52nd IEEE Conference on Decision and Control
December 10-13, 2013. Florence, Italy

Low-rank Modifications of Riccati Factorizations with Applications to Model Predictive Control

Isak Nielsen, Daniel Ankelhed, and Daniel Axehill

Abstract— **In optimization algorithms used for on-line Model Predictive Control (MPC), the main computational effort is spent while solving linear systems of equations to obtain search directions. Hence, it is of greatest interest to solve them efficiently, which commonly is performed using Riccati recursions or generic sparsity exploiting algorithms. The focus in this work is efficient search direction computation for active-set methods. In these methods, the system of equations to be solved in each iteration is only changed by a low-rank modification of the previous one. This highly structured change of the system of equations from one iteration to the next one is an important ingredient in the performance of active-set solvers. It seems very appealing to try to make a structured update of the Riccati factorization, which has not been presented in the literature so far. The main objective of this paper is to present such an algorithm for how to update the Riccati factorization in a structured way in an active-set solver. The result of the work is that the computational complexity of the step direction computation can be significantly reduced for problems with bound constraints on the control signal. This in turn has important implications for the computational performance of active-set solvers used for linear, nonlinear as well as hybrid MPC.**

I. Introduction

Model Predictive Control (MPC) is one of the most commonly used control strategies in industry. Some important reasons for its success include that it can handle multi-variable systems and constraints on control signals and state variables in a structured way. In each sample an optimization problem is solved and in the methods considered in this paper, the optimization problem is assumed to be solved on-line. Note, however, similar linear algebra is also useful off-line in explicit MPC (parametric) solvers. The optimization problem can be of different types depending on which type of system and problem formulation that is used. The most common variants are linear MPC, nonlinear MPC and hybrid MPC. In most cases, the effort spent in the optimization problems boils down to solving Newton-system-like equations. Hence, lots of research has been done in the area of solving this type of system of equations efficiently when it has the special form from MPC. It is well-known that these equations (or at least a large part of them) can be cast in the form of a finite horizon LQ control problem and as such it can be solved using a Riccati recursion. Some examples of how Riccati recursions have been used to speed up optimization routines can be found in, e.g., [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. In [1], the use of a Riccati factorization in an active-set (AS) method is introduced. In this reference the Riccati factorization in itself is not updated in the iterations, but instead a dense system of equations of the size of the number of active constraints. Hence, standard methods for low-rank modifications can be used. In [12] an alternative sparse, non-Riccati, factorization is used and is updated.

The main motivation for this paper is twofold. First, it has a theoretical value in the sense that it shows that it is indeed possible to perform efficient low-rank modifications of a Riccati factorization. This has sometimes been used as an argument against using this type of factorization in active-set solvers for MPC, and instead use an interior point (IP) method where the efficiency of the Riccati factorization can be fully exploited or, to use an active-set method with a generic factorization. Secondly, it is shown that both the computational complexity as well as computational times obtained from numerical experiments can be significantly reduced by this new algorithm. This opens up for even faster active-set Quadratic Programming (QP) solvers tailored for various forms of MPC, which in turn is not only useful for linear MPC, but also for nonlinear and hybrid MPC since these often rely on a QP solver as subroutine.

In this article, $\mathbb{S}^n$ denotes symmetric matrices with n columns. Furthermore, $\mathbb{S}^n_{++}$ ($\mathbb{S}^n_{+}$) denotes symmetric positive (semi) definite matrices with n columns. Furthermore, let $\mathbb{Z}$ be the set of integers, $\mathbb{Z}_{++}$ be the set of positive (non-zero) integers, and $\mathbb{Z}_{i,j} = \{i, i+1, \ldots, j\}$.

II. Problem Formulation

In this work, linear MPC problems are considered over a prediction horizon of length N for systems in the form

$$\begin{aligned} x(0) &= x_0 \\ x(t+1) &= A(t)x(t) + B(t)u(t),\ t \in \mathbb{Z}_{0,N-1} \end{aligned} \tag{1}$$

where the matrices $A(t) \in \mathbb{R}^{n\times n}$ and $B(t) \in \mathbb{R}^{n\times n_u}$ define the system. Furthermore, $x(t) \in \mathbb{R}^n$ denotes the state of the system, $x_0 \in \mathbb{R}^n$ denotes the initial state and $u(t) \in \mathbb{R}^{n_u}$ denotes the control inputs. The quadratic performance measure to be minimized is

$$J = \frac{1}{2}\sum_{t=0}^{N-1} \left\|[x^T(t)\ u^T(t)]^T\right\|^2_{Q(t)} + \frac{1}{2}\left\|x(N)\right\|^2_{Q_x(N)} \tag{2}$$

where $\|v(t)\|^2_{Q(t)} = v(t)^T Q(t) v(t)$ and

$$Q(t) = \begin{bmatrix} Q_x(t) & Q_{xu}(t) \\ Q_{xu}^T(t) & Q_u(t) \end{bmatrix}. \tag{3}$$

Furthermore, the following three assumptions are made

Assumption 1: $Q_u(t) \in \mathbb{S}^{n_u}_{++},\ t \in \mathbb{Z}_{0,N-1}$
Assumption 2: $Q(t) \in \mathbb{S}^{n+n_u}_{+},\ t \in \mathbb{Z}_{0,N-1}$
Assumption 3: $Q_x(N) \in \mathbb{S}^n_{+}$,

I. Nielsen, D. Ankelhed, and D. Axehill are with the Division of Automatic Control, Linköping University, SE-58183 Linköping, Sweden, {isani82,ankelhed,daniel}@isy.liu.se.

978-1-4673-5716-6/13/$31.00 ©2013 IEEE

The system can also in each time instant t be subject to constraints in the form

$$u_{min}(t) \leq u(t) \leq u_{max}(t),\ t \in \mathbb{Z}_{0,N-1}. \tag{4}$$

However, for notational convenience input constraints in the form

$$0 \leq u(t),\ t \in \mathbb{Z}_{0,N-1} \tag{5}$$

have been used in this paper.

