Efficient online update of model predictive control in embedded systems using first-order methods

Victor Gracia, Pablo Krupa, Teodoro Alamo, Daniel Limon

arXiv:2309.07996v2 [eess.SY] 16 Dec 2023

Abstract—**Model Predictive Control (MPC) is typically characterized for being computationally demanding, as it requires solving optimization problems online; a particularly relevant point when considering its implementation in embedded systems. To reduce the computational burden of the optimization algorithm, most solvers perform as many offline operations as possible, typically performing the computation and factorization of its expensive matrices offline and then storing them in the embedded system. This improves the efficiency of the solver, with the disadvantage that online changes on some of the ingredients of the MPC formulation require performing these expensive computations online. This article presents an efficient algorithm for the factorization of the key matrix used in several first-order optimization methods applied to linear MPC formulations, allowing its prediction model and cost function matrices to be updated online at the expense of a small computational cost. We show results comparing the proposed approach with other solvers from the literature applied to a linear time-varying system.**

I. INTRODUCTION

Model Predictive Control (MPC) [1], [2] is a well-known control policy due to its ability to optimize the system operation while satisfying its constraints. However, its control law is computationally demanding, as it requires solving an optimization problem every sample time. Many linear MPC solvers have been proposed in the literature using different optimization methods, such as active set [3], interior point [4], or various first-order methods (FOM) such as gradient descent [5], Douglas-Rachford [6], the alternating direction method of multipliers (ADMM) [7] or FISTA [8], to name a few. In particular, FOMs are a popular choice for their use in embedded systems [9], [10], due to their simplicity, good practical performance, and the availability of certification results on their iteration complexity [11], [12], [13]. Due to the low-resource nature of embedded systems, the implementation of these methods often avoids any non-strictly necessary online computations by computing all possible solver components offline, which are then stored in the embedded system [12], [14], [15]. This approach is particularly advantageous when the computation of these elements requires computationally expensive operations, such as matrix inversion or factorization. In particular, factorization of the key solver ingredients is an approach typically taken to reduce the computational complexity per-iteration of the algorithm, see, e.g., [4], [12], [14], [16]. The disadvantage, however, is that in a practical setting it may

no longer be possible to change online some of the ingredients of the MPC controller, such as its prediction model or cost function matrices. That is, the operations required to update the solver for the new resulting optimization problem may be too computationally expensive for the embedded system given the sample time, mostly due to the previously mentioned matrix computations and factorization.

A typical scenario that requires online adjustment of some of these elements is when dealing with linear time-varying (LTV) or linear parameter-varying (LPV) systems [17], where the prediction model of the MPC controller is typically updated every sample time. In fact, the need to update the prediction model will be a requirement when implementing any adaptive MPC control scheme [18]. Additionally, being able to tune online the cost function matrices of the MPC controller is a welcome feature when working in a practical setting, either to fine tune the controller online to improve its performance when controlling the real system or to adapt to a changing economic criterion on the "best" way to control it.

Some quadratic programming (QP) solvers provide the option to recompute and refactor its computationally expensive ingredients online, e.g., [14], [15]. However, the required computation time can be significant when compared to their solve time. In [19], a Ricatti recursion is used to compute the factorization of the linear system to be solved at each iteration of the algorithm. The computation time of this factorization procedure does not depend on the prediction horizon of the MPC controller if its terminal cost is taken as the solution of the discrete-time Ricatti equation associated to the infinite-horizon unconstrained problem.

In this article, we present an algorithm for updating the ingredients used in several FOM-based solvers, including the popular methods ADMM and FISTA, when applied to standard linear MPC formulations whose prediction model and/or weight matrices change online.This algorithm, in essence, is an efficient computation of the Cholesky decomposition of the main computationally expensive matrix used in several solvers [12], [20]. The key feature of the proposed approach is that the matrix factorization is performed exploiting the particular structure that arises from many standard linear MPC formulations, providing a computational advantage when compared to the online factorization routines provided by solvers for generic QPs from the literature. This may lead to tractable re-factorization of the solver ingredients in cases where the re-factorization schemes of generic QP solvers are too slow due to the short sample time or the limitations of the embedded system where the solver is implemented. To illustrate this, we show numerical results comparing the proposed approach to several state-of-the-art QP solvers applied to an LTV system.

Department of Systems Engineering and Automation, Universidad de Sevilla, Spain (e-mails: `vgracia@us.es`, `pkrupa@us.es`, `talamo@us.es`, `dlm@us.es`). Corresponding author: Victor Gracia.

The article is structured as follows. Section II presents the problem formulation. In Section III, we present the proposed algorithm for the online computation and factorization of the solver ingredient. Section IV shows the numerical results. Finally, Section V summarizes the main conclusions.

