, 2$
       end

  end

- **Case(ii)- rejected model step:**
  if $\rho_k < \eta_1$ **then**

    1. $x_{k+1} = x_k$
    2. $\Delta_{k+1} = \gamma_{dec}\Delta_k$

  end

As explained in Section 8.2.1, our goal is to keep moving in a descent direction without frequently certifying the surrogate model. Consequently, we aim to avoid unnecessary evaluations near the present iterate. We explain this by comparing with Algorithm 8.1.1. Both algorithms differ in Steps 1, 4 and 5:

- **Step 1:** Algorithm 8.1.1 checks and certifies full-linearity, on the other hand Algorithm 8.3.1a has three steps.

  Step 1a and Step 1b: Once the trust region radius shrinks below some threshold value ($\Delta_{threshold}$) or the model gradient is below threshold ($\epsilon_{grad\_magnitude}$), only then it improves the surrogate model.

  Step 1c: This step guarantees that model accuracy is improved and trust-region radius is decreased for moving towards the neighborhood of an optimum.

  Essentially, Step 1 in Algorithm 8.3.1a acts like a "tuning knob" to trade-off aggressive versus cautious exploration behavior. If the user prefers cautious behavior like in the traditional approaches, then larger values of $\Delta_{threshold}$, $\epsilon_{grad\_magnitude}$, and $\gamma_{\epsilon_{dec}}$ and a smaller value for $\epsilon_{accuracy}$ should be set. This choice quickly triggers model and accuracy improvement, very similar to Step 1 of Algorithm 8.1.1. On the other hand, by setting smaller values of $\Delta_{threshold}$, $\epsilon_{grad\_magnitude}$, and $\gamma_{\epsilon_{dec}}$ and a larger value for $\epsilon_{accuracy}$ leads to infrequent calls of the model improvement part.

- **Step 4:** The only difference is that Algorithm 8.1.1 calls for model improvement, while Algorithm 8.3.1a simply incorporated the new point and updates the model.

- **Step 5:** When $\rho_k \geq \eta_1$: The iteration is successful and we reset the accuracy (see Step 5, case (i) in Algorithm 8.3.1a). This essentially means that after a model step is accepted, we do not need a very accurate model as long as the current model provides a descent direction.

  When $\rho_k < \eta_1$: Unlike Algorithm 8.1.1, Algorithm 8.3.1a does not call immediately for model improvement. Instead, it decreases the trust region radius and it tries to continue without sampling.

To sum up, Algorithm 8.3.1a aims to call for model improvement only near a local optimum point or when a surrogate model is extremely inaccurate. This addresses Challenge C2 of (too) frequent checks for model accuracy mentioned in Section 8.2.1. Another feature of the proposed work is that it allows the user to

control how close to optimal the final solution shall be. This depends on $\epsilon_{accuracy}$. Thus, for the same value of η_1 and γ_{dec}, the proposed algorithm allows several performance-speed-accuracy trade-offs which traditional approaches do not offer.

Remark 8.3.1. *Note that, for the sake of readability, we have set the accuracies ϵ_1 and ϵ_2 to be the same (Step 6 of Algorithm 8.3.1a). However, one can choose them to be different. The same holds true for $\gamma_{\epsilon_{dec}}$.* □

Remark 8.3.2 (An intuitive explanation for the convergence)**.** *The following cases are possible:*

- **The current point is not in the neighborhood of a local minimum:**

  The model gradient points to a descent direction and gives sufficient decrease: *This is the simplest case, and the algorithm moves to the new point.*

  The model gradient points to an ascent direction or does not give a sufficient decrease: *In this case, we will have failed trust-region iterations. This leads to shrinking of the trust-region radius Δ_k. After Δ_k is below some threshold value $\Delta_{threshold}$, we reduce $\epsilon_{k_1}, \epsilon_{k_2}$, and thus we keep improving[2] the surrogate model accuracy until we find a descent direction.*

- **The current point is in the neighborhood of a local minimum:** *For this case, we have two possibilities again- (i) accurate model gradient and (ii) inaccurate model gradient.*

  Accurate gradient: *In this case, the step size will be small or zero. Following this, we make the model more accurate (see Step 1 Algorithm 8.3.1a).*

  Inaccurate gradient: *If we move because of model inaccuracy, then this leads to failed iterations (moving along ascent direction); consequently, reduces the trust-region radius below $\Delta_{threshold}$. Hence, eventually this case triggers model improvement (see Step 1 of Algorithm 8.3.1a).* □

While the above has highlighted many appealing features of Algorithm 8.3.1a, now we turn to convergence properties.

Theorem 8.3.1 (Termination in a finite number of iterations)**.** *Let Assumptions 8.1.1, 8.1.4–8.3.1 hold. Then Algorithm 8.3.1a terminates after a finite number of*

[2]By "improvement" we mean to sample the plant i.e. take measurements from the system and improve the surrogate model accuracy.

iterations, i.e. after finitely many iterations it holds that $\epsilon_{k_i} \leq \epsilon_{accuracy}$, $i = 1, 2$ *and*

$$\|\nabla m_k(x_k)\| \leq \epsilon_{grad\_magnitude}.$$

Theorem 8.3.2 (Global convergence in a neighborhood of a first-order critical point)**.** *Let Assumptions 8.1.1, 8.1.4–8.3.1 hold. Then for any* $\beta > 0$*, Algorithm 8.3.1a terminates after a finite number of iterations with* $\|\nabla f(x)\| \leq \beta$*.*

The proofs for both results are given in the Section 8.5.

8.3.3 Constructing Gaussian Processes as ϵ-Linear Surrogate Models

At this point the crucial question of how to certify ϵ-linearity of a given surrogate model is addressed. First, it is important to note that any fully-linear model is ϵ-linear as well. The ϵ-linear model allows one to sample from a larger trust region compared to a fully linear model[3]. In [102], details of how to build a local surrogate model—mainly via linear and nonlinear interpolation/regression—at each iteration and how to certify full linearity are discussed. In principle, the same procedure can be used to certify them as ϵ-linear.

In what follows, we provide a certificate for GPs to be an ϵ-linear model.

Theorem 8.3.3. *Let Assumption 7.3.1 and 7.3.2 hold. If* $2\epsilon_1 < \epsilon_2$ *then there exists a positive integer* $N < \infty$ *such that after* N *sampling steps, the* GP *is certified as an* ϵ*-linear model.*

Proof. See Section 8.5. □

Remark 8.3.3. *Since an* ϵ*-linear surrogate model incorporates a noisy gradient, it enables using surrogate models capturing noise. See also Remark 8.2.2.* □

8.4 Conclusions

In this chapter, we provided certificates for GPs to be fully- and ϵ-linear. We proposed a novel algorithm that allows noisy measurement. The proposed algorithm

[3]Consider that the current trust region radius is Δ_k and thus to satisfy full linearity, one needs to sample all the points with in the radius Δ_k. On the other hand, for ϵ-linear case one can sample data from radius $\Delta_k + min\{k_{eg}^{-1}\epsilon_{k_2}, k_{ef}^{-1}\epsilon_{k_1}\}$. This helps to efficiently recycle past data points.

avoids unnecessary plant evaluation and has gurantee to reach in the neighborhood of a local optimum.

8.5 Appendix: Proofs

In Subsection 8.5.1, we show the convergence property of the proposed Algorithm based on ϵ-linear model. We derive certification of GP as a surrogate model in Subsection 8.5.2.

8.5.1 Convergence Properties of the Proposed Algorithm

First, we recall a technical lemma.

Lemma 8.5.1 ([121]). *If the model $m_k(x)$ has a Lipschitz continuous gradient with Lipschitz constant ν_1^m, then*

$$m_k(x_k) - m_k(x_k + s_k) \geq \frac{\|\nabla m_k(x_k)\|}{2} \min\left\{ \frac{\|\nabla m_k(x_k)\|}{2\nu_1^m}, \Delta_k \right\}$$

for all s_k computed from Step 2 Algorithm 8.3.1a.

Lemma 8.5.2 (Trust-region radius for a successful step). *Given Assumption 8.1.1, Definition 8.3.1 and let $\mathcal{M}$ be a class of ϵ-linear models. If*

$$\Delta_k \leq \min\left\{ \tfrac{\|\nabla m_k(x_k)\|}{2\nu_1^m}, \tfrac{\|\nabla m_k(x_k)\|(1-\eta_1)-4\epsilon_{k_1}}{4\kappa_{ef}} \right\},$$

then $\rho_k \geq \eta_1$ and Step 2 of Algorithm 8.3.1a is accepted.

Proof. Note that Step 1 of Algorithm 8.3.1a either terminates or guarantees $\|\nabla m_k(x_k)\|(1-\eta_1) - 4\epsilon_{k_1} > 0$. Now using the definition of ρ_k (Step 3, Algorithm 8.3.1a),

$$\begin{aligned}
\rho_k - 1 &= \frac{f(x_k + s_k) - f(x_k)}{m_k(x_k + s_k) - m_k(x_k)} - 1 \\
&= \frac{f(x_k + s_k) - m_k(x_k + s_k)}{m(x_k + s_k) - m_k(x_k)} \\
&\quad - \frac{f(x_k) - m_k(x_k)}{m(x_k + s_k) - m_k(x_k)}.
\end{aligned}$$

Using norm properties, we obtain

$$|\rho_k - 1| \leq \left| \frac{f(x_k + s_k) - m_k(x_k + s_k)}{m(x_k + s_k) - m_k(x_k)} \right| + \left| \frac{f(x_k) - m_k(x_k)}{m(x_k + s_k) - m_k(x_k)} \right|$$

Using Definition, 8.3.1 we obtain

$$|\rho_k - 1| \leq \frac{\epsilon_{k_1}\Delta_k + \kappa_{ef}\Delta_k^2}{|m(x_k + s_k) - m_k(x_k)|} + \frac{\epsilon_{k_1}\Delta_k + \kappa_{ef}\Delta_k^2}{|m(x_k + s_k) - m_k(x_k)|}.$$

Next, we apply Lemma 8.5.1

$$|\rho_k - 1| \leq \frac{\epsilon_{k_1}\Delta_k + \kappa_{ef}\Delta_k^2}{\frac{\|\nabla m_k(x_k)\|\Delta_k}{2}} + \frac{\epsilon_{k_1}\Delta_k + \kappa_{ef}\Delta_k^2}{\frac{\|\nabla m_k(x_k)\|\Delta_k}{2}} = \frac{4\left(\epsilon_{k_1}\Delta_k + \kappa_{ef}\Delta_k^2\right)}{\|\nabla m_k(x_k)\|\Delta_k} = \frac{4\left(\epsilon_{k_1} + \kappa_{ef}\Delta_k\right)}{\|\nabla m_k(x_k)\|}$$

Substituting $\Delta_k \leq \frac{\|\nabla m_k(x_k)\|(1-\eta_1)-4\epsilon_{k_1}}{4\kappa_{ef}}$ leads to $|\rho_k - 1| < (1 - \eta_1)$, and therefore, $\rho_k > \eta_1$. □

Now, we show that the trust-region radius for Algorithm 8.3.1a converges to zero. The skeleton of the proof is the same as for Algorithm 8.1.1 (Lemma 10.9 [102]). Observe that both Algorithms have the same trust-region management (see Step 5). The proof is different only due to Step 1, where Algorithm 8.1.1 ensures $\|\nabla m_k(x_k)\| \geq \min\{\epsilon_c, \mu^{-1}\Delta_k\}$, while Algorithm 8.3.1a ensures $\|\nabla m_k(x_k)\| \geq \min\{\frac{4\epsilon_{k_1}}{(1-\eta_1)}, 2v_1^m\Delta_{s_k}\epsilon_{s_{k_2}}\}$. This way, the following proof is the straight adaptation of Lemma 10.9 in [102].

