A Bounded-Variable Least-Squares Solver Based on Stable QR Updates

Nilay Saraf and Alberto Bemporad, *Fellow, IEEE*

***Abstract*— In this paper, a numerically robust solver for least-squares problems with bounded variables (BVLS) is presented for applications including, but not limited to, model predictive control (MPC). The proposed BVLS algorithm solves the problem efficiently by employing a recursive QR factorization method based on Gram-Schmidt orthogonalization. A reorthogonalization procedure that iteratively refines the QR factors provides numerical robustness for the described primal active-set method, which solves a system of linear equations in each of its iteration via recursive updates. The performance of the proposed BVLS solver, that is implemented in C without external software libraries, is compared in terms of computational efficiency against state-of-the-art quadratic programming solvers for small to medium-sized random BVLS problems and a typical example of embedded linear MPC application. The numerical tests demonstrate that the solver performs very well even when solving ill-conditioned problems in single precision floating-point arithmetic.**

***Index Terms*— Active-set methods, bounded-variable least squares, model predictive control, quadratic programming, recursive QR factorization.**

I. Introduction

Linear least-squares (LS) problems with simple bounds on variables arise in several applications of considerable importance in many areas of engineering and applied science [1]. Several algorithms exist that efficiently solve bounded-variable least-squares (BVLS) or quadratic programming (QP) problems including the ones encountered in model predictive control (MPC) and are based on first-order methods such as Nesterov's fast gradient-projection method [2], the accelerated dual gradient-projection method (GPAD) [3], the alternating direction method of multipliers (ADMM) [4], [5]. Because of the penalty functions often used in MPC for practical feasibility guarantee [6]–[9] through constraint relaxations, however, first-order methods require suitable preconditioners due to poor scaling of the problem. This makes active-set methods an attractive alternative for faster solution of small to medium-sized optimization problems arising in embedded MPC, due to their scarce sensitivity to problem scaling. For an overview of state-of-the-art methods in solving QP problems for MPC the reader is referred to the recent papers [10], [11].

In primal active-set methods such as BVLS [1] and nonnegative least-squares (NNLS) [12], at each iteration a change in the working active set corresponds to adding or removing a column from the matrix used to solve an unconstrained linear least-squares (LS) subproblem. By recursively updating the matrix factors computations are significantly reduced, although round-off error may accumulate for example, due to imperfect orthogonalization when using QR factorization. The main contribution of this paper includes methods which aim at overcoming such limitations without compromising numerical robustness through an effective application of stable QR update routines [13] for BVLS.

Submitted for review on November 22, 2018. This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement No 674875 (oCPS).

N. Saraf is with ODYS S.r.l., Via A. Passaglia 185, Lucca, 55100 LU Italy (e-mail: nilay.saraf@odys.it) and IMT School for Advanced Studies Lucca.

A. Bemporad is with IMT School for Advanced Studies Lucca, Piazza San Francesco 19, Lucca, 55100 LU Italy (e-mail: alberto.bemporad@imtlucca.it).

This paper is mainly motivated by the application of BVLS in fast linear and nonlinear MPC [6], [14], in order to solve optimization problems of the form

$$\min_{l \leq x \leq u} \frac{1}{2}\|f(x)\|_2^2, \text{ s.t. } h(x) = \mathbf{0}, \tag{1}$$

where x is the vector of decision variables with bounds l and u, and $h(x) = 0$ represents constraints arising from the prediction of system dynamics and general inequality constraints transformed to equalities using non-negative slack variables. The functions f and h are assumed to be first-order differentiable. Problem (1) can be efficiently solved using BVLS through a single iteration of the quadratic penalty method [15], [14, Algorithm 1] by transforming (1) to the following simple form:

$$\min_{l \leq x \leq u} \frac{1}{2}\|f(x)\|_2^2 + \frac{\rho}{2}\|h(x)\|_2^2,$$

where the quadratic penalty parameter $\rho > 0$ is large. The resulting LS subproblems may have a high condition number and require QR update routines that are robust against numerical errors due to potential poor conditioning. The implementation of stable QR update methods described here for BVLS is straightforwardly applicable to the special case of NNLS, an algorithm which can also be applied to solve general QPs for embedded MPC as shown in [16].

The paper is organized as follows. Section II briefly describes the baseline BVLS algorithm of [1] and its scope for improvement. The proposed algorithm which employs robust and recursive QR factorization is described in Section III. In Section IV, a new approach to recursively update the right hand side (RHS) of the triangular system of equations obtained when solving BVLS via the primal active-set method is described while summarizing the stable QR update procedures. Comparison results with competitive solution methods and the performance of the proposed solver in single precision are presented in Section V. The article is concluded in Section VI.

Notation: We denote the set of real vectors of dimension m as $\mathbb{R}^m$; a real matrix with m rows and n columns as $A \in \mathbb{R}^{m \times n}$, its transpose as $A^\top$, its inverse as A^{-1} and its j^{th} column as A_j. For a vector $a \in \mathbb{R}^m$, its p-norm is $\|a\|_p$, its j^{th} element is a_j, and $\|a\|_2^2 = a^\top a$. A vector or matrix of appropriate dimension(s) with all its elements zero is represented by $\mathbf{0}$. An identity matrix of appropriate dimensions is denoted by I. An empty matrix is denoted by [ ]. If $\mathcal{F}$ denotes a set of indices, then its cardinality is denoted by $|\mathcal{F}|$ and its j^{th} element as $\mathcal{F}_j$. $A_\mathcal{F}$ is a matrix formed from the columns of A corresponding to the indices in the index set $\mathcal{F}$ and $a_\mathcal{F}$ forms a vector (or set) with elements of the vector (or set) a as indexed in $\mathcal{F}$.

II. Baseline BVLS algorithm

This section recalls the algorithm based on the primal active-set method of [15, Algorithm 16.3] for solving the following bounded variable least-squares (BVLS) problem

$$J(x) = \min_{l \leq x \leq u} \quad \frac{1}{2}\|Ax - b\|_2^2, \tag{2}$$

where $A \in \mathbb{R}^{m\times n}, m \geq n$, $b \in \mathbb{R}^m$, and $l, u \in \mathbb{R}^n$ represent consistent lower and upper bounds respectively on the vector of decision variables $x \in \mathbb{R}^n$. To keep the algorithm description simple, we assume that x has finite bounds, although the algorithm can be easily extended to handle components that have no lower and/or upper bound.

