Math. Prog. Comp. (2014) 6:327–363
DOI 10.1007/s12532-014-0071-1

FULL LENGTH PAPER

qpOASES: a parametric active-set algorithm for quadratic programming

Hans Joachim Ferreau · Christian Kirches · Andreas Potschka · Hans Georg Bock · Moritz Diehl

Received: 13 March 2012 / Accepted: 3 April 2014 / Published online: 30 April 2014

Abstract Many practical applications lead to optimization problems that can either be stated as quadratic programming (QP) problems or require the solution of QP problems on a lower algorithmic level. One relatively recent approach to solve QP problems are parametric active-set methods that are based on tracing the solution along a linear homotopy between a QP problem with known solution and the QP problem to be solved. This approach seems to make them particularly suited for applications where a-priori information can be used to speed-up the QP solution or where high solution accuracy is required. In this paper we describe the open-source C++ software package qpOASES, which implements a parametric active-set method in a reliable and efficient way. Numerical tests show that qpOASES can outperform other popular academic and commercial QP solvers on small- to medium-scale convex test examples of the Maros-Mészáros QP collection. Moreover, various interfaces to third-party software packages make it easy to use, even on embedded computer hardware. Finally, we describe how qpOASES can be used to compute critical points of nonconvex QP problems.

H. J. Ferreau (✉)
ABB Corporate Research, Segelhofstrasse 1K, 5405 Baden-Dättwil, Switzerland
e-mail: joachim.ferreau@ch.abb.com

C. Kirches · A. Potschka · H. G. Bock
Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany

M. Diehl
Eletrical Engineering Department (ESAT-STADIUS / OPTEC), KU Leuven, Kasteelpark Arenberg 10, 3001 Leuven, Belgium

M. Diehl
Department of Microsystems Engineering IMTEK, University of Freiburg, Georges-Koehler-Allee 102, 79110 Freiburg, Germany

Keywords Parametric quadratic programming · Active set method · Model predictive control

Mathematics Subject Classification 90C20 (Quadratic programming) · 65K05 (Mathematical programming methods)

1 Introduction

This paper describes the current release 3.0 of the open-source software package qpOASES[1]. This software package implements a parametric active-set method for solving convex quadratic programming (QP) problems and for computing critical points of nonconvex quadratic programming problems.

The class of convex QP problems is important in its own right. Gould and Toint maintain a bibliography [30] of currently about 1000 publications that comprise many application papers from disciplines as diverse as portfolio analysis, structural analysis, VLSI design, discrete-time stabilization, optimal and fuzzy control, finite impulse response design, optimal power flow or economic dispatch. The Maros-Mészáros QP test set [44] collects a number of benchmark and application problems that can be accessed through the CUTEr testing environment [32]. Linear model predictive control (MPC) problems constitute another important subclass of convex QP problems. As MPC is frequently applied to processes with very fast dynamics, it becomes crucial to solve the resulting convex QP problems at very high rates; possibly within a millisecond or less [57]. Moreover, as MPC controllers typically need to run autonomously without further user-interaction, QP solution needs to be highly reliable.

QP problems also arise as subproblems in sequential quadratic programming (SQP) methods. SQP methods aim at solving nonlinear optimization problems by using a linear-quadratic approximation of the original problem in each iteration. Depending on the way the nonlinear objective function is approximated, the resulting QP problems are often convex but may also become nonconvex for certain SQP-type schemes. Again, the fast and reliable solution of QP problems is crucial to make such nonlinear optimization algorithms work efficiently.

1.1 Problem description

We consider quadratic programming problems of the following form,

$$\min_{x\in\mathbb{R}^n} \frac{1}{2} x^T H x + g^T x \tag{1a}$$

$$\text{s. t. } a^{\mathrm{l}} \leq Ax \leq a^{\mathrm{u}}, \tag{1b}$$

comprising a symmetric Hessian matrix $H \in \mathbb{R}^{n\times n}$, a constraint matrix $A \in \mathbb{R}^{m\times n}$, a gradient vector $g \in \mathbb{R}^n$, and lower and upper constraint bound vectors $a^{\mathrm{l}}, a^{\mathrm{u}} \in$

[1] The name qpOASES is derived from the term "online active set strategy" [19] reflecting the fact that the code has been originally developed for use in model predictive control applications.

$\mathbb{R}^m \cup \{\infty, -\infty\}^m$. Throughout this paper, inequality signs between vectors are to be understood element-wise.

For reasons of efficiency, it is important to exploit box constraints on the problem variables that lend the special substructure $A^T = (I \quad \tilde{A}^T)$ to the constraint matrix. qpOASES exploits this substructure in the numerical linear algebra methods.

It is appropriate to introduce some notation at this point. Throughout this article, vectors are column vectors; an index subscript i on a vector denotes the i-th element; a set subscript on a vector denotes the subvector of elements; the operator $\circ$ denotes element-wise multiplication of vectors. We will need to encode a working set W as an m-vector with entries from $\{-1, 0, +1\}$, whose i-th component indicates whether constraint i of the matrix A is inactive ($W_i = 0$) or active at the lower ($W_i = -1$) or upper ($W_i = +1$) bound. We further denote by A_W the matrix that consists of the rows of A indicated by $W_i \neq 0$.

1.2 Optimality conditions

A quadratic program of the form (1) is *convex* (*strictly convex*) if and only if its Hessian matrix H is positive semidefinite (positive definite); it is *nonconvex* otherwise. If feasible, strictly convex QP problems are known to possess a unique global minimum. Uniqueness still holds for feasible convex QP problems if H is positive definite on the null-space of the strictly active constraints in the solution. For nonconvex QP problems, finding the global minimum is NP-hard [45], as is the verification of local optimality of a constrained critical point in the case of weakly active constraints [10]. In these cases, we only strive to find a locally optimal solution or a critical point, respectively.

For the characterization of solutions of QP (1) we partition the index set $\overline{m} = \{1, \ldots, m\}$ for a feasible point x into four disjoint sets

$$\mathscr{A}^{\mathrm{e}}(x) = \{i \in \overline{m} \,|\, a_i^{\mathrm{l}} = (Ax)_i = a_i^{\mathrm{u}}\}, \quad \mathscr{A}^{\mathrm{l}}(x) = \{i \in \overline{m} \,|\, a_i^{\mathrm{l}} = (Ax)_i < a_i^{\mathrm{u}}\},$$
$$\mathscr{A}^{\mathrm{u}}(x) = \{i \in \overline{m} \,|\, a_i^{\mathrm{l}} < (Ax)_i = a_i^{\mathrm{u}}\}, \quad \mathscr{A}^{\mathrm{f}}(x) = \{i \in \overline{m} \,|\, a_i^{\mathrm{l}} < (Ax)_i < a_i^{\mathrm{u}}\}$$

of equality, lower active, upper active, and free constraint indices, respectively. It is well known [see, e.g., 48] that for any solution x^* of QP (1) there exists a vector $y^* \in \mathbb{R}^m$ of Lagrange multipliers or dual variables such that

$$Hx^* + g - A^T y^* = 0, \qquad a^{\mathrm{l}} \leq Ax^* \leq a^{\mathrm{u}}, \tag{2a}$$
$$(Ax^* - a^{\mathrm{l}})_i y_i^* = 0, \quad i \in \mathscr{A}^{\mathrm{l}}(x^*), \qquad y_i^* \geq 0, \quad i \in \mathscr{A}^{\mathrm{l}}(x^*), \tag{2b}$$
$$(Ax^* - a^{\mathrm{u}})_i y_i^* = 0, \quad i \in \mathscr{A}^{\mathrm{u}}(x^*), \qquad y_i^* \leq 0, \quad i \in \mathscr{A}^{\mathrm{u}}(x^*), \tag{2c}$$
$$y_i^* = 0, \quad i \in \mathscr{A}^{\mathrm{f}}(x^*). \tag{2d}$$

A pair $(x^*, y^*) \in \mathbb{R}^{n+m}$ that satisfies (2) is called a *critical point*. In the convex case it additionally holds that every critical point is a global solution of QP (1). The primal-dual solution is furthermore unique if and only if the following two conditions are satisfied:

1. The active constraint rows A_i, $i \in \mathscr{A}^{\mathrm{e}} \cup \mathscr{A}^{\mathrm{l}} \cup \mathscr{A}^{\mathrm{u}}$, are linearly independent.
2. Matrix H is positive definite on the null space of the strictly active constraints.

For the nonconvex case, critical points are not necessarily local minima. To verify local optimality, we can use a sufficient condition of second order. We specialize a result from nonlinear constrained optimization [20, Theorem 9.3.1 and the following remark] to the QP case: Let $(x^*, y^*) \in \mathbb{R}^{n+m}$ be a critical point and define the set of strongly active inequality constraints

$$\mathscr{A}^{\mathrm{l}}_{+}(x^*, y^*) = \mathscr{A}^{\mathrm{l}}(x^*) \cap \{i \in \overline{m} \mid y_i^* > 0\},$$
$$\mathscr{A}^{\mathrm{u}}_{-}(x^*, y^*) = \mathscr{A}^{\mathrm{u}}(x^*) \cap \{i \in \overline{m} \mid y_i^* < 0\}.$$

Then x^* is a strict local minimizer of (1), if

$$s^T H s > 0 \quad \forall s \in \mathbb{R}^n \quad \text{with } s \neq 0,\ A_i s = 0,\ i \in \mathscr{A}^{\mathrm{e}}(x^*) \cup \mathscr{A}^{\mathrm{l}}_{+}(x^*, y^*) \cup \mathscr{A}^{\mathrm{u}}_{-}(x^*, y^*). \tag{3}$$

1.3 Existing methods

A great variety of methods for solving QP problems exists. Many of them can be categorized into one of two main families, namely *active-set* and *interior-point* methods, but other approaches, such as fast gradient methods, exist and have important applications, see for example [46].

Interior-point methods were initially developed for linear programming [39] and were later extended to convex quadratic and general nonlinear programming. They are mainly used in two different variants: primal barrier methods and primal-dual methods. Barrier methods replace the inequality constraints (1b) of the QP problem by a weighted barrier function in the objective. This barrier function is constructed such that it becomes (infinitely) expensive to violate the constraints; typically, a logarithmic barrier is used. The resulting equality constrained nonlinear problem is then solved by Newton's method. In order to ensure convergence to the solution of the original QP problem, this procedure is repeated with a decreased weight on the barrier function. That way, IP methods follow the so-called *central path*, a nonlinear path from a strictly feasible point towards the solution. If the weight is changed only moderately in each step of the outer loop, then the inner Newton iterations can be shown to always remain in the region of quadratic convergence and thus solve the inner problem within very few iterations, cf. [59]. A polynomial runtime guarantee can be given for certain update schemes of the weight, as shown in [47]. Primal-dual IP methods combine the inner and outer loop of primal barrier methods by reducing the weight on the barrier in each iteration of Newton's method.

Active-set methods were originally developed as extensions of the simplex method for solving LP problems [11,58]. The fundamental idea of all active-set methods is to fix a *working set*, a maximal linearly independent subset of the active constraints, and to solve the resulting equality constrained QP problem. The working set is then updated repeatedly until optimality is reached. Active-set methods can be divided into primal, dual, and parametric methods. Once a feasible starting point has been found,

primal active-set methods generate a sequence of primal feasible iterates until dual feasibility and hence an optimal solution is obtained [48]. If no feasible starting point is available, a so-called *Phase I* is employed to generate one or to detect infeasibility; see e.g. [20]. *Dual* active-set methods for convex QP problems generate a sequence of dual-feasible iterates until primal feasibility and hence an optimal solution is obtained. In the strictly convex case, this is equivalent to solving the dual of the QP problem (1) with a primal active-set method [28].

The numerical behavior of active-set and interior-point methods is usually quite different: While active-set methods need on average substantially more iterations than interior-point methods, each active-set iteration is computationally much cheaper. Often, one or the other method will perform favorably on a certain problem instance, indicating that both approaches are important. An advantage of active-set methods is the possibility to warm-start or hot-start[2] the iterations when solving a sequence of related QP problems, which can lead to substantial speed-ups. Contrary to primal active-set or primal barrier methods, dual active-set methods do not require a possibly expensive Phase I.

An up-to-date list of available quadratic programming codes can be found on the web page [31].

A variant of the active-set method that has received comparably little attention are *parametric* active-set methods, which are centered around the idea of tracing the solution of a linear homotopy parameterized by $\tau \in [0, 1]$ between a QP problem with known solution ($\tau = 0$) and the QP problem to be solved ($\tau = 1$),

$$\min_{x(\tau)\in\mathbb{R}^n} \ \tfrac{1}{2}x(\tau)^T H x(\tau) + g(\tau)^T x(\tau) \tag{4a}$$

$$\text{s. t.} \quad a^{\mathrm{l}}(\tau) \le Ax(\tau) \le a^{\mathrm{u}}(\tau). \tag{4b}$$

If $g(\tau)$, $a^{\mathrm{l}}(\tau)$, and $a^{\mathrm{u}}(\tau)$ are affine-linear functions of the homotopy parameter τ, it can be shown that the optimal solutions $x(\tau)$ depend piecewise affine-linearly on τ. The algorithm considered in this paper has been proposed in [7] in the form of the primal-dual *Parametric Quadratic Programming (PQP) method*, and was later adapted for use in model predictive control (MPC), see [19].

