Quadratic programming with ramp functions and fast QP-MPC solutions ⋆

Giorgio Valmorbida [a] Morten Hovd [b]

[a] *Université Paris-Saclay, CNRS, CentraleSupélec, Inria, Laboratoire des Signaux et Systèmes, 91190, Gif-sur-Yvette, France.*

[b] *Department of Engineering Cybernetics, Norwegian University of Science & Technology, 7491 Trondheim, Norway.*

Abstract

A novel method is proposed for solving quadratic programming problems arising in model predictive control. The method is based on an implicit representation of the Karush-Kuhn-Tucker conditions using ramp functions. The method is shown to be highly efficient on both small and fairly large Quadratic Program problems, can be implemented using simple computer code, and has modest memory requirements.

Key words: PWA systems. Ramp functions. Model Predictive Control.

1 Introduction

Model Predictive Control (MPC) has been a great success in industry [10], and since its initial development in the 1970's (see, *e.g.* [12]) it has found application in a wide range of industrial processes. The main feature of MPC distinguishing it from classical control design methods such as Linear Quadratic Regulator is the ability to take into account constraints in both inputs, state, and outputs. However, the strong abilities of MPC do not come without a cost. In the conventional MPC formulation, an optimization problem has to be solved online, and for large systems and/or systems requiring high sampling rates, the computational load may become prohibitive. Many approaches have been studied in order to reduce the computational requirements of MPC, including input blocking [2] and utilizing structure in the optimization problem (*e.g.* [11]). This paper will make no attempt at covering all such approaches.

This article proposes a new approach to solving strictly convex Quadratic Programs (QP) problems resulting from MPC problem formulations, based on an implicit problem formulation via ramp functions. Numerical results are provided that demonstrate that the QP problems are solved very fast, for a wide range of problem sizes. The MPC problem is represented exactly, without any simplifications or approximations. The computer code required to implement the solution method is very simple, and the computer memory requirement is modest.

The ramp functions correspond to the rectified linear unit (ReLU) activation functions commonly used in artificial neural networks. However, no attempt is made here at developing the explicit representation that would be required for a neural network implementation.

The paper is organized as follows. Section 2 provides preliminary results, and describes the MPC formulation used in the paper. Section 3 provides the main result, and in Section 4 the application of the main result for efficiently solving QP problems is described. Section 5 demonstrates the efficiency of the method on a range of MPC problems. A discussion is provided in Section 6, focusing on infeasibility detection and infeasibility handling, while Section 7 concludes the paper.

Notation. For a vector $y \in \mathbb{R}^n$, y_i indicates its ith component. For a matrix $M \in \mathbb{R}^{n\times m}$, $M_{(i,j)}$ denotes its (i,j) component, $M_{(i,\cdot)}$ denotes its ith row, and $M_{(\cdot,j)}$ indicates its jth column. Let $\mathbb{D}^N$ denote the set of diagonal matrices of dimension N, and $\mathcal{A} \subseteq \{1,2,\dots,N\}$, define the matrix $I_{\mathcal{A}} \subset \mathbb{D}^N$, with $I_{\mathcal{A}(i,i)} \in \{0,1\}$ and

⋆ This work was supported by the French-Norwegian exchange programme AURORA under contract 294676. This research is also funded in part by ANR via project HANDY, number ANR-18-CE40-0010.

Email addresses: giorgio.valmorbida@centralesupelec.fr (Giorgio Valmorbida), morten.hovd@itk.ntnu.no (Morten Hovd).

$I_{\mathcal{A}(i,i)} = 1$ if $i \in \mathcal{A}$, $I_{\mathcal{A}(i,i)} = 0$ if $i \notin \mathcal{A}$. We also denote $\mathcal{A}^c = \{1, 2, \ldots, N\} \setminus \mathcal{A}$. We denote e_i the ith vector of the canonical basis of $\mathbb{R}^N$.

2 Preliminary results and MPC formulation

Definition 1 *The ramp function $r(y)$ is given by*

$$r(y) = \begin{cases} 0 & \text{if } y < 0 \\ y & \text{if } y \geq 0 \end{cases} \tag{1}$$

Lemma 1 *The ramp function is the only function satisfying, $\forall y \in \mathbb{R}$*

$$(r(y) - y)r(y) = 0 \tag{2a}$$
$$r(y) \geq 0 \tag{2b}$$
$$(r(y) - y) \geq 0. \tag{2c}$$

Proof. First note that the ramp function can be expressed as the unique solution to the convex optimization problem parameterized in y (thus depending on y) with strictly convex objective function as follows

$$\underset{r}{\text{minimize}} \ \frac{1}{2}(r - y)^2 \quad \text{subject to} \quad r \geq 0. \tag{3}$$