Assumption 1: A is full column rank and $l \leq u$.

For example, Assumption 1 is satisfied when (2) is obtained from an MPC problem as shown in [6].

Stark and Parker's version of the BVLS algorithm [1] has two main loops, as it is typical of primal active-set methods. With reference to the steps of the algorithm described in [1], the inner loop (Steps 6-11) runs until primal feasibility (assessed in Step 7) is achieved in finite iterations and the outer loop (Steps 2-5) runs until the remaining convergence criteria assessed in Step 3 are satisfied. The Lagrange multipliers are simply derived from the gradient vector w and the index ($t^\star$) corresponding to the most negative one is introduced in the index set of free variables ($\mathcal{F}$) in order to initialize the inner loop. In [1], the unconstrained LS problems are solved by computing the QR factorization with column pivoting from scratch at each iteration in order to enforce numerical stability, at the price of computational burden.

III. Robust BVLS solver based on QR updates

In this section we propose a variant of [1] that aims at minimizing computations while maintaining numerical stability. The approach is summarized in Algorithm 1 and the reader is referred to [1] for an easy understanding of the main steps. The idea is to employ a different approach for QR factorization and recursive updates that cost only a fraction of the computations required to solve an LS problem from scratch. At initialization, all variables are placed in the free set (Step 1) unless an initial guess is provided. Next, a thin QR factorization (cf. Theorem 1 in Section IV) is computed in Step 2 (unless provided) by using the Gram-Schmidt orthogonalization procedure `gs`, before subsequent updates in the inner (Steps 14-35) and outer (Steps 3-13) loops through the stable update procedures `qrinsert` (cf. Lemma 1 in Section IV) and `qrdelete` (cf. Lemma 2 in Section IV). The unconstrained least-squares problem solved at each iteration is

$$\min_z \quad \frac{1}{2}\|A_{\mathcal{F}}z - p\|_2^2, \tag{3}$$

where p, the RHS of the linear system, can be computed by the relation in Step 3 of Algorithm 1. Using thin QR factorization, $A_{\mathcal{F}} = QR$, solving (3) reduces to solving the linear triangular system

$$Rz = Q^\top p = d \tag{4}$$

in Step 14 by the back-substitution procedure `solve_triu`. Unlike [1], or approaches in which the RHS of the triangular system (d) to solve the least-squares problem is explicitly computed through $\mathcal{O}(mn)$ operations [17], our approach recursively updates vector d (cf. Propositions 1 and 2 in Section IV) and avoids computing the matrix-vector product $Q^\top p$ in (4), at each iteration of the algorithm. In order to avoid introducing orthogonalization errors due to the product with $Q^\top$ in (4), vector d is initialized (Step 43) instead from the thin QR factors of the augmented system as described in [18, Ch. 19].

Remark 1: In the case of embedded MPC of linear time-invariant systems [6], the QR factors of A can be precomputed *offline* for cold-start. Moreover, during successive calls to the solver, as the matrix A does not change, the previously computed solution and its associated QR factors can be used for warmstarting, in order to significantly reduce the computational burden. Note that if the bounds vary, the components of the initial guess must be modified accordingly in order to have its active sets unchanged from the previous solution. As an additional precaution against error accumulation due to such reuse of the QR factors, they can be reset by not providing them to the solver at every N^{th} call, where N can be tuned depending on the problem size, desired accuracy, and computing precision.

The convergence proof of Algorithm 1 follows the ones of [15, Algorithm 16.3] and BVLS in [1]. However, due to rounding errors in finite-precision computations (especially when working in single-precision floating-point arithmetic), the inner loop (Steps 14-35) may not terminate when the computed α (cf. Step 22) results in no component of $\mathcal{F}$ entering an active set of bounds. Numerical error in the gradient (Step 4) may cause failure in satisfying dual feasibility, and cycling of the outer loop. Moreover, when the columns of A are nearly linearly dependent, the algorithm may cycle [1]. As a possible consequence:

1) a component $t^\star$ inserted in an active set of bounds is introduced in the free set in the immediate next step;
2) as a result, after the least-squares step, another component $\hat{t}$ gets inserted in an active set of bounds;
3) in the next iteration, $\hat{t}$ immediately gets inserted in the free set and causes $t^\star$ to be inserted back in the active set, causing a cycle.

We summarize below the measures included in the proposed algorithm in order to avoid the above described cycles, extending the ideas described in [1], [12]. Convergence of the inner loop is guaranteed by including a feasibility tolerance γ (typically between machine precision and 10^{-6}) and by moving at least the index κ (Step 23) in the respective active set at each iteration, such that κ accounts for the value of α as done in [12]. As suggested in [1], unnecessary failure in satisfying dual feasiblity conditions is detected by Step 15 which signals if the component most recently introduced in the free set would immediately enter an active set of bounds. Steps 16-18 set the Lagrange multiplier of the corresponding component to zero and the algorithm steps back to termination check (Step 5). Cycling between a pair of free components is detected by storing (Step 11) and comparing the previous three values of the index $t^\star$ introduced in the free set as shown in Step 7 of Algorithm 1. The Lagrange multipliers corresponding to the cycling components $t^\star$ and $\hat{t}$ are set to zero in Step 9 and the algorithm then steps back to termination check. If the termination check yet fails and no other change in $\mathcal{F}$ results in termination, we infer that the algorithm will not converge any further. So, the outer loop terminates having the test in Step 7 satisfied for a second instance (Step 9) due to the cycling of a pair of components. Hence, the algorithm itself terminates. In any case, in all the practical applications in which the solver is used on line, such as in embedded MPC, a bound on the maximum number of iterations is enforced to guarantee termination, where this bound depends on problem size and real-time requirements.

IV. Recursive thin QR factorization

In this section, we recall the stable QR update procedures of [13] which are adopted in Algorithm 1, in the context of active-set changes. Based on that, we derive the recursive relations which update the RHS of (4).

Theorem 1 (thin QR factorization): Let $A \in \mathbb{R}^{m\times n}$ be a full rank matrix with $m > n$. Let $A_{\mathcal{F}} \in \mathbb{R}^{m\times|\mathcal{F}|}$ be formed from a subset of the columns of A, indexed by the set of indices $\mathcal{F} \subseteq \{1, 2, \cdots, n\}$, $|\mathcal{F}| \leq n$. Then there exists a matrix $Q \in \mathbb{R}^{m\times|\mathcal{F}|}$ with orthonormal columns, and a full rank upper-triangular matrix $R \in \mathbb{R}^{|\mathcal{F}|\times|\mathcal{F}|}$ such that $A_{\mathcal{F}} = QR$.