In MPC, one is interested in the on-line optimization of a dynamic process over a prediction horizon in time,

$$\min_{\substack{v\in\mathbb{R}^{n_{\mathrm{v}}\times(N+1)},\\ u\in\mathbb{R}^{n_{\mathrm{u}}\times N}}} \ \frac{1}{2}\sum_{k=0}^{N-1}\left(\|v_k - v_k^{\mathrm{r}}\|_{Q_k}^2 + \|u_k - u_k^{\mathrm{r}}\|_{R_k}^2\right) + \frac{1}{2}\|v_N - v_N^{\mathrm{r}}\|_P^2 \tag{5a}$$

$$\text{s. t.} \quad v_0 = v_{\mathrm{est}}, \tag{5b}$$

$$v_{k+1} = A_k v_k + B_k u_k + c_k \quad \forall\, k \in \{0, \ldots, N-1\}, \tag{5c}$$

$$d_k^{\mathrm{l}} \le C_k v_k + D_k u_k \le d_k^{\mathrm{u}} \quad \forall\, k \in \{0, \ldots, N-1\}, \tag{5d}$$

$$d_N^{\mathrm{l}} \le C_N v_N \ \le \ d_N^{\mathrm{u}}. \tag{5e}$$

[2] We use the notion *warm-start* if the QP solution procedure is initialized based on the solution of the previous QP problem. *Hot-start* refers to the case where also internal matrix factorizations are re-used.

Here, the process is described by the discrete-time dynamic system (5c) on N time intervals $[t_i, t_{i+1}]$, $0 \leq i \leq N-1$, which defines the vector $v = (v_0, \ldots, v_N) \in \mathbb{R}^{n_v \times (N+1)}$ of state predictions. The process is affected by a sequence of future control inputs $u = (u_0, \ldots, u_{N-1}) \in \mathbb{R}^{n_u \times N}$ to be determined such that a convex objective function (5a) is minimized subject to process constraints (5d)–(5e) that must be satisfied.

Problem (5) is a quadratic problem that depends parametrically on the initial state vector $v_{\text{est}} \in \mathbb{R}^{n_v}$. Using the dynamic equations (5c) to eliminate the process states v from the problem yields the following equivalent QP problem:

$$\min_{u(v_0) \in \mathbb{R}^n} \ \tfrac{1}{2} u(v_0)^T H u(v_0) + g(y_0)^T u(v_0) \tag{6a}$$

$$\text{s. t.} \quad a^{\text{l}}(v_0) \leq A u(v_0) \leq a^{\text{u}}(v_0), \tag{6b}$$

which can be easily re-parameterized to yield the standard form (4) of a parametric QP problem. The current state vector v_0 is repeatedly estimated from real-world measurements, and at each sampling instant problem (6) is solved on-line to find the optimal *feedback control* $u_0 \in \mathbb{R}^{n_u}$. This optimized control is then used to control the process, until the next, more recent feedback control has been computed from the next state observation. As measurements of v_{est} typically vary slowly, one can expect the optimal solution of the parametric QP problem (6) to also change only moderately with time. Therefore, it is beneficial to exploit the parametric hot-starting capabilities of active-set methods for MPC in order to meet the hard real-time constraints on the available on-line computation time, see [12,19] for further details.

1.4 Structure and contribution of this article

This article presents the open-source software package qpOASES, first mentioned in [14,19], in its most recent release 3.0. In this paper, the new release of qpOASES is shown to constitute a reliable and efficient, object-oriented C++ implementation of a parametric primal-dual active-set method. Section 2 starts by recalling the parametric quadratic programming algorithm as described in [7]. In Sect. 3 we describe new algorithmic extensions that were introduced in release 3.0 of qpOASES. We elaborate on the mathematical background of these extensions and present numerical details of the implementation details. Section 4 outlines the object-oriented design of the C++ implementation in more detail. Small code examples for typical use cases of qpOASES are given, and we mention possibilities for extending the functionality of qpOASES by deriving from existing C++ classes, providing an opportunity to tailor important parts of the code to special problem structures. Section 4.5 discusses important parameters of the algorithm and the qpOASES code. They are based on numerical improvements that allow a trade-off between efficiency and reliability of the algorithm, tailored to the numerical characteristics of the specific QP problem to be solved. Section 5 contains details about interfaces to various third-party software packages and mentions a number of applications of qpOASES to real-world problems, partly on embedded computer hardware. We show in Sect. 6 that qpOASES performs

competitively with popular academic and commercial QP solvers on small- to medium-scale test examples. We also present results of qpOASES finding stationary points of nonconvex QP problems. In addition, we discuss how the homotopy framework can be efficiently exploited for hot-starting QP problems within a given sequence of problems, for example, QP problems arising in model predictive control applications. We briefly address in Sect. 7 how to obtain a copy of qpOASES and how to install it.

2 Algorithm

In this section we describe the PQP method due to [7], which forms the basis for the new algorithmic developments that we discuss in Sect. 3. The choice of matrix decompositions for the linear algebra employed in the computation of step directions is entirely decoupled from the PQP method itself. We briefly address the null-space method implemented in qpOASES and give details on the handling of QP problems with sparse matrix data.

From now on, we generally assume convexity, i.e., positive semi-definiteness of H, unless explicitly stated otherwise.

2.1 The parametric programming paradigm

The idea behind parametric active-set methods is to trace optimal solutions on a homotopy path between two QP instances, parameterized by $\tau \in [0, 1]$. Denote the set of affine-linear functions from $[0, 1]$ to $\mathbb{R}^k$ by

$$\mathscr{H}^k = \{f : [0, 1] \to \mathbb{R}^k \mid f(\tau) = (1 - \tau) f(0) + \tau f(1),\ \tau \in [0, 1]\}, \tag{7}$$

and let $g(\tau) \in \mathscr{H}^n$, $a^{\mathrm{l}}(\tau), a^{\mathrm{u}}(\tau) \in \mathscr{H}^m$. We are interested in the solution of the one-parametric family of QP problems

$$\min_{x(\tau)\in\mathbb{R}^n} \ \tfrac{1}{2} x(\tau)^T H x(\tau) + g(\tau)^T x(\tau) \tag{8a}$$

$$\text{s. t.} \quad a^{\mathrm{l}}(\tau) \le A x(\tau) \le a^{\mathrm{u}}(\tau). \tag{8b}$$

For fixed τ, denote an optimal primal-dual solution by $z(\tau) = (x(\tau), y(\tau))$, where $y(\tau)$ are the Lagrange multipliers of the inequality constraints (8b). It can be shown that optimal solutions $z(\tau)$ depend piecewise linearly but not necessarily continuously on τ, see [7]. The active set is constant on each linear segment. Parametric active-set algorithms follow $z(\tau)$ by jumping from the beginning of one segment to the next, updating the working set $W(\tau)$ accordingly. We take the liberty of dropping the argument τ whenever this does not create ambiguities.

We can immediately observe that this approach allows in a natural way to *hot-start* the QP solver after modifications to the QP vectors. In addition, no Phase 1 is needed to begin with. We could always recede to the homotopy start $g(0) = 0$, $a^{\mathrm{l}}(0) = 0$, $a^{\mathrm{u}}(0) = 0$, $x(0) = 0$, $y(0) = 0$, although better choices are discussed in Sect. 3.

2.2 The classic parametric quadratic programming method

A parametric active-set method was first described by [7] for convex QP under the name *parametric quadratic programming algorithm*. We restate it here using the notation of problem (8).

1. Start with an optimal primal-dual solution $z(0) = (x(0), y(0))$ and associated working set $W(0) \in \{-1, 0, 1\}^m$ of the previously solved QP. Let $\tau := 0$.
2. Determine the step direction $\Delta z = (\Delta x, \Delta y)$ using the current working set $W(\tau)$.
3. Determine the maximum homotopy step length $\Delta\tau$ and possibly the index of a blocking constraint, l, or a blocking multipliers sign change, k.
4. If $\Delta\tau \geq 1 - \tau$, then stop with $z(1) := z(\tau) + (1 - \tau)\Delta z$ as the solution of (8).
5. Set $\tau^+ := \tau + \Delta\tau$, $z(\tau^+) := z(\tau) + \Delta\tau\Delta z$, and $W(\tau^+) := W(\tau)$.
6. If constraint l is blocking:
   (a) Set $W(\tau^+)_l := \pm 1$ (depending on whether an upper or lower constraint is blocking).
   (b) If the new working set $W(\tau^+)$ is linear dependent, find an exchange constraint index k or stop due to infeasibility of (8) for all $\tau > \tau^+$. Adjust dual variables $y(\tau^+)$ and set $W(\tau^+)_k := 0$.
7. If sign change of $y(\tau^+)_k$ is blocking:
   (a) Set $W(\tau^+)_k := 0$.
   (b) If H has nonpositive curvature on the null-space of $W(\tau^+)$, find an exchange constraint index l or stop due to unboundedness of (8) for all $\tau > \tau^+$. Adjust primal variables $x(\tau^+)$ and set $W(\tau^+)_l := \pm 1$.
8. Set $\tau := \tau^+$ and $W(\tau) := W(\tau^+)$. Continue with step 2.

Several steps in this algorithm deserve a more detailed explanation.

Computing the step direction (Step 2)

Denote by $a_W(\tau)$ the vector that consists of entries $a_i^{\mathrm{l}}(\tau)$ or $a_i^{\mathrm{u}}(\tau)$ depending on what bounds (if any) are marked active in $W_i(\tau)$. We can then determine the step direction $(\Delta x, \Delta y)$ by solving

$$K_W \begin{pmatrix} \Delta x \\ -\Delta y_W \end{pmatrix} = \begin{pmatrix} H & A_W^T \\ A_W & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ -\Delta y_W \end{pmatrix} = \begin{pmatrix} -(g(1) - g(\tau)) \\ a_W(1) - a_W(\tau) \end{pmatrix} \tag{9}$$

and letting $\Delta y_i = 0$ if $W_i(\tau) = 0$. Given invertibility of the matrix K_W for $W(0)$, see Sect. 3.2, steps 6 and 7 ensure that this property is maintained for all $\tau > 0$. More details on the numerical linear algebra involved in solving (9) can be found in Sect. 2.3.

Determining the homotopy step length (Step 3)

We can follow $z(\tau)$ in direction Δz along the current segment until either an inactive constraint becomes active (primal blocking), or a dual variable of an active constraint changes its sign (dual blocking).

The step length $\Delta\tau$ onto the first blocking constraint can be determined by *ratio tests*,

$$\mathrm{RT} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R} \cup \{\infty\}, \quad (u, v) \mapsto \min\{u_i/v_i \mid 1 \leq i \leq m, v_i > 0\}. \quad (10)$$

The minimum yields ∞ by convention if the set of ratios is empty. With the help of Eq. (10), the maximum step towards the first blocking constraint is given by

$$\Delta\tau_\mathrm{p} = \min\left\{\mathrm{RT}\left(Ax(\tau) - a^\mathrm{l}(\tau), -A\Delta x\right), \mathrm{RT}\left(a^\mathrm{u}(\tau) - Ax(\tau), A\Delta x\right)\right\} \quad (11)$$

and towards the first blocking sign change by

$$\Delta\tau_\mathrm{d} = \mathrm{RT}(W(\tau) \circ y(\tau), W(\tau) \circ \Delta y), \quad (12)$$

There might be more than one limiting blocking constraint or sign change. This situation, referred to as a *tie*, will be addressed in Sect. 3. The maximum step allowed $\Delta\tau = \min\{\Delta\tau_\mathrm{p}, \Delta\tau_\mathrm{d}\}$, however, is unique.

Determining linear dependence and an exchange constraint (Step 6)

The new working set $W^+ := W(\tau^+)$ is formed by addition of a new constraint l to the working set W, which can lead to rank deficiency of the matrix A_{W^+} and thus loss of invertibility in Eq. (9). We can check for linear dependence of A_l on A_W by solving

$$\begin{pmatrix} H & A_W^T \\ A_W & 0 \end{pmatrix} \begin{pmatrix} p \\ q_W \end{pmatrix} = \begin{pmatrix} A_l^T \\ 0 \end{pmatrix}. \quad (13)$$

This test can be carried out cheaply by reusing the factorization employed to solve (9). A_l is linearly dependent on A_W if and only if $p = 0$. In this case, to determine an exchange constraint resolving linear dependency, let q be constructed from q_W like Δy from Δy_W. Equation (13) then yields

$$0 = -A_l^T + \sum_{i:W_i \neq 0} q_i A_i^T. \quad (14)$$

Multiplying Eq. (14) by λW_l^+ with an arbitrary $\lambda \geq 0$ and adding this as a special form of zero to the stationarity condition in Eqs. (2a) yields

$$\begin{aligned} H(x(\tau) + \Delta\tau\Delta x) + g(\tau^+) &= \sum_{i:W_i \neq 0} y_i(\tau^+) A_i^T \\ &= -\lambda W_l^+ A_l^T + \sum_{i:W_i \neq 0} (y_i(\tau^+) + \lambda W_l^+ q_i) A_i^T. \end{aligned} \quad (15)$$

All coefficients of $A_i, i : W_i^+ \neq 0$ on the right hand side of Eq. (15) are also valid choices $\tilde{y}$ for the dual variables as long as they satisfy the sign condition $W_i^+ \tilde{y}_i \leq 0$. Hence, we need to compute the index k and multiplier λ of the exchange constraint. For this, we obtain q from q_W by letting $q_i = 0$ if $W_i^+ = 0$, and carry out the ratio test

$$\lambda^\star := \mathrm{RT}\left(-W_l^+(W \circ y(\tau^+)), W_l^+(W \circ q)\right), \tag{16}$$

If $\lambda^\star = \infty$, then the parametric QP is infeasible beyond τ^+, in particular in $\tau = 1$ by convexity of the feasible set. Otherwise, let k be a minimizing index of the ratio set and let $y(\tau^+) := \tilde{y}$ where

$$\tilde{y}_l = -\lambda^\star W_l^+, \quad \tilde{y}_i = y_i(\tau^+) + \lambda^\star W_i^+ q_i \quad \text{for} \quad i : W_i \neq 0. \tag{17}$$

Furthermore, $\tilde{y}_k$ vanishes by construction of $\lambda^\star$. Removing constraint k from W^+ restores linear independence. For a proof, we refer to [7].