Lemma 8.5.3. *Let Assumptions 8.1.1, 8.1.4–8.3.1 hold. For Algorithm 8.3.1a, if $k \to \infty$, then $\Delta_k = 0$.*

Proof. We consider three cases:

(i) Finite number of successful iterations

(ii) Infinite number of successful iterations

(iii) Mix of successful and unsuccessful iterations

Case (i) Consider all the iterations after the last successful iteration. The only possibility to increase the trust region radius is accepting a new point, i.e., when Step 5 case (i) of Algorithm 8.3.1a is invoked. Notice that Algorithm 8.3.1b never increases the Δ_k. Since there are no more successful iterations, Step 5 case (i) of Algorithm 8.3.1a is not invoked. This means that there are only unsuccessful iterations reducing the trust-region radius, see Step 5 case (ii). Hence, Δ_k is always reduced and hence as per Step 5 case (ii) Algorithm 8.3.1a converges to zero.

Case (ii) Let $\mathbb{S}$ be the set of indexes of successful iterations (in this case infinite) and $s_k \in \mathbb{S}$. When infinitely many successful iteration occur, then by definition Step 3 Algorithm 8.3.1a, $\rho_{s_k} \geq \eta_1$ for a sufficiently large s_k. Substituting for ρ_{s_k} leads to

$$\begin{aligned} f(x_{s_k}) - f(x_{s_k} + d_{s_k}) &\geq \eta_1 \left[m_{s_k}(x_{s_k}) - m_{s_k}(x_{s_k} + d_{s_k}) \right], \\ &\quad \text{(Next using Lemma 8.5.1)} \\ &\geq \frac{\|\nabla m_{s_k}(x_{s_k})\|}{2} \\ &\quad \min \left\{ \frac{\|\nabla m_{s_k}(x_{s_k})\|}{2\nu_1} \ , \Delta_{s_k} \right\} \end{aligned}$$

Due to Step 1 Algorithm 8.3.1a, we have $\|\nabla m_{s_k}(x_{s_k})\| \geq \min \{ \frac{4\epsilon_{s_{k1}}}{(1-\eta_1)}, 2v_1^m \Delta_{s_k} \epsilon_{s_{k2}} \}$. Since the function $f(x)$ is bounded from below and because all iterations are successful, the left-hand side of the above inequality must go to zero. Therefore, the right-hand side must be zero because it is non-negative. Note that $\epsilon_{s_{k2}}$ and $\epsilon_{s_{k1}}$ will never reach zero due to Step 6 of Algorithm 8.3.1a and Step 2 of Algorithm 8.3.1b. Thus, the only possibility for the right-hand side to be zero is when the radius Δ_{s_k} goes to zero.

Case (iii) Mix of successful and unsuccessful iterations: Let $k \notin \mathbb{S}$ be the index of an iteration after the first successful iteration s_k. Recall that the trust-region radius can only be increased when the iteration is successful (Step 5 Algorithm 8.3.1a). Therefore, $\Delta_k \leq \gamma_{inc} \Delta_{s_k}$. In case (ii), we proved that as $\Delta_{s_k} \to 0$. Hence, $\Delta_k \to 0$. □

Remark 8.5.1. *In the proof of Lemma 8.5.3, it seems paradoxical that if there are infinitely many successful iterations, and if at each successful iteration, the trust region radius is either increased or kept at Δ_{max} (see step 5 case(i) in*

Algorithm 8.3.1a), then how is it possible that Δ_{s_k} goes to zero? The answer lies in Step 1 Algorithm 8.3.1a, which ensures $\|\nabla m_{s_k}(x_{s_k})\| \geq 2\nu_1^m \Delta_{s_k} \epsilon_{s_{k_2}}$ by calling Algorithm 8.3.1b. Note that the gradient decreases because f is bounded from below. Therefore infinite number of successful iterations makes $\|\nabla m_{s_k}(x_{s_k})\|$ decreasing and consequently Δ_{s_k}. Thus, it is Step 2 of Algorithm 8.3.1b which leads Δ_{s_k} to go to zero despite all successful iterations.

Lemma 8.5.4. *Let Assumptions 8.1.1, 8.1.4–8.3.1 hold. Algorithm 8.3.1b terminates in a finite number of iterations.*

Proof. We consider the following two cases:

(i) $\|\nabla m_k(x_k)\| > \epsilon_{grad\_magnitude}$

(ii) $\|\nabla m_k(x_k)\| \leq \epsilon_{grad\_magnitude}$

Observe that there are two termination conditions (Step 4, Algorithm 8.3.1b). We will use the first termination condition to prove the first case and the second termination condition for the second case.

(i) $\|\nabla m_k(x_k)\| > \epsilon_{grad\_magnitude}$: We assume that the Algorithm never terminates, hence, $\|\nabla m_k\| < 2\nu_1^m \Delta_k \epsilon_{k_2}$. Since the trust-region radius is also decreasing (Step 2 Algorithm 8.3.1b), for large enough finite N, there exists $\zeta > 0$ so that for all $k > N$,

$$0 < 2\nu_1^m \Delta_k \epsilon_{k_2} \leq 2\nu_1^m \zeta \epsilon_{k_2} < \|\nabla m_k(x_k)\|,$$

which contradicts the assumption.

Next, we need to show that $0 < \epsilon_{k_1} < \frac{\|\nabla m_k(x_k)\|(1-\eta_1)}{4}$. By definition

$$\epsilon_{accuracy} \in \left\{0, \frac{(1-\eta_1)\epsilon_{grad\_magnitude}}{4}\right\},$$

(see input data of Algorithm 8.3.1a). Step 1 of Algorithm 8.3.1b ensures that in a finite number of iterations $0 < \epsilon_{k_1} \leq \epsilon_{accuracy}$,

$$\epsilon_{k_1} \leq \epsilon_{accuracy} < \frac{(1-\eta_1)\epsilon_{grad\_magnitude}}{4} < \frac{\|\nabla m_k(x_k)\|(1-\eta_1)}{4}.$$

(ii) $\|\nabla m_k(x_k)\| \leq \epsilon_{grad\_magnitude}$: We show that the second termination condition of Step 4 Algorithm 8.3.1b is satisfied. Observe that the case condition $\|\nabla m_k(x_k)\| \leq \epsilon_{grad\_magnitude}$ holds by assumption.

Because Step 1 always reduces ϵ_{k_i}, for exist N, so that for all $k > N$, $\epsilon_{k_i} \leq \epsilon_{accuracy}, i = 1, 2$. □

Next, we prove that if the norm of the model-gradient is non-zero, then the trust-region radius remains strictly positive.

Lemma 8.5.5. *Let Assumptions 8.1.1, 8.1.4–8.3.1 hold. For Algorithm 8.3.1a, if there exists κ_1 such that $\|\nabla m_k(x_k)\| \geq \kappa_1 > 0$ for all k, then there exists $\kappa_2 > 0$ such that $\Delta_k \geq \kappa_2$.*

Proof. We prove this by contradiction. Note that Δ_k is reduced in Step 5 of 8.3.1a and Step 2 of Algorithm 8.3.1b. We know from Lemma 8.5.4 that Algorithm 8.3.1b terminates in a finite number of iteration, and therefore, the only possible way for $\Delta_k \to 0$ is that Algorithm 8.3.1a never terminates and due to Step 5 of Algorithm 8.3.1a the radius Δ_k is shrunk. However, if $\|\nabla m_k(x_k)\| \geq \kappa_1 > 0$ and $\Delta_k = 0$, then according to Lemma 8.5.2, the step is accepted. This contradicts our assumption. □

Lemma 8.5.6. *Let Assumptions 8.1.1, 8.1.4–8.3.1 hold. For Algorithm 8.3.1a, there exists positive integers $k, N < \infty$ such that $\Delta_k < \Delta_{threshold}, \forall k \geq N$.*

Proof. The result is a direct consequence of Lemma 8.5.3. □

Lemma 8.5.7. *Let Assumptions 8.1.1, 8.1.4–8.3.1 hold. For Algorithm 8.3.1a,there exist positive integers $k, N < \infty$ such that such that $\epsilon_{k_1} = \epsilon_{accuracy}$, and $\epsilon_{k_2} = \epsilon_{accuracy} \forall k > N$.*

Proof. The result is a direct consequence of Lemma 8.5.3 and Step 1b of Algorithm 8.3.1a. □

Proof of Theorem 8.3.1

We recall that Lemma 8.5.4 shows that Algorithm 8.3.1b terminates in a finite number of iterations. So the only remaining possibility is that Algorithm 8.3.1a never terminates. We use a contradiction for the proof.

Among the two necessary conditions for termination (see Step 6), the first termination condition is satisfied in a finite iterations due to Lemma 8.5.7. Therefore, the only possibility for Algorithm 8.3.1a never terminates is $\|\nabla m_k(x_k)\| >$

$\epsilon_{grad\_magnitude}$. In that case, it follows from Lemma 8.5.5 that there exists $\kappa > 0$ such that $\Delta > \kappa > 0$ for all the iterations. This, however, contradicts Lemma 8.5.3. □

The following Corollary is a direct consequence of Theorem 8.3.1.

Corollary 8.5.1. *Let Assumptions 8.1.1, 8.1.4–8.3.1 hold. For Algorithm 8.3.1a, there exist positive integers* $k, N < \infty$ *such that* $\forall k > N$, $\|\nabla m_k(k)\| \leq \epsilon_{grad\_magnitude}$.