There exist different equivalent optimization problem formulations of the linear MPC problem. For example, it can be written as a QP problem with only control signals as free variables, or it can be written as a QP problem where control signals and states both are free variables. The derivations of both formulations for a general linear MPC problem can be found in [13], or in [9, pp. 65–67]. It is well-known that the second alternative is often advantageous from a computational point of view, especially for problems with a long prediction horizon, and this formulation is used in this work. Using this second form, the MPC problem can be written as an optimization problem in the form

$$\begin{aligned}
\underset{\mathsf{x},\mathsf{u}}{\text{minimize}} \quad & \frac{1}{2}\sum_{t=0}^{N-1}\big(x^T(t)Q_x(t)x(t) + u^T(t)Q_u(t)u(t) + \\
& 2x^T(t)Q_{xu}(t)u(t)\big) + \frac{1}{2}x^T(N)Q_x(N)x(N) \\
\text{subject to} \quad & x(0) = x_0 \\
& x(t+1) = A(t)x(t) + B(t)u(t),\ t \in \mathbb{Z}_{0,N-1} \\
& 0 \leq u(t),\ t \in \mathbb{Z}_{0,N-1}
\end{aligned} \tag{6}$$

where

$$\begin{aligned}
\mathsf{x} &= \left[x^T(0), \ldots, x^T(N)\right]^T \\
\mathsf{u} &= \left[u^T(0), \ldots, u^T(N-1)\right]^T,
\end{aligned}$$

are the stacked states and control inputs, respectively.

Remark 1: More general MPC problems with reference tracking, output penalties and/or affine dynamics can be solved using the theory in this paper. The formulation in (6) has been chosen for notational convenience.

III. Optimization Preliminaries

In this section, the QP problem is introduced and some basic properties are discussed. Furthermore, a basic active-set method is outlined which is used as a benchmark solver in this work. For an extensive bibliography on QP, see [14].

A. Quadratic programming

The MPC problem in (6) is a QP problem with $\bar{n}$ variables, $\bar{p}$ equality constraints and $\bar{m}$ inequality constraints in the form

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & \frac{1}{2}x^T H x + f^T x \\
\text{subject to} \quad & A_{\mathcal{E}}x = b_{\mathcal{E}}, \quad A_{\mathcal{I}}x \leq b_{\mathcal{I}}
\end{aligned} \tag{7}$$

where $\bar{n} = (N+1)n + Nn_u$, $x \in \mathbb{R}^{\bar{n}}$, $H \in \mathbb{S}_+^{\bar{n}}$, $f \in \mathbb{R}^{\bar{n}}$, $A_{\mathcal{E}} \in \mathbb{R}^{\bar{p}\times\bar{n}}$, $A_{\mathcal{I}} \in \mathbb{R}^{\bar{m}\times\bar{n}}$, $b_{\mathcal{E}} \in \mathbb{R}^{\bar{p}}$ and $b_{\mathcal{I}} \in \mathbb{R}^{\bar{m}}$. Furthermore, $\mathcal{E} \in \mathbb{Z}_{++}^{\bar{p}}$ and $\mathcal{I} \in \mathbb{Z}_{++}^{\bar{m}}$ denote sets of indices to rows representing equality constraints and inequality constraints respectively in $A \in \mathbb{R}^{\bar{p}+\bar{m}\times\bar{n}}$ and $b \in \mathbb{R}^{\bar{p}+\bar{m}}$.

B. Active-set QP solvers

An active-set solver finds the *active-set* of the optimization problem, i.e., the set of constraints that hold with equality at the optimal solution. This is done iteratively by adding and removing constraints from the *working-set* until optimality is obtained. A basic, traditional active-set method is described in [15], where also an introduction to active-set QP methods can be found.

In each active-set iteration j, let the working-set be denoted $W_j(t)$ and its complement $W_j^c(t)$. The set $W_j(t)$ thus contains the indices for which the constraints are forced to hold with equality, while $W_j^c(t)$ contains the indices of the constraints that are temporarily disregarded at iteration j. Since only control input constraints are used, the control inputs and the corresponding B can be partitioned as

$$u(t) = \Pi\begin{bmatrix} w(t) \\ v(t) \end{bmatrix}, \quad B(t) = \begin{bmatrix} B_w(t) & B_v(t) \end{bmatrix}\Pi^T, \tag{8}$$

where Π is a permutation matrix, $n_u = n_w + n_v$ and $w(t) \in \mathbb{R}^{n_w}$, $v(t) \in \mathbb{R}^{n_v}$, $B_w(t) \in \mathbb{R}^{n\times n_w}$ and $B_v(t) \in \mathbb{R}^{n\times n_v}$. The partitioned matrices are defined as (using MATLAB notation for indexing)

$$\begin{aligned}
w(t) &= u_{W_j^c(t)}(t)\ \text{(free)}, \quad & B_w(t) &= B_{:,W_j^c(t)}(t), \\
v(t) &= u_{W_j(t)}(t)\ \text{(fixed)}, \quad & B_v(t) &= B_{:,W_j(t)}(t).
\end{aligned} \tag{9}$$

Also $Q_u(t)$ could be partitioned as

$$Q_u(t) = \Pi\begin{bmatrix} Q_w(t) & Q_{wv}(t) \\ Q_{wv}^T(t) & Q_v(t) \end{bmatrix}\Pi^T, \tag{10}$$

where each block is calculated similarly as in (9).

