PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond

Antoine Bambade, Sarah El-Kazdadi, Adrien Taylor, Justin Carpentier

ProxQP: Yet another Quadratic Programming Solver for Robotics and beyond

Antoine Bambade*†‡, Sarah El-Kazdadi*†, Adrien Taylor*†, Justin Carpentier*†
*Inria, Paris, France
†Département d'informatique de l'ENS, PSL Research University, Paris, France
‡ École des Ponts, Marne-la-Vallée, France
Corresponding author: antoine.bambade@inria.fr

***Abstract*—Quadratic programming (QP) has become a core modelling component in the modern engineering toolkit. This is particularly true for simulation, planning and control in robotics. Yet, modern numerical solvers have not reached the level of efficiency and reliability required in practical applications where speed, robustness, and accuracy are all necessary. In this work, we introduce a few variations of the well-established augmented Lagrangian method, specifically for solving QPs, which include heuristics for improving practical numerical performances. Those variants are embedded within an open-source software which includes an efficient C++ implementation, a modular API, as well as best-performing heuristics for our test-bed. Relying on this framework, we present a benchmark studying the practical performances of modern optimization solvers for convex QPs on generic and complex problems of the literature as well as on common robotic scenarios. This benchmark notably highlights that this approach outperforms modern solvers in terms of efficiency, accuracy and robustness for small to medium-sized problems, while remaining competitive for higher dimensions.**

I. Introduction

Over the past decades, optimization has become an essential component of robotics and a key enabler to simplify and systematize the programming of complex robot movements. Nowadays, many robotic problems – ranging from simulation, control, planning and estimation – are framed as optimization problems. An important class of such optimization problems is that of convex quadratic problems, which allows dealing with, among others, friction-less unilateral contact modelling [34], constrained forward dynamics [3], inverse kinematics and dynamics for task control [8, 17, 22], and legged locomotion [42], to name a few. QPs are also commonly used also as subroutines for solving more complex problems as for instance in the context of constrained optimal control problems [23, 19, 41].

Formally, a QP corresponds to the minimization of a convex quadratic cost under some linear equality and inequality constraints. It is mathematically described as:

$$\begin{aligned} \min_{x\in\mathbb{R}^d} \quad & \frac{1}{2}x^T H x + g^T x \\ \text{s.t.} \begin{cases} Ax = b, \\ Cx \leq u, \end{cases} \end{aligned} \tag{QP}$$

where $H \in \mathbb{R}^{d\times d}$ is a symmetric positive semi-definite matrix (notation $H \in \mathbb{S}^d_+$), $A \in \mathbb{R}^{n_e\times d}$, $C \in \mathbb{R}^{n_i\times d}$, $b \in \mathbb{R}^{n_e}$, and $u \in \mathbb{R}^{n_i}$. d is the so-called problem dimension, while n_e and n_i are the numbers of equality and inequality constraints respectively. In many scenarios, QP instances have to be solved at very high-frequency (e.g., 1 kHz for inverse dynamics), under various levels of accuracy depending on the application, and potentially in relatively large dimension (e.g., model predictive control). Reliable, accurate, fast and numerically robust solvers are thus essential in modern robotic applications. Several methods and their associated solvers have been developed by the optimization community (e.g., qpOASES [9], Quadprog [12], OSQP [39], Gurobi [30], Mosek [28], BPMPD [26], IPOPT [44], OOQP [10], etc.). Interestingly, in the robotics community, Pandala et al. [31] have also proposed a dedicated solver based on off-the-shelf interior-point methods combined with adapted sparse routines, which has shown to operate efficiently for generating different gaits on quadrupedal robots.

To the best of our knowledge, there is currently no numerical QP solver meeting all the aforementioned requirements. In this work, we propose a new approach, based on the well-established augmented Lagrangian formalism [33, 18, 36], for solving generic QPs, and provide a corresponding efficient numerical C++ implementation. In this approach, we notably propose to combine the bounded constraint Lagrangian (BCL) globalization strategy with a (primal-dual) proximal method of multipliers applied to (QP). These contributions pave the way towards more advanced numerical methods for dealing with complex optimization problems in robotics, with the ambition of significantly reducing the computational burden, increase the numerical robustness of the solver while also lowering the need of manual tuning of the underlying hyper-parameters. To validate our approach, we benchmark the new solver against against its state-of-the-art competitors on various problems ranging from randomly generated QPs to the hard Maros-Mészàros [24] QPs of the optimization literature, which includes classical robotic problems.