Determining zero curvature and an exchange constraint (Step 7)

The new working set W^+ is formed by removal of a constraint from W. This may lead to exposition of a direction of zero curvature in the null-space of A_{W^+}, which has higher dimension than the null-space of A_W, again leading to loss of invertibility in Eq. (9). Directions of zero curvature can be detected by solving

$$\begin{pmatrix} H & A_W^T \\ A_W & 0 \end{pmatrix} \begin{pmatrix} s \\ \xi_W \end{pmatrix} = \begin{pmatrix} 0 \\ -(e_k)_W \end{pmatrix}, \tag{18}$$

where $e_k \in \mathbb{R}^m$ denotes the k-th unit vector. H is singular on the null-space of A_{W^+} if and only if $\xi_W = 0$, see [7]. Then, s solves

$$Hs = 0, \quad A_k s = -1, \quad A_{W^+} s = 0 \tag{19}$$

and all points $\tilde{x} = x(\tau^+) + \sigma s, \sigma > 0$ are also optimal solutions if $\tilde{x}$ is primal feasible. We can determine the largest such $\sigma = \min\{\sigma^\mathrm{l}, \sigma^\mathrm{u}\}$ from the ratio tests

$$\sigma^\mathrm{l} = \mathrm{RT}(Ax(\tau^+) - a^\mathrm{l}, -As), \quad \sigma^\mathrm{u} = \mathrm{RT}(a^\mathrm{u} - Ax(\tau^+), As). \tag{20}$$

If $\sigma = \infty$, then the parametric QP is unbounded beyond τ^+, and in particular in $\tau = 1$. Otherwise, let l be a minimizing index of a ratio set that delivered the minimizer σ, and let $x(\tau^+) := x(\tau^+) + \sigma s$. By construction of σ, the constraint row l is active in $x(\tau^+)$ and can be added to the working set via $W_l^+ := \pm 1$. Again, we refer to [7] for a proof.

As for the linear independence test, the nonzero curvature test can be evaluated cheaply by reusing the factorization computed for Eq. (9). Note that this procedure only guarantees regularity of $Z^T H Z$, but not existence of a Cholesky factor if H is

not positive semidefinite. An extension to the case of directions of negative curvature is addressed in Sect. 3.

2.3 Linear algebra

The factorization of K_W in Sect. 2.2 is the basis for determination of the step direction, the linear independence test, and the nonzero curvature test. In this section, we describe the computation of dense null-space factors from both dense and sparse matrices, the exploitation of simple bounds on the QP variables, and quicker version of linear independence tests.

2.3.1 Null-space factorization

In qpOASES, for solving the saddle-point problem K_W we implement a null-space method that can be expected to be numerically stable [6]. In each step, to exploit active simple bounds on the QP variables, we permute and partition $\Delta x = (\Delta x_X, \Delta x_F)$ into fixed and free variables according to the working set W, and $\Delta y_W = (\Delta y_A, \Delta y_X)$ into multipliers for active linear constraints and fixed simple bounds in W. Gradient $g = (g_X, g_F)$ and residual $a = (a_A, a_X)$ as well as Hessian H and constraints submatrix $\tilde{A}$ (see Sect. 1.1) are permuted and partitioned accordingly,

$$H = \begin{pmatrix} H_X & H_M^T \\ H_M & H_F \end{pmatrix}, \quad \tilde{A} = \begin{pmatrix} A_X & A_F \end{pmatrix}. \tag{21}$$

We use the representation $x_F = Z x_Z + Y x_Y$ based on a TQ factorization $A_F(Z\ Y) = (0\ T)$ of the working set constraints matrix, wherein Z and Y are column-orthonormal bases of the null-space and the range-space of A_F, and T is southeast triangular (as proposed in [26] to facilitate matrix updates). Finally, a Cholesky factorization $R^T R = Z^T H_F Z$ of the null-space projection of the Hessian yields

$$\begin{pmatrix} H_X & H_M^T Y & H_M^T Z & A_X^T & I \\ Y^T H_M & Y^T H_F Y & Y^T H_F Z & T^T & \\ Z^T H_M & Z^T H_F Y & R^T R & & \\ A_X & T & & & \\ I & & & & \end{pmatrix} \begin{pmatrix} \Delta x_X \\ \Delta x_Y \\ \Delta x_Z \\ -\Delta y_A \\ -\Delta y_X \end{pmatrix} = \begin{pmatrix} -g_X \\ -Y^T g_F \\ -Z^T g_F \\ a_A \\ a_X \end{pmatrix}, \tag{22}$$

to be solved by block-wise backward substitution. Herein, the right hand side vectors g_X, g_F, a_A, and a_X are generic, to be replaced by the appropriate right hand sides in step 2, step 6b, and step 7b of the PQP algorithm.

2.3.2 Fast linear independence tests

During the linear independence tests, see step 6, a backsolve with the special right hand side $a_W = (a_A, a_X) = 0$ has to be carried out. Hence $(\Delta x_X, \Delta x_Y) = 0$ in (22) and we obtain a test of a nonzero component of $\Delta x_Z = -(R^T R)^{-1} Z^T g_F$ in which

we may safely ignore the basis transformation $(R^T R)^{-1}$. Hence, as mentioned in [14], step 6 may be carried out by simply looking for a nonzero entry in the matrix-vector product $Z^T A_l^T$ in (13).

2.3.3 Special shapes of the Hessian and constraint matrices

In the code qpOASES we automatically detect and exploit special structure of the Hessian matrix H: diagonal and unit diagonal Hessian matrices. In the first case, the Cholesky factorization $R^T R = H_F$ is trivial. In the second case, the KKT system (22) reduces to the much simpler system (23),

$$\begin{pmatrix} I & & A_X^T & I \\ & I & T^T & \\ & & I & \\ A_X & T & & \\ I & & & \end{pmatrix} \begin{pmatrix} \Delta x_X \\ \Delta x_Y \\ \Delta x_Z \\ -\Delta y_A \\ -\Delta y_X \end{pmatrix} = \begin{pmatrix} -g_X \\ -Y^T g_F \\ -Z^T g_F \\ a_A \\ a_X \end{pmatrix} \tag{23}$$

that can be factorized and solved at greatly reduced cost. For the case of bound-constrained QP problems, i.e. problems that do not comprise linear constraints, the KKT system (22) reduces to

$$\begin{pmatrix} H_X & H_M^T & I \\ H_M & R^T R & \\ I & & \end{pmatrix} \begin{pmatrix} \Delta x_X \\ \Delta x_F \\ -\Delta y_X \end{pmatrix} = \begin{pmatrix} -g_X \\ -g_F \\ a_X \end{pmatrix}, \tag{24}$$

again factorized and solved at greatly reduced cost.

2.3.4 Matrix updates

After an active-set exchange, the block KKT matrix of the working set W in (22) changes in a single row and column only. Hence, instead of recomputing this factorization with cubic runtime effort, it is possible to recover it from a previous one by orthogonal transformations and rank one updates involving only a quadratic runtime effort. This technique is central to the efficiency of active-set methods and goes back to [4,25]. It has since been investigated for different factorizations, e.g. [22,23,27,29,36,40]. A detailed presentation of the update techniques implemented in our code qpOASES can be found in [14,26] and fundamentals for null-space updates can be found in [48].

2.3.5 Sparse matrices

In the new release of qpOASES we adopt the paradigm of computing dense factors of sparse matrices, see e.g. [21]. Both the Hessian matrix H and the constraint matrix A may be stored in a compressed row or compressed column storage format to reduce the amount of memory bandwidth and floating-point operations required during computa-

tion of the dense TQ and Cholesky factors as well as during formation of matrix-vector products with H and A.

2.3.6 Limitations and alternatives

The dense factorizations implemented in qpOASES yield satisfactory runtimes only for problems that are not too large in terms of the numbers n of unknowns and m of constraints. The current implementation can be expected to show satisfactory performance for (typically dense) problems with up to about 1,000 unknowns and constraints. Obviously, any exploitation of special structures arising in the Hessian or constraints matrix that extends beyond sparse representation of the QP matrices would result in significant speed-ups. One prominent example is a *condensing* preprocessing step that exploits block structures arising in optimal control and MPC problems, see [9,42], before passing a significantly smaller QP on to the solver. An alternative is the direct exploitation of such structures in a block structured KKT solve in place of (22), see, for example, [5,40]. Finally, the effective treatment of large-scale sparse QPs with an active-set method is only possible using sparse factorizations with suitable update schemes such as those presented in [22,36].

3 New algorithmic developments in qpOASES

In this section we describe the numerical challenges that occur in the PQP algorithm and present countermeasures implemented in qpOASES. Although these challenges are closely interconnected, they can be roughly broken up into a few individual issues: We address the treatment of equality constraints in Sect. 3.1, the choice of the initial working set in Sect. 3.2; the far bounds strategy in Sect. 3.3; rounding errors and ill-conditioning in Sects. 3.4, 3.5, 3.6, and 3.7; comparison with zero in Sects. 3.8 and 3.9; and cycling and ties in Sect. 3.10. These algorithmic developments address the reliability of the PQP algorithm and extend its applicability to nonconvex QP problems. Finally, an alternative strategy for handling positive semidefiniteness of the Hessian matrix as well as an extension of the homotopy framework to varying QP matrices are mentioned. We also relate the new algorithmic developments to caller-accessible parameters of qpOASES solver options.

3.1 Equality constraints

Many QPs contain equality constraints of the form $a_i^{\mathrm{l}} = a_i^{\mathrm{u}}$. Obviously, constraint i must be contained in the optimal active set. If the considered QP is non-degenerate, it must also be in the optimal working set, and $W_i \neq 0$. It may hence be beneficial to exclude W_i from modifications in order to reduce the combinatorial complexity of the active set identification. In the degenerate case, however, it may be necessary to keep constraint i out of the working set if, for example, there are more equalities than variables. qpOASES considers constraints with lower and upper bound differing by less than `boundTolerance` as equalities. Disabling the switch `enableEqualities` leads to relaxation of equalities to $a_i^{\mathrm{l}}(\tau) \leq Ax(\tau) \leq a_i^{\mathrm{u}}(\tau)$ in the initial point $\tau = 0$

of the homotopy, and enforces $a_i^{\mathrm{l}}(1) = a_i^{\mathrm{u}}(1)$ only. The amount of relaxation is set through `boundRelaxation` and may be further changed if ramping is enabled. This may facilitate the choice of a numerically more stable initial working set, e.g., variable bounds only.

3.2 Initial working set

Step 2 of Sect. 2.2 requires the matrix K_W to be regular for the initial working set $W(0)$. Suitable choices may be computed by the caller, using, for example, crashing techniques, see [60] for example. In the event that the caller of `qpOASES` does not provide a suitable initial $W(0)$, we choose $W(0)$ such that all simple bounds are marked active. Existence of finite bounds on every variable is ensured by the far bounds strategy, see Sect. 3.3. Then the matrix A_W is the n-by-n identity matrix, its null-space is $\{0\}$, and the projected Hessian $Z^T H_F Z$ is vacuous. In this case, simple bounds in the initial active set are active at their lower bounds by default. This may be changed by setting `initialStatusBounds` accordingly.

3.3 Far bounds

For QP problems that are missing bounds on one or more of the variables or constraints, $a_i^{\mathrm{l}} = -\infty$ or $a_i^{\mathrm{u}} = \infty$, we employ a *far bounds* strategy. It consists of clipping infinite or large entries in a_i^{l} and a_i^{u} to finite, moderately large values $-M_i^{\mathrm{l}} \approx -10^6$ and $M_i^{\mathrm{u}} \approx 10^6$, respectively. We then solve the QP with far bounds and distinguish three cases:

1. If we find a solution with no far bounds active, then this solution also solves the original QP.
2. If the QP with far bounds is infeasible, we enlarge the values M_i^{l} and M_i^{u} by a certain growth factor, for example by 10^3 and solve again.
3. If the first case has not occurred even after having grown the far bounds to very large value considered to constitute infinity, for example 10^{20}, then the original problem is declared infeasible or unbounded. We declare infeasibility if the last QP with far bounds was infeasible, and unboundedness if at least one of the far bounds is still active in the last optimal solution.

The far bounds strategy can be carried out efficiently by exploiting the parametric nature of the algorithm. We can reuse the current working set and current matrix factorizations via hot-starts for the sequence of QPs with growing far bounds (see also Sect. 6.3). We also apply the far bounds strategy to general linear constraints in order to facilitate *flipping*, as described in Sect. 3.7.

The far bounds strategy may be enabled using the switch `enableFarBounds`. The initial size of the far bounds box is `initialFarBounds`, and the growth factor is `growFarBounds`.

3.4 Iterative refinement

If the KKT matrix K_W has a high condition number, then small perturbations of the right hand side in Eq. (9) due to, for example, truncation errors, can lead to large changes in the solution Δz. As a consequence, the ratio tests (11) and (12) can become unstable, yielding different results for only small perturbations of the right hand side in equation (9). This undesirable behavior can be mitigated by, for example, reducing the forward error of Δz through iterative refinement [56]. Other possibilities for stabilizing ratio tests have been presented, in the framework of the simplex method for linear programming, by [33] and [60].

Iterative refinement can be enabled by setting `numRefinementSteps` to a positive value. Early termination of refinement iterations happens if the KKT residual 2-norm falls below `epsIterRef`.