Proof: We prove the theorem by induction on the cardinality $|\mathcal{F}|$ of $\mathcal{F}$. We first prove the theorem for the trivial case of $|\mathcal{F}| = 1$,

Algorithm 1 Robust BVLS based on recursive QR updates

Inputs: Matrices A, b, l, u; feasibility tolerance $\gamma > 0$.

```
1: $x \leftarrow (l+u)/2$; $\mathcal{F} \leftarrow \{1, \cdots, n\}$; $\mathcal{U}, \mathcal{L} \leftarrow \emptyset$; $\iota \leftarrow 0$; $t' \leftarrow -1$;
   $t'' \leftarrow -2$; $t''' \leftarrow -3$; $t^\circ \leftarrow 0$; $p \leftarrow b - A_{(\mathcal{L}\cup\mathcal{U})}x_{(\mathcal{L}\cup\mathcal{U})}$;
2: $\{Q, R, d\} \leftarrow \texttt{gs}(A, p, \mathcal{F})$; **Go to** Step 14;
3: $p \leftarrow b - A_{(\mathcal{L}\cup\mathcal{U})}x_{(\mathcal{L}\cup\mathcal{U})}$;
4: $w \leftarrow p - A_{\mathcal{F}}x_{\mathcal{F}}$; $w_{(\mathcal{L}\cup\mathcal{U})} \leftarrow A^\top_{(\mathcal{L}\cup\mathcal{U})}w$;
5: **If** $(\mathcal{L}\cup\mathcal{U} = \emptyset)$ **or** $(\max(w_{\mathcal{L}}) \leq \gamma$ **and** $\min(w_{\mathcal{U}}) \geq -\gamma)$, **go to**
   Step 36;
6: $t^\star \leftarrow \min(\arg\max_{t\in\mathcal{L}\cup\mathcal{U}} s_t w_t)$, where $s_t = 1$ if $t \in \mathcal{L}$ and
   $s_t = -1$ if $t \in \mathcal{U}$;
7: **if** $t''' = t'$ **and** $t'' = t^\star$, **then**
8:     **if** $t^\circ = 1$, **go to** Step 36 **end if**;
9:     $w_{t^\star}$, $w_{t'} \leftarrow 0$; $t^\circ \leftarrow 1$; **go to** Step 5
10: **end if**
11: $t''' \leftarrow t''$; $t'' \leftarrow t'$; $t' \leftarrow t^\star$; **if** $t^\star \in \mathcal{L}$, $\iota \leftarrow -1$ **else** $\iota \leftarrow 1$;
12: $\mathcal{F} \leftarrow \mathcal{F} \cup \{t^\star\}$; **if** $t^\star \in \mathcal{L}$, $\mathcal{L} \leftarrow \mathcal{L} \setminus \{t^\star\}$ **else** $\mathcal{U} \leftarrow \mathcal{U} \setminus \{t^\star\}$;
13: $\{Q, R, d\} \leftarrow \texttt{qrinsert}(Q, R, d, p, A_{t^\star}, x_{t^\star}, \mathcal{F}, t^\star)$;
14: **If** $\mathcal{F} \neq \emptyset$, **then** $z \leftarrow$solve_triu$(R, d)$;
15: **if** $(\iota = -1$ **and** $z_j < l_{t^\star})$ **or** $(\iota = 1$ **and** $z_j > u_{t^\star})$, where
    $\mathcal{F}_j = t^\star$, **then**
16:     $\{Q, R, d\} \leftarrow \texttt{qrdelete}(Q, R, d, x_{t^\star}, \mathcal{F}, t^\star)$;
17:     **if** $\iota = -1$, $\mathcal{L} \leftarrow \mathcal{L} \cup \{t^\star\}$; **else** $\mathcal{U} \leftarrow \mathcal{U} \cup \{t^\star\}$; **end if**
18:     $\mathcal{F} \leftarrow \mathcal{F} \setminus \{t^\star\}$; $w_{t^\star} \leftarrow 0$; **go to** Step 5;
19: **end if**
20: **If** $(l_{\mathcal{F}_j} - \gamma) \leq z_j \leq (u_{\mathcal{F}_j} + \gamma)\ \forall j \in \{1, \cdots, |\mathcal{F}|\}$ **or** $\mathcal{F} = \emptyset$,
    **then** $x_{\mathcal{F}} \leftarrow z$ **and go to** Step 3;
21: $\iota \leftarrow 0$; $\mathcal{I} \leftarrow \{\mathcal{F}_j | l_{\mathcal{F}_j} > z_j$ or $z_j > u_{\mathcal{F}_j}, j \in \{1, ..., |\mathcal{F}|\}\}$;
22: $\alpha \leftarrow \min\left\{1, \min_{j\in\{1,\cdots,|\mathcal{F}|\}, \mathcal{F}_j\in\mathcal{I}}\left(\left|\frac{l_{\mathcal{F}_j}-x_{\mathcal{F}_j}}{z_j-x_{\mathcal{F}_j}}\right|, \left|\frac{u_{\mathcal{F}_j}-x_{\mathcal{F}_j}}{z_j-x_{\mathcal{F}_j}}\right|\right)\right\}$;
23: $\kappa \leftarrow \arg_j \alpha$; $k \leftarrow |\mathcal{F}|$;
24: **for** $\forall j \in \{1, \cdots, k\}$ **do** $x_{\mathcal{F}_j} \leftarrow x_{\mathcal{F}_j} + \alpha(z_j - x_{\mathcal{F}_j})$;
25:     **if** $x_{\mathcal{F}_j} \leq l_{\mathcal{F}_j} + \gamma$ **or** $(j = \kappa$ **and** $z_\kappa < l_{\mathcal{F}_\kappa})$ **then**
26:         $\{Q, R, d\} \leftarrow \texttt{qrdelete}(Q, R, d, x_{\mathcal{F}_j}, \mathcal{F}, \mathcal{F}_j)$;
27:         $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{F}_j$; $\mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{F}_j$;
28:     **else**
29:         **if** $x_{\mathcal{F}_j} \geq u_{\mathcal{F}_j} - \gamma$ **or** $(j = \kappa$ **and** $z_\kappa > u_{\mathcal{F}_\kappa})$ **then**
30:             $\{Q, R, d\} \leftarrow \texttt{qrdelete}(Q, R, d, x_{\mathcal{F}_j}, \mathcal{F}, \mathcal{F}_j)$;
31:             $\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{F}_j$; $\mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{F}_j$;
32:         **end if**
33:     **end if**
34: **end for**;
35: **Go to** Step 14;
36: **end.**

37: **procedure** $\texttt{gs}(A, p, \mathcal{F})$
38: $Q \leftarrow [\ ]$; $R \leftarrow [\ ]$;
39: **for** $\forall j \in \mathcal{F}$ **do**
40: $\{q, r, \rho\} \leftarrow \texttt{orthogonalize}(A_j, Q, R)$; (Algorithm 2)
41: $Q \leftarrow \begin{bmatrix} Q & q \end{bmatrix}$; $R \leftarrow \begin{bmatrix} R & r \\ \mathbf{0} & \rho \end{bmatrix}$;
42: **end for**
43: $\{\sim, d, \sim\} \leftarrow \texttt{orthogonalize}(p, Q, R)$;
44: **end procedure**
```

