Mathematical Programming Computation (2020) 12:637–672
https://doi.org/10.1007/s12532-020-00179-2

FULL LENGTH PAPER

OSQP: an operator splitting solver for quadratic programs

Bartolomeo Stellato[1] · Goran Banjac[2] · Paul Goulart[3] · Alberto Bemporad[4] · Stephen Boyd[5]

Received: 7 January 2018 / Accepted: 5 February 2020 / Published online: 20 February 2020

Abstract

We present a general-purpose solver for convex quadratic programs based on the alternating direction method of multipliers, employing a novel operator splitting technique that requires the solution of a quasi-definite linear system with the same coefficient matrix at almost every iteration. Our algorithm is very robust, placing no requirements on the problem data such as positive definiteness of the objective function or linear independence of the constraint functions. It can be configured to be division-free once an initial matrix factorization is carried out, making it suitable for real-time applications in embedded systems. In addition, our technique is the first operator splitting method for quadratic programs able to reliably detect primal and dual infeasible problems from the algorithm iterates. The method also supports factorization caching and warm starting, making it particularly efficient when solving parametrized problems arising in finance, control, and machine learning. Our open-source C implementation OSQP has a small footprint, is library-free, and has been extensively tested on many problem instances from a wide variety of application areas. It is typically ten times faster than competing interior-point methods, and sometimes much more when factorization caching or warm start is used. OSQP has already shown a large impact with tens of thousands of users both in academia and in large corporations.

Keywords Optimization · Quadratic programming · Operator splitting · First-order methods

Mathematics Subject Classification 90C20 · 65K05 · 65K10 · 90C25 · 90C06 · 90C46 · 90C90

This work was supported by the People Programme (Marie Curie Actions) of the European Union Seventh Framework Programme (FP7/2007–2013) under REA Grant agreement No. 607957 (TEMPO).

✉ Bartolomeo Stellato
stellato@mit.edu

Extended author information available on the last page of the article

1 Introduction

1.1 The problem

Consider the following optimization problem

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x \\ \text{subject to} & Ax \in \mathcal{C}, \end{array} \tag{1}$$

where $x \in \mathbf{R}^n$ is the decision variable. The objective function is defined by a positive semidefinite matrix $P \in \mathbf{S}_+^n$ and a vector $q \in \mathbf{R}^n$, and the constraints by a matrix $A \in \mathbf{R}^{m \times n}$ and a nonempty, closed and convex set $\mathcal{C} \subseteq \mathbf{R}^m$. We will refer to it as *general (convex) quadratic program*.

If the set $\mathcal{C}$ takes the form

$$\mathcal{C} = [l, u] = \left\{ z \in \mathbf{R}^m \mid l_i \leq z_i \leq u_i, \ i = 1, \ldots, m \right\},$$

with $l_i \in \{-\infty\} \cup \mathbf{R}$ and $u_i \in \mathbf{R} \cup \{+\infty\}$, we can write problem (1) as

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x \\ \text{subject to} & l \leq Ax \leq u, \end{array} \tag{2}$$

which we will refer to as a *quadratic program* (QP). Linear equality constraints can be encoded in this way by setting $l_i = u_i$ for some or all of the elements in (l, u). Note that any linear program (LP) can be written in this form by setting $P = 0$. We will characterize the size of (2) with the tuple (n, m, N) where N is the sum of the number of nonzero entries in P and A, *i.e.*, $N = \text{nnz}(P) + \text{nnz}(A)$.

Applications Optimization problems of the form (1) arise in a huge variety of applications in engineering, finance, operations research and many other fields. Applications in machine learning include support vector machines (SVM) [21], Lasso [19,90] and Huber fitting [55,56]. Financial applications of (1) include portfolio optimization [13,15,20,67], [17, §4.4.1]. In the field of control engineering, model predictive control (MPC) [42,82] and moving horizon estimation (MHE) [2] techniques require the solution of a QP at each time instant. Several signal processing problems also fall into the same class [17, §6.3.3], [69]. In addition, the numerical solution of QP subproblems is an essential component in nonconvex optimization methods such as sequential quadratic programming (SQP) [76, Chap. 18] and mixed-integer optimization using branch-and-bound algorithms [10,37].

1.2 Solution methods

Convex QPs have been studied since the 1950s [39], following from the seminal work on LPs started by Kantorovich [59]. Several solution methods for both LPs and QPs have been proposed and improved upon throughout the years.

Active-set methods Active-set methods were the first algorithms popularized as solution methods for QPs [94], and were obtained from an extension of Dantzig's simplex method for solving LPs [22]. Active-set algorithms select an active-set (*i.e.*, a set of binding constraints) and then iteratively adapt it by adding and dropping constraints from the index of active ones [76, §16.5]. New active constraints are added based on the cost function gradient and the current dual variables. Active-set methods for QPs differ from the simplex method for LPs because the iterates are not necessarily vertices of the feasible region. These methods can easily be warm started to reduce the number of active-set recalculations required. However, the major drawback of active-set methods is that the worst-case complexity grows exponentially with the number of constraints, since it may be necessary to investigate all possible active-sets before reaching the optimal one [62]. Modern implementations of active-set methods for the solution of QPs can be found in many commercial solvers, such as MOSEK [73] and GUROBI [53], and in the open-source solver qpOASES [36].

Interior-point methods Interior-point algorithms gained popularity in the 1980s as a method for solving LPs in polynomial time [45,60]. In the 90s these techniques were extended to general convex optimization problems, including QPs [74]. Interior-point methods model the problem constraints as parametrized penalty functions, also referred to as *barrier functions*. At each iteration an unconstrained optimization problem is solved for varying barrier function parameters until the optimum is achieved; see [17, Chap. 11] and [76, §16.6] for details. Primal–dual interior-point methods, in particular the Mehrotra predictor–corrector [71] method, became the algorithms of choice for practical implementation [95] because of their good performance across a wide range of problems. However, interior-point methods are not easily warm started and do not scale well for very large problems. Interior-point methods are currently the default algorithms in the commercial solvers MOSEK [73], GUROBI [53] and CVXGEN [70] and in the open-source solver OOQP [43].

First-order methods First-order optimization methods for solving quadratic programs date to the 1950s [39]. These methods iteratively compute an optimal solution using only first-order information about the cost function. Operator splitting techniques such as the Douglas–Rachford splitting [31,64] are a particular class of first-order methods which model the optimization problem as the problem of finding a zero of the sum of monotone operators.

In recent years, the operator splitting method known as the alternating direction method of multipliers (ADMM) [41,48] has received particular attention because of its very good practical convergence behavior; see [16] for a survey. ADMM can be seen as a variant of the classical alternating projections algorithm [8] for finding a point in the intersection of two convex sets, and can also be shown to be equivalent to the Douglas–Rachford splitting [40]. ADMM has been shown to reliably provide modest accuracy solutions to QPs in a relatively small number of computationally inexpensive iterations. It is therefore well suited to applications such as embedded optimization or large-scale optimization, wherein high accuracy solutions are typically not required due to noise in the data and arbitrariness of the cost function. ADMM steps are computationally very cheap and simple to implement, and thus ideal for embedded processors with limited

computing resources such as those found in embedded control systems [58,78,87]. ADMM is also compatible with distributed optimization architectures enabling the solution of very large-scale problems [16].

A drawback of first-order methods is that they are typically unable to detect primal and/or dual infeasibility. In order to address this shortcoming, a homogeneous self-dual embedding has been proposed in conjunction with ADMM for solving conic optimization problems and implemented in the open-source solver SCS [77]. Although every QP can be reformulated as a conic program, this reformulation is not efficient from a computational point of view. A further drawback of ADMM is that number of iterations required to converge is highly dependent on the problem data and on the user's choice of the algorithm's step-size parameters. Despite some recent theoretical results [5,47], it remains unclear how to select those parameters to optimize the algorithm convergence rate. For this reason, even though there are several benefits in using ADMM techniques for solving optimization problems, there exists no reliable general-purpose QP solver based on operator splitting methods.

1.3 Our approach

In this work we present a new general-purpose QP solver based on ADMM that is able to provide high accuracy solutions. The proposed algorithm is based on a novel splitting requiring the solution of a quasi-definite linear system that is always solvable for any choice of problem data. We therefore impose no constraints such as strict convexity of the cost function or linear independence of the constraints. Since the linear system's matrix coefficients remain the same at every iteration when ρ is fixed, our algorithm requires only a single factorization to solve the QP (2). Once this initial factorization is computed, we can fix the linear system matrix coefficients to make the algorithm division-free. If we allow divisions, then we can make occasional updates to the term ρ in this linear system to improve our algorithm's convergence. We find that our algorithm typically updates these coefficients very few times, *e.g.*, 1 or 2 in our experiments. In contrast to other first-order methods, our approach is able to return primal and dual solutions when the problem is solvable or to provide certificates of primal and dual infeasibility without resorting to the homogeneous self-dual embedding.

To obtain high accuracy solutions, we perform *solution polishing* on the iterates obtained from ADMM. By identifying the active constraints from the final dual variable iterates, we construct an ancillary equality-constrained QP whose solution is equivalent to that of the original QP (1). This ancillary problem is then solved by computing the solution of a single linear system of typically much lower dimensions than the one solved during the ADMM iterations. If we identify the active constraints correctly, then the resulting solution of our method has accuracy equal to or even better than interior-point methods.

Our algorithm can be efficiently warm started to reduce the number of iterations. Moreover, if the problem matrices do not change then the quasi-definite system factorization can be reused across multiple solves greatly improving the computation time. This feature is particularly useful when solving multiple instances of parametric

QPs where only a few elements of the problem data change. Examples illustrating the effectiveness of the proposed algorithm in parametric programs arising in embedded applications appear in [7].

We implemented our method in the open-source "Operator Splitting Quadratic Program" (OSQP) solver. OSQP is written in C and can be compiled to be library free. OSQP is robust against noisy and unreliable problem data, has a very small code footprint, and is suitable for both embedded and large-scale applications. We have extensively tested our code and carefully tuned its parameters by solving millions of QPs. We benchmarked our solver against state-of-the-art interior-point and active-set solvers over a benchmark library of 1400 problems from 7 different classes and over the hard QPs Maros–Mészáros test set [68]. Numerical results show that our algorithm is able to provide up to an order of magnitude computational time improvements over existing commercial and open-source solvers in a wide variety of applications. We also showed further time reductions from warm starting and factorization caching.

2 Optimality conditions

We will find it convenient to rewrite problem (1) by introducing an additional decision variable $z \in \mathbf{R}^m$, to obtain the equivalent problem

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x \\ \text{subject to} & Ax = z \\ & z \in \mathcal{C}. \end{array} \tag{3}$$

We can write the optimality conditions of problem (3) as [6, Lem. A.1], [84, Thm. 6.12]

$$Ax = z, \tag{4}$$
$$Px + q + A^T y = 0, \tag{5}$$
$$z \in \mathcal{C}, \quad y \in N_{\mathcal{C}}(z), \tag{6}$$

where $y \in \mathbf{R}^m$ is the Lagrange multiplier associated with the constraint $Ax = z$ and $N_{\mathcal{C}}(z)$ denotes the normal cone of $\mathcal{C}$ at z. If there exist $x \in \mathbf{R}^n$, $z \in \mathbf{R}^m$ and $y \in \mathbf{R}^m$ that satisfy the conditions above, then we say that (x, z) is a *primal* and y is a *dual* solution to problem (3). We define the primal and dual residuals of problem (1) as

$$r_{\text{prim}} = Ax - z, \tag{7}$$
$$r_{\text{dual}} = Px + q + A^T y. \tag{8}$$

Quadratic programs In case of QPs of the form (2), condition (6) reduces to

$$l \leq z \leq u, \qquad y_+^T(z - u) = 0, \qquad y_-^T(z - l) = 0, \tag{9}$$

where $y_+ = \max(y, 0)$ and $y_- = \min(y, 0)$.

2.1 Certificates of primal and dual infeasibility

From the *theorem of strong alternatives* [17, §5.8], [6, Prop. 3.1], exactly one of the following sets is nonempty

$$\mathcal{P} = \left\{x \in \mathbf{R}^n \mid Ax \in \mathcal{C}\right\}, \tag{10}$$

$$\mathcal{D} = \left\{y \in \mathbf{R}^m \mid A^T y = 0, \quad S_{\mathcal{C}}(y) < 0\right\}, \tag{11}$$

where $S_{\mathcal{C}}$ is the support function of $\mathcal{C}$, provided that some type of constraint qualification holds [17]. In other words, any variable $y \in \mathcal{D}$ serves as a *certificate* that problem (1) is primal infeasible.

