

PIQP: A Proximal Interior-Point Quadratic Programming Solver

Roland Schwan, Yuning Jiang, Daniel Kuhn, and Colin N. Jones

Abstract—This paper presents PIQP, a high-performance toolkit for solving generic sparse quadratic programs (QP). Combining an infeasible Interior Point Method (IPM) with the Proximal Method of Multipliers (PMM), the algorithm can handle ill-conditioned convex QP problems without the need for linear independence of the constraints. The open-source implementation is written in C++ with interfaces to C, Python, Matlab, and R leveraging the Eigen3 library. The method uses a pivoting-free factorization routine and allocation-free updates of the problem data, making the solver suitable for embedded applications. The solver is evaluated on the Maros-Mészáros problem set and optimal control problems, demonstrating state-of-the-art performance for both small and large-scale problems, outperforming commercial and open-source solvers.

I. INTRODUCTION

Convex quadratic programs are fundamental in many areas of applied mathematics and engineering. They are utilized in various applications, including portfolio optimization [1], optimal control [2], state estimation [3], and geometry processing [4]. Furthermore, QPs are a crucial building block of powerful optimization techniques, such as sequential quadratic programming [5] for nonlinear programming and branch-and-bound methods [6] for mixed integer quadratic programming. Due to their widespread use, the demand for efficient QP solvers that are both fast and reliable has increased, driven by emerging applications in areas such as optimal control, embedded systems, and signal processing.

In recent decades, significant research efforts have focused on developing efficient QP solvers. Numerous algorithms have been proposed, such as the classical active-set [7]–[9] and interior-point methods [10]–[12], first-order [13]–[16] and second-order Newton-type methods [17], [18], to design QP solvers that achieve high computational efficiency and scalability. As one of the most classical QP approaches, active-set based solvers, such as qpOASES [9], are known for their speed in solving small to medium-sized problems and the ability to warm-start using an estimate of the active constraint set. However, they have limited scalability, struggle to exploit sparsity, and are not robust to early termination. Compared to the active-set method, interior-point based solvers, such as ECOS and QPSWIFT, can be fast for large-scale problems and robust to early termination. They can solve sparse large-scale problems efficiently using

advanced linear algebra routines but are difficult to warm-start.

In contrast to classical approaches, specifically, interior-point methods, first-order solvers - including operator-splitting-based solvers like OSQP [16] and PROXQP [19] - can easily be warm-started, offer simplicity and tight complexity bounds. However, their convergence rates are slower than other methods. In contrast to this, the Newton-type solvers, such as the dual-Newton solver qpDUNES [17], can exploit sparsity and warm-start, but require restrictive assumptions, such as the QP being strongly convex and satisfying the linear independence constraint qualification (LICQ). These assumptions reduce their applicability and may cause robustness issues. Although the primal-dual Newton-type methods, such as FBRs [20], can relax the strong convexity requirement to the second-order sufficient condition (SOSC). However, they still require the LICQ.

Recently, hybrid methods that combine first-order and active set/interior-point methods have been the focus of increasing research attention. An example of such a method is QPNNLS [21]. Using the proximal point algorithm, a sequence of regularized QP problems is solved which converges to the solution of the original problem. The regularized QP problems are strictly convex and are solved using a non-negative least-square based active set method. Since solving QP subproblems is computationally expensive, QPNNLS relies heavily on warm-starting to reduce computational cost. Another related method is QPALM [22], which is based on the Augmented Lagrangian Method. Additionally, FBstab [23], a proximally stabilized Fischer-Burmeister method-based solver, employs a primal-dual version of the proximal point algorithm, in which the proximal subproblems are solved by using a Newton-type method.

In this paper, we present a software contribution, a hybrid approach based QP solver, called PIQP. The underlying algorithm implemented in PIQP follows the framework proposed in [24], which combines the interior-point method and the proximal method of multipliers (PMM). In particular, it uses one-iteration of Mohetra’s predictor-corrector method to deal with the proximal subproblem combining the dual gradient update.

Section II introduces some preliminaries, including the problem formulation and the main idea of PMM. The practical algorithm implemented in PIQP is presented in Section III. Section IV elaborates the numerical implementation details of PIQP. We demonstrate the effectiveness of our solver in Section V, in which it is compared against five existing state-of-art approaches, including two commercial solvers Gurobi and Mosek, three open-source

Open source: <https://github.com/PREDICT-EPFL/piqp>

This work was supported by the Swiss National Science Foundation under the NCCR Automation (grant agreement 51NF40_180545). Roland Schwan, Yuning Jiang, and Colin N. Jones are with the Automatic Control Lab, EPFL, Switzerland. Roland Schwan and Daniel Kuhn are with the Risk Analytics and Optimization Chair, EPFL, Switzerland. {roland.schwan, yuning.jiang, daniel.kuhn, colin.jones}@epfl.ch

solvers OSQP, SCS, and PROXQP. All solvers are evaluated on the Maros-Mészáros benchmark problems [25].

