A Dual Active-Set Solver for Embedded Quadratic Programming Using Recursive LDLT Updates

Daniel Arnström, Alberto Bemporad and Daniel Axehill

Arnström, D., Bemporad, A., Axehill, D., (2022), A Dual Active-Set Solver for Embedded Quadratic Programming Using Recursive LDLT Updates, *IEEE Transactions on Automatic Control*, 67(8), 4362-4369. https://doi.org/10.1109/TAC.2022.3176430

Original publication available at:
https://doi.org/10.1109/TAC.2022.3176430

A Dual Active-Set Solver for Embedded Quadratic Programming Using Recursive LDL^T Updates

Daniel Arnström, Alberto Bemporad, *Fellow, IEEE*, and Daniel Axehill, *Member, IEEE*

***Abstract*—In this technical note we present a dual active-set solver for quadratic programming that has properties suitable for use in embedded model predictive control applications. In particular, the solver is efficient, can easily be warm-started, and is simple to code. Moreover, the exact worst-case computational complexity of the solver can be determined offline and, by using outer proximal-point iterations, ill-conditioned problems can be handled in a robust manner.**

***Index Terms*—Quadratic programming, model predictive control, embedded optimization**

I. Introduction

EFFICIENT, reliable, and simple quadratic programming (QP) solvers are essential when model predictive control (MPC) is used on embedded systems in real-time applications, where a QP has to be solved at each time step under real-time constraints with limited memory and computational resources.

Popular methods for solving these QPs are active-set methods [1]–[4], interior-point methods [5], [6], gradient projection methods [7]–[10], and operator splitting methods [11].

In particular, the active-set method in [1] (QPNNLS), which is based on reformulating the QP as a nonnegative least-squares (NNLS) problem, is simple to implement and has proven to be efficient for solving small to medium size QP problems. Furthermore, its reliability has been improved greatly in [12] where outer proximal-point iterations are used to improve its numerical stability, and in [13], where QPNNLS is shown to be closely related to a primal active-set QP method applied to the dual problem, allowing the complexity certification method in [14] to be used to determine the exact computational complexity of QPNNLS.

In this technical note we use insights from [13] to propose a dual active-set method for quadratic programming which retains the favorable properties of QPNNLS (efficiency and simplicity) by making recursive updates to an LDL^T factorization. In addition to retaining favorable properties, we show that operating directly on the dual QP instead of the NNLS reformulation used in [1] yields additional improvements: (i) Direct reusability of matrix factors when the linear term in the objective function and the constant term in the constraints change, which is relevant for MPC and when the active-set method is combined with outer proximal-point iterations. (ii) Improved numerical stability already in the setup without proximal-point iterations. (iii) Improved efficiency by reducing intermediate computations stemming from the NNLS reformulation.

The main contribution of this paper is, hence, showing how the properties of QPNNLS can be retained and improved by directly operating on the dual QP problem instead of an NNLS problem. Concretely this requires, for example, extending the recursive LDL^T updates to detect and handle singularities. More broadly, we show that a simple dual active-set solver that performs such recursive LDL^T updates can outperform state-of-the-art solvers on small to medium size QPs, which are often encountered in embedded MPC applications.

Problem formulation: We consider QPs in the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & J(x) \triangleq \frac{1}{2}x^T H x + f^T x \\ \text{subject to} \quad & Ax \leq b, \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$. The objective function J is characterized by $H \in \mathbb{S}^n_{++}$ and $f \in \mathbb{R}^n$, and the feasible set is characterized by $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Furthermore, the solution to (1) is denoted by x^*.

Remark 1 (Relaxing strict convexity): The proposed algorithm requires H to be invertible. The case when $H \succeq 0$ can, however, be handled by performing proximal-point iterations, described in Section IV-E.

A set of necessary and, because of the convexity of (1), sufficient conditions for optimality of x^* are the KKT-conditions:

$$Hx^* + A^T\lambda^* = -f, \tag{2a}$$
$$Ax^* \leq b,\ \lambda^* \geq 0, \tag{2b}$$
$$[b - Ax^*]_i[\lambda^*]_i = 0,\ \ \forall i = 1, \ldots, m, \tag{2c}$$

for $\lambda^* \in \mathbb{R}^m$, and where the operator $[\cdot]_i$ extracts the ith row of a matrix, or the ith entry of a vector.

Instead of solving (1) directly, we will solve its so-called *dual problem*:

$$\underset{\lambda\geq 0}{\text{minimize}}\ J_d(\lambda) \triangleq \frac{1}{2}\lambda^T M M^T \lambda + d^T\lambda, \tag{3}$$

where we have, similar to [1], introduced

$$M \triangleq AR^{-1}, \quad v \triangleq R^{-T}f, \quad d \triangleq b + Mv, \tag{4}$$

and where R is an upper triangular Cholesky factor of H ($H = R^TR$). The solution λ^* to (3) satisfies the same KKT-conditions as (1) (see, e.g., [15]) and x^* can, hence, be recovered from λ^* through (2a) when $H \succ 0$.

This work was supported by the Swedish Research Council (VR) under contract number 2017-04710.

D. Arnström and D. Axehill are with the Division of Automatic Control, Linköping University, Sweden daniel.{arnstrom,axehill}@liu.se

A. Bemporad is with the Department of Computer Science and Engineering, IMT School for Advanced Studies Lucca, Lucca, Italy alberto.bemporad@imtlucca.it

Notation: Since the proposed method, soon to be introduced, is an iterative method, λ_k denotes the value of the dual iterate at iteration k. Furthermore, $\mathcal{W}_k$ is the so-called *working set*, which contains indices of the components of λ_k that are free to vary (which can be interpreted as imposing the corresponding primal constraints to hold with equality). Conversely, $\overline{\mathcal{W}}_k$ contains the components of λ_k which are fixed at zero. For M and d, which are constant in all iterations, we let M_k and d_k denote the rows of M and d indexed by $\mathcal{W}_k$, respectively. Likewise, $\overline{M}_k$ and $\overline{d}_k$ denote the rows of M and d indexed by $\overline{\mathcal{W}}_k$, respectively. Finally, $\ker(A)$ denotes the kernel of a matrix A.

II. A Dual Active-Set Algorithm

The dual active-set algorithm that we propose, given in Algorithm 1, can be interpreted as the primal active-set algorithm considered in [14] applied to the dual problem in (3), which is mathematically equivalent to several other popular active-set algorithm formulations (see Section III-C for details and advantages of the proposed formulation). We will now give an overview of the algorithm and then cover specifics, such as how to efficiently solve the subproblems encountered in Step 3 and 12, in Section II-B.

Algorithm 1 Dual active-set method for solving (1).

