

OSQP: An Operator Splitting Solver for Quadratic Programs

Bartolomeo Stellato, Goran Banjac, Paul Goulart,
Alberto Bemporad, and Stephen Boyd

November 23, 2017

Abstract

We present a general purpose solver for 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 in each 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 is 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.

1 Introduction

1.1 The problem

Consider the following optimization problem

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

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

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

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

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

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) [CV95], lasso [Tib96, CWB08] and Huber fitting [Hub64, Hub81]. Financial applications of (1) include portfolio optimization [CT06, Mar52, BMOW14, BBD⁺17] [BV04, §4.4.1]. In the field of control engineering, model predictive control (MPC) [RM09, GPM89] and moving horizon estimation (MHE) [ABQ⁺99] techniques require the solution of a QP at each time instant. Several signal processing problems also fall into the same class [BV04, §6.3.3][MB10]. In addition, the numerical solution of QP subproblems is an essential component in nonconvex optimization methods such as sequential quadratic programming (SQP) [NW06, Chap. 18] and mixed-integer optimization using branch-and-bound algorithms [BKL⁺13, FL98].

1.2 Solution methods

Convex QPs have been studied since the 1950s [FW56], following from the seminal work on LPs started by Kantorovich [Kan60]. 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 [Wol59], and were obtained from an extension of Dantzig's simplex method for solving LPs [Dan63]. 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 [NW06, §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 [KM70]. Modern implementations of active set methods for the solution of QPs can be found in many commercial solvers, such as MOSEK [MOS17] and GUROBI [Gur16], and in the open-source solver qpOASES [FKP⁺14].

Interior point methods. Interior point algorithms gained popularity in the 1980s as a method for solving LPs in polynomial time [Kar84, GMS⁺86]. In the 90s these techniques were extended to general convex optimization problems, including QPs [NN94]. 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 [BV04, Chap. 11] and [NW06, §16.6] for details. Primal-dual interior point methods, in particular the Mehrotra predictor-corrector [Meh92] method, became the algorithms of choice for practical implementation [Wri97] 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 [MOS17], GUROBI [Gur16] and CVXGEN [MB12] and in the open-source solver OOQP [GW03].

First order methods. First order optimization methods for solving quadratic programs date to the 1950s [FW56]. 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 [LM79, DR56] are a particular class of first order methods which model the optimization problem as a sum of nonlinear operators.

In recent years, the operator splitting method known as the alternating direction method of multipliers (ADMM) [GM76, GM75] has received particular attention because of its very good practical convergence behavior; see [BPC⁺11] for a survey. ADMM can be seen as a variant of the classical alternating projections algorithm [BB96] for finding a point in the intersection of two convex sets, and can also be shown to be equivalent to the Douglas-Rachford splitting [Gab83]. 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 [JGR⁺14, OSB13, SSS⁺16]. ADMM is also compatible with distributed optimization architectures enabling the solution of very large scale problems [BPC⁺11].

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 [OCPB16]. 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 [GB17, BG16], it remains unclear how to select those parameters to optimize the algorithm convergence. 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 matrix coefficients remain the same at every iteration, we perform only a single initial factorization at the beginning of the algorithm. Once the initial factorization is computed, the algorithm is *division-free*. 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 quality 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 parametric QP where only a few elements of the problem data change. Example illustrating the effectiveness of the proposed algorithm in parametric programs arising in embedded applications appear in [BSM⁺17].

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 [MM99]. Numerical results show that our algorithm is able to provide up to one order of magnitude computational time improvements over existing commercial and open-source solvers on 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

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

We can write the optimality conditions of problem (3) as [RW98, Thm. 6.12][BV04]

$$Ax = z, \tag{4}$$

$$Px + q + A^T y = 0, \tag{5}$$

$$z \in \mathcal{C}, \quad y \in N_{\mathcal{C}}(z), \tag{6}$$

where $y \in \mathbf{R}^m$ is a 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 of 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

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

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

2.1 Certificates of primal and dual infeasibility

From the *theorem of strong alternatives*, exactly one of the following sets is nonempty

$$\mathcal{P} = \{x \in \mathbf{R}^n \mid Ax \in \mathcal{C}\}, \quad (10)$$

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

where $S_{\mathcal{C}}$ is the support function of \mathcal{C} , provided that some type of constraint qualification holds [BV04]. 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

$$A^T y = 0, \quad u^T y_+ + l^T y_- < 0. \quad (12)$$

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

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

is a certificate of dual infeasibility for problem (2).

3 Solution with ADMM

Our method solves the problem (3) using ADMM [BPC⁺11]. In the proposed novel splitting the subproblems in each algorithm step are always solvable independently from the problem data. By introducing auxiliary variables $\tilde{x} = x$ and $\tilde{z} = z$, we can rewrite problem (3) as

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

where $\mathcal{I}_{Ax=z}$ and $\mathcal{I}_{\mathcal{C}}$ are the indicator functions of the sets $\{(x, z) \in \mathbf{R}^n \times \mathbf{R}^m \mid Ax = z\}$ and \mathcal{C} , respectively.

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

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

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

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

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

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

where $\sigma > 0$ and $\rho > 0$ are the *step-size parameters* and $\alpha \in (0, 2)$ is the *relaxation parameter*. Note that all the derivations hold also for σ and ρ being positive definite diagonal matrices. Π denotes the Euclidean projection onto set \mathcal{C} . 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 , 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 quadratic optimization problem

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

The optimality conditions for this equality constrained QP are

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

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

$$A\tilde{x}^{k+1} = \tilde{z}^{k+1}, \quad (23)$$

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

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

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 has always full rank thanks to the parameters σ and ρ introduced in our splitting. Therefore, (24) has always a unique solution independently from the problem data. We solve the linear system in (24) using either a *direct method* or an *indirect method*.

Direct method. The direct method computes the exact solution of the linear system (24) by first computing a factorization of 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 solve cost, so that each iteration is computed quickly.

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 [Van95]. 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 [Dav06] without regard for the actual non-zero 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* step and requires only the sparsity pattern of the KKT matrix. In the second step, referred to as the *numerical factorization* step, 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 remain the same.

Indirect method. We can also find the solution of (24) 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 [GVL96, NW06]. 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.

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 done using direct or indirect methods as discussed in Section 3.1.