Proof of Theorem 8.3.2

Using (8.3.1b) and norm properties, we can write

$$\epsilon_{k_2} + \kappa_{eg}\Delta_k \geq \|\nabla f(x_k) - \nabla m_k(x_k)\| \geq \|\nabla f(x_k)\| - \|\nabla m_k(x_k)\|. \tag{8.5.1}$$

Moreover, we know from Theorem 8.3.1 that the algorithm terminates in a finite number of iterations. Therefore, the termination condition $\epsilon_{k_2} \leq \epsilon_{accuracy}$ and $\|\nabla m(k)\| \leq \epsilon_{grad\_magnitude}$ are satisfied for k large enough. Substituting in 8.5.1,

$$\begin{aligned}\|\nabla f(x_k)\| &\leq \|\nabla m_k(x_k)\| + \epsilon_{accuracy} + \kappa_{eg}\Delta_k \\ \|\nabla f(x_k)\| &\leq \underbrace{\epsilon_{grad\_magnitude} + \epsilon_{accuracy} + \kappa_{eg}\Delta_k}_{\beta}.\end{aligned}$$

This concludes the proof. □

Remark 8.5.2 (Arbitrarily close to the critical point)**.** *We observe that Lemma 8.5.3 gives* $\Delta_k \to 0$. *Now, by arbitrarily reducing* $\epsilon_{grad\_magnitude}$ *and* $\epsilon_{accuracy}$, *the gradient* $\|\nabla f(x_k)\|$ *can be made arbitrarily close to zero.*

8.5.2 Certifying Gaussian Processes as ϵ-Linear Models

Proof for Theorem 8.3.3

Proof. We need to show that (8.3.1a) and (8.3.1b) hold.

We first consider for (8.3.1a). For a given ϵ_1, ϵ_{ef}, and Δ, we can find $\zeta_{tmp} > 0$ such that $\zeta_{tmp} \leq \epsilon_1\Delta + \epsilon_{ef}\Delta^2$. Since with the increasing sampling the bound of

Lemma 7.3.1 concludes, the following holds true:

$$\sqrt{\beta} z_v(x) \leq \zeta_{tmp} \leq \epsilon_1 \Delta + \epsilon_{ef} \Delta^2.$$

In this way, one can certify (8.3.1a).

Next we turn to (8.3.1b). For ease of notation we define $h(x) = f(x) - m(x)$. Now using Taylor expansion,

$$\begin{aligned} h(x+s) =& h(x) + s^\top \nabla h(x) + s^\top \nabla^2 h(x) s + \mathcal{O}(s^3) \\ |s^\top \nabla h(x)| =& |h(x+s) - h(x) - s^\top \nabla^2 h(x) s - \mathcal{O}(s^3)| \\ \leq& |h(x+s)| + |h(x)| + |s^\top \nabla^2 h(x) s| + |\mathcal{O}(s^3)| \\ \leq& |h(x+s)| + |h(x)| + |s^\top \nabla^2 h(x) s| + \frac{\gamma_{lh}}{6} \|s\|^3 \ , \end{aligned}$$

where the first inequality comes from the norm properties and the second using Lemma 4.1.14 from [140]. Substituting $s = \frac{\nabla h(x) \Delta}{\|\nabla h(x)\|}$ by following Lemma 4.7 in [141] gives,

$$\Delta \|\nabla h(x)\| \leq |h(x+s)| + |h(x)| + \Delta^2 \|\nabla^2 h(x)\| + \frac{\gamma_{lh}}{6} \Delta^3$$

$$\Delta \|\nabla h(x)\| \leq |h(x+s)| + |h(x)| + \Delta^2 \kappa_{bhh} + \frac{\gamma_{lh}}{6} \Delta^3.$$

Here the last inequality arises by applying Assumption 7.3.2. By model improvement, we can guarantee that $|h(x+s)|, |h(x)| \leq \epsilon_1 \Delta + \kappa_{ef} \Delta^2$,

$$\begin{aligned} \Delta \|\nabla h(x)\| \leq& 2\epsilon_1 \Delta + 2\kappa_{ef} \Delta^2 + \Delta^2 \kappa_{bhh} + \frac{\gamma_{lh}}{6} \Delta^3 \\ \|\nabla h(x)\| \leq& 2\epsilon_1 + 2\kappa_{ef} \Delta + \Delta \kappa_{bhh} + \frac{\gamma_{lh}}{6} \Delta^2. \end{aligned}$$

Combining the above with definition of perturbed linear model 8.3.1b, we need to show that for a given ϵ_2 and κ_{eg} the following criterion can be satisfied:

$$2\epsilon_1 + 2\kappa_{ef}\Delta + \Delta\kappa_{bhh} + \frac{\gamma_{lh}}{6}\Delta^2 \leq \epsilon_2 + \kappa_{eg}\Delta$$

For any $2\epsilon_1 < \epsilon_2$, $\exists\Delta > 0$ such that the above equality is satisfied. Thus one can certify GPs as an ϵ-linear models. □

9 Experimental Results - Gaussian Process Based Real-time Optimization of Solid-Oxide Fuel Cells

> Science is built up of facts, as a house is built of stones; but an accumulation of facts is no more a science than a heap of stones is a house."
>
> Henri Poincaré

9.1 Introduction

Renewable energy generation is the need of the hour due to climate change and global warming. Fuel cell technologies are competitive alternatives because a fuel cell system is capable of producing electricity 50% cheaper than the current market prices reaching efficiencies of about 60% at 0.3 W/cm^2 [147]. For safe and sustainable operations, it is crucial that fuel-cells adhere to safe operating conditions, while achieving the maximum efficiency for a changing power demand. Although a plethora of works have modeled Solid-Oxide Fuel Cell (SOFC) systems, plant-model mismatch, degradation, and disturbances prohibits them from reaching optimal operating conditions. For this reason, data-driven optimization methods become an appealing candidate for dealing with these challenges.

The experimental set-up in this study consists of an SOFC short-stack developed by SOLIDpower, a pre-reformer and two electrical heaters as illustrated in Figure 9.1. All the mentioned elements are assembled inside of a high-temperature furnace at 780°C. The degraded stack is at the end of its lifetime and is composed of six planar anode-supported cells that have an active area of 80 cm^2. An end-of-life system has purposefully been selected as it has an I-V (current-voltage characteristic curve) curve steeper and its cell potential is significantly lower, consequently, a shortened

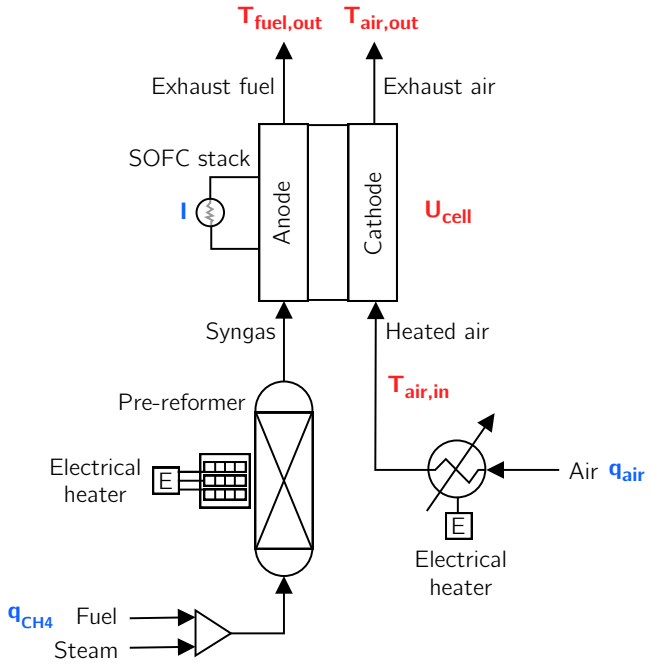


Figure 9.1 – SOFC Flowsheet. The blue and red variables represent inputs and outputs, respectively.

operating constraint window. Under those circumstances, this system is more difficult to be operated from a practical point of view. The cells are concatenated vertically and compressed between a gas-diffusion layer and metallic interconnector plates. The cathode chambers, which allow operating temperatures of 650°C to 850°C, are made of (La,Sr)(Co, Fe)O_3, while the anode consists of Ni-YSZ (nickel/yttriumstabilized-zirconia). The electrolyte, which is a thin barrier layer, consists of YSZ and CGO (Gadolinium-doped ceria).

The purpose of the **pre-reformer** in these types of setups is to convert natural gas (methane) and steam into syngas, which is a mixture of gases primarily composed of hydrogen and carbon monoxide. In this study, the methane is partially reformed in the pre-reformer and the remaining fuel is reformed in the stack. The pre-reformer is a shell-and-tube reactor whose operating temperature ranges from 400°C to 500°C in order to ensure partial fuel reformation in the pre-reforming. This is due to the fact that fuel reforming is a highly endothermic process and does not allow the stack temperature to reach very high values, which can happen as the reaction that enables electricity generation is exothermic. .

The stack mainly performs two tasks: internal reforming, that is, converting unre-

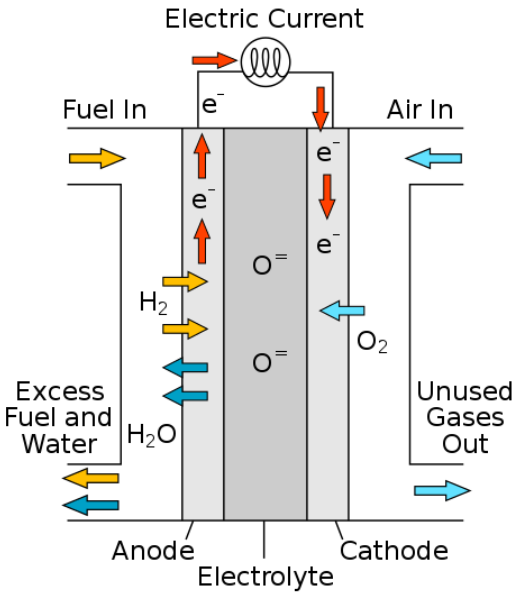


Figure 9.2 – Scheme of a solid oxide fuel cell [148].

acted methane into hydrogen and generating electricity due to an electrochemical reaction (Figure 9.2). The set-up under consideration has 6 fuel-cells assembled in the stack. In order to achieve the desired voltage, the stack is polarized using an exogenous current. The current ionizes hydrogen (H_2) and oxygen (O_2), and due to electrochemical reaction power and heat is produced. The produced power is quantified as $P_{\text{el}} = 6U_{\text{cell}}I$. As mentioned earlier, the stack under consideration is degraded and at the end of its life.

