Automatica 141 (2022) 110293

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

An efficient bounded-variable nonlinear least-squares algorithm for embedded MPC☆

Nilay Saraf, Alberto Bemporad *

IMT School for Advanced Studies, Piazza San Francesco 19, Lucca, Italy

ARTICLE INFO

Article history:
Received 25 March 2021
Received in revised form 17 October 2021
Accepted 22 December 2021
Available online xxxx

Keywords:
Model predictive control
Active-set methods
Nonlinear parameter-varying control
Adaptive control
Nonlinear programming
Recursive QR factorization

ABSTRACT

This paper presents a novel approach to solve linear and nonlinear model predictive control (MPC) problems that requires small memory footprint and throughput and is particularly suitable when the model and/or controller parameters change at runtime. The contributions of the paper include: (*i*) a formulation of the nonlinear MPC problem as a bounded-variable nonlinear least-squares (BVNLS) problem, demonstrating that the use of an appropriate solver can outperform industry-standard solvers; (*ii*) an easily-implementable library-free BVNLS solver with a novel proof of global convergence; (*iii*) a matrix-free method for computing the products of vectors and Jacobians, required by BVNLS; (*iv*) an efficient method for updating sparse QR factors when using active-set methods to solve sparse BVNLS problems. Thanks to explicitly parameterizing the optimization algorithm in terms of the model and MPC tuning parameters, the resulting approach is inherently and immediately adaptive to any changes in the MPC formulation. The same algorithmic framework can cope with linear, nonlinear, and adaptive MPC variants based on a broad class of prediction models and sum-of-squares cost functions.

1. Introduction

Model predictive control has evolved over the years from a method developed for controlling slow processes (Qin & Badgwell, 2003; Rafal & Stevens, 1968) to an advanced multivariable control method that is applicable even to fast-sampling applications, such as in the automotive and aerospace domains (Bemporad, Bernardini, Long, & Verdejo, 2018; Di Cairano & Kolmanovsky, 2018). This evolution has been possible because of the significant amount of research on computationally efficient real-time MPC algorithms. For an incomplete list of such efforts and tools the reader is referred to Cannon (2004), Diehl, Bock, and Schlöder (2005), Diehl, Ferreau, and Haverbeke (2009), Jerez et al. (2014), Kouzoupis, Frison, Zanelli, and Diehl (2018), Ohtsuka (2004), Stella, Themelis, Sopasakis, and Patrinos (2017), Wang and Boyd (2010). Despite the success of MPC, demand for faster algorithms for a wider scope of applications has been reported for instance in Di Cairano and Kolmanovsky (2018). A common approach to reduce computations is to solve the MPC problem suboptimally, see for instance Diehl et al. (2005), Wang and Boyd (2010). However, even such MPC approaches have limitations that could be prohibitive in some resource-constrained applications, especially in the case of (parameter-varying) nonlinear MPC (NMPC). This denotes that there is still a large scope of improvement.

A usual practice in MPC is to first formulate an optimization problem, for instance by computing the Hessian matrix in case of QP formulations, or NLP sub-problems, based on the prediction model and MPC tuning parameters, before passing it in a standard form to an optimization solver. Such a problem construction step can be performed *offline* when the prediction model is time-invariant, e.g., a linear time-invariant (LTI) model, whereas it needs to be repeated at each instance in case of parameter-varying models, as in case of a nonlinear model linearized at the current operating point, or changes in MPC tuning parameters (such as prediction horizon, control horizon, tuning weights). Often, constructing the optimization problem requires a computational effort comparable to that required for solving it. The same occurs in the recently proposed data-driven MPC scheme (Piga, Forgione, Formentin, & Bemporad, 2019) where, due to potentially time-varying model and tuning parameters, re-constructing the MPC optimization problem online becomes necessary, which significantly increases the computational load. Notwithstanding these limitations of MPC, scarcely any effort has been made to date to design a real-time MPC approach which does not need

☆ This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement No 674875 (oCPS). The material in this paper was not presented at any conference. This paper was recommended for publication in revised form by Associate Editor Martin Monnigmann under the direction of Editor Ian R. Petersen.

* Corresponding author.
E-mail addresses: nilay.saraf@alumni.imtlucca.it (N. Saraf), alberto.bemporad@imtlucca.it (A. Bemporad).

https://doi.org/10.1016/j.automatica.2022.110293
0005-1098/

(re-)construction of the optimization problem with varying model and/or MPC tuning parameters. Approaches which partially address this aspect for a limited class of linear parameter-varying (LPV) models with a fixed MPC problem structure include Cavanini, Cimini, and Ippoliti (2018), Wang and Boyd (2010). The methods proposed in this paper aim at reducing the computational complexity of MPC while eliminating the optimization problem construction step, through algorithms that can adapt to changes in the model and/or tuning parameters at runtime. The main ideas employed for this purpose are: (1) a reformulation of the MPC problem through a quadratic penalty function in order to exploit fast solution methods, (2) replacing matrix instances via abstract operators that map the model and tuning parameters to the result of the required matrix operations in the optimization algorithm. Besides this, the contributions of this paper include: (1) an overview on equality constraint elimination in the considered (nonlinear) MPC problems, (2) a discussion on alternative methods to implement MPC with resulting algorithms having negligible memory scaling w.r.t. the number of decision variables, (3) methods to exploit problem sparsity and efficiently implement the active-set method proposed in Saraf and Bemporad (2020) for MPC based on box-constrained (nonlinear) least-squares.

Regarding the last contribution, we note that each iteration of a primal active-set method (Nocedal & Wright, 2006) involves the solution of a linear system, which in the case of the algorithm in Saraf and Bemporad (2020) is a sparse unconstrained linear least-squares (LS) problem. These LS problems between successive iterations are related by a rank-one update. In Saraf and Bemporad (2020), it has been shown that for the numerically *dense* case, as compared to solving an LS problem from scratch, employing a recursive QR factorization scheme that exploits the relation between successive LS problems can significantly increase computational speed without compromising numerical robustness, even without using advanced linear algebra libraries. In the *sparse* case, even though very efficient approaches exist for solving a single LS problem using direct (Davis, 2006) or iterative (Saad, 2003) methods with sparse linear algebra libraries, to the best of the authors' knowledge no methods have been reported for recursively updating the sparse QR factorization of a matrix. A recursive approach for sparse LU factorization has been described in Dongarra, Eijkhout, and Łuszczek (2001); however, such an approach not only needs to store the matrix and its sparsity pattern, which requires constructing the MPC problem and forming the normal equations that could be numerically susceptible to ill-conditioning, but also relies on linear algebra packages that could be difficult to embed in a control platform. In this paper, we present novel methods for numerically stable sparse recursive QR factorizations based on Gram–Schmidt orthogonalization, which are easy to implement and are very efficient even for small-size problems, therefore extending the dense approach of Saraf and Bemporad (2020). Although the proposed methods are designed for the specific MPC application, they may be applicable for other LS problems having similar special structures.

The paper is organized as follows. Section 2 describes the considered general class of discrete-time models and MPC problem formulation. The proposed formulation based on eliminating the equality constraints is reported in Section 3. In Section 4 we describe a solution algorithm for bound-constrained nonlinear least-squares optimization with a theoretical analysis of its global convergence. A parameterized implementation of this algorithm for solving MPC problems without the construction phase and relying on the abstraction of matrix instances is described in Section 5. Methods for sparse recursive thin QR factorization are described in Section 6. Section 7 briefly reports numerical results based on a nonlinear MPC (NMPC) benchmark example that demonstrate the very good computational performance of the proposed methods against other methods. Finally, Section 8 concludes the paper.

Excerpts of Sections 2–3, and Section 4.1 are based on the authors' conference papers (Saraf & Bemporad, 2017; Saraf, Zanon, & Bemporad, 2018).

Notation. We denote the set of real vectors of dimension n as $\mathbb{R}^n$; 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 pseudo-inverse as $A^\dagger$. For a vector $a \in \mathbb{R}^m$, its p-norm is $\|a\|_p$, its jth element is $a(j)$, and $\|a\|_2^2 = a^\top a$. A vector or matrix with all zero elements is represented by $\mathbf{0}$. If $\mathcal{F}$ denotes a set of indices, $A_\mathcal{F}$ denotes a matrix formed from columns of A corresponding to the indices in $\mathcal{F}$. Given N square matrices $A_1, \dots, A_N$, of possible different orders, blockdiag$(A_1, \dots, A_N)$ is the block diagonal matrix with diagonal blocks $A_1, \dots, A_N$. For scalars a and b, $\min(a, b)$ and $\max(a, b)$ denote, respectively, the minimum and maximum of the two values. Depending on the context, $(a, b]$ or $[b, a)$ represent either the set of real numbers or integers between a and b, excluding a and including b. The gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $\bar{x} \in \mathbb{R}^n$ is either denoted by $\nabla_x f(x)|_{\bar{x}}$ or $\nabla_x f(\bar{x})$, the Hessian matrix by $\nabla_x^2 f(\bar{x})$; the Jacobian of a vector function $g : \mathbb{R}^n \to \mathbb{R}^m$ by $\mathbf{J}_x g(x)|_{\bar{x}}$ or $\mathbf{J}g(\bar{x})$.

Finite sets of elements are represented by curly braces containing the elements; $\emptyset$ denotes the empty set. If a set $\mathcal{A}$ is a subset of set $\mathcal{B}$ (i.e., if $\mathcal{B}$ is the superset of $\mathcal{A}$), it is written as $\mathcal{A} \subseteq \mathcal{B}$ (or alternatively $\mathcal{B} \supseteq \mathcal{A}$). The symbols $\cup$, $\cap$, and $\setminus$ between two sets denote, respectively, set union, intersection, and difference. The summation notation for sets is denoted by $\bigcup$. The cardinality (number of elements) of a finite set $\mathcal{A}$ is denoted by $|\mathcal{A}|$.

2. Preliminaries

For maximum generality, we describe the prediction model for MPC by the discrete-time multivariable nonlinear parameter-varying dynamical model equation

$$\mathcal{M}(Y_k,\ U_k,\ S_k) = \mathbf{0}, \tag{1}$$

where $U_k = (u_{k-n_\mathrm{b}}, \dots, u_{k-1})$ with $u_k \in \mathbb{R}^{n_u}$ the input vector at sampled time step k, and $Y_k = (y_{k-n_\mathrm{a}}, \dots, y_k)$ with $y_k \in \mathbb{R}^{n_y}$ the output vector at k. Vector $S_k = (s_{k-n_\mathrm{c}}, \dots, s_{k-1})$, where $s_k \in \mathbb{R}^{n_s}$, $n_s \geq 0$, contains possible exogenous signals, such as measured disturbances. We assume that function $\mathcal{M} : \mathbb{R}^{n_\mathrm{a}n_y} \times \mathbb{R}^{n_\mathrm{b}n_u} \times \mathbb{R}^{n_\mathrm{c}n_s} \to \mathbb{R}^{n_y}$ is differentiable, where n_a, n_b and n_c denote the model order. Special cases include deterministic nonlinear parameter-varying auto-regressive exogenous (NLPV-ARX) models, state–space models (y = state vector, $n_a = n_b = n_c = 1$), neural networks with a smooth activation function, discretized first-principles models and differential algebraic equations. Designing the MPC controller based on (1) has several benefits such as: 1) data-based black-box models which are often identified in input–output form do not need a state–space realization, 2) a state estimator is not required when all output and exogenous variables are measured, 3) the number of variables to be optimized does not scale linearly with the number of states but outputs, which could be fewer in number, 4) input delays can easily be incorporated in the model by simply shifting the sequence in U_k backwards in time.