3.3 Convergence and infeasibility detection

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

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: $z^{k+1} \leftarrow \Pi(\alpha \tilde{z}^{k+1} + (1 - \alpha)z^k + \rho^{-1}y^k)$
 - 7: $y^{k+1} \leftarrow y^k + \rho(\alpha \tilde{z}^{k+1} + (1 - \alpha)z^k - z^{k+1})$
 - 8: **until** termination condition is satisfied
-

It is a well known fact that ADMM for solving problems of the form (14) is equivalent to Douglas-Rachford splitting applied to a specific problem reformulation [Eck89, GFB16].

In particular, as shown in [GFB16], Algorithm 1 is equivalent to

$$(\tilde{x}^k, \tilde{z}^k) \leftarrow \underset{(\tilde{x}, \tilde{z}): A\tilde{x}=\tilde{z}}{\operatorname{argmin}} (1/2)\tilde{x}^T P \tilde{x} + q^T \tilde{x} + (\sigma/2)\|\tilde{x} - x^k\|_2^2 + (\rho/2)\|\tilde{z} - (2\Pi(v^k) - v^k)\|_2^2 \quad (25)$$

$$x^{k+1} \leftarrow x^k + \alpha(\tilde{x}^k - x^k) \quad (26)$$

$$v^{k+1} \leftarrow v^k + \alpha(\tilde{z}^k - \Pi(v^k)) \quad (27)$$

where

$$z^k = \Pi(v^k), \quad y^k = \rho(v^k - \Pi(v^k)). \quad (28)$$

Observe that iterates z^k, y^k satisfy the optimality condition (6) by construction [BC11, Prop. 6.46].

We now show that the primal and dual residuals defined in (7) and (8) converge to zero if the original problem is solvable. The solution of the optimization problem in (25) satisfies the following optimality conditions that are equivalent to (21)–(23),

$$(P + \sigma I)\tilde{x}^k + q - \sigma x^k + \rho A^T(\tilde{z}^k - 2z^k + v^k) = 0 \quad (29)$$

$$A\tilde{x}^k = \tilde{z}^k. \quad (30)$$

Iterations (25)–(27) can be represented as $(x^{k+1}, v^{k+1}) = T(x^k, v^k)$, where T is the Douglas-Rachford operator [BGSB17]. From a general convergence theory of Douglas-Rachford splitting (see [BC11, Cor. 27.4]) it follows that if problem (1) is solvable, then the sequences $\{x^k\}$ and $\{v^k\}$ converge as $k \rightarrow \infty$. Due to (26) and (27) we then have

$$\tilde{x}^k - x^k \rightarrow 0, \quad \tilde{z}^k - z^k \rightarrow 0. \quad (31)$$

It follows from the above conditions and equality (30) that

$$Ax^k - z^k = \underbrace{A\tilde{x}^k - \tilde{z}^k}_{=0} + A \underbrace{(x^k - \tilde{x}^k)}_{\rightarrow 0} - \underbrace{(z^k - \tilde{z}^k)}_{\rightarrow 0},$$

which means that the primal residual converges to zero, *i.e.*,

$$r_{\text{prim}}^k = Ax^k - z^k \rightarrow 0. \quad (32)$$

Similarly, we have

$$\begin{aligned} Px^k + q + A^T y^k &= (P + \sigma I)\tilde{x}^k + q - \sigma x^k + \rho A^T (\tilde{z}^k - 2z^k + v^k) + \\ &\quad (P + \sigma I)(x^k - \tilde{x}^k) + \rho A^T (z^k - \tilde{z}^k), \end{aligned}$$

which taken together with (29) and (31) implies that the dual residual converges to zero, *i.e.*,

$$r_{\text{dual}}^k = Px^k + q + A^T y^k \rightarrow 0. \quad (33)$$

Quadratic programs infeasibility. If problem (1) is primal and/or dual infeasible, then the sequence of iterates (x^k, z^k, y^k) generated by Algorithm 1 will not necessarily converge. However, in the case $\mathcal{C} = [l, u]$, the sequence

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

will always converge [BGSB17], where the δ notation is used to indicate the difference between successive iterates. If the problem is primal infeasible, then $\delta y := \lim_{k \rightarrow \infty} \delta y^k$ will satisfy conditions (12), whereas $\delta x := \lim_{k \rightarrow \infty} \delta x^k$ will satisfy conditions (13) if it is dual infeasible.

3.4 Termination criteria

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

For solvable problems, a reasonable termination criterion for detecting optimality is that the norms of the residuals r_{prim}^k and r_{dual}^k are smaller than some tolerance levels $\epsilon_{\text{prim}} > 0$ and $\epsilon_{\text{dual}} > 0$ [BPC⁺11], *i.e.*,

$$\|r_{\text{prim}}^k\|_{\infty} \leq \epsilon_{\text{prim}}, \quad \|r_{\text{dual}}^k\|_{\infty} \leq \epsilon_{\text{dual}}. \quad (34)$$

We set the tolerance levels as

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

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

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

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

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

for $i = 1, \dots, m$ where $\epsilon_{\text{dinf}} > 0$ 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

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

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

$$\begin{bmatrix} P & A_{\mathcal{L}}^T & A_{\mathcal{U}}^T \\ A_{\mathcal{L}} & & \\ A_{\mathcal{U}} & & \end{bmatrix} \begin{bmatrix} x \\ y_{\mathcal{L}} \\ y_{\mathcal{U}} \end{bmatrix} = \begin{bmatrix} -q \\ l_{\mathcal{L}} \\ u_{\mathcal{U}} \end{bmatrix}, \quad (35)$$

$$y_i = 0, \quad i \notin (\mathcal{L} \cup \mathcal{U}), \quad (36)$$

$$z = Ax. \quad (37)$$

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 (35) is usually much smaller than the KKT system in Section 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 (35) 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

$$\begin{bmatrix} P + \delta I & A_{\mathcal{L}}^T & A_{\mathcal{U}}^T \\ A_{\mathcal{L}} & -\delta I & \\ A_{\mathcal{U}} & & -\delta I \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y}_{\mathcal{L}} \\ \hat{y}_{\mathcal{U}} \end{bmatrix} = \begin{bmatrix} -q \\ l_{\mathcal{L}} \\ u_{\mathcal{U}} \end{bmatrix}, \quad (38)$$

where $\delta > 0$ is a regularization parameter with value $\delta \approx 10^{-6}$. Since the regularized matrix in (38) is quasi-definite, the linear system (38) 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 (35) and (38), 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 [DER89], *i.e.*, we iteratively solve

$$(K + \Delta K)\Delta\hat{t}^k = g - K\hat{t}^k \quad (39)$$

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 (38), iterative refinement requires only a backward- and a forward-solve, and does not require another matrix factorization.