3.5 Drift correction

Rounding errors can also accumulate over several iterations and lead to a parametric "solution" $z(\tau)$ that is optimal with an accuracy significantly worse than machine precision. We call this phenomenon *drift*. Large drift can even lead to breakdown of the algorithm because the general assumption of optimality of $z(\tau)$ is violated. However, the parametric homotopy framework allows for reducing the drift to zero immediately by perturbing the current dual variables, the current constraint vectors, and the current gradient appropriately such that $z(\tau)$ is optimal again. We directly carry out this backwards analysis by requiring for $i = 1, \ldots, m$ exact primal feasibility,

$$a_i^{\mathrm{l}}(\tau) := \begin{cases} A_i x(\tau) & \text{if } W_i = -1, \\ \min(a_i^{\mathrm{l}}(\tau),\, A_i x(\tau)) & \text{if } W_i \in \{0, +1\}, \end{cases} \tag{25a}$$

$$a_i^{\mathrm{u}}(\tau) := \begin{cases} A_i x(\tau) & \text{if } W_i = +1, \\ \max(a_i^{\mathrm{u}}(\tau),\, A_i x(\tau)) & \text{if } W_i \in \{-1, 0\}, \end{cases} \tag{25b}$$

exact dual feasibility,

$$y_i(\tau) := \begin{cases} \max(y_i(\tau), 0) & \text{if } W_i = -1, \\ 0 & \text{if } W_i = 0, \\ \min(y_i(\tau), 0) & \text{if } W_i = +1, \end{cases} \tag{26}$$

and afterwards repairing stationarity via the gradient modification,

$$g(\tau) := A^T y(\tau) - H x(\tau). \tag{27}$$

In case the multiplication with A, A^T, and H is numerically at least as expensive as the remainder of the iteration, drift correction should be applied only infrequently, i.e. every $n_{\mathrm{DC}} \gg 1$ iterations.

Drift correction may be enabled by setting the option `enableDriftCorrection` to a positive value.

3.6 Termination check

A well-designed termination criterion must work reliably on both well- and ill-conditioned problems. While it is tempting to use the homotopy parameter τ in the termination criterion as done in Sect. 2.2, this choice renders the criterion dependent on the choice of the homotopy start. We instead propose to use the relative distance δ in the QP data space

$$\begin{aligned} \Delta^\tau &= (g(\tau), a^{\mathrm{l}}(\tau), a^{\mathrm{u}}(\tau)), \\ s_j &= (\Delta_j^{\mathrm{l}} - \Delta_j^\tau)/\max\{1, |\Delta_j^{\mathrm{l}}|\}, \quad j = 1, \ldots, n+2m, \qquad \delta = \|s\|_\infty, \end{aligned} \tag{28}$$

which yields a termination criterion that is independent of $\mathrm{cond}(K_W(1))$. This modified termination criterion does not give a guarantee for the distance to the exact solution. Instead, a backwards analysis result holds: The computed solution is the exact solution to a perturbed QP that deviates by no more than δ from the one to be solved.

The threshold of the condition-independent termination check may be set through `terminationTolerance`.

3.7 Flipping bounds

Flipping bounds is a strategy similar to long steps in the dual simplex method [41,51], where one variable changes in the working set from upper to lower bound immediately without becoming inactive in between, i.e., it *flips*. Flipping bounds only need to be employed if H is indefinite or positive semidefinite.

Obviously, flipping is only possible if $a_i^{\mathrm{l}}(1)$ and $a_i^{\mathrm{u}}(1)$ have finite entries. We ensure finiteness of the constraint vectors via the far bounds strategy, see Sect. 3.3.

We then perform flipping in the following way: If we remove a constraint l from the active set and no other constraint k enters, we monitor the size of the smallest diagonal entry r_i of the Cholesky factor R that grows by a row and column. If for some $\delta_{\mathrm{curv}} > 0$ it holds that $r_i^2 < \delta_{\mathrm{curv}}$, then we introduce a jump in the QP homotopy by moving the opposite bound of constraint l such that it becomes active immediately. This is easily achieved by setting

$$a_l^{\mathrm{l}}(\tau^+) := a_l^{\mathrm{u}}(\tau^+), \;\; W_l^+ := -1, \qquad \text{if } W_l = +1, \tag{29a}$$

$$a_l^{\mathrm{u}}(\tau^+) := a_l^{\mathrm{l}}(\tau^+), \;\; W_l^+ := +1, \qquad \text{if } W_l = -1. \tag{29b}$$

Consequently, the Cholesky decomposition from the previous step stays valid for the current projected Hessian. We can thus successfully avoid small diagonal entries in the Cholesky factors R.

We employ the same strategy if the Cholesky decomposition breaks down due to a negative eigenvalue in the projected Hessian $Z^T H_F Z$. This allows one to move

along the direction of negative curvature onto the opposite bound and hence permits the treatment of nonconvex QP problems, for which we identify a parametric path of critical points.

The flipping bounds strategy can be enabled using the switch `enableFlippingBounds`; the threshold δ_{curv} for the Cholesky diagonal elements is `epsFlipping`.

3.8 Ratio tests

In the ideal ratio test (10) we take a minimum over a subset of k quotients with strictly positive denominators. The presence of round-off error however makes it necessary to substitute this ideal ratio test by an expression with adjustable tolerances,

$$\begin{aligned} u_i^{\text{cut}} &= \max(u_i, 0), \quad i = 1, \ldots, k, \\ \text{RT}_{\text{r}}(u, v, \varepsilon_{\text{den}}, \varepsilon_{\text{num}}) &= \min\{u_i^{\text{cut}}/v_i \mid i = 1, \ldots, k, v_i \geq \varepsilon_{\text{den}}, u_i^{\text{cut}} \geq \varepsilon_{\text{num}}\}. \end{aligned} \tag{30}$$

The *denominator tolerance* $\varepsilon_{\text{den}} > 0$ is a threshold to consider tiny positive values v_i to be equal to zero. Such values are discarded as candidates for the minimum.

Negative numerators are cut off at zero before the quotients are taken, yielding a nonnegative minimum. For the ratio tests for determination of the step length (11) and (12), this choice is motivated by the fact that, in exact arithmetic, $u_i \geq 0$ for all $i = 1, \ldots, k$ with $v_i > 0$ holds. Hence, only values u_i that are negative due to round-off are manipulated, and the step length satisfies $\Delta\tau \geq 0$ also in finite precision arithmetic.

The tolerances ε_{num} and ε_{den} for numerators and denominators in ratio tests may be set using the options `epsNum` and `epsDen`, respectively.

3.9 Linear independence test and nonzero curvature test

After solving system (13) for (p, q_W) or system (18) for (s, ξ_W), we must compare the norm of p or ξ_W with zero to detect linear dependence of A_W or a direction of zero curvature in its null-space, respectively. We use the relative conditions

$$\|p\|_\infty \leq \varepsilon_{\text{LI}} \|\zeta_q\|_\infty, \ \zeta_q^T = (p^T, q^T) \quad \text{for the linear independence test, and} \tag{31a}$$

$$\|\xi_W\|_\infty \leq \varepsilon_{\text{NZC}} \|\zeta_s\|_\infty, \ \zeta_s^T = (s^T, \xi_W^T) \quad \text{for the nonzero curvature test.} \tag{31b}$$

We also remark that $\|\zeta_q\|_\infty = \|q\|_\infty$ if $p = 0$ and $\|\zeta_s\|_\infty = \|s\|_\infty$ if $\xi_W = 0$. In the implementation, we may thus replace $\|\zeta_q\|_\infty$ by $\|q\|_\infty$ for test (31a) and $\|\zeta_s\|_\infty$ by $\|s\|_\infty$ for test (31b). If we perform the cheaper linear independence test (see Sect. 2.3.2), we compute $\tilde{p}^T = A_l Z$. We declare linear independence of constraint l from the constraints in W if there exists an index i such that

$$|\tilde{p}_i| > \varepsilon_{\text{LI}} \|A_l^T\|_2^2. \tag{32}$$

Users may choose between the full variant (13) and the fast variant of Sect. 2.3.2 by toggling the switch `enableFullLITests`. The threshold $\varepsilon_{\mathrm{LI}}$ for p is `epsLITests`, and invoked dual jumps $\lambda^\star$ must be smaller than `maxDualJump`. Nonzero curvature tests may be enabled through the switch `enableNZCTests`. The threshold $\varepsilon_{\mathrm{NZC}}$ for ξ_W is `epsNZCTests`, and invoked primal jumps σ must be smaller than `maxPrimalJump`.

3.10 Ties and ramping

Instead of trying to treat ties (see Sect. 2.2) rigorously, which may be as computationally expensive as solving the QP itself [55], we try to avoid them in the first place: Let a homotopy start $g(0), a^{\mathrm{l}}(0), a^{\mathrm{u}}(0)$ with optimal solution $(x(0), y(0))$ and working set $W(0)$ be given. Then, for every triple of m-vectors $r^{(0)}, r^{(1)}, r^{(2)} \geq 0$, the primal-dual pair $(x(0), \tilde{y}(0))$ with

$$\tilde{y}_i(0) = \begin{cases} y_i(0) + r_i^{(0)} & \text{if } W_i = -1, \\ y_i(0) & \text{if } W_i = 0, \\ y_i(0) - r_i^{(0)} & \text{if } W_i = +1, \end{cases} \qquad i = 1, \ldots, m, \tag{33}$$

is an optimal solution for the homotopy start $\tilde{g}(0), \tilde{a}^{\mathrm{l}}(0), \tilde{a}^{\mathrm{u}}(0)$, where for $i = 1, \ldots, m$

$$\tilde{a}_i^{\mathrm{l}}(0) = \begin{cases} a_i^{\mathrm{l}}(0), & \text{if } W_i = -1, \\ a_i^{\mathrm{l}}(0) - r_i^{(1)}, & \text{otherwise}, \end{cases} \quad \tilde{a}_i^{\mathrm{u}}(0) = \begin{cases} a_i^{\mathrm{u}}(0), & \text{if } W_i = +1, \\ a_i^{\mathrm{u}}(0) + r_i^{(2)}, & \text{otherwise}, \end{cases} \tag{34a}$$

$$\tilde{g}(0) = A^T \tilde{y}(0) - Hx(0). \tag{34b}$$

In other words, if we move the inactive constraints' bounds further away from $Ax(0)$, and the dual variables of the active constraints further away from zero, then $x(0)$ stays feasible and $g(0)$ can be adapted to restore stationarity of $(x(0), \tilde{y}(0))$ with the same working set $W(0)$.

Recall now that the ratio tests depend on the residuals of the inactive constraints and the dual variables of the active constraints. Many QP problems exhibit special structures such as, for example, identical bounds for several variables. To avoid primal ties for such special structures, it is important to avoid picking the same value for two entries of $r^{(j)}$. Hence, for given parameters $r_{\mathrm{final}} > r_{\mathrm{initial}} > 0$, we choose a linear *ramp* shape

$$r_i^{(j)} = r_{\mathrm{initial}} \frac{m-i}{m-1} + r_{\mathrm{final}} \frac{i-1}{m-1}, \quad j = 0, 1, 2, \quad i = 1, \ldots, m. \tag{35}$$

This ramping ensures progress $\Delta\tau > 0$ because $r_i^j > 0$ for all j, and if residuals are chosen such that all ratios are pairwise different, it also ensures absence of ties in the next iteration.

The ramping strategy may be enabled using the switch `enableRamping` and performs a linear interpolation between the values `initialRamping` and `finalRamping`.

3.11 Iterative regularization procedure

For solving QP problems where the Hessian matrix is positive semidefinite, an iterative regularization procedure that makes use of the homotopy framework has been proposed in [16]. The core idea is to add a regularization term to the objective function to arrive at the following regularized QP problem:

$$\min_{x\in\mathbb{R}^n} \tfrac{1}{2}x^T Hx + g^T x + \tfrac{\varepsilon_{\text{reg}}}{2}(x-\bar{x})^T(x-\bar{x}) \tag{36a}$$

$$\text{s. t. } a^{\text{l}} \leq Ax \leq a^{\text{u}}. \tag{36b}$$

First, this problem is solved for $\bar{x} = 0$ to yield a first approximation $x^\star$ to an optimal primal solution. Next, the regularization is "re-centered" by setting $\bar{x} = x^\star$ and problem (36) is solved again. Repeating this procedure yields a special case of a proximal point algorithm for which linear convergence towards an optimal solution of the original problem has been shown [50].

It is important to note that this iterative regularization procedure perfectly fits into the homotopy framework of `qpOASES` as it requires the solution of a sequence of QP problems with varying gradients. Thus, having solved the first regularized problem, the solution of the subsequent QP problems can be hot-started and typically requires only a few, if any, additional active-set iterations.

The iterative regularization strategy may be enabled using the switch `enableRegularisation`. The number of regularization steps may be set in `numRegularisationSteps`, and the regularization constant ε_{reg} of equation (36) in `epsRegularisation`.

3.12 Warm-starts after changes to the QP matrices

During the solution of a sequence of QP problems, the parametric QP algorithm also allows for warm-starting, i.e. reuse of an available active set together with the primal-dual point, if the QP matrices H or A change [18]: Assume we have an optimal solution $z^* = (x^*, y^*)$ and an optimal working set W^* for the QP problem

$$\min_{x\in\mathbb{R}^n} \tfrac{1}{2}x^T Hx + g^T x \quad \text{s. t.} \quad a^{\text{l}} \leq Ax \leq a^{\text{u}}, \tag{37}$$

and want to solve the perturbed QP

$$\min_{x\in\mathbb{R}^n} \tfrac{1}{2}x^T \tilde{H}x + \tilde{g}^T x \quad \text{s. t.} \quad \tilde{a}^{\text{l}} \leq \tilde{A}x \leq \tilde{a}^{\text{u}}. \tag{38}$$

with new matrix data $\tilde{H},\ \tilde{A}$ and new vector data $\tilde{g},\ \tilde{a}_{\mathrm{l}},\ \tilde{a}_{\mathrm{u}}$. We may then compose a homotopy start ($\tau = 0$) for the second QP problem by setting

$$\tilde{a}^{\mathrm{l}}(0) := a^{\mathrm{l}} + (\tilde{A} - A)x^*, \tag{39a}$$

$$\tilde{a}^{\mathrm{u}}(0) := a^{\mathrm{u}} + (\tilde{A} - A)x^*, \tag{39b}$$

$$\tilde{g}(0) := g - (\tilde{H} - H)x^* + (\tilde{A} - A)^T y^*. \tag{39c}$$

and (for $\tau = 1$) let $\tilde{g}(1) = \tilde{g},\ \tilde{a}^{\mathrm{l}}(1) = \tilde{a}^{\mathrm{l}},\ \tilde{a}^{\mathrm{u}}(1) = \tilde{a}^{\mathrm{u}}$. The working set $W(0) = W^*$ remains unaltered. Alternatively, one could simply ramp an all-zero homotopy start, again keeping the working set $W(0) = W^*$, see Sect. 3.10.