Inputs and Constraints: In this work, we consider three inputs: Methane flow-rate (q_{CH_4}), air flowrate (q_{air}), and current (I). The constraints are listed in Table 9.1. We define fuel utilization (ν) as the ratio between I and q_{CH_4}. The Ratio between q_{air} and q_{CH_4} is called as air-excess ratio (λ_{air}) [149, 150]. The minimum cell voltage (U_{cell}) is set to prevent the system from further degradation. To avoid excess heating, q_{CH_4} and I are constrained. The fuel utilization ν is bounded conservatively to 0.75 for preventing fuel starvation. The constraint on λ_{air} prohibits steep thermal gradients. Temperatures constraints ($T_{\text{fuel}}, T_{\text{air-out}}, T_{\text{air-diff}}$) are set for avoiding material damage and mechanical stress to the system, where $T_{\text{air-diff}}$ is the temperature difference between the incoming and outgoing air temperature.

For any electric power (P_{el}), our objective is to maximize efficiency (η_{el}) defined as:

Constraint	Lower bound	Upper bound
I [A]	0	50
q_{CH_4} [L.min^{-1}]	0.144	0.900
q_{air} [L.min^{-1}]	15	50
λ_{air} [-]	4	-
ν [-]	-	0.75
U_{cell} [V]	0.7	-
$T_{air\text{-}out}$ [°C]	680	800
T_{fuel} [°C]	680	800
$T_{air\text{-}diff}$ [°C]	-	100

Table 9.1 – Operational constraints of SOFC system.

$$\eta_{el} = \frac{P_{el}}{q_{CH_4} LHV_{CH4}}, \tag{9.1.1}$$

where LHV_{CH4} is lower heating value of methane and is a constant. We also include the cost for inlet air ($\delta_{air} \propto q_{air}^2$) [151] and Therefore, the optimization problem is formulated as:

$$\begin{aligned}
\max_u \Phi(u) &:= \eta_{effic} - \delta_{air} \\
\text{subject to} & \\
P_{el}(u) &= P_{el}^S, \\
0 &\leq \text{I} \leq 50[\text{A}], \\
0.144 &\leq q_{CH_4} \leq 0.9[\text{L.min}^{-1}], \\
15 &\leq q_{air} \leq 50[\text{L.min}^{-1}], \\
\lambda_{air}(u) &\geq 4, \\
\nu(u) &\leq 0.75, \\
U_{cell}(u) &\geq 0.7[\text{V}], \\
680 &\leq T_{air\text{-}out}(u) \leq 800[\ ^\circ\text{C}], \\
680 &\leq T_{fuel}(u) \leq 800[\ ^\circ\text{C}], \\
T_{air\text{-}diff}(u) &\leq 100[\ ^\circ\text{C}],
\end{aligned} \tag{9.1.2}$$

where $u = [\text{I}, q_{CH_4}, q_{air}]$ and P_{el}^S is the setpoint for the power demand.

9.2 Learning Plant-Model Mismatch using Gaussian Processes

We aim to use formulation (6.1.3) to learn the plant-model mismatch using the mean of a Gaussian process (GP). To this end, we use the model proposed in [150]. The model is a rather simple lumped model that encompasses the chemical reactions, and the mass and energy balances. The data is collected every 15seconds.

We learn four output quantities (thus four GPs): $U_{\text{cell}}, T_{\text{fuel}}, T_{\text{air-out}}$, and $T_{\text{air-diff}}$. Since, the model cannot provide the fuel inlet temperature, we learn the difference in inlet-outlet fuel temperature ($T_{\text{air-diff}}$) based on measurements only. We train GPs using 12 data points listed in Table 9.3. Observe that most of the points are in a conservative-region, consequently, with rather low efficiency. The last point represents the application of the model optimum inputs at 100W to the fuel-cell system. We provide plant-model mismatch values used for GP training in Table 9.2. Throughout this chapter, we use ARD squared exponential kernel and GPML toolbox [125]. We do not need to learn a GP for P_{el} as P_{el} is computed based U_{cell} and I, which is an input.

U_{cell} [V]	T_{fuel} [°C]	$T_{\text{air-in}}$ [°C]	$T_{\text{air-diff}}$ [°C]
0.0115	-9.1911	6.2949	11.9700
0.0421	0.3161	9.5109	9.7400
-0.0153	-14.6000	4.4822	12.8300
0.0083	-24.4761	-0.4344	13.6700
0.0642	-9.4020	4.2882	9.6400
0.1323	1.2222	7.3288	8.5117
0.0684	-4.9394	7.3885	10.6213
0.0392	-27.0135	-1.4172	14.6544
-0.0237	-20.6120	2.2518	15.8117
0.1076	4.1172	9.0314	8.8191
-0.0044	-14.0413	6.3870	14.6972
0.0763	0.0957	8.6444	9.5320

Table 9.2 – Plant-Model mismatch for training data given in Table 9.3.

q_{CH_4} [L.min^{-1}]	q_{air} [L.min^{-1}]	I [A]	λ_{air}	ν	T_{fuel} [°C]	$T_{air\text{-}in}$ [°C]	$T_{air\text{-}out}$ [°C]	U_{cell} [V]	η_{el}[%]	P_{el} [W]
0.35	25.00	15.00	7.50	0.44	742.67	769.85	757.88	0.80	34.96	72.90
0.28	16.00	12.15	6.00	0.45	752.39	771.15	761.41	0.85	37.25	62.10
0.38	30.00	10.00	8.30	0.27	734.04	765.64	752.81	0.85	22.65	51.30
0.35	40.00	15.00	12.00	0.44	725.20	762.48	748.81	0.80	34.83	72.60
0.55	21.00	38.00	4.00	0.72	756.83	779.88	770.24	0.68	47.58	155.80
0.27	15.00	19.93	5.70	0.74	758.88	773.31	764.80	0.78	56.63	93.80
0.38	20.00	21.96	5.50	0.60	751.69	774.41	763.79	0.78	45.88	103.80
0.45	40.00	32.00	9.30	0.74	732.84	772.58	757.92	0.66	47.52	127.30
0.68	33.00	33.47	5.10	0.51	738.73	777.00	761.19	0.69	34.36	139.20
0.17	15.00	11.90	9.20	0.73	757.33	770.89	762.07	0.80	56.81	57.70
0.68	26.00	35.71	4.00	0.54	747.90	782.70	768.00	0.69	36.63	149.10
0.39	15.00	23.85	4.00	0.63	758.79	776.69	767.16	0.77	47.31	111.00

Table 9.3 – Plant input-output pairs for training data.

9.3 Experiment 1: Real-Time Optimization using Gaussian Processes as a Global Surrogate Model

In this experiment, we consider the formulation given in (6.1.3). We consider the following three electrical power setpoints:

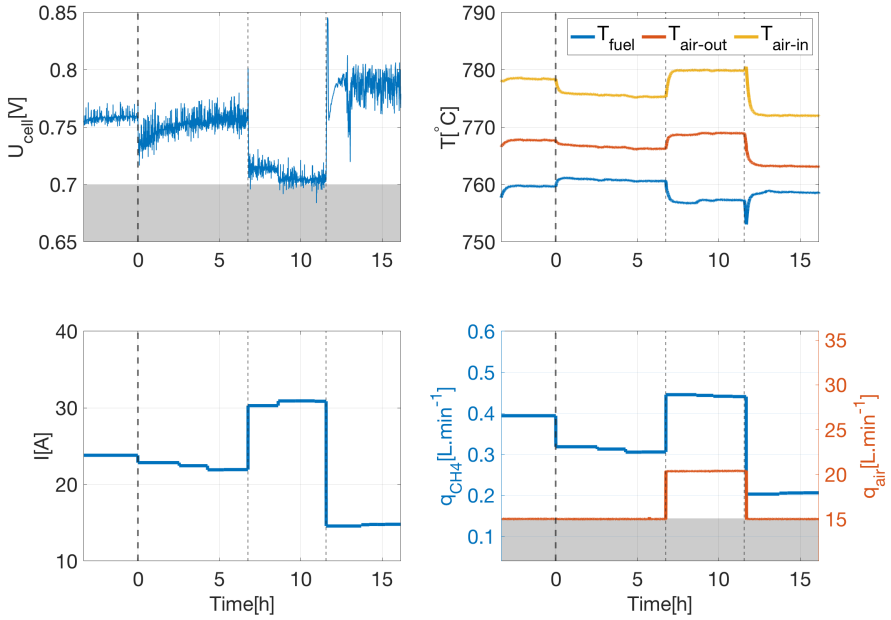
$$P_{\text{el}} = \begin{cases} 100 \text{ W} \\ 130 \text{ W} \\ 70 \text{ W} \end{cases}$$

We apply Algorithm 8.3.1a without incorporating constraints in a penalty function, i.e. we solve the constrained problem. This was the chosen approach because (i) as shown in Figure 7.1, the penalty functions can violate the constraints. Since the system was significantly degraded, meeting constraints was a priority. (ii) Usually, the optimum of SOFC systems tend to be at the intersection of active constraints. Therefore, barrier constraints may not let the system reach the optimum.

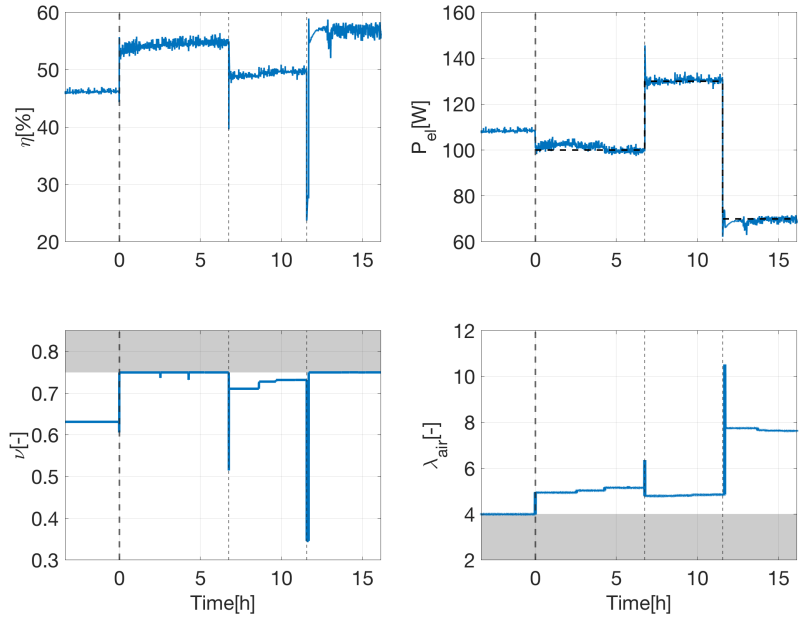
We begin the experiment by applying the model optimum at 100 W to the plant (Figure 9.3). It can be observed that the upper bound of the fuel utilization is reached at the first iteration. The optimizer pushes the system to reach high values of ν so as to increase the system efficiency. Although the data used to learn the GPs are quite conservative in terms of efficiency, the system reaches 103 W in the first iteration and 101 W in the second. The efficiency of the degraded cell is close to 55%. Even though, U_{cell} is noisy, GPs are robust and drive the plant to the preset power of 100 W.