Linearizing (1) w.r.t. a sequence of inputs $\hat{U}$ (that is, $U_k = \hat{U} + \Delta U$) and outputs $\hat{Y}$ ($Y_k = \hat{Y} + \Delta Y$) gives

$$\mathcal{M}(\hat{Y},\ \hat{U},\ S_k) + \left(\mathbf{J}_{Y_k}\mathcal{M}(Y_k,\ U_k,\ S_k)|_{\hat{Y},\ \hat{U}}\right)\Delta Y + \left(\mathbf{J}_{U_k}\mathcal{M}(Y_k,\ U_k,\ S_k)|_{\hat{Y},\ \hat{U}}\right)\Delta U = \mathbf{0},$$

which is equivalently written as the following affine parameter-varying input–output model, i.e.,

$$-A(S_k)_0\,\Delta y_k = \sum_{j=1}^{n_a} A(S_k)_j\,\Delta y_{k-j} + \sum_{j=1}^{n_b} B(S_k)_j\,\Delta u_{k-j} + \mathcal{M}(\hat{Y},\ \hat{U},\ S_k), \tag{2}$$

where the Jacobian matrices

$$A(S_k)_j = \mathbf{J}_{y_{k-j}}\mathcal{M}(Y_k,\ U_k,\ S_k)\big|_{\hat{Y},\ \hat{U}} \in \mathbb{R}^{n_y\times n_y},\ \forall j\in[0,n_a],$$
$$B(S_k)_j = \mathbf{J}_{u_{k-j}}\mathcal{M}(Y_k,\ U_k,\ S_k)\big|_{\hat{Y},\ \hat{U}} \in \mathbb{R}^{n_y\times n_u},\ \forall j\in[1,n_b].$$

Note that for the special case of LTI models in ARX form, in (2) A_0 would be an identity matrix whereas S_k would be absent and $\mathcal{M}(\hat{Y},\ \hat{U}) = \mathbf{0}$, $\hat{Y} = \mathbf{0}$, $\hat{U} = \mathbf{0}$.

We consider a performance index (P) which is commonly employed for reference tracking in MPC:

$$\mathrm{P}_k = \sum_{j=1}^{N_\mathrm{p}} \frac{1}{2}\|W_{y_{k+j}}(y_{k+j}-\bar{y}_{k+j})\|_2^2 + \sum_{j=0}^{N_\mathrm{u}-2} \frac{1}{2}\|W_{u_{k+j}}(u_{k+j}-\bar{u}_{k+j})\|_2^2 + \frac{1}{2}(N_\mathrm{p}-N_\mathrm{u}+1)\cdot\|W_{u_{k+N_\mathrm{u}-1}}(u_{k+N_\mathrm{u}-1}-\bar{u}_{k+N_\mathrm{u}-1})\|_2^2, \tag{3}$$

where N_p and N_u denote the prediction and control horizon respectively. Matrices $W_{y_{(\cdot)}} \in \mathbb{R}^{n_y\times n_y}$, $W_{u_{(\cdot)}} \in \mathbb{R}^{n_u\times n_u}$ denote tuning weights, vectors $\bar{y}$, $\bar{u}$ denote output and input references, respectively.

The methods described later in this paper can straightforwardly be extended to any performance index which is a sum of squares of linear or differentiable nonlinear functions. The MPC optimization problem is formulated based on the cost function (3) subject to constraints on vector $w_k = (u_k,\ldots,u_{k+N_\mathrm{u}-1}, y_{k+1},\ldots,y_{k+N_\mathrm{p}})$. We will consider equality constraints that arise from the prediction model (1), and restrict inequality constraints to only simple bounds on input and output variables. General inequality constraints $g(w_k)\le\mathbf{0}$ can nevertheless be included as equalities by softening them, i.e., by introducing non-negative slack variables $v\in\mathbb{R}^{n_\mathrm{i}}$

$$g(w_k)+v_k=\mathbf{0},\quad v_k\ge 0, \tag{4}$$

where $z_k = (w_k, v_k)$ and $g:\mathbb{R}^{(n_z-n_\mathrm{i})}\to\mathbb{R}^{n_\mathrm{i}}$ is assumed to be differentiable, while the number of decision variables is n_z and of general inequality constraints is n_i. In summary, the problem to be solved at each step k is

$$\min_{z_k}\ \frac{1}{2}\|W_k(z_k-\bar{z}_k)\|_2^2 \tag{5a}$$
$$\text{s.t. } h_k(z_k,\ \phi_k)=\mathbf{0}, \tag{5b}$$
$$p_k\le z_k\le q_k, \tag{5c}$$

where p_k, q_k are vectors defining bounds on the input and output variables, and possible non-negativity constraint on slack variables. Unbounded components of z_k may be passed to the solver we propose later as the largest negative or positive floating-point number in the computing platform, so that we can assume $p_k, q_k \in \mathbb{R}^{n_z}$. Vector $\phi_k = (u_{k-n_\mathrm{b}+1},\ldots,u_{k-1},y_k,\ldots,y_{k-n_\mathrm{a}+1})$ denotes the initial condition whereas $\bar{z}$ contains references on the decision variables. The block-sparse matrix W_k is formed from the tuning weights $(W_{u_{(\cdot)}}, W_{y_{(\cdot)}})$ defined in (3).

3. Eliminating equality constraints

Handling equality constraints via penalty functions, or an augmented Lagrangian method, has proven to be effective for efficiently solving constrained optimization problems (Bertsekas, 1996; Nocedal & Wright, 2006),(Lawson & Hanson, 1995, Chapter 22). This section shows how similar methods can be applied to efficiently solve MPC problems of the form (5). In order to employ fast solution methods, the general constrained problem (5) can be simplified as a box-constrained nonlinear least-squares (BVNLS) problem by using a quadratic penalty function and consequently eliminating the equality constraints (5b) such that (5) becomes

$$\min_{p_k\le z_k\le q_k}\ \frac{1}{2}\left\|\begin{matrix}\frac{1}{\sqrt{\rho}}W_k(z_k-\bar{z}_k)\\ h_k(z_k,\ \phi_k)\end{matrix}\right\|_2^2 \equiv \min_{p_k\le z_k\le q_k}\ \frac{1}{2}\|r_k(z_k)\|_2^2, \tag{6}$$

where the penalty parameter ρ is a positive scalar and $r:\mathbb{R}^{n_z}\to\mathbb{R}^{n_r}$ denotes the vector of residuals. We propose the reformulation (6) of problem (5) for the following reasons: *i*) penalizing the violation of equality constraints makes problem (6) always feasible; *ii*) no additional slack variables are needed for softening output constraints, which would result in inequalities of general type instead of box constraints; *iii*) while solving (6), since we do not include additional slack variables to soften constraints, the function h_k does not need to be analytic beyond bounds, which is discussed in further detail in Section 4 (cf. Remark 1); *iv*) no dual variables need to be optimized to handle equality constraints; *v*) problem (6) is simpler to solve as compared to (5), for instance, when using SQP algorithms (cf. Section 4), initializing a feasible guess is straightforward, the subproblems are feasible even with inconsistent linearizations of h_k, and convergence impeding phenomena such as the Maratos effect (Nocedal & Wright, 2006) are implicitly avoided.

Relaxing the equality constraints as above also has an engineering justification (Saraf & Bemporad, 2017): As the prediction model (1) is only an approximation of the true system dynamics, (opportunistic) violations of the dynamic model equations will only affect the quality of predictions, depending on the magnitude of the violation. Instead of using the iterative quadratic penalty method (QPM) (Nocedal & Wright, 2006, Framework 17.1) with increasing values of ρ in each iteration, we propose to use a single iteration with a large value of ρ for solving (5), owing to the fact that a good initial guess is often available in MPC. It has been proven in Saraf and Bemporad (2017, Theorem 1) that for a quadratic cost (5a) subject to only consistent linear equality constraints, a single QPM iteration with sufficiently large penalty ρ may result in negligible solution inaccuracy. This has been clearly demonstrated by numerical examples in Saraf and Bemporad (2017), Saraf et al. (2018) for the general case. A practical upper bound on ρ depends on the computation precision and numerical robustness of the optimization solver such that the Jacobian of the vector of residuals in (6) is numerically full-rank. The parameter ρ is tuned (cf. Saraf (2019, Section 3.5.3)) based on the fact that a higher value results in a lower solution inaccuracy at the cost of problem scaling which may affect the convergence rate of the adopted solution methods. A theoretical lower bound on ρ exists and has been derived in Saraf and Bemporad (2017) for the case of LTI systems based on closed-loop stability conditions. The extension of such a result to the general case is not immediate and thereby poses a risk given that the bound is not deterministic. However, in practice, based on the arguments and details in the references mentioned in this section, we expect a sufficiently low value of the equality constraint violation.

An alternative approach to solve (5) without the equality constraints (5b) is the bound-constrained Lagrangian method (BLM) (Nocedal & Wright, 2006, Algorithm 17.4). The methods described later may be applied to solve BLM too but since it is not the focus here, a note discussing BLM in relevance to QPM is included in Appendix.

Algorithm 1 Bounded-Variable Nonlinear Least Squares (BVNLLS) solver

Inputs: Bounds $p, q \in \mathbb{R}^{n_z}$, feasible initial guess z, $b = r(z)$, optimality tolerance $\gamma \geq 0$, $c \in (0, 0.5)$, $\tau \in (0, 1)$.

1: $J \leftarrow \mathbf{J}_z r$ (Linearization);
2: $\mathcal{L} \leftarrow \{j | z(j) \leq p(j)\}$; $\mathcal{U} \leftarrow \{j | z(j) \geq q(j)\}$;
3: $d \leftarrow J^\top b$ (Compute gradient of the cost function); $\lambda_p(j) \leftarrow d(j), \forall j \in \mathcal{L}$; $\lambda_q(j) \leftarrow -d(j), \forall j \in \mathcal{U}$;
4: **if** $\lambda_p(j) \geq -\gamma, \forall j \in \mathcal{L}$ **and** $\lambda_q(j) \geq -\gamma, \forall j \in \mathcal{U}$ **and** $|d(j)| \leq \gamma, \forall j \notin \mathcal{L} \cup \mathcal{U}$ **then go to** Step 12 (Stop if converged to a first-order optimal point);
5: $\Delta z \leftarrow \arg\min_{p - z \leq \Delta \hat{z} \leq q - z} \|J \Delta \hat{z} + b\|_2^2$ (search direction);
6: $\alpha = 1$; $\theta \leftarrow c\alpha d^\top \Delta z$; $\psi \leftarrow b^\top b$; $b \leftarrow r(z + \Delta z)$; $\Phi \leftarrow b^\top b$;
7: **while** $\Phi > \psi + \theta$ **do** (Backtracking line search)
8: $\alpha \leftarrow \tau\alpha$; $\theta \leftarrow \alpha\theta$;
9: $b \leftarrow r(z + \alpha \Delta z)$; $\Phi \leftarrow b^\top b$;
10: **end while**
11: $z \leftarrow z + \alpha \Delta z$; **go to** Step 1 (Update the iterate);
12: $z^\star \leftarrow z$; $\lambda_p(j) \leftarrow 0, \forall j \notin \mathcal{L}$; $\lambda_q(j) \leftarrow 0, \forall j \notin \mathcal{U}$;
13: **end.**

Outputs: Local minimum $z^\star$ of (5), objective function value Φ at $z^\star$, and Lagrange multiplier vectors λ_p and λ_q corresponding to lower and upper bounds, respectively.

4. Optimization algorithm

4.1. Bounded-variable nonlinear least squares

In order to efficiently solve the MPC problem (6), it is desirable to have a solution method that benefits from warm-starting information, is robust to problem scaling, and exploits the structure of the problem. The bounded-variable nonlinear least-squares (BVNLLS) method we propose in Algorithm 1 addresses such features. It can be seen as either an *ad hoc* primal-feasible line-search SQP algorithm (Nocedal & Wright, 2006) or an extension of the Gauss–Newton method (Björck, 1996, Section 9.2) to handle box-constraints. The Gauss–Newton approximation of the Hessian is effective for nonlinear least-squares cost functions and it only needs first-order information of the residual. Although the global convergence property of Algorithm 1 follows that of line-search methods for problems with simple bounds (Bertsekas, 1996), we provide below an alternative proof specific to BVNLLS for an insightful overview, which also justifies the backtracking rule (Steps 6–10 of Algorithm 1), that is analogous to the *Armijo condition* (Nocedal & Wright, 2006) for the choice of the step-size α. We also note that the overview in this section is presented for completeness whereas there exist other solvers which may solve BVNLS problems (6), see, e.g., Agarwal, Mierle, and The Ceres Solver Team.

4.2. Global convergence

At the ith iteration of Algorithm 1, the search direction $\Delta z^{(i)}$ is computed at Step 5 as

$$\Delta z^{(i)} = \underset{\bar{p} \leq \Delta \hat{z} \leq \bar{q}}{\arg\min} \|J \Delta \hat{z} + b\|_2^2, \tag{7}$$

where the Jacobian $J = \mathbf{J}_z r\left(z^{(i-1)}\right)$ is assumed full rank, $b = r\left(z^{(i-1)}\right)$, $\bar{p} = p - z^{(i-1)}$, $\bar{q} = q - z^{(i-1)}$, and p, q are the bounds on z. Since the bounds of any component of Δz cannot all be active simultaneously, the optimal set of active constraint gradients of (7) are always linearly independent. Hence, linear independence constraint qualification holds. Based on this and the fact that the Hessian matrix $J^\top J > 0$, problem (7) always has a unique set of optimal primal and dual variables.