As usual in sequential nonlinear methods, this approach requires the matrix $\tilde{K}_W$ for the given working set $W(0)$ to remain regular. If the initial factorization of $\tilde{K}_W$ reveals rank deficiency of $\tilde{K}_W$, we recede to a cold-start with a safe working set as described in Sect. 3.2.

4 Software design and algorithmic parameters

This section outlines the software design of qpOASES and briefly explains how to set up and solve QP problems. We also discuss its most important algorithmic parameters for adjusting the behavior of the solver as described in Sect. 3.

4.1 General remarks

qpOASES is an open-source C++ implementation of the parametric QP algorithm of Sect. 2 and all algorithmic enhancements as presented in Sect. 3. Its object-oriented design introduces separate classes for different QP problem types and encapsulates all matrix-vector operations, such that they can be easily adapted to the respective problem characteristics. Moreover, qpOASES has been implemented in a self-contained manner to facilitate compatibility with different, also embedded, hardware platforms. All matrix-vector operations within qpOASES are performed by customized C implementations of the required LAPACK/BLAS routines, but direct linking against the LAPACK or BLAS library [1,8] is also possible.

qpOASES distinguishes two different ways of solving a QP problem of the form (1). First, it can be solved by performing a cold-start, i.e. without any prior solution information. This is the usual situation if just a single QP problem is to be solved. Second, provided that a QP problem with same dimensions has been already solved before, the current QP problem can be solved by performing a hot-start based on the optimal solution and the internal matrix factorizations of the previously solved QP problem. Hot-starts can be very efficient when solving whole sequences of parameterized QP problems as arising in model predictive control or within SQP algorithms.

4.2 QP solver classes

The current release 3.0 of qpOASES comes with three different QP solver classes that offer user-interfaces for tackling different QP problem formulations:

1. The QProblem class is designed for solving a single QP of the standard form (1). It also allows the user to efficiently solve whole sequences of QP problems with varying gradient or constraint vectors by performing hot-starts based on solution information of the previous QP problem.
2. The class QProblemB offers the same functionality as QProblem, but is tailored to QP problems (or sequences) that only comprise *bounds* on the variables x, often called *box constraints*. The distinction between bounds and general constraints arises naturally in many applications, such as model predictive control, and can lead to substantial computational savings.
3. Finally, the class SQProblem extents the functionality of the QProblem class to also allow hot-starts based on the previous QP solution in case of varying QP matrices (see Sect. 3.12).

All these three classes offer user-functions for passing the QP problem data, setting algorithmic options and for obtaining results and status information. All internal data members of the QP solver are hidden from the user. For example, these classes store information on the bounds and constraints of the QP problem by means of further auxiliary classes that manage lists of free and fixed variables or active and inactive constraints.

4.3 Linear algebra classes

qpOASES has originally been developed for small to medium-scale QP problems with dense Hessian and constraint matrices. Consequently, the internal matrix factorizations have been implemented as dense linear algebra routines.

For enhancing qpOASES's applicability to sparse QP problems, a minimal Matrix base class has been introduced that encapsulates all matrix-vector operations. This framework allows the user to easily switch between special linear algebra routines for dense and sparse QP matrices, and to exploit symmetry of the Hessian. For doing so, QP matrices can be passed as either of the following types: DenseMatrix, SparseMatrix, SymDenseMat, SymSparseMat. Through the ConstraintProduct class, the user may also provide a tailored routine that directly calculates the matrix-vector product Ax.

Internally, dense matrices are stored in arrays, while sparse matrices are stored in either a row compressed or a column compressed storage format.

4.4 Solving QP problems with qpOASES

The following code fragment illustrates the calling syntax of qpOASES for solving a QP problem with 10 variables and 25 constraints. It is assumed that QP data matrices H, A, and QP data vectors g, lb, ub, lbA, and ubA of appropriate dimension have already been defined.

```
// set up QProblem object
QProblem example( 10,25 );

// solve first QP
int maxIter = 100;
example.init( H,g,A,lb,ub,lbA,ubA, maxIter );

// retrieve solution of first QP
double xOpt[10];
example.getPrimalSolution( xOpt );
```

First, the user needs to instantiate an object of the `QProblem` class by passing the dimension of QP problem. Then, the first QP problem is solved by calling the member function `init`, which takes the QP data as well as the maximum number of iterations as arguments. Afterwards, the optimal primal solution is retrieved from the QP object.

In case that a whole sequence of QP problems is to be solved, it is possible to hot-start the QP solution procedure based on solution information of the previously solved QP problem. This is done by calling the member function `hotstart` of the `QProblem` class, which takes the new QP gradient and constraint vectors as well as the maximum number of iterations as arguments:

```
// Solve second QP with new gradient and constraint vectors
maxIter = 100;
example.hotstart( g2,lb2,ub2,lbA2,ubA2, maxIter );
```

For more details on the calling syntax and for a documentation of the classes `QProblemB` and `SQProblem`, we refer to the `qpOASES` User's Manual [15]. Also information on how to specify the QP matrices in dense or sparse matrix format can be found therein.

4.5 Setting algorithmic parameters of `qpOASES`

In this subsection, we address the realization of the new algorithmic developments of Sect. 3 in our implementation `qpOASES`. We show how to set option parameters through the `qpOASES` interface, and we describe convenient pre-configured sets of options that are available for these parameters. Individual option parameters listed in Table 1, together with the default values they assume in the pre-configured sets. The respective meanings of the parameters are addressed in Sect. 3.

4.5.1 Setting options through the interface

The `qpOASES` interface provides access to many of the algorithmic details presented in Sects. 2 and 3 through the `Options` class as follows.

```
// get a copy of the current options
Options opt = example.getOptions( );

// switch to MPC preconfiguration, and relax the termination tolerance
opt.setToMPC( );
opt.terminationTolerance = 1.0e-4;
```

```
// pass the modified set of options to the solver
example.setOptions( opt );
```

4.5.2 Pre-configured sets of options

Through the `Options` class, we provide quick access to three pre-configured sets of options: default, MPC, and reliable. They can be accessed by calls to the member functions `setToDefault`, `setToMPC`, or `setToReliable`. The respective options values are listed in Table 1.

Table 1 Summary of algorithmic parameters in `qpOASES` with their respective default values in the MPC, default, and reliable pre-configurations. The constant $\varepsilon_{\text{mach}}$ denotes the machine precision

`Options` member	Section	MPC	Default	Reliable
`enableEqualities`	Section 3.1	`BT_TRUE`	`BT_FALSE`	`BT_FALSE`
`boundRelaxation`		10^4	10^4	10^4
`boundTolerance`	Section 3.2	$10^6\ \varepsilon_{\text{mach}}$	$10^6\ \varepsilon_{\text{mach}}$	$10^6\ \varepsilon_{\text{mach}}$
`initialStatusBounds`		`ST_INACTIVE`	`ST_LOWER`	`ST_LOWER`
`enableFarBounds`	Section 3.3	`BT_TRUE`	`BT_TRUE`	`BT_TRUE`
`initialFarBounds`		10^6	10^6	10^6
`growFarBounds`		10^3	10^3	10^3
`numRefinementSteps`	Section 3.4	0	1	2
`epsIterRef`			$10^2\ \varepsilon_{\text{mach}}$	$10^2\ \varepsilon_{\text{mach}}$
`enableDriftCorrection`	Section 3.5	0	1	1
`terminationTolerance`	Section 3.6	$10^9\ \varepsilon_{\text{mach}}$	$10^7\ \varepsilon_{\text{mach}}$	$10^7\ \varepsilon_{\text{mach}}$
`enableFlippingBounds`	Section 3.7	`BT_FALSE`	`BT_TRUE`	`BT_TRUE`
`epsFlipping`			$10^3\ \varepsilon_{\text{mach}}$	$10^3\ \varepsilon_{\text{mach}}$
`epsNum`	Section 3.8	$-10^3\ \varepsilon_{\text{mach}}$	$-10^3\ \varepsilon_{\text{mach}}$	$-10^3\ \varepsilon_{\text{mach}}$
`epsDen`		$+10^3\ \varepsilon_{\text{mach}}$	$+10^3\ \varepsilon_{\text{mach}}$	$+10^3\ \varepsilon_{\text{mach}}$
`enableFullLITests`	Section 3.9	`BT_FALSE`	`BT_FALSE`	`BT_TRUE`
`epsLITests`				$10^5\ \varepsilon_{\text{mach}}$
`maxDualJump`				10^8
`enableNZCTests`	Section 3.9	`BT_FALSE`	`BT_TRUE`	`BT_TRUE`
`epsNZCTests`			$3 \cdot 10^3\ \varepsilon_{\text{mach}}$	$3 \cdot 10^3\ \varepsilon_{\text{mach}}$
`maxPrimalJump`			10^8	10^8
`enableRamping`	Section 3.10	`BT_FALSE`	`BT_TRUE`	`BT_TRUE`
`initialRamping`			0.5	0.5
`finalRamping`			1.0	1.0
`esnableRegularisation`	Section 3.11	`BT_TRUE`	`BT_FALSE`	`BT_FALSE`
`numRegularisationSteps`		2		
`epsRegularisation`		$5 \cdot 10^3\ \varepsilon_{\text{mach}}$		
`enableCholeskyRefactorisation`		0	0	1

The default options set collects recommended settings for qpOASES. All vital algorithmic features are enabled, and nonconvex QP problems can be handled by flipping bounds. If no explicit calls to Options are made, this is the configuration all computations will be carried out with.

The MPC set of options is designed for model predictive control applications in which we expect to solve strictly convex problems only. It ensures that a single iteration of the active-set method is carried out as fast as possible by disabling several of the more costly algorithmic features. Moreover, by disabling ramping at the cost of a possibly increased total iteration count, this set of options ensures that the primal homotopy path generated indeed is continuous and can hence be attributed with a physical interpretation [14].

The reliable options set deviates from the default one in only a few choices. The number of iterative refinement steps has been increased to address highly ill-conditioned systems, and full linear independence test are carried out instead of the fast tests of Sect. 2.3.2. Most important, the Cholesky factorization of $Z^T H Z$ is recomputed in every iteration, thus greatly increasing the per-iteration cost of qpOASES at the benefit of increased numerical stability.

5 Interfaces and applications

The qpOASES distribution also includes several interfaces to widely-used third-party software packages in order to facilitate its use. This section gives an overview of these interfaces and outlines several real-world applications of qpOASES that have been completed in the past. For some of them qpOASES has been run on embedded computer hardware.

5.1 Interfaces for Matlab, Octave and Scilab

qpOASES can be directly used within MATLAB®, allowing a user to run the solver without touching its C++ source code. For example, a single QP can be solved by calling

```
[x,fval,exitflag,iter,lambda] = qpOASES( H,g,A,lb,ub,lbA,ubA,
                                         options,x0 );
```

Besides the usual data specifying a QP of the form (1), a set of options and an initial guess for the primal solution can be passed. If no initial guess is given, the usual homotopy starting at the origin is performed. The output arguments contain the optimal primal solution vector as well as optionally the optimal objective function value, a status flag, the number of iterations actually performed, and the optimal dual solution vector, respectively.

Options can be generated using the qpOASES_options command in order to retrieve the full functionality of the C++ version, e.g.

```
options1 = qpOASES_options( 'MPC' );
options2 = qpOASES_options( 'reliable', 'maxIter',500 );
```

for generating two sets of pre-defined algorithmic options that are tailored for fast MPC applications or most reliable QP solution, respectively. In the second case, also the default value for the maximum number of iterations is adapted.

The MATLAB® interface automatically detects whether the QP only comprises box constraints and internally instantiates the corresponding QP object. Moreover, it is possible to pass QP matrices in sparse format. This standard interface always performs a cold-start taking into account the guess for the primal solution (if specified), but also variants exist for solving whole sequences of QP problems using all features of the online active set strategy. For example, the following commands initialize a sequence of QP problems with the data of the initial (`'i'`) QP problem, perform a hot-start (`'h'`) for solving the second QP problem and clean-up (`'c'`) the internal memory of the MATLAB® interface:

```
[QP,x,fval,exitflag,iter] = qpOASES_sequence('i',H,g,A,lb,ub,lbA,ubA);
[x,fval,exitflag,iter] = qpOASES_sequence('h',QP,g2,lb2,ub2,lbA2,ubA2);
qpOASES_sequence( 'c', QP );
```

In order to support open-source alternatives to MATLAB®, `qpOASES` provides similar interfaces to OCTAVE [13] and SCILAB [52].

5.2 Interfaces to YALMIP, the ACADO Toolkit, and MUSCOD-II

`qpOASES` also has been interfaced to `YALMIP` [43], a modelling language for solving convex and nonconvex optimization problems. Moreover, `qpOASES` is the default QP solver within the open-source package `ACADO Toolkit` for automatic control and dynamic optimization [35]. For solving nonlinear optimal control problems, `MUSCOD-II` [42] can use `qpOASES` as underlying QP solver.

5.3 Running qpOASES on dSPACE® and xPC target

The `qpOASES` code also provides an interface to SIMULINK® that allows the user to compile the code within a C MEX S-function. This is particularly useful when using `qpOASES` to solve model predictive control problems. Different variants interfacing the `QProblem`, `QProblemB` and `SQProblem` class to the SIMULINK® workspace are available. The S-function block expects all QP data to be given in signal form. It outputs the first piece of the primal solution as required in the MPC context, the optimal objective function value, a status flag and the number of actually performed QP iterations.