Input: $M, d, v, R^{-1}, \mathcal{W}_0, \lambda_0$
Output: $x^*, \lambda^*, \mathcal{A}^*$

1: **while** true **do**
2: **if** $M_k M_k^T$ is nonsingular **then**
3: $[\lambda_k^*]_{\mathcal{W}_k} \leftarrow$ solution to $M_k M_k^T [\lambda_k^*]_{\mathcal{W}_k} = -d_k$
4: **if** $\lambda_k^* \geq 0$ **then**
5: $[\mu_k]_{\overline{\mathcal{W}}_k} \leftarrow \overline{M}_k M_k^T [\lambda_k^*]_{\mathcal{W}_k} + \overline{d}_k$, $\lambda_{k+1} \leftarrow \lambda_k^*$
6: **if** $\mu_k \geq 0$ **then** optimum found, **goto** 16
7: **else** $j \leftarrow \operatorname{argmin}_{i \in \overline{\mathcal{W}}_k} [\mu_k]_i$, $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \cup \{j\}$
8: **else**
9: $p_k \leftarrow \lambda_k^* - \lambda_k$, $\mathcal{B} \leftarrow \{i \in \mathcal{W}_k : [\lambda_k^*]_i < 0\}$
10: $[\lambda_{k+1}, \mathcal{W}_{k+1}] \leftarrow$ FIXCOMPONENT$(\lambda_k, \mathcal{W}_k, \mathcal{B}, p_k)$
11: **else** ($M_k M_k^T$ singular)
12: $[p_k]_{\mathcal{W}_k} \leftarrow$ solution to $M_k M_k^T [p_k]_{\mathcal{W}_k} = 0$, $p_k^T d < 0$
13: $\mathcal{B} \leftarrow \{i \in \mathcal{W}_k : [p_k]_i < 0\}$
14: $[\lambda_{k+1}, \mathcal{W}_{k+1}] \leftarrow$ FIXCOMPONENT$(\lambda_k, \mathcal{W}_k, \mathcal{B}, p_k)$
15: $k \leftarrow k + 1$
16: **return** $x^* \leftarrow -R^{-1}(M_k^T [\lambda_k^*]_{\mathcal{W}_k} + v), \lambda_k^*, \mathcal{W}_k$

17: **procedure** FIXCOMPONENT$(\lambda_k, \mathcal{W}_k, \mathcal{B}, p_k)$
18: $j \leftarrow \operatorname{argmin}_{i \in \mathcal{B}} -[\lambda_k]_i / [p_k]_i$
19: $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \setminus \{j\}$, $\lambda_{k+1} \leftarrow \lambda_k - ([\lambda_k]_j / [p_k]_j) p_k$

A. Algorithm overview

Like any other active-set method, Algorithm 1 iteratively updates the working set $\mathcal{W}$. An iteration always starts by solving an equality constrained subproblem defined by the current working set $\mathcal{W}_k$:

$$\begin{aligned} \min_{\lambda} \quad & \frac{1}{2} \lambda^T M M^T \lambda + d^T \lambda \\ \text{s.t.} \quad & [\lambda]_i = 0, \ \forall i \notin \mathcal{W}_k, \end{aligned} \tag{5}$$

where k is the current iteration. By using the constraint $[\lambda]_{\overline{\mathcal{W}}_k} = 0$ to eliminate variables, (5) is equivalent to the unconstrained problem

$$\min_{[\lambda]_{\mathcal{W}_k}} \frac{1}{2} [\lambda]_{\mathcal{W}_k}^T M_k M_k^T [\lambda]_{\mathcal{W}_k} + d_k^T [\lambda]_{\mathcal{W}_k}, \quad [\lambda]_{\overline{\mathcal{W}}_k} = 0. \tag{6}$$

If $M_k M_k^T \succ 0$, the unconstrained problem in (6) has a unique solution and the solution λ_k^* to (5) is then given by

$$M_k M_k^T [\lambda_k^*]_{\mathcal{W}_k} = -d_k, \quad [\lambda_k^*]_{\overline{\mathcal{W}}_k} = 0. \tag{7}$$

If $\lambda_k^* \geq 0$, we set $\lambda_{k+1} \leftarrow \lambda_k^*$ and check for primal feasibility (see below). Otherwise, a line-search along the line-segment connecting λ_k and λ_k^* is performed and the first component which becomes zero is removed from $\mathcal{W}_k$, i.e., is fixed at zero.

If $M_k M_k^T$ is singular and $d_k^T p \neq 0$ for some $p \in \ker(M_k M_k^T)$, there is no bounded solution to (6), i.e., the objective function can be made arbitrarily small by moving in a direction p_k which satisfies

$$[p_k]_{\overline{\mathcal{W}}_k} = 0, \quad M_k M_k^T [p_k]_{\mathcal{W}_k} = 0, \quad d^T p_k < 0. \tag{8}$$

Hence, a line-search along the ray $\lambda_k + \alpha p_k, \alpha > 0$, is performed. As is discussed in detail in Section IV-A, at least one component of λ in $\mathcal{W}_k$ will become zero while moving along this ray if (1) is feasible. Again, the first component which becomes zero is removed from $\mathcal{W}_k$, i.e., is fixed at zero.

Remark 2 (Impossibility of $d^T p_k = 0$): For Algorithm 1, one can show that once $M_k M_k^T$ becomes singular, there always exists a solution to (8), see, e.g., Lemma 3.5 in [16] for details.

When $\lambda_k^* \geq 0$, primal feasibility for the constraints not in $\mathcal{W}_k$ is checked by computing the primal slack (which is the dual vector of (3))

$$[\mu_k]_{\overline{\mathcal{W}}_k} = \overline{M}_k M_k \lambda_k^* + \overline{d}_k. \tag{9}$$

Primal feasibility is satisfied if $\mu_k \geq 0$ and, since stationarity, dual feasibility, and complementary slackness already hold, λ_k^* is optimal. Otherwise, the most negative component of μ_k is added to $\mathcal{W}_k$ (making the corresponding component of λ free to vary).

Remark 3 (Selection rule): Adding *the most* negative component of μ_k to $\mathcal{W}_k$ is a common rule in practice, but adding *any* negative component of μ_k to $\mathcal{W}_k$ also leads to convergence.

Remark 4 (Primal feasibility tolerance): When implemented in practice, $\mu_k \geq -\epsilon_p$ is considered instead of $\mu_k \geq 0$ in Step 6 for numerical reasons, where $\epsilon_p > 0$ is the tolerance for primal feasibility. Similar tolerances should also be used for the inequalities in Steps 4, 9, and 13.

After $\mathcal{W}_k$ has been changed by either adding or removing an index to get a new working set $\mathcal{W}_{k+1}$, the algorithm starts another iteration by solving (6) for $\mathcal{W}_{k+1}$ (or by solving (8) if $M_{k+1} M_{k+1}^T$ is singular) and the steps described above are repeated until convergence.

The convergence of Algorithm 1 can be proven by standard arguments for active-set methods (cf. e.g., Section 4 in [17] or Section 3 in [18]).