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 [NW06, Chap. 5], [GTSJ15, Ben02, PC11, GB15, GB17]. The optimal choice

of preconditioners has been studied for at least two decades and remains an active area of research [Kel95, Chap. 2],[Gre97, 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 [BEGFB94]. 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* [Bra10, TJ14, FB17, DB17]. 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

$$M := \begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix}. \quad (40)$$

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

$$S = \begin{bmatrix} D & \\ & E \end{bmatrix}, \quad (41)$$

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

$$\begin{aligned} & \text{minimize} && (1/2)\bar{x}^T \bar{P} \bar{x} + \bar{q}^T \bar{x} \\ & \text{subject to} && \bar{A} \bar{x} \in EC, \end{aligned} \quad (42)$$

where $\bar{x} = D^{-1}x$, $\bar{P} = cDPD$, $\bar{q} = cDq$, $\bar{A} = EAD$ and $EC := \{Ez \in \mathbf{R}^m \mid z \in \mathcal{C}\}$. The dual variables of the new problem are $\bar{y} = cE^{-1}y$. Note that when $\mathcal{C} = [l, u]$ the Euclidean projection onto $EC = [El, Eu]$ 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 [BHT04]. 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 [Bra10] for more details on matrix equilibration.

Ruiz equilibration. In this work we apply a variation of the Ruiz equilibration [Rui01]. This technique was originally proposed to equilibrate square matrices showing fast linear convergence superior to other methods such as the Sinkhorn-Knopp equilibration [SK67]. Ruiz equilibration converges in few tens of iterations even in cases when Sinkhorn-Knopp

Algorithm 2 Modified Ruiz equilibration

```

initialize  $c = 1, S = I, \delta = 0, \bar{P} = P, \bar{q} = q, \bar{A} = A, \bar{l} = l, \bar{u} = u$ 
while  $\|1 - \delta\|_\infty > \epsilon_{\text{equil}}$  do
  for  $i = 1, \dots, n + m$  do
     $\delta_i \leftarrow 1/\sqrt{\|M_i\|_\infty}$   $\triangleright M$  equilibration
   $\bar{P}, \bar{q}, \bar{A}, \bar{l}, \bar{u} \leftarrow \text{Scale } \bar{P}, \bar{q}, \bar{A}, \bar{l}, \bar{u} \text{ using } \mathbf{diag}(\delta)$ 
   $\gamma \leftarrow 1/\max\{\text{mean}(\|\bar{P}_i\|_\infty), \|\bar{q}\|_\infty\}$   $\triangleright$  Cost scaling
   $\bar{P} \leftarrow \gamma \bar{P}, \bar{q} \leftarrow \gamma \bar{q}$ 
   $S \leftarrow \mathbf{diag}(\delta)S, c \leftarrow \gamma c$ 
return  $S, c$ 

```

equilibration takes thousands of iterations [KRU14]. 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 we 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 (42), we would still like to apply the stopping criteria defined in Section 3.4 to an unscaled version of our problem. The primal and dual residuals in (34) can be rewritten in terms of the scaled problem as

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

and the tolerances levels as

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

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

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

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

$$\|D^{-1}\bar{P}\delta\bar{x}^k\|_\infty \leq c\epsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty, \quad \bar{q}^T\delta\bar{x}^k \leq -c\epsilon_{\text{dinf}}\|D\delta\bar{x}^k\|_\infty,$$

$$(E^{-1}\bar{A}\delta\bar{x}^k)_i \begin{cases} \in [-\epsilon_{\text{dinf}}, \epsilon_{\text{dinf}}] \|D\delta\bar{x}^k\|_\infty & u_i, l_i \in \mathbf{R} \\ \geq \epsilon_{\text{dinf}} \|D\delta\bar{x}^k\|_\infty & u_i = +\infty \\ \leq -\epsilon_{\text{dinf}} \|D\delta\bar{x}^k\|_\infty & l_i = -\infty, \end{cases}$$

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 [GTSJ15, NLR⁺15, GB17]. After extensive numerical testing on millions of problem instances and a wide range of dimensions, we chose the algorithm parameters as follows.

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 [Eck94, EF98]. 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 problem, the optimal ρ is defined as $\rho_i = \infty$ for the active constraints and $\rho_i = 0$ for the inactive constraints [GTSJ15, §IV.D]. Unfortunately, it is impossible to know a priori whether any given constraint is active or inactive constraints at optimality, so we must instead adopt some heuristics. We define ρ as follows

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

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 [HYW00, BPC⁺11, Woh17]. 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

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

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 with the proposed rule, if the initial $\bar{\rho}$ is not already tuned, we obtain the optimal parameter value with very few updates, usually 1 or 2. 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.

6 Parametric programs

In application domains such as control, statistics, finance, and sequential quadratic programming, 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 the case 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 [RM09, ABQ⁺99], lasso [Tib96, CWB08], and portfolio optimization [BMOW14, Mar52].

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 [DFH09] and sequential quadratic programming [NW06].

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^*, y^*) by setting $(x^0, z^0, y^0) \leftarrow (x^*, Ax^*, y^*)$.

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

<http://osqp.readthedocs.io>.

Users can call OSQP from C, C++, Python, Matlab and Julia, and via parsers such as CVXPY [DB16, AVDB18], JuMP [DHL17], and YALMIP [Löf04].

To exploit the data sparsity pattern, OSQP accepts matrices in Compressed-Sparse-Column (CSC) format [Dav06]. We implemented the linear system solution described in Section 3.1 as an object-oriented interface to easily switch between efficient algorithms. At present OSQP ships with the open-source SuiteSparse LDL direct solver [ADD04, Dav05] and also supports dynamic loading of more advanced algorithms such as the MKL Pardiso direct solver [Int17].

The default values for the OSQP termination tolerances described in Section 3.3 are

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

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 Section 5.2, with optional user override.

OSQP reports the total computation time divided by the time required to perform pre-processing 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 [DCB13], the open-source active-set solver qpOASES [FKP⁺14], and the commercial interior point solvers GUROBI [Gur16] and MOSEK [MOS17]. We executed the OSQP solver with default settings and polishing disabled.

Note that the solution returned by the other solvers is with high accuracy while OSQP returns a lower accuracy solution. Hence, runtime benchmarks are not perfectly comparable since OSQP might take more time than interior point methods if a high accuracy is required. On the other hand, we used the direct single-threaded linear system solver SuiteSparse LDL [ADD04, Dav05] and very simple linear algebra where other solvers such as GUROBI and MOSEK use advanced multi-threaded linear system solvers and custom linear algebra.