Notations: we denote the set of real numbers by \mathbb{R} , the set of n -dimensional real-valued vectors by \mathbb{R}^n , and the set of $n \times m$ -dimensional real-valued matrices by $\mathbb{R}^{n \times m}$. Moreover, we denote the subspace of symmetric matrices in $\mathbb{R}^{n \times n}$ by \mathbb{S}^n and the cone of positive semi-definite matrices by \mathbb{S}_+^n . We denote the identity matrix as $I_n \in \mathbb{R}^{n \times n}$ and a vector filled with ones as $\mathbf{1}_n \in \mathbb{R}^n$. The \circ operator indicates the element-wise multiplication of two vectors.

II. PRELIMINARIES

This section defines the problem formulation considered in PIQP and briefly reviews the proximal method of multipliers, which will be used later to design the underlying algorithm in PIQP.

A. Problem Formulation

PIQP considers quadratic programs in the form

$$\begin{aligned} \min_x \quad & \frac{1}{2}x^\top Px + c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & Gx \leq h, \end{aligned} \quad (1a) \quad (1b) \quad (1c)$$

with primal decision variables $x \in \mathbb{R}^n$, matrices $P \in \mathbb{S}_+^n$, $A \in \mathbb{R}^{p \times n}$, $G \in \mathbb{R}^{m \times n}$, and vectors $c \in \mathbb{R}^n$, $b \in \mathbb{R}^p$, and $h \in \mathbb{R}^m$. To design a practical numerical solver, it is convenient to rewrite (1) as the equivalent standard form problem

$$\begin{aligned} \min_{x,s} \quad & \frac{1}{2}x^\top Px + c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & Gx - h + s = 0, \\ & s \geq 0, \end{aligned} \quad (2a) \quad (2b) \quad (2c) \quad (2d)$$

where slack variables $s \in \mathbb{R}^m$ are introduced to lift the affine inequality (1c) into the equality (2c). Although introducing slack variables s may compromise strong convexity when $P \succ 0$, it has distinct computational benefits. For example, it makes it particularly easy to project onto the feasible set in the context of operator splitting-based approaches, this reformulation results in an easily computable projection operator; another example is the well-known interior-point method to be discussed in the next section. In the following, we mainly work with formulation (2).

B. Proximal method of multipliers

The augmented Lagrangian of problem (1) is defined as

$$\begin{aligned} \mathcal{L}_\delta^{\text{ALM}}(x, s; \lambda, \nu) := & \frac{1}{2}x^\top Px + c^\top x \\ & + \lambda^\top (Ax - b) + \frac{1}{2\delta} \|Ax - b\|_2^2 \\ & + \nu^\top (Gx - h + s) + \frac{1}{2\delta} \|Gx - h + s\|_2^2, \end{aligned} \quad (3)$$

where variables λ and ν are the Lagrangian multipliers of equality constraints (2b) and (2c), respectively, and $\delta > 0$ is a penalty parameter. We then introduce the iterations of the

proximal method of multipliers originally proposed in [26] as follows:

$$(x^+, s^+) \in \underset{\xi, s \geq 0}{\operatorname{argmin}} \mathcal{L}_\delta^{\text{ALM}}(\xi, s; \lambda, \nu) + \frac{\rho}{2} \|\xi - x\|_2^2, \quad (4a)$$

$$\lambda^+ = \lambda + \frac{1}{\delta} (Ax^+ - b), \quad (4b)$$

$$\nu^+ = \nu + \frac{1}{\delta} (Gx^+ - h + s^+), \quad (4c)$$

where superscript $+$ denotes the iteration update. Note that compared to the standard form for the original problem (1), (4) does not add a penalty term for the slack variable as it does not contribute to the cost of the primal problem (2).

III. PRACTICAL ALGORITHM

The method implemented in PIQP, following the framework proposed in [24], deals with (4a) by applying one iteration of the interior-point method, and uses the Mehrotra predictor-corrector method [27] to update the primal-dual iterates. To detail the algorithm, we define the log-barrier function

$$\Phi_\mu(s) = -\mu \sum_{i=1}^m \ln[s]_i, \quad (5)$$

where $[s]_i$ denotes the i -th element of s , and $\mu > 0$ is usually referred to as the barrier parameter. Replacing the constraint $s \geq 0$ in problem (4a) with the penalty term $\sigma \cdot \Phi_\mu(s)$ in the objective function yields

$$\min_{x,s} \mathcal{L}_{\delta_k}^{\text{ALM}}(x, s; \lambda_k, \nu_k) + \sigma_k \cdot \Phi_{\mu_k}(s) + \frac{\rho_k}{2} \|x - \xi_k\|_2^2$$

at iteration k , where $(\xi_k, \lambda_k, \nu_k)$ defines the primal-dual iterates, and $\sigma_k \in (0, 1]$ is the centering parameter in the predictor-corrector method discussed below. The first-order optimality conditions of the resulting unconstrained problem can then be represented as

$$Px + c + \rho_k(x - \xi_k) + A^\top y + G^\top z = 0, \quad (6a)$$

$$Ax - \delta_k(y - \lambda_k) - b = 0, \quad (6b)$$

$$Gx - \delta_k(z - \nu_k) - h + s = 0, \quad (6c)$$

$$s \circ z - \sigma_k \mu_k \mathbf{1}_m = 0, \quad (6d)$$

with auxiliary variables $y \in \mathbb{R}^p$ and $z \in \mathbb{R}^m$. Introducing auxiliary variables yields a sparser linear system of equations allowing highly efficient numerical routines.