With the Lagrangian associated to the optimization problem, $\mathcal{L}(r, \lambda) = \frac{1}{2}(r - y)^2 - \lambda r$, we obtain the Karush-Kuhn-Tucker (KKT) conditions

$$(r - y) - \lambda = 0; \ \lambda r = 0; \ r \geq 0; \ \lambda \geq 0$$

which are necessary for optimality and also sufficient since the problem is strictly convex. Note that these relations offer a characterization in terms of linear and quadratic identities and inequalities in three variables (y, r, λ). To obtain a description in the variables (y, r) one can use $\lambda = (r - y)$ above to obtain $r \geq 0$, $(r - y) \geq 0$, $r(r - y) = 0$. ∎

For $y \in \mathbb{R}^m$ let us define the function $\phi : \mathbb{R}^m \to \mathbb{R}^m$, the vector-valued ramp function, as

$$\phi(y) = \begin{bmatrix} r(y_1) & r(y_2) & \cdots & r(y_m) \end{bmatrix}^\top$$

Remark 1 Note that, due to the piecewise definition of the ramp function in (1), we have that vector ϕ can also be expressed as the product $\phi(y) = I_{\mathcal{A}} y$ where $I_{\mathcal{A}} \in \mathbb{D}^m$, with its diagonal elements verifying $I_{\mathcal{A}(i,i)} \in \{0, 1\}$. Clearly, the set $\mathcal{A} \subseteq \{1, 2, \ldots, N\}$ depends on y. ⌟

2.1 QP-MPC formulation

The MPC problem is formulated in the same way as in [1]:

$$\underset{\mathbf{u}}{\text{minimize}} \ \mathbf{u}^\top H \mathbf{u} + x(k)^\top F^\top \mathbf{u} \tag{5}$$
$$\text{subject to}$$
$$G\mathbf{u} \leq S_u x(k) + w.$$

Next (also following [1]), the variable change $\mathbf{z} = \mathbf{u} + H^{-1} F x(k)$ results in

$$\underset{\mathbf{z}}{\text{minimize}} \ \mathbf{z}^\top H \mathbf{z} \tag{6}$$
$$\text{subject to}$$
$$G\mathbf{z} \leq S x(k) + w$$

where $S = S_u + G H^{-1} F$. Note that as a consequence of using quadratic weights $R_i > 0$ for each input u_i, then $H > 0$ and hence always invertible.

3 Main results

This section shows how a Linear Complementarity Problem (LCP) can be associated to a QP-MPC to obtain a piece-wise affine (PWA) representation implicitly defined in terms of the vector-valued ramp function ϕ. Section 4 details how this implicit PWA representation can be exploited to quickly calculate solutions to the QP-MPC problem.

Theorem 1 *The solution of the QP-MPC is a PWA function given by*

$$\begin{cases} \mathbf{u}(x) = -H^{-1} F^\top x - H^{-1} G^\top \phi(y) & \text{(7a)} \\ y = -Sx + (I - G H^{-1} G^\top)\phi(y) - w. & \text{(7b)} \end{cases}$$

Proof. The following are the KKT conditions are presented in [1]

$$2H\mathbf{z} + G^\top \lambda = 0 \tag{8a}$$
$$\lambda_i (S_{(i,\cdot)} x + w_i - G_{(i,\cdot)} \mathbf{z}) = 0 \tag{8b}$$
$$\lambda \geq 0 \tag{8c}$$
$$Sx + w - G\mathbf{z} \geq 0. \tag{8d}$$

From (8a) we obtain $\mathbf{z} = -H^{-1} G^\top \lambda$, and using this expression in (8b)-(8d), gives the following LCP constraints

$$\lambda_i (S_{(i,\cdot)} x + w_i + G_{(i,\cdot)} H^{-1} G^\top \lambda) = 0$$
$$\lambda \geq 0$$
$$Sx + w + G H^{-1} G^\top \lambda \geq 0.$$

The above set of constraints can be rewritten as

$$\lambda_i \left(\lambda_i - \left(\underbrace{-S_{(i,\cdot)}x + (I - GH^{-1}G^\top)_{(i,\cdot)}\lambda - w_i}_{y_i}\right)\right) = 0$$
$$\lambda \geq 0$$
$$\left(\lambda - \left(-Sx + (I - GH^{-1}G^\top)\lambda - w\right)\right) \geq 0.$$

By defining

$$y = -Sx + (I - GH^{-1}G^\top)\lambda - w, \tag{9}$$

we observe that the above set of inequalities and the complementarity constraint become

$$\lambda_i (\lambda_i - y_i) = 0,\ \lambda \geq 0,\ (\lambda - y) \geq 0.$$

According to Lemma 1, the set of multipliers λ is given by the ramp function of variable y, namely

$$\lambda = \phi(y) \tag{10}$$

then, from (9), we have that y satisfies the implicit algebraic equation (7b), while (7a) follows from from (??) and (10). ■

From the above theorem, we have that, given $x(k)$, the computation of the control action at instant k, $u(k)$, can be carried out by solving the implicit equation in (7b), yielding $y(k)$. From this solution, the Lagrange multipliers λ can be computed according to (10). The computation of $\mathbf{u}(x(k))$ in (7a) thus boils down to the solution of the implicit equation.