The paper is organized as follows. In Section II, we review different commonly used methods for solving QPs. Section III defines the different notations, reviews the classic augmented Lagrangian techniques and introduces the proposed primal-dual method for solving QPs. The core aspects of the method are detailed in Section IV. The complete algorithm is depicted in Section V. Finally, we benchmark the new solver against the state-of-the-art in Section VI.

II. Related Work

Generic constrained convex QPs are commonly solved with iterative (or indirect) methods. These methods are traditionally divided into two main families: (i) *active-set* methods, and (ii) *penalization methods*. When the problem contains no inequality constraint, direct methods can also be used, as solving the corresponding QP can be done directly by solving the corresponding (linear) Karush-Kuhn-Tucker (KKT) system [29].

Iterative methods of both families typically aim at solving a cascade of simpler intermediary optimization problems whose solutions tend to those of the original problem.

1) Active-set methods: This family of methods, developed in the 50s, aims at determining the set of active constraints at an optimal point of (QP), see, e.g., [29, Section 16.5] for an introduction. Once this set of active constraints is determined, the constrained QP can be solved directly by considering a QP with equality constraints only, and whose solution matches that of the original QP. Popular active set-based convex QP solvers include the open-source qpOASES [9], Quadprog [12], and the QPA module in the open source software GALAHAD [13].

On the negative side, active-set methods typically suffer from undesirable effects, such as "active-set cycling" [29, sections 13.5 and 16.5]. This kind of phenomenon might slow down the method when the QP at hand does not satisfy constraint qualification properties (such as the traditional "linear independence constraint qualification" (LICQ), see, e.g., [29, Definition 12.4]), that are sometimes hard to verify in practice and are often not met.

On the positive side, warm-start strategies can easily be incorporated within active-set methods. This is usually key for solving cascades of similar QPs, which is common in applications such as sequential quadratic programming (SQP) and model predictive control (MPC).

2) Penalization methods: This family of methods transform the original constrained problem into a sequence of problems with either no constraints or very simple ones (such as sign constraints on some variables). The new problems typically have objectives that consist in two terms: (i) the objective of the original problem, and (ii) a term for penalizing points not being feasible for the original problem. There are two very common types of penalization methods used in practice.

a) Interior-point methods: This family of penalization methods forces through a *barrier function* the sequence of intermediary optimization problems to have strictly feasible solutions with respect to the domain of the original problem; see, e.g., [29, Section 16.6] for an introduction. Primal-dual interior-point methods [25, 43] became popular in the 90s due to their good practical performances across a wide range of problems. Standard solvers using an interior-point method are commercial solvers Gurobi [30] and Mosek [28], closed-source BPMPD [26], open-source solver OOQP [10] and qpSWIFT [31].

Because of the homotopy-based structure of this family of methods, one of their main drawbacks is the difficulty to use them with warm-starting procedures, when solving a sequence of related QPs (e.g., within SQP or MPC settings).

b) Augmented Lagrangian-type methods: This family of methods is primarily based on the idea of Lagrangian relaxations with an additional quadratic penalization term (possibly piecewise) for encouraging feasibility of the iterate (see, e.g., [29, Section 17.3]). This kind of techniques emerged in the 70s through the works of Hestenes and Powell [33, 18] and then later with those of Rockafellar [36] which tightly emphasized their linked with the so-called "proximal-point method". Indeed, augmented Lagrangian-type methods naturally arise by applying proximal-point methods on either the dual or saddle-point formulations to (QP), thereby offering the advantage of converging under relatively weak assumptions. Augmented Lagrangian-type methods also have the advantageous property of being able to exploit warm-start procedures. However, they might exhibit slow convergence behaviors in practice. Standard solvers based on augmented Lagrangians for solving QPs include the recent OSQP [39] (via an alternating direction method of multipliers) and QPALM [16].

