Foundations and Trends® in Systems and Control
Vol. 3, No. 3 (2016) 249–362

DOI: 10.1561/2600000008

now
the essence of knowledge

Operator Splitting Methods in Control

Giorgos Stathopoulos
École Polytechnique Fédérale de Lausanne (EPFL)

Harsh Shukla
École Polytechnique Fédérale de Lausanne (EPFL)

Alexander Szűcs
Slovak University of Technology in Bratislava

Ye Pu
École Polytechnique Fédérale de Lausanne (EPFL)

Colin N. Jones
École Polytechnique Fédérale de Lausanne (EPFL)

Contents

1 Introduction **250**
1.1 Notation and Definitions 254

2 The Algorithms **255**
2.1 The dual problem 256
2.2 Proximal methods 258
2.3 Origin of the methods and a unified framework 260
2.4 Alternative interpretations 264
2.5 Relaxation 266
2.6 Termination 266
2.7 Discussion 267

3 Convergence Results and Accelerated Variants **270**
3.1 Sublinear convergence 272
3.2 Accelerated sublinear convergence 273
3.3 Linear convergence 280
3.4 Discussion 281

4 Stepsize Selection and Preconditioning **284**
4.1 Stepsize 284
4.2 Preconditioning 288
4.3 General functions 295

5 Numerical Linear Algebra **297**
5.1 Linear system solve 297
5.2 Matrix-Vector multiplication 307

6 Examples **312**
6.1 Aircraft control 312
6.2 The planetary soft landing problem 318
6.3 Building economic control 323

7 Summary **327**
7.1 When splitting should (and should not) be used 327
7.2 A rough guideline 329
7.3 Extensions and other directions 334

Acknowledgements **338**

Appendices **339**

A Definitions **340**

B Derivation of the Algorithms from Proximal Methods **343**

C Stopping Conditions **349**

References **351**

Abstract

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. More recently, the need for control of a process in the presence of a limited amout of hardware resources has triggered research in the direction of embedded optimization-based control. At the same time, and standing at the other side of the spectrum, the field of big data has emerged, seeking for solutions to problems that classical optimization algorithms are incapable to provide. This triggered some interest to revisit the family of first order methods commonly known as *decomposition schemes* or *operator splitting methods*. Although it is 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. Our purpose is to study the behavior of such algorithms as solvers of control-related problems of that scale. Our effort focuses on identifying special characteristics of these problems and how they can be exploited by some popular splitting methods.

G. Stathopoulos, H. Shukla, A. Szűcs, Y. Pu and C. N. Jones. *Operator Splitting Methods in Control*. Foundations and Trends® in Systems and Control, vol. 3, no. 3, pp. 249–362, 2016.

DOI: 10.1561/2600000008.

1

Introduction

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. More recently, the need for control of a process in the presence of a limited amout 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 [133], QPgen [57],[59], DuQuad [88]), *interior point (IP) methods* (FORCES [43], CVXGEN [84]) and *active set methods* (QPOASES [50]).

In this work we focus on systems with linear dynamics, giving rise to convex control problems. The purpose of the survey is to 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

$$\text{minimize} \quad f(z) + g(Lz) \ , \tag{1.1}$$

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

$$\begin{array}{ll} \text{minimize} & f(z) + g(y) \\ \text{subject to} & Lz = y \ , \end{array} \tag{1.2}$$

by alternatingly (or simultaneously) minimizing over y and z. Clearly, the solutions of problems (1.2) and (1.1) 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 $\mathcal{C}$

$$\delta_{\mathcal{C}}(z) = \begin{cases} 0 & z \in \mathcal{C} \\ \infty & \text{otherwise,} \end{cases} \tag{1.3}$$

which is the reason why both f and g are considered to be *extended-real-valued functions* (see [18, § 3.1.2]). Formulations similar to the above have been studied extensively and we can look for their roots in the method of multipliers [75], [110], the Arrow-Hurwicz method [3], Douglas-Rachford splitting [44] and ADMM [60], [55]. 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 [15] and [45]. 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, [17] and [27]), in cases where off-the-shelf interior point solvers simply fail due to the large dimensions involved. The thesis [47] provides a comprehensive description of the connection of several splitting algorithms under a common framework. Finally, the book [7] provides a mathematically rigorous introduction to operator splitting methods in general Hilbert spaces.

The plethora of different approaches for solving problem (1.2) 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 by several orders of magnitude general purpose solvers. 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 [15], [19], 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 [35], [49], [123], [14]), *augmented Lagrangian relaxations* [13], [117], [116], *alternating minimization (Gauss-Seidel) augmented Lagrangian schemes* (ADMM), mixture of Lagrangian with augmented Lagrangian schemes (AMA [131]), *linearized augmented Lagrangians* or *approximate minimization* schemes ([23], [4]) and, finally, *mixtures of alternating minimization with partial linearization* (PDHG [139], [48], [22], [30] and several similar primal-dual schemes [28], [134], [16]).

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 [102], [56], [57], [59], [105]. 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 special characteristics of these problems and how they can be exploited by some popular splitting methods. Some of the questions that we attempt to answer are:

1. It is very common in practice that optimal control problems come with a quadratic objective, since in this way stability can be proven for regulation or tracking purposes. What is the best way

to exploit this smooth term, along with the special structure of the dynamics equation?

2. Given that a control problem has to be solved repeatedly (*e.g.*, MPC), how does warm-starting of the solution affect the speed?

3. Given the structure of the problem at hand, which algorithms will converge more quickly?

4. Are there ways to precondition the problem in order to reduce the solve time?

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.

The paper is organized as follows: In Chapter 2 we formulate the problem we want to solve and look at it from three different perspectives, resulting in the three algorithms we use. Subsequently we introduce the algorithms under a unified scheme and report their properties. In Chapters 3 and 4 we build on the basic variants of the methods presented before, introduce several enhanced versions and focus on their applicability for solving optimization problems. More specifically, in Chapter 3 we review how one can exploit the structure of the problem to accelerate the theoretical convergence rates. In Chapter 4 we extend the discussion on acceleration to more practical schemes, *i.e.*, stepsize selection and preconditioning. We provide a comprehensive literature review of existing methods and we present generic preconditioned versions of the three algorithms. In Chapter 5 we discuss the computational aspects; we identify the bottlenecks in each method and propose ways to speed up the computation. In Chapter 6 we summarize the

observations that we have made and attempt to construct a guideline about how to choose a splitting scheme given a problem. Finally, the algorithms are illustrated with three examples in Chapter 6.

1.1 Notation and Definitions

Let $\mathcal{Z} = (\mathbb{R}^n, \langle \cdot, \cdot \rangle)$ be a Euclidean space equipped with the inner product $\langle z, x \rangle = z^\top x$ and the corresponding norm $\|z\| = \sqrt{\langle z, z \rangle}$. Symmetric n-dimensional matrices are denoted with $\mathbb{S}^n$, while positive (semi)definite matrices are denoted with $(\mathbb{S}^n_+)\mathbb{S}^n_{++}$. We also consider the scaled norm $\|z\|_P = \sqrt{\langle z, Pz \rangle}$, with $P \in \mathbb{S}_+$. The matrix norm of the linear operator $M \in \mathbb{R}^{m \times n}$ is defined as $\|M\| = \sup_{z \neq 0} \frac{\|Mz\|}{\|z\|}$. The minimum and maximum eigenvalue of a matrix $Q \in \mathbb{R}^{n \times n}$ are denoted by $\lambda_{\min}(Q)$ and $\lambda_{\max}(Q)$, respectively.

The domain of the extended-real-valued function f is defined as $\mathbf{dom}\, f = \{z \in \mathcal{Z} : f(z) < +\infty\}$ and f is proper if $\mathbf{dom}\, f \neq \emptyset$ and $f > -\infty$. The function f is closed if its epigraph $\mathbf{epi}\, f = \{(z, t) \in \mathbb{R}^n \times \mathbb{R} : f(z) \leq t\}$ is a closed nonempty convex set. The range of extended-real-valued functions is denoted with $\mathbb{R} \cup \{+\infty\} = \overline{\mathbb{R}}$. We denote the conjugate of a convex function with $f^\star$, while a minimizer is denoted by an asterisk, *i.e.*, $f(z^*) \leq f(z)\ \forall z \in \mathcal{Z}$. Finally, for succinctness in the notation, we denote the class of all proper, closed, convex functions from $\mathcal{Z}$ to $\overline{\mathbb{R}}$ with $\Gamma_0(\mathcal{Z})$.

The indicator function of a convex set $\mathcal{C}$ is denoted with $\delta_{\mathcal{C}}(\cdot)$. For the common norm balls the notation changes to $\delta_i(z, \alpha),\ i = 1, 2, \infty$, which denotes the constraint $\|z\|_i \leq \alpha$. Similar notation to the 2-norm ball is used for the second-order cone constraint, with the difference that the second argument is a scalar affine function itself, *i.e.*, $\delta_2(Ax+b, c^\top z+d)$ denotes the constraint $\|Az + b\|_2 \leq c^\top z + d$. The most common pairs of indicator functions with their conjugate representation are given in Table A.1.

2

The Algorithms

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

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

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 $b \in \mathbb{R}^m$. The following assumption holds:

Assumption 1. The functions $g_i : \mathbb{R}^{p_i} \to \overline{\mathbb{R}}$ are closed, proper, convex functions, *i.e.*, $g_i \in \Gamma_0(\mathbb{R}^{p_i})$.

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 (1.2) involves two functions in the objective, while in (P) we consider *two*

groups of functions. The first group contains two functions expressed as

$$f(z) := h(z) + \delta_{\mathcal{D}}(z) \ , \tag{2.1}$$

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

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

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 (1.2). 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.

2.1 The dual problem

We start by deriving the *Lagrangian* for (P), which 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. We use the concatenated

variables when it comes to derivations for lighter notation. We make the following standing assumtption:

Assumption 2. The Lagrangian (L) associated to (P) has a saddle point, *i.e.*,

$$\mathcal{L}(z^*, y^*; \lambda) \leq \mathcal{L}(z^*, y^*; \lambda^*) \leq \mathcal{L}(z, y; \lambda^*) \quad \forall z, y, \lambda \in \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^p \ .$$

The dual problem of (P) can be derived by means of the Lagrangian formulation. We have that

$$\begin{aligned} d(\lambda) &= \min_{z,y} \{f(z) + g(y) + \langle \lambda, Lz + l - y \rangle\} \\ &= -\max_{z} \left\{ -f(z) - \langle L^\top \lambda, z \rangle \right\} - \max_{y} \{-g(y) + \langle y, \lambda \rangle\} + \langle l, \lambda \rangle. \end{aligned} \tag{2.2}$$

Making use of the convex conjugate function (see Appendix A), the dual function can be expressed as

$$d(\lambda) = -f^\star(-L^\top \lambda) + \langle l, \lambda \rangle - g^\star(\lambda), \tag{D}$$

where $g^\star(\lambda) = \sum_{i=1}^{M} g_i^\star(\lambda_i)$ and $f^\star$ is the conjugate of the sum of the two functions $h + \delta_{\mathcal{D}}$. The conjugate of a convex function restricted in a subspace is well-defined and given in [76, Proposition 1.3.2]. The dual problem to solve becomes

$$\lambda^* = \operatorname*{argmin}_{\lambda} F(\lambda) + g^\star(\lambda) \ ,$$

where $F(\lambda) := f^\star(-L^\top \lambda) - \langle l, \lambda \rangle$.

Note that if a partial dualization with respect to y is performed in (2.2), we acquire the *saddle formulation* of (P)

$$\mathcal{S}(z; \lambda) = \langle Lz + l, \lambda \rangle + h(z) + \delta_{\mathcal{D}}(z) - g^\star(\lambda) \ . \tag{S}$$

The optimization procedure takes place in two steps; a proximal step involving the dynamics and a *linear approximation* of h, followed by a proximal step that involves the conjugate function $g^\star$.

In the algorithms considered here, all formulations (P), (S) and (D) play a significant role. Although we mostly deal with the primal problem (P), the saddle and dual counterparts are essential for drawing

a complete picture of the relations between the several methods. The derivation of the algorithms is a result of the application of several simple iterative schemes in order to minimize (D) or find a saddle point of (S). These schemes are based on the application of the proximal operator.

2.2 Proximal methods

For $f \in \Gamma_0(\mathbb{R}^n)$, its *proximal operator* $\mathbf{prox}_f : \mathbb{R}^n \to \mathbb{R}^n$ is defined as

$$\mathbf{prox}_f(x) := \operatorname*{argmin}_z \left\{ f(z) + (1/2)\|z - x\|^2 \right\} \ . \tag{2.3}$$

The proximal operator firstly appear in the seminal work by Moreau [86, 87]. 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 [104]. The course notes [135] also provide a detailed reference to the topic.

Proximal Minimization Algorithm (PMA) PMA is mostly a conceptual scheme for minimizing the function f described above by means of the **prox** operator. It is written as the iteration

$$z^{k+1} := \mathbf{prox}_{\rho^k f}(z^k) \ , \tag{2.4}$$

i.e., it minimizes f while not moving too far away from the previous minimizer. The distance to z^k is controlled by the sequence $\{\rho^k\}$. The algorithm converges for $\rho^k > 0$ and $\sum_{k=1}^{\infty} \rho^k = \infty$.

The method was introduced in [82, 83], while a more general form than the one presented here in [117]. Convergence properties have been further analyzed in [68]. Although it is applicable under mild assumptions, the PMA is only useful when the proximal operator of f is easy to compute.

Proximal Gradient Method (PGM) Consider the case that we want to minimize $l(z) = f(z) + g(z)$, $f \in \Gamma_0(\mathbb{R}^n)$ is *smooth* and $g \in \Gamma_0(\mathbb{R}^n)$. The proximal gradient method is the iteration

$$z^{k+1} := \mathbf{prox}_{\rho^k g}\left(z^k - \rho^k \nabla f(z^k)\right) \quad , \tag{2.5}$$

where $\rho^k > 0$ is a stepsize, either constant or determined by line search methods.

It is interesting to rewrite the method as

$$\begin{aligned} z^{k+1} &= \operatorname*{argmin}_z \left\{ g(z) + (1/2)\|z - (z^k - \rho^k \nabla f(z^k))\|^2 \right\} \\ &= \operatorname*{argmin}_z \left\{ g(z) + f(z^k) + \langle \nabla f(z^k), z - z^k \rangle + (1/2\rho^k)\|z - z^k\|_2^2 \right\} \quad . \end{aligned}$$

Consequently, the method minimizes the sum of the (possibly) nonsmooth function g and a *quadratic approximation* of the smooth function f centered at the previously computed optimizer z^k. If the stepsize is chosen to be fixed in the range $(0, 1/L_f]$, where L_f is a Lipschitz constant for ∇f, then the quadratic model upper bounds f around z^k and the method can be shown to converge [8]. In reality the method converges for any $\rho \in (0, 2/L_f)$, but the quadratic model does not necessarily act as an upper bound for stepsizes that are larger than $1/L_f$. Note that, under the extra assumption of smoothness of f, the algorithm can deal with the sum of two functions in the objective, in contrast to the PMA.

Forward-Backward Splitting (FBS) Although not necessarily a proximal method, the FBS algorithm is a generalization of the PGM. More generally, the FBS algorithm can be used to find a root of the sum of two operators A and B that satisfy specific properties [7, § 25.3] (special cases of which are subdifferentials, gradients, proximal and linear operators). Finding a minimizer of the sum of two convex functions can be cast as finding a root of the sum of two (monotone) operators, involving the gradient and subdifferential mappings of the functions. Monotone operator theory is a large and important field, with strong connections to convex optimization theory and algorithmic development [7, 119]. Its treatment goes beyond the scope of this manuscript.

We will, however, touch upon some aspects of it in order to derive one of the schemes presented in this work.

Douglas-Rachford Method (DRM) While PGM allows for dealing with the summation $f + g$ by 'linearizing' f, DRS decomposes the optimization problem and applies an alternating scheme, while dropping the assumption for smoothness of f. The algorithm is composed of a sequence of proximal steps, expressed as

$$\begin{aligned} v^{k+1} &= \mathbf{prox}_{\rho f}\left(\lambda^k - w^k\right) \\ \lambda^{k+1} &= \mathbf{prox}_{\rho g}\left(v^{k+1} + w^k\right) \\ w^{k+1} &= w^k + v^{k+1} - \lambda^{k+1} \end{aligned} \tag{2.6}$$

Here $\rho > 0$ is a stepsize, bearing no further restrictions, in constrast to PGM. The original motivation for the method was the decomposition of the 'difficult' proximal evaluation $\mathbf{prox}_{f+g}(x)$ of the PMA. The scheme dates back in the 50's [60], [55], while a more recent analysis in the framework of convex optimization is performed in [45].

2.3 Origin of the methods and a unified framework

We consider three approaches for solving (P), all based on decomposing the problem into simpler ones with respect to the f and g functions. A common trait of the resulting algorithms is that they are derived by applying the proximal methods presented in the previous section to the dual function (D). Although seemingly close, the algorithms presented below exhibit unique characteristics that can be handy when solving different optimization problems. The distinction mainly regards the easiness of the applicability of the algorithm, as well as the tradeoff between number of iterations and computational load per iteration. We will come back to this discussion in the end of the chapter.

Alternating minimization algorithm (AMA) [131] AMA derives from applying PGM to the dual problem (D), *i.e.*, when considering the smooth part of the objective being $f = F(\lambda)$ and $g = g^\star(\lambda)$ in (2.5). The equivalence is derived in Appendix B.1. This being the case, the

method requires smoothness of $F(\lambda)$, a property that can be recovered if and only if the function f in the original formulation (2.1) is *strongly convex* (see Appendix A, Lemma A.2). Finally, since convergence of the PGM comes under stepsize restrictions, AMA converges for $0 < \rho < \frac{2\sigma_f}{\|L\|^2}$, where σ_f is the strong convexity modulus of f.

Algorithm 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

From the perspective of the Lagrangian function, the first step of AMA is equivalent to the mininimization 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

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

Augmented Lagrangian functions have a long history in the optimization literature [116], [13]. 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 $\{y^k\}$.

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 an inversion. 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 [22, 73]. Later works expanded the previous ones to handle an extra (third) smooth function [28, 30], and even to deal with inexactness in the evaluation of the proximal steps [134]. 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. The proposed algorithm is in line with the recent scheme presented in [26].

Algorithm 2 Primal-Dual Algorithm (PDA)

Require: Initialize $\lambda^0 \in \mathbb{R}^p$, $z^0 \in \mathbb{R}^n$. 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: $\lambda_i^{k+1} = \mathbf{prox}_{\rho_i g_i^\star}\left(\lambda_i^k + \rho_i(L_i(2z^{k+1} - z^k) + l_i)\right),\ i = 1, \ldots, M$

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 (2.1). 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 comsposed of dual variable updates by means

of proximal operators that can be performed in parallel, as is the case for AMA. Using the Moreau identity (Lemma A.2 in Appendix A), we can express this step in terms of $\mathbf{prox}_{g_i/\rho_i}$, getting rid of the conjugate. PDA can be seen as the FBS applied to the saddle problem (S). The derivation is presented in Appendix B.

Alternating direction method of multiplier (ADMM) [60], [55], [61] This 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. It was rediscovered 30 years later under a new name: *split Bregman method* [64]. In the case of ADMM, the functions f and g need only be convex. Application of the PMA to the dual function (D) results in the minimization of a composite augmented Lagrangian, commonly known as the *method of multipliers* [75],[110],[13]. The difficulty to deal with the composite proximal operator, as discussed earlier, motivated research for a scheme that can split the proximal step into two. ADMM can be derived as a special case of DRM applied to the dual problem. The derivation is presented in Appendix B.

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

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 constrast 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 al-

lowed for a cheaper evaluation. This is not the case with AMA and PDA, where the first step remains simple.

2.4 Alternative interpretations

Proximal method of multipliers ADMM and PDA can be analyzed through the lens of Rockafellar's *Proximal method of multipliers* [116]. This approach is followed in the recent manuscript [122], which clarifies several aspects of popular splitting methods and shows their close connection.

For the purpose of analysis, the *proximal augmented Lagrangian* for (P) is defined as

$$\begin{aligned}\mathcal{P}_\rho(z,y;\lambda) =& f(z)+g(y)+\langle\lambda, Lz+l-y\rangle+\\ &\frac{\rho}{2}\|Lz+l-y\|^2+\frac{1}{2}\|z-z^k\|^2_{P_1}+\frac{1}{2}\|y-y^k\|^2_{P_2}\ ,\end{aligned} \tag{PAL}$$

where $P_1 \in \mathbb{S}^n_+$ and $P_2 \in \mathbb{S}^p_+$.

Applying the multiplier method to (PAL), one gets the Proximal method of multipliers (PMM) algorithm:

$$(z^{k+1}, y^{k+1}) \in \operatorname*{argmin}_{z,y}\ \mathcal{P}_\rho(z,y;\lambda) \tag{2.7}$$

$$\lambda^{k+1} = \lambda^k + \rho(Lz^{k+1}+l-y^{k+1}). \tag{2.8}$$

This is the most general form of the multiplier method [75],[110]. One observes that if (L) is considered instead of (PAL), the *dual decomposition* algorithm is recovered, while if one minimizes (AL) we recover the classical *method of multipliers*.

Alternating minimization of (2.7) over the variables z and y with proper choices of the matrices P_1 and P_2 gives rise to ADMM and Chambolle-Pock's method, a simplified version of PDA. The former is recovered for the choice $P_1 = 0$, $P_2 = 0$, while the latter by setting $P_1 = (1/\tau)I - \rho L^\top L$ and $P_2 = 0$. In this case, the matrix P_1 is chosen such that it simplifies the minimization with respect to z, and can be seen as a *partial linearization* of the objective. In general, the proximal terms are introduced in order to ensure strong convexity of the

minimization subproblems. This is important, *e.g.*, in the case that convergence of a variable to a unique optimizer is required (see discussion about convergence in Chapter 3).