Notation: The symbol $\mathbb{Z}_a^b$ denotes the set of integers from the integers a to b, both included. For a matrix A, its element in row i and column j is denoted as $A_{i,j}$. We denote $\|x\|_Q \doteq \sqrt{x^\top Q x}$. The symbol I_n denotes the identity matrix of dimension n. Given two vectors x and y, $x \leq (\geq) y$ denotes componentwise inequalities. For vectors x_1 to x_N, $(x_1, x_2, \ldots, x_N)$ denotes the column vector formed by their concatenation. Given scalars and/or matrices $A_1, A_2, \ldots, A_N$ (not necessarily of the same dimensions), we denote by $\mathtt{diag}(A_1, A_2, \ldots, A_N)$ the block diagonal matrix formed by the concatenation of A_1 to A_N.

II. PROBLEM FORMULATION

Consider a discrete-time system described by

$$x(k+1) = Ax(k) + Bu(k), \tag{1}$$

where $x(k) \in \mathbb{R}^n$ and $u(k) \in \mathbb{R}^m$ are the state and input at sample time k, respectively, subject to box constraints

$$\underline{x}(k) \leq x(k) \leq \overline{x}(k), \tag{2a}$$
$$\underline{u}(k) \leq u(k) \leq \overline{u}(k), \tag{2b}$$

where the bounds $\underline{x}(k)$, $\overline{x}(k) \in \mathbb{R}^n$ and $\underline{u}(k)$, $\overline{u}(k) \in \mathbb{R}^m$ satisfy $\underline{x}(k) < \overline{x}(k)$ and $\underline{u}(k) < \overline{u}(k)$. The control objective is to steer the system to a reference pair (x_r, u_r), assumed to be an admissible equilibrium point of the system.

This objective can be attained using one of many linear MPC formulations proposed in the literature [1], [2], [12]. For instance, consider the standard linear MPC formulation

$$\min_{\mathbf{x},\mathbf{u}} \sum_{j=0}^{N-1} \left(\|x_j - x_r\|_Q^2 + \|u_j - u_r\|_R^2 \right) + \|x_N - x_r\|_T^2 \tag{3a}$$

$$\text{s.t. } x_{j+1} = Ax_j + Bu_j, \; j \in \mathbb{Z}_0^{N-1}, \tag{3b}$$
$$x_0 = x(k), \tag{3c}$$
$$\underline{x}_j \leq x_j \leq \overline{x}_j, \; j \in \mathbb{Z}_1^N, \tag{3d}$$
$$\underline{u}_j \leq u_j \leq \overline{u}_j, \; j \in \mathbb{Z}_0^{N-1}, \tag{3e}$$

where N is the prediction horizon; $Q \in \mathbb{R}^{n \times n}$, $R \in \mathbb{R}^{m \times m}$ and $T \in \mathbb{R}^{n \times n}$ are positive definite matrices; and the decision variables $\mathbf{x}$ and $\mathbf{u}$ contain the state and input variables, respectively, along the prediction horizon.

It is well known that, under mild assumptions, system (1) under the control law $u(k) = u_0^*$, where u_0^* is the optimal solution of the decision variable u_0 of (3), asymptotically converges to the admissible reference (x_r, u_r) without violating the system constraints [2]. Therefore, the implementation of the MPC controller requires obtaining the optimal solution of (3) at each sample time.

We remark that formulation (3) has been selected as an example instead of any other linear MPC formulation in order to provide the reader with concrete expressions. However, as

shown later on, the proposed update procedure can be adapted to other linear MPC formulations with minor modifications.

Problem (3) can be rewritten as a standard QP problem

$$\min_z \frac{1}{2} z^\top H z + q^\top z \tag{4a}$$

$$\text{s.t. } Gz = b, \tag{4b}$$

$$\underline{z} \leq z \leq \overline{z}, \tag{4c}$$

by taking

$$H = \mathtt{diag}(R, Q, R, \ldots, Q, R, T), \tag{5a}$$

$$q = -(Ru_r, Qx_r, Ru_r, \ldots, Qx_r, Ru_r, Tx_r), \tag{5b}$$

$$G = \begin{bmatrix} B & -I_n & 0 & \cdots & \cdots & 0 \\ 0 & A & B & -I_n & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & A & B & -I_n \end{bmatrix}, \tag{5c}$$

$$b = -(Ax(k), 0, 0, \ldots, 0), \tag{5d}$$

$$\underline{z} = (\underline{u}_0, \underline{x}_1, \underline{u}_1, \ldots, \underline{x}_{N-1}, \underline{u}_{N-1}, \underline{x}_N), \tag{5e}$$

$$\overline{z} = (\overline{u}_0, \overline{x}_1, \overline{u}_1, \ldots, \overline{x}_{N-1}, \overline{u}_{N-1}, \overline{x}_N). \tag{5f}$$

There are many optimization methods that can efficiently solve problem (4) (active-set method, interior point methods, FOMs, etc.). Among them, FOMs have received a significant amount of attention from the control community, especially when dealing with the implementation of MPC in embedded systems, whose restricted computational and memory resources are a good match for these simple-to-implement algorithms with low resource requirements. Furthermore, a major advantage of solving (4) using FOMs is that the ingredients of the optimization method typically have simple sparse structures due to the sparse, banded nature of (5a) and (5c); a fact exploited by several QP solvers from the literature to obtain efficient implementations, e.g., [12], [14], [16].