III. Proximal Method of Multipliers for Quadratic Programming

This section introduces the main contribution of the paper: a generic algorithm to solve QP problems of the form of (QP). We start by recalling the optimality conditions related to (QP). Then, we review the notions of augmented Lagrangian and proximal-point algorithms in the simpler context of equality constrained QPs. These concepts are then adapted to properly handle QP problems composed of both equality and inequality constraints. We conclude this section with a few practical considerations for rendering the proposed approach numerically more stable and practically more efficient.

A. Optimality conditions

The Lagrangian $\mathcal{L}$ associated to (QP) is defined by:

$$\mathcal{L}(x,y,z) := \tfrac{1}{2}x^THx + g^Tx + y^T(Ax-b) + z^T(Cx-u), \tag{1}$$

with $x \in \mathbb{R}^d$, $y \in \mathbb{R}^{n_e}$, $z \in \mathbb{R}^{n_i}_+$. For linearly constrained convex optimization problems such as (QP), strong duality holds and the associated KKT conditions are necessary and sufficient for ensuring a primal-dual point (x,y,z) to be optimal (see, e.g., [1, Section 5.2.3] and [39, Section 2, page 5] for more details). For (QP), the KKT system is given by the set of equations:

$$\begin{cases} Hx + g + A^Ty + C^Tz = 0, \\ Ax - b = 0, \\ Cx \leq u, \\ z \odot [Cx-u] = 0, \end{cases} \tag{KKT}$$

where $\odot$ denotes the Hadamard product (i.e., for two vectors $u, v \in \mathbb{R}^d$, $u \odot v \in \mathbb{R}^d$ is the vector whose ith entry is u_iv_i). In practice, we look for a triplet (x,y,z) satisfying these optimality conditions (KKT) up to a certain level of predefined

accuracy $\epsilon_{abs} > 0$ (dependent of the application), leading us to the following natural absolute "stopping criterion":

$$\begin{cases} \|Hx + g + A^T y + C^T z\|_\infty \leq \epsilon_{abs}, \\ \|Ax - b\|_\infty \leq \epsilon_{abs}, \\ \|[Cx - u]_+\|_\infty \leq \epsilon_{abs}. \end{cases} \tag{2}$$

The ℓ_∞ norm is preferred to the ℓ_2 norm as it is independent of the problem dimensions. It is also common to consider relative convergence criteria for early-stopping, as absolute targets might not be reached due to numerical issues [39, 28, 30].

B. Equality-constrained quadratic programs

In this section, we provide a high-level overview on the proximal method of multipliers (PMM), which is at the heart of our approach. PMM is closely related to the probably even more famous augmented Lagrangian method (ALM), which we review first in the context of equality-constrained QPs (i.e., when $n_i = 0$).

a) Augmented Lagrangian: The ALM [36, Section 4] relies on augmenting the standard Lagrangian $\mathcal{L}$, defined in (1), with a squared ℓ_2 penalization of the linear constraints:

$$\mathcal{L}_A(x, y; \mu_e) := \mathcal{L}(x, y, 0) + \tfrac{1}{2\mu_e}\|Ax - b\|_2^2, \tag{3}$$

where $\mu_e > 0$ is a positive penalty parameter. The ALM alternates between the minimization of $\mathcal{L}_A$ with respect to the primal variables x and a simple update rule for the dual variables y:

$$\begin{aligned} x^{k+1} &= \arg\min_x \ \mathcal{L}_A(x, y^k; \mu_e), \\ y^{k+1} &= y^k + \tfrac{1}{\mu_e}(Ax^{k+1} - b). \end{aligned} \tag{4}$$

Using the expression of $\mathcal{L}_A$, one can obtain a more explicit expression for the iterations in the case of equality-constrained QPs:

$$\begin{aligned} x^{k+1} &= -\left(H + \tfrac{1}{\mu_e}A^T A\right)^{-1}[g + A^T(y^k - \mu_e b)], \\ y^{k+1} &= y^k + \tfrac{1}{\mu_e}(Ax^{k+1} - b). \end{aligned} \tag{5}$$