Envelope functions As we have already seen, it is highly desirable to interpret the splitting methods presented above through simple algorithms, the properties of which are well-analyzed. Such an interpretation allows us to exploit this knowledge and improve upon the methods, as will be more clearly demonstrated in the following chapters. In the recent works [107] and [106], AMA and ADMM are shown to be equivalent to a gradient-type method applied to smooth functions inspired from the so called *Moreau envelope function*, defined as the *value function* of the proximal minimization problem (2.3):

$$f^{\rho}(x) := \inf_{z} \left\{ f(z) + (1/2\rho)\|z - x\|^2 \right\} \quad . \tag{2.9}$$

The envelope function f^{ρ} is convex and smooth with $1/\rho$ Lipschitz continuous gradient. Furthermore, the set of minimizers of f and f^{ρ} are the same, which practically means that the nonsmooth optimization problem involving f can be substituted by a smooth one involving f^{ρ}. Finally, the PMA can be seen as an application of the gradient method on f^{ρ}. The interested reader is referred to [104, § 3.1] for a more extended discussion.

Generalizing this idea, Patrinos et al. define the *forward-backward envelope (FBE)* for the composite optimization problem $\min . \; l(z) = f(z) + g(z)$ as:

$$l_{\rho}^{\mathrm{FB}}(x) := \inf_{z} \left\{ f(x) + \langle \nabla f(x), z - x \rangle + g(z) + (1/2\rho)\|z - x\|^2 \right\} \tag{2.10}$$

Although the FBE *is not necessarily convex*, the set of its stationary points turns out to be equal to the set of its minimizers for specific restrictions on ρ, which is in turn equal to the set of minimizers of the original composite function l. All the details are given in [107]. In addition, *AMA can be interpreted as a variable stepsize gradient method applied to the FBE of the dual problem* (D).

In a similar manner, a *Douglas-Rachford envelope (DRE)* is introduced in [106]. It turns out that the DRE is nothing but the FBE

evaluated at $\mathbf{prox}_{\rho f}(x)$ instead of x, *i.e.*, $l_\rho^{\mathrm{DR}}(x) = l_\rho^{\mathrm{FB}}(\mathbf{prox}_{\rho f}(x))$. If the smooth function f is also quadratic, the DRE is convex for $\rho \in (0, 1/L_f)$. This observation gives rise to new ways of analyzing and improving ADMM, expressed as *a variable stepsize gradient method applied to the DRE of the dual problem.* The reader can resort to [106] for a more detailed analysis.

Interpretation of the splitting schemes as an application of the gradient method to a smooth function has important practical implications. Smoothness is a highly desirable property that enables acceleration of the methods, either through application of optimal over-relaxation schemes, as they will be presented in Chapter 3, or through injection of second order information, which will be further discussed in §7.3.

2.5 Relaxation

Relaxation has been used to speed up the convergence of PMA in [45], giving rise to the iteration

$$z^{k+1} = (1 - \theta^k) z^k + \theta^k \, \mathbf{prox}_{\rho^k f}(z^k), \ \theta^k \in (0, 2) \ .$$

The idea roots back to *Successive Over-Relaxation for the solution of linear systems*, variations of which can be applied to any iterative method. It typically speeds up convergence for values $\theta^k > 1$, and it can be used in ADMM by substituting Lz^{k+1} with

$$\theta^k L z^{k+1} - (1 - \theta^k)(-y^k + l) \ .$$

The sequence $\{z^k\}$ is also over-relaxed in PDA where $\theta^k = 2$ is used at every iteration.

2.6 Termination

Given that problem (P) is convex, necessary and sufficient conditions for convergence are the *primal* and *dual feasibility conditions* (see, *e.g.*, [18, Chapter 5]). Writing the conditions for formulations (L) and (S), we get termination criteria for ADMM, AMA and PDA, respectively. The optimality conditions practically translate to primal and dual residuals

that (asymptotically) go to zero as the algorithmic schemes progress. In practice, the algorithms are terminated when the conditions are satisfied to some prespecified accuracy. In Appendix C we present in detail how the derivation is performed for each of the algorithms.

We should note that, traditionally, first order (splitting) methods do not provide information about the feasibility of the problem. In the case of infeasibility, the subproblems to which the original problem is split will not converge to a common solution, something that is reflected in the residuals that do not decrease. After having observed this behavior for some time, the user can terminate the algorithm and claim infeasibility. It is only recently that the authors of [101] suggested an ADMM scheme that also returns a feasibility certificate. The idea is to use *homogeneous self-dual embedding*, a method commonly used with interior-point methods [138], [137]. The original problem is written as a feasibility problem by embedding the KKT conditions into a system of linear equations. From the solution of the embedding problem, one can either recover the solution of the original one, or a certificate for primal or dual infeasibility. Infeasibility detection using ADMM to solve QPs is also treated in the recent work [112]. In this manuscript we do not consider these variants, which should be used if infeasibility detection is crucial for the problem at hand.

2.7 Discussion

A valid question that arises is: which method should be used given an optimal control problem? Throughout the course of this survey we will come back to this question and will attempt to give some guidelines for practitioners who want to use these methods as mere technology.

From a first look to Algorithms 1, 2 and 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 [59, 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

$$\begin{bmatrix} K_{11} & K_{21} \\ K_{21} & K_{22} \end{bmatrix} = \begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix}^{-1} . \tag{2.11}$$

   The KKT matrix in nonsingular, and hence the first step of AMA can be solved by means of a matrix inversion. 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 (2.11), with K_{11} diagonal, while ADMM would densify the matrix. Solving via the Schur complement would require inversion of K_{11} (see [18, 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

$$P_{\mathcal{D}}(p) = p + A^\top (AA^\top)^{-1}(b - Ap) \ , \tag{2.12}$$

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 4.

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.

3

Convergence Results and Accelerated Variants

Slow convergence is usually the case with first order methods, and the algorithms presented above are no exception to this rule. Slow convergence can be caused both by the primal or dual iterations. Indeed, the proximal step usually amounts to a projected gradient iteration, while the dual update is a gradient update step. In this aspect, the algorithms presented here cannot achieve a rate better than the existing *worst case* lower complexity bounds for first order methods [91], [93][1]

In what follows we frequently refer to convergence *in function values* and *in sequence values*. By saying that 'we have $O(1/k^q),\ q \in (0, 2]$ global rate of convergence in *function values* for some function f', we mean that

$$\lim_{k \to +\infty} k^q (f(z^k) - p^*) \leq M \ ,$$

where p^* is the optimal value of f and $M > 0$. Accordingly, 'global $O(1/k^q)$ convergence rate *of a sequence* $\{z^k\}$' means that

$$\lim_{k \to +\infty} k^q (\|z^k - z^*\|) \leq M \ ,$$

where z^* is the optimizer and $M > 0$.

[1] Nesterov's results point to the existence of a problem for which the algorithms cannot converge at a rate faster than the one presented in the aforementioned works. It is often (and hopefully) the case that for specific problem instances the algorithms behave much better than the lower complexity bounds.

Accordingly, we refer to convergence rate of $o(1/k^q)$ for a sequence $\{z^k\}$' (or in the function values of f) when

$$\lim_{k\to+\infty} k^q(\|z^k - z^*\|) = 0, \quad \lim_{k\to+\infty} k^q(f(z^k) - p^*) = 0 \ ,$$

respectively.

Intuitively, big-O says that 'the sequence in the parenthesis (function values or iterates values) *can decay no faster* than the sequence $\{1/k^q\}$', while the little-o convergence rate means that 'the sequence in the parenthesis *will decay strictly faster* than $\{1/k^q\}$', hence is a stronger statement.

Finally, *ergodic convergence rates* can be derived for the sequence and function values making use of the running average $\bar{z}^N = \frac{1}{N}\sum_{k=1}^N z^k$, so that

$$\|\bar{z}^N - z^*\| \leq \frac{M}{N^q}, \quad f(\bar{z}^N) - p^* \leq \frac{M}{N^q} \ ,$$

respectively.

As was shown in the previous chapter, the splitting methods we discuss derive from the application of proximal algorithms to the dual composite function of (P). Consequently, the convergence results for these splitting methods also derive from existing convergence results of the proximal algorithms we use. Common sense dictates that the more knowledge we have about the function at hand, the better the convergence rates we can derive. Accordingly, *structural assumptions* are crucial for converging faster to optimality. The most important ones are *smoothness* and *strong convexity*, with the underlying relation between them. The definitions of the two properties are given below, along with the lemma that associates them (3.1), and can be found in several convex analysis textbooks (see, *e.g.*, [7]).

Definition 3.1 (Smoothness)**.** A function $f \in \Gamma_0(\mathbb{R}^n)$ is β-smooth if it is differentiable in $\mathbb{R}^n$ and

$$f(z) \leq f(x) + \langle \nabla f(x), z - x\rangle + (\beta/2)\|z - x\|^2 \tag{3.1}$$

holds for all $z, x \in \mathbb{R}^n$.

Definition 3.2 (Strong convexity)**.** A function $f \in \Gamma_0(\mathbb{R}^n)$ is β-strongly convex if

$$f(z) \geq f(x) + \langle u, z - x \rangle + (\beta/2)\|z - x\|^2 \tag{3.2}$$

holds for all $z, x \in \mathbb{R}^n$ and all $u \in \partial f(x)$.

Definition 3.3 (Lipschitz continuous gradient)**.** A differentiable function $f \in \Gamma_0(\mathbb{R}^n)$ has a β-Lipschitz continuous gradient if

$$\|\nabla f(z) - \nabla f(x)\| \leq \beta\|z - x\| \tag{3.3}$$

holds for all $z, x \in \mathbb{R}^n$. β-Lipschitz continuous gradient is equivalent to β-smoothness of the function.

Lemma 3.1. Let $f \in \Gamma_0(\mathbb{R}^n)$. The following statements are equivalent:
i. The function f is β-strongly convex function.
ii. The conjugate function $f^\star \in \Gamma_0(\mathbb{R}^n)$ has $1/\beta$-Lipschitz continuous gradient.

3.1 Sublinear convergence

First order methods typically come with *sublinear convergence rates*, both in function values and in terms of the iterates. These rates are given from $q \in (0, 2]$ as presented in the previous section, depending on the structure of the problem at hand. Very roughly speaking, the following hold:

1. When *no structural assumptions on the functions are made*, sublinear convergence rates typically amount to an $o(1/k)$ global rate in function values.

2. Under the assumption of *smoothness*, several acceleration techniques can be applied, resulting in an $O(1/k^2)$ global rate in function values.

3. The convergence rate of the sequences of variables can usually be recovered as $O(1/\sqrt{k^q})$ from the function value's convergence rate.

4. Under the assumption of *smoothness and strong convexity*, linear convergence rates of the form $O(\omega^k),\ \omega \in (0,1)$ can be recovered.

Until very recently, almost all the convergence rate results regarding splitting methods, when no extra structural properties hold, were of big-O complexity. This landscape changed significantly with the works of Davis et al. [37, 36], where a faster rate of little-o complexity is proven to hold for the majority of the known splitting schemes. Since the derivation of convergence rates for decomposition schemes was (and is) a very active area of research, with a vast number of results, we will focus here *on the latest advancements, i.e., the fastest and tightest known rates that exist*. In addition, we will restrict ourselves to presenting *only nonergodic convergence rates*; the interested reader can find more information regarding ergodic rates in [122, 74, 37] for ADMM and [22, 36] for PDA.

More specifically, ADMM converges at a global rate of $o(1/k)$ if function values [36] and at $O(1/\sqrt{k})$ in the sequences of primal and dual variables, for an arbitrarily large stepsize ρ [122]. Since AMA is equivalent to PGM applied to the dual problem (see Appendix B), an $O(1/k)$ convergence rate in the dual function values is proven in [8, Theorem 3.1], enhanced to $o(1/k)$ in [36]. In the case of PDA, a pre-primal dual gap function is shown to shrink with rate $o(D/\sqrt{k})$, where D is a constant involving the diameter of all (bounded) subsets of the primal-dual space where the problem is defined (see [36] for the technical details).

3.2 Accelerated sublinear convergence

By making further assumptions on the function's structure, faster convergence rates can be recovered. In his work [109], Polyak proposed a way to speed up the gradient method, namely to use the modified update

$$\begin{aligned} \hat{z}^k &= z^k + \alpha^k (z^k - z^{k-1}) \\ z^{k+1} &= \hat{z}^k - \rho^k \nabla f(z^k)\ , \end{aligned}$$

on the smooth convex function f. This method is commonly known as the *heavy ball method*. In this way the next iterate depends on the last gradient update and the previous step $z^k - z^{k-1}$, which is called a *momentum sequence*. This seemingly small change of updating the new iterate as a linear combination of the two previous iterates greatly improves the performance of the original gradient scheme.

3.2.1 Nesterov acceleration

In his seminal paper [92], Nesterov modified the heavy ball method by simply evaluating the gradient at the extrapolated point $\hat{z}^k$ instead of z^k. In addition, he proposed a special formula for computing the relaxation sequence $\{\alpha^k\}$, resulting in an *optimal convergence rate for minimizing a smooth convex function using only gradient information*. The simple update formula is:

$$\begin{aligned} \alpha^k &= \left(1 + \sqrt{4(\alpha^{k-1})^2 + 1}\right)/2 \\ \hat{z}^k &= z^k + \frac{\alpha^{k-1} - 1}{\alpha^k}(z^k - z^{k-1}) \\ z^{k+1} &= \hat{z}^k - \rho^k \nabla f(\hat{z}^k) \ , \end{aligned} \tag{3.4}$$

with $\alpha^0 = 1$. Subsequently, Güler extended Nesterov's results for the PMA [69], while Beck and Teboulle extended it for the PGM [8]. Tseng [132] unified the analysis of fast PGM and proposed a condition for the acceleration sequence under which convergence is ensured. More specifically, the sequence $\{\alpha^k\}$ needs to satisfy

$$\frac{1 - \alpha^{k+1}}{(\alpha^{k+1})^2} \leq \frac{1}{(\alpha^k)^2} \ .$$

Application of such schemes results in an $O(1/k^2)$ global rate of convergence in function values; a rate that is *optimal for first order methods involving a smooth and a nonsmooth function*. Convergence in terms of the sequence was not proven until very recently [21], and the derivation of a rate is still open (unless further assumptions on the structure of the function are made). Apart from the theoretical results, the scheme has been observed to practically accelerate convergence in

numerous problem instances. In addition, the extra computational cost is insignificant.

FAMA The acceleration of AMA was derived for the dual objective in [10] as a direct dual application of the fast PGM (FPGM) derived for the composite nonsmooth + smooth minimization in [8], and popularized under the name FISTA. The derivation can be found in Appendix B. A rate of $O(1/k)$ of the primal sequence is also proven in [10]. The algorithm was also independently developed in [63] for slightly different stepsize conditions. FAMA can practically be applied to every problem that AMA can solve.

Algorithm 4 Fast alternating minimization algorithm (FAMA)

Require: Initialize $\alpha^0 = 1$, $\lambda^0 = \lambda^{-1} \in \mathbb{R}^p$, and $\rho \leq \frac{\sigma_f}{\|T\|^2}$

loop

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

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

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

4: $\alpha^{k+1} = \left(1 + \sqrt{4(\alpha^k)^2 + 1} \right) / 2$

5: $\hat{\lambda}_i^{k+1} = \lambda_i^k + ((\alpha^k - 1)/\alpha^{k+1})(\lambda_i^k - \lambda_i^{k-1}),\ i = 1, \ldots, M$

end loop

FADMM A fast version of ADMM (FADMM), based on Nesterov's acceleration, was first presented in [63]. The algorithm is presented below. In the case that f and all the functions $g_i,\ i = 1, \ldots, M$ are strongly convex, a global $O(1/k^2)$ rate is achieved in terms of dual function values. In the absence of assumptions no convergence rates can be proven. In addition, *convergence must be enforced by means of a restart rule*, as will be discussed in the next section.

Remark 3.1. Following the discussion from §2.4, the interpretation of the DR algorithm as a gradient scheme immediately allows for extensions to accelerated variants. Indeed, the authors in [106] propose an

Algorithm 5 Fast alternating direction method of multiplier (FADMM)

Require: Initialize $\hat{y}^0 = y^{-1} \in \mathbb{R}^p$, $\hat{\lambda}^0 = \lambda^{-1} \in \mathbb{R}^p$, $\rho > 0$, $\alpha^0 = 1$.

loop

1: $z^k = \underset{z}{\operatorname{argmin}} \quad f(z) + \sum_{i=1}^M \langle \hat{\lambda}_i^k, L_i z \rangle + \frac{\rho}{2} \sum_{i=1}^M \|L_i z + l_i - \hat{y}_i^k\|^2$

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

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

4: $\alpha^{k+1} = \left(1 + \sqrt{4(\alpha^k)^2 + 1} \right) / 2$

5: $\hat{y}_i^{k+1} = y_i^k + ((\alpha^k - 1)/\alpha^{k+1})(y_i^k - y_i^{k-1}),\ i = 1, \ldots, M$

6: $\hat{\lambda}_i^{k+1} = \lambda_i^k + ((\alpha^k - 1)/\alpha^{k+1})(\lambda_i^k - \lambda_i^{k-1}),\ i = 1, \ldots, M$

end loop

accelerated DR method using Nesterov's sequence. Application of the proposed algorithm to the dual Douglas-Rachford envelope (see §2.4) results in another FADMM which also achieves a global $O(1/k^2)$ rate is achieved in terms of dual function values. This version of FADMM holds only for quadratic f and comes with stepsize restrictions (remember that in this case the DRE is convex).

3.2.2 Adaptive restart

It is frequently the case that the momentum sequence generated from acceleration schemes can result in oscillatory behaviour in the convergence of the function value to the optimal one. The term $\frac{\alpha^k - 1}{\alpha^{k+1}}$ in (3.4) is the amount of required momentum; if this amount is underestimated convergence is slower, while in the opposite case it causes a rippling behaviour, leading again to slower convergence.

The authors in [100] propose a restarting scheme, namely performing a test every (few) iteration(s). Depending on the result of the test, the momentum term is set back to $\alpha^k = 1$ and the procedure is restarted. In order to demonstrate the scheme, consider again the accelerated version of the gradient method presented in (3.4). Two simple tests are provided; one ensuring that the function value keeps

decreasing (restart if $f(z^{k+1}) > f(z^k)$) and the other one checking whether the new direction is pointing towards the negative gradient (restart if $\langle \nabla f(\hat{z}^{k+1}), z^{k+1} - z^k \rangle > 0$). Although a heuristic, the scheme significantly accelerated the convergence of gradient and proximal gradient schemes it was initially applied to. The generalization to ADMM and AMA followed shortly after.

Since AMA is equivalent to PGM, restart can be immediately applied to its fast version, *i.e.*, Algorithm 11 in Appendix B. In order to avoid function evaluations, the use of the gradient-based restart test is preferred. Since the gradient iteration for Algorithm 11 is the proximal step $\lambda^{k+1} = \mathbf{prox}_{\rho g^\star}\left(\hat{\lambda}^k + \rho(Lz^k + l)\right)$, we can view it as a *generalized gradient scheme* [100] of the form:

$$\lambda^{k+1} := \hat{\lambda}^k + \rho G(\hat{\lambda}^k),$$

where $G(\hat{\lambda}^k)$ is the generalized gradient at $\hat{\lambda}^k$ (note that z^k is a function of $\hat{\lambda}^k$ from Step 2 of Algorithm 11). Consequently, the gradient-based restart test can be expressed as $\langle -G(\hat{\lambda}^k), \lambda^{k+1} - \lambda^k \rangle > 0$, and can be applied right after Step 3 of Algorithm 4

4: if $\langle \hat{\lambda}^k - \lambda^k, \lambda^k - \lambda^{k-1} \rangle > 0$
5: $\quad \hat{\lambda}^k = \lambda^{k-1}$
6: $\quad$ Repeat Steps 1,2,3 of Algorithm 4.
7: else
8: $\quad$ Go to Step 4.

Note that the algorithm cannot cycle since the restart condition will not be satisfied once $\hat{\lambda}^k = \lambda^{k-1}$. In the recent work [58], the restarted version of Algorithm 11, and, consequently, the restarted FAMA, is proven to *preserve the optimal* $O(1/k^2)$ *convergence rate* in function values.

Along the same lines, restart schemes have also been applied to ADMM in [63]. In this case, acceleration is applied to both sequences $\{y^k\}$ and $\{\lambda^k\}$. In order to ensure convergence, the combined residual

$c^k = \rho^{-1}\|\lambda^k - \hat{\lambda}^k\|^2 + \rho\|y^k - \hat{y}^k\|^2$ needs to vanish in the sense $\lim_{k\to\infty} c^k = 0$. The restart scheme can be plugged to Algorithm 5 after Step 4 and is presented below. The constant $\eta \in (0,1)$ corresponds to the decay rate of the residual term c^k.

5: $c^k = \rho^{-1}\|\lambda^k - \hat{\lambda}^k\|^2 + \rho\|y^k - \hat{y}^k\|^2$
6: if $c^k < \eta c^{k-1}$
7: $\hat{y}_i^{k+1} = y_i^k + ((\alpha^k - 1)/\alpha^{k+1})(y_i^k - y_i^{k-1}),\ i = 1, \ldots, M$
8: $\hat{\lambda}_i^{k+1} = \lambda_i^k + ((\alpha^k - 1)/\alpha^{k+1})(\lambda_i^k - \lambda_i^{k-1}),\ i = 1, \ldots, M$
9: else
10: $\alpha^{k+1} = 1$, $\hat{y}^{k+1} = y^k$, $\hat{\lambda}^{k+1} = \lambda^k$
11: $c^k \leftarrow \eta^{-1} c^{k-1}$
12: endif

Remark 3.2. It is possible that the restart scheme slows down the convergence of both FAMA and FADMM. Every time a restart happens, one full iteration of the algorithm goes to waste since the previous point is used instead of the over-relaxed one. In the worst-case scenario the algorithm will require *double the number of iterations* it would actually need if no restart was applied. Subsequently, restart methods will not necessarily improve on the fast versions of the two algorithms presented above.

FPDA The accelerated variant of PDA comes under the assumption that $f = h + \delta_{\mathcal{D}}$ is σ_f-strongly convex, denoted hereafter as F(ast)PDA (Algorithm 6). The acceleration is achieved by means of adaptive change of the primal and dual stepsizes τ and ρ_i. Furthermore, instead of taking a fixed momentum sequence $\{2(z^{k+1} - z^k)\}$ as in Algorithm 2, a *variable relaxation parameter* θ^k is introduced. The scheme results in a *global* $O(1/k)$ *convergence rate for the primal sequence* $\{z^k\}$, [16, Theorem 19]. A local $O(1/k^2)$ convergence in function values also applies, as proven in [22, Theorem 2] in the case that two functions comprise the objective of Algorithm 2.