We say that the primal-dual solution (x^*, y^*) returned by each solver is optimal if the following optimality conditions are satisfied with $\epsilon_{\text{abs}} = \epsilon_{\text{rel}} = 10^{-3}$,

$$\begin{aligned} \|(Ax^* - u)_+ + (Ax^* - l)_-\|_\infty &\leq \epsilon_{\text{prim}}, & \|Px^* + q + A^T y^*\|_\infty &\leq \epsilon_{\text{dual}}, \\ \|\min((y^*)_+, |u - Ax^*|)\|_\infty &\leq \epsilon_{\text{slack}}, & \|\min(-(y^*)_-, |Ax^* - l|)\|_\infty &\leq \epsilon_{\text{slack}}, \end{aligned}$$

where ϵ_{prim} and ϵ_{dual} are defined in Section 3.4 and $\epsilon_{\text{slack}} = \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \|Ax^*\|_\infty$. All the experiments were carried out on a system with 32 2.2 GHz cores and 512 GB of RAM, running Linux. The code for all the numerical examples is available online at [SB17].

Performance profiles. We make use of the performance profiles [DM02] to compare the timings of the various solvers. We define the performance ratio

$$r_{p,s} := \frac{t_{p,s}}{\min_s t_{p,s}},$$

where $t_{p,s}$ is the time it takes for solver s to solve problem instance p . If solver s fails at solving problem p , we set $r_{p,s} = \infty$. The performance profile plots the function $f_s : \mathbf{R} \mapsto [0, 1]$ defined as

$$f_s(\tau) := \frac{1}{n_p} \sum_p \mathcal{I}_{\leq \tau}(r_{p,s}),$$

where $\mathcal{I}_{\leq \tau}(r_{p,s}) = 1$ if $r_{p,s} \leq \tau$ or 0 otherwise. n_p is the number of problems considered. The value $f_s(\tau)$ corresponds to the fraction of problems solved within τ times from the best solver.

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. We described generation for each class in Appendix A. All instances were either obtained from real data or from realistic non-trivial random data. 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 .

Computation times. We show in Figure 1 the computation times across all the problem classes for GUROBI and OSQP. Each problem class is represented using different symbols. OSQP shows to be competitive or even faster than GUROBI for several problem classes.

Failure rates and performance profiles. Figure 2 describes the failure rates for all the solvers across the generated problems. In case of OSQP failures, the returned solutions always satisfy the complementary slackness conditions and are anyway relevant for many practical applications where high accuracy is not needed. Figure 3 compares the performance profiles of all the solvers tested. For these problems, OSQP always outperforms the other solvers. GUROBI is the closest solver in terms of computation time. ECOS and MOSEK perform similarly even though ECOS is also single-threaded. qpOASES has a clear disadvantage compared to the other methods when dealing with large problems because the number of active set combinations becomes extremely large for several instances.

8.2 Polishing

We ran the same benchmark problems with the OSQP solver with solution polishing enabled. Polishing succeeded in 59% of the times providing a high-accuracy solution with a median of $1.18\times$ computation time compared to the OSQP solution without it. When polishing succeeds, the solution is as accurate, or even more accurate, than the one obtained with

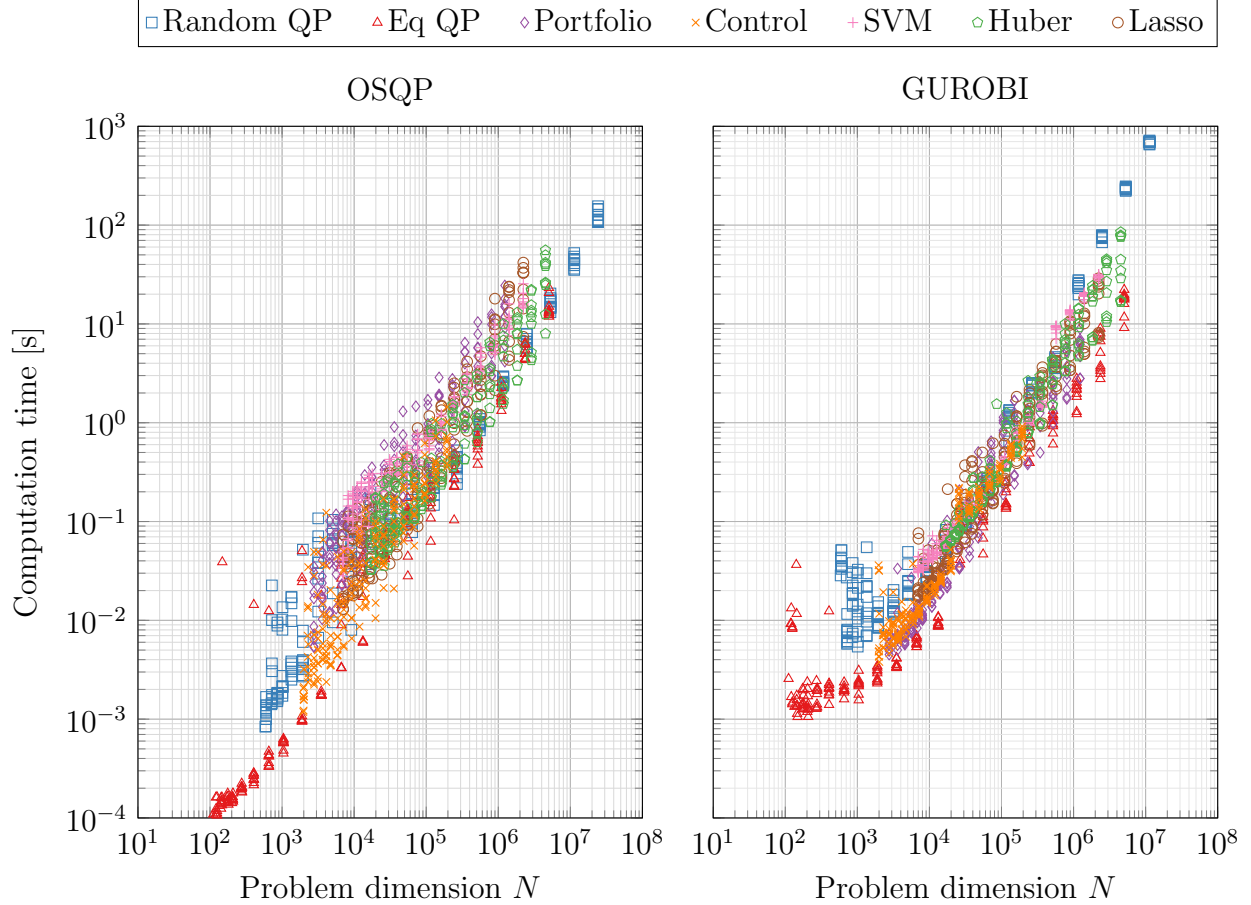


Figure 1: Computation time vs problem dimension for OSQP (left) and GUROBI (right) for the 7 benchmark problem classes.

any other solver. Note that by decreasing the tolerances ϵ_{abs} and ϵ_{rel} we can increase the percentage of times polishing succeeds.