The SIMULINK® interface also allows one to conveniently compile `qpOASES` onto dSPACE® or xPC target hardware by means of the SIMULINK® Real-Time Workshop [38]. The main requirement is the availability of a C++ compiler for the respective hardware. Compilation of `qpOASES` has been tested for dSPACE® boards version 5.3 or higher together with the dSPACE® C++ Integration Kit version 1.0.2 or higher. Also successful use on xPC target hardware has been reported, cf. Sect. 5.4.

5.4 Real-world applications

During the last few years, qpOASES has been used by a number of researchers for a wide range of applications. For example, it has been used for

- model predictive control of a Diesel engine test bench at the University of Linz, Austria, on dSPACE® hardware at sampling times of 50 ms (dense QP problems comprising about 20 bounded variables and 20 constraints) [18];
- trajectory planning for a boom crane at the University of Stuttgart, Germany, similar to the one described in [3], on dSPACE® hardware at sampling times in the order of 100 ms (about 57 bounded variables and 160 constraints);
- solving QP problems for controlling a tendon-driven robot platform at ETH Zurich, Switzerland, on a Standard PC [49];
- model predictive control of beam tip vibrations at the Slovak University of Technology, Bratislava, Slovakia, on an xPC target at sampling times of 10 ms (up to 150 bounded variables) [53];
- time-optimal control of machine tools at KU Leuven, on dSPACE® and xPC hardware at sampling times in the order of 4–10 ms (about 45 bounded variables and 182 constraints) [54].

It is worth stressing that all given QP problem dimensions refer to the dense formulation (6) and that QP solution times were typically much lower than the reported sampling time.

Moreover, qpOASES has been used within several industrial applications solving QP problems with up to a few hundred variables and a few thousand constraints. Due to confidentiality reasons, we only mention its integration into an embedded engine controller developed by Hoerbiger Control Systems AB, Sweden, to control an integral gas engine in the pipeline network of the United States [2]. The dense QP problems comprised about 10–14 variables, up to several hundred constraints and have been solved on embedded hardware with very limited computational resources (compared to a standard PC).

6 Testing

6.1 Convex QP problems

qpOASES is mainly intended for sequences of small to medium size, dense QP problems. We discuss the case of QP sequences in Sect. 6.3 but it is of course also possible to use qpOASES to solve single QP instances. Figure 1 assesses the performance of qpOASES in comparison with other QP solvers. We chose QP solvers that have an appropriate license and provide a MATLAB® interface: The interior-point code OOQP [24] and the primal and dual active-set and barrier variants of Cplex [37] (CPLEXP, CPLEXD, CPLEXB). For each solver we use the default parameters. For qpOASES we test the default (qpOASES) and the "mpc" (qpOASESmpc) parameter variants, where we changed the initial working set of the mpc method from zero to all lower bounds active (as for the default variant) in order to make useful comparisons

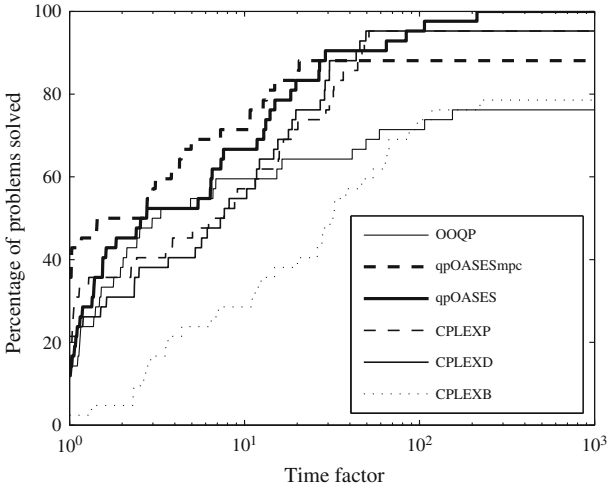

Fig. 1 Performance graph of different QP solvers on all 43 problems of the Maros-Mészáros test set [44] with $n \leq 250$, $m \leq 1{,}001$; matrices passed in dense format

between the two versions of `qpOASES` calls. Unfortunately, the Cplex solvers' interface exhibits a large calling overhead of around 1 s. In addition, the internal timing results of Cplex are rounded to 0.01 s (sometimes even down to 0 s), which makes an appropriate comparison hard to carry out. We have overcome this issue by implementing a wrapper file for the `Cplex` MEX file in C, which carries out high resolution timings but discards "constructor" call timings within the MEX file, which have been found responsible for the calling overhead.

No particular starting points were provided to the solvers. We discard the primal and dual initial values from the CUTEr interface because they are zero for all considered problems. The solvers are allowed to find suitable starting points, for example via ramping (cf. Sect. 3.2).

Figure 1 shows a performance profile, which depicts how many percent of the test set problems (ordinate) are solved by each solver within a certain time factor (abscissa) of the fastest method on each problem. Generally, fast methods have their graph in the left hand part, reliable methods in the upper part of the diagram. All 43 problems from the Maros-Mészáros test set [44] with $n \leq 250$ and $m \leq 1,001$ were used and the matrices were given to the solvers in dense format. We believe that this size is reasonable for problems that can be solved in fast real-time applications. The quality of the solutions (x^*, y^*) was measured using a residual ρ of conditions (2) defined via

$$\rho_{\text{stat}} = \|Hx^* + g - A^T y^*\|_\infty, \qquad \rho^{\text{l}}_{\text{cmpl}} = \max\{\min(|Ax^* - a^{\text{l}}|_i, y_i^*) \mid y_i^* > 0\},$$
$$\rho_{\text{feas}} = \max(0, a^{\text{l}} - Ax^*, Ax^* - a^{\text{u}}), \qquad \rho^{\text{u}}_{\text{cmpl}} = \max\{\min(|Ax^* - a^{\text{u}}|_i, -y_i^*) \mid y_i^* < 0\},$$
$$\rho = \max(\rho_{\text{stat}}, \rho_{\text{feas}}, \rho^{\text{l}}_{\text{cmpl}}, \rho^{\text{u}}_{\text{cmpl}}).$$

For some problems, some solvers terminated without indication of failure but returned a low quality solution. Here, we only consider a problem solved if $\rho \leq 10^{-4}$. We observe that `qpOASES` outperforms the other methods on this test set in terms of speed

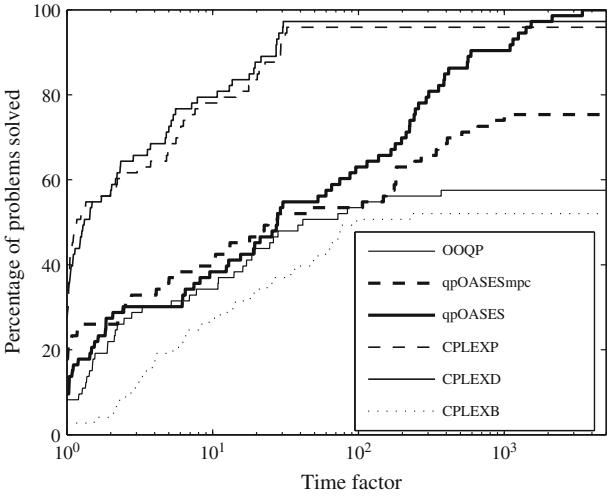

Fig. 2 Performance graph of different QP solvers on all 73 problems of the Maros-Mészáros test set [44] with $n \leq 1{,}000$, $m \leq 1{,}001$; matrices passed in sparse format

and reliability. The `mpc` version performs almost as well, even though no neighboring information of problems can be exploited for hot-starts.

Figure 2 shows the performance profiles for all 73 test problems with $n \leq 1{,}000$ and $m \leq 1{,}001$, passed to the solvers in sparse format. This size represents problems that `qpOASES` can solve in a reasonable time with its dense linear algebra subroutines. On these problems, `OOQP` and `Cplex` can exploit the sparsity of the problems efficiently. The sparse matrix vector products of `qpOASES` are still dominated by the cost of the dense linear algebra routines. In terms of speed, the active-set versions of `Cplex` are performing best. However, `qpOASES` is the most reliable method, closely followed by the active-set versions of `Cplex`. The `mpc` variant of `qpOASES` works less reliably, as expected on that kind of test collection.

Table 5 in the appendix to this paper lists the detailed results that served as the basis for the performance profiles in Figs. 1 and 2. For the `Cplex` solvers, the table only contains the results for the case that matrices were passed in sparse format, because `Cplex` seems to convert dense matrices to sparse format before starting the actual computation.

We have also tested `qpOASES` on the Netlib LP problem QAP8 ($n = 1{,}632$, $m = 912$), which is known for its difficult, almost degenerate constraints: `qpOASES` successfully converges to the solution within 23,111 iterations, while the primal and dual active-set variants of `Cplex` need 29,736 and 39,495 iterations, respectively.

6.2 Nonconvex QP problems

Due to the flipping bounds strategy, `qpOASES` can be used to find critical points of nonconvex QP problems. In Table 2 we show the number of iterations for a few test problems from the CUTEr collection [32]. We also check the second order sufficient condition (3) which is satisfied for every critical point. In order to determine

Table 2 Iterations of qpOASES on nonconvex QP problems from the CUTEr test collection [32]

Problem	n	m	Iters	SOSC
NCVXQP1	100	50	89	sat.
NCVXQP2	100	50	96	sat.
NCVXQP3	100	50	83	sat.
NCVXQP4	100	25	105	sat.
NCVXQP5	100	25	93	sat.
NCVXQP6	100	25	69	sat.
NCVXQP7	100	75	44	sat.
NCVXQP8	100	75	47	sat.
NCVXQP9	100	75	31	sat.
NCVXQP1	1,000	500	4,089	sat.
NCVXQP2	1,000	500	3,836	sat.
NCVXQP3	1,000	500	1,095	sat.
NCVXQP4	1,000	250	1,071	sat.
NCVXQP5	1,000	250	1,032	sat.
NCVXQP6	1,000	250	835	sat.
NCVXQP7	1,000	750	1,112	sat.
NCVXQP8	1,000	750	2,085	sat.
NCVXQP9	1,000	750	834	sat.

All solutions satisfy the second order sufficient conditions (SOSC)

the strongly active inequality constraints, we required the numerical values of the Lagrange multipliers to satisfy $|y_i^*| > 10^{-8}$. The problems were solved with algorithmic parameters of qpOASES that are identical with the default settings except for the explicit treatment of equalities (`enableEqualities`), which was switched on.

6.3 Sequences of QP problems

The parametric active-set strategy as implemented in qpOASES allows warm-starting and hot-starting the solution procedure at any given primal-dual iterate and with any given working set. This feature is particularly useful when solving sequences of QP problems whose optimal solutions are not expected to differ much from one problem to the next. We illustrate this fact by using qpOASES to solve all five test examples of the small online QP benchmark collection [17], which provides sequences of strictly convex QP problems as arising from different linear model predictive control applications. Within each of these QP sequences, the QP matrices are constant and only the QP vectors are varying.

Table 3 summarizes the problem dimensions of each benchmark example and also gives the length of each sequence in terms of number of QP problems. In the first four examples, the optimal active-set changes by up to 18 constraints between successive optimal solutions, while up to 154 changes in the optimal active-set can be observed in the last example.

Table 3 Number of variables (n), number of constraints (m) and number of QP instances for each problem of the online QP benchmark collection [17]

Problem	n	m	no. QPs
chain80	240	0	101
chain80w	240	709	101
diesel	20	20	600
crane	57	160	921
CDU	800	800	7,200

Table 4 Maximum and average number of iterations when solving the problems of the online QP benchmark collection [17] with qpOASES

Problem	Cold-starts		Hot-starts	
	Max.	Avg.	Max.	Avg.
chain80	62	7.4	19	2.3
chain80w	84	10.0	16	2.6
diesel	26	0.4	22	0.2
crane	64	43.9	42	0.3
CDU	263	68.8	313	3.9

Table 4 lists both the maximum and the average number of iterations needed by qpOASES to find the optimal solution. These numbers are given once for cold-started QP solutions and once for QP solutions based on hot-starts as described in Sect. 4. It can be seen that hot-starting the QP solution can reduce both the average and the maximum number of iterations significantly. While interior-point solvers typically have limited warm-starting capabilities, also other active-set QP solvers can benefit from warm-starting. The positive impact on the maximum number of iterations is limited in case of sequences that feature a few situations where the optimal solution changes quite dramatically from one QP problem to the next.

7 Distribution of qpOASES

7.1 Download

The qpOASES software package is available for download from

http://www.qpoases.org

and distributed under the GNU Lesser General Public License (LGPL) to allow linking against proprietary codes. Proceed to the menu point "Download" to obtain a gzipped tar archive of the most current version 3.0 of qpOASES. The user's manual, license text, and extensive source code documentation generated from doxygen [34] are available here as well.

7.2 Installation

To install qpOASES on a 32-bit or 64-bit x86 machine running a standard Linux system, unpack the `tar.gz` archive to obtain a directory named qpOASES. Executing

"make" there will build a static and a dynamic library named libqpOASES.a and libqpOASES.so, and will put them in the directory bin.

The static library libqpOASES.a contains solver code callable from a user-side C++ program that is statically linked against it. The dynamic library libqpOASES.so contains solver code callable from a user-side C++ program that is dynamically linked against it.

qpOASES can be built under the operating systems MacOS or Windows by a straightforward modification of the top-level make file Makefile to make use of the respective OS-specific make file make_osx.mk or make_windows.mk.

The source code of qpOASES is self-contained, but can be optionally linked against LAPACK [1] and BLAS [8] by modifying the appropriate operating system specific make file.

Interfaces to third-party software are excluded from the standard build process, and can be built separately from the subdirectory interfaces/matlab, interfaces/octave, etc. We refer the reader to the qpOASES User's Manual [15] for further details.