When the power setpoint is changed to 130 W, the inputs given by the first RTO iteration already delivers 130 W with 49% efficiency. U_{cell} hits the constraints and consequently prohibits ν from reaching its maximum. The change in power setpoint to 70 W also is achieved within one or two iterations. The "dip" on the system efficiency during the transition between electrical power setpoints are due to the fact that inputs are consecutively applied in order to not violate constraints on ν and λ_{air}. Note that we wait for the SOFC system to reach the steady state before applying the next input. In addition, the RTO iteration frequency changes according to the time the system takes to settle.

We reach a neighborhood of all tested electrical power setpoints in one to two iterations. For this specific setup, the standard steady-state RTO method takes 3 to 4 iterations to reach plant optimum, and an input filter equal to 0.6 (60% filtered) is applied [150]. Input filtering is commonly used in standard RTO so as to avoid constraint violation. However, input filtering was not applied for the GP-based RTO approach as it inherently has more information from the system due to plant data collection and it relies on global surrogate models. The advantage of GP-based RTO over standard RTO approaches is the fact that the former performs higher order corrections of the constraint and objective functions, whereas the latter makes zeroth- and first-order corrections.



(a) Real-time optimization using Gaussian processes: cell voltage, cathod temperature, anode temperature, current, methane, and air flowrates vs. time.



(b) Real-time optimization using Gaussian processes: efficiency, power, fuel utilization, and air-excess ratio vs. time.

Figure 9.3 – Real-time optimization using Gaussian processes as a global surrogate model.

9.4 Experiment 2: A Variant of Modifier Adaptation Technique using Gradients of Gaussian Processes

In this experiment, we propose and test a variant of a Modifier Adaptation (MA) scheme. We perform the second experiment according to formulation (6.0.3). To this end, zeroth- and first-order modifiers ϵ and λ are computed as per equations (6.0.3d), (6.0.3e) and (6.0.3f). In the RTO settings, generally, plant gradients are computed using finite-difference method, and hence perturbing the plant. The settling time of the plant is in the range of 30 minutes to 1.5 hours. Therefore, computing plant gradients experimentally should be avoided.

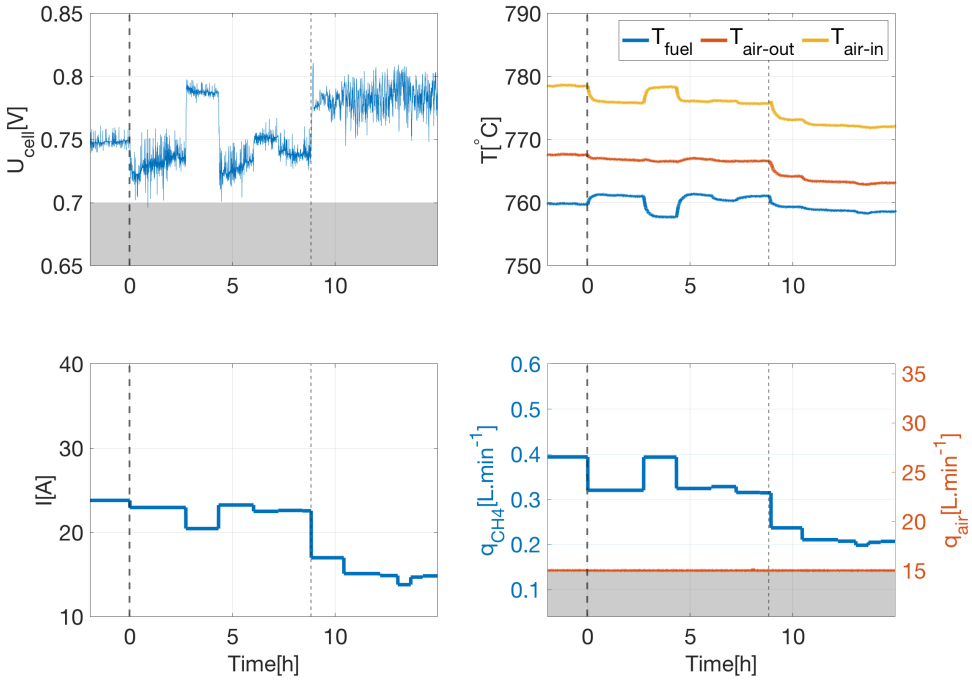
Our idea is to compute the first-order modifiers analytically using the gradient of the GP-mean. This is due to the fact that the GP learns the mismatch between the plant and the model, see (6.1.2). In this way, we estimate the plant gradient experimentally without having to perturb each input separately. Therefore, computing the gradient of (6.1.2) gives us modifiers. In this experiment, we consider the following two electrical power setpoints:

$$P_{\text{el}} = \begin{cases} 100 \text{ W} \\ 70 \text{ W} \end{cases}$$

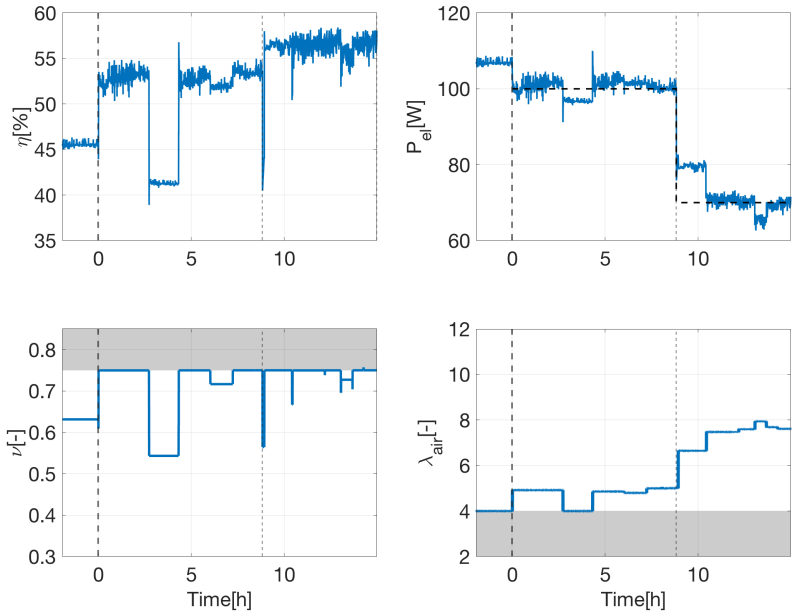
Observe that modifiers (6.0.3d). (6.0.3e) and (6.0.3f) are computed at the current point. Furthermore, only the zeroth and the first-order information of the GP is being used. This limits the information of GPs to be local. Therefore, we expect a local surrogate model type behavior and that is indeed the case, see Figure 9.4. We again start the experiment by applying the model optimum at 100 W to the plant. We take more iterations to reach a neighborhood of the optimum. However, it is important to note here that due to further plant degradation, GPs needed to learn more information from the plant as the training data used is not accurate. We suspect that due to this reason, the second iteration (for 100 W) and the fourth iteration (for 70W), the plant goes to a lower efficiency and then goes back. Nevertheless, we expect similar or slightly better behavior if the plant had not been degraded.

Although convergence is slower than the previous experiment, the results are, if not

better, as promising as well established steady-state RTO methods. The advantage of the proposed method over standard RTO approaches is the fact that the former computes plant gradients from GP functions, whereas the latter relies on finite differences, which take n+1 (where n is the number of inputs) perturbations in order to estimate the plant gradients.



(a) Real-time optimization using Gaussian processes: cell voltage, cathod temperature, anode temperature, current, methane, and air flowrates vs. time.



(b) Real-time optimization using Gaussian processes: efficiency, power, fuel utilization, and air-excess ratio vs. time.

Figure 9.4 – Real-time optimization using gradients of Gaussian processes.

9.5 Analysis and Assessment of Optimality

Three inputs are used to steer the SOFC system to optimal operation: current, fuel and air flowrates. The current is the optimization variable that most impacts the electrical power, whereas the stack temperatures are mostly affected by the air flowrate. On the other hand, the fuel flowrate impacts the stack temperature, due to internal reforming, and is the input responsible for pushing the fuel utilization to its high values, consequently, forcing the system efficiency to increase. Therefore, it is common to expect that the SOFC efficiency is achieved when (1) the highest value of fuel utilization is reached, (2) the demanded electrical power is regulated by the current and (3) the minimal amount of air flowrate is obtained so as to not violate the system temperature bounds.

An optimality assessment is performed in this section for the experimentally tested electrical powers.

Fuel utilization is at its maximum value of 0.75 and the air flowrate is at its minimum at 70 and 100 W. This suggests fuel utilization would be violated by decreasing the amount of fuel flowrate, and efficiency would decrease if the fuel flowrate was increased (see equation (9.1.1)). For the air flowrate, decreasing its value is not a possibility as it is at its lower bound and increasing it would lead to suboptimality. Any alteration of the current would cause a setpoint deviation for the electrical power. Note that decreasing the current, the fuel utilization also decreases, thus reducing the efficiency. And the fuel utilization upper bound would be violated if the current was increased. As an alternative, one could attempt to keep the same fuel utilization by increasing simultaneously the current and fuel flowrate. However, besides decreasing the electrical efficiency of the system, the electrical power would deviate from its setpoint. One could also decrease the current and the fuel flowrate aiming at maintaining the same fuel-utilization value. In this case, the electrical efficiency of the system would be improved, but a violation of the electrical power equality constraint would be observed.

Since no change on the optimization variables would lead to better performance and still remain feasible, the aforementioned arguments suggest that the SOFC system is at a local optimum at both 70 and 100 W.