Lemma 1 (*Primal feasibility*). *Consider $z^{(i)} = z^{(i-1)} + \alpha \Delta z^{(i)}$ as in Step 11 at the ith iteration of Algorithm 1 with any $\alpha \in (0, 1]$. If $p \leq z^{(0)} \leq q$ and $\bar{p} \leq \Delta z^{(i)} \leq \bar{q}$, then $p \leq z^{(i)} \leq q$ at all iterations i.*

Proof. We prove the lemma by induction. The lemma clearly holds for $i = 0$, as by assumption the initial guess z^0 is feasible, $p \leq z^{(0)} \leq q$. Consider the ith iteration of Algorithm 1. From Step 5 we have that $p - z^{(i-1)} \leq \Delta z^{(i)} \leq q - z^{(i-1)}$, which multiplied by α, $\alpha > 0$, gives

$$\alpha p - \alpha z^{(i-1)} \leq \alpha \Delta z^{(i)} \leq \alpha q - \alpha z^{(i-1)}. \tag{8}$$

By adding $z^{(i-1)}$ to each side of the inequalities in (8) we get

$$\alpha p + (1 - \alpha) z^{(i-1)} \leq z^{(i)} \leq \alpha q + (1 - \alpha) z^{(i-1)}. \tag{9}$$

By induction, let us assume that $p \leq z^{(i-1)} \leq q$. Since $\alpha \leq 1$, we further get the inequalities

$$\alpha p + (1 - \alpha) p \leq z^{(i)} \leq \alpha q + (1 - \alpha) q$$

i.e., $p \leq z^{(i)} \leq q$. □

Lemma 2. *The search direction (7) is a descent direction for the cost function $f(z) = \frac{1}{2}\|r(z)\|_2^2$ in (6).*

Proof. If $\mathcal{D}(f(z), \Delta z)$ denotes the directional derivative of $f(z)$ in the direction Δz, then $\Delta z^{(i)}$ is a descent direction if $\mathcal{D}\left(f\left(z^{(i-1)}\right), \Delta z^{(i)}\right) < 0$. By definition of directional derivative (Nocedal & Wright, 2006, Appendix A),

$$\mathcal{D}\left(f\left(z^{(i-1)}\right), \Delta z^{(i)}\right) = \nabla_z f\left(z^{(i-1)}\right)^\top \Delta z^{(i)}. \tag{10}$$

By substituting

$$\nabla_z f\left(z^{(i-1)}\right) = \mathbf{J}_z r\left(z^{(i-1)}\right)^\top r\left(z^{(i-1)}\right) = J^\top b \tag{11}$$

in (10) we get

$$\mathcal{D}\left(f\left(z^{(i-1)}\right), \Delta z^{(i)}\right) = b^\top J \Delta z^{(i)}. \tag{12}$$

Since $\Delta z^{(i)}$ solves the convex subproblem (7), the following Karush–Kuhn–Tucker (KKT) conditions (Borrelli, Bemporad, & Morari, 2017) hold:

$$J^\top\left(J \Delta z^{(i)} + b\right) + \Lambda_{\bar{q}} - \Lambda_{\bar{p}} = \mathbf{0} \tag{13a}$$
$$\Delta z^{(i)} \geq \bar{p} \tag{13b}$$
$$\Delta z^{(i)} \leq \bar{q} \tag{13c}$$
$$\Lambda_{\bar{q}}, \Lambda_{\bar{p}} \geq \mathbf{0} \tag{13d}$$
$$\Lambda_{\bar{q}}(j)\left(\Delta z^{(i)}(j) - \bar{q}(j)\right) = 0 \ \forall j \tag{13e}$$
$$\Lambda_{\bar{p}}(j)\left(\bar{p}(j) - \Delta z^{(i)}(j)\right) = 0 \ \forall j, \tag{13f}$$

where $\Lambda_{\bar{q}}$ and $\Lambda_{\bar{p}}$ denote the optimal Lagrange multipliers of subproblem (7). From (13a) we have,

$$b^\top J \Delta z^{(i)} = \left(\Lambda_{\bar{p}} - \Lambda_{\bar{q}}\right)^\top \Delta z^{(i)} - \Delta z^{(i)\top} J^\top J \Delta z^{(i)}. \tag{14}$$

By substituting $\bar{p} = p - z^{(i-1)}$ and $\bar{q} = q - z^{(i-1)}$ in the complementarity conditions (13e)–(13f), we can write

$$\Lambda_{\bar{q}}^\top\left(\Delta z^{(i)} - q + z^{(i-1)}\right) + \Lambda_{\bar{p}}^\top\left(p - z^{(i-1)} - \Delta z^{(i)}\right) = 0,$$
$$\text{i.e., } \left(\Lambda_{\bar{q}} - \Lambda_{\bar{p}}\right)^\top \Delta z^{(i)} = \Lambda_{\bar{q}}^\top (q - z^{(i-1)}) + \Lambda_{\bar{p}}^\top (z^{(i-1)} - p).$$

From (13b)–(13d) we have $\Lambda_{\bar{q}}, \Lambda_{\bar{p}} \geq \mathbf{0}$, and by Lemma 1 $q - z^{(i-1)} \geq \mathbf{0}$ as well as $z^{(i-1)} - p \geq \mathbf{0}$, which implies that

$$(\Lambda_{\bar{p}} - \Lambda_{\bar{q}})^\top \Delta z^{(i)} \leq \mathbf{0}. \tag{15}$$

Since J is full rank, $J^\top J > 0$. Using this fact and Lemma 4 along with (15) in (14) gives

$$b^\top J \Delta z^{(i)} < 0. \tag{16}$$

Considering (16) and (12), we have that the directional derivative for the considered search direction is negative, which proves the lemma. □

Remark 1. We infer from Lemma 1 and (13b)–(13c) that BVNLLS is a primal-feasible method, which is an important property when the function $r(z)$ is not analytic beyond bounds (Bellavia, Macconi, & Morini, 2003).

Lemma 3. *If the solution of (7) is $\Delta z^{(i)} = \mathbf{0}$, then $z^{(i-1)}$ is a stationary point satisfying the first-order optimality conditions of problem (6).*

Proof. Given $\Delta z^{(i)} = \mathbf{0}$, we need to prove that $z^{(i-1)}$ satisfies the following first-order optimality conditions for problem (6):

$$\mathbf{J}_z r(z)^\top r(z) + \lambda_q - \lambda_p = \mathbf{0} \tag{17a}$$
$$p \leq z \leq q \tag{17b}$$
$$\lambda_q, \lambda_p \geq \mathbf{0} \tag{17c}$$
$$\lambda_q(j)(z(j) - q(j)) = \lambda_p(j)(p(j) - z(j)) = 0, \ \forall j, \tag{17d}$$

where the optimal Lagrange multipliers are denoted by λ_p and λ_q for the lower and upper bounds, respectively.

By substituting $\Delta z^{(i)} = \mathbf{0}$ in (13), and recalling $\bar{q} = q - z^{(i-1)}$ and $\bar{p} = p - z^{(i-1)}$, we obtain

$$J^\top b + \Lambda_{\bar{q}} - \Lambda_{\bar{p}} = \mathbf{0}, \tag{18a}$$
$$p \leq z^{(i-1)} \leq q, \tag{18b}$$
$$\Lambda_{\bar{q}}(j)(z^{(i-1)}(j) - q(j)) = 0 \ \forall j, \tag{18c}$$
$$\Lambda_{\bar{p}}(j)(p(j) - z^{(i-1)}(j)) = 0 \ \forall j. \tag{18d}$$

Clearly, considering (13d) along with the definitions of J, b, and (18), we conclude that $z^{(i-1)}$, $\Lambda_{\bar{q}}$ and $\Lambda_{\bar{p}}$ solve the KKT system (17). □

Lemma 4. *In Algorithm 1, $\Delta z^{(i)} \neq \mathbf{0}$ at any iteration.*

Proof. We prove this lemma by contradiction. Assume that Algorithm 1 reaches an iteration i where Step 5 is executed and returns $\Delta z^{(i)} = \mathbf{0}$. This implies that $z^{(i-1)}$ is a stationary point satisfying the first-order optimality conditions of nonlinear problem (6), as shown in Lemma 3. Then, the termination criterion in Step 4 would end the algorithm without further computations, so that iteration i is never reached, a contradiction. Note that in particular, if the initial guess $z^{(0)}$ is optimal, $\Delta z^{(i)}$ is never computed. □

Theorem 1 (*Global convergence of BVNLLS*)**.** *Consider the optimization problem (6) and define the scalar cost function $f(z) = \frac{1}{2}\|r(z)\|_2^2$. At each iteration i of Algorithm 1, there exists a scalar $\alpha \in (0, 1]$ such that*

$$f\left(z^{(i-1)} + \alpha \Delta z^{(i)}\right) - f\left(z^{(i-1)}\right) < c\alpha \nabla f\left(z^{(i-1)}\right)^\top \Delta z^{(i)} \tag{19}$$

with $0 < \alpha \leq 1$ and $0 < c < 1$, where $z^{(i)} = z^{(i-1)} + \alpha \Delta z^{(i)}$.

Proof. Consider the Taylor series expansion of $f\left(z^{(i)}\right)$

$$f\left(z^{(i-1)} + \alpha \Delta z^{(i)}\right) = f\left(z^{(i-1)}\right) + \alpha \nabla_z f\left(z^{(i-1)}\right)^\top \Delta z^{(i)} + \frac{\alpha^2}{2} \Delta z^{(i)^\top} \nabla_z^2 f\left(z^{(i-1)}\right) \Delta z^{(i)} + \mathcal{E}(\|\alpha \Delta z^{(i)}\|^3), \tag{20}$$

where the term $\mathcal{E}\|\cdot\|^3$ represents third order error. Also,

$$\nabla_z^2 f\left(z^{(i-1)}\right) = \sum_{j=1}^{n_r} r_j\left(z^{(i-1)}\right) \nabla_z^2 r_j\left(z^{(i-1)}\right) + \mathbf{J}_z r\left(z^{(i-1)}\right)^\top \mathbf{J}_z r\left(z^{(i-1)}\right) = H + J^\top J, \tag{21}$$

where r_j denotes the jth element of the residual vector. By substituting the relations (11) and (21) in (20) we get

$$f\left(z^{(i-1)} + \alpha \Delta z^{(i)}\right) - f\left(z^{(i-1)}\right) = \alpha b^\top J \Delta z^{(i)} + \frac{\alpha^2}{2} \Delta z^{(i)^\top} \left(H + J^\top J\right) \Delta z^{(i)} + \mathcal{E}\left(\left\|\alpha \Delta z^{(i)}\right\|^3\right). \tag{22}$$

Using (14), Eq. (22) can be simplified as

$$f\left(z^{(i-1)} + \alpha \Delta z^{(i)}\right) - f\left(z^{(i-1)}\right) = -\frac{\alpha(2-\alpha)}{2} \Delta z^{(i)^\top} J^\top J \Delta z^{(i)} + \alpha(\Lambda_{\bar{p}} - \Lambda_{\bar{q}})^\top \Delta z^{(i)} + \frac{\alpha^2}{2} \Delta z^{(i)^\top} H \Delta z^{(i)} + \mathcal{E}\left(\left\|\alpha \Delta z^{(i)}\right\|^3\right). \tag{23}$$

Referring (11) and (14), on subtracting $c\alpha \nabla f(z^{(i-1)})^\top \Delta z$ from both sides of (23) we get

$$\begin{aligned} f\left(z^{(i-1)} + \alpha \Delta z^{(i)}\right) - f\left(z^{(i-1)}\right) - c\alpha \nabla f\left(z^{(i-1)}\right)^\top \Delta z^{(i)} &= -\frac{\alpha(2-2c-\alpha)}{2} \Delta z^{(i)^\top} J^\top J \Delta z^{(i)} \\ &\quad + \alpha(1-c)(\Lambda_{\bar{p}} - \Lambda_{\bar{q}})^\top \Delta z^{(i)} \\ &\quad + \frac{\alpha^2}{2} \Delta z^{(i)^\top} H \Delta z^{(i)} + \mathcal{E}\left(\left\|\alpha \Delta z^{(i)}\right\|^3\right). \end{aligned} \tag{24}$$

Let

$$\bar{N} = -\tfrac{(2-2c-\alpha)}{2} \Delta z^{(i)^\top} J^\top J \Delta z^{(i)} + (1-c)(\Lambda_{\bar{p}} - \Lambda_{\bar{q}})^\top \Delta z^{(i)}.$$

From (15), Lemma 4, and from the facts that $\alpha \in (0, \ 1]$, $c \in (0, 1)$, and that matrix J has full rank ($J^\top J > 0$), we infer that $\bar{N}$ must be negative for sufficiently small α. Let

$$\bar{M} = \frac{1}{2} \Delta z^{(i)^\top} H \Delta z^{(i)} + \mathcal{E}\left(\left\|\alpha \Delta z^{(i)}\right\|^3\right)$$

Then (24) can be written as

$$f\left(z^{(i-1)} + \alpha \Delta z^{(i)}\right) - f\left(z^{(i-1)}\right) - c\alpha \nabla f\left(z^{(i-1)}\right)^\top \Delta z^{(i)} = \alpha \bar{N} + \alpha^2 \bar{M}. \tag{25}$$

Let $\alpha\bar{N} + \alpha^2\bar{M} + \epsilon = 0$, or $\epsilon = \alpha\left(-\alpha\bar{M} - \bar{N}\right)$. Clearly, since $\bar{N} < 0$, there exists a value of $\alpha > 0$ such that $\epsilon > 0$. This proves that there exists a positive value of α such that $\alpha\bar{N} + \alpha^2\bar{M} < 0$. Hence from (25), $f\left(z^{(i-1)} + \alpha \Delta z^{(i)}\right) - f\left(z^{(i-1)}\right) - c\alpha \nabla f\left(z^{(i-1)}\right)^\top \Delta z^{(i)} < 0$, for a sufficiently small positive value of α. □

Remark 2. In the case of linear MPC i.e., when $h_k(z_k, \ \phi_k)$ is linear in (6), the bounded-variable least-squares (BVLS) problem (7) is solved only once as the KKT conditions (17) coincide with (13). Moreover, the backtracking steps are not required as the higher

order terms in (20) are zero and Theorem 1 holds with $\alpha = 1$ for any $c \in (0,\ 1)$.