For instance, in [12] the authors propose sparse solvers based on ADMM or FISTA for problem (4) arising from simple standard MPC formulations such as (3). The proposed solvers take advantage of the particular structures of H and G that arise from these MPC formulations, i.e., (5a) and (5c). Specifically, they require solving a linear system $Wz = d$ at each iteration of the algorithm, where $W \doteq G(H + \rho I)^{-1}G^\top$ for some non-negative scalar ρ whose value depends on the specific FOM being used. The solvers in [12] solve this linear system by backward-forward substitution using the Cholesky decomposition of W, i.e., the matrix W_c satisfying $W = W_c^\top W_c$, whose simple banded structure is exploited to provide an efficient algorithm. In particular, the structure of W_c resulting from problem (4) is given by

$$W_c = \begin{bmatrix} \beta^1 & \alpha^1 & 0 & \cdots & & 0 \\ 0 & \beta^2 & \alpha^2 & \cdots & & 0 \\ \vdots & \vdots & \beta^3 & \ddots & & \vdots \\ \vdots & \vdots & \vdots & \ddots & \alpha^{N-1} \\ 0 & 0 & 0 & \cdots & & \beta^N \end{bmatrix}, \tag{6}$$

where $\beta^k \in \mathbb{R}^{n \times n}$, $k \in \mathbb{Z}_1^N$, are upper triangular matrices and $\alpha^k \in \mathbb{R}^{n \times n}$, $k \in \mathbb{Z}_1^{N-1}$, are dense. Matrices α^k and β^k, which

---

Algorithm 1: Computation of the sub-matrices of W_c

Require: Igredients that affect W (A, B, Q, R, T)

1 Compute the matrices in Γ (8)
2 **for** $k = 1$ **to** $N - 1$ **do**
3   Compute β^k using (10a) and (10b)
4   Compute α^k using (10c)
5 **end for**
6 Compute β^N using (10a) and (10b)
   Output: β^1 to β^N, α^1 to α^{N-1}

---

depend on the values of the MPC ingredients A, B, Q, R and T, are the main computationally-expensive ingredients of the solver, and are thus typically computed offline and stored in the embedded system, where they are used to efficiently solve the system of equations $Wz = d$ [21, §5.1].

A limitation of these FOM solvers, therefore, is that changing some of the MPC ingredients online requires recomputing and/or refactoring certain matrices, which can be rather time-consuming. In the previous example, changing any of the ingredients A, B, Q, R or T would require an online computation of W_c (6). In other solvers from the literature, such as [14], [15], [16], similar sparse matrices need to be recomputed and refactored if similar MPC ingredients are changed.

The following section presents an efficient algorithm for updating the β^k and α^k sub-matrices of W_c (6) obtained from QP problems (4) whose H and G ingredients have banded structures similar to the ones shown in (5a) and (5c). The motivation behind this algorithm is to provide optimization solvers which can efficiently deal with MPC ingredients that change online, such as when dealing with LTV or LPV systems instead of the time-invariant system (1). We note that MPC formulation (3) is taken as an example whose associated W_c matrix has the structure shown in (6), but that most other standard linear MPC formulations lead to nearly identical structures, requiring, as we will show, only minor modifications of the procedure presented in the following section.

III. ONLINE COMPUTATION OF THE SOLVER INGREDIENTS

We now present equations for recursively computing sub-matrices α^k and β^k of W_c (6) associated with the QP problem (4) for the MPC formulation (3). That is, we provide expressions for computing the non-zero sub-matrices of the Cholesky decomposition of matrix $W \doteq G(H+\rho I)^{-1}G^\top$. We present the final expressions of the equations. The appendix provides an intuitive overview of how they were obtained.

Let us introduce the notation

$$\hat{Q} \doteq (Q + \rho I)^{-1}, \ \hat{R} \doteq (R + \rho I)^{-1}, \ \hat{T} \doteq (T + \rho I)^{-1}, \ \text{(7a)}$$

$$Y \doteq A\hat{Q}A^\top, \ Z \doteq B\hat{R}B^\top, \tag{7b}$$

$$X_Q \doteq Y + Z + \hat{Q}, \ X_T \doteq Y + Z + \hat{T}, \tag{7c}$$

where we recall that the value of $\rho \geq 0$ depends on the FOM being used [12]. Setting $\alpha^0 = 0$ for convenience, we denote

$$\Gamma^k \doteq \begin{cases} Z + \hat{Q}, & \text{if } k = 1, & \text{(8a)} \\ X_Q, & \text{if } k \in \mathbb{Z}_2^{N-1}, & \text{(8b)} \\ X_T, & \text{if } k = N, & \text{(8c)} \end{cases}$$

$$\gamma_{i,j}^k \doteq \sum_{l=1}^{i-1} \beta_{l,i}^k \beta_{l,j}^k + \sum_{q=1}^{n} \alpha_{q,i}^{k-1} \alpha_{q,j}^{k-1}. \tag{9}$$