B. LDL^T factorization

The matrix $M_k M_k^T$ is central in Algorithm 1, partly because of whether it is singular or not results in different modes (Steps 2-10 or Steps 11-14, respectively), but also since it is used to compute λ_k^* in Step 3 or p_k in Step 12. We will now show that the above-mentioned operations can be efficiently performed by factorizing $M_k M_k^T = LDL^T$, where L is a lower unit triangular matrix and D is a diagonal matrix. In particular, we show in Section II-B1 that the singularity of $M_k M_k^T$ can easily be indentified, and in Section II-B2 and Section II-B3 we show that the system of linear equations defining λ_k^* and p_k can be efficiently solved. Moreover, since only a single row of M_k is either added or removed between iterations in Algorithm 1, L and D can be recursively updated, as described in Section II-B4, which reduces the computational complexity of the algorithm significantly.

1) Detecting singularity: Given an L and D it is straightforward to determine whether $M_k M_k^T$ is singular or not since this can directly be seen in D:

Lemma 1: Assume that there exist L and D such that $M_k M_k^T = LDL^T$, with L being a lower unit triangular matrix and D being a diagonal matrix with nonnegative elements. Then $[D]_{ii} \neq 0, \forall i \Leftrightarrow M_k M_k^T$ is nonsingular

Proof: Directly follows from L having full rank and that D is a diagonal matrix. ∎

2) Solving nonsingular KKT-systems: When $M_k M_k^T$ is nonsingular, solving $M_k M_k^T [\lambda_k^*]_{\mathcal{W}_k} = -d_k$ given an LDL^T factorization of $M_k M_k^T$ can be done by solving two triangular linear systems:

$$y \leftarrow \text{Solve } Ly = -d_k \text{ (forward substitution)}, \quad (10a)$$

$$z \leftarrow \text{Scale } y \text{ with } D, \quad [z]_i = \tfrac{[y]_i}{[D]_{ii}}, \quad (10b)$$

$$[\lambda_k^*]_{\mathcal{W}_k} \leftarrow \text{Solve } L^T [\lambda_k^*]_{\mathcal{W}_k} = z \text{ (backward substitution)}. \quad (10c)$$

Remark 5 (Efficient forward substitution): Because a certain number of first rows of the lower-triangular matrix L and of d_k and D remain constant between iterations, both the forward substitution and scaling in (10a) and (10b) do not have to be done from scratch at each iteration of Algorithm 1. The amount of previous computations that can be reused depends on how much the working set $\mathcal{W}_k$ changes. For example, only the last element of y in (10a) has to be solved for after a constraint is added to $\mathcal{W}_k$.

3) Solving singular KKT-systems: Next, we consider the case when $M_k M_k^T$ is singular, which, from Lemma 1, means that at least one diagonal element of D is zero. The following lemma shows that the LDL^T factorization can be used to efficiently compute a p which satisfies $M_k M_k^T p = 0$.

Lemma 2: Assume that $M_k M_k^T = LDL^T$ with the ith diagonal element of D being zero, i.e.,

$$L = \begin{bmatrix} L_1 & 0 & 0 \\ l_i^T & 1 & 0 \\ * & * & * \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & D_2 \end{bmatrix}. \quad (11)$$

Let $\tilde{p}$ be the solution to $L_1^T \tilde{p} = -l_i$. Then $p = \begin{bmatrix} \tilde{p}^T & 1 & 0 \end{bmatrix}^T$ satifies $M_k M_k^T p = 0$.

Proof: By multiplying L^T with the given p we get

$$L^T p = \begin{bmatrix} L_1^T & l_i & * \\ 0 & 1 & * \\ 0 & 0 & * \end{bmatrix} \begin{bmatrix} \tilde{p} \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} L_1^T \tilde{p} + l_i \\ 1 \\ 0 \end{bmatrix} = e_i \quad (12)$$

where e_i is the ith unit vector and we have used that $L_1^T \tilde{p} + l_i = 0$. Using (12) and that the ith element of D is zero gives

$$M_k M_k^T p = LDL^T p = LDe_i = 0, \quad (13)$$

proving the lemma. ∎

Hence, we can find a nontrivial null vector of $M_k M_k^T$ by solving an $(i-1)$-dimensional upper unit triangular system of linear equations by backward substitution. In most cases, $i = |\mathcal{W}_k|$ since $M_k M_k^T$ only becomes singular after an addition of an element to $\mathcal{W}_k$, which in turn implies that the zero element in D will be the last diagonal element (if the updates described below are used).

How null vectors of $M_k M_k^T$ can be used to detect primal infeasibility is discussed in Section IV-A.

4) Updating LDL^T after addition to/removal from $\mathcal{W}$: To recursively update L and D after adding an index to $\mathcal{W}$, we recall the result from Theorem 2 in [1].

Lemma 3: Let L be a unit lower triangular matrix and D be a diagonal matrix such that $M_k M_k^T = LDL^T$. Furthermore let $M^+ = \left[\begin{smallmatrix} M_k \\ [M]_i \end{smallmatrix}\right]$. Then $M^+(M^+)^T = L^+ D^+ (L^+)^T$ with

$$L^+ = \begin{bmatrix} L & 0 \\ l^T & 1 \end{bmatrix}, \quad D^+ = \begin{bmatrix} D & 0 \\ 0 & \delta \end{bmatrix}, \quad (14)$$

where l and δ are defined by

$$LDl = M_k [M]_i^T, \quad \delta = [M]_i [M]_i^T - l^T D l. \quad (15)$$

Proof: Cf. proof of Theorem 2 in [1]. ∎

Similarly, we recall the result from Lemma 2 in [1] for recursively updating L and D after the removal of an index from $\mathcal{W}$.

Lemma 4: Let $M_k M_k^T = LDL^T$ with

$$L = \begin{bmatrix} L_1 & 0 & 0 \\ * & 1 & 0 \\ A & l_i & L_2 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & 0 & 0 \\ 0 & \delta_i & 0 \\ 0 & 0 & D_2 \end{bmatrix}, \quad (16)$$

where l_i and δ_i are placed in the ith column of L and D, respectively, and where L_1 and L_2 are lower unit triangular and D_1 and D_2 are diagonal. Furthermore, let M^- be M_k with the ith row removed. Then $M^-(M^-)^T = L^- D^- (L^-)^T$, where L^- and D^- are given by

$$L^- = \begin{bmatrix} L_1 & 0 \\ A & \tilde{L}_2 \end{bmatrix}, \quad D^- = \begin{bmatrix} D_1 & 0 \\ 0 & \tilde{D}_2 \end{bmatrix}, \quad (17)$$

if $\tilde{L}_2 \tilde{D}_2 \tilde{L}_2^T = L_2 D_2 L_2^T + \delta_i l_i l_i^T$, where $\tilde{L}_2$ is lower unit triangular and $\tilde{D}_2$ is diagonal.