Remark 3. Referring to (24), the value of c is practically kept below 0.5 in Algorithm 1 in order to enforce fast convergence with full steps and is typically chosen to be as small as 10^{-4} (Nocedal & Wright, 2006). As seen in (24), since we only need the matrix J to be full rank for convergence of BVNLLS, the matrix W_k in (6) may be rank-deficient as long as J is full rank.

Remark 4. Suboptimality in solving the BVLS subproblems may result in a smaller decrease in the cost between BVNLLS iterations than the theoretical decrease indicated by Theorem 1. Hence, it is essential to have an accurate BVLS solver in order to have fast convergence. For this reason, we advocate the use of active-set methods to solve BVLS problems.

Each iteration of BVNLLS corresponds to solving a linear MPC problem, a special case of (6). This allows one to have a common framework for linear and nonlinear MPC in our approach. The BVLS problem (7) can be solved efficiently and accurately by using a primal active-set algorithm as shown in Saraf and Bemporad (2020), which uses numerically robust recursive QR factorization routines to solve the LS subproblems. Unlike most of the QP solvers, in which the Hessian $J^\top J$ would be constructed via a matrix multiplication and then factorized, the BVLS solver (Saraf & Bemporad, 2020) only factorizes column subsets of J, whose condition number is square-root as compared to that of $J^\top J$, which makes it numerically preferable. In applications with very restrictive memory requirements, using the methods described in Section 5 with the gradient-projection algorithm (Nesterov, 2004) on the primal problem (7), one may employ a matrix-free solver similar to Diamond and Boyd (2016) and its references. However, when using the gradient-projection algorithm, its low memory usage may come at the cost of slow convergence due to their sensitivity to problem scaling. The following sections show how the Jacobian matrix J can be replaced by using parameterized operators for saving memory and how its sparsity can be exploited for faster execution of the proposed BVLS solver of Saraf and Bemporad (2020).

5. Abstracting matrix instances

5.1. Problem structure

The sparse structure of matrices W_k and $\nabla_z h_k(z_k,\ \phi_k)^\top$, which form the Jacobian J of the residual in (6), completely depends on the MPC tuning parameters, model order, and the ordering of the decision variables.

Assume that no slack variables due to non-box inequality constraints have been introduced as in (4). In case of slack variables, the sparsity pattern will depend on the structure of Jacobian of the inequality constraints, which is not discussed in further detail here for conciseness. By ordering the decision variables in vector z_k as follows

$$z_k = \Big[u_k^\top\ y_{k+1}^\top\ u_{k+1}^\top\ y_{k+2}^\top\ \cdots\ u_{k+N_\mathrm{u}-1}^\top\ y_{k+N_\mathrm{u}}^\top \big| \\ y_{k+N_\mathrm{u}+1}^\top\ \cdots\ y_{k+N_\mathrm{p}-1}^\top\ y_{k+N_\mathrm{p}}^\top\Big]^\top \tag{26}$$

we get the matrix structure described in 27, where the superscript of matrices in parentheses denote the output prediction step the matrices refer to. Note that we dropped the parentheses (S_k) in 27 to simplify the notation and, as defined in (2), $A(S_k)_j = \mathbf{0}$, $\forall j > n_\mathrm{a}$, and $B(S_k)_j = \mathbf{0}$, $\forall j > n_\mathrm{b}$. Clearly, the Jacobian matrix $\mathbf{J}h_k$ of equality constraints only consists of entries from the sequence

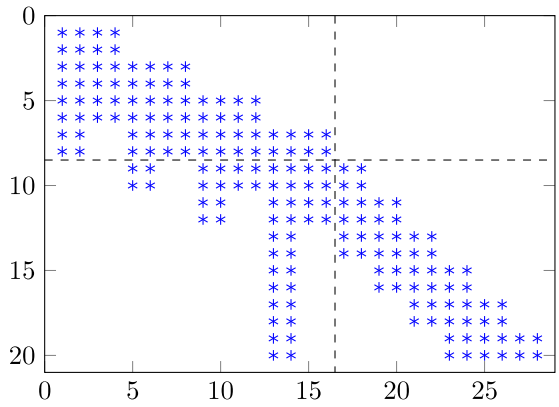

Fig. 1. Sparsity pattern of $\mathbf{J}h_k$ for a random model with $N_\mathrm{p} = 10$, $N_\mathrm{u} = 4$, $n_\mathrm{a} = 2$, $n_\mathrm{b} = 4$, $n_u = 2$ and $n_y = 2$.

of linear models of the form (2) linearized around the initial guess trajectory.

$$\mathbf{J}h_k(z) = \nabla_z h_k(z_k, \phi_k)^\top$$

$$= \left[\begin{array}{cccccc|ccc} B_1^{(1)} & A_0^{(1)} & \mathbf{0} & \mathbf{0} & \cdots & & & \cdots & \mathbf{0} \\ B_2^{(2)} & A_1^{(2)} & B_1^{(2)} & A_0^{(2)} & \mathbf{0} & \cdots & & \cdots & \mathbf{0} \\ \vdots & \ddots & & & & \ddots & & & \vdots \\ B_{N_\mathrm{u}}^{(N_\mathrm{u})} & A_{N_\mathrm{u}-1}^{(N_\mathrm{u})} & & \cdots & & B_1^{(N_\mathrm{u})} & A_0^{(N_\mathrm{u})} & \mathbf{0} \cdots & \mathbf{0} \\ \hline B_{N_\mathrm{u}+1}^{(N_\mathrm{u}+1)} & A_{N_\mathrm{u}}^{(N_\mathrm{u}+1)} & \cdots & & B_3^{(N_\mathrm{u}+1)} & A_2^{(N_\mathrm{u}+1)} & \sum_{i=1}^{2} B_i^{(N_\mathrm{u}+1)} & A_1^{(N_\mathrm{u}+1)} & A_0^{(N_\mathrm{u}+1)} \mathbf{0} \cdots \mathbf{0} \\ B_{N_\mathrm{u}+2}^{(N_\mathrm{u}+2)} & A_{N_\mathrm{u}+1}^{(N_\mathrm{u}+2)} & \ddots & \cdots & B_4^{(N_\mathrm{u}+2)} & A_3^{(N_\mathrm{u}+2)} & \sum_{i=1}^{3} B_i^{(N_\mathrm{u}+2)} & A_2^{(N_\mathrm{u}+2)} & A_1^{(N_\mathrm{u}+2)}\ A_0^{(N_\mathrm{u}+2)}\ \mathbf{0} \cdots \mathbf{0} \\ \vdots & \ddots & \ddots & & \vdots & & \vdots & \vdots & \ddots \mathbf{0} \\ B_{N_\mathrm{p}}^{(N_\mathrm{p})} & A_{N_\mathrm{p}-1}^{(N_\mathrm{p})} & \cdots & & B_{N_\mathrm{p}-N_\mathrm{u}+2}^{(N_\mathrm{p})} & A_{N_\mathrm{p}-N_\mathrm{u}+1}^{(N_\mathrm{p})} & \sum_{i=1}^{N_\mathrm{p}-N_\mathrm{u}+1} B_i^{(N_\mathrm{p})} & A_{N_\mathrm{p}-N_\mathrm{u}}^{(N_\mathrm{p})} & A_{N_\mathrm{p}-N_\mathrm{u}-1}^{(N_\mathrm{p})} \cdots A_1^{(N_\mathrm{p})}\ A_0^{(N_\mathrm{p})} \end{array}\right] \tag{27}$$

Considering the model parameters $n_\mathrm{a}, n_\mathrm{b}$ to be smaller than N_p in 27, as illustrated in Fig. 1, we observe that the top-left part of $\mathbf{J}h_k$ is block sparse, the bottom-right part has a block-banded structure, the bottom-left part has dense columns corresponding to $u_{k+N_\mathrm{u}-1}$, whereas the top-right part is a zero matrix with $n_y N_\mathrm{u}$ rows and $n_y \cdot (N_\mathrm{p} - N_\mathrm{u})$ columns. If n_a, n_b are greater than N_p, then $\mathbf{J}h_k$ would instead have its bottom-left part to be dense with block lower-triangular structure in its top-left and bottom-right parts. All in all, the sparsity pattern of $\mathbf{J}h_k$ is completely defined by the model parameters n_u, n_y, n_a, n_b, and MPC horizons N_u, N_p. Clearly, evaluating $\mathbf{J}h_k$ only requires the sequence of linear models and the sparsity pattern information. Note that in case the linear models are computed by a linearization function, a memory/throughput tradeoff can be chosen here, as they can be either computed once and stored (lowest throughput), or evaluated by the linearization each time they are required (lowest memory allocation). Finally, recalling (6), we obtain the full Jacobian matrix

$$J = \begin{bmatrix} W_k \\ \mathbf{J}h_k \end{bmatrix}$$

required in Algorithm 1, where W_k is the block diagonal matrix

$$W_k = \mathrm{blockdiag}(W_{u_k}, W_{y_{k+1}}, W_{u_{k+1}}, W_{y_{k+2}}, \ldots, \\ W_{u_{k+N_\mathrm{u}-1}}, W_{y_{k+N_\mathrm{u}}}, W_{y_{k+N_\mathrm{u}+1}}, \ldots, W_{y_{k+N_\mathrm{p}}})$$

In the sequel we assume for simplicity that all matrices $W_{u_{(\cdot)}}$, $W_{y_{(\cdot)}}$ are diagonal, so that W_k is actually a diagonal matrix.

Algorithm 2 Operator `Jix`

Inputs: Output memory $v = \mathbf{0} \in \mathbb{R}^{n_z+N_p n_y}$; vector w storing diagonal elements of W_k; scalar x; column number i; parameters n_a, n_b, n_u, n_y, N_u and N_p.

1: $v(i) \leftarrow w(i) \cdot x$;
2: Find integers $\beta \in [0, N_p)$ and $\eta \in [1,\ n_u + n_y]$ such that $i = \beta n_y + n_u \cdot \min(\beta,\ N_u - 1) + \eta$;
3: $\bar{n} \leftarrow N_u n_u + (N_p + \beta) n_y$; $m \leftarrow N_u n_u + 2N_p n_y$; $j \leftarrow 0$;
4: **if** $\beta \neq N_u - 1$ **or** $\eta > n_u$ **then**
5: **if** $\eta > n_u$, $\bar{m} \leftarrow \bar{n} + n_a n_y + n_y$ **else** $\bar{m} \leftarrow \bar{n} + n_b n_y$;
6: **for** $j' \in \{\bar{n}, \bar{n} + n_y, \cdots, \min(\bar{m}, m) - n_y\}$ **do**
7: **if** $\eta > n_u$ **then** $\forall j'' \in \{1, \cdots, n_y\}$,
8: $v(j' + j'') \leftarrow x \cdot A_j^{(\beta+j+1)}(j'',\ \eta - n_u)$;
9: **else**
10: $v(j' + j'') \leftarrow x \cdot B_{j+1}^{(\beta+j+1)}(j'',\ \eta)$;
11: **end if**
12: $j \leftarrow j + 1$;
13: **end for**
14: **else**
15: **for** $j' \in \{\bar{n}, \bar{n} + n_y, \cdots, m - n_y\}$ **do**
16: $j \leftarrow j + 1$;
17: $\bar{B}(j'') \leftarrow \sum_{i'=1}^{\min(j,\ n_b)} B_{i'}^{\beta+j}(j'',\ \eta)$, $\forall j'' \in [1, n_y]$;
18: $v(j' + j'') \leftarrow x \cdot \bar{B}(j'')$, $\forall j'' \in [1, n_y]$;
19: **end for**
20: **end if**
21: **end.**

Output: Vector $v = i$th column of J in (7) scaled by x.