Algorithm 6 Fast Primal-Dual Algorithm (FPDA)

Require: Initialize $\lambda^0 \in \mathbb{R}^p$, $z^0 \in \mathbb{R}^n$, $y^0 \in \mathbb{R}^m$. Choose stepsizes $\tau, \rho_1, \ldots, \rho_M > 0$ such that $\tau < 2\sigma_f / L_h$ $\mu \geq L_h + 1$, $\tau^0 \left(\sum_{i=1}^M \rho_i^0 \|L_i\|^2 \right) \leq \sqrt{1 + \tau^0 (2\sigma_f - L_h \tau^0)/\mu}$ and $\theta^0 = 1/\sqrt{1 + \tau^0 (2\sigma_f - L_h \tau^0)/\mu}$.

loop

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

2: $\hat{z}^{k+1} = z^{k+1} + \theta^k (z^{k+1} - z^k)$

3: $\lambda_i^{k+1} = \mathbf{prox}_{\rho^k g_i^\star} \left(\lambda_i^k + \rho_i^k (L_i \hat{z}^{k+1} + l_i) \right),\ i = 1, \ldots, M$

4: $\tau^{k+1} = \theta^k \tau^k$, $\theta^{k+1} = 1/\sqrt{1 + \tau^0 (2\sigma_f - L_h \tau^{k+1})/\mu}$,
$\rho_i^{k+1} = \rho_i^k / \theta^{k+1},\ i = 1, \ldots, M$

end loop

3.2.3 Smoothing

It is evident by now that smoothness is the key property that allows for accelerated convergence in first order splitting schemes. Since the three algorithms of interest are primal-dual methods, it comes as no surprise that strong convexity of the function f is needed in order to achieve the fast rate. This is a direct result of the celebrated dual relation between smoothness of the dual (primal) and strong convexity of the primal (dual) (see Lemma (3.1) in Appendix A).

Subsequently, the proximal regularization of a convex function, attributed to Moreau, was extended by Nesterov in the work [94]. More specifically, consider the class of nonsmooth convex functions that can be expressed in the following form:

$$q(x) = \max_{z \in \mathcal{Z}} \{ \langle z, Ax \rangle - \phi(z) \} \quad ,$$

for some $x \in \mathbb{R}^n$, where $A \in \mathbb{R}^{m \times n}$, $\mathcal{Z}$ is compact and convex and ϕ some continuous convex function in $\mathcal{Z}$. Nesterov defines a *prox-function* for the set $\mathcal{Z}$, denoted as D, as a σ_D-strongly convex and continuous function (over $\mathcal{Z}$), with $\mathcal{Z} \subseteq \mathbf{dom}\, D$, and its *prox center* as $z_0 = \underset{z \in \mathcal{Z}}{\operatorname{argmin}}\ D(z)$. Assuming that $D(z_0) = 0$, it holds that

$D(z) \geq (\sigma_D/2)\|z - z_0\|_2^2,\ \forall z \in \mathcal{Z}$. The smooth approximation of q is given by

$$q_\rho(x) = \max_{z \in \mathcal{Z}} \{\langle z, Ax\rangle - \phi(z) - \rho D(z)\} \quad .$$

As a result, q_ρ is $(\|A\|^2/\rho\sigma_D)$-smooth.

A useful discussion that provides an intuitive explanation for the method can be found in [96]. The convergence of the method is analyzed in [94]. Smoothing is extended to composite minimization problems in [95], as well as to more general prox-functions [9]. The authors of [9] unified smoothing methods and proved that, for any accelerated first order scheme applied to the smoothed function q_ρ, the original (nonsmooth) objective q will converge with rate $O(1/k)$ to its optimal value. Application of smoothing has also gained attention in the MPC community [90], as well as in solving continuous-time optimal control problems [41].

3.3 Linear convergence

The linear rate's advantage over the sublinear one is that, at least theoretically, any accuracy level can be achieved. *Linear convergence rates can be achieved by making use of the basic versions of the algorithms (Algorithms 1, 2, 3) with the extra assumption of strong convexity and smoothness of at least one of the functions in the objective [93, Chapter 2].* In other words, linear convergence emerges from *the structural properties* of the functions, if these properties are there. This stands in stark contrast to the accelerated versions of the algorithms presented above that require no extra assumptions, achieving the acceleration only by means of the injected relaxation sequences and variable stepsizes.

Linear convergence of ADMM has been proven for several problem formulations. In [40], the authors prove *global linear* $O(\omega^k)$, $\omega \in (0, 1)$ *convergence* under a variety of scenarios in which at least f is strongly convex and smooth. Tighter (and general) linear convergence bounds for the method have been recently presented in [57].

AMA also has a linearly convergent version under the extra assumption that *the dual function* $f^\star(-L^\top\lambda)$ *is strongly convex in the variable*

λ (see [131, Proposition 2]). In this scenario, *all primal and dual sequences* $\{z^k\}$ *and* $\{y^k\}, \{\lambda^k\}$ *converge to their optimizers linearly.* The assumption can be quite restrictive in control problems since L is (almost) always a tall matrix.

Chambolle and Pock propose a third algorithm in their paper, *which achieves global linear convergence of both the primal and dual sequences* [22, Theorem 3]. Strong convexity of the functions f and $g^\star$ is required as well as knowledge of the convexity modula σ_f and $\sigma_{g^\star}$. Strong convexity of the conjugate function translates to smoothness of the original ones (g_i), an assumption that is highly unlikely to hold in our problems of interest. The result is generalized in the case of $M > 1$ functions in the objective [16, Theorem 24].

In Table 3.1 we provide an up-to-date report of the existing convergence rates of the methods and their accelerated variants. Wherever a dash '-' appears, it means that there does not exist (or we are not aware of) such a result. Note that linear convergence always comes under additional assumptions. In some cases, there might be recent advancements that outperform the results presented here.

3.4 Discussion

The most interesting characteristic of the fast versions is that they require no further assumptions in order to work, both for the case of (F)AMA and (F)ADMM. Taking also into accout the insignificant computational cost that they incur renders them competitive alternatives to the original versions.

Following our discussion from § 2.7, we see that quadratic control problems with strongly convex objective (in the effective domain imposed by the constraints) are very common. Correspondingly, fast gradient methods are widely used in the MPC community [115]. FAMA is nothing but a way to apply fast gradient methods to problems where more complicated constraints appear, hence the recent interest in control [59].

	Stepsize restrictions	Strong convexity	Convergence: function values	Convergence: sequences
ADMM	no	no	$o(1/k)$ [37]	$O(1/\sqrt{k})$ [122], $O(\omega^k)$ [40, 81, 56, 98, 57]
AMA	yes	yes on f	$o(1/k)$ on the dual [37]	$O(1/\sqrt{k})$ on the primal [10], $O(\omega^k)$ [131]
PDA	yes	no	$o(D/\sqrt{k})$ in pre-primal dual gap [36]	$O(\omega^k)$ [22, 16]
FADMM	no	no	$O(1/k^2)$ on the dual under assumptions	-
FAMA	yes	yes on f	$O(1/k^2)$ on the dual [63, 10]	$O(1/k)$ on the primal [10]
FPDA	yes	yes on f	-	local $O(1/k)$ on the primal [22], global $O(1/k)$ on the primal [16]

Table 3.1: Convergence rate for the basic and accelerated versions of the three algorithms. The linear rates that appear are achieved under extra assumptions, discussed in the previous section of this chapter. It is important to mention that (F)ADMM is the only algorithm among the ones presented in this table that couples the z and the y variables in the z-update, *i.e.*, that introduces an augmented Lagrangian term.

We close the chapter with a comment on Table 3.1. Although the theoretical rates summarized there might seem of little practical importance, they should not be overlooked. It is remarkable how the theoretical proof for convergence of the gradient method at $O(1/k^2)$ motivated the development of many algorithms that make use of this result,

let alone the toolboxes developed for solving control problems (QPgen, DuQuad, FiOrdOs). The sole reason for this is that *accelerated variants, in practice, often work significantly better than the original versions.*

4

Stepsize Selection and Preconditioning

In this chapter we discuss a crucial issue that affects every first order scheme, namely how the limited information provided by a first order oracle can be optimally used in order to speed up the algorithmic progress. The discussion boils down to two topics: stepsize selection and conditioning of the problem. Although seemingly different, these two aspects are strongly related. We first explore how the stepsize can be selected for the three methods we have presented, moving from well-behaved functions with strong regularity properties to functions for which very little information is provided. We subsequently generalize the notion of the stepsize and show how to select the right metric, *i.e.*, the right space in which the problem should be solved. This is equivalent to preconditioning of the problem.

4.1 Stepsize

As in every first order method, the stepsize is crucial for the speed of the convergence. In augmented Lagrangian methods, where the stepsize (of the dual update) coincides with the penalty parameter and can be practically unbounded, the question of selecting a suitable one remains

open in the general case. We first provide a few results that have to do with optimally choosing the stepsize under some assumptions on the problem structure. We then move to the more general (and practical) case where we do not know much about the problem's structure and hence we adapt the stepsize according to the latest information we get from querying the function.

4.1.1 Optimal stepsize selection

The case where the first term of the composite objective in (2.1), f, is smooth and strongly convex, is well-studied and has strong results of linear convergence, as was discussed in §3.3. Based on these results, the stepsize parameters can be tuned so that the corresponding convergence rates are maximized. These computations, however, require knowledge of both the strong convexity constant (σ_f) and the smoothness constant (L_f) of the function. These quantities are, generally, difficult to determine. Thus, the majority of the authors treat the case of QPs, where both constants are associated to the extreme eigenvalues of the Hessian of the dual function (D).

In the case of ADMM, the authors of [56] determine the optimal stepsize ρ, having first proven a linear rate of convergence for inequality form, strongly convex QPs, and under an additional assumption that the constraint matrix is either full row rank or invertible.These requirements are relaxed in the recent works [113] and [57].

We mentioned is §3.3 that Chambolle and Pock's scheme also achieves a linear convergence rate for the sequences, provided that we have two functions in the objective (*i.e.*, $M = 1$) and that both of them are strongly convex. As is the case with ADMM, this is achieved by picking the optimal values for the primal and dual stepsizes τ and ρ as they appear in Algorithm 2. The optimal stepsizes are again chosen based on the convexity modula. The result is generalized in the case of multiple functions g_i in [16].

4.1.2 Practical stepsize selection

In most cases the problem does not have a favorable structure and/or there is an absence of information needed to optimally compute the

stepsize. Therefore, we resort to more practical schemes in order to speed up the convergence. Two approaches have been mostly followed in the literature to address this issue.

The first approach involves heuristic schemes that give rise to suboptimal fixed stepsizes. The procedures that are followed approximate the ones applied when the problem indeed has the desirable structural properties. Some knowledge of the geometry is still needed, restricting these methods mostly to QPs. Since in MPC and the associated problems of interest the existence of a quadratic term in the objective is common, we are interested in such structures and will return to them in the preconditioning section.

The second approach is adaptive updating of the stepsize(s) based on new information that becomes available as the algorithms iterate. This is commonly achieved by trying to guess the local structural properties of the involved functions either by direct function evaluations, or indirectly, via, *e.g.*, the residuals' evolution. As mentioned in §2.6, splitting algorithms converge when consensus to a common solution between the subproblems has been achieved. A good indication for this is the convergence of the primal and dual residuals r^k and s^k (Appendix C). Since the faster the residuals decrease the faster the termination of the algorithm, it intuitively makes sense to try and balance the residuals so that they converge at the same speed. Furthermore, computation of the residuals per iteration exhibits how they evolve, and thus it is sensible to try and 'correct' their behaviour if it is not desirable. This can be performed via the stepsizes.

AMA Although in the original version of AMA, as presented in [131], stepsize selection rules are not explicitly discussed, the equivalence of the method (and its accelerated version FAMA) with the PGM (and FPGM) algorithms (Algorithms 10 and 11 in Appendix B)allow for variable stepsizes. This is achieved by means of a *backtracking stepsize rule*. Backtracking is highly used in practical optimization for computing a suitable stepsize without having much knowledge about the structure of the problem [99, Chapter 3]. The approach was first developed in [8] for the ISTA and FISTA methods, and proceeds as follows:

First, recall from (2.5) that AMA is PGM applied to the negative dual problem. In the same section we have shown how PGM can be interpreted as a proximal step involving a *quadratic approximation* of the smooth part (F) of the composite objective. Consequently, a quadratic model that upper bounds the negative dual function can be constructed at a given point μ, denoted as $-\hat{d}(\lambda) = Q_{L_F}(\lambda, \mu)$, where

$$Q_{L_F}(\lambda, \mu) := F(\mu) + \langle \lambda - \mu, \nabla F(\mu) \rangle + \frac{L_F}{2} \|\lambda - \mu\|^2 + g^\star(\lambda) \ . \quad (4.1)$$

We denote the solution of (4.1) as

$$p_{L_F}(\mu) = \mathbf{prox}_{g^\star / L_F} \left(\mu - (1/L_F) \nabla F(\mu) \right) \ .$$

A sufficient condition for ISTA to converge is that $-d(p_{L_F}(\mu)) \leq Q_{L_F}(p_{L_F}(\mu), \mu)$ [8, Lemma 2.3], *i.e.*, the original function evaluated at the next iterate is below the corresponding value of the quadratic model. Subsequently, an *estimate of the local Lipschitz constant* $\bar{L}_F$ can be computed at every iteration by successively increasing $\bar{L}_F$, starting from a small estimate, until the condition is satisfied. The scheme can give rise to significantly smaller estimates of the Lipschitz constant compared to the conservative upper bound L_F, and, hence results in larger stepsizes $\rho^k = 1/\bar{L}_F^k$. The same backtracking rule applies to FISTA, and thus to the FAMA algorithm.

PDA Same as with ADMM, PDA's primal and dual residuals are inversely propotional to the (primal and dual) stepsizes τ and ρ_i, as indicated in (C.8). Consequently, one can achieve some residual balancing, and thus faster convergence, by adaptively choosing the stepsizes based on the latest residuals' update. The authors of [62] suggest checking a backtracking condition in order to guarantee convergence. In the case of PDA, convergence is based on ensuring that the primal and dual sequences move toward the solution set at every iteration, hence we have monotonicity of the sequences. The backtracking scheme reduces the stepsizes when this monotonic decrease is about to be violated. It is quite useful in practice since it often leads to long stepsizes that violate the stability conditions, exactly as with AMA. The full details can be found in [62].

ADMM Observing the ADMM iterates (Algorithm 3), one easily identifies that the dual stepsize or penalty parameter ρ is nothing but a weight on the penalization of the primal feasibility condition $y - Lz - l$, *i.e.*, a weight on the primal residual r^k (C.10). Consequently, by monitoring the primal residual one can either increase ρ when it gets large or decrease it when it is small compared to s^k. This is exactly the scheme proposed in [72], that reads as follows:

$$\rho^{k+1} = \begin{cases} 2\rho^k & \|r^k\| > c\|s^k\|, \\ 0.5\rho^k & \|s^k\| > \|r^k\|/c, \\ \rho^k & \text{otherwise} \end{cases} \tag{4.2}$$

with $c > 1$. The numbers suggested above are indicative and can be scaled according to the problem. This adaptive stepsize scheme is proven to converge and can frequently reduce the required number of iterations in practical applications. It is, however, a fact, that the absence of restrictions in the penalty parameter is often a major drawback in ADMM in comparison to the other splitting schemes, leading to extensive offline tuning until a sensible stepsize is calculated.

4.2 Preconditioning

The most pronounced weakness of general first order methods is their inability to efficiently deal with ill-conditioned problems. There are two ways to deal with bad conditioning: Either to a) choose a new coordinate system that adjusts better to the geometry of the problem or b) to cast the problem in a different form such that its geometry becomes more favorable, and then apply the algorithms to solve the recast problem.

Since strong duality allows us to treat a problem in both its primal and dual forms, we have the option to alter the geometry of both spaces. Consider the original problem (P). *Primal preconditioning* is equivalent to left preconditioning of the primal variables, where *dual preconditioning* is equivalent to left preconditioning of the constraints.

Consider the scaling variables $z_p = Dz$, $y^d = Ey$, with $D \in \mathbb{R}^{n\times n}$ and $E \in \mathbb{R}^{p\times p}$, with $E = \mathbf{diag}(E_1, \ldots, E_{M+1})$ and $E_i \in \mathbb{R}^{p_i \times p_i}$, $i = 1, \ldots, M$. The sub(super)scripts p and d indicate scaled variables from

the primal or from the dual preconditioning matrices. Problem (P) is then transformed to

$$\begin{array}{ll} \text{minimize} & f(D^{-1}z_p) + \sum_{i=1}^{M} g_i(E_i^{-1}y_i^d) \\ \text{subject to} & ELD^{-1}z_p + El = y^d \ , \end{array} \tag{4.3}$$

or, by setting $(\cdot)_p = D(\cdot)$, $(\cdot)^d = E(\cdot)$ and $(\cdot)_p^d = E(\cdot)D^{-1}$, the problem can be expressed in its preconditioned version as

$$\begin{array}{ll} \text{minimize} & f(D^{-1}z_p) + \sum_{i=1}^{M} g_i(E_i^{-1}y_i^d) \\ \text{subject to} & L_p^d z_p + l^d = y^d \ , \end{array} \tag{PrecP}$$

with variables z_p and y^d.

Remark 4.1. Although the use of the word 'preconditioning' generally refers to multiplication with *any* matrix, while 'scaling' refers to preconditioning with diagonal matrices, in this context we use both words to refer to any general preconditioner. If the matrix has some special structure, this is explicitly mentioned.

There are two points to consider before casting the original problem to (PrecP).

1. The preconditioned version should be faster to solve in terms of iterations.

2. The subproblems should not become computationally too costly to solve.

In practice, we look for a reformulation that is, in total, cheaper to solve than the original one. This requirement already poses serious restrictions on the structure of the scaling matrices D and E. Furthermore, the most important question that arises is *how to choose a good preconditioner*. As we will see, a common way is to optimize the known convergence rate of an algorithm equivalent to the splitting scheme of interest.

4.2.1 Preconditioners under structural assumptions

Same as with the stepsize, knowledge of the structure of the function f enables the computation of (in many cases optimal) preconditioners. In order to generalize the notion of the stepsize, we need to extend the notions of smoothness and strong convexity, given in Definitions 3.1 and 3.2. The idea is that if there exist positive definite matrices M and H such that the first composite term of the negative dual function (D), *i.e.*, the function $F(\lambda) = f^\star(-L^\top\lambda) - \langle l, \lambda\rangle$, satisfies the following two properties:

$$\begin{aligned} &\text{The function } F - (1/2)\|\cdot\|_H^2 \text{ is convex.} \\ &\text{The function } (1/2)\|\cdot\|_M^2 - F \text{ is convex,} \end{aligned} \tag{4.4}$$

then F can be lower and upper-bounded by quadratic functions. Manipulation of the level sets of the bounding functions so that they resemble those of F allows for faster convergence to the solution of the original problem. Structurally, properties (4.4) imply that F is $\lambda_{\min}(H)$-strongly convex and $\lambda_{\max}(M)$-smooth.

AMA Applying AMA to problem (PrecP) we recover the preconditioned version of AMA, denoted as PrecAMA, which reads as follows:

Algorithm 7 Preconditioned Alternating Minimization Algorithm (PrecAMA)

Require: Initialize $\lambda^0 \in \mathbb{R}^p$, $\|EL\Sigma_f^{-1}L^\top E^\top\| = 1$.

loop

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

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

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

end loop

The matrix $E = \mathbf{diag}(E_i),\ i = 1, \ldots, M$ is the constraint preconditioner mentioned above, while $P = E^\top E$. There are several interesting observations to be made about Algorithm 7. Firstly, the stepsize ρ has been absorbed by the preconditioner. The initial restriction on the

stepsize in the basic version of AMA is now translated into the condition involving the matrix norm, where $f - \|\cdot\|^2_{\Sigma_f}$ is convex, for some $\Sigma_f \succ 0$. In that case, smoothness of F can be determined using the dual relation between smoothness and strong convexity (Lemma 3.1), resulting in F being $\lambda_{\min}(L\Sigma_f^{-1}L^\top)$-smooth. Furthermore, the primal update in Step 1 can be expressed in the original variable z, since AMA is invariant when scaling the primal variables. The proximal step, however, changes significantly. In order to write it, we need to introduce the *generalized proximal operator* for a function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$, defined as

$$\mathbf{prox}^P_{\rho f}(x) := \inf_z \left\{ f(z) + \frac{1}{2\rho}\|z - x\|^2_P \right\} \quad . \tag{4.5}$$

If P is not diagonal, the proximal step is not separable down to the component anymore, as is the case with several functions g_i (*e.g.*, indicator for nonnnegative orthant, l_1 norm). Consequently, the proximal step cannot be solved efficiently. Even in the case of diagonal scaling, the closed form representation of the proximal step can be ruined. Table 4.1 at the end of the chapter illustrates some common cases of proximal operators and how diagonal scaling affects their representation. Finally, note that the unscaled variable y does not appear in any of the updates, conveniently allowing us to work directly with the scaled version y^d without making any conversions.

Next, we discuss how to choose the matrix E. The motivation comes from the convergence properties of the PGM applied to the dual problem (D) (see Chapter 2 and Appendix B). The approach is discussed in detail in the recent work [59]. Starting from the equivalence of the two algorithms, the analysis is performed on the iteration $\lambda_i^{k+1} = \mathbf{prox}_{\rho g_i^\star}\left(\lambda_i^k + \rho(L_i z^{k+1} + l_i)\right)$ (Step 2, Algorithm 10). Note that this amounts to a simple proximal iteration in the direction of the negative gradient of F, and consider the stepsize to be $\rho = 1/L_F$. As mentioned before, this is equivalent to the minimization of the quadratic upper bound (4.1) of $-d(\lambda)$, $Q_L(\lambda, \mu) = F(\mu) + \langle \lambda - \mu, \nabla F(\mu)\rangle + \frac{L_F}{2}\|\lambda - \mu\|^2 + g^\star(\lambda)$, repeated here for clarity. The quadratic model has uniform curvature L_F in all directions, being a poor approximation of the original composite dual function in the case that F is ill-conditioned. It is shown in [59, Proposi-

tions 9, 10] that F is actually well approximated by the quadratic model $F(\mu)+\langle\lambda-\mu,\nabla F(\mu)\rangle+\frac{1}{2}\|\lambda-\mu\|_P^2$, where $P^{-1}=L\Sigma_f^{-1}L^\top$. In this way, the quadratic model is scaled to better approximate the level curves of F. This bound is tight for many functions, however it results in complicated proximal operators. It is thus preferable to approximate with some diagonal P such that $P^{-1}\approx L\Sigma_f^{-1}L^\top$, and $P^{-1}\succeq L\Sigma_f^{-1}L^\top$ for guaranteeing convergence. Rewriting $P=E^\top E$, the convergence condition is expressed as $I\succeq EL\Sigma_f^{-1}L^\top E^\top$, while the quadratic model can be refined by minimizing its condition number, *i.e.*, $\frac{\lambda_{\max}(EL\Sigma_f^{-1}L^\top E^\top)}{\lambda_{\min}(EL\Sigma_f^{-1}L^\top E^\top)}$. Small modifications have to be performed in the (common) case when $L\Sigma_f^{-1}L^\top$ is singular, but, ultimately, the problem can be cast as an SDP, or solved by means of several heuristics [59, § 6].