In (8), Γ^k can be seen intuitively as a variable that gathers the elements of the diagonal band of W that affect β^k. In (9), γ^k groups terms of the non-diagonal elements of β^k and elements of α^{k-1} that affect subsequent elements of β^k. Keeping this in mind, the nonzero elements $\beta_{i,j}^k$ of $\beta^k \in \mathbb{R}^{n \times n}$, $k \in \mathbb{Z}_1^N$, and the elements $\alpha_{i,j}^k$ of $\alpha^k \in \mathbb{R}^{n \times n}$, $k \in \mathbb{Z}_1^{N-1}$, are given by

$$\beta_{i,i}^k = \sqrt{\Gamma_{i,i}^k - \gamma_{i,i}^k}, \quad i \in \mathbb{Z}_1^n, \tag{10a}$$

$$\beta_{i,j}^k = \frac{\Gamma_{i,j}^k - \gamma_{i,j}^k}{\beta_{i,i}^k}, \quad j \in \mathbb{Z}_{i+1}^n, \tag{10b}$$

$$(\beta^k)^\top \alpha^k = -(A\hat{Q})^\top. \tag{10c}$$

Note that the expressions in (7a) require matrix inversions, which are cheap when Q, R and T are small and/or diagonal. However, if this is not the case, one could directly provide $\hat{Q}, \hat{R}$ and $\hat{T}$ in order to reduce its computation time. The computation of α^k using (10c) is simple, since each of its columns is computed as the solution of the upper-triangular linear system $(\beta^k)^\top \alpha_{[j]}^k = -(A\hat{Q})_{[j]}^\top$, where we use subindex $[j]$ to indicate column j of the matrix.

Algorithm 1 shows the procedure for computing α^k and β^k. The computational complexity of Algorithm 1 when applied to the W resulting from (3) is $\mathcal{O}(m^3 + n^3 N)$ flops.

Remark 1. *The computational complexity of Algorithm 1 can be reduced if certain additional assumptions are made, such as matrices Q and R being diagonal.*

An interesting point about equations (9) and (10) is that they are a consequence of the structure of matrix W_c shown in (6). That is, equations (9) and (10) are valid for any MPC formulation whose solver requires solving a linear system $Wz = d$ whose Cholesky factorization W_c has the structure shown in (6). Thus, the only difference between Algorithm 1 applied to each formulation is the matrices required to determine the value of Γ^k (8) and, possibly, the matrix on the right-hand side of (10c). For the MPC formulation (3), the values of Γ^k for each k are shown in (8) and (7), but for other MPC formulations these expressions may differ.

Indeed, let us now consider the standard MPC formulation with terminal equality constraint [12, Eq. (8)], given by

$$\min_{\mathbf{x},\mathbf{u}} \quad \sum_{j=0}^{N-1} ||x_j - x_r||_Q^2 + ||u_j - u_r||_R^2 \tag{11a}$$

$$\text{s.t.} \quad x_{j+1} = Ax_j + Bu_j, j \in \mathbb{Z}_1^{N-1}, \tag{11b}$$

$$x_0 = x(k), \tag{11c}$$

$$\underline{x}_j \leq x_j \leq \overline{x}_j, j \in \mathbb{Z}_1^N, \tag{11d}$$

$$\underline{u}_j \leq u_j \leq \overline{u}_j, j \in \mathbb{Z}_0^{N-1}, \tag{11e}$$

$$x_N = x_r. \tag{11f}$$

In this case, matrices H and G of the resulting QP problem (4) are nearly identical to the ones shown in (5a) and (5c); the only differences being in the final rows of said matrices

due to its different terminal ingredients. Thus, equations (8)-(10) are still valid for updating its α^k and β^k matrices, with the exception that (8c) now uses $Y + Z$ instead of X_T. This change is a consequence of the inclusion of the terminal equality constraint and the removal of the terminal cost in (11), which results in a different expression for Γ^N. However, the expressions for the other submatrices of W_c do not change. For other standard linear MPC formulations whose H and G matrices maintain banded structures similar to the ones shown in (5a) and (5c), similar small modifications of Γ will be required, but overall Algorithm 1 will remain the same. This is the case, for instance, with the MPC formulations from [4, §V.A], [10], [12], [19]. Furthermore, we find that this same procedure may also apply to some non-standard MPC formulations, such as in the MPC for tracking solver presented in [20] (c.f., [20, Eq. (11)]).