Quadratic programs In case $\mathcal{C} = [l, u]$, certifying primal infeasibility of (2) amounts to finding a vector $y \in \mathbf{R}^m$ such that

$$A^T y = 0, \quad u^T y_+ + l^T y_- < 0. \tag{12}$$

Similarly, it can be shown that a vector $x \in \mathbf{R}^n$ satisfying

$$Px = 0, \quad q^T x < 0, \quad (Ax)_i \begin{cases} = 0 & l_i, u_i \in \mathbf{R} \\ \geq 0 & u_i = +\infty,\ l_i \in \mathbf{R} \\ \leq 0 & l_i = -\infty,\ u_i \in \mathbf{R} \end{cases} \tag{13}$$

is a certificate of dual infeasibility for problem (2); see [6, Prop. 3.1] for more details.

3 Solution with ADMM

Our method solves problem (3) using ADMM [16]. By introducing auxiliary variables $\tilde{x} \in \mathbf{R}^n$ and $\tilde{z} \in \mathbf{R}^m$, we can rewrite problem (3) as

$$\begin{array}{ll} \text{minimize} & (1/2)\tilde{x}^T P\tilde{x} + q^T\tilde{x} + \mathcal{I}_{Ax=z}(\tilde{x}, \tilde{z}) + \mathcal{I}_{\mathcal{C}}(z) \\ \text{subject to} & (\tilde{x}, \tilde{z}) = (x, z), \end{array} \tag{14}$$

where $\mathcal{I}_{Ax=z}$ and $\mathcal{I}_{\mathcal{C}}$ are the indicator functions given by

$$\mathcal{I}_{Ax=z}(x, z) = \begin{cases} 0 & Ax = z \\ +\infty & \text{otherwise} \end{cases}, \quad \mathcal{I}_{\mathcal{C}}(z) = \begin{cases} 0 & z \in \mathcal{C} \\ +\infty & \text{otherwise} \end{cases}.$$

An iteration of ADMM for solving problem (14) consists of the following steps:

$$\begin{aligned} (\tilde{x}^{k+1}, \tilde{z}^{k+1}) \leftarrow & \underset{(\tilde{x},\tilde{z}):A\tilde{x}=\tilde{z}}{\text{argmin}} \ (1/2)\tilde{x}^T P\tilde{x} + q^T\tilde{x} \\ & + (\sigma/2)\|\tilde{x} - x^k + \sigma^{-1}w^k\|_2^2 + (\rho/2)\|\tilde{z} - z^k + \rho^{-1}y^k\|_2^2 \end{aligned} \tag{15}$$

$$x^{k+1} \leftarrow \alpha\tilde{x}^{k+1} + (1-\alpha)x^k + \sigma^{-1}w^k \tag{16}$$

$$z^{k+1} \leftarrow \Pi\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k + \rho^{-1}y^k\right) \tag{17}$$

$$w^{k+1} \leftarrow w^k + \sigma\left(\alpha\tilde{x}^{k+1} + (1-\alpha)x^k - x^{k+1}\right) \tag{18}$$

$$y^{k+1} \leftarrow y^k + \rho\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k - z^{k+1}\right) \tag{19}$$

where $\sigma > 0$ and $\rho > 0$ are the *step-size parameters*, $\alpha \in (0, 2)$ is the *relaxation parameter*, and Π denotes the Euclidean projection onto $\mathcal{C}$. The introduction of the splitting variable $\tilde{x}$ ensures that the subproblem in (15) is always solvable for any $P \in \mathbf{S}^n_+$ which can also be 0 for LPs. Note that all the derivations hold also for σ and ρ being positive definite diagonal matrices. The iterates w^k and y^k are associated with the dual variables of the equality constraints $\tilde{x} = x$ and $\tilde{z} = z$, respectively. Observe from steps (16) and (18) that $w^{k+1} = 0$ for all $k \geq 0$, and consequently the w-iterate and the step (18) can be disregarded.

3.1 Solving the linear system

Evaluating the ADMM step (15) involves solving the equality constrained QP

$$\begin{array}{ll} \text{minimize} & (1/2)\tilde{x}^T P\tilde{x} + q^T\tilde{x} + (\sigma/2)\|\tilde{x} - x^k\|_2^2 + (\rho/2)\|\tilde{z} - z^k + \rho^{-1}y^k\|_2^2 \\ \text{subject to} & A\tilde{x} = \tilde{z}. \end{array} \tag{20}$$

The optimality conditions for this equality constrained QP are

$$P\tilde{x}^{k+1} + q + \sigma(\tilde{x}^{k+1} - x^k) + A^T\nu^{k+1} = 0, \tag{21}$$

$$\rho(\tilde{z}^{k+1} - z^k) + y^k - \nu^{k+1} = 0, \tag{22}$$

$$A\tilde{x}^{k+1} - \tilde{z}^{k+1} = 0, \tag{23}$$

where $\nu^{k+1} \in \mathbf{R}^m$ is the Lagrange multiplier associated with the constraint $Ax = z$. By eliminating the variable $\tilde{z}^{k+1}$ from (22), the above linear system reduces to

$$\begin{bmatrix} P + \sigma I & A^T \\ A & -\rho^{-1}I \end{bmatrix}\begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \rho^{-1}y^k \end{bmatrix}, \tag{24}$$

with $\tilde{z}^{k+1}$ recoverable as

$$\tilde{z}^{k+1} = z^k + \rho^{-1}(\nu^{k+1} - y^k).$$

We will refer to the coefficient matrix in (24) as the *KKT matrix*. This matrix always has full rank thanks to the positive parameters σ and ρ introduced in our splitting, so (24) always has a unique solution for any matrices $P \in \mathbf{S}^n_+$ and $A \in \mathbf{R}^{m\times n}$. In other words, we do not impose any additional assumptions on the problem data such as strong convexity of the objective function or linear independence of the constraints as was done in [44,80,81].

Direct method A direct method for solving the linear system (24) computes its solution by first factoring the KKT matrix and then performing forward and backward substitution. Since the KKT matrix remains the same for every iteration of ADMM, we only need to perform the factorization once prior to the first iteration and cache the factors so that we can reuse them in subsequent iterations. This approach is very efficient when the factorization cost is considerably higher than the cost of forward and backward substitutions, so that each iteration is computed quickly. Note that if ρ or σ change, the KKT matrix needs to be factored again.

Our particular choice of splitting results in a KKT matrix that is quasi-definite, *i.e.*, it can be written as a 2-by-2 block-symmetric matrix where the $(1, 1)$-block is positive definite, and the $(2, 2)$-block is negative definite. It therefore always has a well defined LDL^T factorization, with L being a lower triangular matrix with unit diagonal elements and D a diagonal matrix with nonzero diagonal elements [91]. Note that once the factorization is carried out, computing the solution of (24) can be made division-free by storing D^{-1} instead of D.

When the KKT matrix is sparse and quasi-definite, efficient algorithms can be used for computing a suitable permutation matrix P for which the factorization of PKP^T results in a sparse factor L [3,24] without regard for the actual nonzero values appearing in the KKT matrix. The LDL^T factorization consists of two steps. In the first step we compute the sparsity pattern of the factor L. This step is referred to as the *symbolic factorization* and requires only the sparsity pattern of the KKT matrix. In the second step, referred to as the *numerical factorization*, we determine the values of nonzero elements in L and D. Note that we do not need to update the symbolic factorization if the nonzero entries of the KKT matrix change but the sparsity pattern remains the same.

Indirect method With large-scale QPs, factoring linear system (24) might be prohibitive. In these cases it might be more convenient to use an indirect method by solving instead the linear system

$$(P + \sigma I + \rho A^T A)\tilde{x}^{k+1} = \sigma x^k - q + A^T(\rho z^k - y^k)$$

obtained by eliminating ν^{k+1} from (24). We then compute $\tilde{z}^{k+1}$ as $\tilde{z}^{k+1} = A\tilde{x}^{k+1}$. Note that the coefficient matrix in the above linear system is always positive definite. The linear system can therefore be solved with an iterative scheme such as the conjugate gradient method [49,76]. When the linear system is solved up to some predefined accuracy, we terminate the method. We can also warm start the method using the linear system solution at the previous iteration of ADMM to speed up its convergence. In contrast to direct methods, the complexity of indirect methods does not change if we update ρ and σ since there is no factorization required. This allows for more updates to take place without any overhead.

3.2 Final algorithm

By simplifying the ADMM iterations according to the previous discussion, we obtain Algorithm 1. Steps 4, 5, 6 and 7 of Algorithm 1 are very easy to evaluate since they involve only vector addition and subtraction, scalar-vector multiplication and projection onto a box. Moreover, they are component-wise separable and can be easily parallelized. The most computationally expensive part is solving the linear system in Step 3, which can be performed as discussed in Sect. 3.1.

Algorithm 1

1: **given** initial values x^0, z^0, y^0 and parameters $\rho > 0$, $\sigma > 0$, $\alpha \in (0, 2)$
2: **repeat**
3: $(\tilde{x}^{k+1}, \nu^{k+1}) \leftarrow$ solve linear system $\begin{bmatrix} P + \sigma I & A^T \\ A & -\rho^{-1} I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \rho^{-1} y^k \end{bmatrix}$
4: $\tilde{z}^{k+1} \leftarrow z^k + \rho^{-1}(\nu^{k+1} - y^k)$
5: $x^{k+1} \leftarrow \alpha \tilde{x}^{k+1} + (1 - \alpha) x^k$
6: $z^{k+1} \leftarrow \Pi\left(\alpha \tilde{z}^{k+1} + (1 - \alpha) z^k + \rho^{-1} y^k\right)$
7: $y^{k+1} \leftarrow y^k + \rho\left(\alpha \tilde{z}^{k+1} + (1 - \alpha) z^k - z^{k+1}\right)$
8: **until** termination criterion is satisfied

3.3 Convergence and infeasibility detection

We show in this section that the proposed algorithm generates a sequence of iterates (x^k, z^k, y^k) that in the limit satisfy the optimality conditions (4)–(6) when problem (1) is solvable, or provides a certificate of primal or dual infeasibility otherwise.

If we denote the argument of the projection operator in step 6 of Algorithm 1 by v^{k+1}, then we can express z^k and y^k as

$$z^k = \Pi(v^k) \quad \text{and} \quad y^k = \rho\left(v^k - \Pi(v^k)\right). \tag{25}$$

Observe from (25) that iterates z^k and y^k satisfy optimality condition (6) for all $k > 0$ by construction [9, Prop. 6.46]. Therefore, it only remains to show that optimality conditions (4)–(5) are satisfied in the limit.

As shown in [6, Prop. 5.3], if problem (2) is solvable, then Algorithm 1 produces a convergent sequence of iterates (x^k, z^k, y^k) so that

$$\lim_{k \to \infty} r^k_{\text{prim}} = 0,$$
$$\lim_{k \to \infty} r^k_{\text{dual}} = 0,$$

where r^k_{prim} and r^k_{dual} correspond to the residuals defined in (7) and (8) respectively.

On the other hand, if problem (2) is primal and/or dual infeasible, then the sequence of iterates (x^k, z^k, y^k) generated by Algorithm 1 does not converge. However, the sequence

$$(\delta x^k, \delta z^k, \delta y^k) = (x^k - x^{k-1}, z^k - z^{k-1}, y^k - y^{k-1})$$

always converges and can be used to certify infeasibility of the problem. According to [6, Thm. 5.1], if the problem is primal infeasible, then $\delta y = \lim_{k\to\infty} \delta y^k$ satisfies conditions (12), whereas $\delta x = \lim_{k\to\infty} \delta x^k$ satisfies conditions (13) if it is dual infeasible.

3.4 Termination criteria

We can define termination criteria for Algorithm 1 so that the iterations stop when either a primal–dual solution or a certificate of primal or dual infeasibility is found up to some predefined accuracy.

A reasonable termination criterion for detecting optimality is that the norms of the residuals r^k_{prim} and r^k_{dual} are smaller than some tolerance levels $\varepsilon_{\text{prim}} > 0$ and $\varepsilon_{\text{dual}} > 0$ [16], *i.e.*,

$$\|r^k_{\text{prim}}\|_\infty \le \varepsilon_{\text{prim}}, \quad \|r^k_{\text{dual}}\|_\infty \le \varepsilon_{\text{dual}}. \tag{26}$$

We set the tolerance levels as

$$\begin{aligned}
\varepsilon_{\text{prim}} &= \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}} \max\{\|Ax^k\|_\infty, \|z^k\|_\infty\} \\
\varepsilon_{\text{dual}} &= \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}} \max\{\|Px^k\|_\infty, \|A^T y^k\|_\infty, \|q\|_\infty\},
\end{aligned}$$

where $\varepsilon_{\text{abs}} > 0$ and $\varepsilon_{\text{rel}} > 0$ are absolute and relative tolerances, respectively.