At this stage, there are two conflicting goals to balance in this iterative process. First, the smaller the value of μ_e the faster the convergence: indeed, $\frac{1}{\mu_e}$ can be seen as a step-size of a proximal point method applied on the dual of the QP, as (4) can also be equivalently written as:

$$y^{k+1} = \arg\max_y \min_x \ \mathcal{L}(x, y, 0) - \tfrac{\mu_e}{2}\|y - y^k\|_2^2, \tag{6}$$

(see [36, Section 4] for details), and the x^{k+1} update might be chosen as

$$x^{k+1} \in \arg\min_x \ \mathcal{L}(x, y^{k+1}, 0) - \tfrac{\mu_e}{2}\|y^{k+1} - y^k\|_2^2. \tag{7}$$

Second, as μ_e gets smaller, the conditioning of $(H + \frac{1}{\mu_e}A^T A)$ gets worse, thereby limiting the numerical applicability of the approach, particularly when the condition number of A, thus the one of $A^T A$ (as $\text{cond}(A^T A) = \text{cond}(A)^2$), is already potentially large [29, Section 17.1]. Such cases are typical in robotics, for instance in the context of kinematic singularity or redundant constraints, just to name a few.

b) Proximal method of multipliers: PMM is an alternative to ALM with an additional proximal term on the primal variables [36, Equation 1.9], following the scheme:

$$\begin{aligned} (x^{k+1}, y^{k+1}) = \arg\min_{x,y}\max \mathcal{L}(x, y, 0) - \tfrac{\mu_e}{2}\|y - y^k\|_2^2 \\ + \tfrac{\rho}{2}\|x - x^k\|_2^2, \end{aligned} \tag{8}$$

where $\rho > 0$ is an additional penalty parameter ($\frac{1}{\rho}$ corresponds to a step-size for the primal proximal term). Explicit maximization with respect to the dual variable allows recovering a method similar in spirit with the ALM formulation (4):

$$\begin{cases} x^{k+1} = \arg\min_x \Phi_\rho^k(x), \\ y^{k+1} = y^k + \frac{1}{\mu_e}(Ax^{k+1} - b), \end{cases} \tag{9}$$

where $\Phi_\rho^k(x) := \mathcal{L}_A(x, y^k; \mu_e) + \frac{\rho}{2}\|x - x^k\|_2^2$ is often referred to as the proximal augmented Lagrangian (PAL) [15]. In practice, one can directly solve (8) via its optimality conditions, encoded in the following linear system of equations:

$$\begin{bmatrix} H + \rho I & A^T \\ A & -\mu_e I \end{bmatrix} \begin{bmatrix} x^{k+1} \\ y^{k+1} \end{bmatrix} = \begin{bmatrix} \rho x_k - g \\ b - \mu_e y_k \end{bmatrix}. \tag{10}$$

It is worth mentioning that this linear system involves a matrix that is always nonsingular thanks to the two regularization terms $\frac{\rho}{2}\|x - x^k\|_2^2$ and $\frac{\mu_e}{2}\|y - y^k\|_2^2$. In other words, the problem (8) is always well-defined in the iterative process. As a comparison, one can notice that the linear system depicting optimality conditions for ALM might be singular, encoding the fact that an acceptable x^{k+1} might not be unique in (7). As a consequence, intermediary computations involved in PMM are numerically more stable by construction. Yet, another advantageous feature of PMM is that it is guaranteed to converge to an optimal primal-dual pair $(x^k, y^k) \to (x^*, y^*)$ under relatively weak assumptions (existence of an optimal primal-dual pair with zero duality gap; see [36, Theorem 7]). Under similar weak assumptions (existence of an optimal dual solution; see [36, Theorem 4]), ALM is only guaranteed to converge on the dual variable $y^k \to y^*$. It is also worth noticing that in the case of a QP involving only linear equality constraints, one iteration of either ALM or PMM can be cast as the solution of single linear system. We see in the next section that the situation changes markedly with inequalities.