Applying Newton's method to solve (6) results in the following linear equations

$$\underbrace{\begin{bmatrix} P + \rho_k I_n & A^\top & G^\top & 0 \\ A & -\delta_k I_p & 0 & 0 \\ G & 0 & -\delta_k I_m & I_m \\ 0 & 0 & S_k & Z_k \end{bmatrix}}_{J(s_k, z_k)} \underbrace{\begin{bmatrix} \Delta x_k \\ \Delta y_k \\ \Delta z_k \\ \Delta s_k \end{bmatrix}}_{\Delta \omega_k} = \underbrace{\begin{bmatrix} r_k^x \\ r_k^y \\ r_k^z \\ r_k^s \end{bmatrix}}_{r_k} \quad (7)$$

with initialization (x_k, y_k, z_k, s_k) and

$$r_k^x = -(Px_k + c + \rho_k(x_k - \xi_k) + A^\top y_k + G^\top z_k),$$

$$r_k^y = -(Ax_k + \delta_k(\lambda_k - y_k) - b),$$

$$r_k^z = -(Gx_k + \delta_k(\nu_k - z_k) - h + s_k),$$

$$r_k^s = -s_k \circ z_k + \sigma_k \mu_k \mathbf{1}_m$$

at iteration k .

Remark 1 In the implementation, we can eliminate Δs_k . To this end, we compute the Nesterov-Todd scaling $W_k = Z_k^{-1} S_k$ [28] such that (7) can be rewritten as

$$\underbrace{\begin{bmatrix} P + \rho_k I_n & A^\top & G^\top \\ A & -\delta_k I_p & 0 \\ G & 0 & -(W_k + \delta_k I_m) \end{bmatrix}}_{\tilde{J}(s_k, z_k)} \underbrace{\begin{bmatrix} \Delta x_k \\ \Delta y_k \\ \Delta z_k \end{bmatrix}}_{\tilde{r}_k} = \underbrace{\begin{bmatrix} r_k^x \\ r_k^y \\ \tilde{r}_k^z \end{bmatrix}}_{\tilde{r}_k}$$

with $\tilde{r}_k^z = r_k^z - Z_k^{-1} r_k^s$. Here, $Z_k \in \mathbb{R}^{m \times m}$ is a diagonal matrix with z_k on its diagonal. Note that the slack direction Δs_k can be reconstructed with $\Delta s_k = Z_k^{-1}(r_k^s - S_k \Delta z_k)$. This also ensures that $\tilde{J}(s_k, z_k)$ is symmetric.

Next, we present the three main steps of the Mehrotra predictor-corrector method:

1) *Prediction*: solve (7) with $\mu_k = 0$ and solution $\Delta \omega_k^a = (\Delta x_k^a, \Delta y_k^a, \Delta z_k^a, \Delta s_k^a)$

2) *Step Size and Centering Parameter*: compute the primal and dual step sizes

$$\alpha_p^a = \max \{ \alpha \in [0, 1] | s_k + \alpha \Delta s_k^a \geq (1 - \tau) s_k \},$$

$$\alpha_d^a = \max \{ \alpha \in [0, 1] | z_k + \alpha \Delta z_k^a \geq (1 - \tau) z_k \}, \quad (8)$$

with the scaling parameter $\tau = 0.995$ chosen heuristically [24], [28] that ensures the iterates do not get too close to the boundary of the feasible set, and evaluate the centering parameter following [27]

$$\sigma_k = \max\{0, \min\{1, \eta_k\}\}^3 \quad (9)$$

with $\eta_k = \left((s_k + \alpha_p^a \Delta s_k^a)^\top (z_k + \alpha_d^a \Delta z_k^a) \right) / \left((s_k^\top z_k) / m \right)$.

3) *Combined Correction and Centering*: compute $\mu_k = (s_k^\top z_k) / m$ and solving (7) with replacing r_k^s by

$$r_k^s = \underbrace{-S_k z_k}_{\text{prediction}} + \underbrace{-\Delta s_k^a \circ \Delta z_k^a}_{\text{correction}} + \underbrace{\sigma_k \cdot \mu_k \cdot \mathbf{1}_m}_{\text{centering}} \quad (10)$$

yields solution $\Delta \omega_k^c = (\Delta x_k^c, \Delta y_k^c, \Delta z_k^c, \Delta s_k^c)$. Then, update $(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1}) :=$

$$(x_k, s_k, y_k, z_k) + (\alpha_p^c \Delta x_k^c, \alpha_p^c \Delta s_k^c, \alpha_d^c \Delta y_k^c, \alpha_d^c \Delta z_k^c)$$

with step sizes

$$\alpha_p^c = \max \{ \alpha \in [0, 1] | s_k + \alpha \Delta s_k^c \geq (1 - \tau) s_k \},$$

$$\alpha_d^c = \max \{ \alpha \in [0, 1] | z_k + \alpha \Delta z_k^c \geq (1 - \tau) z_k \}. \quad (11)$$

Based on the aforementioned discussion, the primal-dual iterate $(\xi_k, \lambda_k, \nu_k)$ is optionally updated as outlined in Algorithm 2 proposed by [24, Section 5.1.4]. Here, we introduce the primal-dual residual with respect to the k -th iteration

$$p_k = \left\| \begin{bmatrix} Ax_k - b \\ Gx_k - h + s_k \end{bmatrix} \right\|_\infty,$$

$$d_k = \| Px_k + c + A^\top y_k + G^\top z_k \|_\infty.$$

Moreover, δ and ρ are limited by $\underline{\delta}$ and $\underline{\rho}$ for numerical stability.