8.3 Warm start and factorization caching

To show the benefits of warm starting and factorization caching, we solved a sequence of QPs with the data varying according to some parameters.

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 $n = 50$ features, $m = 5000$ data points, and λ logarithmically spaced taking 100 values between $\lambda_{\text{max}} = \|A^T b\|_{\infty}$ and $0.01\lambda_{\text{max}}$.

Since the parameters only enter linearly in the cost, we can reuse the matrix factorization

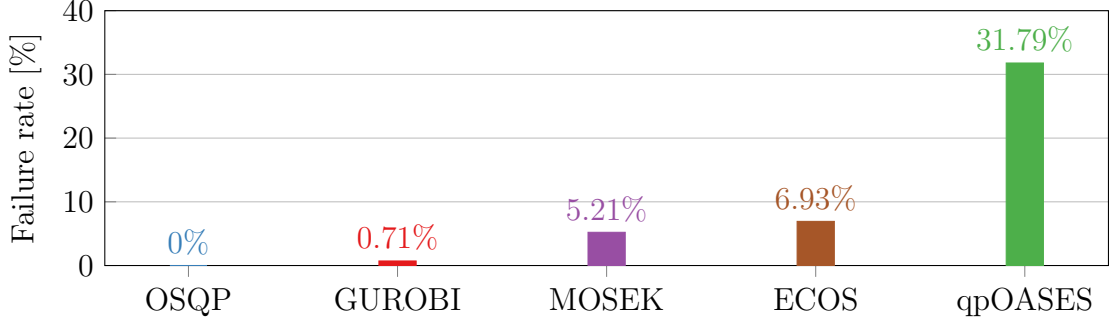


Figure 2: Failure rates for the 7 benchmark problem classes.

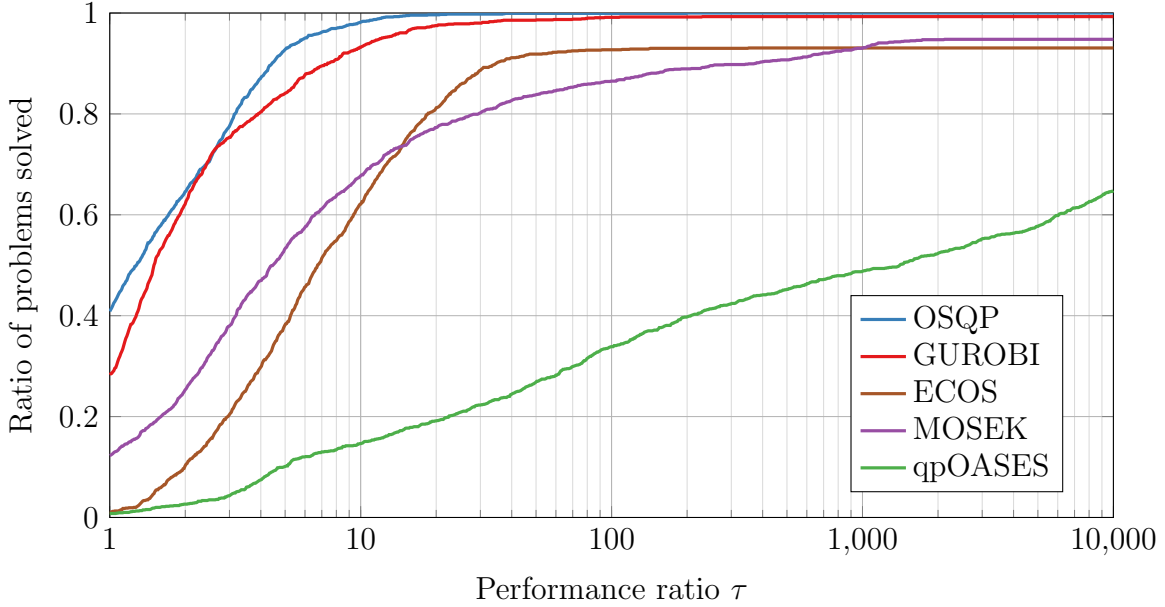


Figure 3: Performance profiles for 7 benchmark problem classes.

and enable warm starting to reduce the computation time as discussed in Section 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 repeated with an updated initial state x_{init} .

We solved the control problem with $n_x = 10$ states, $n_u = 5$ 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.*, $x_{\text{init}} \sim \mathcal{U}(-0.5\bar{x}, 0.5\bar{x})$.

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 Section 6.

Portfolio back test. Consider the portfolio optimization problem in Appendix A.4 with $n = 3000$ assets and $k = 100$ factors.

We run a 4 years back test to compute the optimal assets investment depending on varying expected returns and factor models [BBD⁺17]. 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 Section 6, we exploited the following computations during the QP updates to reduce the computation times. Since μ only enter linearly in the constraints bounds, we can reuse the matrix factorization and enable warm starting. Since the sparsity pattern of D and F does not change during the monthly updates, we can reuse the symbolic factorization and exploit warm starting to speedup the computations.

Results. We show the results in Figure 4. For the lasso example, warm starting and factorization caching bring an average improvement in computation time of $8.5\times$ going from 255.9ms to 29.7ms. In the MPC example, warm starting brings an average $2.8\times$ improvement from 22.8ms to 8.0ms. In the portfolio example, we obtain an average improvement of $6.2\times$ from 200.6ms to 32.2ms. Depending on the problem type and size, warm start and factorization caching can greatly improve the performance allowing the solution in few tens of milliseconds.