In each iteration of the active-set solver, an equality constrained QP in the form

$$\begin{aligned}
\underset{\mathsf{x},\mathsf{w}}{\text{minimize}} \quad & \frac{1}{2}\sum_{t=0}^{N-1}\big(x^T(t)Q_x(t)x(t) + w^T(t)Q_w(t)w(t) + \\
& 2x^T(t)Q_{xw}(t)w(t)\big) + \frac{1}{2}x^T(N)Q_x(N)x(N) \\
\text{subject to} \quad & x(0) = x_0 \\
& x(t+1) = A(t)x(t) + B_w(t)w(t), \\
& \forall t \in \mathbb{Z}_{0,N-1}
\end{aligned} \tag{11}$$

is solved to obtain the optimal solution for a given W_j. By the definition of the control signal partition in (8) and the constraints (5) it follows that $v(t) = 0$. Hence, for all constraints in the working-set the corresponding control input is constrained to zero. As a consequence, adding a constraint to the working-set at time t_m affects the optimal solution equivalently to as when the corresponding control signal is removed (removing the corresponding column in $B(t_m)$). Similarly, when a constraint is removed from the working-set at time t_m, the corresponding control signal becomes a free optimization variable. This affect the optimal solution equivalently to adding the control input (corresponds to add a column to $B(t_m)$) to the MPC sub-problem.

Constraints (control inputs) are added and removed (removed and added) iteratively until the optimal solution x^* of (11) satisfies the optimality conditions for (6). When the

active-set algorithm terminates, x^* is the optimal solution to the original inequality constrained MPC problem.

IV. Standard Riccati Recursion

The MPC sub-problem (11) has a special structure with an almost block diagonal KKT system. Hence, it is possible to calculate the step direction very efficiently using a Riccati factorization of the KKT coefficient matrix, followed by simple forward and backward recursions to obtain the optimal solution. This reduces the computational complexity from roughly $\mathcal{O}(N^2) - \mathcal{O}(N^3)$ to $\mathcal{O}(N)$. For more background information on Riccati factorizations, see e.g., [1], [2] or [9].

Let the matrices $F(t), P(t), Q_x(t) \in \mathbb{S}^n_+$, $G(t), Q_w(t) \in \mathbb{S}^{n_w}_{++}$, $H(t), Q_{xw}(t) \in \mathbb{R}^{n \times n_w}$ and $K(t) \in \mathbb{R}^{n_w \times n}$. The Riccati factorization is then given by Algorithm 1. Once the KKT coefficient matrix is factored, the optimal solution can be computed by solving the system of linear equations using Algorithm 2-4, which contain tailored backward and forward recursions.

Algorithm 1 Factorization (Riccati recursion)

1: $P(N) := Q_x(N)$
2: **for** $t = N-1, \dots, 0$ **do**
3: $\quad F(t+1) := Q_x(t) + A^T(t)P(t+1)A(t)$
4: $\quad G(t+1) := Q_w(t) + B_w^T(t)P(t+1)B_w(t)$
5: $\quad H(t+1) := Q_{xw}(t) + A^T(t)P(t+1)B_w(t)$
6: $\quad$ Compute and store a factorization of $G(t+1)$.
7: $\quad$ Compute a solution $K(t+1)$ to $G(t+1)K(t+1) = -H^T(t+1)$
8: $\quad P(t) := F(t+1) - K^T(t+1)G(t+1)K(t+1)$
9: **end for**

Algorithm 2 Backward recursion

1: $\Psi(N) = 0$
2: **for** $\tau = N-1, \dots, 0$ **do**
3: $\quad u_0(\tau+1) = G^{-1}(\tau+1)B_w^T(\tau)\Psi(\tau+1)$
4: $\quad \Psi(\tau) = A^T(\tau)\Psi(\tau+1) - H(\tau+1)u_0(\tau+1)$
5: **end for**

Algorithm 3 Forward recursion

1: $x(0) = x_0$
2: **for** $\tau = 0, \dots, N-1$ **do**
3: $\quad w(\tau) = u_0(\tau+1) + K(\tau+1)x(\tau)$
4: $\quad x(\tau+1) = A(\tau)x(\tau) + B_w(\tau)w(\tau)$
5: $\quad \lambda(\tau) = P(\tau)x(\tau) - \Psi(\tau)$
6: **end for**
7: $\lambda(N) = P(N)x(N) - \Psi(N)$

Algorithm 4 Forward recursion (Dual variables)

1: **for** $\tau = 0, \dots, N-1$ **do**
2: $\quad \mu(\tau) = Q_{xv}^T(\tau)x(\tau) + B_v^T(\tau)\lambda(\tau+1) + Q_{wv}^T(\tau)w(\tau)$
3: **end for**

V. Low-rank Modifications of Riccati Factorizations

In this section, it will be shown that the computational complexity of the main operations in an active-set method can be significantly reduced by using low-rank modifications to re-factor the Riccati factorization of the KKT system between each iteration of the solver.

To simplify calculations it can be noted that Line 8 in Algorithm 1 can be written as a Schur complement of the matrix

$$M(t+1) \triangleq \begin{bmatrix} F(t+1) & H(t+1) \\ H^T(t+1) & G(t+1) \end{bmatrix}. \tag{12}$$

Define the Schur complement operator

$$M(t+1)/G(t+1) \triangleq F(t+1) - H(t+1)G^{-1}(t+1)H^T(t+1). \tag{13}$$

Using this notation $P(t)$ can be calculated as

$$P(t) = M(t+1)/G(t+1), \tag{14}$$

i.e., $P(t)$ is a Schur complement of the matrix $M(t+1)$. For a detailed description of Schur complements, see e.g. [16] or [17].

In the following sections, a tilde is used to denote that a matrix is modified. The modified version of X is denoted $\tilde{X}$.