Quadratic programs infeasibility If $\mathcal{C} = [l, u]$, we check the following conditions for primal infeasibility

$$\|A^T \delta y^k\|_\infty \le \varepsilon_{\text{pinf}} \|\delta y^k\|_\infty, \quad u^T (\delta y^k)_+ + l^T (\delta y^k)_- \le \varepsilon_{\text{pinf}} \|\delta y^k\|_\infty,$$

where $\varepsilon_{\text{pinf}} > 0$ is some tolerance level. Similarly, we define the following criterion for detecting dual infeasibility

$$\begin{aligned}
&\|P\delta x^k\|_\infty \le \varepsilon_{\text{dinf}} \|\delta x^k\|_\infty, \quad q^T \delta x^k \le \varepsilon_{\text{dinf}} \|\delta x^k\|_\infty, \\
&(A\delta x^k)_i \begin{cases} \in [-\varepsilon_{\text{dinf}}, \varepsilon_{\text{dinf}}] \|\delta x^k\|_\infty & u_i, l_i \in \mathbf{R} \\ \ge -\varepsilon_{\text{dinf}} \|\delta x^k\|_\infty & u_i = +\infty \\ \le \varepsilon_{\text{dinf}} \|\delta x^k\|_\infty & l_i = -\infty, \end{cases}
\end{aligned}$$

for $i = 1, \ldots, m$ where $\varepsilon_{\text{dinf}} > 0$ is some tolerance level. Note that $\|\delta x^k\|_\infty$ and $\|\delta y^k\|_\infty$ appear in the right-hand sides to avoid division when considering normalized vectors δx^k and δy^k in the termination criteria.

4 Solution polishing

Operator splitting methods are typically used for obtaining solution of an optimization problem with a low or medium accuracy. However, even if a solution is not very accurate we can often guess which constraints are active from an approximate primal–dual solution. When dealing with QPs of the form (2), we can obtain high accuracy solutions from the final ADMM iterates by solving one additional system of equations.

Given a dual solution y of the problem, we define the sets of lower- and upper-active constraints

$$\mathcal{L} = \{i \in \{1, \dots, m\} \mid y_i < 0\},$$
$$\mathcal{U} = \{i \in \{1, \dots, m\} \mid y_i > 0\}.$$

According to (9) we have that $z_{\mathcal{L}} = l_{\mathcal{L}}$ and $z_{\mathcal{U}} = u_{\mathcal{U}}$, where $l_{\mathcal{L}}$ denotes the vector composed of elements of l corresponding to the indices in $\mathcal{L}$. Similarly, we will denote by $A_{\mathcal{L}}$ the matrix composed of rows of A corresponding to the indices in $\mathcal{L}$.

If the sets of active constraints are known *a priori*, then a primal–dual solution (x, y, z) can be found by solving the following linear system

$$\begin{bmatrix} P & A_{\mathcal{L}}^T & A_{\mathcal{U}}^T \\ A_{\mathcal{L}} & & \\ A_{\mathcal{U}} & & \end{bmatrix} \begin{bmatrix} x \\ y_{\mathcal{L}} \\ y_{\mathcal{U}} \end{bmatrix} = \begin{bmatrix} -q \\ l_{\mathcal{L}} \\ u_{\mathcal{U}} \end{bmatrix}, \tag{27}$$
$$y_i = 0, \quad i \notin (\mathcal{L} \cup \mathcal{U}), \tag{28}$$
$$z = Ax. \tag{29}$$

We can then apply the aforementioned procedure to obtain a candidate solution (x, y, z). If (x, y, z) satisfies the optimality conditions (4)–(6), then our guess is correct and (x, y, z) is a primal–dual solution of problem (3). This approach is referred to as *solution polishing*. Note that the dimension of the linear system (27) is usually much smaller than the KKT system in Sect. 3.1 because the number of active constraints at optimality is less than or equal to n for non-degenerate QPs.

However, the linear system (27) is not necessarily solvable even if the sets of active constraints $\mathcal{L}$ and $\mathcal{U}$ have been correctly identified. This can happen, *e.g.*, if the solution is degenerate, *i.e.*, if it has one or more redundant active constraints. We make the solution polishing procedure more robust by solving instead the following linear system

$$\begin{bmatrix} P + \delta I & A_{\mathcal{L}}^T & A_{\mathcal{U}}^T \\ A_{\mathcal{L}} & -\delta I & \\ A_{\mathcal{U}} & & -\delta I \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y}_{\mathcal{L}} \\ \hat{y}_{\mathcal{U}} \end{bmatrix} = \begin{bmatrix} -q \\ l_{\mathcal{L}} \\ u_{\mathcal{U}} \end{bmatrix}, \tag{30}$$

where $\delta > 0$ is a regularization parameter with value $\delta \approx 10^{-6}$. Since the regularized matrix in (30) is quasi-definite, the linear system (30) is always solvable.

By using regularization, we actually solve a perturbed linear system and thus introduce a small error to the polished solution. If we denote by K and $(K + \Delta K)$ the coefficient matrices in (27) and (30), respectively, then we can represent the two linear

systems as $Kt = g$ and $(K + \Delta K)\hat{t} = g$. To compensate for this error, we apply an *iterative refinement* procedure [92], *i.e.*, we iteratively solve

$$(K + \Delta K)\Delta\hat{t}^k = g - K\hat{t}^k \tag{31}$$

and update $\hat{t}^{k+1} = \hat{t}^k + \Delta\hat{t}^k$. The sequence $\{\hat{t}^k\}$ converges to the true solution t, provided that it exists. Observe that, compared to solving the linear system (30), iterative refinement requires only a backward- and a forward-solve, and does not require another matrix factorization. Since the iterative refinement iterations converge very quickly in practice, we just run them for a fixed number of passes without imposing any termination condition to satisfy. Note that this is the same strategy used in commercial linear system solvers using iterative refinement [57].

5 Preconditioning and parameter selection

A known weakness of first-order methods is their inability to deal effectively with ill-conditioned problems, and their convergence rate can vary significantly when data are badly scaled. In this section we describe how to precondition the data and choose the optimal parameters to speed up the convergence of our algorithm.

5.1 Preconditioning

Preconditioning is a common heuristic aiming to reduce the number of iterations in first-order methods [76, Chap. 5], [11,44,46,47,79]. The optimal choice of preconditioners has been studied for at least two decades and remains an active area of research [61, Chap. 2], [52, Chap. 10]. For example, the optimal diagonal preconditioner required to minimize the condition number of a matrix can be found exactly by solving a semidefinite program [14]. However, this computation is typically *more* complicated than solving the original QP, and is therefore unlikely to be worth the effort since preconditioning is only a heuristic to minimize the number of iterations.

In order to keep the preconditioning procedure simple, we instead make use of a simple heuristic called *matrix equilibration* [18,27,38,89]. Our goal is to rescale the problem data to reduce the condition number of the symmetric matrix $M \in \mathbf{S}^{n+m}$ representing the problem data, defined as

$$M = \begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix}. \tag{32}$$

In particular, we use *symmetric matrix equilibration* by computing the diagonal matrix $S \in \mathbf{S}^{n+m}_{++}$ to decrease the condition number of SMS. We can write matrix S as

$$S = \begin{bmatrix} D & \\ & E \end{bmatrix}, \tag{33}$$

Algorithm 2 Modified Ruiz equilibration

initialize $c = 1$, $S = I$, $\delta = 0$, $\bar{P} = P, \bar{q} = q, \bar{A} = A, \bar{\mathcal{C}} = \mathcal{C}$
while $\|1 - \delta\|_\infty > \varepsilon_{\text{equil}}$ **do**
  for $i = 1, \ldots, n + m$ **do**
    $\delta_i \leftarrow 1/\sqrt{\|M_i\|_\infty}$ ▷ M equilibration
  $\bar{P}, \bar{q}, \bar{A}, \bar{\mathcal{C}} \leftarrow$ Scale $\bar{P}, \bar{q}, \bar{A}, \bar{\mathcal{C}}$ using **diag**(δ)
  $\gamma \leftarrow 1/\max\{\text{mean}(\|\bar{P}_i\|_\infty), \|\bar{q}\|_\infty\}$ ▷ Cost scaling
  $\bar{P} \leftarrow \gamma \bar{P}, \bar{q} \leftarrow \gamma \bar{q}$
  $S \leftarrow$ **diag**$(\delta)S, c \leftarrow \gamma c$
return S, c

where $D \in \mathbf{S}^n_{++}$ and $E \in \mathbf{S}^m_{++}$ are both diagonal. In addition, we would like to normalize the cost function to prevent the dual variables from being too large. We can achieve this by multiplying the cost function by the scalar $c > 0$.

Preconditioning effectively modifies problem (1) into the following

$$\begin{array}{ll} \text{minimize} & (1/2)\bar{x}^T \bar{P}\bar{x} + \bar{q}^T \bar{x} \\ \text{subject to} & \bar{A}\bar{x} \in \bar{\mathcal{C}}, \end{array} \tag{34}$$

where $\bar{x} = D^{-1}x$, $\bar{P} = cDPD$, $\bar{q} = cDq$, $\bar{A} = EAD$ and $\bar{\mathcal{C}} = \{Ez \in \mathbf{R}^m \mid z \in \mathcal{C}\}$. The dual variables of the new problem are $\bar{y} = cE^{-1}y$. Note that when $\mathcal{C} = [l, u]$ the Euclidean projection onto $\bar{\mathcal{C}} = [El, Eu]$ is as easy to evaluate as the projection onto $\mathcal{C}$.

The main idea of the equilibration procedure is to scale the rows of matrix M so that they all have equal ℓ_p norm. It is possible to show that finding such a scaling matrix S can be cast as a convex optimization problem [4]. However, it is computationally more convenient to solve this problem with heuristic iterative methods, rather than continuous optimization algorithms such as interior-point methods. We refer the reader to [18] for more details on matrix equilibration.

Ruiz equilibration In this work we apply a variation of the Ruiz equilibration [85]. This technique was originally proposed to equilibrate square matrices showing fast linear convergence superior to other methods such as the Sinkhorn–Knopp equilibration [86]. Ruiz equilibration converges in few tens of iterations even in cases when Sinkhorn–Knopp equilibration takes thousands of iterations [63]. The steps are outlined in Algorithm 2 and differ from the original Ruiz algorithm by adding a cost scaling step that takes into account very large values of the cost. The first part is the usual Ruiz equilibration step. Since M is symmetric, we focus only on the columns M_i and apply the scaling to both sides of M. At each iteration, we compute the ∞-norm of each column and normalize that column by the inverse of its square root. The second part is a cost scaling step. The scalar γ is the current cost normalization coefficient taking into account the maximum between the average norm of the columns of $\bar{P}$ and the norm of $\bar{q}$. We normalize problem data $\bar{P}, \bar{q}, \bar{A}, \bar{l}, \bar{u}$ in place at each iteration using the current values of δ and γ.

Unscaled termination criteria Although we rescale our problem in the form (34), we would still like to apply the stopping criteria defined in Sect. 3.4 to an unscaled version of our problem. The primal and dual residuals in (26) can be rewritten in terms of the scaled problem as

$$r_{\text{prim}}^k = E^{-1}\bar{r}_{\text{prim}}^k = E^{-1}(\bar{A}\bar{x}^k - \bar{z}^k),$$
$$r_{\text{dual}}^k = c^{-1}D^{-1}\bar{r}_{\text{dual}}^k = c^{-1}D^{-1}(\bar{P}\bar{x}^k + \bar{q} + \bar{A}^T\bar{y}^k),$$

and the tolerances levels as

$$\varepsilon_{\text{prim}} = \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}} \max\{\|E^{-1}\bar{A}\bar{x}^k\|_\infty, \|E^{-1}\bar{z}^k\|_\infty\}$$
$$\varepsilon_{\text{dual}} = \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}} c^{-1} \max\{\|D^{-1}\bar{P}\bar{x}^k\|_\infty, \|D^{-1}\bar{A}^T\bar{y}^k\|_\infty, \|D^{-1}\bar{q}\|_\infty\}.$$

Quadratic programs infeasibility When $\mathcal{C} = [l, u]$, the primal infeasibility conditions become

$$\|D^{-1}\bar{A}^T\delta\bar{y}^k\|_\infty \le \varepsilon_{\text{pinf}}\|E\delta\bar{y}^k\|_\infty, \quad \bar{u}^T(\delta\bar{y}^k)_+ + \bar{l}^T(\delta\bar{y}^k)_- \le \varepsilon_{\text{pinf}}\|E\delta\bar{y}^k\|_\infty,$$

where the primal infeasibility certificate is $c^{-1}E\delta\bar{y}^k$. The dual infeasibility criteria are

$$\|D^{-1}\bar{P}\delta\bar{x}^k\|_\infty \le c\varepsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty, \quad \bar{q}^T\delta\bar{x}^k \le c\varepsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty,$$
$$(E^{-1}\bar{A}\delta\bar{x}^k)_i \begin{cases} \in [-\varepsilon_{\text{dinf}}, \varepsilon_{\text{dinf}}]\,\|D\delta\bar{x}^k\|_\infty & u_i, l_i \in \mathbf{R} \\ \ge -\varepsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty & u_i = +\infty \\ \le \varepsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty & l_i = -\infty, \end{cases}$$

where the dual infeasibility certificate is $D\delta\bar{x}^k$.