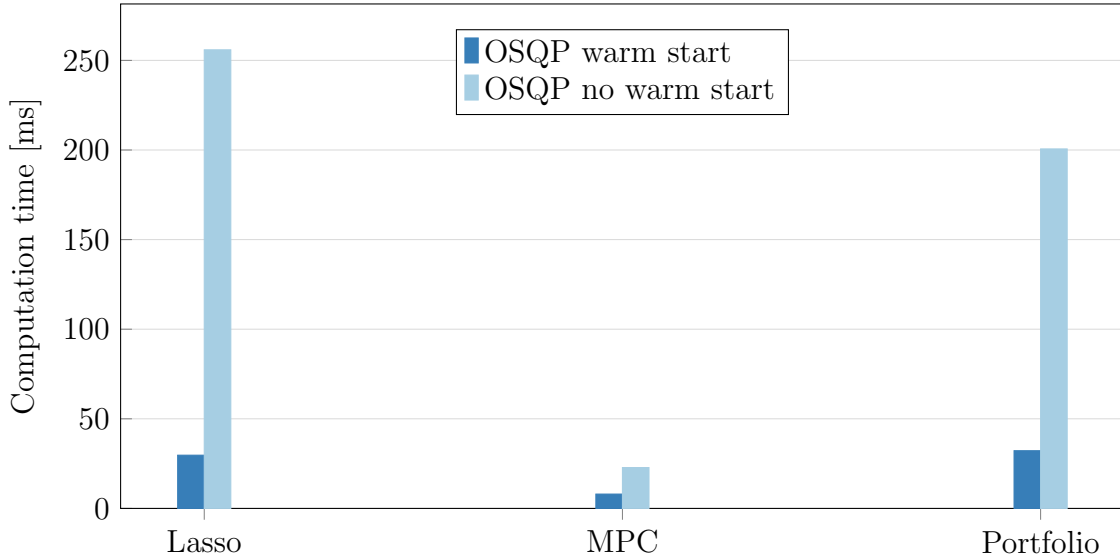


Figure 4: OSQP warm start and factorization caching benchmarks.

8.4 Maros-Mészáros problems

We considered the 138 problems from the Maros-Mészáros test set [MM99] 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.

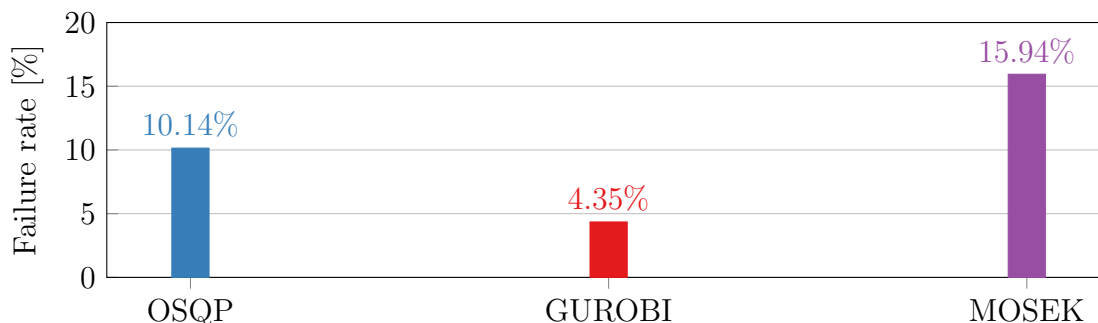


Figure 5: Failure rates for the Maros-Mészáros test set.

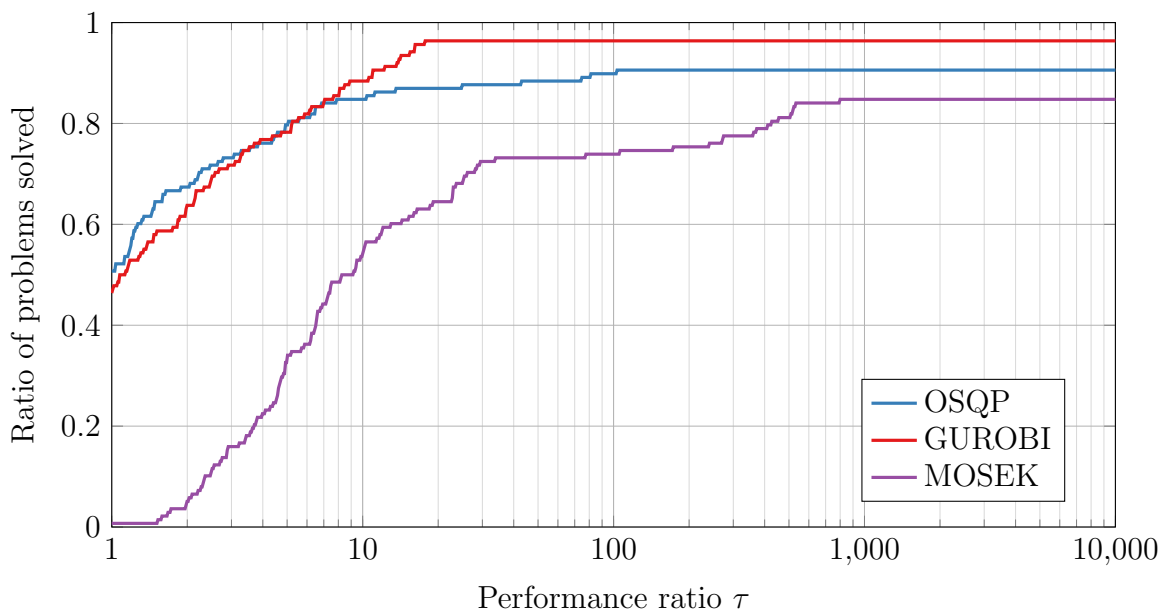


Figure 6: Performance profiles for Maros-Mészáros problems.

Failure rates and performance profiles. The failure rates are shown in Figure 5 and the performance profiles in Figure 6. OSQP shows competitive or even better performance than

GUROBI up to around 85% of the cases. Its failure rate is 10.14% of the examples. GUROBI exhibits the best profiles overall even though it still fails in 4.35 % of the cases. MOSEK is the slowest with 15.94 % failure rate.

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.

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 [MM99]. Timing and failure rate results showed great improvements over state-of-the-art academic and commercial QP solvers.

A Problem classes

A.1 Random QP

Consider the following QP

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

Problem instances. The number of variables and constraints in our problem instances are n and $m = 10n$. We generated random matrix $P = MM^T$ where $M \in \mathbf{R}^{n \times n}$ and 50% nonzero elements $M_{ij} \sim \mathcal{N}(0, 1)$. We set the elements of $A \in \mathbf{R}^{m \times n}$ as $A_{ij} \sim \mathcal{N}(0, 1)$ with only 50% 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

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

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 $P = MM^T$ where $M \in \mathbf{R}^{n \times n}$ and 50% nonzero elements $M_{ij} \sim \mathcal{N}(0, 1)$. We set the elements of $A \in \mathbf{R}^{m \times n}$ as $A_{ij} \sim \mathcal{N}(0, 1)$ with only 50% being nonzero. The vectors are all normally distributed, *i.e.*, $q_i, l_i, u_i \sim \mathcal{N}(0, 1)$.