Proof: Cf. the proof of Lemma 2 in [1]. ∎

In Lemma 4, the LDL^T factorization of $L_2 D_2 L_2^T + \delta_i l_i l_i^T$ is a rank-one update of an existing LDL^T factorization, which can be done efficiently by, e.g., Algorithm C1 in [19].

III. Comparison to similar QP algorithms

A. QPNNLS

The proposed method is similar to the QP method described in [1] (QPNNLS), which is based on transforming the QP to a nonnegative least-squares (NNLS) problem. This transformation results in solving least-squares subproblems on the form $\min_y \|A_k^T y - b_k\|_2^2$ at each iteration, which in turn results in solving the linear equation $A_k A_k^T y = A_k b_k$, similar to the linear equation (7) solved in Algorithm 1. Similar to the proposed algorithm, an LDL^T factorization of $A_k A_k^T$ is used to efficiently solve these linear equations. Explicitly, $A_k = [M_k \;\; d_k]$ in [1], and both having M_k and d_k in A_k, in contrast to only M_k as in the proposed method, leads to some undesirable properties.

First of, elements in M_k and d_k can be of different magnitude which can lead to numerical problems. This was partly resolved in [12] by introducing a scaling factor β at the cost of some extra overhead.

Secondly, and more critically, M_k depends on H and A while d_k also depends on f and b. This is of importance when numerical stability of the methods are improved with outer proximal-point iterations [12] and when the QPs are solved in the context of linear MPC. In both of these applications, H and A remain constant while only f and/or b change between QP instances, which means that the computational complexity can be significantly reduced by starting the solver with the previous solution and reusing the L and D factors. For the proposed method this is straightforward since L and D are related to M which, in turn, is related to the unchanged H and A. For QPNNLS, however, L and D are related to M *and* d, where d changes between iterations. Hence, to be able to reuse L and D between QP instances in QPNNLS, two rank-one updates are necessary. These rank-one updates introduce additional complexity compared with the proposed method.

Finally, the NNLS reformulation used in QPNNLS introduces an additional scaling factor that depends on the current iterate, which is not necessary in the proposed method, leading to additional simplifications.

B. Goldfarb-Idnani

Warm-starting the proposed method is straightforward: given $\mathcal{W}_0$, any λ_0 satisfying $[\lambda_0]_{\mathcal{W}_0} \geq 0, [\lambda_0]_{\overline{\mathcal{W}}_0} = 0$, suffices (however $[\lambda_0]_{\mathcal{W}_0} > 0$ is preferable for numerical reasons). Warm-starting is not as straightforward for the popular dual active-set method in [18] (GI), which needs to be started in a nonnegative complementary basic solution, i.e., a nonnegative solution to (7). Warm-starting GI, hence, requires an additional procedure which finds a nonnegative complementary basic solution given an initial working set $\mathcal{W}_0$, increasing the computational burden. In fact, such a procedure is often similar to Steps 9-10 in Algorithm 1, i.e., is already embedded in the proposed method (cf, e.g., Alg. 5.3 in [20]).

Another important difference between GI and the proposed algorithm is that GI, similar to many other active-set QP methods, computes a step direction from the current iterate λ_k to λ_k^* at each iteration, whereas Algorithm 1 computes λ_k^* directly. Some advantages of computing λ_k^* directly are:

(i) The availability of λ_k^* allows for the update $\lambda_{k+1} \leftarrow \lambda_k^*$ in Algorithm 1 when $\lambda_k^* \geq 0$. This means that numerical errors in λ_{k+1} only stem from solving (7) *one* time. In contrast, by computing a step direction, numerical errors in the iterate accumulate each time a step is taken, i.e., there is no "resetting" mechanism for the numerical errors in the iterate in GI, as for Algorithm 1, when $\lambda_k^* \geq 0$.

(ii) Less constraints can be classified as blocking when λ_k^* is computed directly since only negative components of λ_k^* can be blocking while, similarly, all negative components of the step direction are seen as possible blocking constraint for GI, where the former is always a subset of the latter. This ultimately leads to fewer computations for the former when removing constraints from $\mathcal{W}$, since fewer ratio tests (Step 18 in Algorithm 1) have to be done.

(iii) In particular for GI, λ_k^* can not be determined directly from the computed step direction since the step length along the step direction to reach λ_k^* is unknown. Therefore, a primal iterate is also updated and used to determine which step length should be taken along the step direction from the iterate λ_k to reach λ_k^* (this is referred to the full step length in [18]). Performing these primal updates increases the computations needed compared with Algorithm 1, where only the dual iterate λ_k needs to be updated in each iteration.

C. Other active-set methods

As is shown in [21], many active-set methods reported in the literature are mathematically equivalent, in the sense that they produce the same sequence of iterates before reaching the solution, and Algorithm 1 belongs to the family of methods considered therein. For example, the proposed method is mathematically equivalent to the active-set algorithms presented in [17], [18], [22]. The differences between these active-set algorithms are numerical, e.g., how the systems of linear equations are solved and book-keeping of iterates.

The proposed method also shares the interpretation of the dual active-set methods in [23] (QPDAS) and in [24] (DRQP) as a primal active-set method applied to the dual problem (3). Instead of handling unbounded subproblems directly, as is done in the proposed method, QPDAS performs proximal-point iterations on the dual problem. Furthermore, the factorization used for solving the subproblems is different. DRQP differs from the proposed method in that it works on a sparse QP formulation rather than a dense one. By doing so, the subproblems are solved using a Riccati recursion, which leads to a linear computational complexity in the horizon of the MPC problem when solving the subproblems. DRQP cannot, however, detect primal infeasibility directly and does not perform low-rank updates to reduce computations, while the proposed algorithm does both (see Section IV-A and Section II-B4, respectively, for details).

Recursively updating an LDL^T factorization when solving the subproblems in an active-set method is described in the context of primal active-set methods in [19]. Since the constraints are particularly simple for the dual problem (3), some of the computations described in [19] simplify when the factorization is used in the context of the proposed dual active-set method.

IV. Extensions

A. Detecting infeasibility

For a QP method to be reliable it needs to be able to detect if (1) has a primal feasible solution at all, i.e., if $\{x \in \mathbb{R}^n : Ax \leq b\} \neq \emptyset$, otherwise the QP method might not be able to terminate in finite time for infeasible problems. Primal infeasibility can be detected in Step 13 in Algorithm 1 if $\mathcal{B} = \emptyset$, as is shown by the following lemma.

Lemma 5: Let p_k satisfy $[p_k]_{\overline{\mathcal{W}}_k} = 0, M_k M_k^T [p_k]_{\mathcal{W}_k} = 0$, and $d^T p_k < 0$. Furthermore, let $\mathcal{B} \triangleq \{i \in \mathcal{W}_k : [p_k]_i < 0\}$. Then if $\mathcal{B} = \emptyset$, i.e., if $p_k \geq 0$, the QP in (1) is infeasible.