ADMM The scaled version of the algorithm reads as follows:

Algorithm 8 Preconditioned Alternating Direction Method of Multiplier (PrecADMM)

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

loop

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

2: $(y_i^d)^{k+1}=E_i\,\mathbf{prox}_{(1/\rho)g_i}^{P_i}\left(E_i^{-1}(L_i^dz^{k+1}+l_i^d+\lambda_i^k/\rho)\right),\quad i=1,\ldots,M$

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

end loop

The algorithm is quite similar to PrecAMA, except for the addition of the augmented Lagrangian term in the objective.

Same as with AMA, the purpose is to compute an optimal preconditioner E based on the properties of the dual function (D). Ignore for a moment the dynamics equation from the function f, *i.e.*, resulting in $f(z)=\frac{1}{2}z^\top Qz+c^\top z$, and assume that the matrix L is full row rank. The properties of the dual term F can be determined from the properties of the primal objective f. Under the assumption that $f-\|\cdot\|_{\Sigma_f}^2$ is convex

and $\|\cdot\|^2_{M_f} - f$ is convex, it holds that F is $\lambda_{\min}(LM_f^{-1}L^\top)$-strongly convex and $\lambda_{\max}(L\Sigma_f^{-1}L^\top)$-smooth (see also [57, Proposition 8]). In addition, application of the Douglas-Rachford algorithm to (D) with the function F having these properties results in a linear convergence rate of the dual sequence that improves with the ratio $\frac{\lambda_{\max}(L\Sigma_f^{-1}L^\top)}{\lambda_{\min}(LM_f^{-1}L^\top)}$ [57, Proposition 6]. Note that the above expression corresponds to some notion of condition number, formulated as the ratio of the maximum eigenvalue of the quadratic upper bound to the minimum eigenvalue of the quadratic lower bound for the function F. Consider now the preconditioned version of the problem, *i.e.*, (PrecP), where only the constraints are scaled with E, while the primal variables remain unscaled. The new condition number for the scaled problem becomes $\frac{\lambda_{\max}(EL\Sigma_f^{-1}L^\top E^\top)}{\lambda_{\min}(ELM_f^{-1}L^\top E^\top)}$. Consequently, we can freely choose the matrix E so that the ratio becomes one. However, choosing any positive definite matrix would result in a complicated proximal step (Step 2) in Algorithm 8, as discussed before. This is why E is typically set to a diagonal matrix. The matrix is commonly determined by solving an SDP (see [56], [57]) or by means of (suboptimal) heuristic schemes if the SDP is too expensive to solve [59]. Note that the preconditioning problem is solved once and offline.

Unfortunately, the indicator function for the dynamics $\delta_{\mathcal{D}}$ results in loss of smoothness for f. Furthermore, in MPC problems we typically encounter L to be a tall, full column rank matrix. In these cases there are several heuristics for determining a matrix E that does its best in scaling the problem given the restrictions. These schemes are analyzed in the works [56] and [57].

PDA In the case of the PDA, the philisophy behind the derivation of preconditioners is quite different from that followed with AMA and ADMM. The results appearing in the literature derive mostly from the fact that PDA can be written as the application of the PMA (Chapter 2) to a variational inequality involving the optimality conditions of (S) [73]. However, preconditioning can be intuitively interpreted in the same way as before, *i.e.*, through the construction of quadratic

approximations for the saddle function (S), followed by tuning of the curvature in different directions. The scaled version of the algorithm is:

Algorithm 9 Preconditioned Primal-Dual Algorithm (PrecPDA)

Require: Initialize $\lambda^0 \in \mathbb{R}^p$, $z^0 \in \mathbb{R}^n$. Choose T, P such that $\sqrt{\sum_{i=1}^{M} \|P_i^{1/2} L T^{-1/2}\|^2} < 1 - (L_h/2)\|T\|$.

loop

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

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

end loop

The matrices T and P are related to the preconditioners D and E via the equations $T = (D^\top D)^{-1}$ and $P_i = E_i^\top E_i,\ i = 1, \ldots, M$.

The preconditioned version of PDA is a direct generalization of the work [108] with $M = 1$. The stepsizes τ and ρ of the original version (Algorithm 2) are now picked to be different along the dimensions of both the primal and the dual problem. This can be easily shown if we take a close look at the iterations of Algorithm 9. The first iteration can be expressed as

$$z^{k+1} = \underset{z}{\operatorname{argmin}} \quad \delta_{\mathcal{D}}(z) + \langle z, Qz^k + c + L^\top (\lambda^d)^k \rangle + \frac{1}{2}\|z - z^k\|_{T^{-1}} \ .$$

For fixed λ^d, this step corresponds to the minimization of the convex part of the saddle function (S), using a linear approximation of h and regularized by a quadratic term with different curvature in each direction. Note that, in a manner similar to AMA, the minimization problem involves a quadratic upper bound of the original function. Picking the curvature (inverse stepsize) of the proximal term results in shaping the level sets of the original problem. Along the same lines, the second iteration

$$(\lambda_i^d)^{k+1} = \underset{\lambda_i^d}{\operatorname{argmin}} \quad g_i^\star(\lambda^d) - \langle \lambda^d, L(2z^{k+1} - z^k) + l \rangle + \frac{1}{2}\|\lambda_i^d - (\lambda_i^d)^k\|_{P_i^{-1}}$$

involves the maximization of a quadratic lower bound of the concave

part of the saddle function (S). The level curves are shaped through P_i.

Convergence of Algorithm 9 follows as long as the condition $\sqrt{\sum_{i=1}^{M} \|P_i^{1/2} L T^{-1/2}\|^2} < 1 - (L_h/2)\|T\|$ holds. This is a sufficient condition that proves convergence of PDA, initially derived in [134] and [30]. In the preconditioned version developed in [108], where $h = 0$, the condition becomes less conservative (allows for larger stepsizes).

4.3 General functions

In the previous section we focused on strongly convex and smooth functions f so that the smooth part of the optimization problem is nicely approximated by quadratic functions. In the general case, though, such a property might be missing, hence the techniques for deriving preconditioners that were discussed before are not applicable.

When this is the case, is has been practically observed that improving the conditioning of the L matrix speeds up the convergence of the corresponding algorithms [24], [51], [59], [108]. This is typically done by means of *equilibration*, *i.e.*, by choosing E and D^{-1} in (PrecP) so that the rows and columns of the equilibrated ELD^{-1} matrix have (approximately) the same norm. There is a variety of heuristic methods that perform equilibration, *e.g.*, [124], [20] and the impact on the convergence speed of the methods can be significant.

Remark 4.2. Since preconditioning only affects the data of the optimization problem, it is obvious that we can combine it with the acceleration techniques discussed previously without any further modifications. Consequently, both adaptive stepsize rules and Nesterov acceleration (with restart) can be blended into one algorithm that solves the preconditioned problem (PrecP). In the recent work [65], PGM is combined with all the above to give an improved version of the algorithm. Once applied to the dual problem (D), the proposed algorithm results in a preconditioned FAMA (with possibly variable stepsize).

		Non-Preconditioned		Preconditioned	
		Fully Separable	Closed Form	Fully Separable	Closed Form
Orthant	Positive / Negative	✓	✓	✓	✓
Second-order cone		✗	✓	✗	✗
Norm balls	l_∞	✓	✓	✓	✓
	l_1	✗	✗	✗	✗
	l_2	✗	✓	✗	✗
Norms	l_∞	✗	✗	✗	✗
	l_1	✓	✓	✓	✓
	l_2	✗	✓	✗	✗

Table 4.1: Properties of the generalized prox operator for a diagonal metric. Preconditioning cannot be applied to problems with conic constraints, other than the orthant, without increasing the computational complexity of the proximal step.

5

Numerical Linear Algebra

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 (and their variants) 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.

5.1 Linear system solve

The requirement for solving a linear system arises in the z-minimization step of AMA and ADMM (Algorithms 1, 3 and variants). 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 [99, 18].

AMA Recalling the Lagrangian definition (L), the first step of Algorithm 1 reads:

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

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

$$\begin{bmatrix} Q & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} z \\ \nu \end{bmatrix} = \begin{bmatrix} -c - \sum_{i=1}^{M} L_i^\top \lambda_i^k \\ b \end{bmatrix} \ . \tag{5.1}$$

ADMM With the only difference from AMA being the minimization of the augmented Lagrangian (AL), the first step of Algorithm 3 is:

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

expressed as the linear system

$$\begin{bmatrix} \left(Q + \rho \sum_{i=1}^{M} L_i^\top L_i\right) & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} z \\ \nu \end{bmatrix} = \begin{bmatrix} -c - \sum_{i=1}^{M} L_i^\top \lambda_i - \rho \sum_{i=1}^{M} L_i^\top (l_i - y_i^k) \\ b \end{bmatrix} \ . \tag{5.2}$$

Clearly, the linear systems (5.1) and (5.2) have very similar structure, commonly refered to as a *KKT system*. From now on we denote a general KKT system as

$$\begin{bmatrix} K_{11} & K_{21}^\top \\ K_{21} & 0 \end{bmatrix} \begin{bmatrix} z \\ \nu \end{bmatrix} = \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} \ , \tag{5.3}$$

where

$$K = \begin{bmatrix} K_{11} & K_{21}^\top \\ K_{21} & 0 \end{bmatrix} \tag{5.4}$$

is the *KKT matrix*. For a typical MPC problem with horizon N, n_x states and n_u inputs, (5.4) 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,

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

we consider from now on that (5.4) is of 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 (5.4) 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 (5.3) 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 [18, 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 [80] 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 (5.3) 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 §2.7:

$$P_{\mathcal{D}}(p) = p + A^\top (AA^\top)^{-1}(b - Ap) \ . \tag{5.5}$$

The matrix $AA^\top \in \mathbb{S}_{++}$ can be treated offline by means of a Cholesky factorization. From Step 1 of Algorithm 2 one can see that the stepsize

does not enter the inversion.

5.1.1 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 (5.3) for AMA,ADMM and (5.5) for PDA: Factor and Solve. In this section we discuss various methods to perform these two steps. We consider the problem:

$$Kz = k \ , \tag{5.6}$$

where $K \in \mathbb{S}^{n \times n}$. The matrix K refers either to the KKT matrix (5.4), or $K = AA^\top$, encompassing both (5.3) and (5.5). The interpretation will be clear from the context.

Numerical methods to solve (5.6) 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 (5.3) might be sparse, once inverting, the sparsity is lost. Thus, the computational and memory complexities of $K^{-1}k$ are $O(2n^2)$ and $O(n^2)$, respectively. The computational and memory complexi-

ties for computing the inverse of the matrix (performed offline) are $O(n^3)$ and $O(n^2)$, 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, $\text{LDL}^\top$, 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 [66, 136]. $\text{LDL}^\top$ 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 $\text{LDL}^\top$, 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 [2] performs these operations treating the matrices as dense. Hence, the computational and memory complexities for performing the factorization (offline) are $O(n^3/3)$ and $O(n^2)$, respectively, while for the forward-backward substitution (online step) $O(2n^2)$ and $O(n^2)$. On the contrary, SuiteSparse [39] 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 (5.3)), then the computational complexity to perform the factorization is much less than $O(n^3/3)$. The memory complexity is also reduced to $O(\text{nnz}+n)$, where nnz is the number of nonzeros in L. The forward-backward substitution step ends up having computational and memory complexities of (roughly) $O(2\text{nnz}+n)$ and $O(\text{nnz}+n)$, respectively.

3. **Block elimination:** Block elimination is suitable for systems that have the KKT form (5.3). 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 (5.3) 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 [18, §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 $O(n_1^2)$ and $O(n_2^2)$ 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 [12] 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 $O((n_1 - n_2)^2)$, for the fully dense factorization.

(b) Once y is computed, z of (5.3) is calculate using $z = Zy + \hat{z}$. This step involves a matrix-vector multiplication and a vector addition, resulting in a computational complexity of $O(n_1 n_2)$.

(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 $O\left(n_2^2\right)$ assuming the factorization is fully dense.

5. **Riccati recursion:** Suppose that the KKT system (5.3) 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 [52, 54, 6, 53, 97] 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.*, [84]). 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 [34, 39]. 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 $O(2\text{nnz} + n)$ and $O(2\text{nnz} + n)$.

Table 5.1 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	$O\left(2n^2\right)$	$O\left(n^2\right)$
CLAPACK	$O\left(2n^2\right)$	$O\left(n^2\right)$
SuiteSparse	$O\left(2\text{nnz} + n\right)$	$O\left(\text{nnz} + n\right)$
Block elimination	$O\left(N^2(n_x + n_u)^2\right)$	$O\left(N^2(n_x + n_u)^2\right)$
Nullspace method	$O\left(N^2 n_x(n_x + n_u)\right)$	$O\left(N^2 n_x(n_x + n_u)\right)$
Riccati	$O\left(N(6n_x^2 + 8n_x n_u + 2n_u^2)\right)$	$O\left(N(2n_x^2 + 3n_x n_u + n_u^2/2)\right)$
Custom	$O\left(2\text{nnz} + n\right)$	$O\left(2\text{nnz} + n\right)$

Table 5.1: Computational and memory complexities for online linear system solve. The block elimination and nullspace methods perform on the KKT system (5.3), 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 (5.3), or Nn_x for (5.5).

5.1.2 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. We consider a multi-stage optimal control problem of the form:

$$\begin{array}{ll} \text{minimize} & (1/2)\sum_{i=1}^{N}\left(x_i^\top Q x_i + u_i^\top R u_i\right) + (1/2)x_{N+1}^\top Q x_{N+1}, \\ \text{subject to} & x_{i+1} = Ax_i + Bu_i \ , i = 0, \ldots, N \\ & u_{\min} \leq u_i \leq u_{\max} \ , i = 0, \ldots, N \\ & \|F_x x_i\|_2 \leq f_x \ , i = 0, \ldots, N+1 \\ & \|F_u u_i\|_2 \leq f_u \ , i = 0, \ldots, N \ . \end{array} \tag{5.7}$$

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 5.2.

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

Table 5.2: Problem (5.7) 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 5.1, 5.2 and 5.3 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 constrast to SuiteSparse), allows for sequential access of the memory or spatial locality, which is important for problems for which the size

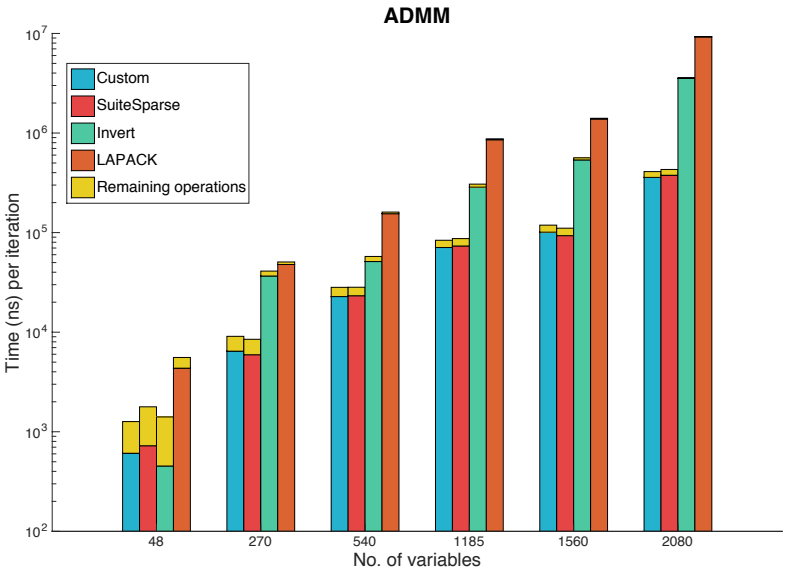


Figure 5.1: Computational load distribution per iteration of ADMM. The remaining operations in yellow color refer to the remaining algebraic operations of the algorithm.

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 total iteration time.

5.2 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.

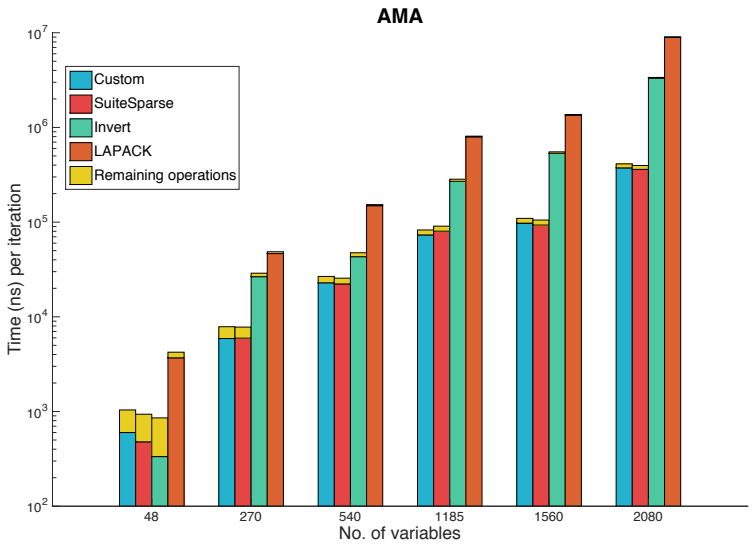


Figure 5.2: Computational load distribution per iteration of AMA.

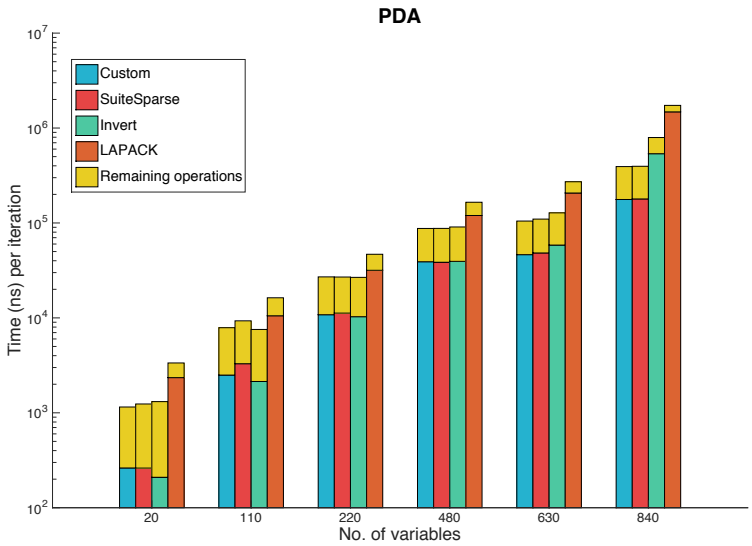


Figure 5.3: Computational load distribution per iteration of PDA.

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 $O(n^2)$ and $O(n^2)$.
- **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 complexity of $O(n^2)$ and $O(n^2)$.
- **SuiteSparse :** SuiteSparse representes the matrices in CSR format, as we already mentioned in §5.1.1. The matvec operation has computational and memory complexities of $O(\text{nnz})$ and $O(\text{nnz})$.
- **Custom methods :** We perform exhaustive code generation for this method [84]. The idea is to explicitly write the entries of the matrix in a generated C file.

5.2.1 Experiments

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

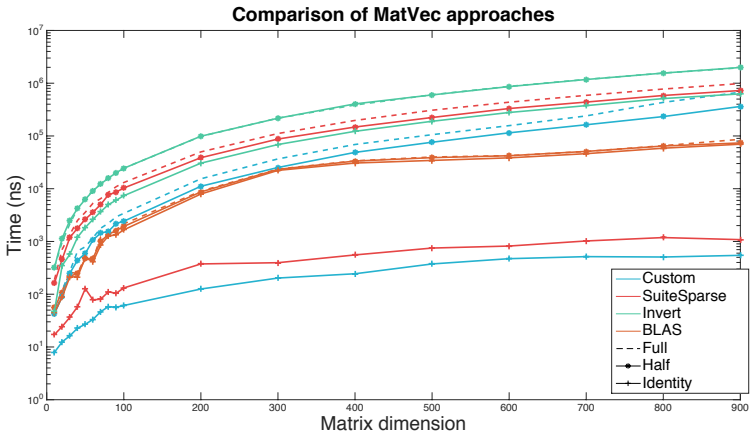


Figure 5.4: Custom code generation performs well, but at the cost of very long preprocessing periods. Once the matrices become half-full, BLAS (in orange) clearly outperforms the alternatives, keeping an almost constant cost per solve, regardless of the sparsity.

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 5.4. 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 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 5.5).

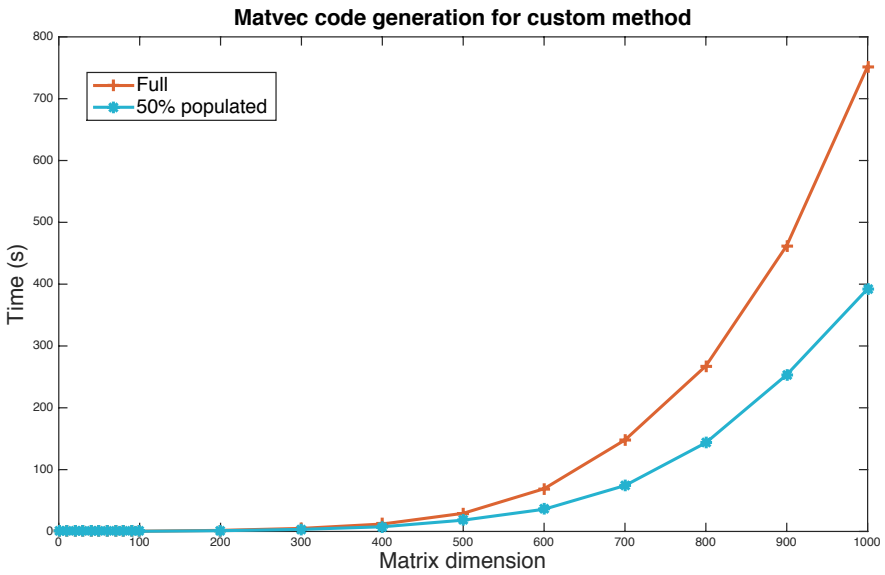


Figure 5.5: Code generation for matvec operations of varying size (horizontal axis) versus time (vertical axis). The matrices involved are fully populated (red) and 50% populated lower triangular. The generation time increases exponentially.