The next section proposes an algorithm to solve this implicit equation by exploiting the fact that the ramp function can be written as a matrix multiplication $\phi(y) = I_{\mathcal{A}(x(k))}y$ where we indicate the dependence of the set of *active constraints* $\mathcal{A}$ on the value of $x(k)$. Indeed, if the solution, depending on $x(k)$, gives $y_i < 0$ then $i \notin \mathcal{A}$. If instead, $y_i \geq 0$, then $i \in \mathcal{A}$.

4 Implementation

It is clear from Theorem 3 that the KKT conditions for the MPC optimization problem is an LCP. This is well known both for QPs in general (see, *e.g.* [9]), and within the MPC literature (*e.g.* [3, 6]). However, this knowledge has not been utilized to devise an algorithm for solving the MPC QP problem, using ramp functions to take advantage of the structure and continuity of the problem.

In this section, we present an algorithm to solve the algebraic equation (7b). The algorithm exploits the fact that the ramp function can be expressed as a matrix multiplication by a diagonal matrix as pointed out in Remark 1. That is, that (7b) can be written as

$$y - (I - GH^{-1}G^\top)\underbrace{I_{\mathcal{A}}y}_{\Phi(y)} = -Sx - w \tag{11}$$

(7b) $y = -Sx + \left(I - GH^{-1}G^\top\right)\Phi(y) - w$ 3

for some matrix $I_{\mathcal{A}}$ to be determined. The set $\mathcal{A}$ corresponds to the set of *active constraints*, namely to the set for which the values of multipliers λ are not zero. Therefore, according to the ramp function defining the multipliers, if $y_i < 0$ then the corresponding value of the multiplier is $\lambda_i = 0$, and in this case, $I_{\mathcal{A}(i,i)} = 0$, that is, the the corresponding diagonal element of $I_{\mathcal{A}}$ is zero. If instead $y_i \geq 0$ then $I_{\mathcal{A}(i,i)} = 1$. The following definition gives the definition of the pairs $(y, I_{\mathcal{A}})$ allowing to establish an equivalence between (7b) and (11).

Definition 2 *For a given x, a pair is said $(y, I_{\mathcal{A}})$* compatible *if $(y, I_{\mathcal{A}})$ satisfy (11) and $I_{\mathcal{A}(i,i)} = 1$ if $y_i \geq 0$ and $I_{\mathcal{A}(i,i)} = 0$ if $y_i < 0$.*

Based on the above observations, the solution to the implicit equation (7b) is obtained whenever we satisfy (11) with a *compatible* pair $(y, I_{\mathcal{A}})$. We shall denote $\mathcal{A}(x(k))$ as the set of active constraints for a given $x(k)$ defining the right hand side of (7b). To search for a solution to (7b), we propose below an algorithm that searches for the set of active constraints by adding elements to or removing elements from the set $\mathcal{A}$, thus modifying matrix $I_{\mathcal{A}}$ one element at the time, and updating the solutions y using the matrix inversion lemma, which is recalled below.

Lemma 2 (Matrix Inversion Lemma) *Let $Q \in \mathbb{R}^{N\times N}$ be an invertible matrix, we have*

$$(Q + q_u q_c q_v)^{-1} = Q^{-1} - \underbrace{(q_c^{-1} + q_v Q^{-1} q_u)}_{\text{scalar}}{}^{-1} \left(Q^{-1} q_u q_v Q^{-1}\right)$$

where $q_c \in \mathbb{R} \setminus \{0\}$ is a scalar, $q_u \in \mathbb{R}^N$, and $q_v^\top \in \mathbb{R}^N$, that is q_u is a column vector and q_v is a row vector.

Remark 2 Noting that $(q_c^{-1} + q_v Q^{-1} q_u)$ is a scalar, it is clear that the update of Q^{-1} when adding the rank one term $Q^{-1} q_u q_v Q^{-1}$ requires only sums and multiplications and a single division by a scalar. ⌟

Let us define

$$Q(\mathcal{A}) = \left(I_{\mathcal{A}^c} + GH^{-1}G^\top I_{\mathcal{A}}\right)$$

and observe that the solution to the linear system (11) is obtained following

$$y - (I - GH^{-1}G^\top)I_{\mathcal{A}}y = -Sx - w$$
$$Q(\mathcal{A})y = -Sx - w$$
$$y = Q(\mathcal{A})^{-1}(-Sx - w).$$

Consider the case where $i \notin \mathcal{A}$ and define $\mathcal{A}_{+i} = \mathcal{A} \cup \{i\}$, namely i *enters* the set $\mathcal{A}$. Since $I_{\mathcal{A}_{+i}} = I_{\mathcal{A}} + e_i e_i^\top$, and $I_{\mathcal{A}^c_{+i}} = I_{\mathcal{A}^c} - I_{\mathcal{A}^c(\cdot,i)} e_i^\top$, we have

$$Q(\mathcal{A}_{+i}) = Q(\mathcal{A}) - (I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)} e_i^\top.$$