Proof: Inserting αp_k, for an arbitrary constant $\alpha > 0$, in the dual objective gives

$$\begin{aligned} V(\alpha p_k) =& \alpha^2 \frac{1}{2} {p_k}^T M M^T p_k + \alpha d^T p_k \\ =& \alpha^2 \frac{1}{2} [p_k]_{\mathcal{W}_k}^T M_k M_k^T [p_k]_{\mathcal{W}_k} + \alpha d^T p_k = \alpha d^T p_k, \end{aligned}$$

where the second equality follows from $[p_k]_{\overline{\mathcal{W}}_k} = 0$ and the last equality follows from $M_k M_k^T [p_k]_{\mathcal{W}_k} = 0$. Now, since $d^T p_k < 0$, $V(\alpha p_k) \to -\infty$ as $\alpha \to \infty$. Furthermore, αp_k is dual feasible since $\alpha p_k \geq 0, \forall \alpha > 0$ ($p_k \geq 0$ by construction), making (3) unbounded. The desired result follows from an unbounded dual problem being equivalent to an infeasible primal problem, see, e.g., [25, Sec. 5.2.2]. ∎

B. Intermediary lower bounds on $J(x^*)$

In some applications, for example when QP subproblems are solved as a part of solving mixed-integer quadratic programs (MIQPs) with branch-and-bound, having a lower bound on $J(x^*)$ can reduce computations significantly [26].

Since Algorithm 1 operates on the dual problem, the well-known result from convex optimization that the dual function evaluated at a dual feasible point yields lower bounds on $J(x^*)$ (see, e.g., [25, Sec. 5.1.3]) can be used to efficiently compute such bounds. Concretely, we get the lower bound

$$J(x^*) \geq \frac{1}{2}(\|M_k^T \lambda_k^*\|_2^2 - \|v\|_2^2) \tag{18}$$

every time $\lambda_k^* \geq 0$ in Algorithm 1.

Moreover, by inserting λ_k^* in (3) and using (7) one gets that $J_d(\lambda_k^*) = -\frac{1}{2}\|M_k^T \lambda_k^*\|_2^2$, and, since Algorithm 1 is a descent method, $\|M_k^T \lambda_k^*\|_2^2$ will increase in subsequent iterations, resulting in the lower bounds in (18) increasing (and becoming tight once $\mu_k \geq 0$).

Remark 6 (Detecting cycling): $\|M_k^T \lambda_k^*\|_2^2$ can also be used to detect cycling in Algorithm 1, which can occur for ill-conditioned problems due to rounding errors. This can be done by checking whether $\|M_k^T \lambda_k^*\|$ increases every time $\lambda_k^* \geq 0$, which ensures that Algorithm 1 is making progress in each iteration.

C. Bound constraints

Often the constraints in QPs encountered in applications, for example in MPC, are given by upper and lower bounds in the form $b^- \leq Ax \leq b^+$. A naive way of handling these is to reformulate the constraints as $\tilde{A}x \leq \tilde{b}$, with $\tilde{A} \triangleq \left[\begin{smallmatrix} A \\ -A \end{smallmatrix}\right], \tilde{b} \triangleq \left[\begin{smallmatrix} b^+ \\ -b^- \end{smallmatrix}\right]$, which puts the QP in the form in (1). The structure of the bound constraints can, however, be used to reduce the computational complexity and memory footprint of Algorithm 1, primarily when computing μ_k. When exploiting the bound structure, each component of $[\lambda]_i$ corresponds to, instead of just the one-sided constraint $[A]_i x \leq [b]_i$, the two-sided constraint $[b^-]_i \leq [A]_i x \leq [b^+]_i$. To distinguish between whether the upper or lower bound is active, the sets $\mathcal{W}^+$ and $\mathcal{W}^-$, containing components corresponding to active upper and lower bounds, respectively, are introduced. Note that $\mathcal{W}^+ \cup \mathcal{W}^- = \mathcal{W}$.

To determine if optimality has been achieved or whether a constraint needs to be added to $\mathcal{W}$, we consider the primal slacks μ_k^+ and μ_k^- for the upper and lower bounds, respectively, for the constraints not in $\mathcal{W}_k$, computed by

$$\begin{aligned} [\mu_k^+]_{\overline{\mathcal{W}}_k} &= \overline{M}_k M_k^T \lambda_k^* + [b^+ + Mv]_{\overline{\mathcal{W}}_k}, \\ [\mu_k^-]_{\overline{\mathcal{W}}_k} &= -\overline{M}_k M_k^T \lambda_k^* - [b^- + Mv]_{\overline{\mathcal{W}}_k}. \end{aligned} \tag{19}$$

Hence, instead of d, the algorithm uses $d^+ \triangleq b^+ + Mv$ and $d^- \triangleq -(b^- + Mv)$. Importantly, the relatively expensive matrix multiplication $\overline{M}_k M_k^T \lambda_k^*$ only has to be computed once in (19), while it has to be computed twice if the naive formulation with $\tilde{A}$ and $\tilde{b}$ is used. Furthermore, if a component of μ_k^+ is negative, the corresponding component of μ_k^- does not have to be computed since both the upper and lower bounds cannot be violated simultaneously (under the assumption that $b^- \leq b^+$, i.e. that the QP problem is not trivially infeasible). Optimality has been achieved if $\mu_k^+ \geq 0$ and $\mu_k^- \geq 0$. Otherwise, the most negative component of μ_k^+ or μ_k^- is added to $\mathcal{W}_k$.

When each component of λ is the multiplier for both an upper and lower bound simultaneously, it does not have to be nonnegative anymore. Instead, components of λ_k corresponding to active *upper* bounds have to be *nonnegative* while, conversely, components corresponding of active *lower* bounds have to be *nonpositive*. The condition $\lambda_k^* \geq 0$ is, hence, replaced by

$$[\lambda_k^*]_i \geq 0 \ \forall i \in \mathcal{W}_k^+, \text{ and } [\lambda_k^*]_i \leq 0 \ \forall i \in \mathcal{W}_k^-. \tag{20}$$

Moreover, $\mathcal{B}$ is redefined as $\mathcal{B} \triangleq \{i \in \mathcal{W}^+ : [\lambda_k^*]_i < 0\} \cap \{i \in \mathcal{W}^- : [\lambda_k^*]_i > 0\}$. For the singular case, $\mathcal{B}$ is redefined in a similar way but in terms of p_k rather than λ_k^*.

Finally, when solving the subproblems $M_k M_k^T \lambda_k^* = -d_k$, the components of d_k corresponding to active upper bounds should be replaced by elements of d^+ and, likewise, the components of d_k corresponding to active lower bounds should be replaced by elements of $-d^-$.