5.2 Parameter selection

The choice of parameters (ρ, σ, α) in Algorithm 1 is a key factor in determining the number of iterations required to find an optimal solution. Unfortunately, it is still an open research question how to select the optimal ADMM parameters, see [44,47,75]. After extensive numerical testing on millions of problem instances and a wide range of dimensions, we chose the algorithm parameters as follows for QPs.

Choosing σ and α The parameter σ is a regularization term which is used to ensure that a unique solution of (15) will always exist, even when P has one or more zero eigenvalues. After scaling P in order to minimize its condition number, we choose σ as small as possible to preserve numerical stability without slowing down the algorithm. We set the default value as $\sigma = 10^{-6}$. The relaxation parameter α in the range $[1.5, 1.8]$ has empirically shown to improve the convergence rate [34,35]. In the proposed method, we set the default value of $\alpha = 1.6$.

Choosing ρ The most crucial parameter is the step-size ρ. Numerical testing showed that having different values of ρ for different constraints, can greatly improve the performance. For this reason, without altering the algorithm steps, we chose $\rho \in \mathbf{S}^m_{++}$ being a positive definite diagonal matrix with different elements ρ_i.

For a specific QP, if we know the active and inactive constraints, then we can rewrite it simply as an equality constrained QP. In this case the optimal ρ is defined as $\rho_i = \infty$ for the active constraints and $\rho_i = 0$ for the inactive constraints, therefore reducing the linear system (24) to the optimality conditions of the equivalent equality constrained QP (after setting $\sigma = 0$). Unfortunately, it is impossible to know a priori whether any given constraint is active or inactive at optimality, so we must instead adopt some heuristics. We define ρ as follows

$$\rho = \mathbf{diag}(\rho_1, \ldots, \rho_m), \quad \rho_i = \begin{cases} \bar{\rho} & l_i \neq u_i \\ 10^3 \bar{\rho} & l_i = u_i, \end{cases}$$

where $\bar{\rho} > 0$. In this way we assign a high value to the step-size related to the equality constraints since they will be active at the optimum. Having a fixed value of $\bar{\rho}$ cannot provide fast convergence for different kind of problems since the optimal solution and the active constraints vary greatly. To compensate for this issue, we adopt an adaptive scheme which updates $\bar{\rho}$ during the iterations based on the ratio between primal and dual residuals. The idea of introducing "feedback" in the algorithm steps makes ADMM more robust to bad scaling in the data; see [16,54,93]. Contrary to the adaptation approaches in the literature where the update increases or decreases the value of the step-size by a fixed amount, we adopt the following rule

$$\bar{\rho}^{k+1} \leftarrow \bar{\rho}^k \sqrt{\frac{\|\bar{r}^k_{\text{prim}}\|_\infty / \max\{\|\bar{A}\bar{x}^k\|_\infty, \|\bar{z}^k\|_\infty\}}{\|\bar{r}^k_{\text{dual}}\|_\infty / \max\{\|\bar{P}\bar{x}^k\|_\infty, \|\bar{A}^T\bar{y}^k\|_\infty, \|\bar{q}\|_\infty\}}}.$$

In other words we update $\bar{\rho}^k$ using the square root of the ratio between the scaled residuals normalized by the magnitudes of the relative part of the tolerances. We set the initial value as $\bar{\rho}^0 = 0.1$. In our benchmarks, if $\bar{\rho}^0$ does not already give a low number of ADMM iterations, it gets usually tuned with a maximum of 1 or 2 updates. The adaptation causes the KKT matrix in (24) to change and, if the linear system solver solution method is direct, it requires a new numerical factorization. We do not require a new symbolic factorization because the sparsity pattern of the KKT matrix does not change. Since the numerical factorization can be costly, we perform the adaptation only when it is really necessary. In particular, we allow an update if the accumulated iterations time is greater than a certain percentage of the factorization time (nominally 40%) and if the new parameter is sufficiently different than the current one, *i.e.*, 5 times larger or smaller. Note that in the case of an indirect method this rule allows for more frequent changes of ρ since there is no need to factor the KKT matrix and the update is numerically much cheaper. Note that the convergence of the ADMM algorithm is hard to prove in general if the ρ updates happen at each iteration. However, if we

assume that the updates stop after a fixed number of iterations the convergence results hold [16, Section 3.4.1].

6 Parametric programs

In application domains such as control, statistics, finance, and SQP, problem (1) is solved repeatedly for varying data. For these problems, usually referred to as *parametric programs*, we can speed up the repeated OSQP calls by re-using the computations across multiple solves.

We make the distinction between cases in which only the vectors or all data in (1) change between subsequent problem instances. We assume that the problem dimensions n and m and the sparsity patterns of P and A are fixed.

Vectors as parameters If the vectors q, l, and u are the only parameters that vary, then the KKT coefficient matrix in Algorithm 1 does not change across different instances of the parametric program. Thus, if a direct method is used, we perform and store its factorization only once before the first solution and reuse it across all subsequent iterations. Since the matrix factorization is the computationally most expensive step of the algorithm, this approach reduces significantly the amount of time OSQP takes to solve subsequent problems. This class of problems arises very frequently in many applications including linear MPC and MHE [2,82], Lasso [19,90], and portfolio optimization [15,67].

Matrices and vectors as parameters We separately consider the case in which the values (but not the locations) of the nonzero entries of matrices P and A are updated. In this case, in a direct method, we need to refactor the matrix in Algorithm 1. However, since the sparsity pattern does not change we need only to recompute the *numerical factorization* while reusing the *symbolic factorization* from the previous solution. This results in a modest reduction in the computation time. This class of problems encompasses several applications such as nonlinear MPC and MHE [28] and sequential quadratic programming [76].

Warm starting In contrast to interior-point methods, OSQP is easily initialized by providing an initial guess of both the primal and dual solutions to the QP. This approach is known as *warm starting* and is particularly effective when the subsequent QP solutions do not vary significantly, which is the case for most *parametric programs* applications. We can warm start the ADMM iterates from the previous OSQP solution $(x^\star, y^\star)$ by setting $(x^0, z^0, y^0) \leftarrow (x^\star, Ax^\star, y^\star)$. Note that we can warm-start the ρ estimation described in Sect. 7 to exploit the ratio between the primal and dual residuals to speed up convergence in subsequent solves.

7 OSQP

We have implemented our proposed approach in the "Operator Splitting Quadratic Program" (OSQP) solver, an open-source software package in the C language. OSQP can solve any QP of the form (2) and makes no assumptions about the problem data other than convexity. OSQP is available online at

https://osqp.org.

Users can call OSQP from C, C++, Fortran, Python, Matlab, R, Julia, Ruby and Rust, and via parsers such as CVXPY [1,26], JuMP [33], and YALMIP [65].

To exploit the data sparsity pattern, OSQP accepts matrices in Compressed-Sparse-Column (CSC) format [24]. We implemented the linear system solution described in Sect. 3.1 as an object-oriented interface to easily switch between efficient algorithms. At present, OSQP ships with the open-source QDLDL direct solver which is our independent implementation based on [23], and also supports dynamic loading of more advanced algorithms such as the MKL Pardiso direct solver [57]. We plan to add iterative indirect solvers and other direct solvers in future versions.

The default values for the OSQP termination tolerances described in Sect. 3.4 are

$$\varepsilon_{\text{abs}} = \varepsilon_{\text{rel}} = 10^{-3}, \quad \varepsilon_{\text{pinf}} = \varepsilon_{\text{dinf}} = 10^{-4}.$$

The default step-size parameter σ and the relaxation parameter α are set to

$$\sigma = 10^{-6}, \quad \alpha = 1.6,$$

while ρ is automatically chosen by default as described in Sect. 5.2, with optional user override. We set the default fixed number of iterative refinement steps to 3.

OSQP reports the total computation time divided by the time required to perform preprocessing operations such as scaling or matrix factorization and the time to carry out the ADMM iterations. If the solver is called multiple times reusing the same matrix factorization, it will report only the ADMM solve time as total computation time. For more details we refer the reader to the solver documentation on the OSQP project website.

8 Numerical examples

We benchmarked OSQP against the open-source interior-point solver ECOS [30], the open-source active-set solver qpOASES [36], and the commercial interior-point solvers GUROBI [53] and MOSEK [73]. We executed every benchmark comparing different solvers with both low accuracy, *i.e.*, $\varepsilon_{\text{abs}} = \varepsilon_{\text{rel}} = 10^{-3}$, and high accuracy, *i.e.*, $\varepsilon_{\text{abs}} = \varepsilon_{\text{rel}} = 10^{-5}$. We set GUROBI, ECOS, MOSEK and OSQP primal and dual feasibility tolerances to our low and high accuracy tolerances. Since qpOASES is an active-set method and does not allow the user to tune primal nor dual feasibility tolerances, we set it to its default termination settings. In addition, the maximum time we allow each solver to run is 1000 sec and no limit on the maximum number of

iterations. Note that the use of maximum time limits with no bounds on the number of iterations is the default setting in commercial solvers such as MOSEK. For every solver we leave all the other settings to the internal defaults.

In general it is hard to compare the solution accuracies because all the solvers, especially commercial ones, use an internal problem scaling and verify that the termination conditions are satisfied against their scaled version of the problem. In contrast, OSQP allows the option to check the termination conditions against the internally scaled or the original problem. Therefore, to make the benchmark fair, we say that the primal–dual solution $(x^\star, y^\star)$ returned by each solver is optimal if the following optimality conditions are satisfied with tolerances defined above with low and high accuracy modes,

$$\|(Ax^\star - u)_+ + (Ax^\star - l)_-\|_\infty \leq \varepsilon_{\text{prim}}, \qquad \|Px^\star + q + A^T y^\star\|_\infty \leq \varepsilon_{\text{dual}},$$

where $\varepsilon_{\text{prim}}$ and $\varepsilon_{\text{dual}}$ are defined in Sect. 3.4. If the primal–dual solution returned by a solver does not satisfy the optimality conditions defined above, we consider it a failure. Note that we decided not to include checks on the complementary slackness satisfaction because interior-point solvers satisfied them with different metrics and scalings, therefore failing very often. In contrast OSQP always satisfies complementary slackness conditions with machine precision by construction.

In addition, we used the direct single-threaded linear system solver QDLDL [50] based on [3,23] and very simple linear algebra where other solvers such as GUROBI and MOSEK use advanced multi-threaded linear system solvers and custom linear algebra.

All the experiments were carried out on the MIT SuperCloud facility in collaboration with the Lincoln Laboratory [83] with 16 Intel Xeon E5-2650 cores. The code for all the numerical examples is available online at [88].

Shifted geometric mean As in most common benchmarks [72], we make use of the normalized shifted geometric mean to compare the timings of the various solvers. Given the time required by solver s to solve problem p $t_{p,s}$, we define the shifted geometric mean as

$$g_s = \sqrt[n]{\prod_p (t_{p,s} + k)} - k,$$

where n is the number of problem instances considered and $k = 1$ is the shift [72]. The normalized shifted geometric mean is therefore

$$r_s = g_s / \min_s g_s.$$

This value shows the factor at which a specific solver is slower than the fastest one with scaled value of 1.00. If solver s fails at solving problem p, we set the time as the maximum allowed, *i.e.*, $t_{p,s} = 1000$ sec. Note that to avoid memory overflows in the product, we compute in practice the shifted geometric mean as $e^{\ln g_s}$.

Performance profiles We also make use of the performance profiles [29] to compare the solver timings. We define the performance ratio

$$u_{p,s} = t_{p,s} / \min_s t_{p,s}.$$

The performance profile plots the function $f_s : \mathbf{R} \mapsto [0, 1]$ defined as

$$f_s(\tau) = \frac{1}{n} \sum_p \mathcal{I}_{\leq \tau}(u_{p,s}),$$

where $\mathcal{I}_{\leq \tau}(u_{p,s}) = 1$ if $u_{p,s} \leq \tau$ or 0 otherwise. The value $f_s(\tau)$ corresponds to the fraction of problems solved within τ times from the best solver. Note that while we cannot necessarily assess the performance of one solver relative to another with performance profiles, they still represent a viable choice to benchmark the performance of a solver with respect to the best one [51].