Remark 2. *The expression of Γ^N for (11) differs from the one for (3) because (11f) adds an equality constraint, affecting the structure of (5a) and (5c), and thus W. However, if we consider the more general terminal constraint $x_N \in \mathcal{X}_f$, where $\mathcal{X}_f$ is a polyhedron, inequality constraints are added instead of an inequality constraint. Therefore, Algorithm 1 would still be valid as long as the inequality constraints are handled so as to not affect the banded structure of W exploited by Algorithm 1, e.g., uncoupling the equality and inequality constraints using ADMM as in [12] or by using a suitable dualization approach of the inequality constraints as in [19].*

Remark 3. *An alternative approach to the one taken here would be to implement a generic sparse Cholesky decomposition procedure for sparse matrices. The use of generic sparse matrix factorization routines is used by some QP solvers to provide an efficient way to update the solver ingredients online, such as in the OSQP solver [14], which implements a generic sparse LDL factorization routine. This approach, while efficient due to its sparse implementation, is for generic sparse matrices, i.e., using generic sparse matrix representations such as the "compressed sparse column". In contrast, our approach does not make use of sparse matrix representations, but instead directly computes the non-zero elements of the Cholesky decomposition of W by exploiting the specific structure of the matrices H and G shown in (5a) and (5c). That is, our approach loses generality in favor of improving its efficiency by being particularized to the specific structure that typically arises in standard linear MPC formulations, thus providing better computation times than the ones obtained using the update procedures of generic QP solvers.*

Remark 4. *The expressions provided in (7)-(10) for the computation of α^k, $k \in \mathbb{Z}_1^{N-1}$, and β^k, $k \in \mathbb{Z}_1^N$, can be easily adapted in order to handle variations in A, B, Q and R between prediction steps, i.e., having A_j, B_j, Q_j, R_j, $j \in \mathbb{Z}_1^N$, in problem (3). In this case, (7) includes the inversion of N matrices, i.e., computing $\hat{Q}_j^{-1}$, $\hat{R}_j^{-1}$, $j \in \mathbb{Z}_1^N$, as well as the matrix multiplications required to compute $Y_j = A_j \hat{Q}_j A_j^\top$ and $Z_j = B_j \hat{R}_j B_j^\top$, $j \in \mathbb{Z}_1^N$. Consequently, the number of operations of Algorithm 1 increases. Particularly, it presents a computational complexity of $\mathcal{O}((m^3 + n^3)N)$ flops.*

IV. NUMERICAL RESULTS

We consider the continuous stirred-tank reactor (CSTR) system from [22], where a reactant A is transformed into a product B by an exothermic and irreversible chemical reaction. The objective is to control the reaction volume (V), the concentration of A (C_A), and the temperature inside the reactor (T). The manipulable variables are the output flow rate (q_s) and the input coolant flow rate (q_c). The non-linear continuous-time equations of the system are given by

$$\frac{dV(t)}{dt} = q_e - q_s(t),$$

$$\frac{dC_A(t)}{dt} = \frac{q_e}{V(t)}(C_{Ae} - C_A(t)) - k_0 e^{\frac{-E}{RT(t)}} C_A(t),$$

$$\frac{dT(t)}{dt} = \frac{q_e}{V(t)}(T_e - T(t)) - k_1 e^{\frac{-E}{T(t)}} C_A(t)$$
$$+ \frac{q_c(t)}{V(t)} k_2 (1 - e^{\frac{-k_3}{q_c(t)}})(T_{ce} - T(t)),$$

where $k_1 = \frac{\Delta H k_0}{\rho C_p}$, $k_2 = \frac{\rho C_{pc}}{\rho_c C_p}$, $k_3 = \frac{h_A}{\rho_c C_{pc}}$ and the values of the system parameters are presented in [22]. We consider an LPV model of the system, obtained from the above non-linear equations as shown in [22], with a frozen scheduling parameter, i.e., the matrices of the discrete LPV system model change at every sample time, and thus also the MPC prediction model, which remains constant along its prediction horizon.

We take the sampling time of the system as 0.1 seconds, and control it using the MPC formulation (3). We implement the ADMM and FISTA solvers from [12], available in the SPCIES toolbox (version v0.3.8) [23], which include Algorithm 1. We also solve the MPC optimization problem using the OSQP (version 0.6.2) [14] and SCS (version 3.2.3) [15] solvers, which include a generic update procedure, as discussed in Remark 3. We note that OSQP also implements ADMM and SCS implements a similar operator splitting algorithm. The simulations are run on an Intel Core i5-1135G7.

Table I shows the computation times when taking average results from 200 simulations using random initial conditions and references within the system constraints described in [22]. Computation times are separated into "update-time", which is the time taken by the procedure to update the solver ingredients due to the modifications of the MPC parameters and ingredients (this includes the computation time of Algorithm 1 in the case of the SPCIES solvers), and "solve-time", which is the remaining computation time. The results show a clear difference in terms of computation time between the update-time of Algorithm 1 w.r.t. OSQP or SCS.