Outputs: Primal solution x of (2), Active set of lower and upper bounds $\mathcal{L}$ and $\mathcal{U}$ respectively, set $\mathcal{F}$ of free variables.

where we can write $A_{\mathcal{F}_1} = Q^{(1)}R^{(1)}$, $Q^{(1)} = A_{\mathcal{F}_1}/\|A_{\mathcal{F}_1}\|_2$ and $R^{(1)} = \|A_{\mathcal{F}_1}\|_2$. Assume that the theorem holds $\forall\mathcal{F}$ of cardinality $|\mathcal{F}| = k-1$. For a generic set $\mathcal{F}$ of k indices, if $\mathcal{F}'$ denotes its first $k-1$ indices, we can write $A_{\mathcal{F}} = \begin{bmatrix} A_{\mathcal{F}'} & A_{\mathcal{F}_k} \end{bmatrix}$. Considering that $|\mathcal{F}'| = k-1$, using the induction hypothesis we can state that $\exists\ Q^{(k-1)}$ orthonormal and $R^{(k-1)}$ full rank upper triangular, such that $A_{\mathcal{F}'} = Q^{(k-1)}R^{(k-1)}$. Since A is full rank (Assumption 1), via Gram-Schmidt orthogonalization [13] we can find an orthonormal vector $q \in \mathbb{R}^m$ $(q^\top Q^{(k-1)} = \mathbf{0})$, a vector $r \in \mathbb{R}^{(k-1)}$, and a scalar $\rho \neq 0$ such that $A_{\mathcal{F}_k} = Q^{(k-1)}r + \rho q$. Thus, we can rewrite $A_{\mathcal{F}} = \begin{bmatrix} A_{\mathcal{F}'} & A_{\mathcal{F}_k} \end{bmatrix} = \begin{bmatrix} Q^{(k-1)}R^{(k-1)} & Q^{(k-1)}r + q\rho \end{bmatrix} = Q^{(k)}R^{(k)}$, where $Q^{(k)} = \begin{bmatrix} Q^{(k)} & q \end{bmatrix}$ and $R^{(k)} = \begin{bmatrix} R^{(k-1)} & r \\ \mathbf{0} & \rho \end{bmatrix}$. Matrix $Q^{(k)}$ has orthonormal columns and $R^{(k)}$ is full rank upper triangular by inspection, which proves the theorem for $|\mathcal{F}| = k$. ■

Remark 2: In [13], in order to enforce precise orthogonality the Gram-Schmidt procedure on any vector is repeated in situations when numerical cancellation is detected and is termed as reorthogonalization. If the ratio $\|A_{\mathcal{F}_k} - Q^{(k-1)}r\|_2/\|A_{\mathcal{F}_k}\|_2$ is small or if ρ is extremely small, loss of orthogonality is likely and the same ratio is used to determine whether reorthogonalization must be performed [13]. This argument yields Algorithm 2 that iteratively "refines" the computed vectors by reducing orthogonality loss. The parameter η in Algorithm 2 sets an upper bound on the loss of orthogonality as shown by the detailed error analysis in [13]. Increasing η increases the chance of reorthogonalization and tightens the tolerance on orthogonality closer to the machine precision. We use $\eta = 1/\sqrt{2}$ and $k_{\max} = 4$ in Algorithm 2 based on the analysis in [13]. For double precision computations, reorthogonalizations may not be performed at all for small to medium sized problems unless they are ill-conditioned.

Based on Theorem 1, Lemma 1 and 2 respectively delineate the recursive relations that update the thin QR factorization when an index is either inserted or deleted arbitrarily from the set $\mathcal{F}$.

Lemma 1: Given $A_{\mathcal{F}} = QR$, if $A_{\bar{\mathcal{F}}} = \bar{Q}\bar{R}$ denotes the thin QR factorization of $A_{\bar{\mathcal{F}}}$ for $\bar{\mathcal{F}} := \mathcal{F} \cup \{t^\star\}$ and $|\bar{\mathcal{F}}| = k+1$ with $k < n$, then there exists an orthogonal matrix G, two vectors q, r, and a scalar ρ, such that $\bar{Q} = \begin{bmatrix} Q & q \end{bmatrix} G^\top$ and $\bar{R} = G\begin{bmatrix} R_{\mathcal{F}'} & r & R_{\mathcal{F}''} \\ \mathbf{0} & \rho & \mathbf{0} \end{bmatrix}$, where $\mathcal{F}' := \{j | \mathcal{F}_j < t^\star, j \in \{1, \cdots, k\}\}$ and $\mathcal{F}'' := \{j | \mathcal{F}_j > t^\star, j \in \{1, \cdots, k\}\}$.