Algorithm 3 Operator `JtiX`

Inputs: Vector w storing diagonal elements of W_k; vector X; column number i; parameters n_a, n_b, n_u, n_y, N_u and N_p.

1: $v' \leftarrow w(i) \cdot X(i)$;
2: Steps 2–3 of Algorithm 2;
3: **if** $\beta \neq N_u - 1$ **or** $\eta > n_u$ **then**
4: **if** $\eta > n_u$, $\bar{m} \leftarrow \bar{n} + n_a n_y + n_y$ **else** $\bar{m} \leftarrow \bar{n} + n_b n_y$;
5: **for** $j' \in \{\bar{n}, \bar{n} + n_y, \cdots, \min(\bar{m}, m) - n_y\}$ **do**
6: **if** $\eta > n_u$ **then** $\forall j'' \in \{1, \cdots, n_y\}$,
7: $v' \leftarrow v' + X(j' + j'') \cdot A_j^{(\beta+j+1)}(j'',\ \eta - n_u)$;
8: **else**
9: $v' \leftarrow v' + X(j' + j'') \cdot B_{j+1}^{(\beta+j+1)}(j'',\ \eta)$;
10: **end if**
11: $j \leftarrow j + 1$;
12: **end for**
13: **else**
14: **for** $j' \in \{\bar{n}, \bar{n} + n_y, \cdots, m - n_y\}$ **do**
15: Steps 16–17 of Algorithm 2;
16: $v' \leftarrow v' + X(j' + j'') \cdot \bar{B}(j'')$, $\forall j'' \in [1, n_y]$;
17: **end for**
18: **end if**
19: **end.**

Output: $v' =$ inner product of ith row of $J^\top$ in (7) and X.

5.2. Abstract operators

All matrix–vector operations involving J in Algorithm 1 and in the BVLS solver (Saraf & Bemporad, 2020), including the matrix factorization routines that will be described in Section 6, only need the product of a column subset of J or a row-subset of $J^\top$ with a vector. Hence, rather than explicitly forming and storing J, all the operations involving J can be represented by two operators `Jix` (ith column of J times a scalar x) and `JtiX` (ith column of J times a vector X) defined by Algorithms 2 and 3, respectively.

The basic principle of both Algorithms 2 and 3 is to extract nonzero entries indexed in J from the corresponding model coefficients based on the given model and MPC tuning parameters. Since the top part W_k of J is a diagonal matrix, the first nonzero entry in any column of J is obtained from the vector of weights (cf. Step 1 of `Jix` and `JtiX`). The remaining steps only concern evaluating $\mathbf{J}h_k$ as in 27, in which the coefficients in each column match the corresponding element in z_k as in (26). Referring to the sparsity pattern of $\mathbf{J}h_k$ in 27, each of its columns only contains either model coefficients related to the input or to the output, and in the columns corresponding to the inputs u_{k+N_u-1} some of the input coefficients are summed due to the finite control horizon $N_u < N_p$. The location of the first nonzero term in each column of $\mathbf{J}h_k$ depends on the corresponding stage of the input or output variable in prediction, whereas the last entry depends on n_a or n_b and N_p. Hence, in Step 2 of Algorithm 2, the integer β is computed such that $\beta n_y + 1$ is the index of the first nonzero entry in $\mathbf{J}h_k(z)$ (cf. Steps 3, 6 and 15). The integer η computed in the same step denotes the input or output channel to which the column corresponds, in order to accordingly index and extract the coefficients to be scaled as shown in Steps 8, 10 and 17 of Algorithm 2. Depending on the column index i of J, computing β and η only needs a trivial number of integer operations including at most one integer division, for instance, if $i \leq N_u(n_u + n_y)$, β is obtained by an integer division of i by $(n_u + n_y)$ and $\eta = i - \beta(n_u + n_y)$. The same computation is straightforward for the only other possible case in which $i > N_u(n_u + n_y)$.

Clearly, since the rows of $J^\top$ are the columns of J, Algorithm 3 differs from Algorithm 2 only in Steps 7, 9 and 16 in which the scaled coefficient is accumulated to the resulting inner product instead of a plain assignment operation. It is possible to easily extend Algorithm 3 for the special case in which X in `JtiX` is the ith column of J i.e., to efficiently compute the 2-norm of the ith column of J, which may be required in the linear algebra routines. Replacing the instances of J by `Jix` and `JtiX` in the BVNLLS and in the inner BVLS solver has the following advantages:

(1) The problem (re-)construction step in MPC is eliminated, as the entire optimization algorithm is parameterized in terms of the specified tuning parameters and the model coefficients, i.e., the output of the linearization step, that would comprise J.

(2) The code of the two operators does not change with any change in the required data or dimensions as all the indexing steps are parameterized in terms of MPC tuning parameters, i.e., known data. Hence, the resulting optimization solver does not need to be code-generated with a change in problem dimensions or data. The same fact also allows real-time changes in the MPC problem data and tuning parameters without any change in the solver. A structural change in the BVNLLS optimization problem formulation, such as the type of performance index, is already decided in the MPC design phase and can be simply accommodated by only modifying Algorithms 2 and 3.

(3) Unlike basic sparse-matrix storage schemes (Saad, 2003) which would store the nonzeros of J along with indexing information, we only store the sequence of linear models at most, resulting in a significantly lower memory requirement. Alternatively, as mentioned earlier, even the coefficients $A_*^{(*)}$, $B_*^{(*)}$ can be generated during the execution of Algorithms 2–3 using linearization functions applied on the current trajectory.

4) The number of floating-point operations (*flops*) involving instances of J, both in the BVNLLS and the BVLS solvers, is minimal and is reduced essentially to what sparse linear algebra routines can achieve, assuming that $A_*^{(*)}$, $B_*^{(*)}$ are not sparse.

(5) A matrix-free implementation can be achieved when using the gradient-projection algorithm (Nesterov, 2004) to solve (7) in BVNLLS, as the operators `Jix` and `JtiX` can be used for computing the gradient. In addition, considering that even the model coefficients are optional to store, the resulting NMPC algorithm will have negligible increase in memory requirement w.r.t. the prediction horizon.

6. Recursive thin QR factorization

The primal active-set method for solving BVLS problems described in Saraf and Bemporad (2020) efficiently solves a sequence of related LS problems using recursive thin QR factorization. The reader is referred to Daniel, Gragg, Kaufman, and Stewart (1976), Golub and Loan (2013), Saraf and Bemporad (2020) for an overview on thin QR factorization and the recursive update routines in the context of the BVLS solver. This section shows how the sparsity of matrix J can be exploited for significantly reducing the computations involved in the recursive updates of its QR factors, without the use of sparse-matrix storage or conventional sparse linear algebra routines. The main idea is to have the location of nonzeros in the matrix factors expressed in terms of model and MPC tuning parameters, as described above. We first analyze how the sparse structure of column subsets of J is reflected in their thin QR factors based on Gram–Schmidt orthogonalization, then characterize the recursive update routines.

6.1. Gram–Schmidt orthogonalization

Recall that $J \in \mathbb{R}^{m\times n}$, where $n = N_{\mathrm{u}}n_u + N_{\mathrm{p}}n_y$ and $m = n + N_{\mathrm{p}}n_y$, i.e., $m > n$ (see (6), (7), 26). Let $J_{\mathcal{F}}$ denote the matrix formed from those columns of J with indices in the set $\mathcal{F}$. Then there exists a unique thin QR factorization (Golub & Loan, 2013, Theorem 5.2.3) of $J_{\mathcal{F}}$ which may be expressed via the Gram–Schmidt orthonormalization procedure $\forall i \in [1, |\mathcal{F}|]$ as

$$Q_i' = J_{\mathcal{F}_i} - \sum_{j=1}^{i-1} Q_j Q_j^\top J_{\mathcal{F}_i}, \tag{28a}$$

$$Q_i = Q_i' / \left\|Q_i'\right\|_2, \tag{28b}$$

$$R(j,i) = Q_j^\top J_{\mathcal{F}_i}, \forall j \in [1, i-1], \tag{28c}$$

$$R(i,i) = \left\|Q_i'\right\|_2, \tag{28d}$$

where $Q \in \mathbb{R}^{m\times|\mathcal{F}|} := [Q_1, Q_2, \ldots, Q_{|\mathcal{F}|}]$ has orthonormal columns, $R \in \mathbb{R}^{|\mathcal{F}|\times|\mathcal{F}|}$ is upper triangular and $J_{\mathcal{F}} = QR$. In (28), with a slight abuse of notation, the subscripts denote column number, i.e., Q_i denotes the ith column of Q, whereas $\mathcal{F}_i$ denotes the ith index in $\mathcal{F}$. As shown in (28a), starting from the first column of $J_{\mathcal{F}}$, the procedure constructs an orthogonal basis by sequentially orthogonalizing the subsequent columns w.r.t. the basis. The orthogonalization procedure shown in (28a) is referred to as the classical Gram–Schmidt (CGS) method (Golub & Loan, 2013, Section 5.2.7). Since the CGS method is practically prone to numerical cancellation due to finite-precision arithmetic, we use the modified Gram–Schmidt (MGS) method (Golub & Loan, 2013, Section 5.2.8) in which the orthogonalization is performed using the working value of Q_i' instead of $J_{\mathcal{F}_i}$ in each iteration of the procedure. When applying MGS to solve the linear system before recursive updates, we also orthogonalize the right hand side (RHS) of the equations, i.e., we use an augmented system of equations in order to compensate the orthogonalization error (cf. Trefethen and Bau (1997, Chapter 19)). Moreover, for numerical robustness in limited precision, in the proposed MGS procedure a reorthogonalization step is automatically performed which iteratively refines the QR factors for reducing the orthogonalization error in case it exceeds a given threshold (cf. Saraf & Bemporad, 2020, Algorithm 2, Daniel et al., 1976).

6.2. Sparsity analysis

In order to avoid redundant *flops* due to multiplying zero entries while solving the LS problems without sparse storage schemes, we first determine the sparsity pattern of Q and R approximately, based on the relations described in (28). While doing so, the following notions will be used.

Definition 1 (*Nonzero structure*)**.** We define the nonzero structure of a vector x to be the set of indices $\mathcal{S}(x)$ such that $x(i) \neq 0$, $\forall i \in \mathcal{S}(x)$, and $x(j) = 0$, $\forall j \notin \mathcal{S}(x)$.

Definition 2 (*Predicted nonzero structure*)**.** If $\hat{\mathcal{S}}(x)$ denotes the predicted nonzero structure of a vector x, then $x(j) = 0\ \forall j \notin \hat{\mathcal{S}}(x)$ i.e., $\hat{\mathcal{S}}(x) \supseteq \mathcal{S}(x)$.