A. Removing input signal constraints from the working-set

In this section it is investigated how the matrices in the factorization given by Algorithm 1 are affected by releasing k input signal constraints that were previously active in step t. This corresponds to adding k extra control inputs (i.e. columns in $B_w(t)$) such that $\tilde{B}_w(t) \in \mathbb{R}^{n \times (n_w+k)}$, and thus also adding k rows and k columns in $Q_w(t)$ such that $\tilde{Q}_w(t) \in \mathbb{S}^{n_w+k}_{++}$. Assume, without loss of generality, that the k new columns are appended to $B_w(t)$ such that

$$\tilde{B}_w(t) \triangleq \begin{bmatrix} B_w(t) & b \end{bmatrix}, \qquad \tilde{Q}_w(t) \triangleq \begin{bmatrix} Q_w(t) & q_w \\ q_w^T & q_w^0 \end{bmatrix}, \tag{15}$$

implying that $\tilde{G}(t+1) \in \mathbb{S}^{n_w+k}_{++}$ is computed as

$$\tilde{G}(t+1) \triangleq \tilde{Q}_w(t) + \tilde{B}_w^T(t)P(t+1)\tilde{B}_w(t), \tag{16}$$

which block-wise is

$$\begin{aligned} \tilde{G}(t+1) &= \begin{bmatrix} G(t+1) & q_w + B_w^T(t)P(t+1)b \\ q_w^T + b^T P(t+1)B_w(t) & q_w^0 + b^T P(t+1)b \end{bmatrix} \\ &\triangleq \begin{bmatrix} G(t+1) & g \\ g^T & g^0 \end{bmatrix} \end{aligned} \tag{17}$$

where $G(t+1) = Q_w(t) + B_w^T(t)P(t+1)B_w(t)$ is used for the $(1,1)$-element. Similarly the modified version of $H(t+1)$ is $\tilde{H}(t+1) \in \mathbb{R}^{n \times (n_w+k)}$ and is computed as

$$\tilde{H}(t+1) \triangleq \tilde{Q}_{xw}(t) + A^T(t)P(t+1)\tilde{B}_w(t) \tag{18}$$

which block-wise can be written

$$\begin{aligned} \tilde{H}(t+1) &= \begin{bmatrix} H(t+1) & q_{xw} + A^T(t)P(t+1)b \end{bmatrix} \\ &\triangleq \begin{bmatrix} H(t+1) & h \end{bmatrix} \end{aligned} \tag{19}$$

Next, study the modified version of $M(t+1)$ in (12) where $\tilde{G}(t+1)$ and $\tilde{H}(t+1)$ are used instead of $G(t+1)$ and $H(t+1)$,

$$\tilde{M}(t+1) = \begin{bmatrix} F(t+1) & H(t+1) & h \\ H^T(t+1) & G(t+1) & g \\ h^T & g^T & g^0 \end{bmatrix}. \tag{20}$$

In analogy with (12)-(14) $\tilde{P}(t)$ can be calculated as $\tilde{P}(t) = \tilde{M}(t+1)/\tilde{G}(t+1)$. Using (51) and letting $M_{11} = F(t+1)$, $M_{12} = H(t+1)$, $M_{13} = h$, $M_{22} = G(t+1)$, $M_{23} = g$ and $M_{33} = g^0$, $\tilde{P}(t)$ can be calculated as (omitting arguments for brevity)

$$\tilde{P}(t) = \underbrace{F - HG^{-1}H^T}_{P(t)} - \underbrace{\left(h - HG^{-1}g\right)}_{V_-(t)} \underbrace{\left(g^0 - g^T G^{-1} g\right)}_{C_-(t)}{}^{-1} \underbrace{\left(h - HG^{-1}g\right)^T}_{V_-^T(t)}, \tag{21}$$

where $C_-(t) \in \mathbb{S}^k_{++}$ and $V_-(t) \in \mathbb{R}^{n\times k}$ have full column rank. Consequently, releasing k control input constraints corresponds to a rank-k downdate of $P(t)$ which results in

$$\tilde{P}(t) = P(t) - V_-(t)C_-^{-1}(t)V_-^T(t). \tag{22}$$

The effect on the modified version of $K(t+1)$, i.e. $\tilde{K}(t+1) \in \mathbb{R}^{(n_w+k)\times n}$, can be analyzed by studying the following system of equations,

$$\begin{bmatrix} G(t+1) & g \\ g^T & g^0 \end{bmatrix} \underbrace{\begin{bmatrix} K_1 \\ k_1^T \end{bmatrix}}_{\tilde{K}(t+1)} = -\begin{bmatrix} H^T(t+1) \\ h^T \end{bmatrix}, \tag{23}$$

or equivalently (since $\tilde{G}(t+1) \in \mathbb{S}^{n_w+k}_{++}$)

$$\tilde{K}(t+1) = \begin{bmatrix} K_1 \\ k_1^T \end{bmatrix} = -\begin{bmatrix} G(t+1) & g \\ g^T & g^0 \end{bmatrix}^{-1} \begin{bmatrix} H^T(t+1) \\ h^T \end{bmatrix}. \tag{24}$$

By using the definitions of $C_-(t)$ and $V_-(t)$ from (21), the expression for $\tilde{K}(t+1)$ can be written

$$\tilde{K}(t+1) = \begin{bmatrix} K_1 \\ k_1^T \end{bmatrix} = \begin{bmatrix} K(t+1) - G^{-1}(t+1)gC_-^{-1}(t)V_-^T(t) \\ -C_-^{-1}(t)V_-^T(t) \end{bmatrix}. \tag{25}$$

B. Adding input signal constraints to the working-set

Adding new constraints to the working-set, i.e. activating constraints that were disregarded in the previous active-set iteration, has a similar impact on $P(t)$ as when constraints are disregarded. Assume, without loss of generality, that the last k columns of $B_w(t)$ are removed, giving $\tilde{B}_w(t)$ as the first $(n_w - k)$ columns of $B_w(t)$, i.e.,

$$B_w(t) = \begin{bmatrix} \tilde{B}_w(t) & b \end{bmatrix}. \tag{26}$$

The matrices (17) and (19) then represents the case before the removal, and the modified $G(t+1)$ and $H(t+1)$, i.e. $\tilde{G}(t+1)$ and $\tilde{H}(t+1)$, are given by the $(1,1)$-blocks in

$$G(t+1) = \begin{bmatrix} \tilde{G}(t+1) & g \\ g^T & g^0 \end{bmatrix} \text{ and } H(t+1) = \begin{bmatrix} \tilde{H}(t+1) & h \end{bmatrix}. \tag{27}$$