In order to show that the updating procedure we propose works properly when combined with time-varying MPC approaches, and that the time results we shown in Table I are obtained from a suitably designed MPC controller (in the sense of proper closed-loop behavior), we include Figure 1, which shows the closed-loop result of one of the simulations of Table I using the solver from [23], where the MPC model changes every sample time due to the LPV approach used, and the cost function matrices Q and R were changed from $Q = 10I_n$ to $Q = I_n$ and from $R = \texttt{diag}(10, 0.1)$ to $R = I_m$

Solver	N	Update-time [microseconds]				Solve-time [microseconds]				%
		Average	Median	Maximum	Minimum	Average	Median	Maximum	Minimum	
FISTA	5	1.03 (0.08)	1.02 (0.07)	1.30 (0.17)	0.97 (0.06)	2.69	2.06	12.25	1.36	27.75
ADMM	5	1.05 (0.11)	1.04 (0.10)	1.36 (0.20)	0.96 (0.08)	71.01	23.49	686.3	15.14	1.45
OSQP	5	18.03	18.01	19.81	17.45	20.41	18.39	43.29	17.59	46.91
SCS	5	101.6	101.6	109.9	99.20	180.1	212.2	245.7	104.1	36.08
FISTA	15	1.68 (0.20)	1.67 (0.19)	2.11 (0.57)	1.62 (0.17)	6.14	4.47	29.40	2.71	21.52
ADMM	15	1.81 (0.22)	1.80 (0.21)	2.35 (0.47)	1.72 (0.19)	156.5	100.2	1428.3	50.83	1.15
OSQP	15	36.95	36.93	38.73	35.94	53.39	47.54	159.65	46.05	40.90
SCS	15	233.7	233.7	244.9	229.6	560.8	597.7	663.6	339.1	29.42

The update times of algorithms FISTA and ADMM have been obtained using SPCIES with Algorithm 1 in its update procedure. In brackets, we include the update time of the same SPCIES solvers when Algorithm 1 is not executed, i.e., without updating A, B, Q and/or R. Column "%" shows (Avg. update-time)/(Avg. update-time + Avg. solve-time). The ADMM penalty parameter of SPCIES is set to $\rho = 0.01$.

TABLE I: Computation times of the MPC solvers applied to the CSTR system for the MPC formulation (3).
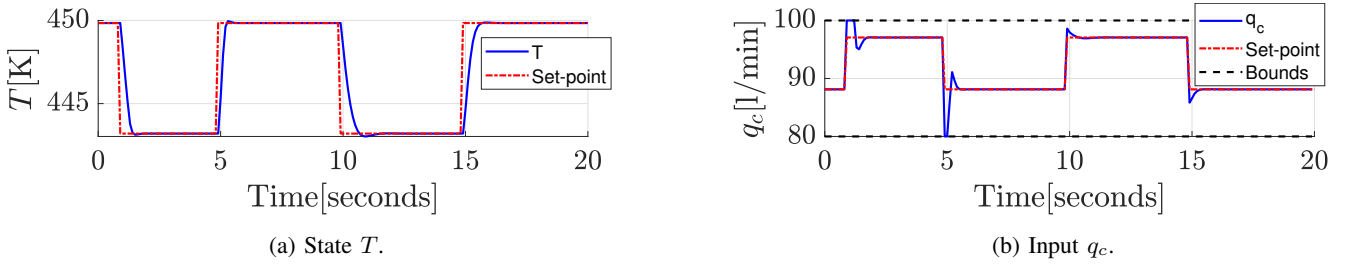

(a) State T.

(b) Input q_c.

Fig. 1: Non-linear CSTR model controlled with LPV frozen scheduling parameter MPC approach.

(a) Dependency on n ($m = 2$, $N = 5$)

(b) Dependency on m ($n = 60$, $N = 5$)

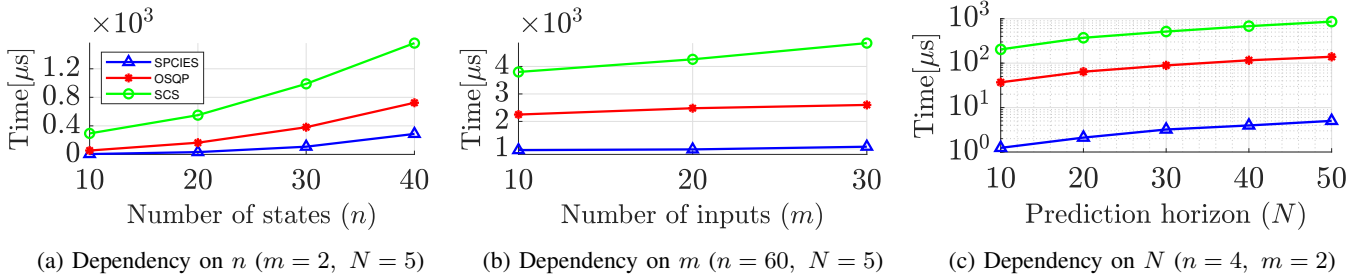(c) Dependency on N ($n = 4$, $m = 2$)