Based on Definition 1, $x = x'$ implies

$$\mathcal{S}(x) = \mathcal{S}(x'). \tag{29}$$

$$\mathcal{S}(x' + x'') \subseteq \left\{\mathcal{S}(x') \cup \mathcal{S}(x'')\right\}, \tag{30}$$

which holds with equality, i.e., $\mathcal{S}(x'+x'') = \left\{\mathcal{S}(x') \cup \mathcal{S}(x'')\right\}$, if and only if the set

$\{i | x'(i) + x''(i) = 0, x'(i) \neq 0, x''(i) \neq 0\} = \emptyset$. Likewise,

$$\mathcal{S}(\kappa x) \subseteq \mathcal{S}(x), \kappa \in \mathbb{R},$$

because $\mathcal{S}(\kappa x) = \emptyset$ for $\kappa = 0$, whereas

$$\mathcal{S}(\kappa x) = \mathcal{S}(x), \forall \kappa \in \mathbb{R} \setminus \{0\}. \tag{31}$$

Theorem 2. *Consider an arbitrary sparse matrix $M \in \mathbb{R}^{n_1\times n_2}$ of full rank such that $n_1 \geq n_2$ and let $\tilde{Q}$ denote the Q-factor from its thin QR factorization i.e., $M = \tilde{Q}\tilde{R}$. The nonzero structure of each column $\tilde{Q}_i$ of $\tilde{Q}$ satisfies*

$$\mathcal{S}\left(\tilde{Q}_i\right) \subseteq \bigcup_{j=1}^{i} \mathcal{S}\left(M_j\right), \forall i \in [1, n_2], \tag{32a}$$

and $$\mathcal{S}\left(\tilde{Q}_1\right) = \mathcal{S}\left(M_1\right). \tag{32b}$$

Proof. We consider the Gram–Schmidt orthogonalization procedure described in (28) applied to M with $\mathcal{F} = [1, n_2]$ (this simplifies the notation, i.e., $\mathcal{F}_i = i$). Referring to (28b), since $\tilde{Q}'$ represents an orthogonal basis of the full rank matrix M with real numbers, $1/\left\|\tilde{Q}_i'\right\| \neq 0\ \forall i$, and hence from (31),

$$\mathcal{S}\left(\tilde{Q}_i\right) = \mathcal{S}\left(\tilde{Q}_i'\right), \forall i. \tag{33}$$

From (28a),

$$\tilde{Q}_1' = M_1. \tag{34}$$

Thus, considering (34) with (29) and (33) proves (32b). Again, considering (28a) with (29) and (33),

$$\mathcal{S}\left(\tilde{Q}_i\right) = \mathcal{S}\left(M_i - \sum_{j=1}^{i-1} \tilde{Q}_j \tilde{Q}_j^\top M_i\right) = \mathcal{S}\left(M_i + \sum_{j=1}^{i-1} \tilde{Q}_j \kappa_j\right) \tag{35}$$

where $\kappa_j = -\tilde{Q}_j^\top M_i \in \mathbb{R}$, $\forall j \in [1, i-1]$, as κ_j represents the result of an inner product of two real vectors. From (30) and (35),

$$\mathcal{S}\left(\tilde{Q}_i\right) \subseteq \left\{ \mathcal{S}\left(M_i\right) \cup \left\{ \bigcup_{j=1}^{i-1} \mathcal{S}\left(\tilde{Q}_j\right) \right\} \right\}. \tag{36}$$

Applying (36) recursively,

$$\mathcal{S}\left(\tilde{Q}_i\right) \subseteq \left\{ \left\{ \bigcup_{j=2}^{i} \mathcal{S}\left(M_j\right) \right\} \cup \mathcal{S}\left(\tilde{Q}_1\right) \right\}. \tag{37}$$

Thus, substituting (32b) in (37) completes the proof. □

Corollary 1. *Given $i \in [1, n_2]$ and $j' \in [1, n_2]$,*

$$\text{if } \left\{ \bigcup_{j=1}^{j'} \mathcal{S}\left(M_j\right) \right\} \cap \mathcal{S}\left(M_i\right) = \emptyset, \text{ then } \tilde{R}(j,i) = 0 \ \forall j \in [1, j']. \tag{38}$$

Proof. Based on result (32a) of Theorem 2, we can say that $\bigcup_{j=1}^{i} \mathcal{S}\left(M_j\right)$ is a predicted nonzero structure of $\tilde{Q}_i$ i.e.,

$$\bigcup_{j=1}^{i} \mathcal{S}\left(M_j\right) = \hat{\mathcal{S}}\left(\tilde{Q}_i\right), \tag{39}$$

and hence

$$\hat{\mathcal{S}}\left(\tilde{Q}_i\right) = \mathcal{S}\left(M_i\right) \cup \hat{\mathcal{S}}\left(\tilde{Q}_{i-1}\right), \forall i \in [1, n_2]. \tag{40}$$

If $\mathcal{S}\left(\tilde{Q}_j\right) \cap \mathcal{S}\left(M_i\right) = \emptyset$, then $\tilde{Q}_j$ and M_i have disjoint nonzero structures and hence, referring to (28c),

$$\mathcal{S}\left(\tilde{Q}_j\right) \cap \mathcal{S}\left(M_i\right) = \emptyset \implies R(j,i) = \tilde{Q}_j^\top M_i = 0. \tag{41}$$

From (40) we have that

$$\hat{\mathcal{S}}\left(\tilde{Q}_i\right) \supseteq \hat{\mathcal{S}}\left(\tilde{Q}_{i'}\right), \forall i' < i. \tag{42}$$

From (39), (42) and Definition 2, i.e., $\hat{\mathcal{S}}\left(\tilde{Q}_i\right) \supseteq \mathcal{S}\left(\tilde{Q}_i\right)$, it follows that $\left\{\bigcup_{j=1}^{j'} \mathcal{S}\left(M_j\right)\right\} \cap \mathcal{S}\left(M_i\right) = \emptyset$ implies $\hat{\mathcal{S}}\left(\tilde{Q}_j\right) \cap \mathcal{S}\left(M_i\right) = \emptyset$, $\forall j < j'$. The corollary result is then immediate given (41). □

Theorem 2 and Corollary 1 establish rigorous upper bounds on the nonzero structure of the QR factors based on the nonzero structure of the factorized matrix.

Since the nonzero structure of $J_{\mathcal{F}}$ is completely determined in terms of model and tuning parameters as shown in Section 5.2, the predicted nonzero structure of its QR factors consequently depends only on them, as will be shown in the remaining part of this section.

Corollary 2. *Consider the matrix $J \in \mathbb{R}^{m \times n}$ whose first n rows form a diagonal matrix and the last $m - n$ rows contain $\mathbf{J}h_k(z)$ as shown in 27. Let $J_{\mathcal{F}}$ denote the matrix formed from the columns of J indexed in the index set $\mathcal{F}$ such that $\mathcal{F}_{i+1} > \mathcal{F}_i$, $\forall i \in [1, |\mathcal{F}|]$. If $Q \in \mathbb{R}^{m \times |\mathcal{F}|}$ denotes the Q-factor from the thin QR factorization of $J_{\mathcal{F}}$, then $\forall i \in [2, |\mathcal{F}|]$, $\left\{\bigcup_{j=1}^{i} \left\{\mathcal{F}_j\right\}\right\} \cup \left(\bar{n}_{\mathcal{F}_1}, \max\left(\mathcal{B}_{i-1}, \min\left(\bar{m}_{\mathcal{F}_i}, m\right)\right)\right] = \hat{\mathcal{S}}\left(Q_i\right)$, where the positive integers $\bar{n}_{j'}$, $\bar{m}_{j'}$ respectively denote the values of $\bar{n}$, $\bar{m}$ computed in Steps 2–5 of Algorithm 2 for j'th column of J, and $\mathcal{B}$ is an index set such that its ith element stores the largest index of $\hat{\mathcal{S}}\left(Q_i\right)$.*

Proof. Considering the structure of matrix J, Definition 1 and the fact that $\min\left(\bar{m}_j, m\right) > \bar{n}_j \geq n \geq |\mathcal{F}|$, $\forall j$ by construction, we have that

$$\mathcal{S}\left(J_{\mathcal{F}_i}\right) = \{\mathcal{F}_i\} \cup \left(\bar{n}_{\mathcal{F}_i}, \min\left(\bar{m}_{\mathcal{F}_i}, m\right)\right]. \tag{43}$$

From (39) we note that $\bigcup_{j=1}^{i} \mathcal{S}\left(J_{\mathcal{F}_j}\right) = \hat{\mathcal{S}}\left(Q_i\right)$, and using (43) we can rewrite

$$\begin{aligned} \hat{\mathcal{S}}\left(Q_i\right) &= \bigcup_{j=1}^{i} \mathcal{S}\left(J_{\mathcal{F}_j}\right) \\ &= \left\{ \bigcup_{j=1}^{i} \left\{\mathcal{F}_j\right\} \right\} \cup \left\{ \bigcup_{j=1}^{i} \left(\bar{n}_{\mathcal{F}_j}, \min\left(\bar{m}_{\mathcal{F}_j}, m\right)\right] \right\}, \\ &= \left\{ \bigcup_{j=1}^{i} \left\{\mathcal{F}_j\right\} \right\} \cup \left(\bar{n}_{\mathcal{F}_1}, \mathcal{B}_i\right], \end{aligned} \tag{44}$$

because observing (27), $\mathcal{F}_{j+1} > \mathcal{F}_j$ implies $\bar{n}_{\mathcal{F}_j} \leq \bar{n}_{\mathcal{F}_{j+1}}$. From result (40), (43) and definition of set $\mathcal{B}$,

$$\mathcal{B}_i = \max\left(\mathcal{B}_{i-1}, \min\left(\bar{m}_{\mathcal{F}_i}, m\right)\right), \tag{45}$$

which on substitution in (44) completes the proof. □

Note that from (43) and result (32b),

$$\mathcal{S}\left(Q_1\right) = \mathcal{S}\left(J_{\mathcal{F}_1}\right) = \left\{\mathcal{F}_1, \left(\bar{n}_{\mathcal{F}_1}, \min\left(\bar{m}_{\mathcal{F}_1}, m\right)\right]\right\}. \tag{46}$$

By definition of set $\mathcal{B}$ we have $\mathcal{B}_1 = \min\left(\bar{m}_{\mathcal{F}_1}, m\right)$ from (46), and hence $\mathcal{B}_i$ can be determined $\forall i$ by using (45).

Corollary 3. *$Q(i,j) = 0$ $\forall i \in [1, n] \setminus \mathcal{F}$, $\forall j \in [1, |\mathcal{F}|]$. Also, $\forall j' \in [1, |\mathcal{F}|]$, $Q(i,j) = 0$ $\forall j \in [1, j')$ such that $i = \mathcal{F}_{j'}$.*

Proof. Let Q'' denote the submatrix formed from the first n rows of Q. Since $\bar{n}_{\mathcal{F}_1} > n$, from Corollary 2 we can write $\bigcup_{j=1}^{i} \left\{\mathcal{F}_j\right\} = \hat{\mathcal{S}}\left(Q_i''\right)$. Thus, referring this relation and Definition 2, if an index is not in the set $\mathcal{F}$, the corresponding row of Q'' and hence Q has no nonzero element. The latter part is proved by (40) considering the facts that J is diagonal and $\mathcal{F}_{i+1} > \mathcal{F}_i$. □

From Corollaries 2 and 3 we infer that the nonzero structure of all the $|\mathcal{F}|$ columns of Q can be stored using a scalar for $\bar{n}_{\mathcal{F}_1}$ and two integer vectors of dimension $|\mathcal{F}|$ containing the index sets $\mathcal{F}$ and $\mathcal{B}$, where $\mathcal{B}_i = \max\left(\min\left(\bar{m}_i, m\right), \bar{m}_{i-1}\right)$. In order to only compute the nonzeros of R, while constructing each of its column, we need to find and store a scalar j' as shown in Corollary 1. This is done by using the relations described in Theorem 2, Corollary 1 and (44). Specifically, when computing the ith column of R ($i > 1$), j' is found by counting the number of times $B_j < \bar{n}_{\mathcal{F}_i}$ for increasing values of $j \in \left(\hat{j}, i\right)$ until the condition is not satisfied, where $\hat{j}$ denotes the value of j' for the $(i-1)$th column of R.

6.3. Recursive updates

In the primal active-set method, a change in the active-set corresponds to an index inserted in or deleted from the set $\mathcal{F}$. We exploit the uniqueness of thin QR factorization in order to update the structure indicating sets $\mathcal{F}$ and $\mathcal{B}$. When an index t is inserted in the active-set of bounds, the last column of Q and the ith column of R are deleted such that $t = \mathcal{F}_i$, and the QR factorization is updated by applying Givens rotations that triangularize R. In this case $\mathcal{F}$ is simply updated to $\mathcal{F}' = \mathcal{F} \setminus \{t\}$ and $\mathcal{B}$ is updated such that (45) is satisfied after removing its ith index. Moreover, using Corollary 3, the Givens rotations are not applied on the tth row of Q which is simply zeroed. On the other hand, when an index t is removed from the active-set of bounds, $\mathcal{F}$ is updated to $\mathcal{F} \cup \{t\}$ such that $\mathcal{F}_{j+1} > \mathcal{F}_j$, $\forall j$. If t is inserted in $\mathcal{F}$ in the jth position, an index is inserted in the jth

position of $\mathcal{B}$ using (45) and the elements with position greater than j are updated to satisfy (45). Since the sparse structure of the updated QR factors is known during recursive updates, using $\mathcal{F}$, $\mathcal{B}$ and Corollary 3, the *flops* for applying Givens rotations on rows of Q and matrix–vector multiplications in the Gram–Schmidt (re)orthogonalization procedure are performed only on nonzero elements. This makes the QR update routines significantly faster as is reflected in the numerical results described in Section 7.