6

Examples

We demonstrate some of the methods presented in the previous sections with three optimal control problems. The first example involves MPC for tracking of a reference signal, boiling down to solving a sequence of QPs. The second example considers the planetary soft landing problem, an originally nonconvex problem that is relaxed to a second-order cone representable problem. The third example is an economic optimization problem targeting the minimization of a building's energy consumption given price and weather forecasts, which can be expressed as an LP.

In all the examples termination is based on the decrease of the corresponding residuals, as they are introduced in Appendix C. The residuals are measured in the l_1 norm.

6.1 Aircraft control

In this example the linearized model of a Boeing 747-200 (B747) is considered [71]. The model has $n_x = 12$ states and $n_u = 17$ inputs and the aim is the tracking of a reference signal $r(k)$ for two of the states, namely the airspeed and the roll angle. We discretize with sampling period $T_s = 0.2s$ and consider a total simulation period of

$40s$. First, a *steady state target calculator* computes a pair of setpoints $(\Delta x_s(k), \Delta u_s(k))$ for the aircraft, according to a desired reference signal. Subsequently, an MPC controller tracks the delivered setpoint.

Problem formulation We are interested in tracking a specific profile in the airspeed as well as a change in the roll angle. More specifically, the airspeed experiences a drop of $3m/s$, lasting from the 23^{rd} to the 28^{th} seconds of the simulation, while the roll angle changes from 0° to 30° and back. The first change occurs between the 6^{th} and the 9^{th} second of the simulation, while the second one between the 12^{th} and the 15^{th}. The original problem can be cast as

$$\begin{array}{ll} \text{minimize} & \|\Delta x_N - \Delta x_s\|_P^2 + \sum_{i=0}^{N-1} \left(\|\Delta x_i - \Delta x_s\|_Q^2 + \|\Delta u_i - \Delta u_s\|_R^2 \right) \\ \text{subject to} & \Delta x_0 = \Delta \bar{x}_0 \\ & \Delta x_{i+1} = A\Delta x_i + B\Delta u_i,\ i = 0, \dots, N-1 \\ & \Delta u_{min} \leq \Delta u_i \leq \Delta u_{max}\ i = 0, \dots, N-1\ . \end{array} \tag{6.1}$$

We condense the problem, *i.e.*, we eliminate the states and express it in terms of the inputs. This will give rise to a dense Hessian matrix $H \in \mathbb{S}_{++}^{n_u N}$. The problem to solve becomes

$$\begin{array}{ll} \text{minimize} & \Delta u^\top H \Delta u + h(k)^\top \Delta u \\ \text{subject to} & \Delta u_{min} \leq \Delta u \leq \Delta u_{max}\ , \end{array} \tag{6.2}$$

with variables $\Delta u = (\Delta u_0, \Delta u_1, \dots, \Delta u_{N-1}) \in \mathbb{R}^{n_u N}$. Note that the reference signals parametrize the affine part of the objective (term $h(k)$). This is a smooth and strongly convex QP. *Since only box constraints on the inputs are applied,* (6.2) *can be solved by applying directly the PGM, and obtain a linear convergence rate.* Although this might be the most efficient way to deal with the problem at hand, the aim of this work is to demonstrate how decomposition methods can be applied as an alternative to solve the problem. This being the case, we can solve it by means of the accelerated methods presented in Chapter 3, making use of the strong convexity of the objective. Subsequently, we decide to mitigate the bad conditioning of the dense Hessian by applying a Cholesky factorization, writing it as $H = KK^\top$, K being lower triangular and invertible. A change of basis is performed, $\hat{\Delta} u = K^\top \Delta u$.

Now the problem can be reformulated as

$$\begin{array}{ll} \text{minimize} & \hat{\Delta}u^\top \hat{\Delta}u + \hat{h}(k)^\top \hat{\Delta}u \\ \text{subject to} & L\hat{\Delta}u \leq l \ , \end{array} \tag{6.3}$$

with variables $\hat{\Delta}u \in \mathbb{R}^{n_u N}$. The matrix-vector pair (L, l) describes the polytopic constraints that are now imposed in the place of the simple bound constraints that we had in (6.2). This is the price paid for eliminating the dense Hessian in the objective. Note that the problem can be cast in the form (1.2), with $f(z) = z^\top z + \hat{h}(k)^\top z$ and $g(y) = \delta_-(y)$, $y = Lz - l$. Consequently, we are able to apply left preconditioning to the constraints by means of the methods discussed in Chapter 4. We solve the problem for the following scenarios: $N = 5$, $N = 12$, cold-started, warm-started at the primal and dual optimal points of the previous solve. The outputs are reported in Table 6.1.

The resulting QP makes a nice benchmark for applying splitting algorithms, since it exhibits favorable properties that allow for acceleration in different ways. Strong convexity encourages the use of accelerated relaxation, while the simple form of the indicator function g allows for preconditioning without altering the complexity of the proximal step. Note, however, that the linear convergence rate that would hold if PGM was applied directly to (6.2) cannot be recovered, since for both AMA and ADMM *smoothness and strong convexity of the objective is not sufficient for ensuring a linear rate (§3.3).*

The problem is solved in a rolling horizon fashion until the end of the simulation period. At every solve, the initial state is perturbed by 1% with respect to the previously computed one. Preconditioning is used in all the cases, performed only on the dual variables, *i.e.*, a diagonal matrix E was chosen based on the heuristic row-column equilibration algorithm proposed in [124]. We make use of nine, in total, versions of the splitting methods presented in this work. The idea is to start with the basic versions of the algorithms and observe how the performance increases as we gradually enhance them by means of the techniques discussed in Chapters 3 and 4.

- FAMA (Algorithm 4), preconditioned (Algorithm 7) and restarted. In the basic version, the stepsize is chosen as $\rho = 1/L_f$.

No stepsize selection is needed for the preconditioned versions.

- Relaxed ADMM (Algorithm 3), preconditioned (Algorithm 8) with and without adaptive stepsize selection. In the case where $N = 5$, the 'optimal' pair of preconditioner and stepsize was chosen by solving the SDP proposed in [56]. For the case $N = 12$ the SDP becomes too big to solve and we resort back to the equilibration heuristic. When preconditioned, the stepsize is chosen to be $\rho = 1$. In the basic version, further tuning is required (the problem data affects the selection more), and ρ was set to 0.005. The relaxation parameter generally speeds-up convergence, as discussed in §2.5, and it is set to 1.8.

- FADMM (Algorithm 5), preconditioned with and without adaptive stepsize. The stepsizes are set as in the previous case, and the restarting constant η is set to 0.999.

PDA did not behave particulary well for this example, hence it was not involved in the comparison. A possible reason for this is the sufficient stepsize condition as given in Algorithm 2 that gave rise to conservative stepsizes in this case. The termination threshold was set to 10^{-3}. Finally, we restrict the number of iterations to a maximum of 10^4, since an algorithm that reaches this number is already impractical for our purposes.

Discussion The comparison's results are depicted in Table 6.1. It is typically the case for all nine algorithms that the iteration count increases once the reference signals change violently from one problem to the next. This effect is depicted in Figure 6.1. As a first observation, FAMA works significantly better than the other methods for this problem. One reason for this is the effective restarting policy, which seems to reduce drastically the iteration count when the reference changes. Although ADMM also includes a restarting scheme, there were more outliers observed at the points of change. A possible remedy for this is further tuning of the η parameter.

	Average Iters		Peak Iters	
N = 5	**Cold**	**Warm**	**Cold**	**Warm**
FAMA	3196.2	3068.9	7386.3	6407.5
+ precondition	133.4	35.0	330.2	76.2
+ restart	36.1	22.5	82.4	48.9
ADMM relax	1393.6	857.8	2715.5	1555.3
+ precondition	312.2	96.0	313.2	136.5
+ adaptive	92.8	33.4	187.7	69.6
FADMM	1278.8	1148.0	2578.5	2347.5
+ precondition	264.7	324.8	354.9	390.2
+ adaptive	76.40	73.3	167.1	120.1
N = 12	**Cold**	**Warm**	**Cold**	**Warm**
FAMA	5384.9	1626.4	9111.9	2490.6
+ precondition	401.3	150.3	617.1	194.7
+ restart	51.8	38.7	84.7	61.5
ADMM relax	4059.0	3502.9	5234.3	3856.5
+ precondition	1111.1	507.4	1110.6	620.6
+ adaptive	207.4	90.4	336.4	153.0
FADMM	2814.4	2294.6	3749.3	2483.9
+ precondition	876.6	965.2	1085.1	1175.2
+ adaptive	110.6	117.0	165.4	155.8

Table 6.1: Iteration count for several versions of the three algorithms for Problem (6.3). The first column contains average number of iiterations over 195 problem instances when $N = 5$, and over 188 problem instances when $N = 12$. The second column contains the average number of iterations counted over the periods when the reference signals change, *i.e.*, $6^{\text{th}} - 9^{\text{th}}$, $12^{\text{th}} - 15^{\text{th}}$ and $23^{\text{rd}} - 28^{\text{th}}$ seconds.

Preconditioning has a dramatic effect on the behaviour of all the methods, reducing the number of iterations by an approximate factor of 10 to 40 in the case of FAMA.

Looking more carefully into FAMA, we see that reseting the acceleration scheme has a positive effect. Regarding ADMM, the accelerated variant behaves better (on average) in all problem instances. The adaptive stepsize speeds up the convergence in all cases.

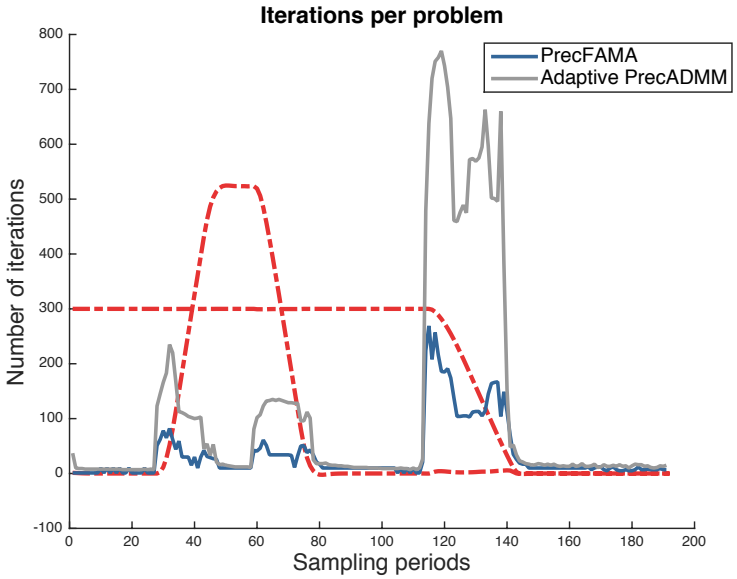


Figure 6.1: Number of iterations in correlation to the varying reference signals. The two references (red line) are scaled and translated so that their effect on the number of iterations is visible. The dashed lines represent the iterations for 195 subproblems, solved by means of PrecFAMA and Adaptive PreADMM with warm-starting. It is obvious that when the references change, the iteration count goes up accordingly. However, FAMA handles the changes much better than ADMM.

Warm-starting is quite effective as well. Contrary to the other methods, FADMM seems to behave in a puzzling way, since warm-starting can potentially deteriorate its performance, at least for this example. A possible reason might be that the extrapolated sequences generated from the FADMM change more rapidly from one iteration to the next and so the effect of warm-starting is negated. This could also explain why FAMA seems to support warm-starting better, since it has a more effective (less prone to tuning) restarting policy. Adaptation of the stepsize seems to be reducing dependency on warm-starting, robustifying, in some sense, the optimization problem. This argument is further supported by the observation that the adaptive and preconditioned versions of (F)ADMM do not exhibit such a big difference in the number of iterations when the problem size changes, *i.e.*, from $N = 5$ to

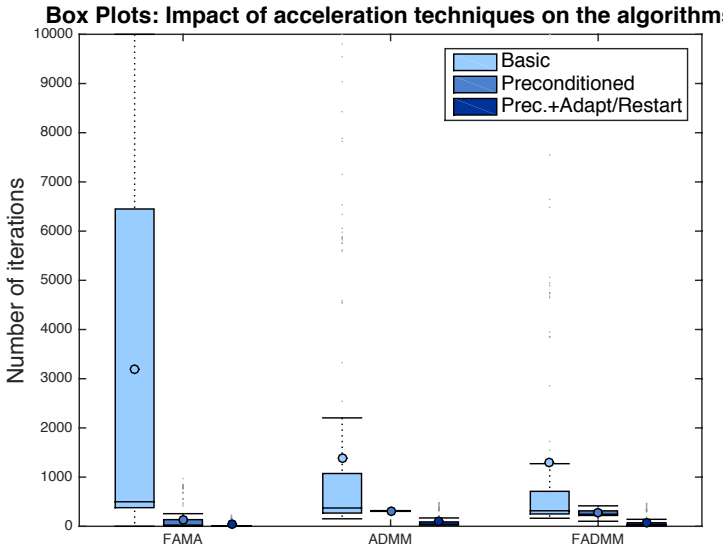


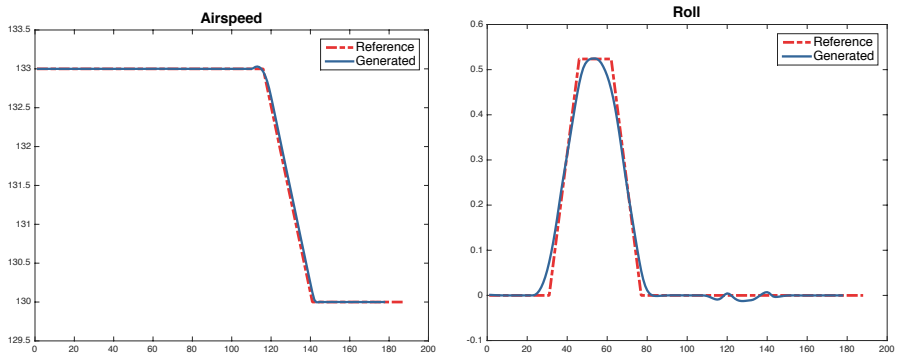
Figure 6.2: Iterations' statistics for the three variants of the three algorithms, depicted as box plot. Preconditioning has a double effect: it reduces both the absolute number of iterations as well as its variance. The case demonstrated is for $N = 5$, cold-started.

$N = 12$. The effect of the enhancements is depicted in Figure 6.2.

The solutions achieved are accurate enough in all cases, with a normed relative error ($\|u - u^*\|/\|u^*\|$) of $\approx 10^{-2}$ for $N = 5$ and $\approx 10^{-1}$ for $N = 12$. Figure 6.3 is representative of the quality of tracking.

6.2 The planetary soft landing problem

The problem presented in [1] regards the situation where an autonomous spacecraft lands on the surface of a planet by using thrusters, which produce a force vector that has both an upper and a nonzero lower bound on its magnitude. The control constraints are thus represented by a nonconvex set which has the form of a ring. These kind of constraints appears in a plethora of optimal control problems, but the case of planetary landing is of interest because, under some assumptions, the optimal trajectories of a relaxed version of the problem (where the nonconvex constraint set is replaced with a



(a) Tracking the airspeed reference **(b)** Tracking the roll angle reference

Figure 6.3: Tracking performance for the preconditioned FADMM algorithm with adaptive stepsize selection in the case that $N = 12$. Even for the relatively big error relative to the optimal solution (10^{-1}), and given the small perturbation on the initial state, the quality of tracking is still satisfactory.

convex one) are also optimal for the original problem. Hence a lossless convexification can be achieved. The relaxed problem can be written as an SOCP.

Problem formulation We first present the original (nonconvex) problem:

$$\begin{array}{ll} \text{minimize} & (x_0 - z_0)^\top Q(x_0 - z_0) + \alpha \sum_{i=0}^{N-1} \|u_i\|_2 \\ \text{subject to} & x_{i+1} = Ax_i + Bu_i + Ew_i,\ i = 0, \ldots, N-1 \\ & x_N = x_f, \\ & \gamma |e_1^\top x_i| \leq e_2^\top x_i,\ i = 0, \ldots, N \\ & 1 \leq \|u_i\| \leq c,\ i = 0, \ldots, N-1\ , \end{array} \tag{6.4}$$

with variables x_i, u_i and $\alpha \in \mathbb{R}_{++}$ a positive weight. Variable $x = (p_x, p_y, v_x, v_y)$ is the state of the $x - y$ position and the corresponding velocity coordinates of the spacecraft. The conic constraint corresponds to a *glide slope* constraint, ensuring that the spacecraft remains in a cone defined by a minimum slope angle. The nonconvex constraint on the inputs can be convexified by lifting, as discussed in [1, § 3]. The

relaxed problem can be written as

$$\begin{array}{ll} \text{minimize} & (x_0 - z_0)^\top Q(x_0 - z_0) + \alpha \sum_{i=0}^{N-1} \sigma_i \\ \text{subject to} & x_{i+1} = Ax_i + Bu_i + Ew_i,\ i = 0, \ldots, N-1 \\ & x_N = x_f, \\ & \gamma |e_1^\top x_i| \leq e_2^\top x_i,\ i = 0, \ldots, N \\ & \|u_i\| \leq c,\ i = 0, \ldots, N-1 \\ & \|u_i\| \leq \sigma_i,\ i = 0, \ldots, N-1 \\ & \sigma_i \geq 1,\ i = 0, \ldots, N-1\ , \end{array} \tag{6.5}$$

with variables x_i, u_i, σ_i.

There exist several scenaria in which the solutions of (6.5) and (6.4) coincide. We consider one such scenario, similar to the numerical instances presented in [1, § 5]. The data is

$$A = \begin{bmatrix} 0 & I \\ -\theta^2 I & \theta S \end{bmatrix} \in \mathbb{R}^n,\ B = E = \begin{bmatrix} 0 \\ I \end{bmatrix} \in \mathbb{R}^m\ ,\ S = \begin{bmatrix} 0 & 2 \\ -2 & 0 \end{bmatrix}\ ,$$

$g = 3.7114$ is the gravitational acceleration of Mars, $w = -e_2 g$, $\theta = 1/T_r$ is its rotation rate with rotation period $T_r = 1.026$ days, $c = 4$, $\gamma = 1/\sqrt{3}$, $z_0 = (-15, 20, -10, 1)$, $x_f = (0, 0.1, 0, 0)$. The system is discretized using a zero-order hold with sampling period of $0.33s$. For a final time of $15s$ this gives $N = 45$. The weight matrix that penalized the initial state is chosen as $Q = \mathbf{diag}(2, 2, 1, 1)$. The problem can be expressed as (P) with:

$$f(x, u, \sigma) = \left\{ (x_0 - z_0)^\top Q(x_0 - z_0) + \sum_{i=0}^{N-1} \sigma_i \colon \boldsymbol{A}(x, u) = \boldsymbol{b} \right\}\ , \tag{6.6}$$

which can be further split into a smooth and a nonsmooth part according to (2.1), *i.e.*, $h(x, \sigma) = (x_0 - z_0)^\top Q(x_0 - z_0) + \sum_{i=0}^{N-1} \sigma_i$ and $\delta_{\mathcal{D}}(x, u) = \delta_{\{\boldsymbol{A}(x,u)=\boldsymbol{b}\}}(x, u)$. The proximal terms are:

$$g_{1,i}(Gx_i) = \delta_2(e_1^\top x_i, (1/\gamma) e_2^\top x_i),\ i = 0, \ldots, N \tag{6.7}$$

$$g_{2,i}(u_i) = \delta_2(u_i, c),\ i = 0, \ldots, N-1 \tag{6.8}$$

$$g_{3,i}(u_i, \sigma_i) = \delta_2(u_i, \sigma_i),\ i = 0, \ldots, N-1 \tag{6.9}$$

$$g_{4,i}(\sigma_i) = \delta_+(\sigma_i - 1),\ i = 0, \ldots, N-1\ , \tag{6.10}$$

where $\delta_2(x,t)$ denotes the indicator function for the second-order cone (Table A.1, Appendix A). We overloaded notation for the dynamics equation by denoting

$$\boldsymbol{A} = \begin{bmatrix} -A & -B & I & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & -A & -B & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & I & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & -A & -B & I \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & I \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} Ew_0 \\ Ew_1 \\ \vdots \\ Ew_{N-1} \\ x_f \end{bmatrix},$$

and G is defined as $G = \begin{bmatrix} e_1^\top \\ (1/\gamma)e_2^\top \end{bmatrix}$.

Problem's (6.5) objective function is not strongly convex, and thus cannot benefit from an accelerated version of the methods mentioned above. In addition, the presence of the 2-norm ball and SOC constraints render preconditioning expensive (see Table 4.1). Consequently, we end up with few options in terms of choosing an algorithm. We try ADMM (with and without over-relaxation), FADMM (which comes with no extra assumptions) as well as PDA. For ADMM we set the stepsize to $\rho = 0.3$, while for the fast version some retuning needed to be done, setting it to $\rho = 1$. This is typically done using a trial-and-error procedure, having as indicator the balance between the primal and the dual residuals' decrease (see also (4.2)). For PDA the primal stepsize is set to $\tau = 1/L_h$, while ρ is chosen such that the inequality in Algorithm 2 is satisfied. Finally, the over-relaxation constant for ADMM is set to $\theta = 1.8$. The simulation we run involves the computation of the optimal open loop state-input trajectories for 20 different initial conditions spread uniformly around z_0, perturbed by at most 10% of its nominal value.