Now, we can summarize a predictor-corrector-based practical computational framework for the interior-point proximal method of multipliers to solve (2) in Algorithm 1.

Remark 2 Algorithm 1 is a practical variant of the standard interior-point proximal method of multipliers as presented in [24, Algorithm 1], which substitutes the correction step and regularization update in [24, Algorithm 1] by the Mehrotra's predictor-corrector method and Algorithm 2, respectively. Note that Algorithm 2 is a numerical heuristic to update the primal-dual iterates $(\xi_k, \lambda_k, \nu_k)$ such that the convergence analysis proposed in [24, Section 3] cannot be rigorously established step by step. However, this heuristic leads to a more reliable and effective numerical convergence, although compared to the update in [24, Algorithm 1] without theoretical guarantees. Analyzing the theoretical convergence guarantee of practical Algorithm 1 is beyond the scope of this paper and will be investigated in our future work.

Algorithm 1 Interior-Point Proximal Method of Multipliers for Convex Quadratic Programming

Initialization: choose $(\xi_0, s_0, \lambda_0, \nu_0)$, set $\delta_0, \rho_0 > 0$.

- 1: **for** $k = 0, 1, \dots$ **do**
 - 2: Set $(x_k, y_k, z_k) = (\xi_k, \lambda_k, \nu_k)$;
 - 3: Solve (7) with $r_k^s = -S_k z_k$ for $\Delta \omega_k^a$;
 - 4: Compute (α_p^a, α_d^a) by (8);
 - 5: Update $\mu_k = (s_k^\top z_k) / m$ and σ_k by (9);
 - 6: Solve (7) with r_k^s by (10) for $\Delta \omega_k^c$;
 - 7: Compute (α_p^c, α_d^c) by (11) and update
 $(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1}) \leftarrow$
 $(x_k, s_k, y_k, z_k) + (\alpha_p^c \Delta x_k^c, \alpha_p^c \Delta s_k^c, \alpha_d^c \Delta y_k^c, \alpha_d^c \Delta z_k^c)$;
 - 8: Run Alg. 2 to get $(\xi_{k+1}, \lambda_{k+1}, \nu_{k+1})$, $(\delta_{k+1}, \rho_{k+1})$.
 - 9: **end for**
-

Algorithm 2 Penalty and Estimate Updates

Input: $r = |s_k^\top z_k - s_{k+1}^\top z_{k+1}| / s_k^\top z_k$.

Output: $(\xi_{k+1}, \lambda_{k+1}, \nu_{k+1})$ and $(\delta_{k+1}, \rho_{k+1})$.

- 1: **if** $p_{k+1} \leq 0.95 \cdot p_k$ **then**
 - 2: $(\lambda_{k+1}, \nu_{k+1}) \leftarrow (y_{k+1}, z_{k+1})$, $\delta_{k+1} \leftarrow (1 - r) \delta_k$
 - 3: **else**
 - 4: $(\lambda_{k+1}, \nu_{k+1}) \leftarrow (\lambda_k, \nu_k)$, $\delta_{k+1} \leftarrow (1 - r/3) \delta_k$
 - 5: **end if**
 - 6: **if** $d_{k+1} \leq 0.95 \cdot d_k$ **then**
 - 7: $\xi_{k+1} \leftarrow x_{k+1}$, $\rho_{k+1} \leftarrow (1 - r) \rho_k$
 - 8: **else**
 - 9: $\xi_{k+1} \leftarrow \xi_k$, $\rho_{k+1} \leftarrow (1 - r/3) \rho_k$
 - 10: **end if**
 - 11: $\delta_{k+1} \leftarrow \max\{\delta_{k+1}, \underline{\delta}\}$, $\rho_{k+1} \leftarrow \max\{\rho_{k+1}, \underline{\rho}\}$
-

IV. NUMERICAL IMPLEMENTATION

A. Initialization

We initialize the primal and dual variables using the standard method proposed in [29] by minimizing the unconstrained optimization problem

$$\min_{\xi_0, \tilde{s}_0} \mathcal{L}_{\delta_0}^{\text{ALM}}(\xi_0, \tilde{s}_0; 0, 0) + \frac{\rho_0}{2} \|\xi_0\|_2^2 + \frac{1}{2} \|\tilde{s}_0\|_2^2,$$

which can be posed as the solution to the linear system of equations

$$\begin{bmatrix} P + \rho_0 I_n & A^\top & G^\top \\ A & -\delta_0 I_p & 0 \\ G & 0 & -(1 + \delta_0) I_m \end{bmatrix} \begin{bmatrix} \xi_0 \\ \lambda_0 \\ \tilde{\nu}_0 \end{bmatrix} = \begin{bmatrix} -c \\ b \\ h \end{bmatrix} \quad (12)$$

with potentially negative slack variable $\tilde{s}_0 = -\tilde{\nu}_0$. Note that the structure is the same as for $\tilde{J}(\tilde{s}_0, \tilde{\nu}_0)$; hence, we can reuse the symbolic factorization, resulting in lower computational cost.