8.1 Benchmark problems

We considered QPs in the form (2) from 7 problem classes ranging from standard random programs to applications in the areas of control, portfolio optimization and machine learning. For each problem class, we generated 10 different instances for 20 dimensions giving a total of 1400 problem instances. All instances were obtained from either real data or from non-trivial random data. Note that the random QPs and random equality constrained QPs problem classes might not closely correspond to a real-world application. However, they have a typical number of nonzero elements appearing in practice. We described generation for each class in "Appendix A". Throughout all the problem classes, n ranges between 10^1 and 10^4, m between 10^2 and 10^5, and the number of nonzeros N between 10^2 and 10^8.

Results We show in Figs. 1 and 2 the OSQP and GUROBI computation times across all the problem classes for low and high accuracy solutions respectively. OSQP is competitive or even faster than GUROBI for several problem classes. Results are shown in Table 1 and Fig. 3. OSQP shows the best performance across these benchmarks with MOSEK performing better at lower accuracy and GUROBI at higher accuracy. ECOS is generally slower than the other interior-point solvers but faster than qpOASES that shows issues with many constraints. Table 2 contains the OSQP statistics for this benchmark class. Because of the good convergence behavior of OSQP on these problems, the setup time is significant compared to the solve time, especially at low accuracy. Solution polishing increases the solution time by a median of 10 to 20 percent due to the additional factorization used. The worst-case time increase is very high and happens for the problems that converge in very few iterations. Note that with high accuracy, polishing succeeds in 83% of test cases while on low accuracy it succeeds in only 42% of cases. The number of ρ updates is in general very low, usually requiring just more matrix factorization to adjust, with up to 5 refactorisations used in the worst case when solving with high accuracy.

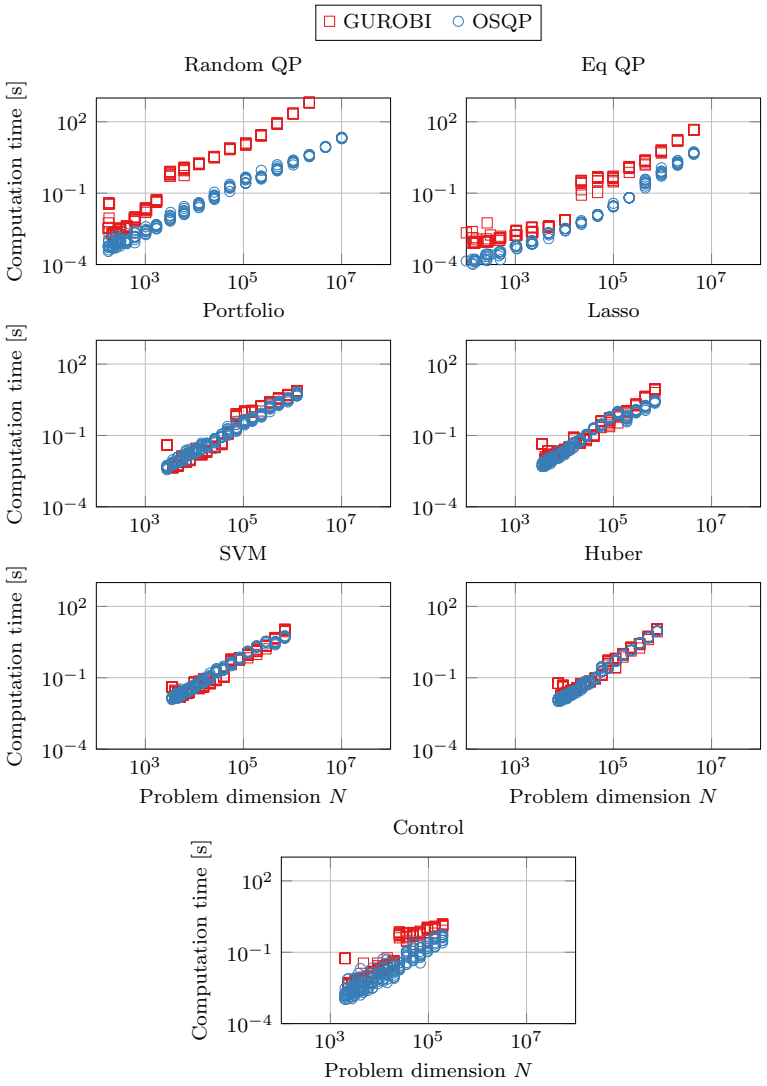

Fig. 1 Computation time vs problem dimension for OSQP and GUROBI for low accuracy mode

8.2 SuiteSparse matrix collection least squares problems

We considered 30 least squares problem in the form $Ax \approx b$ from the SuiteSparse Matrix Collection library [25]. Using the Lasso and Huber problem setups from "Appendix A" we formulate 60 QPs that we solve with OSQP, GUROBI and MOSEK. We excluded ECOS because its interior-point algorithm showed numerical issues for several problems of the test set. We also excluded qpOASES because it is not designed for large linear systems.

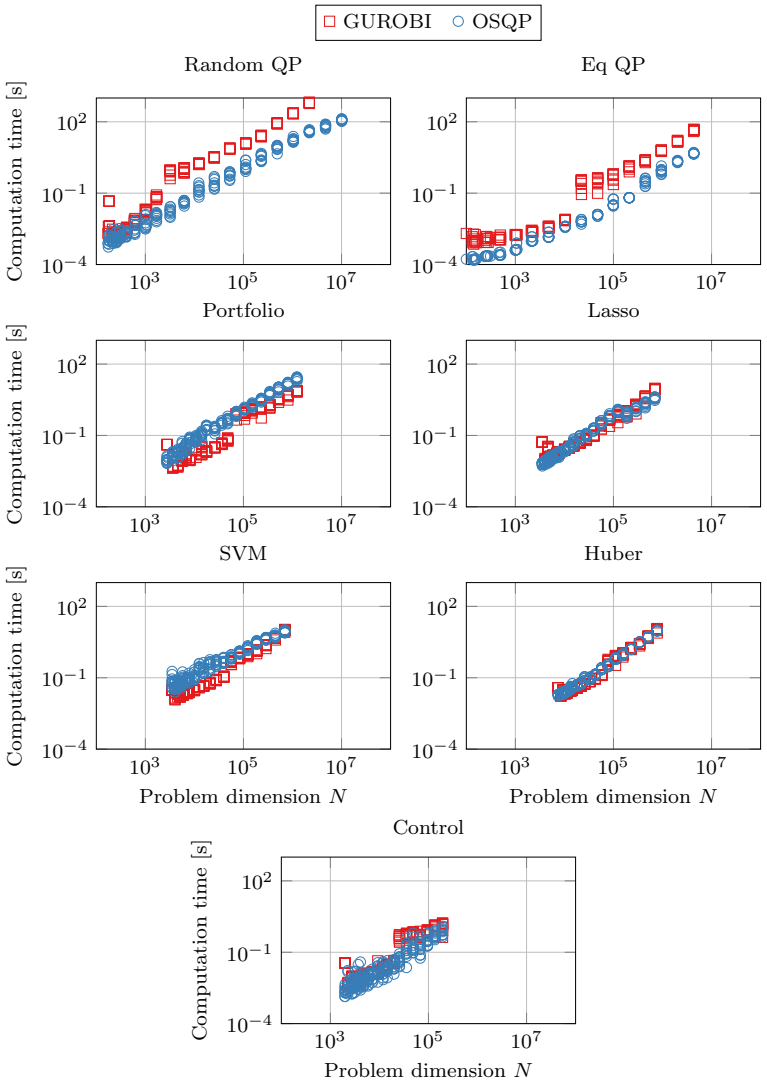

Fig. 2 Computation time vs problem dimension for OSQP and GUROBI for high accuracy mode

Table 1 Benchmark problems comparison with timings as shifted geometric mean and failure rates

		OSQP	GUROBI	MOSEK	ECOS	qpOASES
Shifted geometric means	Low accuracy	1.000	4.285	2.522	28.847	149.932
	High accuracy	1.000	1.886	6.234	52.718	66.254
Failure rates [%]	Low accuracy	0.000	1.429	0.071	20.714	31.857
	High accuracy	0.000	1.429	11.000	45.571	31.714

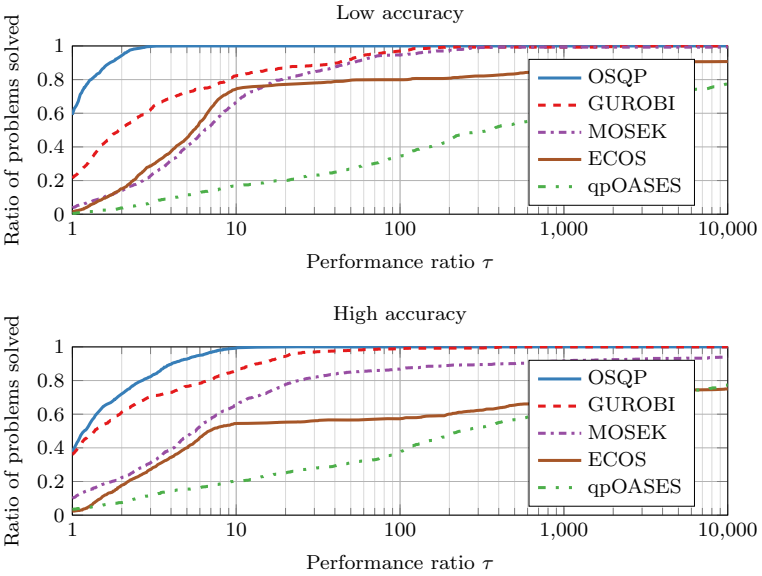

Fig. 3 Benchmark problems comparison with performance profiles

Table 2 Benchmark problems OSQP statistics

		Median	Max
Setup/solve time [%]	Low accuracy	60.23	1550.19
	High accuracy	29.65	1373.18
Polish time increase [%]	Low accuracy	19.20	876.80
	High accuracy	10.63	1408.83
Number of ρ updates	Low accuracy	1	3
	High accuracy	1	5
		Mean	
Polish success [%]	Low accuracy	42.79	
	High accuracy	83.21	

Results Results are shown in Table 3 and Fig. 4. OSQP shows the best performance with GUROBI slightly slower and MOSEK third. The failure rates for GUROBI and MOSEK are higher because the reported solution does not satisfy the optimality conditions of the original problem. We display the OSQP statistics in Table 4. The setup phase takes a significant amount of time compared to the solve phase, especially when OSQP converges in a few iterations. This happens because the large problem dimensions result in a large initial factorization time. Polish time is in general 22 to 32% of the total solution time. However, the success is usually reliable, succeeding 78% of the times with very high quality solutions. The number of matrix refactorizations

Table 3 SuiteSparse matrix problems comparison with timings as shifted geometric mean and failure rates

		OSQP	GUROBI	MOSEK
Shifted geometric means	Low accuracy	1.000	1.630	1.745
	High accuracy	1.000	1.489	4.498
Failure rates [%]	Low accuracy	0.000	14.286	12.500
	High accuracy	1.786	16.071	33.929

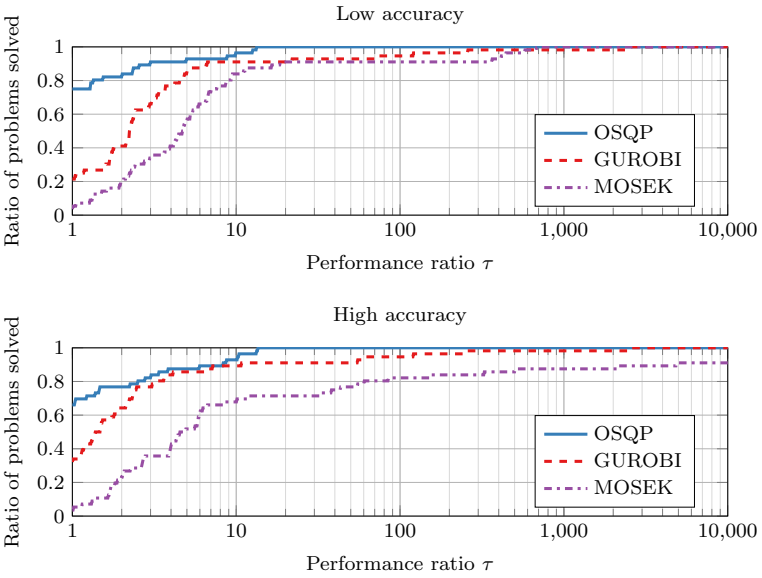

Fig. 4 SuiteSparse matrix problems comparison with performance profiles

required due to ρ updates is very low in these examples, with a maximum of 2 or 3 even for high accuracy.