Proof: With $A_{\mathcal{F}'} \in \mathbb{R}^{m\times|\mathcal{F}'|}$, $A_{\mathcal{F}''} \in \mathbb{R}^{m\times(k-|\mathcal{F}'|)}$, and $A_{t^\star}$ as the inserted column, considering $R = \begin{bmatrix} R_{\mathcal{F}'} & R_{\mathcal{F}''} \end{bmatrix}$, we have the following relations

$$A_{\bar{\mathcal{F}}} = \begin{bmatrix} A_{\mathcal{F}_{\mathcal{F}'}} & A_{t^\star} & A_{\mathcal{F}_{\mathcal{F}''}} \end{bmatrix} = \begin{bmatrix} Q & A_{t^\star} \end{bmatrix} \begin{bmatrix} R_{\mathcal{F}'} & \mathbf{0} & R_{\mathcal{F}''} \\ \mathbf{0} & 1 & \mathbf{0} \end{bmatrix}.$$

By orthogonalizing $A_{t^\star}$ via Algorithm 2, i.e., by the Gram-Schmidt procedure [13], we obtain

$$A_{t^\star} = Qr + \rho q, \text{ and hence} \tag{5}$$

Algorithm 2 Orthogonalization procedure [13]

```
1: $r^0 \leftarrow \mathbf{0}$, $v^0 \leftarrow v$, $0 << \eta < 1$;
2: **for** $k = 1, 2, \cdots$, until $\|v^k\|_2 > \eta\|v^{k-1}\|_2$ or $k = k_{\max}$ **do**:
3:     $s^k \leftarrow Q^\top v^{k-1}$;
4:     $r^k \leftarrow r^{k-1} + s^k$;
5:     $v^k \leftarrow v^{k-1} - Qs^k$;
6: **end for**
7: $r \leftarrow r^k$; $\rho \leftarrow \|v^k\|_2$; $q \leftarrow v^k/\rho$;
8: **end.**
```

$$A_{\bar{\mathcal{F}}} = \underbrace{\begin{bmatrix} Q & q \end{bmatrix}}_{\tilde{Q}} \underbrace{\begin{bmatrix} R_{\mathcal{F}'} & r & R_{\mathcal{F}''} \\ \mathbf{0} & \rho & \mathbf{0} \end{bmatrix}}_{\tilde{R}}, \tag{6}$$

where q, r and ρ are computed such that $\tilde{Q}$ has orthonormal columns and $\tilde{R}$ has subdiagonal elements in its $(|\mathcal{F}'|+1)^{\text{th}}$ column. These subdiagonal elements can be zeroed-out by successive application of Givens matrices [17], which converts $\tilde{R}$ to the upper triangular matrix

$$\bar{R} = G\tilde{R}, \tag{7}$$

where $G \in \mathbb{R}^{|\bar{\mathcal{F}}|\times|\bar{\mathcal{F}}|}$ denotes the product of the Givens matrices, which are orthogonal by definition. Hence, $G^\top G = I$, and from (6)-(7), $A_{\bar{\mathcal{F}}} = \tilde{Q}G^\top G\tilde{R} = \tilde{Q}G^\top \bar{R}$, which implies that $\bar{Q} = \tilde{Q}G^\top$. ■

Lemma 2: Given $A_{\mathcal{F}} = QR$, if $A_{\hat{\mathcal{F}}} = \hat{Q}\hat{R}$ denotes the thin QR factorization of $A_{\hat{\mathcal{F}}}$ for $\hat{\mathcal{F}} := \mathcal{F}\setminus\{t^\star\}$, then there exists a matrix H such that $\hat{Q} = QH^\top$ and $\hat{R} = HR_{\tilde{\mathcal{F}}}$, where $\tilde{\mathcal{F}} := \{j|\mathcal{F}_j \neq t^\star, j \in \{1,\cdots,|\mathcal{F}|\}\}$.

Proof: Given $A_{\mathcal{F}} = QR$, with $\mathcal{F}'$ and $\mathcal{F}''$ as defined in Lemma 1, we have

$$A_{\mathcal{F}} = \begin{bmatrix} A_{\mathcal{F}_{\mathcal{F}'}} & A_{t^\star} & A_{\mathcal{F}_{\mathcal{F}''}} \end{bmatrix} = Q\begin{bmatrix} R_{\mathcal{F}'} & \tilde{r} & R_{\mathcal{F}''} \end{bmatrix}, \tag{8}$$

and $A_{\hat{\mathcal{F}}} = \begin{bmatrix} A_{\mathcal{F}_{\mathcal{F}'}} & A_{\mathcal{F}_{\mathcal{F}''}} \end{bmatrix} = Q\begin{bmatrix} R_{\mathcal{F}'} & R_{\mathcal{F}''} \end{bmatrix} = QR_{\tilde{\mathcal{F}}}$, where $R_{\tilde{\mathcal{F}}}$ is upper Hessenberg [17]. Zeroing the off-diagonal elements of $R_{\tilde{\mathcal{F}}}$ using Givens rotations, we obtain $G'R_{\tilde{\mathcal{F}}} = \begin{bmatrix} \hat{R} \\ \mathbf{0} \end{bmatrix}$ where $G' \in \mathbb{R}^{|\mathcal{F}|\times|\mathcal{F}|}$ denotes the product of the Givens matrices. Hence, $\hat{R} = EG'R_{\tilde{\mathcal{F}}} = HR_{\tilde{\mathcal{F}}}$, where $E = \begin{bmatrix} I & \mathbf{0} \end{bmatrix}^\top$ and $H = EG'$. Since G' is orthogonal, $G'^\top E^\top EG' = I$, which implies that $H^\top H = I$. As $A_{\mathcal{F}} = \hat{Q}\hat{R} = QR_{\tilde{\mathcal{F}}} = QH^\top HR_{\tilde{\mathcal{F}}}$, we obtain $\hat{Q} = QH^\top$. ■

Remark 3: Updating the thin QR factorization when inserting or deleting a column as above requires $4lm(k+1)+6|\mathcal{F}''|(m+|\mathcal{F}''|/2)$ and $6|\mathcal{F}''|(m+|\mathcal{F}''|/2)$ *flops* (floating-point operations excluding indexing and looping overhead) respectively, where l is the number of orthogonalizations performed. The update routines are numerically stable as precise orthogonality is retained during the recursions.

Proposition 1: Consider $d = Q^\top p$ as defined in (4) where $A_{\mathcal{F}} = QR$, and the updated factorization $A_{\bar{\mathcal{F}}} = \bar{Q}\bar{R}$ for $\bar{\mathcal{F}} = \mathcal{F}\cup t^\star$ as in Lemma 1. The update $\bar{d}$ of d can be obtained recursively as $\bar{d} = G\tilde{d}$, where $\tilde{d} = \begin{bmatrix} d + rx_{t^\star} \\ q^\top p + \rho x_{t^\star} \end{bmatrix}$ and $x_{t^\star}$ denotes the $t^\star$th element of solution vector x at the current iteration of Algorithm 1 when solving (2).