To guarantee that s_0 and ν_0 are in the non-negative orthant with sufficient magnitude, we calculate the conservative step sizes as

$$\begin{aligned}\Delta\tilde{s}_0 &= \max\{0, -1.5 \cdot \min(\tilde{s}_0)\}, \\ \Delta\tilde{\nu}_0 &= \max\{0, -1.5 \cdot \min(\tilde{\nu}_0)\},\end{aligned}$$

and similar to [24], we shift initial solutions further away from the barrier based on the normalized complementarity violation

$$\begin{aligned}\Delta s_0 &= \Delta\tilde{s}_0 + 0.5 \cdot \frac{(\tilde{s}_0 + \Delta\tilde{s}_0)^\top (\tilde{\nu}_0 + \Delta\tilde{\nu}_0)}{\sum_{i=0}^m ([\tilde{\nu}_0]_i + [\Delta\tilde{\nu}_0]_i)}, \\ \Delta\nu_0 &= \Delta\tilde{\nu}_0 + 0.5 \cdot \frac{(\tilde{s}_0 + \Delta\tilde{s}_0)^\top (\tilde{\nu}_0 + \Delta\tilde{\nu}_0)}{\sum_{i=0}^m ([\tilde{s}_0]_i + [\Delta\tilde{s}_0]_i)},\end{aligned}$$

resulting in the initial values for slack and inequality Lagrange multipliers $s_0 = \tilde{s}_0 + \Delta s_0$, $\nu_0 = \tilde{\nu}_0 + \Delta\nu_0$

B. Termination Criteria

We adopt the same terminal criteria for convergence as in SCS v3.0 [30]. More specifically, PIQP terminates when it finds primal variables $x \in \mathbb{R}^n$, $s \in \mathbb{R}^m$, and dual variables $y \in \mathbb{R}^p$, $z \in \mathbb{R}^m$ which satisfy the conditions

$$\left\| \begin{bmatrix} Ax - b \\ Gx - h + s \end{bmatrix} \right\|_\infty \leq \epsilon_{\text{abs}} \quad (13a)$$

$$+ \epsilon_{\text{rel}} \max(\|Ax\|_\infty, \|b\|_\infty, \|Gx\|_\infty, \|h\|_\infty, \|s\|_\infty),$$

$$\|Px + A^\top y + G^\top z + c\|_\infty \leq \epsilon_{\text{abs}} \quad (13b)$$

$$+ \epsilon_{\text{rel}} \max(\|Px\|_\infty, \|A^\top y\|_\infty, \|G^\top z\|_\infty, \|c\|_\infty),$$

$$|x^\top Px + c^\top x + b^\top y + h^\top z| \leq \epsilon_{\text{abs}} \quad (13c)$$

$$+ \epsilon_{\text{rel}} \max(|x^\top Px|, |c^\top x|, |b^\top y|, |h^\top z|),$$

where $\epsilon_{\text{abs}} > 0$ and $\epsilon_{\text{rel}} \geq 0$ are the user defined absolute and relative accuracies. Condition (13a) corresponds to the primal feasibility, and (13b) to the dual feasibility, which is common in most solvers like OSQP [16] or qpSWIFT [31]. The condition on the duality gap (13c) is less commonly checked, but if neglected, it can result in poor solution quality. The Maros-Mészáros problem set, for example, includes problems that are solved inaccurately without the criteria on the duality gap as discussed in [30, Section 7.2].

C. Sparse Pivot-Free LDL Factorization

The solution of the KKT system (7) constitutes the most computationally expensive step in any interior-point method. In our work, we have chosen to employ a direct method that is particularly well-suited for use in embedded applications. Specifically, we have opted for a modified version of Tim Davis' sparse pivot-free LDL factorization [32] with approximate minimum degree (AMD) ordering [33], resulting in the factorization

$$\Gamma K \Gamma^\top = LDL^\top,$$

where Γ is the permutation matrix of the AMD ordering reducing fill-in of the lower triangular matrix L , and D is a diagonal matrix.

To ensure that the LDL factorization of any symmetric permutation of the KKT matrix exists, it is sufficient if K is quasi-definite [34]. Although adding terms to the diagonal of (7) through the proximal method of multipliers typically ensures that K is almost certainly quasi-definite, there may be rare instances when it is not. In such cases, we slightly perturb the regularization parameters and retry the factorization. Thanks to this approach, there is no need to resort to techniques such as dynamic regularization with subsequent iterative refinement, which are employed in ECOS [11], resulting in less computational overhead.