Discussion The soft-landing problem is a nice and challenging benchmark for first order splitting algorithms since, although it might not have favorable numerical properties, it is highly decomposable, involving lots of easy-to-compute proximal steps (equations (6.7),(6.8),(6.9),(6.10)). All the updates can be expressed in closed form

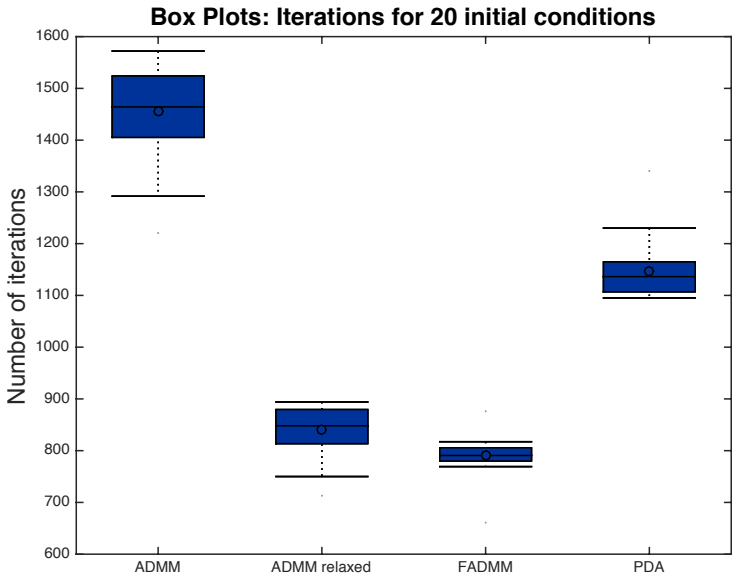
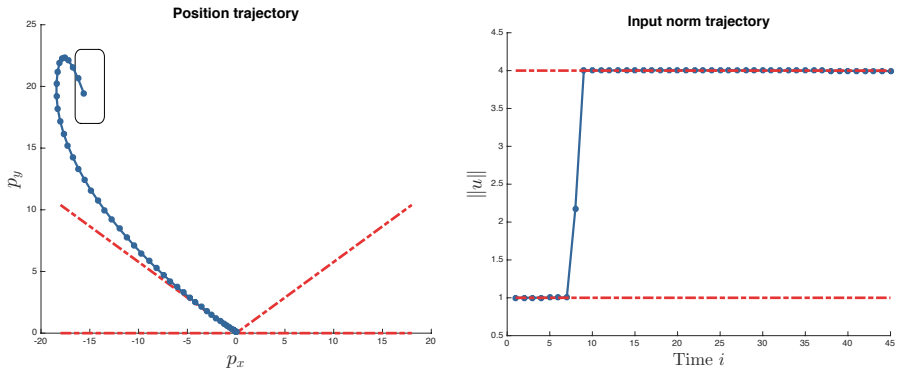


Figure 6.4: Iterations' statistics for the four algorithms.

and are separable across time. They involve projections onto the non-negative orthant, the second-order cone and the 2-norm ball, translated into the corresponding conjugate operations when PDA is used (see Table A.1 in Appendix A).

The numerical results are demonstrated in the form of a box plot in Figure 6.4. Overall, ADMM with over-relaxation and FADMM perform the best among the proposed methods, while the basic version of ADMM (no over-relaxation) performs the worst. PDA behaves similarly to ADMM, being a competitive option for this problem.

It should be pointed out that the average number of iterations needed to converge to a relative accuracy of 10^{-3} is high for all four methods. This is a specifically difficult problem to solve due to the large number of active constraints at optimality, as depicted in Figure 6.5. This is the reason that even augmented Lagrangian-based methods, which usually behave well, take so many iterations for convergence. Nevertheless, the figure suggests that the quality of the solution is good enough, even with this small accuracy.



(a) $x - y$ coordinates of spacecraft **(b)** Norm of inputs

Figure 6.5: Trajectory of the position as the spacecraft lands to the specified point $x_f = (0, 0.1)$. The initial state lies within some interval away from the desired one $z_0 = (-15, 20)$. With red color is depicted the glide angle constraint. Observe that many state constraints are active at the optimal trajectory. In the second plot the optimal input trajectory is depicted. As expected, the convex relaxation is exact, *i.e.*, the inputs stay in between 1 and 4. Note that almost all of them are saturated at optimality.

6.3 Building economic control

In this section, we consider a control problem for a building heating system and design an MPC controller to minimize the total cost of operation that takes into account the prediction of the weather. The heating system is modeled by a discrete linear system

$$\begin{aligned} x_{i+1} &= Ax_i + Bu_i + Bd_i \\ y_i &= Cx_i \ . \end{aligned}$$

The linear model is extracted by means of the platform OpenBuild [67]. The linear system has ten states in x_i without specific physical interpretation, since they have resulted from model reduction. The system input $u_i = (u_1, u_2, u_3)$ includes the electrical power input (kW) to the heating system of the three zones. The disturbance input $d_i = (d_1, d_2, d_3)$ constitutes the outside temperature ($^\circ$C), solar gains (kW), and internal gains (kW). The output of the system $y_i = (y_1, y_2, y_3)$ represents the temperature in the three building zones. The sampling rate

of the system is $T_s = 20$ min. All data is recovered from the software EnergyPlus [33].

Problem formulation The purpose is to design an economic MPC controller with the objective to keep the temperature of the building zones within the comfort constraints, while minimizing the energy bill. The economic MPC controller uses an economic linear cost function in Problem 6.11, as opposed to the usual quadratic cost function used in regulation problems. The inputs and the outputs of the model are subject to box-constraints.

$$\begin{array}{ll} \text{minimize} & \sum_{i=0}^{N} c_i^T u_i \\ \text{subject to} & x_{i+1} = Ax_i + B_u u_i + B_d d_i,\ i = 0, \ldots, N-1 \\ & y_i = Cx_i, \\ & u_{min} \leq u_i \leq u_{max}\ i = 0, \ldots, N-1 \\ & y_{min} \leq y_i \leq y_{max},\ i = 1, \ldots, N\ , \end{array} \tag{6.11}$$

The input constraints capture the heating capacity of the building system with $u_{min} = 0\ kW$ and $u_{max} = 15\ kW$, while the output constraints ensure comfort in the building with $y_{min} = 22\,^{\circ}\text{C}$ and $y_{max} = 26\,^{\circ}\text{C}$. The sequence d_i denotes the disturbance prediction, and c_i is the electricity prices with a periodic high price and low price periods, i.e., $c_i = 0.04\ \$/kWh$ (between 00:00Hrs to 10:00Hrs, and 16:00Hrs to 24:00Hrs) and $c_i = 0.2\ \$/kWh$ (between 10:00Hrs to 16:00Hrs). The problem can be formulated as a linear program.

The fact that the ratio of states to inputs is relatively large (10/3) makes the condensed formulation (state elimination) an attractive option. Since the cost is linear and involves only the inputs, condensing the problem will not complicate the objective more. The extra complication comes in terms of the output constraints which turn into polytopic ones, as was the case with the Boeing example. Hence, the part of the constraints matrix associated to the outputs becomes full lower triangular and the problem can be recast as an inequality form

LP:

$$\begin{array}{ll} \text{minimize} & c^\top u \\ \text{subject to} & Lu \leq l \ , \end{array} \tag{6.12}$$

with variables $u = (u_0, \ldots, u_{N-1}) \in \mathbb{R}^{3N}$. We solve the problem for $N = 72$ periods (24h), in a receding horizon fashion, assuming that the disturbance prediction d_i varies in a uniform way and up to 20% around the predicted value. We run the simulation for a duration of 24h.

Due to the linear objective, only (F)ADMM and PDA can solve the problem. The heuristic preconditioning approach used in the Boeing example is also used here, hence a left preconditioner for the constraints is computed. The comparison is performed between ADMM, FADMM and PDA, for their preconditioned and original versions, both for cold and warm-starting. The stepsize is chosen to be $\rho = 0.1$ for ADMM, while a primal preconditioner T is computed in the case of PDA in order to ensure that the stepsize inequality of Algorithm 9 is satisfied. The same technique for the derivation of preconditioners is suggested in [108].

Discussion The numbers of iterations and timings are presented in Table 6.2 and depicted in Figure 6.6. We have put a ceiling of 1000 iterations per solve. As commonly observed (see, *e.g.*, [62]), PDA does not behave well when solving LPs. However, preconditioning has an impressive effect on PDA, reducing the number of iterations (for the same termination threshold) by a factor of 10-20. The effect is beneficial for the other methods, as well. Additionally, warm-starting makes a big difference, with its effect being more pronounced in the cases of ADMM and PDA.

Another interesting observation is that FADMM does not perform as well as ADMM in this case. A possible reason for this is the frequent reset of the acceleration scheme, which reached an average of 20% over the total number of iterations.

	Original		Preconditioned	
No. Iters	**Cold**	**Warm**	**Cold**	**Warm**
ADMM	465.8	248.7	155.1	73.9
FADMM	490.4	367.3	185.6	169.7
PDA	NA	NA	543.7	212.2

Table 6.2: Number of iterations and solve times averaged over 72 optimal control problems solved in a rolling horizon. The indication 'NA' means that the average was above the threshold of 1000 iterations.

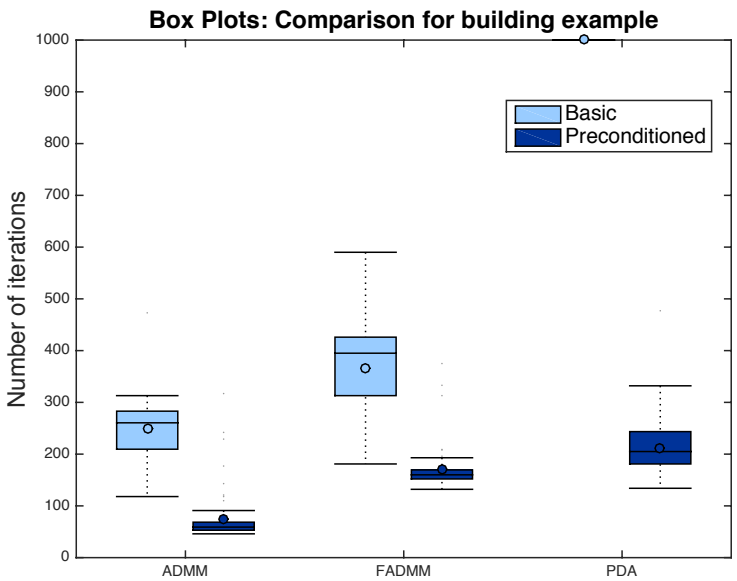


Figure 6.6: Iterations' statistics for the original and preconditioned versions of the three algorithms, depicted as box plot. PDA's iterations count is saturated, since it failed (on average) to reach optimality within 1000 iteations

7

Summary

Operator splitting methods have made their way to the control community, being recently applied in embedded applications modeled as small to medium scale optimization problems, typically covering a range of hundreds to few thousands of variables. Optimization-based control and estimation regularly involve sequences of convex (quadratic) problems, parameterized by some quantity that varies over time. These problems frequently need to be solved in real time, under tight sampling periods. This is in contrast to large-scale optimization problems arising in fields like machine learning, where the purpose is to recover a suboptimal solution to one instance of a problem, involving an immense number of variables. In this work we attempt to analyze splitting methods from this perspective and assess whether they make a good candidate for control.

7.1 When splitting should (and should not) be used

As this survey progressed, we gradually revealed ways that mitigate the dependence of splitting methods on the problem data and the tuning parameters. One can picture a continuum between first order and

higher order splitting schemes, starting from the original versions of the algorithms, as presented in Chapter 2, proceeding to adaptive and preconditioned versions in Chapter 4, finding at the end more sophisticated splitting schemes that incorporate second order information, briefly mentioned in §7.3 that follows. As we move along this spectrum, robustness and accuracy is increased at the expense of more time spent in pre-processing (computation of efficient preconditioners) and an increase in the per iteration cost.

The algorithms presented in this work involve a reasonable preprocessing period and result in reasonably simple updates. Our claim is that first order splitting methods are *not* suitable for:

1. High accuracy requirements.
2. Solving a family of problems that differ significantly from each other.

The second case arises due to the fact that first order splitting methods *need offline tuning*. When the problem data changes this tuning has to be redone. There is no evidence that a once well-tuned splitting algorithm will provide a good solution to a modified problem. The latter suggests that, although some robustification of the methods has been achieved in the ways described above, they are not on par with higher order methods in this aspect.

On the other hand, splitting methods *are suitable for*:

1. Problems that need to be repeatedly solved and are similar to each other.
2. Platforms where resources and/or computational power are limited.

The first case regards MPC problems, especially involving linear time-invariant (LTI) systems, where the data does not change significantly. In addition, warm-starting the variables to previous solutions offers a significant advantage. Concerning the second point, a requirement for low computational cost is the ability to end up with simple updates for the subproblems. This happens, *e.g.*, when the matrices that appear

in the problem stay constant, hence no inversion and/or factorization needs to be performed online. This is typically the case for LTI systems, with tracking or regulation objective.

7.2 A rough guideline

Once it is decided that the problem at hand is amenable to splitting, the first question that comes to mind is how to perform this splitting. This choice can heavily affect the speed of the algorithm. Choosing a splitting pattern is equivalent to formulating the subproblems that have to be solved in the three algorithmic schemes presented in Chapter 2. Consequently, the choice will also confine the options for acceleration and preconditioning. A general guideline would be the following:

1. All subproblems should have a closed form solution if possible; if not, they should be cheap to solve. The whole purpose of using splitting on (P) is to end up with simpler subproblems.

For a given set of data, there ideally exist theoretically optimal stepsize and relaxation constants. Due to the fact that these constants cannot be computed in most of the cases, we reduce the dependency on those by means of several heuristics. More specifically, we have observed that:

2. Preconditioning should always be used if possible. It has been empirically observed and experimentally supported that it is the most decisive factor for speeding up convergence.

3. Adaptation of the stepsize, especially in ADMM, can speed up convergence greatly. In the case that simultaneous diagonalization can be used (see Chapter 5), adaptation should be used. It is observed that, even in the case that the optimal stepsize is known, adaptation might further reduce the number of iterations.

4. If an accelerated version of an algorithm can be used without heavily altering a well-structured problem, then it should be used. Acceleration improves significantly on the number of iterations needed for convergence. Restarting of the scheme is optional in the case of AMA. Although restarting can end up increasing the

number of iterations needed for convergence, in our experience it generally acts beneficially to the method. In both FADMM and FAMA, the combination of the accelerated restarted versions with an adaptive stepsize strategy works well in practice. For FPDA the adaptation is already embedded in the method.

In the subsequent Figures 7.1, 7.2 and 7.3 we attempt to give a rough guideline on how an algorithm should be modified, once selected. The flow chart should be conceived as a proposed sequence of steps, in the sense that they usually (but not necessarily) enhance the performance of the algorithm. The steps suggested are also allowable, in the sense that *convergence guarantees are ensured and the computational cost is not significantly increased.*

The choice of the algorithm should be motivated from the problem formulation. It is important, *e.g.*, to choose whether to solve the control problem in its dense or sparse form. This choice is motivated from the ratio of states to inputs, the length of the horizon and the sparsity patterns, as discussed in Chapter 5. This option typically comes with a tradeoff between expensive operations and the number of iterations. For example, it is common that the more decomposable the problem is, the more iterations it will take to converge. This tradeoff has to be evaluated on the specific application. Time-critical applications might tolerate an increased computational cost per iteration in order to achieve a (relatively) accurate solution in less iterations. Resource-limited applications might, on the other hand, prefer lighter computations at the expense of more iterations for convergence. It is difficult, however, to know in advance which of the two approaches will take less time.

Finally, the experimental results we have on both randomly generated and benchmark control problems have lead us to the following conclusions:

5. Restarted FAMA and its preconditioned version work well for QPs. In almost all the examples we have generated, the method outperformed the other two candidates.

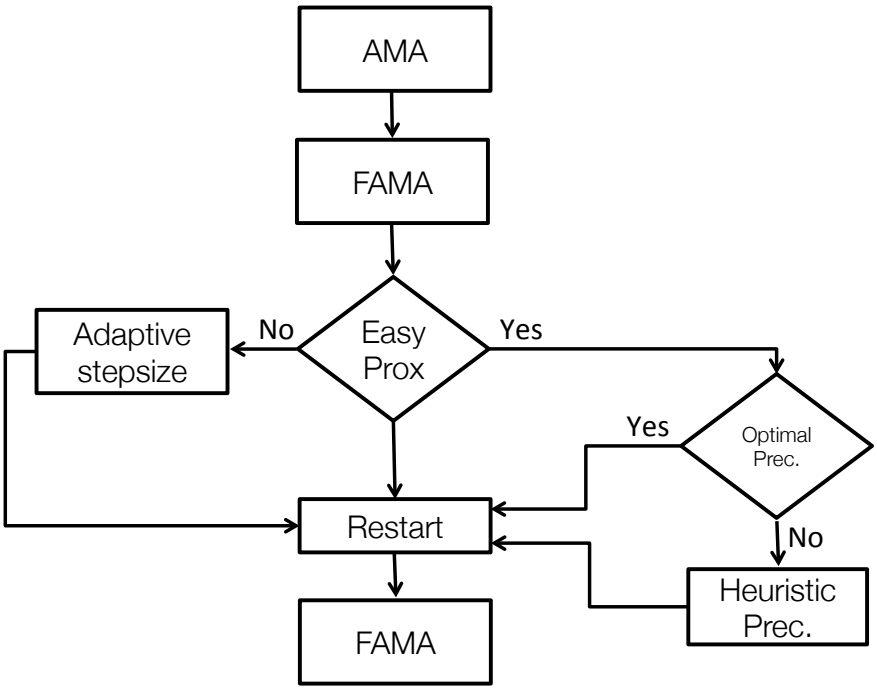


Figure 7.1: Flow chart for AMA. The block 'Easy Prox' refers to whether a diagonal preconditioner would complicate the prox step, according to Table 4.1. The optimal preconditioner refers to minimizing the exact condition number of the scaled dual function, as discussed in Chapter 4. In case the SDP is too big, or it fails to be solved, a heuristic method is employed. Note that there is an unconditional transission from AMA to FAMA for the reason that FAMA can be applied to any problem that AMA can solve, while the former one is almost always faster.

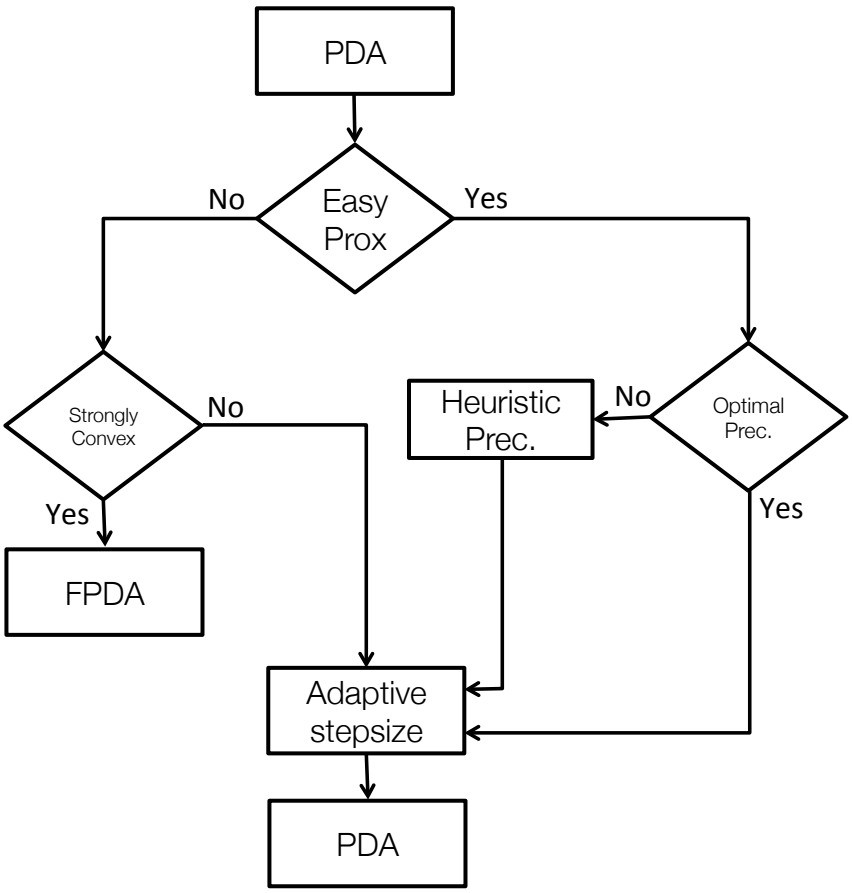


Figure 7.2: Flow chart for PDA. An optimal preconditioner can be derived by solving the same SDP as with AMA. This will give raise to a constraint preconditioner P, that subsequently has to be scaled by a primal preconditioner T so that the stepsize condition is satisfied (see Algorithm 9).

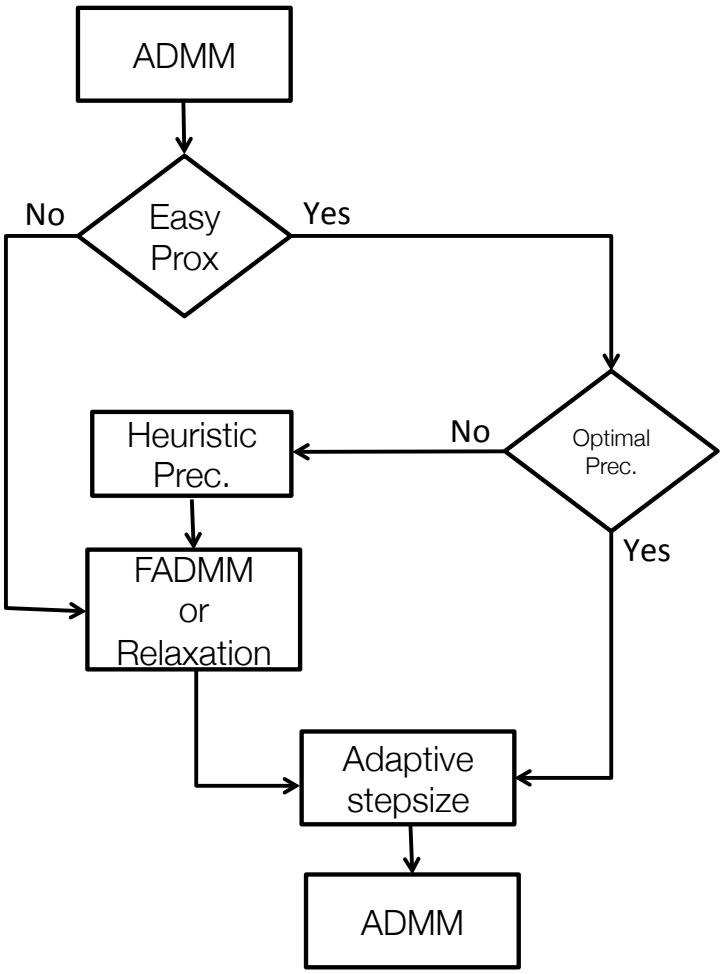


Figure 7.3: Flow chart for ADMM. Derivation of the optimal diagonal preconditioner follows the same reasoning as in the case of AMA.