8 Summary

We have described the open-source QP solver qpOASES in its most recent release 3.0. The new numerical features, including iterative refinement, data-space termination criterion, ramping, and flipping bounds, improve its reliability and even enable the determination of critical points of nonconvex QP problems. We have explained how the new algorithmic parameters can be modified through the various user interfaces for C, MATLAB®, GNU Octave, and Scilab. Moreover, we have presented a numerical comparison with other popular QP solvers that shows the competitiveness of qpOASES as a stand-alone solver in terms of speed and reliability for small- to medium-scale, dense QP problems.

Acknowledgments The authors would like to thank the three anonymous referees whose insightful comments have helped to improve this article. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007–2013) under grant agreement no FP7-ICT-2009-4 248940 (EMBOCON). The Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences (HGS MathComp, DFG GSC 220/2) supported this work by providing travel grants. Moreover, this research was supported by the German Research Association (DFG) under grants BO 864/12, BO 864/13, and 864/15, and the Research Council KUL: PFV/10/002 Optimization in Engineering Center OPTEC, GOA/10/09 MaNet and GOA/10/11 Global real-time optimal control of autonomous robots and mechatronic systems. Furthermore, this research was supported by the Flemish Government: IOF / KP / SCORES4CHEM; by FWO: PhD/postdoc grants, projects G.0320.08 (convex MPC) and G.0377.09 (Mechatronics MPC); by IWT: PhD Grants, SBO LeCoPro project; by the Belgian Federal Science Policy Office: IUAP P7 (DYSCO, Dynamical systems, control and optimization, 2012-2017); and by the EU: FP7-SADCO (MC ITN-264735), FP7-TEMPO (MC ITN-607957), ERC ST HIGHWIND (259 166), Eurostars SMART, ACCM. Development of a first prototype version of qpOASES has been supported by the REGINS-PREDIMOT European project. At the time of initial submission, the first author was with the Eletrical Engineering Department of KU Leuven (Belgium) and held a PhD fellowship of the Research Foundation – Flanders (FWO).

9 Appendix

See Table 5.

Table 5 Comparison of QP solvers on problems from the Maros-Mészáros test set ([44]) with at most $n = 1{,}000$ variables and $m = 1{,}001$ two-sided inequality constraints (not counting variable bound constraints). The solution time in seconds and the optimality condition residual ρ are given for each solver. Failure of the solver is denoted by blank time fields

Problem name	m	n	OOQP dense		OOQP sparse		qpOASESmpc dense		qpOASESmpc sparse		qpOASES dense		qpOASES sparse		CPLEXP sparse		CPLEXD sparse		CPLEXB sparse	
			Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ
CVXQP1_M	500	1,000	2.652	6e−06	2.650	6e−06	39.876	2e−10	36.163	2e−10	173.151	6e−11	144.286	9e−11	0.244	3e−06	0.915	1e−07		6e−03
CVXQP1_S	50	100	0.006	1e−07	0.006	1e−07	0.032	2e−12	0.032	2e−12	0.127	1e−12	0.137	7e−13	0.014	2e−11	0.015	2e−11	0.025	2e−05
CVXQP2_M	250	1,000		2e−04		2e−04	25.938	1e−11	23.671	8e−12	45.035	2e−12	37.885	4e−12	0.097	2e−07	0.215	2e−07		1e−02
CVXQP2_S	25	100	0.006	2e−09	0.006	2e−09	0.025	4e−12	0.025	5e−13	0.042	2e−13	0.047	1e−13	0.014	5e−11	0.013	5e−11		1e−04
CVXQP3_M	750	1,000		2e−04		2e−04	130.975	4e−09	117.736	4e−09	586.043	8e−10	486.458	2e−09	0.345	5e−09	1.850	2e−09		7e−04
CVXQP3_S	75	100	0.011	1e−10	0.011	1e−10	0.031	2e−12	0.032	1e−12	0.208	6e−13	0.222	4e−13	0.015	1e−12	0.017	5e−13	0.026	3e−06
DPKLO1	77	133	0.009	7e−15	0.009	7e−15	0.107	4e−12	0.100	9e−14	0.543	3e−14	0.470	1e−14	0.008	6e−14	0.009	4e−14	0.012	9e−14
DUAL1	1	85	0.003	8e−05	0.003	8e−05	0.039	9e−14	0.072	9e−15	0.035	2e−15	0.091	3e−15	0.011	3e−15	0.011	3e−15	0.034	3e−06
DUAL2	1	96		3e−04		3e−04	0.038	4e−14	0.070	6e−15	0.068	9e−16	0.162	4e−15	0.013	2e−15	0.013	2e−15	0.035	8e−07
DUAL3	1	111		3e−04		3e−04	0.059	4e−14	0.110	5e−15	0.091	4e−15	0.215	3e−15	0.014	3e−15	0.014	3e−15	0.041	4e−06
DUAL4	1	75	0.002	6e−05	0.002	6e−05	0.014	4e−14	0.026	4e−15	0.025	3e−15	0.058	2e−15	0.010	1e−14	0.010	1e−14	0.029	2e−05
DUALC1	215	9	0.033	2e−07	0.033	2e−07	0.001	6e−11	0.002	2e−11	0.001	5e−10	0.002	5e−10	0.013	1e−10	0.012	1e−10	0.023	1e−05
DUALC2	229	7	0.039	3e−11	0.039	3e−11	0.001	3e−11	0.002	7e−12	0.001	3e−11	0.002	2e−12	0.013	5e−12	0.012	5e−12	0.022	7e−06
DUALC5	278	8	0.051	2e−09	0.051	2e−09	0.001	1e−12	0.003	7e−13	0.001	5e−13	0.003	7e−13	0.013	2e−12	0.013	2e−12	0.021	6e−07
DUALC8	503	8	0.254	1e−09	0.254	1e−09	0.002	1e−10		3e+04	0.002	1e−10	0.007	2e−10	0.014	2e−10	0.013	2e−10	0.021	2e−05
GENHS28	8	10	0.001	1e−16	0.001	1e−16	0.001	3e−13	0.001	2e−15	0.001	1e−15	0.001	2e−15	0.008	4e−16	0.007	3e−15	0.014	1e−09
GOULDQP2	349	699	0.570	1e−04	0.569	1e−04		3e+01	29.381	4e−13	113.434	1e−07	75.260	1e−07	0.179	1e−06	0.046	2e−06	0.035	4e−06
GOULDQP3	349	699		6e−04		6e−04	10.820	4e−11	10.160	1e−13	18.206	2e−14	14.129	1e−14	0.026	3e−14	0.025	3e−14		1e−03
HS118	17	15	0.002	2e−12	0.003	2e−12	0.001	9e−11	0.001	8e−13	0.001	3e−14	0.001	2e−14	0.005	2e−16	0.004	2e−16	0.018	6e−07
HS21	1	2	0.001	5e−11	0.001	5e−11	0.000	2e−12	0.000	0e+00	0.000	1e−16	0.000	1e−16	0.005	0e+00	0.004	0e+00	0.021	2e−08
HS268	5	5		NaN		NaN	0.000	7e−12	0.000	8e−12	0.000	7e−12	0.001	1e−05	0.012	2e−11	0.012	2e−11		2e+10
HS35	1	3	0.001	1e−14	0.001	1e−14	0.000	4e−12	0.000	9e−13	0.000	9e−16	0.000	2e−15	0.012	0e+00	0.011	0e+00	0.031	8e−08
HS35MOD	1	3		1e+00		1e+00	0.000	2e−12	0.000	9e−16	0.000	4e−16	0.000	0e+00	0.012	0e+00	0.011	0e+00	0.027	7e−05
HS51	3	5	0.001	2e−32	0.001	2e−32	0.000	1e−12	0.000	9e−16	0.000	3e−10	0.000	2e−14	0.008	0e+00	0.007	0e+00	0.014	4e−09
HS52	3	5	0.001	2e−16	0.001	2e−16	0.000	6e−13	0.000	3e−15	0.001	4e−15	0.001	2e−15	0.008	5e−15	0.008	2e−15	0.014	1e−09
HS53	3	5	0.000	2e−09	0.001	2e−09	0.000	9e−13	0.000	3e−15	0.000	2e−15	0.001	2e−15	0.008	0e+00	0.008	0e+00	0.025	2e−11

Table 5 continued

Problem name	m	n	OOQP dense		OOQP sparse		qpOASESmpc dense		qpOASESmpc sparse		qpOASES dense		qpOASES sparse		CPLEXP sparse		CPLEXD sparse		CPLEXB sparse	
			Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ
HS76	3	4	0.001	2e−13	0.001	2e−13	0.000	4e−12	0.000	2e−12	0.000	4e−16	0.000	2e−15	0.012	0e+00	0.012	0e+00	0.022	2e−08
KSIP	1,001	20	3.945	4e−05	3.940	4e−05	0.391	5e−12	1.012	8e−13	0.476	3e−16	2.374	1e−15	0.229	7e−16	0.036	7e−16	8.170	2e−06
LOTSCHD	7	12	0.001	5e−07	0.001	5e−07	0.001	3e−11	0.001	8e−14	0.001	6e−14	0.001	7e−14	0.005	2e−13	0.004	2e−13	0.016	5e−06
MOSARQP2	600	900	12.145	6e−05	12.123	6e−05	30.834	5e−12	21.697	5e−12	57.092	4e−15	37.727	4e−15	0.199	8e−14	0.033	9e−14		2e−04
PRIMAL1	85	325		2e−04		2e−04	1.563	8e−11	1.405	7e−11	2.721	2e−15	2.206	2e−15	0.051	7e−14	0.030	7e−14	0.119	1e−06
PRIMAL2	96	649		4e−04		4e−04	11.042	1e−10	9.422	1e−10	20.284	6e−15	15.197	7e−15	0.104	4e−14	0.052	4e−14	0.132	4e−07
PRIMAL3	111	745		6e−04		6e−04	16.780	2e−10	14.331	3e−10	32.054	2e−15	24.082	2e−15	0.409	1e−13	0.135	1e−13	0.280	5e−06
PRIMALC1	9	230	0.085	4e−11	0.085	4e−11	0.017	1e−08	0.012	1e−11	0.036	4e−12	0.031	6e−12	0.006	8e−12	0.005	2e−11	0.031	2e−07
PRIMALC2	7	231	0.088	1e−10	0.088	1e−10	0.008	5e−09	0.006	2e−11	0.013	2e−12	0.010	8e−13	0.005	8e−12	0.005	8e−12	0.032	6e−08
PRIMALC5	8	287	0.136	3e−11	0.137	3e−11	0.019	6e−10	0.013	5e−11	0.041	3e−13	0.033	3e−13	0.006	7e−13	0.005	4e−12	0.033	1e−08
PRIMALC8	8	520	0.551	3e−11	0.549	3e−11	0.112	4e−08	0.068	8e−11	0.175	1e−11	0.125	4e−11	0.007	3e−10	0.007	6e−10	0.035	4e−06
QADLITTL	56	97		9e+43		9e+43		6e+03		6e+03	0.101	2e−11	0.095	1e−08	0.015	1e−12	0.015	1e−12	0.032	1e−06
QAFIRO	27	32	0.004	9e−09	0.004	9e−09	0.004	1e−11	0.004	3e−11	0.004	3e−08	0.002	1e−15	0.013	2e−14	0.012	2e−14	0.033	2e−09
QBANDM	305	472		7e+12		7e+12		8e+01		8e+01	45.766	4e−13	27.961	3e−13	0.026	1e−12	0.031	1e−12		3e−04
QBEACONF	173	262		1e+12		1e+12		2e+03		2e+03	0.397	5e−11	0.380	5e−11	0.014	5e−11	0.015	5e−11		5e+20
QBORE3D	233	315		1e+40		1e+40		4e+02		4e+02	2.017	1e−10	1.093	8e−12	0.010	2e−12	0.010	1e−12	0.040	4e−06
QBRANDY	220	249		NaN		NaN		8e+01		8e+01	5.573	2e−12	4.617	2e−12	0.020	2e−12	0.022	3e−12		1e+03
QCAPRI	271	353		3e+13		3e+13		3e+03		3e+03	4.053	3e−06	3.455	1e−09	0.024	3e−09	0.021	3e−09		6e+20
QE226	223	282		1e+12		1e+12		4e+01		4e+01	10.438	1e−13	7.019	2e−13	0.033	2e−13	0.037	2e−13		1e+20
QETAMACR	400	688		8e+11		8e+11	24.011	4e−10	22.939	4e−10	51.252	8e−07	47.470	1e−06		9e+02	0.085	9e−08		2e−03
QFFFFF80	524	854		1e+12		1e+12		2e+05		2e+05	123.683	9e−11	110.974	3e−09	0.032	1e−10	0.047	9e−11		9e+21
QFORPLAN	161	421		3e+10		3e+10		7e+06		7e+06	12.475	5e−09	8.855	2e−09	0.023	3e−09	0.040	2e−09		4e−01
QGROW15	300	645	1.814	1e−04	1.811	1e−04	43.078	1e−06	33.013	1e−07	40.511	5e−10	15.872	3e−08	0.065	7e−09	0.139	8e−08		1e−01
QGROW22	440	946	5.981	2e−05	6.000	2e−05	148.756	1e−06	144.492	2e−07	147.260	8e−10	51.213	9e−07	0.145	3e−09	0.413	1e−09		4e−02
QGROW7	140	301	0.250	3e−08	0.251	3e−08	4.139	1e−06	3.543	5e−08	3.631	5e−10	2.193	3e−09	0.023	2e−08	0.023	5e−09		3e−02