The above matrix $Q(\mathcal{A}_{+i})$ presents a sum of $Q(\mathcal{A})$ with a rank one matrix as in Lemma 2, with, respectively $q_c = -1$, $q_v = e_i^\top$ and $q_u = (I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)}$. Following the matrix inversion lemma, we have the update of the inverse by adding i to $\mathcal{A}$ as

$$\begin{aligned} Q(\mathcal{A}_{+i})^{-1} =& Q(\mathcal{A})^{-1} \\ & - (-1 + e_i^\top Q(\mathcal{A})^{-1}(I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)})^{-1} \times \\ & (Q(\mathcal{A})^{-1}(I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)} e_i^\top Q(\mathcal{A})^{-1}) . \end{aligned}$$

By defining

$$v(\mathcal{A}, i) = Q^{-1}(\mathcal{A})(I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)},$$

we obtain

$$\begin{aligned} Q(\mathcal{A}_{+i})^{-1} =& Q(\mathcal{A})^{-1} \\ & - (-1 + v(\mathcal{A}, i)_i)^{-1} v(\mathcal{A}, i) \left(Q(\mathcal{A})^{-1}\right)_{(i,\cdot)} . \end{aligned} \tag{12}$$

We also have the update of y, denoted $y_+[i]$ as

$$\begin{aligned} y_+[i] =& Q(\mathcal{A}_{+i})^{-1}(-Sx - w) \\ =& Q(\mathcal{A})^{-1}(-Sx - w) \\ & - (-1 + e_i^\top Q(\mathcal{A})^{-1}(I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)})^{-1} \times \\ & (Q(\mathcal{A})^{-1}(I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)} e_i^\top Q(\mathcal{A})^{-1})(-Sx - w) \\ =& y - (-1 + e_i^\top Q(\mathcal{A})^{-1}(I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)})^{-1} \times \\ & (Q(\mathcal{A})^{-1}(I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)})\, y_i, \end{aligned}$$

that is,

$$y_+[i] = y - y_i(-1 + v(\mathcal{A}, i)_i)^{-1} v(\mathcal{A}, i) \tag{13}$$

Similarly, consider the case where $i \in \mathcal{A}$ and define $\mathcal{A}_{-i} = \mathcal{A} \setminus \{i\}$, namely i *is removed from* the set $\mathcal{A}$,

$$Q(\mathcal{A}_{-i}) = Q(\mathcal{A}) + e_i^\top (I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)}$$

The above matrix $Q(\mathcal{A}_{-i})$ presents a sum of $Q(\mathcal{A})$ with a rank one matrix as in Lemma 2, with, respectively $q_c = 1$, and, as above, $q_v = e_i^\top$ and $q_u = (I_{\mathcal{A}} - GH^{-1}G^\top)_{(\cdot,i)}$. thus, similarly to above, the following expressions for the inverse and solution updates

$$\begin{aligned} Q(\mathcal{A}_{-i})^{-1} =& Q(\mathcal{A})^{-1} \\ & - (1 + v(\mathcal{A}, i)_i)^{-1} v(\mathcal{A}, i) \left(Q(\mathcal{A})^{-1}\right)_{(i,\cdot)} , \end{aligned} \tag{14}$$

$$y_-[i] = y - y_i(1 + v(\mathcal{A}, i)_i)^{-1} v(\mathcal{A}, i). \tag{15}$$

Given the above steps to update the inverse and y, we use the following criteria to choose which elements to add to or to remove from the current set of active constraints $\mathcal{A}$. This criteria is based on elements of the pair $(y, \mathcal{A})$

- If $y_i < 0$ for $i \in \mathcal{A}$, then constraint i is removed from $\mathcal{A}$ and the inverse of $Q(\mathcal{A}_{-i})$ and an update of y are computed. → y_i negativ og constraint i er aktiv → fjerne fra aktivt set
- If $y_i \geq 0$ for $i \notin \mathcal{A}$, then constraint i is added to $\mathcal{A}$ and the inverse of $Q(\mathcal{A}_{+i})$ and an update of y are computed. hvis y_i er positiv men ikke constraint i aktivt set → legge til

The order in which we carry out the inclusion and removal of elements in $\mathcal{A}$ is as follows

metoden

- Remove first from $\mathcal{A}$ constraints corresponding to negative y_i. If there are more then one such constraint, remove first the one corresponding to the most negative y_i.
- Then add to $\mathcal{A}$ a constraint that is not a member of $\mathcal{A}$ and corresponds to a positive value of y_i. If there is more than one such constraint, add first the one corresponding to the largest y_i.

The above steps are described in Algorithm 1 below. The solution of the algebraic loop in (7b) is obtained with Algorithm 1 and the corresponding control input $u(k)$ from the vector $\mathbf{u}$ as detailed in Algorithm 2.