6. ADMM is the most robust option when little pre-processing is desired. This holds both in terms of its ability to solve a wide variety of convex problems, as well as its decent performance even with little tuning.

7. PDA seems to be the most sensitive to tuning, as well as the slowest to converge in terms of iterations. However, the method has a potentially low cost per iteration and it works under general assumptions. For this reason it deserves more development, since the majority of the works concerning it are mostly theoretical, with almost no solid practitioners' guidelines on how to tune and use it.

7.3 Extensions and other directions

Second order information A reasonable extension to the preconditioned versions of the algorithms presented in Chapter 4 would be to look for *variable metric schemes*, *i.e.*, preconditioners that adapt while iterating. Higher order methods emanate from the idea of variable preconditioners since exact (or approximate) Hessians of the problem at hand are evaluated while iterating, resulting in significantly faster convergence than ordinary first order methods. The trade-off comes again with the complexity of the proximal operator, as well as the computation of the preconditioner that has to be performed frequently. In addition, convergence of the algorithms becomes more subtle to prove. Few works exist that combine splitting algorithms with more sophisticated preconditioners such as [11], [29], [107]. Although the injection of second order information in the splitting schemes is not covered extensively in this survey, it is almost always the case that, if the evaluation of a solution to the subproblems is easily computable (see, *e.g.*, [107]), the speedup gained can be remarkable.

Distributed control Leaving behind the range of small to medium scale problems discussed in this survey, splitting methods have been as well used in the framework of distributed control, where traditional centralized control methods would require strong computational power

in one entity as well as a dense communication scheme over the network. Distributed MPC [120] overcomes these difficulties by allocating computational tasks on different subsystems and requiring only partial communication between them. Similar to centralized MPC, distributed MPC allows coupling among the dynamics and the costs of the sub-systems. However, the coupling is limited by certain communication constraints, hence distributed optimization algorithms are required to take these constraints into consideration. The research in this area has mainly focused on the impact of distributed optimization on system properties such as stability and feasibility, and on developing efficient distributed optimization algorithms. Currently available distributed optimization algorithms which have been studied for distributed MPC [127], [32] require a variable amount of communication per time-step. The practical performance of these methods is highly dependent on the network topology and sensitive to communication errors.

Inexact splitting Motivated by distributed optimization, where inexact solutions of the local problems and communication errors may occur, we introduce another area of recent interest regarding splitting methods, *i.e.*, convergence under inexact updates. The main effort is put into analysing the effect of errors on the overall algorithm and providing conditions under which convergence can still be guaranteed. An analysis of first order methods with erroneous gradients (inexact oracles) was performed in [42]. This work derives the interesting conclusion that the accelerated versions of the algorithms, based on the over-relaxation sequences presented in Chapter 3, result in error accumulation in this case, hence they are not necessarily superior to the original versions. Convergence rates of inexact proximal gradient methods are derived in [121]. Inexact versions of a generalized form of PGM and of PDA have been introduced in [114] and [30], respectively. In the control field, related works include [111], [130] and [89].

Infinite-dimensional problems We can roughly state that all the theoretical results stated for the three algorithms presented in this work are

originally derived in Hilbert-space settings [7], [128], [5], [85], [21], [30]. The authors of [125] use these converegence guarantees to solve the constrained linear quadratic regulator problem, *i.e.*, an infinite-horizon MPC regulation problem by means of the AMA and FAMA [126]. In [41], the authors consider an infinite-dimensional optimal control problem constrained in a finite dimensional space. Consequently, the dual problem is expressed in this lower-dimensional space, where the problem becomes tractable. A fast dual gradient method is then used on the dual function.

Three-operator splitting The generalization of decomposition methods to the case where more than two functions appear in the objective of (P), without performing any reformulations (*e.g.*, by introducing copies of variables, as we do throughout this work), has been an open area of research for many years. The recent breakthrough from Davis et al. [38] allows for splitting an objective with three composite terms. The scheme results in a novel three-block ADMM, where (at least) one of the three terms needs to be strongly convex, with guaranteed convergence under certain stepsize restrictions. Analyzing three-operator splitting schemes from a more general viewpoint, the authors in [79] recover a different novel version of ADMM, as well as a unification of several existing primal-dual splitting schemes which involve two or more composite terms (including the PDA presented in this work). The area of multiple-operator splitting methods is at its infancy and several developments are expected in the coming years.

Non-convex splitting Whereas a fair number of well-established techniques exist for decomposing convex Non-Linear Programs (NLPs), there is, as yet, no consensus around a set of splitting methods applicable to nonconvex programs. However, one can make a clear distinction between Sequential Convex Programming (SCP) approaches and augmented Lagrangian techniques.

An SCP method consists in iteratively solving convex NLPs, which are local approximations of the original nonconvex NLP, by means of splitting techniques similar to the ones described in this survey. A case

in point is the approach of [129], in which an l_1 penalty is used to relax the nonconvex constraints. On the contrary, an augmented Lagrangian method aims at decomposing a nonconvex auxiliary problem inside a dual loop [25], [70], [78]. While convergence guarantees can be derived in both frameworks, computational drawbacks also exist on both sides. For instance, it is not clear how to preserve the convergence properties of SCP schemes when subproblems are solved to a low level of accuracy, which is likely to occur if a first-order method is applied as an inner solver. On the contrary, the inexactness issue can be rigorously handled inside an augmented Lagrangian algorithm [31], [46]. However, it is still not clear how the primal nonconvex subproblems should be solved efficiently via a splitting technique [77].

Acknowledgements

We would like to thank Jean-Hubert Hours for contributing to this manuscript with his knowledge in noncnovex optimization and Ivan Pejcic for thorough readings in the early stages of this work. We are also grateful to Panos Patrinos, Pontus Giselsson and the anonymous reviewers for their very useful comments and suggestions.

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ ERC Grant Agreement n. 307608, as well as from the People Programme (Marie Curie Actions) under REA grant agreement no 607957 (TEMPO).

Appendices

A

Definitions

In this Appendix we give several definitions of notions that appear throughout the survey.

Subdifferential and Conjugacy

Definition A.1 (Subdifferential)**.** The subdifferential of a convex function f at x is defined as

$$\partial f(x) = \{u : \langle u, z - x\rangle \leq f(z) - f(x)\ \forall z \in \mathbf{dom}\, f\}.$$

Note that when f is differentiable, $\partial f(x) = \{\nabla f(x)\}$.

Definition A.2 (Conjugate)**.** The conjugate of a convex function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$, denoted by $f^\star : \mathbb{R}^n \to \overline{\mathbb{R}}$, is defined as

$$f^\star(\lambda) = \sup_z \{\langle z, \lambda\rangle - f(z)\} \ .$$

Theorem A.1. Let $f \in \Gamma_0(\mathbb{R}^n)$. The following relation holds:

$$u \in \partial f(x), \text{ for some } x \in \mathbb{R}^n, \Leftrightarrow\ x \in \partial f^\star(u) \ .$$

The notion of the conjugate function is quite useful in the framework of first order methods for convex optimization. Below, we give

Description	$f(z)$	$f^\star(\lambda)$
Nonnegative orthant	$\delta_+(z)$	$\delta_-(\lambda)$
Box/l_∞-norm ball ($\|z\|_\infty \leq \alpha$)	$\delta_\infty(z, \alpha)$	$\alpha\|\lambda\|_1$
l_1-norm ball ($\sum_i \|z_i\| \leq \alpha$)	$\delta_1(z, \alpha)$	$\alpha\|\lambda\|_\infty$
l_2-norm ball ($\|z\|_2 \leq \alpha$)	$\delta_2(z, \alpha)$	$\alpha\|\lambda\|_2$
Second-order cone ($\|z\|_2 \leq t$, $(z,t) \in \mathbb{R}^{n+1}$)	$\delta_2(z, t)$	$\delta_2(\lambda, y)$

Table A.1: Conjugate relations.

some pairs of conjugate functions that we usually meet in the problems of interest. Table A.1 by no means covers the wide spectrum of convex conjugate functions that can be analytically derived. We refer the interested reader to the work [27] for an exhaustive list of conjugate functions. Similarly, since proximal operators of convex functions have been derived in numerous works, we do not intend to list them here. We once more refer the interested reader to [27] for a list, as well as [104] for the derivation procedure of the most common ones. Finally, the Matlab library [103] implements many proximal operators for direct use, serving as a supplement to [104].

Useful identities Moreau identity is a very useful Lemma that associates a convex function with its conjugate. This is instrumental for deriving all the algorithms presented in this work, since, as we saw, their derivation depends on the application of proximal methods to the dual function (D), which is expressed via conjugate functions. Below we give Moreau identity associated to the proximal, as well as the generalized proximal operator (4.5).

Lemma A.2. Let $f \in \Gamma_0(\mathbb{R}^n)$. Then for any $x \in \mathbb{R}^n$

$$\mathbf{prox}_{\rho f^\star}(x) + \rho\, \mathbf{prox}_{f/\rho}(x/\rho) = x, \;\; \forall\, 0 < \rho < +\infty \;\; .$$

Lemma A.3. Let $f \in \Gamma_0(\mathbb{R}^n)$ and $P \in \mathbb{S}_{++}$. Then for any $x \in \mathbb{R}^n$

$$\mathbf{prox}^P_{\rho f}(x) + \rho P^{-1}\, \mathbf{prox}^{P^{-1}}_{f/\rho}(P(x/\rho)) = x, \;\; \forall\, 0 < \rho < +\infty \;\; .$$

The Moreau identity is, hence, very useful for computing the prox of a conjugate function knowing the prox of the original one (and vice versa).

B

Derivation of the Algorithms from Proximal Methods

The subsequent results are instrumental as they demonstrate the equivalence of the AMA, PDA and ADMM to proximal algorithmic schemes, as presented in Chapter 2, when applied to the dual problem (D) or the saddle formulation (S). We repeat here the dual function for clarity:

$$-d(\lambda) = F(\lambda) + g^\star(\lambda) \ ,$$

where $F(\lambda) := f^\star(-L^\top \lambda) - \langle \lambda, l \rangle$.

AMA and PGM. The Alternating Minimization Algorithm can be derived from the application of the PGM iteration (2.5) to $-d$. The proof is inspired from [10, Lemma 3.2], where the equivalence between the accelerated versions of AMA and PGM is drawn.

Since f is strongly convex, it follows from Lemma 3.1 that $f^\star$ is smooth. Subsequently, F is also smooth. Then ∇F is Lipschitz continuous with some constant L_F, and the proximal gradient algorithm reads

Algorithm 10 Dual Proximal Gradient Method

Require: Initialize $\rho < 2/L_F$.
loop
1: $z^{k+1} = \underset{z}{\operatorname{argmin}} \quad f(z) + \langle L^\top \lambda^k, z \rangle$
2: $\lambda_i^{k+1} = \mathbf{prox}_{\rho g_i^\star}\left(\lambda_i^k + \rho(L_i z^{k+1} + l_i)\right),\ i = 1, \ldots, M$
end loop

The following Lemma holds:

Lemma B.1. The second step of Algorithm 10 is equivalent to steps 2 and 3 of Algorithm 1 combined, written as

$$y_i^{k+1} = \mathbf{prox}_{g_i/\rho}\left(L_i z^{k+1} + l_i + \lambda_i^k/\rho\right)$$
$$\lambda_i^{k+1} = \lambda_i^k + \rho(L_i z^{k+1} + l_i - y_i^{k+1}) \ .$$

Proof. Step 1 of Algorithm 10 amounts to computing the gradient of $f^\star(-L^\top \lambda) - \langle \lambda, l \rangle$. Using Theorem A.1,

$$-L^\top \lambda^k \in \partial f(z^{k+1}) \Leftrightarrow z^{k+1} = \nabla f^\star(-L^\top \lambda^k) \ .$$

Thus, the gradient of F can be expressed as

$$\nabla F(\lambda^k) = -L z^{k+1} - l \ .$$

Step 2 is, apparently, the proximal step with respect to $g_i^\star$ in the direction of the negative gradient.

We are now ready to prove the assertion. To this end, we first denote $\mu_i^k = \lambda_i^k + \rho(L_i z^{k+1} + l_i)$, and subsequently substitute the result of the proximal step y_i^{k+1} to the dual update λ_i^{k+1} :

$$\begin{aligned}\lambda_i^{k+1} &= \lambda_i^k + \rho(L_i z^{k+1} + l_i) - \rho\, \mathbf{prox}_{g_i/\rho}(\lambda_i^k/\rho + L_i z^{k+1} + l_i) \\ &= \mu_i^k - \rho\, \mathbf{prox}_{g_i/\rho}(\mu_i^k/\rho) \ .\end{aligned}$$

Using Moreau identity (A.2), it directly follows that

$$\lambda_i^{k+1} = \mu_i^k - \mu_i^k + \mathbf{prox}_{\rho g_i^\star}(\mu_i^k) = \mathbf{prox}_{\rho g_i^\star}(\mu_i^k) \ .$$

□

Finally, the accelerated version of PGM, named *Fast Proximal Gradient Method (FPGM)* [10],[59], applied to the dual problem, is given below. The method is the result of simply applying Nesterov's momentum scheme to Algorithm 10, as discussed in Chapter 3. Any other acceleration sequence [132] would result in similar algorithms.

Algorithm 11 Fast Dual Proximal Gradient Method

Require: Initialize $\rho \in (0, 1/L_F]$, $\lambda^{-1} = \lambda^0 \in \mathbb{R}^p$, $\alpha^{-1} = \alpha^0 = 1$.

loop

1: $\hat{\lambda}^k = \lambda^k + ((\alpha^{k-1} - 1)/\alpha^k)(\lambda^k - \lambda^{k-1})$

2: $z^k = \underset{z}{\operatorname{argmin}} \quad f(z) + \langle L^\top \hat{\lambda}^k, z \rangle$

3: $\lambda_i^{k+1} = \mathbf{prox}_{\rho g_i^\star}\left(\hat{\lambda}_i^k + \rho(L_i z^k + l_i)\right),\ i = 1, \ldots, M$

end loop

PDA and FBS. Consider the problem (P) that can be written as

$$\text{minimize} \quad h(z) + \delta_{\mathcal{D}}(z) + g(Lz + l) \ ,$$

with $h(z)$, $\delta_{\mathcal{D}}(z)$ as defined in (2.1). The saddle formulation (S) is repeated here for clarity:

$$\mathcal{S}(z; \lambda) = \langle Lz + l, \lambda \rangle + h(z) + \delta_{\mathcal{D}}(z) - g^\star(\lambda) \ . \tag{B.1}$$

The *saddle subdifferential* of (S) [118] is defined as

$$\partial \mathcal{S}(z; \lambda) = \begin{bmatrix} \partial_z \mathcal{S}(z; \lambda) \\ \partial_\lambda \mathcal{S}(z; \lambda) \end{bmatrix} \ .$$

A solution to the minimization problem can be derived by finding a saddle point (z^*, λ^*) of $\mathcal{S}$, so that $(0, 0) \in \partial \mathcal{S}(z^*; \lambda^*)$. The relation can be easily derived from strong duality and the primal-dual optimality conditions.

After taking the partial derivatives we have

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \in \begin{bmatrix} L^\top \lambda^* + \nabla h(z^*) + \partial \delta_{\mathcal{D}}(z^*) \\ -Lz^* - l + \partial g^\star(\lambda^*) \end{bmatrix} \ . \tag{B.2}$$

Let us now derive the optimality conditions for the two iterations of Algorithm 2. It is easy to see that the following inclusion is satisfied [30]:

$$\begin{bmatrix} -\nabla h(z^k) \\ 0 \end{bmatrix} \in \begin{bmatrix} \partial\delta_{\mathcal{D}}(z^{k+1}) + L^\top \lambda^{k+1} \\ -Lz^{k+1} - l + \partial g^\star(\lambda^{k+1}) \end{bmatrix} + \begin{bmatrix} (1/\tau)I & -L^\top \\ -L & (1/\sigma)I \end{bmatrix} \begin{bmatrix} z^{k+1} - z^k \\ \lambda^{k+1} - \lambda^k \end{bmatrix} ,$$

which can be further written as

$$B(w^k) \in A(w^{k+1}) + P(w^{k+1} - w^k) \ , \tag{B.3}$$

where $w = (z, \lambda) \in \mathbb{R}^n \times \mathbb{R}^p$ and

$$B(w) = \begin{bmatrix} -\nabla h(z) \\ 0 \end{bmatrix}, A(w) = \begin{bmatrix} \partial\delta_{\mathcal{D}}(z) + L^\top \lambda \\ -Lz - l + \partial g^\star(\lambda) \end{bmatrix}, P = \begin{bmatrix} (1/\tau)I & -L^\top \\ -L & (1/\sigma)I \end{bmatrix} .$$

Note that, at optimality (when $w^{k+1} = w^k = w^*$), (B.3) reduces to (B.2).

For an invertible operator P, (B.3) can be reordered as

$$w^{k+1} = (I + P^{-1} \circ A)^{-1} \circ (I - P^{-1} \circ B)(w^k) \ . \tag{B.4}$$

Equation (B.4) is nothing but a generalized form of the PGM, where instead of $\mathbf{prox}_g(\cdot)$ and $\nabla f(\cdot)$ we have the operators $(I + A)^{-1}$ and B, respectively, while the operator P^{-1} acts as a generalization of the stepsize ρ. Under several assumption on the operators A and B, *PDA is a preconditioned version of FBS operating at the primal-dual space* $\mathbb{R}^n \times \mathbb{R}^p$ *seeking for a* w *such that* $(0, 0) \in A(w) + B(w)$*, while AMA is FBS operating at the dual space* $\mathbb{R}^p$ *seeking a solution* λ *to the inclusion* $0 \in \partial g^\star(\lambda) + \nabla F(\lambda)$.

ADMM and DRS. The DRS scheme constitutes of three iterations, and when applied to the dual problem results in:

$$v^{k+1} = \mathbf{prox}_{\rho F}\left(\lambda^k - w^k\right) \tag{B.5}$$

$$\lambda^{k+1} = \mathbf{prox}_{\rho g^\star}\left(v^{k+1} + w^k\right) \tag{B.6}$$

$$w^{k+1} = w^k + v^{k+1} - \lambda^{k+1} \ , \tag{B.7}$$

where $d = F + g^\star$, as defined in the begining of the chapter. The function F does not need to be smooth in this case. We are going to

analyze the three iterations sequentially, following the approach form the lecture notes [135].

For (B.5), we have that

$$\mathbf{prox}_{\rho F}\left(\lambda^k - w^k\right) = \underset{v}{\operatorname{argmin}} \left\{ F(v) + (1/2\rho)\|v - \lambda^k + w^k\|_2^2 \right\} \ ,$$

the optimality condition of which is

$$0 \in -L\partial f^\star(-L^\top v) - l + (1/\rho)(v - \lambda^k + w^k) \tag{B.8}$$

We are going to show that the proximal step (B.5) is equivalent to an augmented Lagrangian minimization update. For this purpose, consider the minimization problem

$$\text{minimize} \quad f(z) + (\rho/2)\|Lz + l + (\lambda^k - w^k)/\rho\|_2^2$$

with variable $z \in \mathbb{R}^n$, which can be equivalently written as

$$\begin{array}{ll} \text{minimize} & f(z) + (\rho/2)\|u\|_2^2 \\ \text{subject to} & Lz + l + (\lambda^k - w^k)/\rho = u, \end{array} \tag{B.9}$$

with variables $z \in \mathbb{R}^p, u \in \mathbb{R}^p$. Introducing a Lagrange multiplier $v \in \mathbb{R}^p$, the optimality conditions for problem (B.9) become:

$$-L^\top v \in \partial f(z), \quad \rho u = v, \quad Lz + l + (\lambda^k - w^k)/\rho - u = 0 \ ,$$

which, by elimination of the variables z, u, can be written as

$$0 \in -L\partial f^\star(-L^\top v) - l + (1/\rho)(v - \lambda^k + w^k),$$

which is (B.8).

Furthermore, we have from (B.5) that

$$\begin{aligned} v^{k+1} - \lambda^k + w^k &\in -\rho\partial F(v^{k+1}) \\ &= -\rho(-L\partial f^\star(-L^\top v^{k+1}) - l) \\ &= \rho(Lz^{k+1} + l) \ , \end{aligned}$$

where z^{k+1} is a minimizer of problem (B.9). To wrap up, Step (B.5) can be written as

$$\begin{aligned} z^{k+1} &= \underset{z}{\operatorname{argmin}} \left\{ f(z) + (\rho/2)\|Lz + l + (\lambda^k - w^k)/\rho\|_2^2 \right\} \\ v^{k+1} &= \lambda^k - w^k + \rho(Lz^{k+1} + l) \ . \end{aligned} \tag{B.10}$$

Step (B.6) reads as $\lambda^{k+1} = \mathbf{prox}_{\rho g^\star}\Big(\lambda^k + \rho(Lz^{k+1} + l)\Big)$. From Lemma (B.1), Step (B.6) is equivalent to

$$\begin{aligned} y_i^{k+1} &= \mathbf{prox}_{g_i/\rho}\Big(L_i z^{k+1} + l_i + \lambda_i^k/\rho\Big) \\ \lambda_i^{k+1} &= \lambda_i^k + \rho(L_i z^{k+1} + l_i - y_i^{k+1}) \ . \end{aligned}$$

Finally, Step (B.7) results in $w^{k+1} = \rho y^{k+1}$. Substituting w^{k+1} back to the augmented Lagrangian in (B.10), we end up with the ADMM iterations.