Hence, the matrix $M(t+1)$ in (12) is

$$M(t+1) = \begin{bmatrix} \tilde{F}(t+1) & \tilde{H}(t+1) & h \\ \tilde{H}^T(t+1) & \tilde{G}(t+1) & g \\ h^T & g^T & g^0 \end{bmatrix}, \tag{28}$$

which in analogy with (12)-(14) gives that $P(t) = M(t+1)/G(t+1)$. Using (51) and omitting arguments, $P(t)$ can be written as

$$P(t) = \underbrace{\tilde{F} - \tilde{H}\tilde{G}^{-1}\tilde{H}^T}_{\tilde{P}(t)} - \underbrace{\left(h - \tilde{H}\tilde{G}^{-1}g\right)}_{V_+(t)} \underbrace{\left(g^0 - g^T\tilde{G}^{-1}g\right)}_{C_+(t)}{}^{-1} \underbrace{\left(h - \tilde{H}\tilde{G}^{-1}g\right)^T}_{V_+^T(t)}, \tag{29}$$

where $C_+(t) \in \mathbb{S}^k_{++}$ and $V_+(t) \in \mathbb{R}^{n\times k}$ have full column rank. This is similar to the result in the previous section, and it is trivially equivalent to

$$\tilde{P}(t) = P(t) + V_+(t)C_+^{-1}(t)V_+^T(t), \tag{30}$$

which is the modification of $P(t)$ and is known as a rank-k update of $P(t)$.

The expressions for the modified $K(t+1)$ can be derived in a similar fashion as in the previous section. Now (23) is instead

$$\begin{bmatrix} \tilde{G}(t+1) & g \\ g^T & g^0 \end{bmatrix} \underbrace{\begin{bmatrix} K_1 \\ k_1^T \end{bmatrix}}_{K(t+1)} = -\begin{bmatrix} \tilde{H}^T(t+1) \\ h^T \end{bmatrix}, \tag{31}$$

and by identifying the changes to the first row in (25), $\tilde{K}(t+1)$ can be calculated as

$$\tilde{K}(t+1) = K_1 + \tilde{G}^{-1}(t+1)gC_+^{-1}(t)V_+^T(t), \tag{32}$$

$$K(t+1) = \begin{bmatrix} K_1 \\ k_1^T \end{bmatrix}. \tag{33}$$

C. The impact on the subsequent time-steps $t_m - 1, \ldots, 0$

In Section V-A and V-B it was shown that removing or adding k input constraints to the working-set at time t_m (in Algorithm 1) result in a rank-k modification of $P(t_m)$ in either of the forms (22) or (30). This will affect the matrices in Algorithm 1 in the subsequent time-steps $t = t_m - 1, \ldots, 0$. For a rank-k downdate as in (22) straightforward calculations give, by inserting (22) in Line 3 to 5 in Algorithm 1,

$$\begin{aligned} \tilde{F}(t+1) &= Q_x(t) + A^T(t)\tilde{P}(t+1)A(t) = \\ &Q_x(t) + A^T(t)P(t+1)A(t) - \\ &A^T(t)V_-(t+1)C_-^{-1}(t+1)V_-(t+1)^T A(t) = \\ &F(t+1) - A^T(t)V_-(t+1)C_-^{-1}(t+1)V_-^T(t+1)A(t), \end{aligned} \tag{34}$$

$$\begin{aligned} \tilde{G}(t+1) &= Q_w(t) + B_w^T(t)\tilde{P}(t+1)B_w(t) = \\ &Q_w(t) + B_w^T(t)P(t+1)B_w(t) - \\ &B_w^T(t)V_-(t+1)C_-^{-1}(t+1)V_-^T(t+1)B_w(t) = \\ &G(t+1) - B_w^T(t)V_-(t+1)C_-^{-1}(t+1)V_-^T(t+1)B_w(t), \end{aligned} \tag{35}$$

and

$$\tilde{H}(t+1) = Q_{xw}(t) + A^T(t)\tilde{P}(t+1)B_w(t) = Q_{xw}(t) + A^T(t)P(t+1)B_w(t) - A^T(t)V_-(t+1)C_-^{-1}(t+1)V_-^T(t+1)B_w(t) = H(t+1) - A^T(t)V_-(t+1)C_-^{-1}(t+1)V_-^T(t+1)B_w(t). \quad (36)$$

This can be written in block matrix form as

$$\tilde{M}(t+1) = \quad (37)$$

$$\begin{bmatrix} \tilde{F}(t+1) & \tilde{H}(t+1) \\ \tilde{H}^T(t+1) & \tilde{G}(t+1) \end{bmatrix} = \begin{bmatrix} F(t+1) & H(t+1) \\ H^T(t+1) & G(t+1) \end{bmatrix} - \begin{bmatrix} A^T(t)V_-(t+1) \\ B_w^T(t)V_-(t+1) \end{bmatrix} C_-^{-1}(t+1) \begin{bmatrix} A^T(t)V_-(t+1) \\ B_w^T(t)V_-(t+1) \end{bmatrix}^T \quad (38)$$

Now define $\hat{M}(t+1)$ and $\bar{M}(t+1)$ as

$$\hat{M}(t+1) = \begin{bmatrix} F(t+1) & H(t+1) & A^T(t)V_-(t+1) \\ H^T(t+1) & G(t+1) & B_w^T(t)V_-(t+1) \\ V_-^T(t+1)A(t) & V_-^T(t+1)B_w(t) & C_-(t+1) \end{bmatrix}, \quad (39)$$

$$\bar{M}(t+1) = \begin{bmatrix} G(t+1) & B_w^T(t)V_-(t+1) \\ V_-^T(t+1)B_w(t) & C_-(t+1) \end{bmatrix}. \quad (40)$$

Then it holds that $\tilde{M}(t+1) = \hat{M}(t+1)/C_-(t+1)$. Also, $\tilde{P}(t) = \tilde{M}(t+1)/\tilde{G}(t+1) = \hat{M}(t+1)/\bar{M}(t+1)$ which by using (51) gives (omitting arguments)

$$\tilde{P}(t) = \underbrace{F - HG^{-1}H^T}_{P(t)} - \underbrace{(A^T - HG^{-1}B_w^T)V_-}_{V_-(t)} \underbrace{(C_- - V_-^T B_w G^{-1} B_w^T V_-)}_{C_-(t)}{}^{-1} \underbrace{V_-^T (A^T - HG^{-1}B_w^T)^T}_{V_-^T(t)}, \quad (41)$$

where $C_-(t) \in \mathbb{S}_{++}^k$ and $V_-(t) \in \mathbb{R}^{n\times k}$ have full column rank. Thus, a rank-k downdate of $P(t+1)$ will cause a rank-k downdate also in $P(t)$.