Algorithm 1 Solution to the algebraic equation (7b)

Require: $GH^{-1}G$, $\mathcal{A}$, $invQ$, y satisfying $y = invQ(-Sx - w)$ → Q(A) ?

while $(\mathcal{A}, y)$ not compatible **do**
  if $ind(sign(y, -1)) \cap \mathcal{A} \neq \emptyset$ **then** settet av elementene i det aktive settet som har negativ y
    $L \leftarrow ind(sign(y, -1)) \cap \mathcal{A}$ → L = indexene til const. med neg. som er i A
    $i \leftarrow ind(min(y, L))$ → i = index til det minste neg. elem
    $v \leftarrow invQ(I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)}$
    $\mathcal{A} \leftarrow \mathcal{A} \setminus \{i\}$ → fjerner i fra A
    $q_0 \leftarrow 1$
  else if $ind(sign(y, 1)) \neq \mathcal{A}$ **then** → hvis det finnes pos. y som ikke er i A
    $L \leftarrow ind(sign(y, 1)) \setminus \mathcal{A}$ → indexene til neg. pos. y som ikke er i A
    $i \leftarrow ind(max(y, L))$ → indx til største pos. elementet som ikke er i A
    $v \leftarrow invQ(I_{\mathcal{A}^c} - GH^{-1}G^\top)_{(\cdot,i)}$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$ → adder i til A
    $q_0 \leftarrow -1$
  end if
  $invQ \leftarrow invQ - (q_0 + v_i)^{-1} v\, (invQ)_{(i,\cdot)}$
  $y \leftarrow y - y_i (q_0 + v_i)^{-1} v$
end while
return $\mathcal{A}$, $invQ$, y

4.1 Algorithm discussion

Although, in the authors' experience, Algorithm 1 (with the infeasibility detection proposed in Section 6) works very well for QP problems arising from MPC formulations, strictly convex QP formulations can be formulated

(7b) $y = -Sx + (I - GH^{-1}G^\top)\Phi(y) - w$

require: $GH^{-1}G$

$\mathcal{A}$: set of active constraints

* sign $(y, -1)$ → true for de elementer som er negative (samme fortegn som -1)

(5)

minimize $u^T H u + x(k)^T F^T u$
u

s.t. $Gu \leq S_x x(k) + w$

Algorithm 2 Solution to the QP-MPC (5)

Require: $GH^{-1}G$, S, w, F as in (7b) and (5)
while MPC running **do**
  Obtain $x(k)$
  $Qinv \leftarrow I$
  $\mathcal{A} \leftarrow \emptyset$
  $y \leftarrow (-Sx(k) - w)$
  $y \leftarrow Algorithm1(GH^{-1}G, \mathcal{A}, invQ, y)$
  $\mathbf{u}(x_k) \leftarrow -H^{-1}F^\top x_k - H^{-1}G^\top \phi(y)$ (7a)
  Apply $u(x_k)$ to the plant
end while

for which a modification of the algorithm is required. Consider the problem (6) with $S = 0$ and

$$H = \begin{bmatrix} 11 & 9 \\ 9 & 11 \end{bmatrix}, \; G = \begin{bmatrix} 1 & 0 \\ 0 & -1 \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ -\frac{3}{\sqrt{10}} & -\frac{1}{\sqrt{10}} \end{bmatrix}, \; w = \begin{bmatrix} -0.5 \\ -0.8 \\ -\frac{1}{2\sqrt{2}} \\ -\frac{0.15}{\sqrt{10}} \end{bmatrix}.$$

Starting with an empty active set, constraint 2 is the first constraint to add to the active set, and then constraint 4 should be added. The active set $\mathcal{A} = \{2, 4\}$, results in

constraint 1 2 3 4 — positive ↓ should be added to active set

$$y = \begin{bmatrix} 0.2833 & 5.2444 & -0.0589 & 5.0772 \end{bmatrix}^T.$$

Thus, constraint 1 should be added to the active set - but since the problem has only two degrees of freedom, this would lead to a rank defect Q matrix. Therefore, one of the constraints in the active set has to be removed, even though all constraints in the active set have positive y values.

problem: when # of constraints in $\mathcal{A}$ is m

→ The following procedure can be used to identify the constraint to be removed. Let the index i denote the constraint to be added to the active set, and note that the problem only arises when the number of constraints already in the active set equals m. The current (non-optimal) solution point can be calculated from (7a) and the current value of y. Clearly, the solution point needs to move in a direction for which the value of constraint i decreases. Start with $v_0 = -G_{(i,:)}^\top$ as the candidate direction for changing the solution point. However, the solution point cannot move in a direction in which the values of the constraints in the active set increase.