The LDL factorization process consists of two phases: symbolic and numeric. During the symbolic phase, the elimination tree and the fill-in pattern of the lower triangular matrix L are determined, which provides the necessary information regarding the required memory. In the numeric phase, the factorization of K is performed using the previously calculated elimination tree, which fills in both L and D . Since the structure of $\tilde{J}(s_k, z_k)$ remains constant, we can reuse the elimination tree and fill-in pattern for every subsequent solve. As a result, symbolic computation can be avoided, and all previously allocated memory can be reused. This provides a significant advantage in terms of computational efficiency, particularly in the context of optimal control, where the structure of the sequence of solved problems stays constant.

D. Preconditioning

Preconditioning is a well-established technique for reducing the number of iterations for first-order methods by decreasing the condition number of the KKT system [28, Chapter 5]. Although commonly used for first-order methods, preconditioners can also be beneficial for interior-point methods by improving numerical stability and convergence. In our implementation, we adopt the Ruiz equilibration as our default preconditioner [35]. This technique scales the problem data diagonally to reduce the conditioning number of the unregularized KKT conditions (7) while being relatively cheap to compute.

Remark 3 *With the implementation of the Ruiz equilibration technique, we observed a notable improvement, achieving an average speedup of 22% on the Maros-Mészáros problem set. Before preconditioning, one less problem could be solved.*

E. Interface and Memory Allocation

In addition to the QP formulation (1) discussed in the paper, our implementation includes an interface for incorporating box constraints of the form $l \leq x \leq u$ with $l, u \in \mathbb{R}^n$. By exploiting this underlying structure, we are able to achieve computational speedups when factoring the corresponding KKT matrix.

Our QP solver, PIQP, leverages the high performance of C/C++ and the efficient vectorized matrix/vector operations

provided by the popular `Eigen3` library [36]. In addition to the C/C++ interface, we provide a `Python`, `Matlab`, and an `R` interface, making it easy to integrate `PIQP` into a wide range of computational pipelines. The code structure and `Python` interface have been adopted from `ProxSuite` [19].

The interface of `PIQP` is comprised of three main routines: `setup`, `update`, and `solve`. During the `setup` routine, all necessary memory structures are set up, the problem data is preconditioned, and the symbolic factorization, including the AMD ordering, is computed. The `update` routine is designed for updating problem data in subsequent solves, reusing the existing memory and factorization, assuming the sparsity structure remains unchanged. Similar to the `setup`, the updated problem data gets preconditioned. Upon executing the algorithm, the `solve` routine returns the solution, as well as the status of the optimization.

In predictive optimal control applications, it is common to solve the same problem repeatedly without any changes in structure. By utilizing the `update` routine, the problem data can be updated while reusing the symbolic factorization and already allocated memory. This significantly speeds up subsequent solutions.

Similar to solvers like `OSQP` [16], `ECOS` [11], and `PROXQP` [19], dynamic memory is only allocated during the `setup` routine, ensuring that the `update` and `solve` functions remain `malloc`-free. This feature is particularly important for embedded systems, where memory should be allocated statically or at least only once, without any subsequent dynamic allocations.

V. NUMERICAL EXAMPLES

To demonstrate the performance of `PIQP`, we benchmark it against the open-source toolkits `OSQP` [16], `SCS` [37], and `PROXQP` [19], as well as commercial solvers `Gurobi` [38] and `Mosek` [39]. Note that we did not compare our solver with the dense solver `qpOASES` [9] or conic programming solvers such as `ECOS` [11], as they do not support quadratic objectives natively.

To make the comparison fair, we enforced the same termination criteria across all solvers. Specifically, all solvers verify the primal feasibility (13a) and dual feasibility (13b). However, the duality gap (13c) is only checked by `PIQP`, `SCS`, and `PROXQP`. As a result, other solvers may report an optimal solution that fails to satisfy the duality gap condition and may require more time or even fail to find a solution that meets our termination criteria. Thus, it is important to note that the reported results for these solvers may be overly optimistic. For example, `Mosek` would fail on 75% of problems if we check the termination criteria of the solution.

All benchmarks are run on a workstation with an AMD Ryzen Threadripper 3990X 4.3 GHz CPU. `Gurobi` and `Mosek` were limited to a single thread, and all solvers were subject to a 1000 second time limit. We use an adapted benchmark framework from `OSQP` [16].

A. Maros-Mészáros problems

We consider the standard Maros-Mészáros problem set, comprised out of 138 *hard* QP problems [25]. Most problems

are very sparse, and a certain subset is numerically extremely ill-conditioned.

Our implementation is based on the `Python` interface of all solvers. Moreover, we only use the internally measured times reported by the solver. This includes setup and solution time.

We conduct two sets of experiments to evaluate the performance of our QP solver. In the first scenario, we aim to find solutions with low accuracy, setting $\epsilon_{\text{abs}} = 10^{-3}$ and $\epsilon_{\text{rel}} = 10^{-4}$. In the second scenario, we target highly accurate solutions, setting $\epsilon_{\text{abs}} = 10^{-8}$ and $\epsilon_{\text{rel}} = 10^{-9}$.