Remark 7 (Box-constrained QP): When the constraints are in the simple form $b^- \leq x \leq b^+$, common in, e.g., MPC, then $M = R^{-1}$, which is upper triangular. This additional structure can be exploited to reduce computations and the memory footprint further.

D. Equality constraints

Algorithm 1 can also easily be extended to handle equality constraints in (1). If the equality constraints are given as

$Gx = h$, we can, similar to (4), define

$$N \triangleq GR^{-1}, \quad w \triangleq R^{-1}f, \quad e \triangleq h + Nw. \tag{21}$$

The dual of the QP can then be stated as

$$\underset{\lambda \geq 0,\, \nu}{\text{minimize}} \ \frac{1}{2}\begin{bmatrix}\nu\\ \lambda\end{bmatrix}^T \begin{bmatrix}N\\ M\end{bmatrix}\begin{bmatrix}N\\ M\end{bmatrix}^T \begin{bmatrix}\nu\\ \lambda\end{bmatrix} + \begin{bmatrix}e\\ d\end{bmatrix}^T \begin{bmatrix}\nu\\ \lambda\end{bmatrix} \tag{22}$$

Essentially, equality constraints can be interpreted to always be active, i.e., be treated as being in $\mathcal{W}_k$ for all k. The corresponding modifications to Algorithm 1 are, hence, to replace (7), (8) and (9) with

$$\begin{bmatrix}N\\ M_k\end{bmatrix}\begin{bmatrix}N\\ M_k\end{bmatrix}^T \begin{bmatrix}\nu_k^*\\ [\lambda_k^*]_{\mathcal{W}_k}\end{bmatrix} = -\begin{bmatrix}e\\ d_k\end{bmatrix}, \tag{23a}$$

$$\begin{bmatrix}N\\ M_k\end{bmatrix}\begin{bmatrix}N\\ M_k\end{bmatrix}^T \begin{bmatrix}\tilde{\nu}_k\\ [p_k]_{\mathcal{W}_k}\end{bmatrix} = 0, \quad d^T p_k < 0, \tag{23b}$$

$$\mu_k = \overline{M}_k \begin{bmatrix}N\\ M_k\end{bmatrix}^T \begin{bmatrix}\nu_k^*\\ [\lambda_k^*]_{\mathcal{W}_k}\end{bmatrix} + \overline{d}_k, \tag{23c}$$

respectively.

Finally, factors L and D such that $NN^T = LDL^T$ are computed in the start of the algorithm.

E. Proximal-point iterations

The numerics of any QP solver can be improved by performing outer proximal-point iterations, which results in a sequence of better conditioned QPs being solved. In particular, proximal-point iterations are given by

$$\begin{aligned} x_{k+1} = \underset{x}{\text{argmin}} \quad & \frac{1}{2}x^T(H+\epsilon I)x + (f - \epsilon x_k)^T x \\ \text{s.t.} \quad & Ax \leq b, \end{aligned} \tag{24}$$

where $\epsilon > 0$ is a regularization parameter. It can be shown (cf. [12, Corollary 1]) that $\lim_{k\to\infty} x_k = x^*$ when (24) is iteratively applied. As was shown in [12], combining outer proximal-point iterations with an active-set algorithm can lead to an efficient and numerically stable solver. The numerical stability of the proposed method can, hence, be improved by amending it with outer proximal-point iterations, summarized in Algorithm 2. In Algorithm 2, R_ϵ is an upper Cholesky factor to $H + \epsilon I$, $M_\epsilon \triangleq AR_\epsilon$, $\eta > 0$ is a tolerance for determining if a fixed point has approximately been reached, and DAQP refers to Algorithm 1. Also note that DAQP is heavily warm-started in Step 4 when each perturbed QP in the form (24) is solved, reducing the computational burden significantly.

Algorithm 2 Proximal-point iterations.

Input: $\epsilon > 0, M_\epsilon, b, f, R_\epsilon^{-1}, \mathcal{W}, \lambda, x$
Output: $x^*, \lambda^*, \mathcal{A}^*$

1: **while** true **do**
2: $\quad v \leftarrow R_\epsilon^{-T}(f - \epsilon x); \quad d \leftarrow b + M_\epsilon v$
3: $\quad x_{\text{old}} \leftarrow x$
4: $\quad [x, \lambda, \mathcal{W}] \leftarrow \textbf{DAQP}(M_\epsilon, d, v, R_\epsilon^{-1}, \lambda, \mathcal{W})$
5: $\quad$ **if** $\|x - x_{\text{old}}\| < \eta$ **then**
6: $\quad\quad$ **return** $x, \lambda, \mathcal{W}$

F. Exact complexity certification

As was mentioned in Section II, Algorithm 1 can be interpreted as the active-set algorithm considered in [14]. Therein, a complexity certification method is proposed, which exactly determines the computational complexity for this active-set algorithm when QPs originating from a given multi-parametric quadratic program are to be solved. This complexity certification method can, hence, be used to determine the exact computational complexity of Algorithm 1. By doing so, worst-case bounds on the number of iterations and/or floating operations of Algorithm 1 can be determined *before* it is used in, e.g., an embedded MPC application.

V. Numerical Experiments

We will now empirically substantiate the claim in Section III-A that DAQP is a direct improvement, both in terms of numerical robustness and computational complexity, of QPNNLS. We then compare DAQP with other state-of-the-art QP solvers that are commonly used in MPC applications.

A. Comparison with QPNNLS

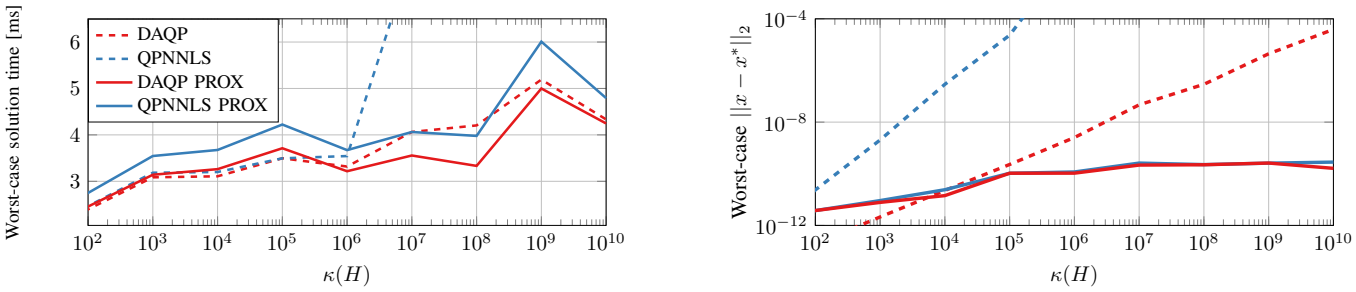
Numerical stability of the proposed method is compared with QPNNLS on a set of randomly generated small-scale QPs with varying condition numbers $\kappa(H)$. For each $\kappa(H)$, 100 QPs of size $n = 25, m = 100$ are generated and the worst-case distance from the optimal solution x^* as well as the worst-case solution time are measured.