- For $k = 1 : m$, define j as the constraint index (row index in G) for element k of $\mathcal{A}$.
- If $G_{(j,:)} v_{k-1} \leq 0$, $v_k = v_{k-1}$, else

$$v_k = P_j(v_{k-1}),$$

where P_j denotes the projection onto the subspace perpendicular to $G_{(j,:)}$. Note that if the rows of G are scaled to be of unit length (as in the example above)

$$P_j(v_{k-1}) = (I - G_{(j,:)}^\top G_{(j,:)}) v_{k-1}.$$

The constraint to be removed from $\mathcal{A}$ is then the first constraint to become inactive when moving the solution point in the direction v_m. If $v_m = 0$ the problem is infeasible.

In the example above we find that constraint 2 should be removed from the active set, and constraint 1 should be added to the active set. It is easily checked that the resulting solution is optimal.

The required update to Q^{-1} can be done with a rank 2 update using the matrix inversion lemma, or alternatively by two rank 1 updates, removing first the constraint to leave the active set.

5 Numerical Results

To illustrate the performance of the proposed approach, numerical results are provided for MPC applied to three systems of different size. The simulation times are reported for simulating 100 timesteps using the implicit QP solution approach described in the previous section, and compared to the simulation times when using `qpOASES`[4, 5] (starting from the same initial state). The simulations are performed in Matlab on a Windows PC. The calculation times can vary a little from run to run, since Windows is not a real time operating system. The simulations are therefore repeated multiple times, and the average time is reported.

The solver `qpOASES` is a highly regarded QP solver for MPC problems. It takes advantage of the observation that the state x on the right hand side of the constraints in (5) is unlikely to change very much from one timestep to the next. At time k, a parametric active set approach is used to follow a homotopy path from the optimal solution for $x(k-1)$ to the optimal solution for $x(k)$. In the examples, the upper and lower constraints for $\mathbf{u}$ are separated from the other constraints in (5), and given to `qpOASES` separately, as recommended in [5]. At each timestep except the initial one, the hot start functionality in `qpOASES` is used.

The two optimization solvers give essentially the same results, and they would be indistinguishable in plots showing the closed-loop behavior of the systems. Therefore we do not illustrate the time responses of the closed loop. Instead, the focus here are on the simulation times, which document the efficiency of the proposed method.

Example 1 Consider the double integrator example

from [8]. The system dynamics are given by

$$x(k+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 1 \\ 0.3 \end{bmatrix} u(k)$$

with constraints

$$-1 \leq u(k) \leq 1, \quad \begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq x(k) \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}.$$

The prediction horizon $N = 10$ is used, and the system with control is simulated for 100 time steps. The average time for 5 runs starting from $x_0 = [5 \ -2]^\top$ when solving the QP using ramp functions is $0.0020s$, whereas the average time when using `qpOASES` is $0.0023s$. ⌟

Example 2 We now study the four-state system from [8, 7]. The discrete-time dynamics are given by

$$x(k+1) = \begin{bmatrix} 0.928 & 0.002 & -0.003 & -0.004 \\ 0.041 & 0.954 & 0.012 & 0.006 \\ -0.052 & -0.046 & 0.893 & -0.003 \\ -0.069 & 0.051 & 0.032 & 0.935 \end{bmatrix} x(k) + \begin{bmatrix} 0 & 0.336 \\ 0.183 & 0.007 \\ 0.090 & -0.009 \\ 0.042 & 0.012 \end{bmatrix} u(k)$$

$$y(k) = Cx(k) = \begin{bmatrix} 0 & 0 & -0.098 & 0.269 \\ 0 & 0 & 0.080 & 0.327 \end{bmatrix} x(k)$$

with constraints

$$\begin{bmatrix} -1 \\ -1 \end{bmatrix} \leq u(k) \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} -1 \\ -1 \end{bmatrix} \leq Cx(k) \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

The prediction horizon is $N = 30$, and the initial state $x_0 = [25.5724 \ 25.3546 \ 9.7892 \ 0.2448]^\top$. Simulating 100 timesteps when solving the QP using ramp functions took on average $0.0041s$, while the average time for `qpOASES` was $0.0090s$. ⌟

Example 3 The final example is based on the 82-state binary distillation column model by Skogestad [13]. The control inputs are four in total: the top product flowrate, the top reflux flowrate, the bottom product flowrate, and bottom boil-up (energy supply). The outputs are the top and bottom compositions, and the liquid levels in the top accumulator and the column bottoms. The levels are open loop integrators.

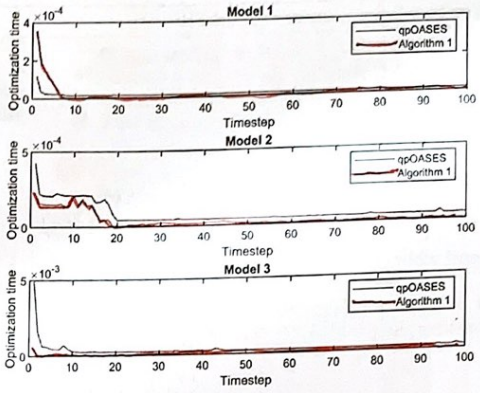

Fig. 1. Solution times in seconds per timestep for a single run of Algorithm 1 and `qpOASES`, for each example.