Proof: From (4), we have

$$d = Q^\top p, \text{ and} \tag{9}$$

$$\bar{d} = \bar{Q}^\top \bar{p}, \tag{10}$$

where $\bar{p}$ denotes the vector p of (4) after an index $t^\star$ is inserted in the free set $\mathcal{F}$. Then by its definition,

$$\bar{p} = p + A_{t^\star}x_{t^\star}. \tag{11}$$

Using Lemma 1 and substituting (11) in (10) gives

$$\bar{d} = G\begin{bmatrix} Q & q \end{bmatrix}^\top (p + A_{t^\star}x_{t^\star}). \tag{12}$$

On substituting (5) in (12), we get

$$\begin{aligned} \bar{d} &= G\begin{bmatrix} Q & q \end{bmatrix}^\top (p + Qrx_{t^\star} + \rho qx_{t^\star}) \\ &= G\begin{bmatrix} Q^\top p + Q^\top Qrx_{t^\star} + \rho Q^\top qx_{t^\star} \\ q^\top p + q^\top Qrx_{t^\star} + \rho q^\top qx_{t^\star} \end{bmatrix}. \end{aligned} \tag{13}$$

Since Q has orthonormal columns and q is orthogonal to the span of Q, we have $Q^\top Q = I$, $q^\top q = 1$, $Q^\top q = \mathbf{0}$ and $q^\top Q = \mathbf{0}$. Hence, by substituting these relations and (9), (13) simplifies as

$$\bar{d} = G\begin{bmatrix} d + rx_{t^\star} \\ q^\top p + \rho x_{t^\star} \end{bmatrix}. \tag{14}$$

■

Proposition 1 shows through (14) that computing $\tilde{d}$ from d only needs $2(|\mathcal{F}|+m+1)$ *flops* and $\bar{d}$ is obtained by using $6|\mathcal{F}''|$ *flops* for applying the Givens rotations [17] on $\tilde{d}$. Updating d to $\bar{d}$ without the recursive relation (14) would cost $2m(|\mathcal{F}|+2)$ *flops* instead, due to the computations in (11) and (10), at each iteration of the outer loop of Algorithm 1. Proposition 2 establishes the recursive relation to update d for the case in which an index is removed from the free set $\mathcal{F}$.

Proposition 2: Consider $d = Q^\top p$ as defined in (4) where $A_{\mathcal{F}} = QR$, and the updated factorization $A_{\hat{\mathcal{F}}} = \hat{Q}\hat{R}$ for $\hat{\mathcal{F}} = \mathcal{F}\setminus t^\star$ as in Lemma 2, then the update $\hat{d}$ of d can be obtained recursively as $\hat{d} = H(d - \tilde{r}x_{t^\star})$.

Proof: Let $\hat{p}$ denote the vector p of (4) after an index $t^\star$ is removed from $\mathcal{F}$, then

$$\hat{p} = p - A_{t^\star}x_{t^\star}, \tag{15}$$

where from (8) the deleted column

$$A_{t^\star} = Q\tilde{r}. \tag{16}$$

From (4), (15), (16), and Lemma 2,

$$\hat{d} = \hat{Q}^\top \hat{p} = HQ^\top (p - Q\tilde{r}x_{t^\star}).$$

On substituting $Q^\top Q = I$ and (9) in the above equation, we get

$$\hat{d} = H(d - \tilde{r}x_{t^\star}). \tag{17}$$

■

Updating d to $\hat{d}$ recursively via (17) needs only $2|\mathcal{F}| + 6(|\mathcal{F}''|-1)$ *flops* instead of $2m|\mathcal{F}|$ *flops* for computing $\hat{Q}^\top \hat{p}$ and (15), in each iteration of the inner loop of Algorithm 1.

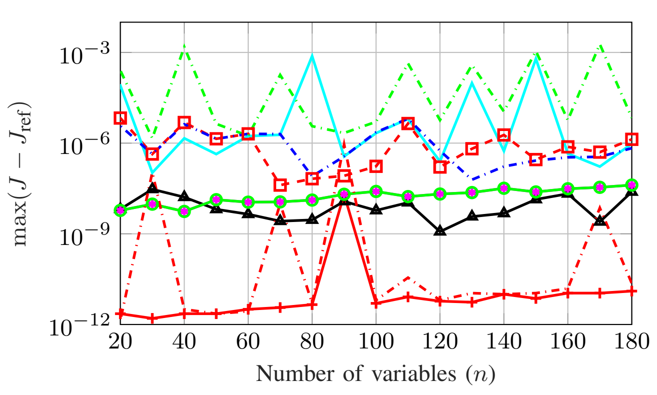
Remark 4: Even though the vector d is initialized with machine precision accuracy, as a precaution to avoid the error accumulated over potentially several recursive updates in large sized problems it is recommended to reinitialize d via Step 43 in Algorithm 1 after every N iterations, with N chosen according to the available computing precision.

V. Numerical results

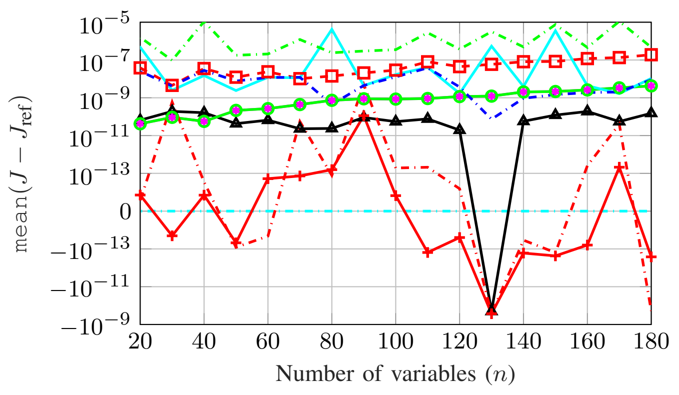
A. Random BVLS problems

This section describes the results obtained in MATLAB[1] by testing the proposed solver (Algorithm 1) and various others on random BVLS problems. In order to test for numerical robustness and performance while dealing with ill-conditioned or nearly rank-deficient problems, the condition number of the A matrix is set to 10^8 ($\equiv 10^{16}$ for the Hessian of the equivalent QP $\min_{l\leq x\leq u} \frac{1}{2}x^\top A^\top Ax - b^\top Ax$). The following solvers are considered for the numerical tests: 1) BVLS_SP - Stark and Parker's BVLS algorithm [1] implemented in embedded MATLAB; 2) RBVLS - Algorithm 1; 3) BVLS2 - a variant of Algorithm 1 in which the number of orthogonalizations in QR update procedures is restricted to one for faster execution;