6.4. Advantages and limitations

The predicted nonzero structure of the Q-factor via (40) is exact if and only if the set relation (32a) holds with equality. For (32a) to hold with equality for Q, $Q_j^\top J_{\mathcal{F}_i}$ must be nonzero for all pairs of indices i and j referring the CGS orthogonalization in (28a) and moreover the summation of nonzeros in the RHS of (28a) must result in a nonzero . Even though theoretically this may not be the case for the matrices that we consider, due to finite precision computations which disallow perfect orthogonality, and the use of MGS with potentially multiple orthogonalizations to compute columns of Q, the predicted nonzero structure of columns of Q via Corollary 2 rarely contains indices of zero elements, i.e., numerically it is an accurate estimate and often the exact nonzero structure. Referring to Corollary 1 and Saraf and Bemporad (2020, Algorithm 2), the same fact leads to the conclusion that if multiple orthogonalizations (for numerical robustness) are performed, in the worst case, the upper-triangular part of the R factor may have no zero elements. Nevertheless, the initial sparsity in R before reorthogonalization is still exploited in its construction but the worst-case fill-in makes it necessary to use R as a *dense* upper-triangular matrix when solving the triangular system by back-substitution to compute the solution of the underlying LS problem.

From Theorem 2, we observe that the predicted nonzero structure of columns $Q_j, \forall j \geq i$, would contain at least the indices of nonzero elements in the ith column of $J_{\mathcal{F}}$. Hence, in case $N_\mathrm{u} < N_\mathrm{p}$, referring the analysis in Section 6.2, the fill-in of Q can be reduced by a re-ordering of the decision variable vector in (26) such that the columns of J corresponding to the variables $u_{k+N_\mathrm{u}-1}$ are moved to become its last columns. Note that even though this re-ordering does not optimize the fill-in of Q, for which dedicated routines exist in literature (cf. Davis (2006)), it still allows a relatively simple and a computationally effective implementation of recursive thin QR factorization for the matrix of interest through a straightforward extension of the methods described in Section 6.3.

In order to benefit computationally by exploiting sparsity while performing the recursive updates, the thin QR factors are stored in numerically dense format. This causes greater memory requirement beyond a certain large problem size where a sparse-storage scheme would need smaller memory considering that with conventional sparse linear algebra, one would only compute and store the R factor while always solving the LS problem from scratch instead. However, the latter approach could turn out to be computationally much more expensive. Using the techniques discussed in Sections 6.2 and 6.3 with a sparse-storage scheme could address this limitation specific to large-scale problems for memory-constrained applications but it needs a much more intricate implementation with cumbersome indexing, that is beyond the scope of this paper.

7. Numerical results

7.1. Software framework

In order to implement the (nonlinear) MPC controller based on formulation (6) or (A.2) one only needs the code for Algorithm 1. The inner BVLS solver of Saraf and Bemporad (2020) could be replaced by another algorithm that exploits sparsity via the abstract operators, such as the gradient-projection algorithm of Nesterov (2004) we mentioned earlier. Besides, routines that evaluate the model (1) and the Jacobian matrices, i.e., the model coefficients in (2), are required from the user in order to evaluate the residual and perform the linearization step (or alternatively finite-differences) in BVNLLS. Note that an optimized self-contained code for these routines can easily be generated or derived by using symbolic tools such as those of MATLAB or the excellent open-source software `CasADi` (Andersson, Gillis, Horn, Rawlings, & Diehl, 2019). This means that, except for the user-defined model and tuning parameters, the software does not need any code-generation, as for a given class of performance indices the code for Algorithms 1–3 does not change with the application. The user is only required to provide the MPC tuning parameters and a symbolic expression for the model (1), which eases the deployment of the proposed MPC solution algorithm in embedded control hardware.

7.2. Computational performance

The results presented in this section are based on a library-free C implementation of BVNLLS based on Algorithms 2 and 3, and the BVLS solver based on the recursive thin QR factorization routines discussed in Section 6. The reader is referred to Saraf et al. (2018, Section 5) for complete details on simulation settings, tuning parameters, constraints, and benchmark solvers related to the following discussion on the example problem, which consists of NMPC applied to a CSTR (continuous stirred tank reactor). All the optimization problems in the MATLAB simulations referred below were solved until convergence,[1] on a Macbook Pro 2013 equipped with 8 GB RAM and 2.6 GHz Intel Core i5 processor. The sparse NLP solver `IPOPT` (Wächter & Biegler, 2006) referred below was compiled in MATLAB with the `MA57` linear system solver and was provided with exact sparse Jacobian evaluation functions including sparsity patterns, and an initial guess for the primal and dual variables in order to incorporate all benefits of the tool. The `fmincon` SQP solver of MATLAB was also provided with gradient evaluation functions and warmstarts for faster convergence. Fig. 2 illustrates the specific simulation scenario for which the execution time of the solvers were compared for increasing values of the prediction horizon. Fig. 3 shows that the equality constraints were satisfied with sufficient accuracy by BVNLLS. Hence, as also demonstrated in Saraf et al. (2018, Section 5, Figure 2), all the solvers including the proposed one yield the same control performance.

Fig. 4 shows that the proposed methods (*custom-sparse*) allow BVNLLS to outperform its *dense* variant even on small-sized test problems by almost an order of magnitude on average. For comparison, `fmincon` and `IPOPT` are applied to the benchmark formulation (5). The computation time of BVNLLS is about two orders of magnitude smaller on the considered small-sized test problems. This reduction can be credited to the fact that `IPOPT`, which is based on sparse linear algebra routines, is more effective for large-sized problems, and that BVNLLS exploits warmstarts based on the previously computed solution, which is provided from the second instance onwards. The main reason for comparing to `fmincon` and `IPOPT` is that they are widely used and therefore provide a baseline that is easy to reproduce. Regarding comparisons with other embedded nonlinear MPC solvers, we refer the reader to the recent paper (Verschueren et al., 2021),

[1] The optimality and feasibility tolerances for all solvers are 10^{-6} and 10^{-8}, respectively, to achieve the same quality of solution at convergence for a fair comparison.

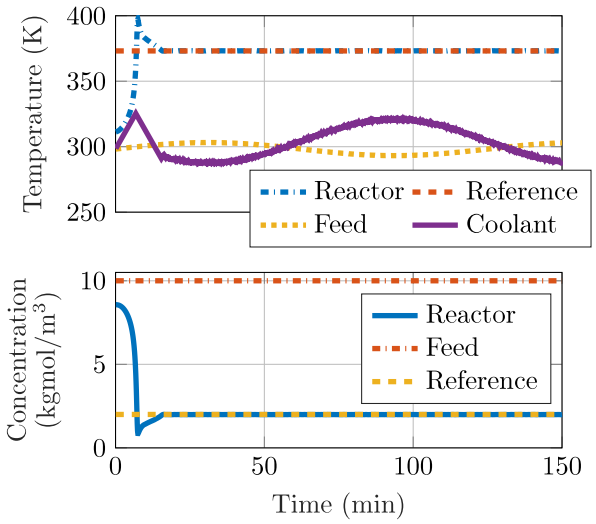

Fig. 2. Closed-loop NMPC simulation trajectories of CSTR variables with $N_p = N_u = 160$ (16 min).

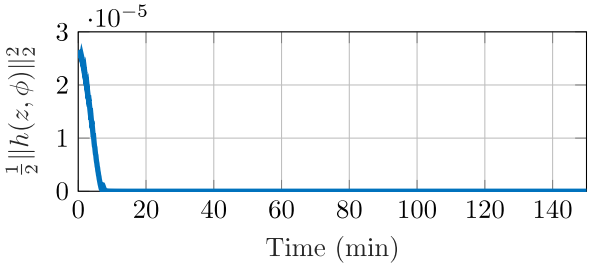

Fig. 3. Worst-case equality constraint residuals ($N_p = N_u = 160$) for BVNLLS with $\sqrt{\rho} = 10^4$ during the simulation described by Fig. 2.

in which thorough comparisons between IPOPT and other state-of-the-art-solvers are reported. While computation performance clearly depends on the particular control problem solved, the time-ratio with respect to IPOPT is a good indicator for comparing different methods. Finally, Fig. 5 suggests that, despite being based on an active-set algorithm, the proposed sparsity-exploiting methods empower BVNLLS to perform significantly faster than the baseline fmincon/IPOPT solvers even for large problems.

8. Conclusions

This paper has presented a new approach to solving constrained linear and nonlinear MPC problems that, by relaxing the equality constraints generated by the prediction model into quadratic penalties, allows the use of a very efficient bounded-variable nonlinear least squares solver. The linear algebra behind the latter has been specialized in detail to take into account the particular structure of the MPC problem, so that the resulting required memory footprint and throughput are minimized for efficient real-time implementation, without the need of any external advanced linear algebra library.

Acknowledgments

The authors thank Laura Ferrarotti and Mario Zanon (IMT Lucca) and Stefania Bellavia (University of Florence) for stimulating discussions concerning the convergence of BVNLLS.

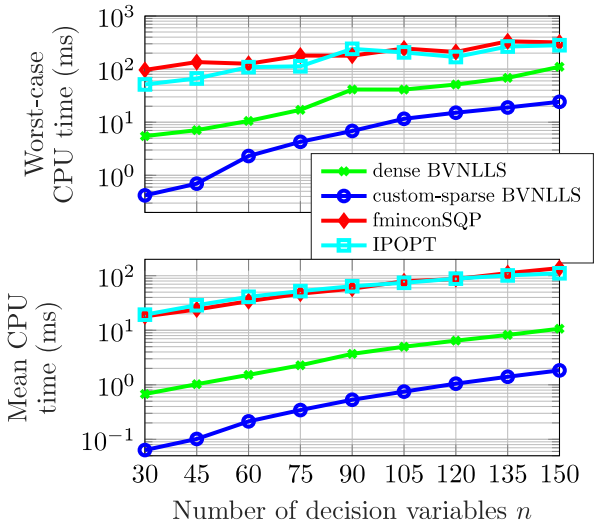

Fig. 4. Computation time spent by each solver during NMPC simulation of CSTR (Saraf et al., 2018) for increasing values of $N_p = N_u = n/3$, n set of box-constraints and $2N_p$ equality constraints.

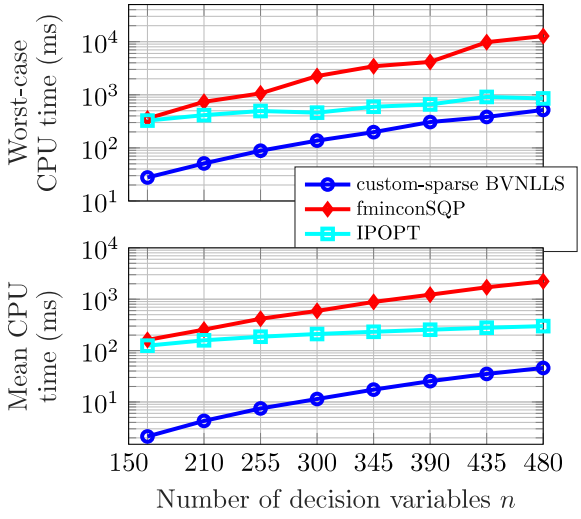

Fig. 5. Computation time spent by each solver during NMPC simulation of CSTR for large values of $N_p = N_u = n/3$ with n set of box-constraints and $2N_p$ equality constraints.

Appendix. Bound-constrained Lagrangian method

This section compares the Bound-constrained Lagrangian method (BLM) with the quadratic penalty method (QPM)) discussed in Section 3 and shows that the same tools discussed in the paper can be applied to solve problems formulated as they are in BLM. At each iteration (i) of the BLM, one solves

$$z_k^{(i+1)} = \arg\min_{p_k \le z_k \le q_k} \frac{1}{2} \left\| \begin{matrix} W_k(z_k - \bar{z}_k) \\ \sqrt{\rho^{(i)}} h_k(z_k,\ \phi_k) \end{matrix} \right\|_2^2 + \Lambda_k^{\top(i)} h_k(z_k, \phi_k) \tag{A.1}$$

where Λ denotes the vector of Lagrange multipliers corresponding to the equality constraints, and updates the estimates $\Lambda^{(i)}$ and $\rho^{(i)}$, until convergence (cf. Nocedal and Wright (2006, Algorithm 17.4)).