8.3 Maros–Mészáros problems

We considered the Maros–Mészáros test set [68] of hard QPs. We compared the OSQP solver against GUROBI and MOSEK against all the problems in the set. We decided to exclude ECOS because its interior-point algorithm showed numerical issues for several problems of the test set. We also excluded qpOASES because it could not solve most of the problems since it is not suited for large QPs – it is based on an active-set method with dense linear algebra.

Results Results are shown in Table 5 and Fig. 5. GUROBI shows the best performance and OSQP, while slower, is still competitive on both low and high accuracy tests. MOSEK remains the slowest in every case. Table 6 shows the statistics relative to OSQP. Since these hard problems require a larger number of iterations to converge,

Table 4 SuiteSparse problems OSQP statistics

		Median	Max
Setup/solve time [%]	Low accuracy	71.37	2910.37
	High accuracy	48.03	1451.56
Polish time increase [%]	Low accuracy	32.27	178.23
	High accuracy	22.68	115.77
Number of ρ updates	Low accuracy	0	2
	High accuracy	1	3
		Mean	
Polish success [%]	Low accuracy	67.86	
	High accuracy	78:18	

Table 5 Maros–Mészáros problems comparison with timings as shifted geometric mean and failure rates

		OSQP	GUROBI	MOSEK
Shifted geometric means	Low accuracy	1.464	1.000	6.121
	High accuracy	5.247	1.000	14.897
Failure rates [%]	Low accuracy	1.449	2.174	14.493
	High accuracy	10.145	2.899	30.435

the setup time overhead compared to the solution time is in general lower than the other benchmark sets. Moreover, since the problems are badly scaled and degenerate, the polishing strategy rarely succeeds. However, the median time increase from the polish step is less than 10% of the total computation time for both low and high accuracy modes. Note that the number of ρ updates is usually very low with a median of 1 or 2. However, there are some worst-case problems when it is very high because the bad scaling causes issues in our ρ estimation. However, from our data we have seen that in more than 95% of the cases the number of ρ updates is less than 5.

8.4 Warm start and factorization caching

To show the benefits of warm starting and factorization caching, we solved a sequence of QPs using OSQP with the data varying according to some parameters. Since we are not comparing OSQP with other high accuracy solvers in these benchmarks, we use its default settings with accuracy 10^{-3}.

Lasso regularization path We solved a Lasso problem described in "Appendix A.5" with varying λ in order to choose a regressor with good validation set performance. We solved one problem instance with $n = 50, 100, 150, 200$ features, $m = 100n$ data points, and λ logarithmically spaced taking 100 values between $\lambda_{\max} = \|A^T b\|_\infty$ and $0.01\lambda_{\max}$.

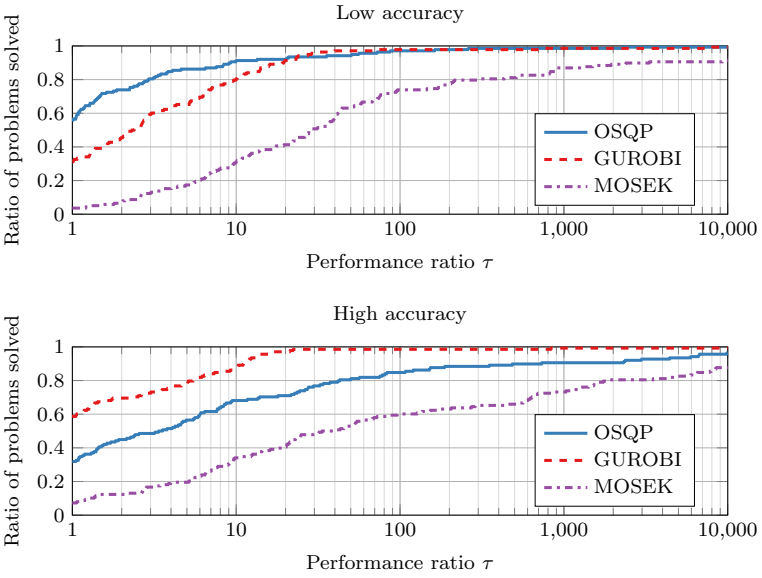

Fig. 5 Maros–Mészáros problems comparison with performance profiles

Table 6 Maros–Mészáros problems OSQP statistics

		Median	Max
Setup/solve time [%]	Low accuracy	31.59	643.29
	High accuracy	2.89	326.11
Polish time increase [%]	Low accuracy	9.49	127.55
	High accuracy	1.55	76.36
Number of ρ updates	Low accuracy	1	70
	High accuracy	2	2498
		Mean	
Polish success [%]	Low accuracy	30.15	
	High accuracy	37.90	

Since the parameters only enter linearly in the cost, we can reuse the matrix factorization and enable warm starting to reduce the computation time as discussed in Sect. 6.

Model predictive control In MPC, we solve the optimal control problem described in "Appendix A.3" at each time step to compute an optimal input sequence over the horizon. Then, we apply only the first input to the system and propagate the state to the next time step. The whole procedure is repeated with an updated initial state x_{init}. We solved the control problem with $n_x = 20, 40, 60, 80$ states, $n_u = n_x/2$

inputs, horizon $T = 10$ and 100 simulation steps. The initial state of the simulation is uniformly distributed and constrained to be within the feasible region, *i.e.*, $x_{\text{init}} \sim \mathcal{U}(-0.5\overline{x}, 0.5\overline{x})$.

Since the parameters only enter linearly in the constraints bounds, we can reuse the matrix factorization and enable warm starting to reduce the computation time as discussed in Sect. 6.

Portfolio back test Consider the portfolio optimization problem in "Appendix A.4" with $n = 10k$ assets and $k = 100, 200, 300, 400$ factors.

We run a 4 years back test to compute the optimal assets investment depending on varying expected returns and factor models [13]. We solved 240 QPs per year giving a total of 960 QPs. Each month we solved 20 QPs corresponding to the trading days. Every day, we updated the expected returns μ by randomly generating another vector with $\mu_i \sim 0.9\hat{\mu}_i + \mathcal{N}(0, 0.1)$, where $\hat{\mu}_i$ comes from the previous expected returns. The risk model was updated every month by updating the nonzero elements of D and F according to $D_{ii} \sim 0.9\hat{D}_{ii} + \mathcal{U}[0, 0.1\sqrt{k}]$ and $F_{ij} \sim 0.9\hat{F}_{ij} + \mathcal{N}(0, 0.1)$ where $\hat{D}_{ii}$ and $\hat{F}_{ij}$ come from the previous risk model.

As discussed in Sect. 6, we exploited the following computations during the QP updates to reduce the computation times. Since μ only enters in the linear part of the objective, we can reuse the matrix factorization and enable warm starting. Since the sparsity patterns of D and F do not change during the monthly updates, we can reuse the symbolic factorization and exploit warm starting to speed up the computations.

Results We show the results in Table 7. For the Lasso problem we see more than ten-fold improvement in time and between 8 and 11 times reduction in number of iterations depending on the dimension. For the MPC problem the number of iterations does not significantly decrease because the number of iterations is already low in cold-start. However we get from 2.6 to four-fold time improvement from factorization caching. OSQP shows from 5.8 to 7 times reduction in time for the portfolio problem and from 2.9 to 3.6 times reduction in number of iterations.

9 Conclusions

We presented a novel general-purpose QP solver based on ADMM. Our method uses a new splitting requiring the solution of a quasi-definite linear system that is always solvable independently from the problem data. We impose no assumptions on the problem data other than convexity, resulting in a general-purpose and very robust algorithm.

For the first time, we propose a first-order QP solution method able to provide primal and dual infeasibility certificates if the problem is unsolvable without resorting to homogeneous self-dual embedding or additional complexity in the iterations.

Table 7 OSQP parametric problem results with warm start (ws) and without warm start (no ws) in terms of time in seconds and number of iterations for different leading problem dimensions of Lasso, MPC and Portfolio classes

Problem	dim.	Time no ws	Time ws	Time improv.	Iter no ws	Iter ws	Iter improv.
Lasso	50	0.225	0.012	19.353	210.250	25.750	8.165
	100	0.423	0.040	10.556	224.000	25.750	8.699
	150	1.022	0.086	11.886	235.500	25.750	9.146
	200	2.089	0.149	13.986	281.750	26.000	10.837
MPC	20	0.007	0.002	4.021	89.500	32.750	2.733
	40	0.014	0.005	2.691	29.000	27.250	1.064
	60	0.035	0.013	2.673	33.750	33.000	1.023
	80	0.067	0.022	3.079	32.000	31.750	1.008
Portfolio	100	0.177	0.030	5.817	93.333	25.417	3.672
	200	0.416	0.061	6.871	86.875	25.391	3.422
	300	0.646	0.097	6.635	80.521	25.521	3.155
	400	0.976	0.139	7.003	76.458	26.094	2.930

In contrast to other first-order methods, our solver can provide high-quality solutions by performing *solution polishing*. After guessing which constraints are active, we compute the solutions of an additional small equality constrained QP by solving a linear system. If the constraints are identified correctly, the returned solution has accuracy equal or higher than interior-point methods.

The proposed method is easily warm started to reduce the number of iterations. If the problem matrices do not change, the linear system matrix factorization can be cached and reused across multiple solves greatly improving the computation time. This technique can be extremely effective, especially when solving parametric QPs where only part of the problem data change.

We have implemented our algorithm in the open-source OSQP solver written in C and interfaced with multiple other languages and parsers. OSQP is based on sparse linear algebra and is able to exploit the structure of QPs arising in different application areas. OSQP is robust against noisy and unreliable data and, after the first factorization is computed, can be compiled to be library-free and division-free, making it suitable for embedded applications. Thanks to its simple and parallelizable iterations, OSQP can handle large-scale problems with millions of nonzeros.

We extensively benchmarked the OSQP solver with problems arising in several application domains including finance, control and machine learning. In addition, we benchmarked it against the hard problems from the Maros–Mészáros test set [68] and Lasso and Huber fitting problems generated with sparse matrices from the SuiteSparse Matrix Collection [25]. Timing and failure rate results showed great improvements over state-of-the-art academic and commercial QP solvers.

OSQP has already a large userbase with tens of thousands of users both from top academic institutions and large corporations.

A Problem classes

In this section we describe the random problem classes used in the benchmarks and derive formulations with explicit linear equalities and inequalities that can be directly written in the form $Ax \in \mathcal{C}$ with $\mathcal{C} = [l, u]$.

A.1 Random QP

Consider the following QP

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x \\ \text{subject to} & l \leq Ax \leq u. \end{array}$$

Problem instances The number of variables and constraints in our problem instances are n and $m = 10n$. We generated random matrix $P = MM^T + \alpha I$ where $M \in \mathbf{R}^{n \times n}$ and 15% nonzero elements $M_{ij} \sim \mathcal{N}(0, 1)$. We add the regularization αI with $\alpha = 10^{-2}$ to ensure that the problem is not unbounded. We set the elements of $A \in \mathbf{R}^{m \times n}$ as $A_{ij} \sim \mathcal{N}(0, 1)$ with only 15% being nonzero. The linear part of the cost is normally distributed, *i.e.*, $q_i \sim \mathcal{N}(0, 1)$. We generated the constraint bounds as $u_i \sim \mathcal{U}(0, 1)$, $l_i \sim -\mathcal{U}(0, 1)$.

A.2 Equality constrained QP

Consider the following equality constrained QP

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x \\ \text{subject to} & Ax = b. \end{array}$$

This problem can be rewritten as (1) by setting $l = u = b$.

Problem instances The number of variables and constraints in our problem instances are n and $m = \lfloor n/2 \rfloor$.

We generated random matrix $P = MM^T + \alpha I$ where $M \in \mathbf{R}^{n \times n}$ and 15% nonzero elements $M_{ij} \sim \mathcal{N}(0, 1)$. We add the regularization αI with $\alpha = 10^{-2}$ to ensure that the problem is not unbounded. We set the elements of $A \in \mathbf{R}^{m \times n}$ as $A_{ij} \sim \mathcal{N}(0, 1)$ with only 15% being nonzero. The vectors are all normally distributed, *i.e.*, $q_i, b_i \sim \mathcal{N}(0, 1)$.