Similar calculations for a rank-k update of $P(t+1)$, i.e. $\tilde{P}(t+1) = P(t+1) + V_+(t+1)C_+^{-1}(t+1)V_+^T(t+1)$, gives

$$\tilde{F}(t+1) = F(t+1) + A^T(t)V_+(t+1)C_+^{-1}(t+1)V_+^T(t+1)A(t) \quad (42)$$

$$\tilde{G}(t+1) = G(t+1) + B_w^T(t)V_+(t+1)C_+^{-1}(t+1)V_+^T(t+1)B_w(t) \quad (43)$$

$$\tilde{H}(t+1) = H(t+1) + A^T(t)V_+(t+1)C_+^{-1}(t+1)V_+^T(t+1)B_w(t). \quad (44)$$

The updated $P(t)$ is

$$\tilde{P}(t) = P(t) + V_+(t)C_+^{-1}(t)V_+^T(t), \quad (45)$$

with

$$C_+(t) \triangleq C_+(t+1) - V_+^T(t+1)B_w(t)\tilde{G}^{-1}(t+1)B_w^T(t)V_+(t+1) \quad (46)$$

$$V_+(t) \triangleq (A^T(t) - \tilde{H}(t+1)\tilde{G}^{-1}(t+1)B_w^T(t))V_+(t+1). \quad (47)$$

Hence, analogously to a downdate, a rank-k update of $P(t+1)$ will result in a rank-k update of $P(t)$. This is summarized in Lemma 1.

Lemma 1: Consider a rank-k modification of $P(t)$ for a single time instant $t_m \in \{1, \dots, N\}$ in either of the forms

$$\begin{cases} \tilde{P}(t_m) = P(t_m) - V_-(t_m)C_-^{-1}(t_m)V_-^T(t_m) \text{ (downdate)} \\ \tilde{P}(t_m) = P(t_m) + V_+(t_m)C_+^{-1}(t_m)V_+^T(t_m), \text{ (update)} \end{cases} \quad (48)$$

where the corresponding $C_-(t_m), C_+(t_m) \in \mathbb{S}_{++}^n$ and $V_-(t_m), V_+(t_m) \in \mathbb{R}^{n\times k}$ have rank k. Then, for all $t \in \{t_m - 1, \dots, 0\}$, it holds that $\tilde{P}(t)$ is modified as

$$\begin{cases} \tilde{P}(t) = P(t) - V_-(t)C_-^{-1}(t)V_-^T(t) \text{ (downdate)} \\ \tilde{P}(t) = P(t) + V_+(t)C_+^{-1}(t)V_+^T(t), \text{ (update)} \end{cases} \quad (49)$$

with $C_-(t), C_+(t) \in \mathbb{S}_{++}^n$ invertible and $V_-(t), V_+(t) \in \mathbb{R}^{n\times k}$ of rank k.

Proof: The result follows immediately from the derivations above combined with an induction argument; a rank-k modification of $P(t)$ implies a corresponding rank-k modification of $P(t-1)$. Since this holds for all t of interest, the desired result follows. ■

D. Algorithms for efficiently modifying Riccati factorizations

The theory in Sections V-A to V-C are summarized in Algorithm 5. Since locking or releasing control input constraints at time t_m will only affect matrices for $t = t_m, \dots, 0$, it is only necessary to modify the matrices in Algorithm 1 for these time indices. In Algorithm 5, the first step is to assign the correct value to α depending on whether constraints are added or removed. Then $g, g^0, h, V_{+/-}(t_m), C_{+/-}(t_m), k_1$ and K_1 are calculated according to Sections V-A and V-B, and the matrices $\tilde{H}(t_m+1)$, $\tilde{K}(t_m+1)$ and $\tilde{P}(t_m+1)$ are modified. The factorization of $\tilde{G}(t_m+1)$ is modified using standard methods for up- and downdating, e.g., Cholesky factorizations and should not be calculated from scratch. These steps are followed by the main loop that modifies all the matrices for $t < t_m$, and the rank-k modification of the Riccati factorization is completed.

Modifying the factorization instead of re-computing it reduces the computational complexity from approximately $N(1/3n_u^3 + 4n^3 + 4n_u^2 n + 6n^2 n_u)$ flops to roughly $t_m(2n_u^2 n + 7n_u^2 + 10kn_u n + 8kn^2)$ for a rank-k modification. The analytical complexity is illustrated in Fig. 1, where the reduction in terms of computational complexity of the modification algorithm over the standard (still tailored) algorithm is clear (for problem sizes $n \geq 3$). Note that a standard dense method that does not utilize the structure in the MPC-problem in (6) has a computational complexity of order $\mathcal{O}(N^3 n_u^3)$, which is significantly larger than both Algorithm 1 and 5.

Since the system of equations need to be re-solved after a rank-k modification of the factorization, also Algorithm 2, 3 and 4 have to be executed to get a new solution, but it is clear that the backward recursion in Algorithm 2 is only affected for $t \leq t_m$. Hence, it is only partly executed in order to improve performance even further. The new, modified version of the backward recursion is given by Algorithm 6.

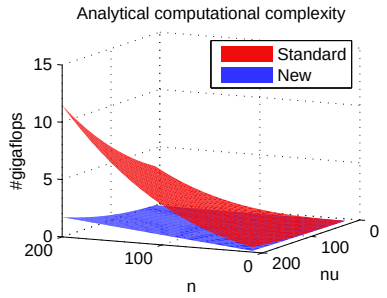


Fig. 1: The analytical complexity for the two tailored algorithms when $N = 100$ and the full factorization is modified are computed as the number of floating point operations, and the results indicate a significantly lower complexity of the new modification algorithm (blue surface) for all investigated instances.