Proposition 1. *The optimization problem (A.1) is equivalent to the BVNLS problem*

$$z_k^{(i+1)} = \underset{p_k \le z_k \le q_k}{\arg\min} \frac{1}{2} \left\| \begin{matrix} \frac{1}{\sqrt{\rho^{(i)}}} W_k(z_k - \bar{z}_k) \\ h_k(z_k,\ \phi_k) + \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \end{matrix} \right\|_2^2 \tag{A.2}$$

Proof. We have that problem

$$\underset{p_k \le z_k \le q_k}{\arg\min} \frac{1}{2} \left\| \begin{matrix} W_k(z_k - \bar{z}_k) \\ \sqrt{\rho^{(i)}} h_k(z_k,\ \phi_k) \end{matrix} \right\|_2^2 + \Lambda_k^{\top(i)} h_k(z_k, \phi_k)$$

and $\underset{p_k \le z_k \le q_k}{\arg\min} \frac{1}{2} \|W_k(z_k - \bar{z}_k)\|_2^2 + \mathcal{H}(z_k)$, where $\mathcal{H}(z_k)$

$$\begin{aligned} &= \frac{\rho^{(i)}}{2} \|h_k(z_k,\ \phi_k)\|_2^2 + \Lambda_k^{\top(i)} h_k(z_k, \phi_k) + \frac{\left\|\Lambda_k^{(i)}\right\|_2^2}{2\rho^{(i)}} \\ &= \frac{\rho^{(i)}}{2} \left(\|h_k(z_k,\ \phi_k)\|_2^2 + \frac{2\Lambda_k^{\top(i)} h_k(z_k, \phi_k)}{\rho^{(i)}} + \left\| \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \right\|_2^2 \right) \\ &= \frac{\rho^{(i)}}{2} \left(h_k(z_k,\ \phi_k) + \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \right)^\top \left(h_k(z_k,\ \phi_k) + \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \right) \\ &= \frac{\rho^{(i)}}{2} \left\| h_k(z_k,\ \phi_k) + \frac{\Lambda_k^{(i)}}{\rho^{(i)}} \right\|_2^2, \text{ are equivalent.} \end{aligned}$$

Scaling by the constant $1/\rho^{(i)}$ yields the result. □

Remark 5. Proposition 1 holds for any sum-of-squares cost function with (5a) as the special case, for instance $\|\mathcal{S}(z_k)\|_2^2$, where $\mathcal{S}$ is any vector-valued function.

Proposition 1 shows that we can employ the same BVNLS solvers to solve (A.1), which may be more efficient and numerically robust (cf. Section 4) as compared to the use of other NLP solvers. When using BLM, sequences of $z_k^{(i)}$ and $\Lambda_k^{(i)}$ respectively converge to their optimal values $z_k^\star$ and $\Lambda_k^\star$ whereas $h_k(z_k^\star, \phi_k) \approx \mathbf{0}$, numerically. Then via Proposition 1, we note that for a fixed value of $\rho \gg \left\|\Lambda_k^\star\right\|_\infty$ in the equality-constrained case, we obtain $h_k\left(z_k^{(i+1)}, \phi_k\right) \approx \Lambda_k^{(i+1)}/\rho \approx \mathbf{0}$ (Nocedal & Wright, 2006, Chapter 17), which is simply the solution obtained using a single iteration of QPM for the same ρ and is consistent with the special case described by Saraf and Bemporad (2017, Theorem 1).

Although with BLM it is possible to initialize ρ to arbitrarily low values and solve numerically easier problems, which is its main advantage over QPM, the final value of ρ is not guaranteed to remain low. A main disadvantage of BLM over QPM is that it needs problem (5) to be feasible, otherwise the problem must be formulated with *soft constraints* on output variables (Kerrigan & Maciejowski, 2000), which typically results in the use of penalty functions with large values of the penalty parameter and non-box inequality constraints, making the problems relatively more difficult to solve. Moreover, even if the feasibility of (5) is given, it may take significantly longer to solve multiple instances of (A.1) as compared to a single iteration of QPM with a large penalty, which is more suitable for MPC problems where slight suboptimality may be preferable to a longer computation time. However, in the presence of *hard* general (nonlinear) inequality constraints where QPM might not be applicable, using BLM for feasible problems with the proposed solver and sparsity exploiting methods described in the following sections may be an efficient alternative. BLM is not discussed further as the scope of this paper is limited to MPC problems with box constraints on decision variables.

References

Agarwal, S., Mierle, K., & The Ceres Solver Team Ceres Solver, http://ceres-solver.org.

Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., & Diehl, M. (2019). CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, *11*(1), 1–36.

Bellavia, S., Macconi, M., & Morini, B. (2003). An affine scaling trust-region approach to bound-constrained nonlinear systems. *Applied Numerical Mathematics*, *44*(3), 257–280.

Bemporad, A., Bernardini, D., Long, R., & Verdejo, J. (2018). Model predictive control of turbocharged gasoline engines for mass production. In *WCX™: SAE world congress experience*. Detroit, MI, USA.

Bertsekas, D. (1996). *Constrained optimization and lagrange multiplier methods*. Athena Scientific.

Björck, Å. (1996). *Numerical methods for least squares problems*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Borrelli, F., Bemporad, A., & Morari, M. (2017). *Predictive control for linear and hybrid systems*. Cambridge University Press.

Cannon, M. (2004). Efficient nonlinear model predictive control algorithms. *Annual Reviews in Control*, *28*(2), 229–237.

Cavanini, L., Cimini, G., & Ippoliti, G. (2018). Computationally efficient model predictive control for a class of linear parameter-varying systems. *IET Control Theory & Applications*, *12*(10), 1384–1392.

Daniel, J. W., Gragg, W. B., Kaufman, L., & Stewart, G. W. (1976). Reorthogonalization and stable algorithms for updating the gram-Schmidt QR factorization. *Mathematics of Computation*, *30*(136), 772–795.

Davis, T. A. (2006). *Direct methods for sparse linear systems*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Di Cairano, S., & Kolmanovsky, I. V. (2018). Real-time optimization and model predictive control for aerospace and automotive applications. In *Proc. American Control Conference* (pp. 2392–2409). Milwaukee, WI.

Diamond, S., & Boyd, S. (2016). Matrix-free convex optimization modeling. In G. B. (Ed.), *Optimization and Its Applications in Control and Data Sciences* (pp. 221–264). Springer, Cham.

Diehl, M., Bock, H. G., & Schlöder, J. P. (2005). A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, *43*(5), 1714–1736.

Diehl, M., Ferreau, H. J., & Haverbeke, N. (2009). Efficient numerical methods for nonlinear MPC and moving horizon estimation. In L. Magni, D. M. Raimondo, & F. Allgöwer (Eds.), *Lecture Notes in Control and Information Sciences: vol. 384, Nonlinear model predictive control* (pp. 56–98). Berlin, Heidelberg: Springer.

Dongarra, J., Eijkhout, V., & Łuszczek, P. (2001). Recursive approach in sparse matrix LU factorization. *Scientific Programming*, *9*(1), 51–60.

Golub, G. H., & Loan, C. F. V. (2013). *Matrix computations* (4th ed.). Baltimore, MD: The John Hopkins University Press.

Jerez, J. L., Goulart, P. J., Richter, S., Constantinides, G. A., Kerrigan, E. C., & Morari, M. (2014). Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, *59*, 3238–3251.

Kerrigan, E. C., & Maciejowski, J. M. (2000). Soft constraints and exact penalty functions in model predictive control. In *Proc. UKACC international conference (control)*. Cambridge, UK.

Kouzoupis, D., Frison, G., Zanelli, A., & Diehl, M. (2018). Recent advances in quadratic programming algorithms for nonlinear model predictive control. *Vietnam Journal of Mathematics*, *46*, 863–882.

Lawson, C. L., & Hanson, R. J. (1995). *Classics in applied mathematics, Solving least squares problems*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Nesterov, Y. (2004). *Introductory lectures on convex optimization: A basic course*. Dordrecht, The Netherlands: Kluwer Academic.

Nocedal, J., & Wright, S. (2006). *Numerical optimization* (2nd ed.). Springer.

Ohtsuka, T. (2004). A continuation/GMRES method for fast computation of nonlinear receding horizon control. *Automatica*, *40*(4), 563–574.

Piga, D., Forgione, M., Formentin, S., & Bemporad, A. (2019). Performance-oriented model learning for data-driven MPC design. *IEEE Control Systems Letters*, *3*(3), 577–582.

Qin, S., & Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control Engineering Practice*, *11*(7), 733–764.

Rafal, M. D., & Stevens, W. F. (1968). Discrete dynamic optimization applied to on-line optimal control. *AiChE Journal*, *14*(1), 85–91.

Saad, Y. (2003). *Iterative methods for sparse linear systems* (2nd ed.). Society for Industrial and Applied Mathematics.

Saraf, N. (2019). *Bounded-variable least-squares methods for linear and nonlinear model predictive control*. Italy: Ph.D. dissertation, IMT School for Advanced Studies Lucca.

Saraf, N., & Bemporad, A. (2017). Fast model predictive control based on linear input/output models and bounded-variable least squares. In *Proc. 56th IEEE conference on decision and control* (pp. 1919–1924). Melbourne, Australia.

Saraf, N., & Bemporad, A. (2020). A bounded-variable least-squares solver based on stable QR updates. *IEEE Transactions on Automatic Control*, *65*(3), 1242–1247.

Saraf, N., Zanon, M., & Bemporad, A. (2018). A fast NMPC approach based on bounded-variable nonlinear least squares. In *Proc. 6th IFAC conference on nonlinear model predictive control* (pp. 337–342). Madison, WI.

Stella, L., Themelis, A., Sopasakis, P., & Patrinos, P. (2017). A simple and efficient algorithm for nonlinear model predictive control. In *Proc. 56th IEEE conference on decision and control* (pp. 1939–1944). Melbourne, Australia.

Trefethen, L. N., & Bau, D. (1997). *Numerical linear algebra*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Verschueren, R., Frison, G., Kouzoupis, D., van Duijkeren, N., Zanelli, A., Novoselnik, B., et al. (2021). `Acados`: A modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 1–37.

Wächter, A., & Biegler, L. T. (2006). On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25–57.

Wang, Y., & Boyd, S. (2010). Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, *18*, 267–278.

Nilay Saraf was born in Amalner, India, in 1992. He received the B.E. degree in mechanical engineering from the University of Mumbai, India, in 2013, the M.Sc. degree in control engineering from the Delft University of Technology, the Netherlands, in 2015, and the Ph.D. degree in computer science and systems engineering from the IMT School for Advanced Studies Lucca, Italy, in 2019. In 2014/15 he was a research assistant at the Fraunhofer Institute for Solar Energy Systems, Freiburg, Germany. During 2016–2019 he was a researcher at ODYS S.r.l. and is currently a computational scientist at Enel S.p.A., in Italy. His current research interests include model predictive control, numerical optimization, system identification and their practical applications.

Alberto Bemporad received his Master's degree in Electrical Engineering in 1993 and his Ph.D. in Control Engineering in 1997 from the University of Florence, Italy. In 1996/97 he was with the Center for Robotics and Automation, Department of Systems Science & Mathematics, Washington University, St. Louis. In 1997–1999 he held a postdoctoral position at the Automatic Control Laboratory, ETH Zurich, Switzerland, where he collaborated as a senior researcher until 2002. In 1999-2009 he was with the Department of Information Engineering of the University of Siena, Italy, becoming an Associate Professor in 2005. In 2010–2011 he was with the Department of Mechanical and Structural Engineering of the University of Trento, Italy. Since 2011 he is Full Professor at the IMT School for Advanced Studies Lucca, Italy, where he served as the Director of the institute in 2012–2015. He spent visiting periods at Stanford University, University of Michigan, and Zhejiang University. In 2011 he co-founded ODYS S.r.l., a company specialized in developing model predictive control systems for industrial production. He has published more than 400 papers in the areas of model predictive control, hybrid systems, optimization, automotive control, and is the co-inventor of 16 patents. He is author or coauthor of various software packages for model predictive control design and implementation, including the Model Predictive Control Toolbox (The Mathworks, Inc.) and the Hybrid Toolbox for MATLAB. He was an Associate Editor of the IEEE Transactions on Automatic Control during 2001–2004 and Chair of the Technical Committee on Hybrid Systems of the IEEE Control Systems Society in 2002–2010. He received the IFAC High-Impact Paper Award for the 2011–14 triennial, the IEEE CSS Transition to Practice Award in 2019, and the 2021 SAE Environmental Excellence in Transportation Award. He is an IEEE Fellow since 2010.