Iterative refinement interpretation Solution of the above problem can be found directly by solving the following linear system

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \nu \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}. \tag{35}$$

If we apply the ADMM iterations (15)–(19) for solving the above problem, and by setting $\alpha = 1$ and $y^0 = b$, the algorithm boils down to the following iteration

$$\begin{bmatrix} x^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} x^k \\ \nu^k \end{bmatrix} + \begin{bmatrix} P + \sigma I & A^T \\ A & -\rho^{-1} I \end{bmatrix}^{-1} \left(\begin{bmatrix} -q \\ b \end{bmatrix} - \begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^k \\ \nu^k \end{bmatrix} \right),$$

which is equivalent to (31) with $g = (-q, b)$ and $\hat{t}^k = (x^k, \nu^k)$. This means that Algorithm 1 applied to solve an equality constrained QP is equivalent to applying iterative refinement [32,92] to solve the KKT system (35). Note that the perturbation matrix in this case is

$$\Delta K = \begin{bmatrix} \sigma I & \\ & -\rho^{-1} I \end{bmatrix},$$

which justifies using a low value of σ and a high value of ρ for equality constraints.

A.3 Optimal control

We consider the problem of controlling a constrained linear time-invariant dynamical system. To achieve this, we formulate the following optimization problem [12]

$$\begin{array}{ll} \text{minimize} & x_T^T Q_T x_T + \sum_{t=0}^{T-1} x_t^T Q x_t + u_t^T R u_t \\ \text{subject to} & x_{t+1} = A x_t + B u_t \\ & x_t \in \mathcal{X}, u_t \in \mathcal{U} \\ & x_0 = x_{\text{init}}. \end{array} \tag{36}$$

The states $x_t \in \mathbf{R}^{n_x}$ and the inputs $u_k \in \mathbf{R}^{n_u}$ are subject to polyhedral constraints defined by the sets $\mathcal{X}$ and $\mathcal{U}$. The horizon length is T and the initial state is $x_{\text{init}} \in \mathbf{R}^{n_x}$. Matrices $Q \in \mathbf{S}_+^{n_x}$ and $R \in \mathbf{S}_{++}^{n_u}$ define the state and input costs at each stage of the horizon, and $Q_T \in \mathbf{S}_+^{n_x}$ defines the final stage cost.

By defining the new variable $z = (x_0, \ldots, x_T, u_0, \ldots, u_{T-1})$, problem (36) can be written as a sparse QP of the form (2) with a total of $n_x(T+1) + n_u T$ variables.

Problem instances We defined the linear systems with $n = n_x$ states and $n_u = 0.5 n_x$ inputs. We set the horizon length to $T = 10$. We generated the dynamics as $A = I + \Delta$ with $\Delta_{ij} \sim \mathcal{N}(0, 0.01)$. We chose only stable dynamics by enforcing the norm of the eigenvalues of A to be less than 1. The input action is modeled as B with $B_{ij} \sim \mathcal{N}(0, 1)$.

The state cost is defined as $Q = \mathbf{diag}(q)$ where $q_i \sim \mathcal{U}(0, 10)$ and 70% nonzero elements in q. We chose the input cost as $R = 0.1I$. The terminal cost Q_T is chosen as the optimal cost for the linear quadratic regulator (LQR) applied to A, B, Q, R by solving a discrete algebraic Riccati equation (DARE) [12]. We generated input and state constraints as

$$\mathcal{X} = \{x_t \in \mathbf{R}^{n_x} \mid -\overline{x} \leq x_t \leq \overline{x}\}, \qquad \mathcal{U} = \{u_t \in \mathbf{R}^{n_u} \mid -\overline{u} \leq u_t \leq \overline{u}\},$$

where $\overline{x}_i \sim \mathcal{U}(1, 2)$ and $\overline{u}_i \sim \mathcal{U}(0, 0.1)$. The initial state is uniformly distributed with $x_{\text{init}} \sim \mathcal{U}(-0.5\overline{x}, 0.5\overline{x})$.

A.4 Portfolio optimization

Portfolio optimization is a problem arising in finance that seeks to allocate assets in a way that maximizes the risk adjusted return [13,15,67], [17, §4.4.1],

$$\begin{array}{ll} \text{maximize} & \mu^T x - \gamma(x^T \Sigma x) \\ \text{subject to} & \mathbf{1}^T x = 1 \\ & x \geq 0, \end{array}$$

where the variable $x \in \mathbf{R}^n$ represents the portfolio, $\mu \in \mathbf{R}^n$ the vector of expected returns, $\gamma > 0$ the risk aversion parameter, and $\Sigma \in \mathbf{S}^n_+$ the risk model covariance matrix. The risk model is usually assumed to be the sum of a diagonal and a rank $k < n$ matrix

$$\Sigma = FF^T + D,$$

where $F \in \mathbf{R}^{n \times k}$ is the factor loading matrix and $D \in \mathbf{R}^{n \times n}$ is a diagonal matrix describing the asset-specific risk.

We introduce a new variable $y = F^T x$ and solve the resulting problem in variables x and y

$$\begin{array}{ll} \text{minimize} & x^T Dx + y^T y - \gamma^{-1} \mu^T x \\ \text{subject to} & y = F^T x \\ & \mathbf{1}^T x = 1 \\ & x \geq 0, \end{array} \tag{37}$$

Note that the Hessian of the objective in (37) is a diagonal matrix. Also, observe that FF^T does not appear in problem (37).

Problem instances We generated portfolio problems for increasing number of factors k and number of assets $n = 100k$. The elements of matrix F were chosen as $F_{ij} \sim \mathcal{N}(0, 1)$ with 50% nonzero elements. The diagonal matrix D is chosen as $D_{ii} \sim \mathcal{U}[0, \sqrt{k}]$. The mean return was generated as $\mu_i \sim \mathcal{N}(0, 1)$. We set $\gamma = 1$.

A.5 Lasso

The *least absolute shrinkage and selection operator (Lasso)* is a well known linear regression technique obtained by adding an ℓ_1 regularization term in the objective [19,90]. It can be formulated as

$$\text{minimize} \quad \|Ax - b\|_2^2 + \lambda \|x\|_1,$$

where $x \in \mathbf{R}^n$ is the vector of parameters and $A \in \mathbf{R}^{m \times n}$ is the data matrix and λ is the weighting parameter.

We convert this problem to the following QP

$$\begin{array}{ll} \text{minimize} & y^T y + \lambda \mathbf{1}^T t \\ \text{subject to} & y = Ax - b \\ & -t \leq x \leq t, \end{array}$$

where $y \in \mathbf{R}^m$ and $t \in \mathbf{R}^n$ are two newly introduced variables.

Problem instances The elements of matrix A are generated as $A_{ij} \sim \mathcal{N}(0, 1)$ with 15% nonzero elements. To construct the vector b, we generated the true sparse vector $v \in \mathbf{R}^n$ to be learned

$$v_i \sim \begin{cases} 0 & \text{with probability } p = 0.5 \\ \mathcal{N}(0, 1/n) & \text{otherwise.} \end{cases}$$

Then we let $b = Av + \varepsilon$ where ε is the noise generated as $\varepsilon_i \sim \mathcal{N}(0, 1)$. We generated the instances with varying n features and $m = 100n$ data points. The parameter λ is chosen as $(1/5)\|A^T b\|_\infty$ since $\|A^T b\|_\infty$ is the critical value above which the solution of the problem is $x = 0$.

A.6 Huber fitting

Huber fitting or the *robust least-squares problem* performs linear regression under the assumption that there are outliers in the data [55,56]. The fitting problem is written as

$$\text{minimize} \quad \sum_{i=1}^m \phi_{\text{hub}}(a_i^T x - b_i), \tag{38}$$

with the Huber penalty function $\phi_{\text{hub}} : \mathbf{R} \to \mathbf{R}$ defined as

$$\phi_{\text{hub}}(u) = \begin{cases} u^2 & |u| \leq M \\ M(2|u| - M) & |u| > M. \end{cases}$$

Problem (38) is equivalent to the following QP [66, Eq. (24)]

$$\begin{array}{ll} \text{minimize} & u^T u + 2M\mathbf{1}^T(r + s) \\ \text{subject to} & Ax - b - u = r - s \\ & r \geq 0 \\ & s \geq 0. \end{array}$$

Problem instances We generate the elements of A as $A_{ij} \sim \mathcal{N}(0, 1)$ with 15% nonzero elements. To construct $b \in \mathbf{R}^m$ we first generate a vector $v \in \mathbf{R}^n$ as $v_i \sim \mathcal{N}(0, 1/n)$ and a noise vector $\varepsilon \in \mathbf{R}^m$ with elements

$$\varepsilon_i \sim \begin{cases} \mathcal{N}(0, 1/4) & \text{with probability } p = 0.95 \\ \mathcal{U}[0, 10] & \text{otherwise.} \end{cases}$$

We then set $b = Av + \varepsilon$. For each instance we choose $m = 100n$ and $M = 1$.

A.7 Support vector machine

Support vector machine problem seeks an affine function that approximately classifies the two sets of points [21]. The problem can be stated as

$$\text{minimize} \quad x^T x + \lambda \sum_{i=1}^{m} \max(0, b_i a_i^T x + 1),$$

where $b_i \in \{-1, +1\}$ is a set label, and a_i is a vector of features for the ith point. The problem can be equivalently represented as the following QP

$$\begin{array}{ll} \text{minimize} & x^T x + \lambda \mathbf{1}^T t \\ \text{subject to} & t \geq \mathbf{diag}(b) A x + \mathbf{1} \\ & t \geq 0, \end{array}$$

where $\mathbf{diag}(b)$ denotes the diagonal matrix with elements of b on its diagonal.

Problem instances We choose the vector b so that

$$b_i = \begin{cases} +1 & i \leq m/2 \\ -1 & \text{otherwise}, \end{cases}$$

and the elements of A as

$$A_{ij} \sim \begin{cases} \mathcal{N}(+1/n, 1/n) & i \leq m/2 \\ \mathcal{N}(-1/n, 1/n) & \text{otherwise}, \end{cases}$$

with 15% nonzeros per case.

References

1. Agrawal, A., Verschueren, R., Diamond, S., Boyd, S.: A rewriting system for convex optimization problems. J. Control Decis. **5**(1), 42–60 (2018). https://doi.org/10.1080/23307706.2017.1397554
2. Allgöwer, F., Badgwell, T.A., Qin, J.S., Rawlings, J.B., Wright, S.J.: Nonlinear Predictive Control and Moving Horizon Estimation–An Introductory Overview, pp. 391–449. Springer, London (1999)
3. Amestoy, P.R., Davis, T.A., Duff, I.S.: Algorithm 837: AMD, an approximate minimum degree ordering algorithm. ACM Trans. Math. Softw. **30**(3), 381–388 (2004)
4. Balakrishnan, H., Hwang, I., Tomlin, C.J.: Polynomial approximation algorithms for belief matrix maintenance in identity management. In: IEEE Conference on Decision and Control (CDC), pp. 4874–4879 (2004)
5. Banjac, G., Goulart, P.: Tight global linear convergence rate bounds for operator splitting methods. IEEE Trans. Autom. Control **63**(12), 4126–4139 (2018). https://doi.org/10.1109/TAC.2018.2808442
6. Banjac, G., Goulart, P., Stellato, B., Boyd, S.: Infeasibility detection in the alternating direction method of multipliers for convex optimization. J. Optim. Theory Appl. **183**(2), 490–519 (2019). https://doi.org/10.1007/s10957-019-01575-y
7. Banjac, G., Stellato, B., Moehle, N., Goulart, P., Bemporad, A., Boyd, S.: Embedded code generation using the OSQP solver. In: IEEE Conference on Decision and Control (CDC) (2017)