Table 5 continued

Problem name	m	n	OOQP dense		OOQP sparse		qpOASESmpc dense		qpOASESmpc sparse		qpOASES dense		qpOASES sparse		CPLEXP sparse		CPLEXD sparse		CPLEXB sparse	
			Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ	Time	ρ
QISRAEL	174	142	0.149	2e−05	0.149	2e−05	0.323	4e−08	0.384	4e−08	0.577	1e−11	0.597	1e−11	0.022	3e−11	0.021	1e−11		2e+21
QPCBLEND	74	83	0.052	7e−05	0.052	7e−05	0.106	2e−12	0.098	3e−12	0.106	2e−15	0.099	1e−15		5e+00		5e+00	0.041	5e−06
QPCBOEI1	351	384		7e+10		7e+10	6.166	1e−09	5.526	1e−09	8.808	1e−10	7.371	3e−11	0.038	2e−09	0.031	2e−09		3e−02
QPCBOEI2	166	143		6e+06		6e+06	0.314	8e−08	0.303	2e−08	0.443	5e−09	0.415	1e−08	0.015	2e−09	0.015	2e−09		4e−02
QPCSTAIR	356	467		2e+13		2e+13	9.904	3e−10	8.406	5e−10	15.225	4e−11	12.339	3e−11	0.048	3e−09	0.060	4e−09		5e−03
QPTEST	2	2	0.001	2e−12	0.001	2e−12	0.000	1e−12	0.000	3e−12	0.000	0e+00	0.000	9e−16	0.012	2e−15	0.012	2e−15	0.025	1e−08
QRECIPE	91	180		1e+12		1e+12	0.143	4e−11	0.133	2e−11	0.185	2e−14	0.096	8e−15	0.014	2e−16	0.013	2e−16		3e+20
QSC205	205	203		6e−03		6e−03	0.326	4e−12	0.313	2e−12	0.432	7e−16	0.355	1e−15	0.015	7e−18	0.014	7e−18		1e+18
QSCAGR25	471	500	1.901	2e−11	1.900	2e−11		8e+03		8e+03	67.016	9e−11	32.950	2e−11	0.026	1e−10	0.053	1e−10		1e−03
QSCAGR7	129	140	0.074	3e−11	0.074	3e−11		8e+03		8e+03	1.269	1e−11	0.753	1e−11	0.014	1e−11	0.018	1e−11		8e−03
QSCFXM1	330	457		NaN		NaN		1e+03		1e+03	11.336	8e−11	9.893	7e−11	0.024	3e−08	0.031	3e−08		3e+20
QSCFXM2	660	914		NaN		NaN		1e+03		1e+03	142.043	7e−10	79.030	8e−11	0.052	5e−08	0.054	5e−08		3e+20
QSCORPIO	388	358		5e+27		5e+27		2e+02		2e+02	5.320	1e−11	4.736	7e−12	0.016	9e−13	0.016	9e−13		2e+17
QSCSD1	77	760	2.157	6e−12	2.149	6e−12		6e+00		6e+00	3.210	5e−09	1.963	5e−09	0.019	2e−08	0.014	9e−09	0.050	3e−09
QSCTAP1	300	480		1e−04		1e−04	8.697	2e−10	6.978	6e−13	7.843	6e−08	4.448	5e−08	0.020	4e−14	0.021	4e−14		8e+15
QSHARE1B	117	225	0.151	2e−08	0.151	2e−08		3e+03		3e+03	2.436	1e−10	1.708	1e−09	0.020	3e−10	0.020	3e−10		2e−03
QSHARE2B	96	79	0.029	2e−11	0.029	2e−11	0.063	3e−10	0.063	8e−10	0.245	4e−12	0.190	8e−12	0.016	5e−12	0.015	5e−12		2e+15
QSTAIR	356	467		2e+09		2e+09	12.765	3e−10	9.073	2e−10	12.034	1e−08	9.851	2e−11	0.050	1e−10	0.072	8e−11		1e+17
S268	5	5		NaN		NaN	0.000	7e−12	0.000	8e−12	0.001	7e−12	0.001	1e−05	0.012	2e−11	0.012	2e−11		2e+10
TAME	1	2	0.000	2e−12	0.001	2e−12	0.000	6e−13	0.000	4e−16	0.000	6e−16	0.000	6e−16	0.008	0e+00	0.007	0e+00	0.033	2e−18
VALUES	1	202	0.016	1e−05	0.017	1e−05		1e+00		1e+00	0.125	3e−14	0.159	1e−14		1e+00		1e+00		1e+00
ZECEVIC2	2	2	0.001	1e−15	0.001	1e−15	0.000	4e−12	0.000	4e−12	0.000	7e−16	0.000	7e−16	0.005	0e+00	0.004	0e+00	0.019	1e−08

References

1. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide, 3rd edn. SIAM, Philadelphia, PA (1999)
2. Ängeby, J., Huschenbett, M., Alberer, D.: Automotive Model Predictive Control, MIMO Model Predictive Control for Integral Gas Engines. In: Lecture Notes in Control and Information Sciences, vol. 402, pp. 257–272, Springer, Berlin (2010)
3. Arnold, E., Neupert, J., Sawodny, O., Schneider, K.: Trajectory tracking for boom cranes based on nonlinear control and optimal trajectory generation. In: IEEE International Conference on Control Applications, pp. 1444–1449 (2007). doi:10.1109/CCA.2007.4389439
4. Bartels, R., Golub, G.: The simplex method for linear programming using LU decomposition. Commun. ACM **12**(5), 266–268 (1969)
5. Bartlett, R., Biegler, L.: QPSchur: a dual, active set, Schur complement method for large-scale and structured convex quadratic programming algorithm. Optim. Eng. **7**, 5–32 (2006)
6. Benzi, M., Golub, G., Liesen, J.: Numerical solution of saddle-point problems. Acta Numerica **14**, 1–137 (2005)
7. Best, M.: An Algorithm for the Solution of the Parametric Quadratic Programming Problem. In: Applied Mathematics and Parallel Computing, pp. 57–76. Physica, Heidelberg (1996)
8. Blackford, L.S., Demmel, J., Dongarra, J., Duff, I.: An updated set of basic linear algebra subprograms (BLAS). ACM Trans. Math. Softw. **28**, 135–151 (2002)
9. Bock, H., Plitt, K.: A multiple shooting algorithm for direct solution of optimal control problems. In: Proceedings 9th IFAC World Congress, pp. 242–247. Pergamon Press, Budapest (1984)
10. Contesse, L.: Une charactérisation complète des minima locaux en programmation quadratique. Numerische Mathematik **34**, 315–332 (1980)
11. Dantzig, G.B.: Linear Programming and Extensions. Princeton University Press, Princeton (1963)
12. Diehl, M., Ferreau, H.J., Haverbeke, N.: Nonlinear model predictive control, Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation. In: Lecture Notes in Control and Information Sciences, vol. 384, pp. 391–417. Springer, Berlin (2009)
13. Eaton, J.W.: GNU Octave Manual. Network Theory Limited (2002)
14. Ferreau, H.J.: An Online Active Set Strategy for Fast Solution of Parametric Quadratic Programs with Applications to Predictive Engine Control. Master's thesis, University of Heidelberg (2006)
15. Ferreau, H.J., et al.: qpOASES User's Manual. http://www.qpOASES.org (2007–2014)
16. Ferreau, H.J.: Model predictive control algorithms for applications with millisecond timescales. PhD thesis, KU Leuven (2011)
17. Ferreau, H.J., Diehl, M.: Online QP Benchmark Collection. http://www.qpOASES.org/onlineQP (2006–2008)
18. Ferreau, H.J., Ortner, P., Langthaler, P., del Re, L., Diehl, M.: Predictive control of a real-world diesel engine using an extended online active set strategy. Annu. Rev. Control **31**(2), 293–301 (2007)
19. Ferreau, H.J., Bock, H.G., Diehl, M.: An online active set strategy to overcome the limitations of explicit MPC. Int. J. Robust Nonlinear Control **18**(8), 816–830 (2008)
20. Fletcher, R.: Practical Methods of Optimization, 2nd edn. Wiley, Chichester (1987)
21. Fletcher, R.: Approximation Theory and Optimization, Dense factors of sparse matrices. In: Tributes to M.J.D. Powell, pp. 145–166. Cambridge University Press, Cambridge (1997)
22. Fletcher, R.: Stable reduced Hessian updates for indefinite quadratic programming. Numerical Analysis Report NA/187, Department of Mathematics and Computer Science, University of Dundee, Dundee DD1 4HN, Scotland, UK (1999)
23. Fletcher, R., Matthews, S.: Stable modification of explicit LU factors for simplex updates. Math. Progr. **30**(3), 267–284 (1984)
24. Gertz, E., Wright, S.: Object-oriented software for quadratic programming. ACM Trans. Math. Softw. **29**(1), 58–81 (2003)
25. Gill, P., Golub, G., Murray, W., Saunders, M.A.: Methods for modifying matrix factorizations. Math. Comput. **28**(126), 505–535 (1974)
26. Gill, P., Murray, W., Saunders, M., Wright, M.: Procedures for optimization problems with a mixture of bounds and general linear constraints. ACM Trans. Math. Softw. **10**(3), 282–298 (1984)
27. Gill, P., Murray, W., Saunders, M., Wright, M.: Maintaining LU factors of a general sparse matrix. Linear Algebra Appl. **88**(89), 239–270 (1987)

28. Goldfarb, D., Idnani, A.: A numerically stable dual method for solving strictly convex quadratic programs. Math. Progr. **27**, 1–33 (1983)
29. Gould, N.: An algorithm for large-scale quadratic programming. IMA J. Numer. Anal. **11**(3), 299–324 (1991)
30. Gould, N., Toint, P.: A quadratic programming bibliography. Tech. Rep. 2000–1, Rutherford Appleton Laboratory, Computational Science and Engineering Department (2010)
31. Gould, N., Toint, P.: A quadratic programming page. http://www.numerical.rl.ac.uk/qp/qp.html (2012)
32. Gould, N., Orban, D., Toint, P.: CUTEr testing environment for optimization and linear algebra solvers. http://cuter.rl.ac.uk/cuter-www/, (2002)
33. Harris, P.: Pivot selection methods of the DEVEXLP code. Math. Progr. **5**, 1–29 (1973)
34. van Heesch, D.: Doxygen homepage (1997–2011). http://www.doxygen.org
35. Houska, B., Ferreau, H.J., Diehl, M.: ACADO Toolkit—an open source framework for automatic control and dynamic optimization. Optim. Control Appl. Methods **32**(3), 298–312 (2011)
36. Huynh, H.: A large-scale quadratic programming solver based on block-LU updates of the KKT system. PhD thesis, Stanford University (2008)
37. IBM Corp: IBM ILOG CPLEX V12.1, User's Manual for CPLEX (2009)
38. Inc TM: Real-Time Workshop for Use with SIMULINK, User's Guide (1999)
39. Karmarkar, N.: A new polynomial time algorithm for linear programming. Combinatorica **4**, 373–395 (1984)
40. Kirches, C., Bock, H., Schlöder, J., Sager, S.: A factorization with update procedures for a KKT matrix arising in direct optimal control. Math. Progr. Comput. **3**(4), 319–348 (2011)
41. Kostina, E.: The long step rule in the bounded-variable dual simplex method: numerical experiments. Math. Methods Oper. Res. **55**(3), 413–429 (2002)
42. Leineweber, D., Bauer, I., Schäfer, A., Bock, H., Schlöder, J.: An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization (parts I and II). Comput. Chem. Eng. **27**, 157–174 (2003)
43. Löfberg, J.: YALMIP: A toolbox for modeling and optimization in MATLAB. In: Proceedings of the CACSD Conference, Taipei, Taiwan (2004). http://users.isy.liu.se/johanl/yalmip
44. Maros, I., Meszaros, C.: A repository of convex quadratic programming problems. Optim. Methods Softw. **11**, 431–449 (1999)
45. Murty, K.: Some NP-complete problems in quadratic and nonlinear programming. Math. Progr. **39**, 117–129 (1987)
46. Nesterov, Y.: Introductory lectures on convex optimization: a basic course. In: Applied Optimization, vol. 87. Kluwer Academic Publishers, Dordrecht (2003)
47. Nesterov, Y., Nemirovski, A.: Interior-point Polynomial Algorithms in Convex Programming. In: Society for Industrial Mathematics (1994)
48. Nocedal, J., Wright, S.: Springer series in operations research and financial engineering. In: Numerical Optimization, 2nd edn. Springer, Berlin (2006)
49. Rauter, G., von Zitzewitz, J., Duschau-Wicke, A., Vallery, H., Riener, R.: A tendon-based parallel robot applied to motor learning in sports. In: Proceedings of the IEEE International Conference on Biomedical Robotics and Biomechatronics 2010, Japan (2010)
50. Rockafellar, R.: Monotone operators and the proximal point algorithm. SIAM J. Control Optim. **14**, 877–898 (1976)
51. Sager, S.: Lange Schritte im Dualen Simplex-Algorithmus. Master's thesis, Universität Heidelberg (2001)
52. Scilab Consortium: Scilab: The free software for numerical computation. Scilab Consortium, Digiteo, Paris, France (2011). http://www.scilab.org
53. Takács, G., Rohal'-Ilkiv, B.: Predictive vibration control: efficient constrained MPC vibration control for lightly damped mechanical systems. Springer, Berlin (2012)
54. Van den Broeck, L.: Time optimal control of mechatronic systems through embedded optimization. PhD thesis, KU Leuven (2011)
55. Wang, X.: Resolution of Ties in Parametric Quadratic Programming. Master's thesis, University of Waterloo, Ontario, Canada (2004)
56. Wilkinson, J.: The Algebraic Eigenvalue Problem. Clarendon Press, Oxford (1965)
57. Wills, A., Bates, D., Fleming, A., Ninness, B., Moheimani, S.: Application of MPC to an active structure using sampling rates up to 25kHz. In: 44th IEEE Conference on Decision and Control and European Control Conference ECC'05, Seville (2005)

58. Wolfe, P.: The simplex method for quadratic programming. Econometrica **27**, 382–398 (1959)
59. Wright, S.: Primal-Dual Interior-Point Methods. SIAM Publications, Philadelphia (1997)
60. Wunderling, R.: Paralleler und Objektorientierter Simplex-Algorithmus. PhD thesis, Konrad-Zuse-Zentrum Berlin (1996)