First the model is balanced and reduced to 25 states. Next, the reduced model is converted to discrete time. Both the original system and the reduced model equations and initial state are not listed here for space reasons [1].

Using a prediction horizon $N = 30$, the resulting QP problem has 1562 constraints and 120 degrees of freedom. Simulating 100 timesteps when solving the QP using ramp functions took on average $0.0055s$, while the average time for `qpOASES` was $0.0401s$. ⌟

Table 2 summarizes indicators for the performance.Note, in particular, that few of the time steps required the maximum number of iterations in Algorithm (1). Note also the sparsity of matrix $Q(\mathcal{A})^{-1}$, which is smaller for the example with more states.

The solution time per timestep is shown, for each example, in Fig. 1. `qpOASES` is the faster solver initially for Example 1, but the rest of the time Algorithm 1 is faster.

Figure 2 shows the simulation times for 100 simulation runs, each of length 100 timesteps, for Algorithm 1 and `qpOASES`. For run $i, i = \{1, \ldots, 100\}$ the same initial conditions are used for Algorithm 1 and `qpOASES`. The initial conditions are randomly generated, although only feasible initial conditions are retained for comparison of simulation times. Algorithm 1 more often results in shorter simulation times.

6 Discussion

Relationship to active set methods.
Algorithm 1 updates the set of active constraints at each

[1] They can be obtained for morten.hovd@itk.ntnu.no on request.

			Algorithm 2						
Ex.	n	N	max. # iterations	k with max. # iterations	average # iterations	max. # elements in $\mathcal{A}$	Worst sparsity of $Q^{-1}(\mathcal{A})$	avg. time per time step (μs)	worst time ($k = 1$) (μs)
1	2	10	6	1-2	1.2	7.6%	8.7%	20	465
2	4	30	4	1-14	1.47	0.95%	1.3%	41	388
3	25	30	4	1	1.02	0.26%	0.26%	55	962

Table 1
Order of the system (n), prediction horizon (N). All examples ran 100 time-steps from the reported initial conditions. For Algorithm 2 the table reports the maximum number of iterations of Algorithm 1 within a time step, time steps in which the maximum number iterations of Algorithm 1 were executed, average number of iterations of Algorithm 1 for the 100 time-steps, maximum number of active constratins (as percentage of total), percentage non-zero elements in $Q^{-1}(\mathcal{A})$, average time step for the 100 time steps and maximum time taken within one time-step (corresponding to the first time $k = 1$).

	Algorithm 2	qpOASES
Ex.	Total time (ms)	Total time (ms)
1	2.0	2.3
2	4.1	9.0
3	5.5	40.1

Table 2
Total execution time for 100 time steps of Algorithm 2 and qpOASES considering the same initial conditions (both from the average of several runs).

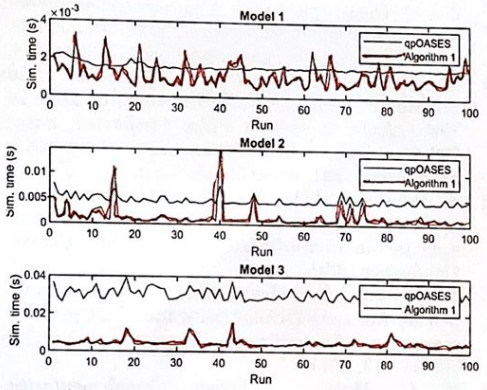

Fig. 2. Solution times in seconds for simulation runs of length 100 timesteps for Algorithm 1 and qpOASES, for each example.

iteration, and may thus be categorized as an active set method. However, conventional active set methods solve the KKT conditions (in the form of a set of linear equations) for each candidate active set tried, in order to determine a step direction and thereby find which constraint to add or drop. In contrast, Algorithm 1 updates $invQ$ through a rank 1 update, and the constraint to be added or dropped is found simply from looking at the values of the elements of the y vector, taking into account whether the corresponding constraint is in the active set.

Memory requirements for Algorithm 1
The computer memory required for implementing Algorithm 1 is clearly very modest, consisting mainly of the storage of $GH^{-1}G^{\top})$, S, and w, which is the minimum required to define the LCP, as well as the leading m rows of $H^{-1}F^{\top}$ and $H^{-1}G^{\top}$ that are required for calculating $u(k)$ from x and λ (see Theorem 3). In addition, Q^{-1} and the set of active constraints $\mathcal{A}$ needs to be stored (and updated), but storing these as sparse matrices can save significant memory, since, as noted above, the number of active constraints cannot exceed the number of degrees of freedom. Indeed, for the above numerical examples, a full matrix Q^{-1} would contain n_c^2 elements, where $n_c = 66$, $n_c = 316$ and $n_c = 1562$ respectively for examples 1, 2 and 3. However, since only a few of these elements are not zero, as detailed in Table 2, column *Worst sparsity of* $Q^{-1}(\mathcal{A})$, the sparse storage allows to reduce the storage space to a small fraction of the dimensions of $Q^{-1}(\mathcal{A})$.