Cell potential is at its lower bound at 130 W. Owing to the fact that an increase of the fuel utilization reduces the cell potential value, one cannot increase the current or decrease the fuel flow rate without decreasing even more the cell potential and,

consequently, violating its constraint. Additionally, as mentioned before, an increase of the fuel flowrate would incur in suboptimality. And, a current decrease would remove the system from its power setpoint at 130 W. Similarly to 100 W, one could simultaneously increase the current and fuel flowrate while keeping the same fuel utilization, but as the current has a stronger impact on the cell potential, the cell potential constraint would be violated. And, if current and fuel flow rate are decreased, although the fuel utilization would remain the same, the equality constraint on the electrical power would be violated. Note that the current has a larger impact on the cell potential than the remaining optimization variables. Despite the air flowrate having lower effect on the cell potential compared to the fuel flow rate or the current, the air flowrate can cause cell potential changes. A reduction of the air flowrate causes a decay on the cell potential, which would, consequently, violate its lower bound. Whereas an air flowrate increase would cause a decrease of the system efficiency.

The above reasons suggest that the system is also locally optimal for the power of 130 W.

9.6 Conclusion

In this chapter, we performed experiments on a solid-oxide fuel cell system. We used Gaussian Processes (GPs) as a surrogate model. Amongst two performed experiments, in the first we validate our proposed idea to use GPs to correct the plant-model mismatch. Our proposed approach required one or two iterations to achieve the setpoint with high efficiency. In the second experiment, we proposed a new variant of Modifier Adaptation (MA) that computes modifiers analytically and therefore, avoids expensive computations of plant gradients. Despite continuous deterioration of the plant, the proposed approach achieved power setpoints with high efficiency.

Part III Concluding Remarks

10 Conclusions and Perspectives

> One never notices what has been done; one can only see what remains to be done.
>
> Marie Curie

10.1 Embedded Optimization on Programmable Hardware

In this part, we focused on deploying Model Predictive Control (MPC) on embedded platforms using operator splitting methods. We first analyzed the numerical properties of splitting methods and argued that they are appealing candidates for small to medium scale embedded control applications. Solving linear system and matrix-vector multiplications are the computational bottlenecks for splitting methods. In Chapter 2, we presented different ways of efficiently deploying splitting methods on processors. To that end, we also used various modern linear algebra packages. We then focused on deployment of MPC on Field Programmable Gate Arrays (FPGAs) and heterogeneous platforms in Chapter 3 and Chapter 4. First, various techniques were presented for efficient deployment on reconfigurable platforms and next we presented a code generating toolbox - SPLIT. By performing hardware-in-the-loop experiment, we showed that FPGAs outperform embedded processors and heterogeneous platforms in terms of execution time. In the more complex case of limited resources, we discussed how operations could be distributed between FPGAs and heterogeneous platforms. We also showed that, if a C program does not exploit various FPGA features, embedded processors outperform FPGAs in terms of execution time. Finally, we solved a co-design problem with a priori

guarantees, avoiding the laborious task of exploring the design space. We also developed a second toolbox - LAFF - having more general capabilities in terms of algorithms for FPGA targets.

There are numerous open challenges and opportunities when it comes to the deployment of control laws on embedded platforms. First and foremost, it would be desirable to make the deployment process even easier for FPGAs. Typically, even if all computations are performed in hardware, soft processors have to be instantiated to handle the communication tasks between the FPGA and the external world. Interfacing the two modules is far from trivial and still has to be done by the end user. Xilinx's tools SDSoC and SDAccel are steps in this direction, but only support a limited number of chips as of now. OpenCL is another promising language that could be explored for FPGAs, processors and heterogeneous. For the co-design problems, results obtained by exploring the design space should be compare with "roofline" modeling, which in the field of computer science is used to quantify the best "theoretically" possible performance for an algorithm. From a control perspective, comparing the Riccati recursion with linear solvers on FPGAs would also be interesting. Finally, we pose two theoretical questions that still need to be answered: How can one ensure that no overflow errors will occur when applying splitting methods? And is there a way of establishing the stability of splitting algorithms under arithmetic errors for general convex constraints?

10.2 Gaussian Processes Based Constrained Process Optimization

In the second part of the thesis, we combined three research areas: (i) Gaussian processes (GPs), (ii) real-time optimization (RTO) and (iii) derivative-free trust region methods. These topics were introduced in the context of process optimization in Chapter 5. In Chapter 6, we proposed a novel modifier adaptation technique where we learned the plant-model mismatch using GPs. We showed in a case study that our proposed approach converged faster and was more robust against noise compared with standard RTO techniques. In Chapter 7, we discussed probabilistic derivative-free trust-region method and proved that GPs enjoy the probabilistic full-linearity property. These proofs enabled GPs to be combined with probabilistic derivative-free trust-region methods with guarantees of global convergence. Since GPs are global surrogate models, fewer plant evaluations are required compare to local surrogate models. Consequently saving plant operation cost and coping with

changes in operating condition quickly. In Chapter 8 we proved that GPs are also deterministic fully linear, and hence can be coupled with deterministic derivative-free trust-region methods as well. We then pointed out a limitation of derivative-free trust-region methods and addressed two main challenges: frequent model checking and handling of noise. We proposed a novel algorithm – DMT – and showed its convergence to a neighborhood of an optimum. We finally showed superiority of GPs as global surrogate models by performing experiments on solid-oxide fuel cells. We converged to an optimum of different power setpoints within one or two iterations, outperforming standard RTO methods. In the second experiment, we proposed a variant of the modifier adaptation technique where computing plant gradients by perturbation is not required as GPs give the required gradient for computing modifiers.

There are at least two interesting future directions in the RTO domain related to this work. (i) Constrained derivative-free optimization: As discussed in Chapter 9, incorporating constraints in the penalty function is not ideal when process optimization. Therefore, it is necessary to develop robust derivative-free trust-region methods that can handle constraints in a more direct fashion. This is currently an active area of research and a good starting point is the work [152]. (ii) Introducing the ϵ-linear definition allowed us to handle noisy measurements. Nevertheless, further developing the field of derivative-free trust region methods under different noise models is still needed. Moreover, the use of other machine learning-based global surrogate models in RTO and derivative-free optimization is still to be investigated.

Bibliography

[1] G. Stathopoulos, H. A. Shukla, A. Szuecs, Y. Pu, and C. Jones, "Operator splitting methods in control," *Foundations and Trends in Systems and Control*, vol. 3, no. ARTICLE, pp. 249–362, 2016.

[2] H. A. Shukla, B. Khusainov, E. C. Kerrigan, and C. N. Jones, "Software and hardware code generation for predictive control using splitting methods," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 386–14 391, 2017.

[3] T. de Avila Ferreira, H. Shukla, T. Faulwasser, C. Jones, and D. Bonvin, "Real-time optimization of uncertain process systems via modifier adaptation and gaussian processes," *2018 European Control Conference*, pp. 465–470, 2018.

[4] E. Ullmann, "A Matlab toolbox for C-code generation for first order methods," Master's thesis, ETH Zurich, 2011.

[5] P. Giselsson and S. Boyd, "Linear convergence and metric selection for Douglas-Rachford splitting and ADMM," *arXiv e-prints*, p. arXiv:1410.8479, Oct. 2014.

[6] P. Giselsson and S. Boyd, "Metric selection in fast dual forward–backward splitting," *Automatica*, vol. 62, pp. 1–10, 2015.

[7] I. Necoara and A. Patrascu, "DuQuad: An inexact (augmented) dual first order algorithm for quadratic programming," *arXiv preprint arXiv:1504.05708*, 2015.

[8] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, pp. 1–36, 2020.

[9] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient interior point methods for multistage problems arising in receding

horizon control," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 668–674.

[10] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[11] H. J. Ferreau, H. G. Bock, and M. Diehl, "An online active set strategy to overcome the limitations of explicit MPC," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 18, no. 8, pp. 816–830, 2008.

[12] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[13] M. R. Hestenes, "Multiplier and gradient methods," *Journal of optimization theory and applications*, vol. 4, no. 5, pp. 303–320, 1969.

[14] L. D. Popov, "A modification of the arrow-hurwicz method for search of saddle points," *Mathematical notes of the Academy of Sciences of the USSR*, vol. 28, no. 5, pp. 845–848, 1980.

[15] K. J. Arrow, L. Hurwicz, and H. Uzawa, "Studies in linear and non-linear programming," *Stanford University Press*, 1958.

[16] J. Douglas and H. H. Rachford, "On the numerical solution of heat conduction problems in two and three space variables," *Transactions of the American mathematical Society*, vol. 82, no. 2, pp. 421–439, 1956.

[17] L. D. Popov, "A modification of the arrow-hurwicz method for search of saddle points," *Mathematical notes of the Academy of Sciences of the USSR*, vol. 28, no. 5, pp. 845–848, 1980.

[18] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & mathematics with applications*, vol. 2, no. 1, pp. 17–40, 1976.

[19] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., 1989.

[20] J. Eckstein and D. P. Bertsekas, "On the Douglas—Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 55, no. 1-3, pp. 293–318, 1992.

[21] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, 2011.

[22] P. L. Combettes and J.-C. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-point algorithms for inverse problems in science and engineering*. Springer New York, 2011, pp. 185–212.

[23] J. E. Esser, "Primal dual algorithms for convex models and applications to image restoration, registration and nonlocal inpainting," Ph.D. dissertation, University of California at Los Angeles, USA, 2010.

[24] H. H. Bauschke, P. L. Combettes *et al.*, *Convex analysis and monotone operator theory in Hilbert spaces*. Springer, 2011, vol. 408.

[25] S. Boyd, L. Xiao, A. Mutapcic, and J. Mattingley, "Notes on decomposition methods," *Notes for EE364B, Stanford University*, 2007.

[26] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations research*, vol. 8, no. 1, pp. 101–111, 1960.

[27] H. Everett III, "Generalized lagrange multiplier method for solving problems of optimum allocation of resources," *Operations research*, vol. 11, no. 3, pp. 399–417, 1963.

[28] N. Z. Shor, K. C. Kiwiel, and A. Ruszcayǹski, *Minimization Methods for Non-differentiable Functions*. Springer-Verlag New York, Inc., 1985.

[29] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.

[30] ——, *Constrained Optimization and Lagrange Multiplier Methods (Optimization and Neural Computation Series)*. Athena Scientific, 1996.

[31] R. T. Rockafellar, "Monotone operators and the proximal point algorithm," *SIAM journal on control and optimization*, vol. 14, no. 5, pp. 877–898, 1976.

[32] ——, "Augmented Lagrangians and applications of the proximal point algorithm in convex programming," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 97–116, 1976.