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

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \nu \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}. \quad (43)$$

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 (39) with $g = (-q, b)$ and $\hat{t}^k = (x^k, \nu^k)$. This means that Algorithm 1 applied to solve equality constrained QP is equivalent to applying iterative refinement [DER89] to solve the KKT system (43).

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 [BBM17]

$$\begin{aligned}
& \text{minimize} && x_N^T Q_T x_T + \sum_{t=0}^{T-1} x_t^T Q x_t + u_t^T R u_t \\
& \text{subject to} && x_{t+1} = A x_t + B u_t \\
& && x_t \in \mathcal{X}, u_t \in \mathcal{U} \\
& && x_0 = x_{\text{init}}.
\end{aligned} \tag{44}$$

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 $z = (x_0, \dots, x_T, u_0, \dots, u_{T-1})$, problem (44) 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 $n_u = 0.5n_x$ inputs. We set the horizon length to $T = 10$. We generated the dynamics as $A = I + \Delta$ with $\Delta_{ij} \sim \mathcal{N}(0, 0.01)$ with 50% nonzero elements. 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 $B_{ij} = \mathcal{N}(0, 1)$.

The state cost is defined as $Q = \text{diag}(q)$ where $q_i \sim \mathcal{U}(0, 10)$ and 70% nonzero elements in q . We chose the input cost as $R = .1I$. 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) [BBM17]. We generated input and state constraints as

$$\mathcal{X} = \{x_t \in \mathbf{R}^{n_x} \mid \|x_t\|_\infty \leq \bar{x}\}, \quad \mathcal{U} = \{u_t \in \mathbf{R}^{n_u} \mid \|u_t\|_\infty \leq \bar{u}\},$$

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

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 [BMOW14, Mar52, BBD⁺17], [BV04, §4.4.1],

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

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

$$\begin{aligned} & \text{minimize} && x^T D x + y^T y - \frac{1}{\gamma} \mu^T x \\ & \text{subject to} && y = F^T x \\ & && \mathbf{1}^T x = 1 \\ & && x \geq 0, \end{aligned} \tag{45}$$

Note that the Hessian of the objective in (45) is a diagonal matrix. Also, observe that FF^T does not appear in problem (45).

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 [Tib96, CWB08]. 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

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

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 50% 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 $\varepsilon_i \sim \mathcal{N}(0, 1/4)$. 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 [Hub64, Hub81]. The fitting problem is written as

$$\text{minimize } \sum_{i=1}^m \phi_{\text{hub}}(a_i^T x - b_i), \quad (46)$$

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

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

Problem (46) is equivalent to the following QP [BV04, Pag. 190]

$$\begin{aligned} & \text{minimize} && \frac{1}{2}u^T u + M\mathbf{1}^T v \\ & \text{subject to} && -u - v \leq Ax - b \leq u + v \\ & && 0 \leq u \leq M\mathbf{1} \\ & && v \geq 0. \end{aligned}$$

Problem instances. We generate the elements of A as $A_{ij} \sim \mathcal{N}(0, 1)$. 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 $m = 10n$ and $M = 1$, solve it 10 times and take the mean computation time.

A.7 Support vector machine

Support vector machine problem seeks an affine function that approximately classifies the two sets of points [CV95]. The problem can be stated as

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

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

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

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

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

References

- [ABQ⁺99] F. Allgöwer, T. A. Badgwell, J. S. Qin, J. B. Rawlings, and S. J. Wright. *Nonlinear Predictive Control and Moving Horizon Estimation – An Introductory Overview*, pages 391–449. Springer London, London, 1999.
- [ADD04] P. R. Amestoy, T. A. Davis, and I. S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 30(3):381–388, 2004.
- [AVDB18] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision (to appear)*, January 2018.
- [BB96] H. H. Bauschke and J. M. Borwein. On projection algorithms for solving convex feasibility problems. *SIAM Review*, 38(3):367–426, 1996.

- [BBD⁺17] Stephen Boyd, Enzo Busseti, Steve Diamond, Ronald N. Kahn, Kwangmoo Koh, Peter Nystrup, and Jan Speth. Multi-period trading via convex optimization. *Foundations and Trends in Optimization*, 3(1):1–76, 2017.
- [BBM17] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, June 2017.
- [BC11] H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 1st edition, 2011.
- [BEGFB94] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. Society for Industrial and Applied Mathematics, 1994.
- [Ben02] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418 – 477, 2002.
- [BG16] G. Banjac and P. Goulart. Tight global linear convergence rate bounds for operator splitting methods. www.optimization-online.org, October 2016.
- [BGSB17] G. Banjac, P. Goulart, B. Stellato, and S. Boyd. Infeasibility detection in the alternating direction method of multipliers for convex optimization. www.optimization-online.org, June 2017.
- [BHT04] H. Balakrishnan, I. Hwang, and C. J. Tomlin. Polynomial approximation algorithms for belief matrix maintenance in identity management. In *IEEE Conference on Decision and Control (CDC)*, volume 5, pages 4874–4879 Vol.5, December 2004.
- [BKL⁺13] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, April 2013.
- [BMOW14] S. Boyd, M. T. Mueller, B. O’Donoghue, and Y. Wang. Performance bounds and suboptimal policies for multiperiod investment. *Foundations and Trends in Optimization*, 1(1):1–72, 2014.
- [BPC⁺11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.
- [Bra10] A. Bradley. *Algorithms for the equilibration of matrices and their application to limited-memory Quasi-Newton methods*. PhD thesis, Stanford University, 2010.
- [BSM⁺17] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd. Embedded code generation using the OSQP solver. In *IEEE Conference on Decision and Control (CDC)*, 2017.
- [BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

- [CT06] G. Cornuejols and R. Tütüncü. *Optimization Methods in Finance*. Mathematics, Finance and Risk. Cambridge University Press, 2006.
- [CV95] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- [CWB08] E. J. Candés, M. B. Wakin, and S. Boyd. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier Analysis and Applications*, 14(5):877–905, 2008.
- [Dan63] G. B. Dantzig. *Linear programming and extensions*. Princeton University Press Princeton, N.J., 1963.
- [Dav05] T. A. Davis. Algorithm 849: A concise sparse Cholesky factorization package. *ACM Transactions on Mathematical Software*, 31(4):587–591, 2005.
- [Dav06] T. A. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006.
- [DB16] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.
- [DB17] S. Diamond and S. Boyd. Stochastic matrix-free equilibration. *Journal of Optimization Theory and Applications*, 172(2):436–454, February 2017.
- [DCB13] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pages 3071–3076, 2013.
- [DER89] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct methods for sparse matrices*. Oxford University Press, London, 1989.
- [DFH09] M. Diehl, H. J. Ferreau, and N. Haverbeke. *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation*, pages 391–417. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
- [DHL17] I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.
- [DM02] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, January 2002.
- [DR56] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82(2):421–439, 1956.
- [Eck89] J. Eckstein. *Splitting methods for monotone operators with applications to parallel optimization*. PhD thesis, MIT, 1989.