We compare the original formulation of QPNNLS presented in [1] (QPNNLS), the extended version of QPNNLS presented in [12] in which numerical stability is improved by scaling and by performing outer proximal-point iterations (QPNNLS PROX), the proposed method given by Algorithm 1 (DAQP), and, finally, Algorithm 1 in conjunction with outer proximal-point iterations given by Algorithm 2 (DAQP PROX).

For all experiments, the primal feasibility tolerance is $\epsilon_p = 10^{-6}$ and, for the proximal-point iterations, $\epsilon = 10^{-4}$ and $\eta = \sqrt{2^{-52}} \approx 1.5 \cdot 10^{-8}$ (square root of machine epsilon for double precision). All of the methods are implemented in MATLAB using double precision and reference x^* are obtained by using CPLEX with settings emphasizing numerical precision. Each QP is solved five times and the median execution time for these five runs is the reported solution time.

Worst-case results are shown Figure 1, with DAQP showing better numerical properties and worst-case solution time compared with QPNNLS, as is to be expected from the discussion in Section III-A. Furthermore, Figure 1 also illustrates that the robust numerical properties of QPNNLS PROX reported in [12] extend to when DAQP is amended with proximal-point iterations. The worst-case solution time for DAQP PROX is, however, less than that of QPNNLS PROX, mainly because DAQP PROX can directly reuse the LDL^T factorization between the outer proximal-point iterations, while QPNNLS PROX has to perform low-rank updates before reusing the factors.

Remark 8: For some QPs with $\kappa(H) > 10^6$, QPNNLS reached the iteration limit and was, hence, unable to return a solution within the infeasibility tolerance. This explains the jump in solution time for $\kappa(H) > 10^6$ in Figure 1a.



(a) Worst-case solution time

(b) Worst-case distance to optimizer

Fig. 1. Comparison of DAQP, QPNNLS, with and without performing outer proximal-point iterations, on QPs with varying condition number $\kappa(H)$. The worst case solution time and distance to the optimizer reported for 100 randomly generated QPs with $n = 25$ and $m = 100$, for each $\kappa(H)$.

B. Model predictive control application

The proposed algorithm is also tested for MPC of an ATFI-16 aircraft [27], which is a tutorial problem in the MATLAB Model Predictive Control Toolbox and was also considered in [1], [12]. The system consists of two inputs, flaperon and elevator angles, which are upper and lower constrained, and two outputs, attack and pitch angles, with upper and lower limits imposed on the pitch angle. The system is simulated for 10 seconds (with a sampling time of 0.05 seconds) with an angle of attack reference of 0 and a pitch angle reference shifting between $\pm 10°$. For prediction horizon N, the resulting QPs have dimensions $n = 2N+1$, $m = 4N+2(N-1)$, where the extra optimization variable is due to the output constraints being softened. Because of the system having unstable poles, the Hessian has a fairly high condition number ($\kappa(H) > 10^{10}$). For the weights in the MPC formulation, the same settings as in the MATLAB MPC Toolbox tutorial were used.

Since the same MPC problem also was considered in [1], [12] and that the proposed method is a direct improvement of the methods therein (motivated in Section III-A and highlighted by the experiments in Section V-A), the favourable results for QPNNLS reported therein immediately also holds for the proposed method, and are in fact even strengthened. Nevertheless, we compare the worst-case solution time for the proposed method with some additional QP solvers which are used in the context of MPC. Herein, we focus on the case when the MPC problem is reformulated as a dense QP.

Remark 9 (Exploiting sparsity): We want to stress that our focus here is on solving dense QPs from the condensed MPC problem formulation. When solving large problems, other solvers that use sparse formulations [6], [11], [24], [28], [29] might be more efficient. If the ideas herein can be modified to exploit sparsity is a topic for future research.

We compare a C implementation[1] of Algorithm 1 (DAQP) and Algorithm 2 (DAQP PROX) with: (i) The parametric active-set method presented in [2] (qpOASES_e), coded in C; (ii) The operator splitting method presented in [11] (OSQP), coded in C; (iii) The interior-point method presented in [6] (HPIPM), coded in C; (iv) The dual active-set method presented in [3] (QPKWIK) which is based on [18], with generated C code from MATLAB.

[1]Available at https://github.com/darnstrom/daqp.

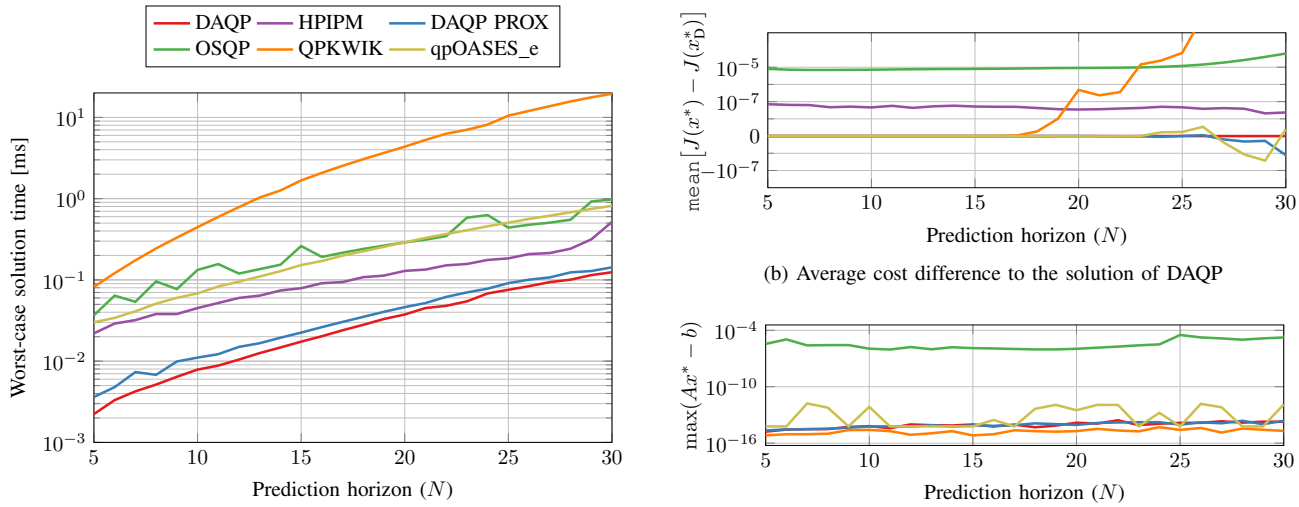
To get reliable solution times, each QP was solved 15 times and the median solution time was used as the reported solution time for the QP. When possible and relevant, the solvers were provided with precomputed matrix factorizations, e.g., the Cholesky factor of H was precomputed for DAQP, qpOASES_e and QPKWIK and only solution time was considered for OSQP. The memory footprint for DAQP for the largest QPs ($N = 30$), including code and problem data, was 48kB/70kB for single/double precision, respectively.