[33] P. Tseng, "Applications of a splitting algorithm to decomposition in convex programming and variational inequalities," *SIAM Journal on Control and Optimization*, vol. 29, no. 1, pp. 119–138, 1991.

[34] G. Chen and M. Teboulle, "A proximal-based decomposition method for convex minimization problems," *Mathematical Programming*, vol. 64, no. 1-3, pp. 81–101, 1994.

[35] H. Attouch, J. Bolte, P. Redont, and A. Soubeyran, "Alternating proximal algorithms for weakly coupled convex minimization problems. Applications to dynamical games and PDE's," *Journal of Convex Analysis*, vol. 15, no. 3, p. 485, 2008.

[36] M. Zhu and T. Chan, "An efficient primal-dual hybrid gradient algorithm for total variation image restoration," UCLA CAM Report, 2008.

[37] E. Esser, X. Zhang, and T. F. Chan, "A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science," *SIAM Journal on Imaging Sciences*, vol. 3, no. 4, pp. 1015–1046, 2010.

[38] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *Journal of mathematical imaging and vision*, vol. 40, no. 1, pp. 120–145, 2011.

[39] L. Condat, "A primal–dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms," *Journal of Optimization Theory and Applications*, vol. 158, no. 2, pp. 460–479, 2013.

[40] P. L. Combettes and J.-C. Pesquet, "Primal-dual splitting algorithm for solving inclusions with mixtures of composite, Lipschitzian, and parallel-sum type monotone operators," *Set-Valued and Variational Analysis*, vol. 20, no. 2, pp. 307–330, 2012.

[41] B. C. Vũ, "A splitting algorithm for dual monotone inclusions involving cocoercive operators," *Advances in Computational Mathematics*, vol. 38, no. 3, pp. 667–681, 2013.

[42] R. I. Bot, E. R. Csetnek, and A. Heinrich, "On the convergence rate improvement of a primal-dual splitting algorithm for solving monotone inclusion problems," *arXiv preprint arXiv:1303.2875*, 2013.

[43] B. O'Donoghue, G. Stathopoulos, and S. Boyd, "A splitting method for optimal control," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2432–2442, 2013.

[44] E. Ghadimi, A. Teixeira, I. Shames, and M. Johansson, "Optimal parameter selection for the alternating direction method of multipliers (ADMM): quadratic problems," *IEEE Transactions on Automatic Control*, vol. 60, no. 3, pp. 644–658, 2014.

[45] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 18–33, 2014.

[46] J. J. Moreau, "Fonctions convexes duales et points proximaux dans un espace hilbertien." *Comptes Rendus de l'Académie des Sciences (Paris), Série A*, vol. 255, 1962.

[47] J.-J. Moreau, "Proximité et dualité dans un espace hilbertien," *Bulletin de la Société mathématique de France*, vol. 93, pp. 273–299, 1965.

[48] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[49] L. Vandenberghe, "Optimization methods for large-scale systems," UCLA EE 236C lecture notes, 2010.

[50] B. He and X. Yuan, "Convergence analysis of primal-dual algorithms for a saddle-point problem: from contraction perspective," *SIAM Journal on Imaging Sciences*, vol. 5, no. 1, pp. 119–149, 2012.

[51] P. L. Combettes, L. Condat, J.-C. Pesquet, and B. Vũ, "A forward-backward view of some primal-dual optimization methods in image recovery," in *2014 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 4141–4145.

[52] R. Glowinski and P. L. Tallec, *Augmented Lagrangian and operator-splitting Methods in nonlinear mechanics*. Society for Industrial and Applied Mathematics, 1989.

[53] T. Goldstein and S. Osher, "The split bregman method for l1-regularized problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 2, pp. 323–343, 2009.

[54] A. Alessandro and A. Bemporad, *A survey on explicit model predictive control*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 345–369.

[55] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[56] Z. Liu, A. Hansson, and L. Vandenberghe, "Nuclear norm system identification with missing inputs and outputs," *Systems & Control Letters*, vol. 62, no. 8, pp. 605–612, 2013.

[57] G. G. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.

[58] D. S. Watkins, *Fundamentals of matrix computations*, ser. Pure and applied mathematics. New York: Wiley-Interscience, 2010.

[59] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[60] T. Davis, *Direct methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2006. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9780898718881

[61] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems," *Acta Numerica*, vol. 14, pp. 1–137, 5 2005.

[62] G. Frison, "Numerical methods for model predictive control," Master's thesis, Technical University of Denmark, DTU, Asmussens Alle, Building 305, DK-2800 Kgs. Lyngby, Denmark, 2012.

[63] G. Frison, H. H. B. Sørensen, B. Dammann, and J. B. Jørgensen, "High-performance small-scale solvers for linear model predictive control," in *2014 European Control Conference (ECC)*. IEEE, 2014, pp. 128–133.

[64] D. Axehill and M. Morari, "An alternative use of the Riccati recursion for efficient optimization," *Systems & Control Letters*, vol. 61, no. 1, pp. 37–40, 2012.

[65] G. Frison and J. B. Jørgensen, "Efficient implementation of the Riccati recursion for solving linear-quadratic control problems," in *2013 IEEE International Conference on Control Applications (CCA)*. IEEE, 2013, pp. 1117–1122.

[66] I. Nielsen, D. Ankelhed, and D. Axehill, "Low-rank modifications of Riccati factorizations with applications to model predictive control," in *52nd IEEE Conference on Decision and Control*. IEEE, 2013, pp. 3684–3690.

[67] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*, 1969, pp. 157–172.

[68] B. Khusainov, "Co-design of FPGA implementations for model predictive control," Ph.D. dissertation, Imperial College London, 2017.

[69] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, Oct 2016.

[70] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are We There Yet? A Study on the State of High-Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, pp. 898–911, 2019.

[71] N. Kapre and S. Bayliss, "Survey of domain-specific languages for FPGA computing," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–12.

[72] T. De Matteis, J. d. F. Licht, and T. Hoefler, "Fblas: streaming linear algebra on fpga," *arXiv preprint arXiv:1907.07929*, 2019.

[73] I. McInerney, G. A. Constantinides, and E. C. Kerrigan, "A survey of the implementation of linear model predictive control on FPGAs," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 381 – 387, 2018, 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018.

[74] S. Lucia, D. Navarro, O. Lucia, P. Zometa, and R. Findeisen, "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 137–145, Jan 2018.

[75] K. V. Ling, S. P. Yue, and J. M. Maciejowski, "A FPGA implementation of model predictive control," in *2006 American Control Conference*. IEEE, 2006, pp. 1930–1935.

[76] S. Saeidi, "FPGA-based nonlinear model predictive control of electric drives," Ph.D. dissertation, Technische Universität München, 2015.

[77] V. K. Singh, R. N. Tripathi, and T. Hanamoto, "HIL co-simulation of finite set-model predictive control using FPGA for a three-phase VSI system," *Energies*, vol. 11, no. 4, p. 909, 2018.

[78] B. Khusainov, E. Kerrigan, A. Suardi, and G. Constantinides, "Nonlinear predictive control on a heterogeneous computing platform," *Control Engineering Practice*, vol. 78, pp. 105–115, 2018.

[79] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, pp. 1–25, 2011.

[80] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for Fortran usage," *ACM Transactions on Mathematical Software*, vol. 5, no. 3, pp. 308–323, 1979.

[81] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' guide*. SIAM, 1999, vol. 9.

[82] L. Crockett, R. Elliot, and M. Enderwitz, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, 2014. [Online]. Available: https://books.google.com/books?id=9dfvoAEACAAJ

[83] A. Suardi, G. A. Constantinides, and E. C. Kerrigan, "Software development kit for FPGA: A fast FPGA prototyping tool for embedded optimization," in *European Control Conference*, 2015.

[84] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Glasgow, GBR: Strathclyde Academic Media, 2014.

[85] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Y. Young, and Z. Zhang, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2018, pp. 129–132.

[86] H. M. Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. M. P. Dinakarrao, H. Homayoun, and S. Rafatirad, "Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 397–403.

[87] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[88] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 18–33, Jan 2014.

[89] P. Milosavljevic, T. Faulwasser, A. Marchetti, and D. Bonvin, "Time-optimal path-following operation in the presence of uncertainty," in *Proc. of 15th European Control Conference (ECC)*, Aalborg, Denmark, June 29 - July 1 2016, pp. 2228–2233.

[90] S. Costello, G. François, and D. Bonvin, "Real-time optimizing control of an experimental crosswind power kite," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 2, pp. 507–522, 2018.

[91] M. Jonin, M. Singhal, S. Diwale, C. N. Jones, and D. Bonvin, "Active directional modifier adaptation with trust region-application to energy-harvesting kites," in *2018 European Control Conference (ECC)*. IEEE, 2018, pp. 2312–2317.

[92] G. A. Bunin, Z. Wuillemin, G. François, A. Nakajo, L. Tsikonis, and D. Bonvin, "Experimental real-time optimization of a solid oxide fuel cell stack via constraint adaptation," *Energy*, vol. 39, no. 1, pp. 54–62, 2012.

[93] T. de Avila Ferreira, Z. Wuillemin, T. Faulwasser, C. Salzmann, D. Bonvin *et al.*, "Enforcing optimal operation in solid-oxide fuel-cell systems," *Energy*, vol. 181, pp. 281–293, 2019.

[94] U. Rosolia and F. Borrelli, "Learning model predictive control for iterative tasks. a data-driven control framework," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2018.

[95] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[96] N. V. Sahinidis, "Optimization under uncertainty: State-of-the-art and opportunities," *Comput. Chem. Eng.*, vol. 28, no. 6, pp. 971 – 983, 2004.

[97] K. B. Ariyur and M. Krstic, *Real Time Optimization by Extremum Seeking Control*. New York, NY, USA: John Wiley & Sons, Inc., 2003.

[98] D. D., P. M., and M. Guay, "Extremum seeking control and its application to process and reaction systems: A survey," *Mathematics and Computers in Simulation*, vol. 82, no. 3, pp. 369 – 380, 2011, 6th Vienna International Conference on Mathematical Modelling.

[99] S. Engell, "Feedback control for optimal process operation," *Journal of Process Control*, vol. 17, pp. 203–219, 2007.

[100] A. Marchetti, G. François, T. Faulwasser, and D. Bonvin, "Modifier adaptation for real-time optimization – methods and applications," *Processes*, vol. 4, no. 55, pp. 1–35, 2016.

[101] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293, Jul 2013.

[102] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009.

[103] M. Ellis, H. Durand, and P. D. Christofides, "A tutorial review of economic model predictive control methods," *Journal of Process Control*, vol. 24, no. 8, pp. 1156–1178, 2014.