Fig. 2: Dependency of update times on n, m and N.

at time $t = 9$ seconds, explaining the different system behavior between the first and second half of the simulation

Finally, Figure 2 presents results on the computation time of Algorithm 1 in terms of the number of states (n), inputs (m) and the prediction horizon (N) as well as the update times when using the OSQP and SCS solvers. Note that Figure 2c is shown in logarithmic scale in order to improve its readability. The results show that the proposed approach provides faster computation times than when using state-of-the-art generic QP solvers, as well as better scaling as the dimensions of the MPC problem increase. This is due to our particularization of the update procedure to the specific matrix structure that arises in linear MPC problems. We note that the update-time of Algorithm 1 grows linearly with the prediction horizon N, as discussed in Section III, since the number of matrices α^k and β^k is linear in N.

Remark 5. *We do not include a comparison with [19] because we focus on comparisons with state-of-the-art solvers available online. The computational complexity of the "update-step" [19, Algorithm 3] does not depend on the prediction horizon N if the terminal cost of the MPC controller is taken as the solution of the discrete LQR Riccati equation. In this case, its computation time would be smaller than the one obtained with Algorithm 1. However, if this is not the case, or if the system is not LTI along the prediction horizon, Algorithm 1 outperforms it in terms of the number of required operations.*

V. Conclusion

This article has presented an efficient method to perform the matrix factorization required by several FOM optimization solvers from the literature for linear MPC formulations. The proposed update method is based on exploiting the particular structure that typically arises from standard MPC formulations, thus providing a computational advantage w.r.t. the general sparse update procedures used by many state-of-the-art QP solvers, as shown in the numerical results we have presented, where computation times were up to an order of magnitude faster. Combined with the solvers presented in [12] (available

in [23]), which are also based on exploiting MPC-specific matrix structures, this becomes an interesting option for applying MPC in under-powered devices when an online change of the prediction model or the tuning parameters of the MPC controller are required.

APPENDIX

This section provides an intuitive overview of how the update expressions for α^k and β^k presented in Section III are derived. Recalling the definition (7), $W \doteq G(H + I_\rho)^{-1}G^\top$ is given by

$$W = \begin{bmatrix} Z + \hat{Q} & -\hat{Q}A^\top & 0 & \cdots & 0 \\ -A\hat{Q} & X_Q & -\hat{Q}A^\top & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & -A\hat{Q} & X_Q & -\hat{Q}A^\top \\ 0 & 0 & 0 & -A\hat{Q} & X_T \end{bmatrix}. \quad (13)$$

The structure of its Cholesky decomposition is given by (6). Using the identity $W = W_c^\top W_c$, we can derive the following equations relating α^k and β^k with the matrices in (13):

$$(\beta^1)^\top \beta^1 = Z + \hat{Q}, \ (\beta^1)^\top \alpha^1 = -(A\hat{Q})^\top,$$
$$(\beta^2)^\top \beta^2 = X_Q - (\alpha^1)^\top \alpha^1, \ (\beta^2)^\top \alpha^2 = -(A\hat{Q})^\top,$$
$$\vdots$$
$$(\beta^N)^\top \beta^N = X_T - (\alpha^{N-1})^\top \alpha^{N-1}.$$

For matrices β^k, we can use their upper-triangular nature to further expand the above equations to obtain expressions for their non-zero elements. For instance, for β^1, we have

$$(\beta^1)^\top \beta^1 = \begin{bmatrix} \beta^1_{1,1} & 0 & 0 \\ \vdots & \ddots & 0 \\ \beta^1_{1,n} & \cdots & \beta^1_{n,n} \end{bmatrix} \begin{bmatrix} \beta^1_{1,1} & \cdots & \beta^1_{1,n} \\ 0 & \ddots & \vdots \\ 0 & 0 & \beta^1_{n,n} \end{bmatrix} = Z + \hat{Q},$$

from where we derive

$$(\beta^1_{1,1})^2 = (Z + \hat{Q})_{1,1}, \ \beta^1_{1,2} = (Z + \hat{Q})_{1,2}/\beta^1_{1,1}, \ \dots,$$
$$\beta^1_{1,n} = (Z + \hat{Q})_{1,n}/\beta^1_{1,1}, \ (\beta^1_{2,2})^2 = (Z + \hat{Q})_{2,2} - (\beta^1_{1,2})^2,$$
$$\beta^1_{2,3} = [(Z + \hat{Q})_{2,3} - \beta^1_{1,2}\beta^1_{1,3}]/\beta^1_{2,2}, \ \dots,$$
$$\beta^1_{2,n} = [(Z + \hat{Q})_{2,n} - \beta^1_{1,2}\beta^1_{1,n}]/\beta^1_{2,2}, \ \dots,$$
$$(\beta^1_{n,n})^2 = (Z + \hat{Q})_{n,n} - (\beta^1_{1,n})^2 - \cdots - (\beta^1_{n-1,n})^2. \quad (14)$$