The worst-case solution times are reported in Figure 2a, where DAQP and DAQP PROX outperform the other solvers. Note, however, that HPIPM and OSQP scale better with N and will, as N grows larger, sooner or later outperform active-set methods. Still, DAQP seems superior on small to medium size MPC problems, which is a common scope of problems where embedded model predictive control is employed. Moreover, the average quality of the solutions computed by the solvers are compared in Figure 2b and Figure 2c by considering the average difference in the objective function and the worst-case constraint violations, respectively. A positive value in Figure 2b, implies that the solution is on average worse than that of DAQP. Figure 2b illustrates that the second-order solvers (except QPKWIK for larger horizons) give a better performance compared with OSQP. Moreover, qpOASES and DAQP PROX yield slightly better solutions than DAQP for larger horizons (where $\kappa(H) \approx 10^{14}$). Finally, Figure 2c illustrates that the constraint violation is low and similar for the active-set methods, while OSQP has, comparatively, large worst-case constraint violations. Also note that HPIPM has no constraint violation since it is an interior-point method.

VI. Conclusion

In this paper we have presented a dual active-set solver which is efficient, reliable, and simple to code, making it suitable for use in real-time model predictive control applications. Numerical experiments show that the proposed method can outperform state-of-the-art QP algorithms for QPs encountered in embedded MPC applications, both in terms of computational complexity and numerical robustness.

Future work includes combining the ideas herein with low-rank updates of the Riccati factorization [30], to investigate if these ideas can lead to an efficient solver for problems of larger size.



(a) Worst-case solution time

(b) Average cost difference to the solution of DAQP

(c) Worst-case constraint violation

Fig. 2. Worst-case solution time and solution quality when solving QPs encountered during the MPC of an ATFI-16 aircraft in simulation for varying prediction horizons N. The QPs have dimensions $n = 2N + 1$, $m = 4N + 2(N - 1)$. The solvers were exectued on an Intel 2.7 GHz i7-7500U CPU.

REFERENCES

[1] A. Bemporad, "A quadratic programming algorithm based on nonnegative least squares with applications to embedded model predictive control," *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 1111–1116, 2015.

[2] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[3] C. Schmid and L. T. Biegler, "Quadratic programming methods for reduced hessian SQP," *Computers & chemical engineering*, vol. 18, no. 9, pp. 817–832, 1994.

[4] N. Saraf and A. Bemporad, "A bounded-variable least-squares solver based on stable QR updates," *IEEE Transactions on Automatic Control*, vol. 65, no. 3, pp. 1242–1247, 2020.

[5] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on control systems technology*, vol. 18, no. 2, pp. 267–278, 2010.

[6] G. Frison and M. Diehl, "HPIPM: a high-performance quadratic programming framework for model predictive control," *arXiv preprint arXiv:2003.02547*, 2020.

[7] D. Axehill and A. Hansson, "A dual gradient projection quadratic programming algorithm tailored for model predictive control," in *2008 47th IEEE Conference on Decision and Control*, 12, pp. 3057–3064.

[8] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 18–33, 2014.

[9] S. Richter, C. N. Jones, and M. Morari, "Real-time input-constrained MPC using fast gradient methods," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE, 2009, pp. 7387–7393.

[10] P. Giselsson, "Improved fast dual gradient methods for embedded model predictive control," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 2303–2309, 2014.

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

[12] A. Bemporad, "A numerically stable solver for positive semidefinite quadratic programs based on nonnegative least squares," *IEEE Transactions on Automatic Control*, vol. 63, no. 2, pp. 525–531, 2017.

[13] D. Arnström, A. Bemporad, and D. Axehill, "Exact complexity certification of a nonnegative least-squares method for quadratic programming," *IEEE Control Systems Letters*, vol. 4, no. 4, pp. 1036–1041, 2020.

[14] D. Arnström and D. Axehill, "A unifying complexity certification framework for active-set methods for convex quadratic programming," *IEEE Transactions on Automatic Control*, 2022.

[15] W. S. Dorn, "Duality in quadratic programming," *Quarterly of Applied Mathematics*, vol. 18, no. 2, pp. 155–162, 1960.

[16] D. Arnström, "On complexity certification of active-set QP methods with applications to linear MPC," Licentiate Thesis, Linköping University Electronic Press, 2021.

[17] R. Fletcher, "A general quadratic programming algorithm," *IMA Journal of Applied Mathematics*, vol. 7, no. 1, pp. 76–91, 1971.

[18] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical Programming*, vol. 27, pp. 1–33, 9 1983.

[19] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, "Methods for modifying matrix factorizations," *Mathematics of computation*, vol. 28, no. 126, pp. 505–535, 1974.

[20] R. A. Bartlett and L. T. Biegler, "QPSchur: a dual, active-set, schur-complement method for large-scale and structured convex quadratic programming," *Optimization and Engineering*, vol. 7, no. 1, pp. 5–32, 2006.

[21] M. J. Best, "Equivalence of some quadratic programming algorithms," *Mathematical Programming*, vol. 30, no. 1, p. 71, 1984.

[22] G. B. Dantzig, *Linear programming and extensions*. Princeton university press, 1998, vol. 48.

[23] M. Fält and P. Giselsson, "QPDAS: Dual active set solver for mixed constraint quadratic programming," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 4891–4897.

[24] 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*. IEEE, 2006, pp. 5693–5698.

[25] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[26] R. Fletcher and S. Leyffer, "Numerical experience with lower bounds for MIQP branch-and-bound," *SIAM Journal on Optimization*, vol. 8, no. 2, pp. 604–616, 1998.

[27] A. Bemporad, A. Casavola, and E. Mosca, "Nonlinear control of constrained linear systems via predictive reference management," *IEEE transactions on Automatic Control*, vol. 42, no. 3, pp. 340–349, 1997.

[28] R. Quirynen and S. Di Cairano, "PRESAS: Block-structured preconditioning of iterative solvers within a primal active-set method for fast model predictive control," *Optimal Control Applications and Methods*, vol. 41, no. 6, pp. 2282–2307, 2020.

[29] J. V. Frasch, S. Sager, and M. Diehl, "A parallel quadratic programming method for dynamic optimization problems," *Mathematical Programming Computation*, vol. 7, no. 3, pp. 289–329, 2015.

[30] I. Nielsen and D. Axehill, "Low-rank modifications of Riccati factorizations for model predictive control," *IEEE Transactions on Automatic Control*, vol. 63, no. 3, pp. 872–879, 2017.