[104] N. V. Sahinidis, "Optimization under uncertainty: state-of-the-art and opportunities," *Computers & Chemical Engineering*, vol. 28, no. 6-7, pp. 971–983, 2004.

[105] C. Y. Chen and B. Joseph, "On-line optimization using a two-phase approach: An application study," *Industrial & engineering chemistry research*, vol. 26, no. 9, pp. 1924–1930, 1987.

[106] M. L. Darby, M. Nikolaou, J. Jones, and D. Nicholson, "RTO: An overview and assessment of current practice," *Journal of Process Control*, vol. 21, no. 6, pp. 874–884, 2011.

[107] S. S. Jang, B. Joseph, and H. Mukai, "On-line optimization of constrained multivariable chemical processes," *AIChE Journal*, vol. 33, no. 1, pp. 26–35, 1987.

[108] T. E. Marlin and A. N. Hrymak, "Real-time operations optimization of continuous processes," in *AIChE Symposium Series*, vol. 93, no. 316, 1997, pp. 156–164.

[109] A. G. Marchetti, B. Chachuat, and D. Bonvin, "Modifier-adaptation methodology for real-time optimization," *Industrial & Engineering Chemistry Research*, vol. 48, pp. 6022–6033, 2009.

[110] P. Tatjewski, "Iterative optimizing set-point control–the basic principle redesigned," *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 49–54, 2002.

[111] W. Gao and S. Engell, "Iterative set-point optimization of batch chromatography," *Computers & Chemical Engineering*, vol. 29, no. 6, pp. 1401–1409, 2005.

[112] P. D. Roberts, "An algorithm for steady-state system optimization and parameter estimation," *International Journal of Systems Science*, vol. 10, pp. 719–734, 1979.

[113] P. D. Roberts and T. W. C. Williams, "On an algorithm for combined system optimization and parameter estimation," *Automatica*, vol. 17, pp. 199–209, 1981.

[114] M. A. Brdyś and P. Tatjewski, *Iterative algorithms for multilayer optimizing control*. Imperial College Press, London UK, 2005.

[115] A. G. Marchetti, G. François, T. Faulwasser, and D. Bonvin, "Modifier adaptation for real-time optimization: methods and applications," *Processes*, vol. 4, no. 4, p. 55, 2016.

[116] G. A. Bunin, "On the equivalence between the modifier-adaptation and trust-region frameworks," *Computers & chemical engineering*, vol. 71, pp. 154–157, 2014.

[117] T. Faulwasser and D. Bonvin, "On the use of second-order modifiers for real-time optimization," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 7622–7628, 2014.

[118] W. Gao, R. Hernández, and S. Engell, "A study of explorative moves during modifier adaptation with quadratic approximation," *Processes*, vol. 4, no. 4, p. 45, 2016.

[119] W. Gao, S. Wenzel, and S. Engell, "A reliable modifier-adaptation strategy for real-time optimization," *Computers & Chemical Engineering*, vol. 91, pp. 318–328, 2016.

[120] A. S. Bandeira, K. Scheinberg, and L. N. Vicente, "Convergence of trust-region methods based on probabilistic models," *SIAM Journal on Optimization*, vol. 24, no. 3, pp. 1238–1264, 2014.

[121] J. Larson and S. C. Billups, "Stochastic derivative-free optimization using a trust region framework," *Computational Optimization and Applications*, vol. 64, no. 3, pp. 619–645, 2016.

[122] F. Augustin and Y. M. Marzouk, "NOWPAC: A provably convergent derivative-free nonlinear optimizer with path-augmented constraints," *arXiv e-prints*, p. arXiv:1403.1931, Mar 2014.

[123] S. Lucia and B. Karg, "A deep learning-based approach to robust nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 511–516, 2018.

[124] L. Hewing, A. Liniger, and M. N. Zeilinger, "Cautious NMPC with gaussian process dynamics for autonomous miniature race cars," *2018 European Control Conference (ECC)*, pp. 1341–1348, 2018.

[125] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[126] S. Shalev-Shwartz and N. Srebro, "SVM optimization: Inverse dependence on training set size," in *Proc. 25th Int. Conf. on Machine Learning*. New York, NY, USA: ACM, 2008, pp. 928–935.

[127] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of Global Optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[128] J. Kocijan, *Modelling and Control of Dynamic Systems using Gaussian Process Models*. Springer, 2016.

[129] F. Augustin and Y. M. Marzouk, "A trust-region method for derivative-free nonlinear constrained stochastic optimization," *ArXiv e-prints*, Mar. 2017.

[130] D. H. Jeong and J. M. Lee, "Enhancement of modifier adaptation scheme via feedforward decision maker using historical disturbance data and deep machine learning," *Computers & Chemical Engineering*, vol. 108, pp. 31–46, 2018.

[131] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[132] T. J. Williams and R. E. Otto, "A generalized chemical processing model for the investigation of computer control," *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 79, pp. 458–473, 1960.

[133] J. F. Forbes, T. E. Marlin, and J. F. MacGregor, "Model adequacy requirements for optimizing plant operations," *Computers & Chemical Engineering*, vol. 18, no. 6, pp. 497–510, 1994.

[134] Y. Zhang and J. F. Forbes, "Extended design cost, a performance criterion for real-time optimization systems," *Computers & Chemical Engineering*, vol. 24, no. 8, pp. 1829–1841, 2000.

[135] A. G. Marchetti, M. Singhal, T. Faulwasser, and D. Bonvin, "Modifier adaptation with guaranteed feasibility in the presence of gradient uncertainty," *Computers & Chemical Engineering*, vol. 98, pp. 61–69, 2017.

[136] L. T. Biegler, Y.-d. Lang, and W. Lin, "Multi-scale optimization for process systems engineering," *Computers & Chemical Engineering*, vol. 60, pp. 17–30, 2014.

[137] E. A. del Rio Chanona, J. E. Alves Graciano, E. Bradford, and B. Chachuat, "Modier-adaptation schemes employing Gaussian processes and trust regions for real-time optimization," *IFAC-PapersOnLine*, pp. 52–57, Apr. 2019.

[138] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, "Information-theoretic regret bounds for gaussian process optimization in the bandit setting," *IEEE Transactions on Information Theory*, vol. 58, no. 5, pp. 3250–3265, 2012.

[139] A. Lederer, J. Umlauft, and S. Hirche, "Uniform error bounds for gaussian process regression with application to safe control," *arXiv e-prints*, p. arXiv:1906.01376, Jun 2019.

[140] J. Dennis and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1996.

[141] J. Thomann and G. Eichfelder, "A trust-region algorithm for heterogeneous multiobjective optimization," *SIAM Journal on Optimization*, vol. 29, no. 2, pp. 1017–1047, 2019.

[142] B. Chachuat, B. Srinivasan, and D. Bonvin, "Adaptation strategies for real-time optimization," *Computers & Chemical Engineering*, vol. 33, no. 10, pp. 1557–1567, 2009.

[143] G. François, B. Srinivasan, and D. Bonvin, "Use of measurements for enforcing the necessary conditions of optimality in the presence of constraints and uncertainty," *J. of Process Control*, vol. 15, no. 6, pp. 701–712, 2005.

[144] K. Hashimoto, A. Saoud, M. Kishida, T. Ushio, and D. Dimarogonas, "Learning-based safe symbolic abstractions for nonlinear control systems," *arXiv e-prints*, p. arXiv:2004.01879, 2020.

[145] E. T. Maddalena, P. Scharnhorst, and C. N. Jones, "Deterministic error bounds for kernel-based learning techniques under bounded noise," *arXiv preprint arXiv:2008.04005*, 2020.

[146] T. Suzuki, "Fast generalization error bound of deep learning from a kernel perspective," in *International Conference on Artificial Intelligence and Statistics*, 2018, pp. 1397–1406.

[147] "Solidpower," https://www.solidpower.com/en/, accessed: 2020-09-29.

[148] "Solid oxide fuel cell," https://en.wikipedia.org/wiki/Solid_oxide_fuel_cell, accessed: 2020-09-29.

[149] G. A. Bunin, Z. Wuillemin, G. François, A. Nakajo, L. Tsikonis, and D. Bonvin, "Experimental real-time optimization of a solid oxide fuel cell stack via constraint adaptation," *Energy*, vol. 39, no. 1, pp. 54–62, 2012.

[150] T. de Avila Ferreira, Z. Wuillemin, A. Marchetti, C. Salzmann, J. Van Herle, and D. Bonvin, "Real-time optimization of an experimental solid-oxide fuel-cell system," *Journal of Power Sources*, vol. 429, pp. 168 – 179, 2019.

[151] A. Marchetti, A. Gopalakrishnan, B. Chachuat, D. Bonvin, L. Tsikonis, A. Nakajo, Z. Wuillemin, and J. Van herle, "Robust real-time optimization of

a solid oxide fuel cell stack," *Journal of Fuel Cell Sci. Technol.*, vol. 8, no. 5, 2011.

[152] J. P. Eason and L. T. Biegler, "A trust region filter method for glass box/black box optimization," *AIChE Journal*, vol. 62, no. 9, pp. 3124–3136, 2016. [Online]. Available: https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.15325

Harsh Shukla

MEC2-396, EPFL
1015 CH, Lausanne
Switzerland
+41762217689
harsh_a _shukla@ieee.org

Sa Vidya Ya Vimuktaye

Education

2014–2020 **Doctoral of Philosophy**, *École Polytechnique Fédérale de Lausanne* , Lausanne, Switzerland, *Electrical Engineering*.

2012–2014 **Master of Science**, *Delft University of Technology*, Delft, The Netherlands, *Systems and Control*.

2006–2010 **Bachelor of Technology**, *National Institute of Technology*, Surat, India, *Electrical Engineering*.

Experience

October 2017–February 2018 **Visiting Researcher**, *ABB Research Center*, Baden, Swtizerland.
Sponsored by a Marie-Curie Fellowship

Summer–2016/17 **Visiting Researcher**, *Imperial College London*, U.K.
Sponsored by a Marie-Curie Fellowship

2010–2012 **Project Engineer**, *Indian Institute of Technology*, Bombay, India.
Sponsored by Indian Space Research Organization and Department of Science & Technology

June–September 2009 **Summer Intern**, *Heinz Nixdorf Institute*, Paderborn, Germany.
Sponsored by DAAD (German Academic Exchange Service)

Skills

Programming Competences: C, MATLAB, Python, VHDL

Areas of Interest: Computational aspects in control, Optimization, Programming