Initial data for Algorithm 1
One possible input data for the solution of the implicit equation using Algorithm 1 is $\mathcal{A} = \emptyset$ corresponding to no active constraints, giving $Q_{\mathcal{A}} = I$, hence $y = -Sx - w$.

This initial input is clearly very efficient when there are no active constraints. However, with the proposed rules for removing or adding constraints to the active set given in Section 4, the numerical experiments showed that the solution method is still very efficient, and the number of solver iterations within one timestep have not been found to exceed the number of (actually) active constraints by much.

Infeasibility detection
To detect infeasibily we have relied on the heuristics of checking the value of $(-1 + v(\mathcal{A}, i)_i)$ when adding constraints. Indeed, we observe that these values become very small when an inconsistent set of constraints is selected.

Infeasibility has been investigated for all examples above by scaling the initial state until the problem becomes infeasible. Using a threshold value of 10^{-13} for $(-1 + v(\mathcal{A}, i)_i)$, infeasibility was correctly determined in all three examples, to within an accuracy of three decimal places in the scaling factor.The appropriate threshold value will depend on, and should be significantly larger than, the machine precision. Clearly, setting the threshold too large will result in the algorithm declaring infeasibility unnecessarily. Even though we implemented

this procedure for infeasibility detection, it has not been detailed Algorithm 1.

Soft Constraints
From an application point of view, a quite different issue is what *should* happen when an MPC problem is infeasible. The basic options are either to have some backup functionality bringing the system to a safe state (which often means shutting down the system entirely), or to design the MPC to "make the best of it", trying to minimize whatever damage might be caused by operation in an infeasible region of the state space. Such damage minimization can be achieved by using *soft constraints*, by adding slack variables to constraints where violations are physically possible and operationally (temporarily) tolerable, and adding corresponding penalty terms to the objective function.

A desirable property is for the soft constraints to be *exact*. This is achieved by a sufficiently large weight on a penalty term linear in the slack variable. What is sufficiently large can be calculated as described in [8]. Quadratic terms in the penalty function do not affect whether the soft constraint is exact, and quadratic terms are therefore sometimes dropped. However, when solving the MPC QP using ramp functions, the Hessian matrix needs to be invertible (positive definite), and hence weights on quadratic terms in the penalty functions are required.

Precision of the Lagrange Multipliers and complementarity KKT conditions
Although of little importance in practical applications, it is interesting to observe that the complementarity constraint of the KKT conditions is fulfilled with very high accuracy for the proposed method. That is, the λ's for the inactive constraints are identically zero (or of magnitude similar to machine precision), whereas conventional optimization routines will give λ's for inactive constraints in the range of some tolerance specification – often in the range of 10^{-8}.

7 Conclusions and Perspectives

A novel method for solving QPs arising from MPC problems has been proposed. The method is shown to be efficient for a wide range of problem sizes, and can be implemented using short and simple computer code. The method allows for simple re-tuning of the MPC, something which is very cumbersome when using explicit MPC. The method is currently limited to strictly convex QP problems, semi-definite Hessian matrices cannot be accommodated.

Future work will address this limitation, and also attempt to extend the method to LP-MPC. Finally, both for general QPs and for QP-MPC in particular, we did not study the convergence of the proposed algorithm, this is a topic for future research.

References

[1] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38:3–20, 2002.

[2] R. Cagienard, P. Grieder, E. Kerrigan, and M. Morari. Move blocking strategies in receding horizon control. *Journal of Process Control*, 17(6):563–570, 2007.

[3] E. F. Camacho. Constrained generalized predictive control. *IEEE Transactions on Automatic Control*, 38(2):327–332, 1993.

[4] H.J. Ferreau, H.G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit mpc. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008.

[5] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[6] M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013. `http://control.ee.ethz.ch/~mpt`.

[7] M. Hovd and R. D. Braatz. Handling state and output constraints in MPC using time-dependent weights. In *Proceedings of the American Control Conference*, 2001.

[8] M. Hovd and F. Stoican. On the design of exact penalty functions for mpc using mixed integer programming. *Computers & Chemical Engineering*, 70:104–113, 2014.

[9] K. G. Murty. *Linear Complementarity, Linear and Nonlinear Programming*. Helderman Verlag, Berlin, Germany, 1988. `http://www-personal.umich.edu/~murty/books/linear_complementarity_webbook`.

[10] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, pages 733–764, 2003.

[11] C. V. Rao, S. J. Wright, and J. B. Rawlings. Application of interior-point methods to model predictive control. *J. of Optimization Theory and Applications*, 99(3):723–757, December 1998.

[12] J. Richalet, A. Raoult, J. L. Testud, and J. Papon. Model predictive heuristic control: Application to industrial processes. *Automatica*, pages 413–428, 1978.

[13] S. Skogestad. Matlab distillation column model, Accessed October 2021. `http://folk.ntnu.no/skoge/distillation/`.