Table I summarizes the failure rates of the solvers for the different accuracy settings. Compared to the other solvers, our solver, `PIQP`, demonstrates remarkable robustness, successfully solving all problems with low accuracy settings and failing to solve only a single problem with high accuracy settings.

TABLE I
FAILURE RATES ON THE MAROS-MÉSZÁROS PROBLEM SET

	PIQP	OSQP	SCS	PROXQP	GUROBI	MOSEK
Low Accuracy	0.00%	4.35%	4.35%	12.32%	6.52%	5.07%
High Accuracy	0.72%	31.16%	23.19%	23.19%	6.52%	10.14%

We provide the individual solve times for each problem in the Maros-Meszaros problem set for high accuracy settings in Figure 1. The problems are sorted by `PIQP` solve time, highlighting the speed of our solver relative to the others. Notably, our solver is almost always the fastest for most problems. These results highlight the advantages of interior-point methods in high-accuracy settings, where they outperform most first-order methods.

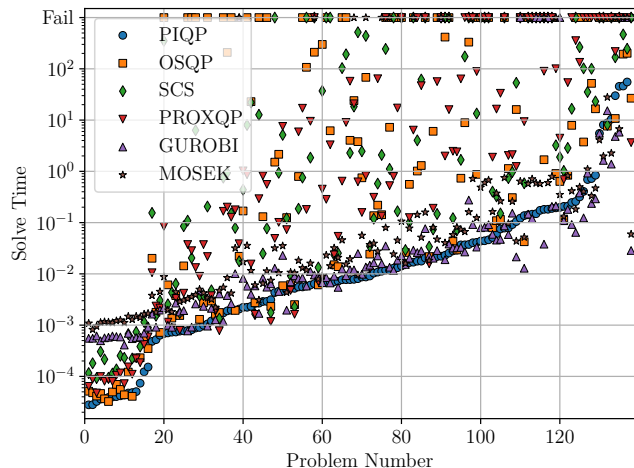


Fig. 1. Solve times for high accuracy settings for individual problems in the Maros-Mészáros problem set, ordered by `PIQP` solve time. The lowest point in each column corresponds to the fastest solver.

We include the Dolan-Moré performance profiles [40] for the high accuracy settings in Figure 2. The x-axis of the graphs shows the solve time normalized by the fastest solver, with a performance ratio of one indicating that the solver was the fastest and a performance ratio of ten indicating that the solver was ten times slower than the fastest solver for

a particular problem. The y-axis of the graphs shows the ratio of problems solved. For instance, PIQP was the fastest solver for 65% of the problems and took at most five times longer for 96% of the problems.

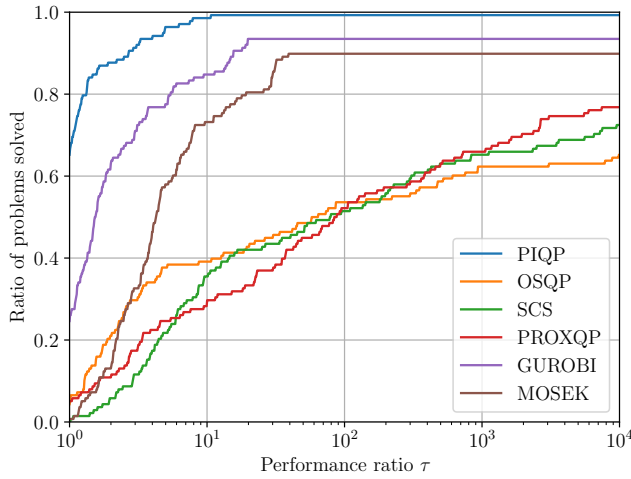


Fig. 2. Performance profiles on the Maros-Mészáros problem set for high accuracy settings.