- [Eck94] J. Eckstein. Parallel alternating direction multiplier decomposition of convex programs. *Journal of Optimization Theory and Applications*, 80(1):39–62, 1994.
- [EF98] J. Eckstein and M. C. Ferris. Operator-splitting methods for monotone affine variational inequalities, with a parallel application to optimal control. *INFORMS Journal on Computing*, 10(2):218–235, 1998.
- [FB17] C. Fougner and S. Boyd. Parameter selection and pre-conditioning for a graph form solver. In S. Yurkovich R. Tempo and P. Misra, editors, *Emerging Applications of Control and System Theory (To appear)*. Springer, September 2017.
- [FKP⁺14] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl. qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.
- [FL98] R. Fletcher and S. Leyffer. Numerical experience with lower bounds for MIQP branch-and-bound. *SIAM Journal on Optimization*, 8(2):604–616, 1998.
- [FW56] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.
- [Gab83] D. Gabay. Chapter IX Applications of the method of multipliers to variational inequalities. *Studies in Mathematics and Its Applications*, 15:299 – 331, 1983.
- [GB15] P. Giselsson and S. Boyd. Metric selection in fast dual forward–backward splitting. *Automatica*, 62:1–10, 2015.
- [GB17] P. Giselsson and S. Boyd. Linear convergence and metric selection for Douglas-Rachford splitting and ADMM. *IEEE Transactions on Automatic Control*, 62(2):532–544, February 2017.
- [GFB16] P. Giselsson, M. Fält, and S. Boyd. Line Search for Averaged Operator Iteration. [arXiv:1603.06772](https://arxiv.org/abs/1603.06772), 2016.
- [GM75] R. Glowinski and A. Marroco. 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.
- [GM76] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17 – 40, 1976.
- [GMS⁺86] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright. On projected newton barrier methods for linear programming and an equivalence to karmarkar’s projective method. *Mathematical Programming*, 36(2):183–209, 1986.

- [GPM89] C. E. Garca, D. M. Prett, and M. Morari. Model predictive control: Theory and practicea survey. *Automatica*, 25(3):335 – 348, 1989.
- [Gre97] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. Society for Industrial and Applied Mathematics, 1997.
- [GTSJ15] E. Ghadimi, A. Teixeira, I. Shames, and M. Johansson. Optimal parameter selection for the alternating direction method of multipliers (ADMM): Quadratic problems. *IEEE Transactions on Automatic Control*, 60(3):644–658, 2015.
- [Gur16] Gurobi Optimization Inc. Gurobi optimizer reference manual. <http://www.gurobi.com>, 2016.
- [GVL96] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.
- [GW03] E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Trans. Math. Softw.*, 29(1):58–81, March 2003.
- [Hub64] P. J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.
- [Hub81] P. J. Huber. *Robust Statistics*. John Wiley & Sons, 1981.
- [HYW00] B. S. He, H. Yang, and S. L. Wang. Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *Journal of Optimization Theory and Applications*, 106(2):337–356, August 2000.
- [Int17] Intel Corporation. *Intel Math Kernel Library. User’s Guide*, 2017.
- [JGR⁺14] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari. Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, 59(12):3238–3251, December 2014.
- [Kan60] L. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960. English translation.
- [Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.
- [Kel95] C. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1995.
- [KM70] V. Klee and G. Minty. How good is the simplex algorithm. Technical report, Department of Mathematics, University of Washington, 1970.
- [KRU14] P. A. Knight, D. Ruiz, and B. Uçar. A symmetry preserving algorithm for matrix scaling. *SIAM Journal on Matrix Analysis and Applications*, 35(3):931–955, 2014.

- [LM79] P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.
- [Löf04] J. Löfberg. Yalmip : A toolbox for modeling and optimization in matlab. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.
- [Mar52] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.
- [MB10] J. Mattingley and S. Boyd. Real-time convex optimization in signal processing. *IEEE Signal Processing Magazine*, 27(3):50–61, May 2010.
- [MB12] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.
- [Meh92] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.
- [MM99] I. Maros and C. Mészáros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1-4):671–681, 1999.
- [MOS17] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 8.0 (Revision 57).*, 2017.
- [NLR⁺15] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. I. Jordan. A general analysis of the convergence of ADMM. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML’15, pages 343–352. JMLR.org, 2015.
- [NN94] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.
- [NW06] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer, Berlin, 2006.
- [OCPB16] B. O’Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, June 2016.
- [OSB13] B. O’Donoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, 21(6):2432–2442, November 2013.
- [PC11] T. Pock and A. Chambolle. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *2011 International Conference on Computer Vision*, pages 1762–1769, November 2011.
- [RM09] J. B. Rawlings and D. Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.

- [Rui01] D. Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical Report RAL-TR-2001-034, Rutherford Appleton Laboratory, Oxon, UK, 2001.
- [RW98] R. T. Rockafellar and R. J.-B. Wets. *Variational analysis*. Grundlehren der mathematischen Wissenschaften. Springer, 1998.
- [SB17] B. Stellato and G. Banjac. Benchmark examples for the OSQP solver. https://github.com/oxfordcontrol/osqp_benchmarks, 2017.
- [SK67] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.
- [SSS⁺16] G. Stathopoulos, H. Shukla, A. Szucs, Y. Pu, and C. N. Jones. Operator splitting methods in control. *Foundations and Trends in Systems and Control*, 3(3):249–362, 2016.
- [Tib96] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288, 1996.
- [TJ14] R. Takapoui and H. Javadi. Preconditioning via diagonal scaling. *EE364b: Convex Optimization II Class Project*, 2014.
- [Van95] R. Vanderbei. Symmetric quasi-definite matrices. *SIAM Journal on Optimization*, 5(1):100–113, 1995.
- [Woh17] B. Wohlberg. ADMM penalty parameter selection by residual balancing. [arXiv:1704.06209v1](https://arxiv.org/abs/1704.06209v1), 2017.
- [Wol59] P. Wolfe. The simplex method for quadratic programming. *Econometrica*, 27(3):382–398, 1959.
- [Wri97] S. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.