Algorithm 5 Low-rank modifications of Riccati factorization

1: Assign $\alpha := 1$ (update) or $\alpha := -1$ (downdate)
2: Calculate $g, g^0, h, V(t_m), C(t_m), k_1$ and K_1
3: Modify the factorization of $G(t_m + 1)$
4: Calculate $\tilde{H}(t_m + 1)$ and $\tilde{K}(t_m + 1)$
5: $\tilde{P}(t_m) := P(t_m) + \alpha V(t_m)C^{-1}(t_m)V^T(t_m)$
6: **for** $t = t_m - 1, \ldots, 0$ **do**
7: $\tilde{F}(t+1) := F(t+1) + \alpha A^T(t)V(t+1)C^{-1}(t+1)V^T(t+1)A(t)$
8: Modify factorization for $\tilde{G}(t+1) = G(t+1) + \alpha B_w^T(t)V(t+1)C^{-1}(t+1)V^T(t+1)B_w(t)$
9: $\tilde{H}(t+1) := H(t+1) + \alpha A^T(t)V(t+1)C^{-1}(t+1)V^T(t+1)B_w(t)$
10: Compute a solution $\tilde{K}(t+1)$ to $\tilde{G}(t+1)\tilde{K}(t+1) = -\tilde{H}^T(t+1)$
11: **if** $\alpha == 1$ **then**
12: $C(t) := C(t+1) - V^T(t+1)B_w(t)\tilde{G}(t+1)^{-1}B_w^T(t)V(t+1)$
13: $V(t) := \big(A(t) + B_w(t)\tilde{K}(t+1)\big)^T V(t+1)$
14: **else**
15: $C(t) := C(t+1) - V^T(t+1)B_w(t)G(t+1)^{-1}B_w^T(t)V(t+1)$
16: $V(t) := \big(A(t) + B_w(t)K(t+1)\big)^T V(t+1)$
17: **end if**
18: $\tilde{P}(t) := P(t) + \alpha V(t)C^{-1}(t)V^T(t)$
19: **end for**

Algorithm 6 Backward recursion, modified

1: **for** $\tau = t_m, \ldots, 0$ **do**
2: $u_0(\tau+1) = \tilde{G}^{-1}(\tau+1)B_w^T(\tau)\Psi(\tau+1)$
3: $\Psi(\tau) = A^T(\tau)\Psi(\tau+1) - \tilde{H}(\tau+1)u_0(\tau+1)$
4: **end for**

VI. Numerical Experiments

The modification algorithm has been evaluated and compared to the standard full Riccati factorization. It is an evaluation of the algorithms' relative performance, and not absolute times. The main goal with the numerical experiments is to show that Algorithm 5 works, and also show the performance gains in computational complexity. In the implemented version of Algorithm 1 a Cholesky factorization of $G(t+1)$ has been used at Line 6, and for Algorithm 5, standard Cholesky up- and downdating methods have been used at Line 3 and 8. All simulations for the evaluation have been made using randomized stable LTI-systems, which are averaged over several runs.

The simulations were performed on an Intel® Xeon® X5675 processor running Linux (version 2.6.32-279.19.1.el6.x86_6) and MATLAB (version 8.0.0.783, R2012b).

A. Modified Riccati recursion

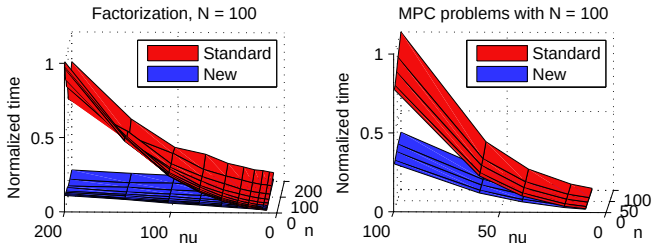
The new modification algorithm given as Algorithm 5 is evaluated through simulations for systems of different orders. The maximum number of computational threads was set to one, preventing MATLAB from using multi-threaded linear algebra computations. Time was measured using the commands `tic` and `toc`. In order to get a fair comparison between the two algorithms, as many operations as possible have been implemented in m-code. Standard MATLAB implementation of some operations, e.g., basic matrix operators such as multiplication, addition etc., were however used. Note that the numerical results are thus in favour of Algorithm 1, since the dominating n^3-complexities are efficiently implemented in MATLAB.

The prediction horizon has been chosen to $N = 100$ in all simulations and the full factorization is modified, giving the worst case scenario for the modifying algorithm. The state and control signal dimensions (n and n_u) are chosen from the interval $[10, 200]$. Fig. 2a presents the normalized computation times for both algorithms. The computation times have been normalized to stress the relative performance of the two algorithms. From the figure it is clear that the new modification algorithm requires significantly lower computational times than the standard Riccati recursion. It is also clear that the computational time is mostly dependent on the number of control signals, i.e. n_u. This could be due to the fact that all n^3-complexities stem from matrix multiplications, which are very efficiently implemented in MATLAB, while the n_u^3-complexity is due to the Cholesky factorization which was implemented in m-code. Slicing the surface plot along $n = 20$ gives Fig. 3a where the flat line for the modification algorithm is seen as the solid line. A similar plot is seen in Fig. 3b where the slicing is along the n_u-axis.

Although the Riccati factorization (Algorithm 1) is already very efficient compared to standard methods, it is clear that Algorithm 5 outperforms it for almost all system dimensions. There are however some cases where the opposite is true and those are supposed to be due to implementation issues that can be solved, which is indicated by the analytical complexity in Fig. 1.