REFERENCES

- [1] M. Rubinstein, “Markowitz’s” portfolio selection”: A fifty-year retrospective,” *The Journal of Finance*, vol. 57, no. 3, pp. 1041–1045, 2002.
- [2] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing Madison, WI, 2017.
- [3] 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,” *Advances in control: Highlights of ECC’99*, pp. 391–449, 1999.
- [4] Y. Zhu, R. Bridson, and D. M. Kaufman, “Blended cured quasi-Newton for distortion optimization,” *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–14, 2018.
- [5] P. T. Boggs and J. W. Tolle, “Sequential quadratic programming,” *Acta Numerica*, vol. 4, pp. 1–51, 1995.
- [6] R. Fletcher and S. Leyffer, “Numerical experience with lower bounds for MIQP branch-and-bound,” *SIAM Journal on Optimization*, vol. 8, no. 2, pp. 604–616, 1998.
- [7] A. Bemporad, “A quadratic programming algorithm based on nonnegative least squares with applications to embedded model predictive control,” *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 1111–1116, 2015.
- [8] G. Cimini and A. Bemporad, “Complexity and convergence certification of a block principal pivoting method for box-constrained quadratic programs,” *Automatica*, vol. 100, pp. 29–37, 2019.
- [9] 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*, vol. 6, pp. 327–363, 2014.
- [10] S. J. Wright, *Primal-Dual Interior-Point Methods*. SIAM, 1997.
- [11] A. Domahidi, E. Chu, and S. Boyd, “ECOS: An SOCP solver for embedded systems,” in *European Control Conference*, 2013, pp. 3071–3076.
- [12] G. Frison and M. Diehl, “HPIPM: A high-performance quadratic programming framework for model predictive control,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020.
- [13] T. Goldstein, B. O’Donoghue, S. Setzer, and R. Baraniuk, “Fast alternating direction optimization methods,” *SIAM Journal on Imaging Sciences*, vol. 7, no. 3, pp. 1588–1623, 2014.
- [14] P. Patrinos and A. Bemporad, “An accelerated dual gradient-projection algorithm for embedded linear model predictive control,” *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 18–33, 2013.
- [15] P. Patrinos, A. Guiggiani, and A. Bemporad, “A dual gradient-projection algorithm for model predictive control in fixed-point arithmetic,” *Automatica*, vol. 55, pp. 226–235, 2015.
- [16] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, “OSQP: An operator splitting solver for quadratic programs,” *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.
- [17] J. V. Frasch, S. Sager, and M. Diehl, “A parallel quadratic programming method for dynamic optimization problems,” *Mathematical Programming Computation*, vol. 7, pp. 289–329, 2015.
- [18] P. Patrinos, P. Sopasakis, and H. Sarimveis, “A global piecewise smooth Newton method for fast large-scale model predictive control,” *Automatica*, vol. 47, no. 9, pp. 2016–2022, 2011.
- [19] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, “PROX-QP: yet another quadratic programming solver for robotics and beyond,” in *Robotics: Science and Systems*, 2022.
- [20] D. Liao-McPherson, M. Huang, and I. Kolmanovsky, “A regularized and smoothed Fischer–Burmeister method for quadratic programming with applications to model predictive control,” *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2937–2944, 2018.
- [21] A. Bemporad, “A numerically stable solver for positive semidefinite quadratic programs based on nonnegative least squares,” *IEEE Transactions on Automatic Control*, vol. 63, no. 2, pp. 525–531, 2017.
- [22] B. Hermans, A. Themelis, and P. Patrinos, “QPALM: A proximal augmented Lagrangian method for nonconvex quadratic programs,” *Mathematical Programming Computation*, vol. 14, no. 3, pp. 497–541, 2022.
- [23] D. Liao-McPherson and I. Kolmanovsky, “FBstab: A proximally stabilized semismooth algorithm for convex quadratic programming,” *Automatica*, vol. 113, p. 108801, 2020.
- [24] S. Pougkakiotis and J. Gondzio, “An interior point-proximal method of multipliers for convex quadratic programming,” *Computational Optimization and Applications*, vol. 78, no. 2, pp. 307–351, 2021.
- [25] I. Maros and C. Mészáros, “A repository of convex quadratic programming problems,” *Optimization Methods and Software*, vol. 11, no. 1–4, pp. 671–681, 1999.
- [26] R. T. Rockafellar, “Augmented Lagrangians and applications of the proximal point algorithm in convex programming,” *Mathematics of Operations Research*, vol. 1, no. 2, pp. 97–116, 1976.
- [27] S. Mehrotra, “On the implementation of a primal-dual interior point method,” *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.
- [28] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2006.
- [29] M. Andersen, J. Dahl, Z. Liu, and L. Vandenbergh, “Interior-point methods for large-scale cone programming,” in *Optimization for Machine Learning*. The MIT Press, 2011.
- [30] B. O’Donoghue, “Operator splitting for a homogeneous embedding of the linear complementarity problem,” *SIAM Journal on Optimization*, vol. 31, no. 3, pp. 1999–2023, 2021.
- [31] A. G. Pandala, Y. Ding, and H.-W. Park, “qpSWIFT: A real-time sparse quadratic program solver for robotic applications,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3355–3362, 2019.
- [32] T. A. Davis, “Algorithm 849: A concise sparse Cholesky factorization package,” *ACM Transactions on Mathematical Software*, vol. 31, no. 4, pp. 587–591, 2005.
- [33] P. R. Amestoy, T. A. Davis, and I. S. Duff, “Algorithm 837: AMD, an approximate minimum degree ordering algorithm,” *ACM Transactions on Mathematical Software*, vol. 30, no. 3, pp. 381–388, 2004.
- [34] R. J. Vanderbei, “Symmetric quasidefinite matrices,” *SIAM Journal on Optimization*, vol. 5, no. 1, pp. 100–113, 1995.
- [35] R. Daniel, “A scaling algorithm to equilibrate both rows and columns norms in matrices,” Rutherford Appleton Laboratory, Tech. Rep., 2001.
- [36] G. Guennebaud, B. Jacob *et al.*, “Eigen v3,” <http://eigen.tuxfamily.org>, 2010.
- [37] 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*, vol. 169, no. 3, pp. 1042–1068, 2016.
- [38] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2023.
- [39] M. ApS, *MOSEK Optimizer API for Python 10.0.39*, 2023.
- [40] E. D. Dolan and J. J. Moré, “Benchmarking optimization software with performance profiles,” *Mathematical Programming*, vol. 91, no. 2, pp. 201–213, 2002.