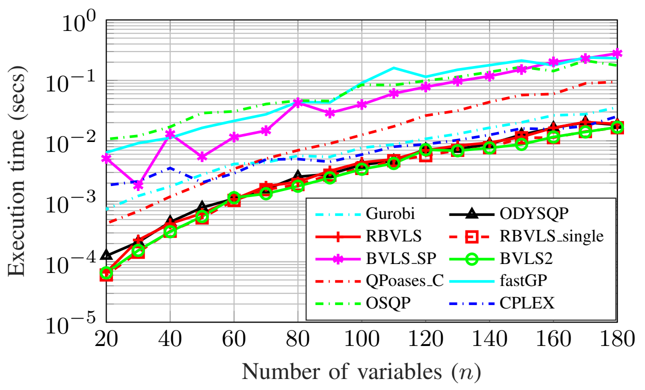
[1]All the numerical tests (details available through the link http://dysco.imtlucca.it/nilay/bvlsgen.pdf) have been run on a Macbook Pro 2.6 GHz Intel Core i5 with 8GB RAM. All matrix operations have been computed through plain C code interfaced to MATLAB. The execution time of each solver is obtained from its internally measured time in order to avoid counting the time for interfacing it with MATLAB.



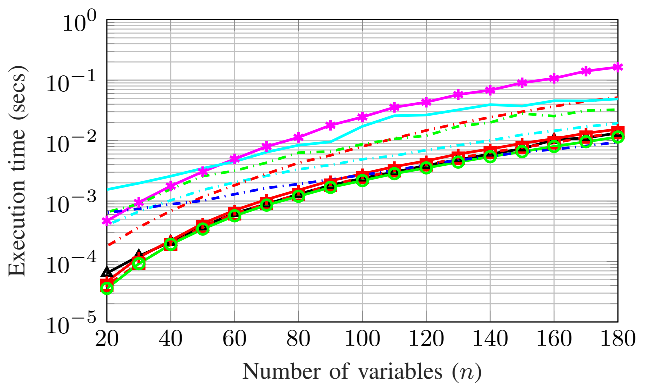
(a) Worst-case difference from the benchmark cost.



(b) Average difference from the benchmark cost.



(c) Worst-case computational time.



(d) Average computational time.

Fig. 1: Solver performance comparison for feasible BVLS test problems[1] with condition number of matrix $A = 10^8$, 180 random instances for each $A \in \mathbb{R}^{1.5n \times n}$, and all possible non-zero cardinality values of the optimal active set. In Figures 1a-1b, the optimal cost J of (2) obtained using each solver is compared with the benchmark value $J_{\rm ref}$ obtained using Gurobi.

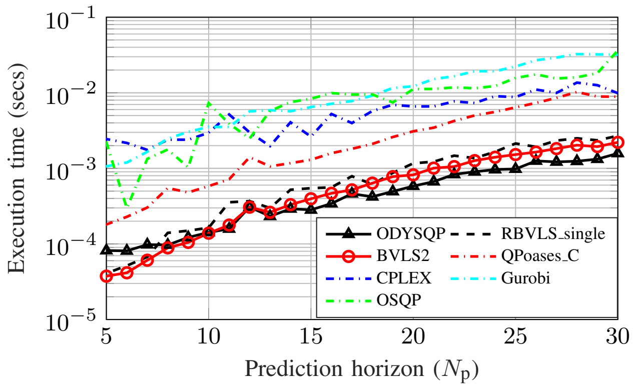
4) QPoases_C - box-constrained variant of the open source QP solver QPOASES version 3.2.0 [19] with C backend and main setting *reliable*, where we also enable settings that avoid additional computational overhead while solving nearly rank-deficient problems; 5) OSQP - solver version 0.3.0 of the QP method [5] based on ADMM using sparse matrices and 5000 maximum iterations to limit its maximum execution time; 6) Gurobi - the dual simplex algorithm of Gurobi 7.5.2 [20] was chosen for the tests as it performed best amongst its other available algorithms; 7) fastGP - the fast gradient projection algorithm [2] with restart in every 50 off 3000 maximum iterations and termination criterion based on [21, Equation 6.18]; 8) ODYSQP - ODYS QP solver[2] [22]; 9) CPLEX - primal simplex algorithm (method `cplexlsqlin`) of CPLEX 12.6.3 [23]. For all solvers, the feasibility or optimality tolerance was set to 10^{-9} with all computations in double precision (machine precision $\epsilon \approx 10^{-16}$), except for the single precision ($\epsilon \approx 10^{-7}$) version RBVLS_single of Algorithm 1, where the same tolerance was set to 10^{-6}. The tolerances for all solvers are relaxed to 10^{-6} for the MPC example in Section V-B. For first-order methods OSQP and fastGP, the tolerances are internally relaxed based on problem-specific termination criteria [5], [21]. In Figures 1a-1b, the cost function values compare the quality of the solution instead of the solution vector obtained from different solvers because in the presence of tolerances and poor-conditioning, the solution values obtained may differ while numerically yielding the same value of the cost function. Figure 1 demonstrates that the proposed BVLS solver is suitable for applications like embedded MPC where numerical robustness and computational efficiency are necessary requirements.

[2]ODYS QP solver version "General purpose" 2017 has been used for all tests reported in this paper.

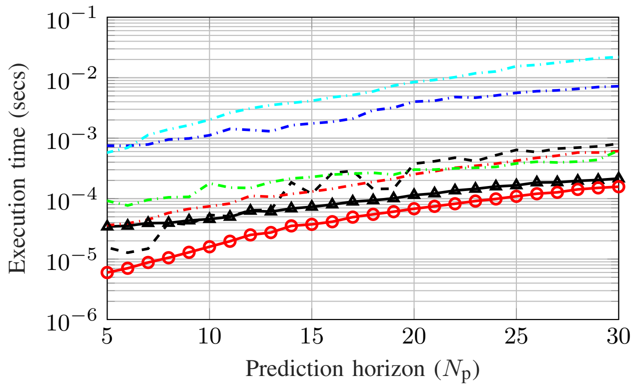
B. Application: embedded linear model predictive control