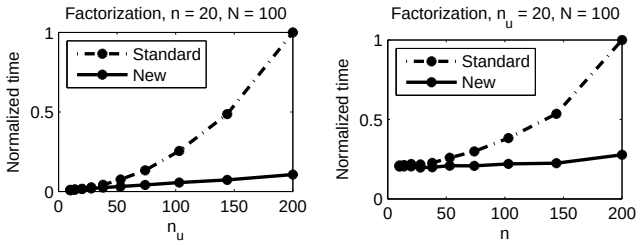
B. Traditional active-set solver

Once again it is stressed that the purpose with this paper is not to present a complete solver; the focus is the computation of the search direction. However, for completeness some



(a) (b)

Fig. 2: (a) Computational times for computing a single search direction using Algorithm 1 (Standard) and Algorithm 5 (New). (b) Computational times for a complete QP-solver involving several search direction computations using Algorithm 1 (Standard) and Algorithm 5 (New).



(a) (b)

Fig. 3: Computation times for the full Algorithm 1 (dash-dotted) and modification algorithm (solid). (a) With state dimension $n = 20$ kept fixed, (b) with control signal dimension $n_u = 20$ kept fixed.

simple examples are solved to illustrate how it works in a traditional active-set method. The basic AS solver in [15] has been implemented and used to solve random stable systems. The step directions were computed using both the standard algorithm (Algorithm 1) and the new, modification algorithm (Algorithm 5). Fig. 2b shows the normalized computation times for the solvers. The solver with the new, modification algorithm for step direction computation has significantly lower execution times than the solver that re-computes the factorization in each iteration.

VII. Conclusions

The main contribution in this paper is algorithms for efficient low-rank modifications of a Riccati factorization. The theory is developed as a proof of concept for a positive definite problem with only simple control signal constraints. The performance gain in terms of reduced computational complexity and computation times are significant already for quite small systems, and they are expected to be possible to reduce even further with an improved implementation in compiled programming languages. The algorithms have been tested in a traditional MPC solver with good results, but they can also be combined with more recent algorithms like the one using projections presented in [9] and [10] which has shown very promising performance already without exploiting low-rank modifications. For future work, systems with state-constraints as well as problems where $Q_u(t) \in \mathbb{S}_+^{n_u}$ will be considered to complete the theory.

Appendix

Let M and $\bar{M}$ be matrices partitioned into 3×3 and 2×2 blocks respectively,

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{12}^T & M_{22} & M_{23} \\ M_{13}^T & M_{23}^T & M_{33} \end{bmatrix}, \quad \bar{M} = \begin{bmatrix} M_{22} & M_{23} \\ M_{23}^T & M_{33} \end{bmatrix} \tag{50}$$

and define the Schur complement of $\bar{M}$ with respect to M_{22} as $\bar{M}/M_{22} = M_{33} - M_{23}^T M_{22}^{-1} M_{23}$. Then,

$$M/\bar{M} = M_{11} - M_{12}M_{22}^{-1}M_{12}^T - \ldots \\ \big(M_{13} - M_{12}M_{22}^{-1}M_{23}\big) S^{-1} \big(M_{13} - M_{12}M_{22}^{-1}M_{23}\big)^T \tag{51}$$

where $S = \bar{M}/M_{22}$.

References

[1] H. Jonson, "A Newton method for solving non-linear optimal control problems with general constraints," Ph.D. dissertation, Linköpings Tekniska Högskola, 1983.

[2] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *Journal of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, Dec. 1998.

[3] A. Hansson, "A primal-dual interior-point method for robust optimal control of linear discrete-time systems," *IEEE Transactions on Automatic Control*, vol. 45, no. 9, pp. 1639–1655, Sep. 2000.

[4] R. A. Bartlett, L. T. Biegler, J. Backstrom, and V. Gopal, "Quadratic programming algorithms for large-scale model predictive control," *Journal of Process Control*, vol. 12, pp. 775–795, 2002.

[5] L. Vandenberghe, S. Boyd, and M. Nouralishahi, "Robust linear programming and optimal control," Department of Electrical Engineering, University of California Los Angeles, Tech. Rep., 2002.

[6] M. Åkerblad and A. Hansson, "Efficient solution of second order cone program for model predictive control," *International Journal of Control*, vol. 77, no. 1, pp. 55–77, 2004.

[7] D. Axehill and A. Hansson, "A mixed integer dual quadratic programming algorithm tailored for MPC," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Manchester Grand Hyatt, San Diego, USA, Dec. 2006, pp. 5693–5698.

[8] D. Axehill, A. Hansson, and L. Vandenberghe, "Relaxations applicable to mixed integer predictive control – comparisons and efficient computations," in *Proceedings of the 46th IEEE Conference on Decision and Control*, Hilton New Orleans Riverside, New Orleans, USA, Dec. 2007, pp. 4103–4109.

[9] D. Axehill, "Integer quadratic programming for control and communication," Ph.D. dissertation, Linköping Univ., 2008. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-10642

[10] D. Axehill and A. Hansson, "A dual gradient projection quadratic programming algorithm tailored for model predictive control," in *Proceedings of the 47th IEEE Conference on Decision and Control*, Fiesta Americana Grand Coral Beach, Cancun, Mexico, Dec. 2008, pp. 3057–3064.

[11] M. Diehl, H. J. Ferreau, and N. Haverbeke, *Nonlinear Model Predictive Control*. Springer Berlin / Heidelberg, 2009, ch. Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation, pp. 391–417.

[12] C. Kirches, H. G. Bock, J. P. Schlöder, and S. Sager, "A factorization with update procedures for a KKT matrix arising in direct optimal control," *Mathematical Programming Computation*, vol. 3, no. 4, pp. 319–348, 2011.

[13] B. Lie, M. Dueñas Díez, and T. A. Hauge, "A comparison of implementation strategies for MPC," *Modeling, identification and control*, vol. 26, no. 1, pp. 39–50, Jan. 2005.

[14] N. I. M. Gould and P. L. Toint, "A quadratic programming bibliography," Numerical Analysis Group, Rutherford Appleton Laboratory, Tech. Rep., Feb. 2001.

[15] J. Nocedal and S. Wright, *Numerical Optimization*. Springer-Verlag, 2006.

[16] F. Zhang, *The Schur complement and its applications*. Springer, 2005, vol. 4.

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