8. Bauschke, H.H., Borwein, J.M.: On projection algorithms for solving convex feasibility problems. SIAM Rev. **38**(3), 367–426 (1996)
9. Bauschke, H.H., Combettes, P.L.: Convex Analysis and Monotone Operator Theory in Hilbert Spaces, 1st edn. Springer, Berlin (2011)
10. Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A.: Mixed-integer nonlinear optimization. Acta Numer. **22**, 1–131 (2013)
11. Benzi, M.: Preconditioning techniques for large linear systems: a survey. J. Comput. Phys. **182**(2), 418–477 (2002)
12. Borrelli, F., Bemporad, A., Morari, M.: Predictive Control for Linear and Hybrid Systems. Cambridge University Press, Cambridge (2017)
13. Boyd, S., Busseti, E., Diamond, S., Kahn, R.N., Koh, K., Nystrup, P., Speth, J.: Multi-period trading via convex optimization. Found. Trends Optim. **3**(1), 1–76 (2017). https://doi.org/10.1561/2400000023
14. Boyd, S., El Ghaoui, L., Feron, E., Balakrishnan, V.: Linear Matrix Inequalities in System and Control Theory. Society for Industrial and Applied Mathematics, Philadelphia (1994)
15. Boyd, S., Mueller, M.T., O'Donoghue, B., Wang, Y.: Performance bounds and suboptimal policies for multi-period investment. Found. Trends Optim. **1**(1), 1–72 (2014)
16. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011)
17. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
18. Bradley, A.: Algorithms for the equilibration of matrices and their application to limited-memory quasi-Newton methods. Ph.D. thesis, Stanford University (2010)
19. Candés, E.J., Wakin, M.B., Boyd, S.: Enhancing sparsity by reweighted ℓ_1 minimization. J. Fourier Anal. Appl. **14**(5), 877–905 (2008)
20. Cornuejols, G., Tütüncü, R.: Optimization Methods in Finance. Finance and Risk. Cambridge University Press, Cambridge (2006)
21. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
22. Dantzig, G.B.: Linear Programming and Extensions. Princeton University Press, Princeton (1963)
23. Davis, T.A.: Algorithm 849: a concise sparse Cholesky factorization package. ACM Trans. Math. Softw. **31**(4), 587–591 (2005)
24. Davis, T.A.: Direct Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia (2006)
25. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (2011). https://doi.org/10.1145/2049662.2049663
26. Diamond, S., Boyd, S.: CVXPY: a python-embedded modeling language for convex optimization. J. Mach. Learn. Res. **17**(83), 1–5 (2016)
27. Diamond, S., Boyd, S.: Stochastic matrix-free equilibration. J. Optim. Theory Appl. **172**(2), 436–454 (2017)
28. Diehl, M., Ferreau, H.J., Haverbeke, N.: Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation, pp. 391–417. Springer, Berlin (2009)
29. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
30. Domahidi, A., Chu, E., Boyd, S.: ECOS: an SOCP solver for embedded systems. In: European Control Conference (ECC), pp. 3071–3076 (2013)
31. Douglas, J., Rachford, H.H.: On the numerical solution of heat conduction problems in two and three space variables. Trans. Am. Math. Soc. **82**(2), 421–439 (1956)
32. Duff, I.S., Erisman, A.M., Reid, J.K.: Direct Methods for Sparse Matrices. Oxford University Press, London (1989)
33. Dunning, I., Huchette, J., Lubin, M.: JuMP: a modeling language for mathematical optimization. SIAM Rev. **59**(2), 295–320 (2017)
34. Eckstein, J.: Parallel alternating direction multiplier decomposition of convex programs. J. Optim. Theory Appl. **80**(1), 39–62 (1994)
35. Eckstein, J., Ferris, M.C.: Operator-splitting methods for monotone affine variational inequalities, with a parallel application to optimal control. INFORMS J. Comput. **10**(2), 218–235 (1998)
36. Ferreau, H.J., Kirches, C., Potschka, A., Bock, H.G., Diehl, M.: qpOASES: a parametric active-set algorithm for quadratic programming. Math. Program. Comput. **6**(4), 327–363 (2014)
37. Fletcher, R., Leyffer, S.: Numerical experience with lower bounds for MIQP branch-and-bound. SIAM J. Optim. **8**(2), 604–616 (1998)

38. Fougner, C., Boyd, S.: Parameter Selection and Preconditioning for a Graph Form Solver, pp. 41–61. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-67068-3_4
39. Frank, M., Wolfe, P.: An algorithm for quadratic programming. Naval Res. Log. Q. **3**(1–2), 95–110 (1956)
40. Gabay, D.: Chapter IX applications of the method of multipliers to variational inequalities. Stud. Math. Appl. **15**, 299–331 (1983)
41. Gabay, D., Mercier, B.: A dual algorithm for the solution of nonlinear variational problems via finite element approximation. Comput. Math. Appl. **2**(1), 17–40 (1976)
42. García, C.E., Prett, D.M., Morari, M.: Model predictive control: theory and practice—a survey. Automatica **25**(3), 335–348 (1989)
43. Gertz, E.M., Wright, S.J.: Object-oriented software for quadratic programming. ACM Trans. Math. Softw. **29**(1), 58–81 (2003)
44. Ghadimi, E., Teixeira, A., Shames, I., Johansson, M.: Optimal parameter selection for the alternating direction method of multipliers (ADMM): quadratic problems. IEEE Trans. Autom. Control **60**(3), 644–658 (2015)
45. Gill, P.E., Murray, W., Saunders, M.A., Tomlin, J.A., Wright, M.H.: On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. Math. Program. **36**(2), 183–209 (1986)
46. Giselsson, P., Boyd, S.: Metric selection in fast dual forward–backward splitting. Automatica **62**, 1–10 (2015)
47. Giselsson, P., Boyd, S.: Linear convergence and metric selection for Douglas–Rachford splitting and ADMM. IEEE Trans. Autom. Control **62**(2), 532–544 (2017)
48. Glowinski, R., Marroco, A.: Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique **9**(R2), 41–76 (1975)
49. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore (1996)
50. Goulart, P., Stellato, B., Banjac, G.: QDLDL (2018). https://github.com/oxfordcontrol/qdldl. Accessed 6 Feb 2020
51. Gould, N., Scott, J.: A note on performance profiles for benchmarking software. ACM Trans. Math. Softw. **43**(2), 15:1–15:5 (2016). https://doi.org/10.1145/2950048
52. Greenbaum, A.: Iterative Methods for Solving Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia (1997)
53. Gurobi Optimization Inc.: Gurobi optimizer reference manual (2016). http://www.gurobi.com. Accessed 6 Feb 2020
54. He, B.S., Yang, H., Wang, S.L.: Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. J. Optim. Theory Appl. **106**(2), 337–356 (2000)
55. Huber, P.J.: Robust estimation of a location parameter. Ann. Math. Stat. **35**(1), 73–101 (1964)
56. Huber, P.J.: Robust Statistics. Wiley, New York (1981)
57. Intel Corporation: Intel Math Kernel Library. User's Guide (2017)
58. Jerez, J.L., Goulart, P.J., Richter, S., Constantinides, G.A., Kerrigan, E.C., Morari, M.: Embedded online optimization for model predictive control at megahertz rates. IEEE Trans. Autom. Control **59**(12), 3238–3251 (2014)
59. Kantorovich, L.: Mathematical methods of organizing and planning production. Manag. Sci. **6**(4), 366–422 (1960). English translation
60. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**(4), 373–395 (1984)
61. Kelley, C.: Iterative Methods for Linear and Nonlinear Equations. Society for Industrial and Applied Mathematics, Philadelphia (1995)
62. Klee, V., Minty, G.: How good is the simplex algorithm. Department of Mathematics, University of Washington, Technical report (1970)
63. Knight, P.A., Ruiz, D., Uçar, B.: A symmetry preserving algorithm for matrix scaling. SIAM J. Matrix Anal. Appl. **35**(3), 931–955 (2014)
64. Lions, P.L., Mercier, B.: Splitting algorithms for the sum of two nonlinear operators. SIAM J. Numer. Anal. **16**(6), 964–979 (1979)

65. Löfberg, J.: YALMIP: a toolbox for modeling and optimization in MATLAB. In: IEEE International Conference on Robotics and Automation, pp. 284–289 (2004). https://doi.org/10.1109/CACSD.2004.1393890
66. Mangasarian, O.L., Musicant, D.R.: Robust linear and support vector regression. IEEE Trans. Pattern Anal. Mach. Intell. **22**(9), 950–955 (2000). https://doi.org/10.1109/34.877518
67. Markowitz, H.: Portfolio selection. J. Finance **7**(1), 77–91 (1952)
68. Maros, I., Mészáros, C.: A repository of convex quadratic programming problems. Optim. Methods Softw. **11**(1–4), 671–681 (1999)
69. Mattingley, J., Boyd, S.: Real-time convex optimization in signal processing. IEEE Signal Process. Mag. **27**(3), 50–61 (2010)
70. Mattingley, J., Boyd, S.: CVXGEN: a code generator for embedded convex optimization. Optim. Eng. **13**(1), 1–27 (2012)
71. Mehrotra, S.: On the implementation of a primal–dual interior point method. SIAM J. Optim. **2**(4), 575–601 (1992)
72. Mittelmann, H.: Benchmarks for optimization software. http://plato.asu.edu/bench.html. Accessed 08 Nov 2019
73. MOSEK ApS: The MOSEK optimization toolbox for MATLAB manual. Version 8.0 (Revision 57) (2017). http://docs.mosek.com/8.0/toolbox/index.html. Accessed 6 Feb 2020
74. Nesterov, Y., Nemirovskii, A.: Interior-Point Polynomial Algorithms in Convex Programming. Society for Industrial and Applied Mathematics, Philadelphia (1994)
75. Nishihara, R., Lessard, L., Recht, B., Packard, A., Jordan, M.I.: A general analysis of the convergence of ADMM. In: International Conference on Machine Learning (ICML), pp. 343–352 (2015)
76. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer Series in Operations Research and Financial Engineering. Springer, Berlin (2006)
77. O'Donoghue, B., Chu, E., Parikh, N., Boyd, S.: Conic optimization via operator splitting and homogeneous self-dual embedding. J. Optim. Theory Appl. **169**(3), 1042–1068 (2016)
78. O'Donoghue, B., Stathopoulos, G., Boyd, S.: A splitting method for optimal control. IEEE Trans. Control Syst. Technol. **21**(6), 2432–2442 (2013)
79. Pock, T., Chambolle, A.: Diagonal preconditioning for first order primal–dual algorithms in convex optimization. In: 2011 International Conference on Computer Vision, pp. 1762–1769 (2011)
80. Raghunathan, A.U., Di Cairano, S.: ADMM for convex quadratic programs: Q-linear convergence and infeasibility detection (2014). arXiv:1411.7288
81. Raghunathan, A.U., Di Cairano, S.: Infeasibility detection in alternating direction method of multipliers for convex quadratic programs. In: IEEE Conference on Decision and Control (CDC), pp. 5819–5824 (2014). https://doi.org/10.1109/CDC.2014.7040300
82. Rawlings, J.B., Mayne, D.Q.: Model Predictive Control: Theory and Design. Nob Hill Publishing, San Francisco (2009)
83. Reuther, A., Kepner, J., Byun, C., Samsi, S., Arcand, W., Bestor, D., Bergeron, B., Gadepally, V., Houle, M., Hubbell, M., Jones, M., Klein, A., Milechin, L., Mullen, J., Prout, A., Rosa, A., Yee, C., Michaleas, P.: Interactive supercomputing on 40,000 cores for machine learning and data analysis. In: 2018 IEEE High Performance extreme Computing Conference (HPEC), pp. 1–6 (2018). https://doi.org/10.1109/HPEC.2018.8547629
84. Rockafellar, R.T., Wets, R.J.B.: Variational Analysis. Grundlehren der mathematischen Wissenschaften. Springer, Berlin (1998)
85. Ruiz, D.: A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical Report RAL-TR-2001-034, Rutherford Appleton Laboratory, Oxon, UL (2001)
86. Sinkhorn, R., Knopp, P.: Concerning nonnegative matrices and doubly stochastic matrices. Pac. J. Math. **21**(2), 343–348 (1967)
87. Stathopoulos, G., Shukla, H., Szucs, A., Pu, Y., Jones, C.N.: Operator splitting methods in control. Found. Trends Syst. Control **3**(3), 249–362 (2016)
88. Stellato, B., Banjac, G.: Benchmark examples for the OSQP solver (2019). https://github.com/oxfordcontrol/osqp_benchmarks. Accessed 6 Feb 2020
89. Takapoui, R., Javadi, H.: Preconditioning via diagonal scaling (2014). https://web.stanford.edu/~hrhakim/projects/EE364B.pdf. Accessed 6 Feb 2020
90. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B **58**(1), 267–288 (1996)
91. Vanderbei, R.: Symmetric quasi-definite matrices. SIAM J. Optim. **5**(1), 100–113 (1995)

92. Wilkinson, J.H.: Rounding Errors in Algebraic Processes. Prentice Hall, Englewood Cliffs (1963)
93. Wohlberg, B.: ADMM penalty parameter selection by residual balancing (2017). arXiv:1704.06209v1
94. Wolfe, P.: The simplex method for quadratic programming. Econometrica **27**(3), 382–398 (1959)
95. Wright, S.: Primal–Dual Interior-Point Methods. Society for Industrial and Applied Mathematics, Philadelphia (1997)

Affiliations

Bartolomeo Stellato[1] · Goran Banjac[2] · Paul Goulart[3] · Alberto Bemporad[4] · Stephen Boyd[5]

Goran Banjac
gbanjac@control.ee.ethz.ch

Paul Goulart
paul.goulart@eng.ox.ac.uk

Alberto Bemporad
alberto.bemporad@imtlucca.it

Stephen Boyd
boyd@stanford.edu

1 Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA

2 ETH Zürich, Rämistrasse 101, 8092 Zurich, Switzerland

3 University of Oxford, Parks Road, Oxford OX1 3PJ, UK

4 IMT Institute for Advanced Studies Lucca, Piazza S. Francesco 19, 55100 Lucca, Italy

5 Stanford University, 450 Serra Mall, Stanford, CA 94305, USA