We now consider BVLS problems that arise when solving the MPC problem for control of an AFTI-F16 aircraft based on a linear time-invariant (LTI) model [6]. These problems have a high condition number, a property typically encountered in MPC problems involving the use of penalty functions and open-loop unstable prediction models, which hinders the convergence of first-order methods such as fastGP. In order to count only the time required for *online* computations, for all the QP solvers, the time required for computing the Hessian matrix and its factors is not accounted, as it can be done *offline* once and stored. For the same reason, the QPoases_C and ODYSQP solvers are provided with pre-computed dense matrix factorizations of the Hessian, and in case of OSQP only *solve time* was measured. Since the considered MPC optimization problem is numerically sparse, the solvers OSQP, Gurobi, CPLEX and QPoases_C are provided with sparse matrices for a faster performance. The proposed algorithm is warm-started with the previous solution (cf. Remark 1 in Section III) and all other solvers are warm-started from the shifted previous solution from the second instance onward, except for ODYSQP which is always cold-started. Figure 2a shows that the proposed algorithm is competitive in computational efficiency as compared to the benchmark solvers, whereas Figure 2b shows that it considerably exploits warmstarts. Although in embedded MPC applications the worst-case CPU time is the most relevant measure, as it is used to guarantee meeting hard real-time constraints, the average time may still be of interest when the same CPU is shared with other tasks.

Solvers exploiting the block-sparse structure of MPC problems



(a) Worst-case computational time.



(b) Average computational time.

Fig. 2: Solver performance comparison for BVLS formulation based tracking-MPC simulation of AFTI-F16 aircraft. The number of decision variables and box constraints each $= 4N_p$ resulting in matrix $A \in \mathbb{R}^{6N_p \times 4N_p}$. For each N_p, the simulation is run for 100 time instances.

have their computational complexity scale linearly with N_p [24], [25], and over a certain value of N_p they may outperform the *dense* solvers, for which the computations scale quadratically as seen in Figure 2. A subject of current research is devoted to adapt the proposed algorithm for embedded MPC applications with a reduced requirement for memory and computations by exploiting the specific structure of the resulting BVLS problems.

VI. Conclusion

In this paper we have proposed a new method for solving BVLS problems. The algorithm is numerically robust, is computationally efficient, and is competitive with respect to state-of-the-art algorithms. The numerical results demonstrate its competitiveness against fast solution methods in solving general small to medium sized BVLS problems such as those arising in embedded MPC, that may also be nearly rank deficient. Numerical stability has been observed even in single precision floating-point arithmetic due to the proposed stable QR update procedures. Several steps of the solution method, such as the classical Gram-Schmidt orthogonalization, involve vector operations that provide scope for much faster implementations on parallel computing platforms.

Acknowledgment

We thank the reviewers for their feedback, Laura Ferrarotti (IMT Lucca) for her helpful comments on the original manuscript and Dr. Daniele Bernardini (ODYS S.r.l.) for discussions, review, and technical support regarding the C implementation of the BVLS solver.

References

[1] P. Stark and R. Parker, “Bounded-variable least-squares: An algorithm and applications,” *Computational Statistics*, vol. 10, pp. 129–141, 1995.
[2] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*. Dordrecht, The Netherlands: Kluwer Academic, 2004.
[3] P. Patrinos and A. Bemporad, “An accelerated dual gradient-projection algorithm for embedded linear model predictive control,” *IEEE Transactions on Automatic Control*, vol. 59, pp. 18–33, 2014.
[4] 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*, vol. 3, pp. 1–122, 2011.
[5] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd, “Embedded code generation using the OSQP solver,” in *Proc. 56th IEEE Conference on Decision and Control*, Melbourne, Australia, 2017, pp. 1906–1911.
[6] N. Saraf and A. Bemporad, “Fast model predictive control based on linear input/output models and bounded-variable least squares,” in *Proc. 56th IEEE Conference on Decision and Control*, Melbourne, Australia, 2017, pp. 1919–1924.
[7] P. Scokaert and J. Rawlings, “Feasibility issues in linear model predictive control,” *AIChE J.*, vol. 45, no. 8, pp. 1649–1659, 1999.
[8] E. Kerrigan and J. Maciejowski, “Soft constraints and exact penalty functions in model predictive control,” in *Proc. UKACC International Conference (Control)*, Cambridge, UK, 2000.
[9] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
[10] S. J. Wright, *Efficient Convex Optimization for Linear MPC*. Cham: Springer International Publishing, 2019, pp. 287–303. [Online]. Available: https://doi.org/10.1007/978-3-319-77489-3_13
[11] D. Kouzoupis, G. Frison, A. Zanelli, and M. Diehl, “Recent advances in quadratic programming algorithms for nonlinear model predictive control,” *Vietnam Journal of Mathematics*, Sept. 2018.
[12] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, ser. Classics in Applied Mathematics. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1995.
[13] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart, “Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization,” *Mathematics of Computation*, vol. 30, no. 136, pp. 772–795, 1976.
[14] N. Saraf, M. Zanon, and A. Bemporad, “A fast NMPC approach based on bounded-variable nonlinear least squares,” in *Proc. 6th IFAC Conference on Nonlinear Model Predictive Control*, Madison, WI, August 2018, pp. 337–342.
[15] J. Nocedal and S. Wright, *Numerical Optimization*. 2nd ed. Springer, 2006.
[16] 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, pp. 1111–1116, 2016.
[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: 4th ed. The John Hopkins University Press, 2013.
[18] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.
[19] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, “qpOASES: A parametric active-set algorithm for quadratic programming,” *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.
[20] Gurobi Optimization, Inc., “Gurobi optimizer reference manual,” 2016. [Online]. Available: http://www.gurobi.com
[21] S. Richter, *Computational complexity certification of gradient methods for real-time model predictive control*. ETH Zurich, Switzerland: Ph.D. dissertation, 2012.
[22] G. Cimini, A. Bemporad, and D. Bernardini, “ODYS QP Solver,” ODYS S.r.l. (https://odys.it/qp), Sept. 2017.
[23] IBM ILOG CPLEX V12.1: User’s Manual for CPLEX, International Business Machines Corporation, 2009.
[24] M. Vukov, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl, “Auto-generated algorithms for nonlinear model predictive control on long and on short horizons,” in *52nd IEEE Conference on Decision and Control*, Florence, Italy, 2013, pp. 5113–5118.
[25] R. Quirynen, A. Knyazev, and S. Di Cairano, “Block structured preconditioning within an active-set method for real-time optimal control,” in *Proc. 2018 European Control Conference (ECC)*, Limassol, Cyprus, 2018, pp. 1154–1159.