Continuing with β^2, we have

$$(\beta^2)^\top \beta^2 = X_Q - (\alpha^1)^\top \alpha^1,$$

which leads to the same expressions as (14) but substituting β^1 with β^2 and $Z + \hat{Q}$ with $X_Q - (\alpha^1)^\top \alpha^1$. The same procedure can be performed for each β^k up to β^N, which satisfies

$$(\beta^N)^\top \beta^N = X_T - (\alpha^{N-1})^\top \alpha^{N-1}.$$

Equations (10a) and (10b) are a generalization of (14), where Γ^k (8) accounts for the matrices of (13) that affect each β^k, e.g., $Z + \hat{Q}$ for β^1 or X_T for β^N, and γ (9) accounts for the effect of the "previous" values of the elements of β^k and α^{k-1} that affect the current $\beta^k_{i,j}$, as shown in (14). Finally, we take the convention $\alpha^0 = 0$ so that we can also add α^0 to the expression for β^1 in (10).

REFERENCES

[1] E. F. Camacho and C. B. Alba, *Model Predictive Control*. Springer Science & Business Media, 2013.

[2] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*, 2nd ed. Madison, Wisconsin: Nob Hill Publishing, 2017.

[3] D. Arnström, A. Bemporad, and D. Axehill, "A dual active-set solver for embedded quadratic programming using recursive LDL^T updates," *IEEE Transactions on Automatic Control*, vol. 67, no. 8, pp. 4362–4369, 2022.

[4] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *51st IEEE Conference on Decision and Control (CDC)*, 2012, pp. 668–674.

[5] T. Skibik and M. M. Nicotra, "Analysis of time-distributed model predictive control when using a regularized primal–dual gradient optimizer," *IEEE Control Systems Letters*, vol. 7, pp. 235–240, 2023.

[6] R. Halvgaard, L. Vandenberghe, N. K. Poulsen, H. Madsen, and J. B. Jørgensen, "Distributed model predictive control for smart energy systems," *IEEE Transactions on Smart Grid*, vol. 7, no. 3, pp. 1675–1682, 2016.

[7] 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, no. 1, pp. 1–122, 2011.

[8] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

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

[10] D. Kouzoupis, H. J. Ferreau, H. Peyrl, and M. Diehl, "First-order methods in embedded nonlinear model predictive control," in *European Control Conference (ECC)*, 2015, pp. 2617–2622.

[11] P. Zometa, M. Kögel, T. Faulwasser, and R. Findeisen, "Implementation aspects of model predictive control for embedded systems," in *American Control Conference (ACC)*, 2012, pp. 1205–1210.

[12] P. Krupa, D. Limon, and T. Alamo, "Implementation of model predictive control in programmable logic controllers," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 1117–1130, 2021.

[13] P. Krupa, O. Inverso, M. Tribastone, and A. Bemporad, "Certification of the proximal gradient method under fixed-point arithmetic for box-constrained QP problems," *Automatica*, vol. 159, 2024.

[14] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[15] B. O'Donoghue, "Operator splitting for a homogeneous embedding of the linear complementarity problem," *SIAM Journal on Optimization*, vol. 31, pp. 1999–2023, August 2021.

[16] M. Garstka, M. Cannon, and P. Goulart, "COSMO: A conic operator splitting method for convex conic problems," *Journal of Optimization Theory and Applications*, vol. 190, no. 3, pp. 779–810, 2021.

[17] J. Bokor and G. Balas, "Linear parameter varying systems: A geometric theory and applications," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 12–22, 2005, 16th IFAC World Congress.

[18] G. Tao, "Multivariable adaptive control: A survey," *Automatica*, vol. 50, no. 11, pp. 2737–2764, 2014.

[19] 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.

[20] P. Krupa, I. Alvarado, D. Limon, and T. Alamo, "Implementation of model predictive control for tracking in embedded systems using a sparse extended ADMM algorithm," *IEEE Transactions on Control Systems Technology*, vol. 30, no. 4, pp. 1798–1805, 2022.

[21] P. Krupa, "Implementation of MPC in embedded systems using first order methods," Ph.D. dissertation, University of Seville, 2021, available at arXiv:2109.02140.

[22] E. Bernardi, H. A. Pipino, and E. J. Adam, "Full-order output observer applied to a linear parameter varying system with unknown input," in *IEEE Congreso Bienal de Argentina (ARGENCON)*, 2020, pp. 1–8.

[23] P. Krupa, V. Gracia, D. Limon, and T. Alamo, "Spcies: Suite of predictive controllers for industrial embedded systems," 2020. [Online]. Available: https://github.com/GepocUS/Spcies