C

Stopping Conditions

We make use of the KKT conditions in order to terminate the algorithms presented in this work. We write down the optimality conditions for the Lagrangian (L) or for the saddle function (S) and express them in terms of the optimality conditions derived from each iterate of the corresponding algorithms. The algorithm is terminated once the KKT conditions are satisfied to some prespecified accuracy.

AMA Consider the Lagrangian (L). The KKT conditions are:

$$0 = L_i z^* + l_i - y_i^*, \quad i = 1, \dots, M \tag{C.1}$$

$$0 = \nabla f(z^*) + \sum_{i=1}^{M} L_i^\top \lambda_i^* \tag{C.2}$$

$$0 \in \partial g_i(y_i^*) - \lambda_i^*, \quad i = 1, \dots, M \tag{C.3}$$

Taking the optimality condition for Step 1 of Algorithm 1, we have that

$$\nabla f(z^{k+1}) + \sum_{i=1}^{M} L_i^\top \lambda_i^k = 0$$

$$\nabla f(z^{k+1}) + \sum_{i=1}^{M} L_i^\top \lambda_i^{k+1} + \sum_{i=1}^{M} L_i^\top (\lambda_i^k - \lambda_i^{k+1}) = 0 \ ,$$

hence condition (C.2) is satisfied if $L^\top(\lambda^{k+1} - \lambda^k) = 0$. Accordingly we have for Step 2 that

$$\begin{aligned} &\partial g_i(y_i^{k+1}) + \lambda_i^k + \rho(L_i z^{k+1} + l_i - y_i^{k+1}) \ni 0 \\ &\partial g_i(y_i^{k+1}) + \lambda_i^{k+1} \ni 0 \ , \end{aligned}$$

which means that condition (C.3) is always satisfied each time Step 2 is executed. Finally, the primal optimality condition reads $L_i z^{k+1} + l_i - y_i^{k+1} = 0,\ i = 1, \ldots, M$. We can thus write the primal and dual residuals as

$$r^{k+1} = Lz^{k+1} + t - y^{k+1} \tag{C.4}$$

$$s^{k+1} = L^\top(\lambda^{k+1} - \lambda^k) \ . \tag{C.5}$$

PDA Consider the saddle function (S) with $h(z) = (1/2)z^\top Q z + c^\top z$. The KKT conditions are:

$$0 = Qz^* + c + \partial\delta_{\mathcal{D}}(z^*) + \sum_{i=1}^{M} L_i^\top \lambda_i^* \tag{C.6}$$

$$0 \in \partial g_i^\star(p_i^*) - L_i z^* - l_i, \quad i = 1, \ldots, M \tag{C.7}$$

As before, we write down the optimality conditions for each step of Algorithm 2. The residuals read

$$r^{k+1} = (Q - (1/\tau^k)I)(z^{k+1} - z^k) + L^\top(\lambda^{k+1} - \lambda^k) \tag{C.8}$$

$$s^{k+1} = P^k(\lambda^k - \lambda^{k+1}) + L(z^{k+1} - z^k) \tag{C.9}$$

where $P^k = \mathbf{diag}(\frac{1}{\rho_1^k}, \ldots, \frac{1}{\rho_M^k})$.

ADMM Writing the optimality conditions for each step of Algorithm 3, we derive the formulas for the primal and dual residuals,

$$r^{k+1} = Lz^{k+1} + l - y^{k+1}, \tag{C.10}$$

$$s^{k+1} = \rho L^\top(y^k - y^{k+1}) \ , \tag{C.11}$$

as given in [17].

Remark C.1. When the preconditioned versions of the algorithms are considered (Chapter 4), the residuals can be expressed in terms of the scaled variables in a similar manner.

References

[1] A. B. Açikmese and L. Blackmore. Lossless convexification of a class of optimal control problems with non-convex control constraints. *Automatica*, 2011.

[2] 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*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3rd edition, 1999.

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

[4] 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*, 15(3):485, 2008.

[5] H. Attouch and M. Soueycatt. Augmented Lagrangian and Proximal Alternating Direction Methods of Multipliers in Hilbert spaces. Applications to Games, PDE's and Control. *Pacific Journal of Optimization 5*, 2008.

[6] D. Axehill and M. Morari. An alternative use of the Riccati recursion for efficient optimization. *Systems & Control Letters*, 61(1):37–40, 2012.

[7] H. H. Bauschke and P. L. Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*. Springer Science+ Business Media, 2011.

[8] A. Beck and M. Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal of Imaging Sciences*, 2(1):183–202, 2009.

[9] A. Beck and M. Teboulle. Smoothing and first order methods: A unified framework. *SIAM Journal on Optimization*, 22(2):557–580, 2012.

[10] A. Beck and M. Teboulle. A fast dual proximal gradient algorithm for convex minimization and applications. *Operations Research Letters*, 2014.

[11] S. Becker and J. Fadili. A quasi-Newton proximal splitting method. In *Advances in Neural Information Processing Systems 25*, pages 2618–2626. 2012.

[12] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 5 2005.

[13] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods (Optimization and Neural Computation Series)*. Athena Scientific, 1996.

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

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

[16] 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.

[17] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

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

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

[20] A. M. Bradley. Algorithms for the Equilibration of Matrices and Their Application to Limited-Memory Quasi-Newton Methods,. PhD thesis, Stanford ICME, 2010.

[21] A. Chambolle and C. Dossal. On the convergence of the iterates of "FISTA". 2014.

[22] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 2011.

[23] G. Chen and M. Teboulle. A proximal-based decomposition method for convex minimization problems. *Mathematical Programming*, 1994.

[24] E. Chu, B. O'Donoghue, N. Parikh, and S. Boyd. A Primal-Dual Operator Splitting Method for Conic Optimization. Technical report, Stanford Internal Report, 2013.

[25] G. Cohen. Auxiliary problem principle and decomposition of optimization problems. *Journal of Optimization Theory and Applications*, 32(3):277–305, 1980.

[26] P. L. Combettes, L. Condat, J. C. Pesquet, and B. C. Vũ. A forward-backward view of some primal-dual optimization methods in image recovery. In *The IEEE International Conference on Image Processing*, pages 4141–4145, 2014.

[27] P. L. Combettes and J. C. Pesquet. Proximal splitting methods in signal processing. In *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pages 185–212. Springer New York, 2011.

[28] 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*, 20(2):307–330, 2012.

[29] P. L. Combettes and B. C. Vũ. Variable metric forward–backward splitting with applications to monotone inclusions in duality. *Optimization*, 63(9):1289–1318, 2014.

[30] L. Condat. A Primal-Dual Splitting Method for Convex Optimization Involving Lipschitzian, Proximable and Linear Composite Terms. *Journal of Optimization Theory and Applications*, 158(2):460–479, 2013.

[31] A. R. Conn, N. I. M. Gould, and P. L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28:545–572, 1991.

[32] C. Conte, T. Summers, M. N. Zeilinger, M. Morari, and C. N. Jones. Computational aspects of distributed optimization in model predictive control. In *The 51st IEEE Annual Conference on Decision and Control*, pages 6819–6824, 2012.

[33] D. B. Crawley, L. K. Lawrie, F. C. Winkelmann, W. F. Buhl, Y. J. Huang, C. O. Pedersen, R. K. Strand, R. J. Liesen, D. E. Fisher, M. J. Witte, and J. Glazer. EnergyPlus: creating a new-generation building energy simulation program. *Energy and Buildings*, 33(4):319–331, 2001.

[34] E. Cuthill and J. McKee. Reducing the Bandwidth of Sparse Symmetric Matrices. In *Proceedings of the ACM 24th National Conference*, pages 157–172, New York, NY, USA, 1969.

[35] G. B. Dantzig and P. Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 1960.

[36] D. Davis. Convergence rate analysis of primal-dual splitting schemes. *SIAM Journal on Optimization*, 25(3):1912–1943, 2015.

[37] D. Davis and W. Yin. Convergence rate analysis of several splitting schemes. *arXiv preprint arXiv:1406.4834*, 2014.

[38] D. Davis and W. Yin. A Three-Operator Splitting Scheme and its Optimization Applications. Technical Report CAM 15-13, University of California, Los Angeles, 2015.

[39] T. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006.

[40] W. Deng and W. Yin. On the Global and Linear Convergence of the Generalized Alternating Direction Method of Multipliers. *Technical Report Rice CAAM TR12-14*, 2012.

[41] O. Devolder, F. Glineur, and Y. Nesterov. Double smoothing technique for large-scale linearly constrained convex optimization. *SIAM Journal on Optimization*, 22(2):702–727, 2012.

[42] O. Devolder, F. Glineur, and Y. Nesterov. First-order methods of smooth convex optimization with inexact oracle. *Mathematical Programming*, 146(1-2):37–75, 2014.

[43] A. Domahidi, A. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones. Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control. In *The IEEE Conference on Decision and Control*, pages 668 – 674, Maui, HI, USA, 2012.

[44] J. Douglas and H. H. Rachford. On the Numerical Solution of Heat Conduction Problems in Two and Three Space Variables. *Transaction of the American Mathematical Society*, 82:421–489, 1956.

[45] J. Eckstein and D. P. Bertsekas. On the Douglas-Rachford Splitting Method and the Proximal Point Algorithm for Maximal Monotone Operators. *Mathematical Programming*, 1992.

[46] J. Eckstein and P. J. S. Silva. A practical relative error criterion for augmented lagrangians. *Mathematical Programming*, 141:319–348, 2013.

[47] E. Esser. Primal Dual Algorithms for Convex Models and Applications to Image Restoration, Registration and Nonlocal Inpainting. PhD thesis, UCLA, 2010.

[48] 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 of Imaging Sciences*, 2010.

[49] H. Everett. Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources. *Operations Research*, 1963.

[50] 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*, 2008.

[51] C. Fougner and S. Boyd. Parameter selection and pre-conditioning for a graph form solver. *arXiv preprint arXiv:1503.08366*, 2015.

[52] G. Frison. Numerical methods for model predictive control. 2012.

[53] G. Frison and J. B. Jørgensen. Efficient implementation of the Riccati recursion for solving linear-quadratic control problems. In *The IEEE International Conference on Control Applications*, pages 1117–1122, 2013.

[54] G. Frison, H. B. Sørensen, B. Dammann, and J. B. Jørgensen. High-performance small-scale solvers for linear Model Predictive Control. In *The IEEE European Control Conference*, pages 128–133, 2014.

[55] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite-element approximations. *Computers & Mathematics with Applications*, 1976.

[56] 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*, 60(3):644–658, 2015.

[57] P. Giselsson and S. Boyd. Linear Convergence and Metric Selection in Douglas Rachford Splitting and ADMM. *To appear in IEEE Transactions on Automatic Control*.

[58] P. Giselsson and S. Boyd. Monotonicity and restart in fast gradient methods. In *The 53rd IEEE Annual Conference on Decision and Control*, pages 5058–5063, 2014.

[59] P. Giselsson and S. Boyd. Metric Selection in Fast Dual Forward Backward Splitting. *Automatica*, 2015.

[60] Marroco A. Glowinski, R. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de Dirichlet non linéaires. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 9(R2):41–76, 1975.

[61] R. Glowinski and P. Le Tallec. *Augmented Lagrangian And Operator-splitting Methods In Nonlinear Mechanics*. Society for Industrial and Applied Mathematics, 1989.

[62] T. Goldstein, E. Esser, and R. Baraniuk. Adaptive primal-dual hybrid gradient methods for saddle-point problems. *arXiv preprint arXiv:1305.0546*, 2013.

[63] T. Goldstein, B. O'Donoghue, S. Setzer, and R. Baraniuk. Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences*, 7(3):1588–1623, 2014.

[64] T. Goldstein and S. Osher. The Split Bregman Method for l_1-Regularized Problems. *SIAM Journal of Imaging Sciences*, 2(2):323–343, 2009.

[65] T. Goldstein, C. Studer, and R. Baraniuk. A Field Guide to Forward-Backward Splitting with a FASTA Implementation. *arXiv preprint arXiv:1411.3406*, 2015.

[66] G. G. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[67] T. T. Gorecki, F. A. Qureshi, and C. Jones. Openbuild : An integrated simulation environment for building control. In *The Multi-Conference on Systems and Control*, 2015.

[68] O. Güler. On the Convergence of the Proximal Point Algorithm for Convex Minimization. *SIAM Journal on Control and Optimization*, 29(2):403–419, 1991.

[69] O. Güler. New proximal point algorithms for convex minimization. *SIAM Journal on Optimization*, 2(4):649–664, 1992.

[70] A. Hamdi and S. K. Mishra. *Decomposition Methods Based on Augmented Lagrangians: A Survey*, pages 175–203. 2011.

[71] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. Constantinides. Predictive Control using an FPGA with Application to Aircraft Control. *IEEE Transactions on Control Systems Technology*, 2013.

[72] B. He, H. Yang, and S.L. Wang. Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *Journal of Optimization Theory and Applications*, 2000.

[73] B. He and X. Yuan. Convergence Analysis of Primal-Dual Algorithms for a Saddle-Point Problem: From Contraction Perspective. *SIAM Journal on Imaging Sciences*, 5(1):119–149, 2012.

[74] B. He and X. Yuan. On the $O(1/n)$ Convergence Rate of the Douglas-Rachford Alternating Direction Method. *SIAM Journal on Numerical Analysis*, 2012.

[75] R. M. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 1969.

[76] J. B. Hiriart-Urruty and C. Lemarechal. *Convex Analysis and Minimization Algorithms: Part 2: Advanced Theory and Bundle Methods*. Springer, 2010.

[77] J. H. Hours and C. N. Jones. An alternating trust region algorithm for distributed linearly constrained nonlinear programs, application to the AC optimal power flow. EPFL-REPORT-205056, 2015.

[78] J. H. Hours and C. N. Jones. A parametric nonconvex decomposition algorithm for real-time and distributed nmpc. *IEEE Transactions on Automatic Control*, 2016.

[79] P. Latafat and P. Patrinos. Asymmetric Forward-Backward-Adjoint Splitting for Solving Monotone Inclusions Involving Three Operators. *arXiv.org*, 2016.

[80] Z. Liu, A. Hansson, and L. Vandenberghe. Nuclear norm system identification with missing inputs and outputs. *Systems & Control Letters*, 62(8):605–612, 2013.

[81] Z-Q. Luo M. Hong. On the linear convergence of the alternating direction method of multipliers, 2012.

[82] B. Martinet. Régularisation d'inéquations variationnelles par approximations successives. *Revue Française de Informatique et Recherche Opérationelle*, 4(R3):154–158, 1970.

[83] B. Martinet. Détermination approchée d'un point fixe d'une application pseudo-contractante. *C.R. Acad. Sci. Paris*, 274A:163–165, 1972.

[84] J. Mattingley and S. Boyd. CVXGEN: a code generator for embedded convex optimization. *Optimization and Engineering*, 2012.

[85] A. Moradifam and A. Nachman. Convergence of the alternating split Bregman algorithm in infinite-dimensional Hilbert spaces. *arXiv:1112.1960*, page 113, 2011.

[86] J. J. Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes Rendus de l'Académie des Sciences (Paris), Série A*, 255, 1962.

[87] J. J. Moreau. Proximité et dualité dans un espace Hilbertien. *Bulletin de la Société Mathématique de France*, 93(2):273–299, 1965.

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

[89] I. Necoara, A. Patrascu, and A. Nedić. *Complexity Certifications of First-Order Inexact Lagrangian Methods for General Convex Programming: Application to Real-Time MPC*, pages 3–26. Springer International Publishing, 2015.

[90] I. Necoara and J. A. K. Suykens. Application of a smoothing technique to decomposition in convex optimization. *IEEE Transactions on Automatic Control*, 53(11):2674–2679, 2008.

[91] A. Nemirovski and D. B. Yudin. *Problem complexity and method efficiency in optimization*. Wiley-Interscience series in discrete mathematics. Wiley-Interscience, Chichester, New York, 1983.

[92] Y. Nesterov. A method for solving a convex programming problem with rate of convergence $O(1/k^2)$. *Doklady Mathematics*, 269(3):543–547, 1983.

[93] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer, 2004.

[94] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 2005.

[95] Y. Nesterov. Gradient methods for minimizing composite objective function. Technical report, CORE, Catholic University of Louvain, 2007.

[96] Y. Nesterov. How to advance in structural convex optimization. *OPTIMA: Mathematical Programming Society Newsletter*, 78:2–5, 2008.

[97] I. Nielsen, D. Ankelhed, and D. Axehill. Low-rank modifications of riccati factorizations with applications to model predictive control. In *The 52nd IEEE Annual Conference on Decision and Control*, pages 3684–3690, 2013.

[98] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. I. Jordan. A general analysis of the convergence of ADMM. In *In 32nd International Conference on Machine Learning*, 2015.

[99] J. Nocedal and S. J. Wright. *Numerical Optimization 2nd edition.* Springer, 2006.

[100] B. O'Donoghue and E.J. Candes. Adaptive Restart for Accelerated Gradient Schemes. *arXiv.org*, 2012.

[101] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Operator splitting for conic optimization via homogeneous self-dual embedding. *arXiv preprint arXiv:1312.3039*, 2013.

[102] B. O'Donoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, 2012.

[103] N. Parikh. Proximal operators. https://github.com/cvxgrp/proximal.

[104] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[105] P. Patrinos and A. Bemporad. An accelerated dual gradient-projection algorithm for embedded linear model predictive control. *IEEE Transactions on Automatic Control*, 59(1):18–33, 2014.

[106] P. Patrinos, L. Stella, and A. Bemporad. Douglas-Rachford splitting: Complexity estimates and accelerated variants. In *The 53rd IEEE Annual Conference on Decision and Control*, pages 4234–4239, 2014.

[107] P. Patrinos, L. Stella, and A. Bemporad. Forward-backward truncated Newton methods for convex composite optimization. *arXiv:1402.6655*, 2014.

[108] T. Pock and A. Chambolle. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *The IEEE International Conference on Computer Vision*, pages 1762–1769. IEEE, 2011.

[109] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics* , 4(5):1–17, 1964.

[110] M. J. D. Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*, pages 283–298. Academic Press, 1969.

[111] Y. Pu, M. N. Zeilinger, and C. N. Jones. Inexact Fast Alternating Minimization Algorithm for Distributed Model Predictive Control. In *The 53rd IEEE Annual Conference on Decision and Control*, pages 5915–5921, 2014.

[112] A. U. Raghunathan and S. Di Cairano. Infeasibility detection in alternating direction method of multipliers for convex quadratic programs. In *The 53rd IEEE Annual Conference on Decision and Control*, pages 5819–5824, 2014.

[113] A. U. Raghunathan and S. Di Cairano. Optimal step-size selection in alternating direction method of multipliers for convex quadratic programs and model predictive control. In *International Symposium on Mathematical Theory of Networks and Systems*, pages 807–814, 2014.

[114] H. Raguet, J. Fadili, and G. Peyré. A Generalized Forward-Backward Splitting. *SIAM Journal on Imaging Sciences*, 6(3):1199–1226, 2013.

[115] S. Richter. Computational complexity certification of gradient methods for real-time model predictive control. PhD thesis, ETH Zurich, 2012.

[116] R. T. Rockafellar. Augmented Lagrangians and Applications of the Proximal Point Algorithm in Convex Programming. *Mathematics of Operations Research*, 1976.

[117] R. T. Rockafellar. Monotone Operators and the Proximal Point Algorithm. *SIAM Joiurnal on Control and Optimization*, 1976.

[118] R. T. Rockafellar and R. Rockajellm. *Monotone Operators Associated with Saddle Functions and Minimax Problems*, pages 241–250. American Mathematical Society, 1970.

[119] E. K. Ruy and S. Boyd. A Primer on Monotone Operator Methods. *Applied and Computational Mathematics*, 15(1), 2016.

[120] R. Scattolini. Architectures for distributed and hierarchical model predictive control - a review. *Journal of Process Control*, 19(5):723–731, 2009.

[121] M. Schmidt, N. L. Roux, and F. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. In *The 25th Annual Conference on Neural Information Processing Systems*, pages 6819–6824, 2011.

[122] R. Shefi and M. Teboulle. Rate of Convergence Analysis of Decomposition Methods Based on the Proximal Method of Multipliers for Convex Minimization. *SIAM Journal on Optimization*, 2014.

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

[124] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[125] G. Stathopoulos, M. Korda, and C. N. Jones. Solving the infinite-horizon constrained LQR problem using splitting techniques. In *The 19th IFAC World Congress*, 2014.

[126] G. Stathopoulos, M. Korda, and C. N. Jones. Solving the Infinite-Horizon Constrained LQR Problem using Accelerated Dual Proximal Methods. *To appear in IEEE Transactions on Automatic Control*, 2016.

[127] T. H. Summers and J. Lygeros. Distributed model predictive consensus via the alternating direction method of multipliers. In *The 50th Annual Allerton Conference on Communication, Control and Computing*, pages 79–84, 2012.

[128] B. F. Svaiter. On weak convergence of the Douglas-Rachford method. *SIAM Journal on Control and Optimization*, 2011.

[129] Q. Tran-Dinh, I. Necoara, and M. Diehl. A dual decomposition algorithm for separable nonconvex optimization using the penalty framework. In *The 52nd IEEE Annual Conference on Decision and Control*, 2013.

[130] Q. Tran-Dinh, I. Necoara, and M. Diehl. Fast inexact decomposition algorithms for large-scale separable convex optimization. *Optimization*, pages 1–32, 2015.

[131] P. Tseng. Applications of splitting algorithm to decomposition in convex programming and variational inequalities. *SIAM Journal on Control and Optimization*, 1991.

[132] P. Tseng. On accelerated proximal gradient methods for convex-concave optimization. *submitted to SIAM Journal on Optimization*, 2008.

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

[134] B. C. Vũ. A splitting algorithm for dual monotone inclusions involving cocoercive operators. *Advances in Computational Mathematics*, 38(3):667–681, 2013.

[135] L. Vandenberghe. Optimization methods for large-scale systems. UCLA EE 236C lecture notes, 2010.

[136] D. S. Watkins. *Fundamentals of matrix computations*. Pure and applied mathematics. Wiley-Interscience, New York, 2010.

[137] X. Xu, P. F. Hung, and Y. Ye. A simplified homogeneous and self-dual linear programming algorithm and its implementation. *Annals of Operations Research*, 62(1):151–171, 1996.

[138] Y. Ye, M. J. Todd, and S. Mizuno. An $O(\sqrt{nL})$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994.

[139] M. Zhu and T. Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. UCLA CAM Report, 2008.