C. Inequality constrained quadratic programs

In the presence of inequality constraints in (QP), a natural extension of (8) consists in iterating

$$\begin{aligned} &(x^{k+1}, y^{k+1}, z^{k+1}) \\ &\quad = \arg\min_{x,y,z\geq 0}\max \ \mathcal{L}(x, y, z) + \tfrac{\rho}{2}\|x - x^k\|_2^2 \\ &\qquad\qquad - \tfrac{\mu_e}{2}\|y - y^k\|_2^2 - \tfrac{\mu_i}{2}\|z - z^k\|_2^2. \end{aligned} \tag{11}$$

By explicit maximization in (y, z), one can also reach an equivalent formulation in terms of

$$\begin{cases} x^{k+1} = & \arg\min_x \Phi_\rho^k(x), \\ y^{k+1} = & y^k + \frac{1}{\mu_e}(Ax^{k+1} - b), \\ z^{k+1} = & [z^k + \frac{1}{\mu_i}(Cx^{k+1} - u)]_+ \end{cases} \tag{12}$$

where $[.]_+$ stands for the (componentwise) nonnegative part, and where the PAL $\Phi^k_\rho(x)$ is now defined using the augmented Lagrangian formulation for the problem involving inequalities [36, Eqs 1.4–1.5]:

$$\begin{aligned}\mathcal{L}_A(x,y,z;\mu_e,\mu_i) &:= \mathcal{L}(x,y,0) + \tfrac{1}{2\mu_e}\|Ax-b\|_2^2 \\ &+ \tfrac{1}{2\mu_i}\Big(\big\|[Cx-u+\mu_i z]_+\big\|_2^2 - \|\mu_i z\|_2^2\Big),\end{aligned} \tag{13}$$

$$\Phi^k_\rho(x) := \mathcal{L}_A(x,y^k,z^k;\mu_e,\mu_i) + \tfrac{\rho}{2}\|x-x^k\|_2^2. \tag{14}$$

One can note that this PAL is a piecewise quadratic function. When only equality constraints are involved in the QP, the PAL is a simple quadratic function, and the solution of the intermediary subproblems are obtained by solving a single linear system. The situation becomes a bit more subtle in the context of inequality constraints. For this reason, the practical algorithm that is proposed and investigated below is based on an *approximate* version of PMM (first proposed in [36]):

$$\begin{aligned}&(x^{k+1},y^{k+1},z^{k+1}) \\ &\approx_{\epsilon^k} \arg\min_{x,y,z\geq 0}\max \mathcal{L}(x,y,z) + \tfrac{\rho}{2}\|x-x^k\|_2^2 \\ &\qquad - \tfrac{\mu_e}{2}\|y-y^k\|_2^2 - \tfrac{\mu_i}{2}\|z-z^k\|_2^2\end{aligned} \tag{15}$$

where $\approx_{\epsilon^k}$ stands for requiring $(x^{k+1},y^{k+1},z^{k+1})$ to be an ϵ^k-approximate solution to the intermediate saddle-point subproblem. We control the accuracy level of this approximation via the following condition:

$$\|r^k(x^{k+1},y^{k+1},z^{k+1})\|_\infty \leq \epsilon^k, \tag{16}$$

where r^k is some nonlinear operator gathering the KKT conditions for the saddle-point subproblem at iteration k. In others words $\|r^k(x,y,z)\|_\infty = 0$ iff (x,y,z) is the solution to (11). For more details on r^k, see Section IV-B.

For going further, we must address two important remaining aspects: (i) choosing appropriate rules for setting ϵ^k (large values for ϵ^k will obviously not lead to convergence of this numerical method) and μ_e^k and μ_i^k (appropriate tuning of those step-size rules critically impacts the practical performance of the method); (ii) computing suitable triplets $(x^{k+1},y^{k+1},z^{k+1})$ satisfying (16) in an efficient way.

In the next section, we review the BCL strategy (originating from [4]) for dealing with (i). This strategy is key in our practical implementation of the QP solver. The problem (ii) of finding suitable approximations for the proximal subproblems is handled in Section IV.

D. BCL globalization strategy

In this section, we explain our strategy for fixing the hyper-parameters of the solver (tolerance on subproblems ϵ^k, step-sizes μ_e and μ_i). We rely on BCL (see [4] and [29, Algorithm 17.4]) which has been proved to perform well in advanced optimization packages such as LANCELOT [5] and also in robotics for solving constrained optimal control problems [32, 7, 20]. For solving QPs, we propose to rely on the combination of BCL with the proximal method of multipliers from Section III-C.

The main idea underlying BCL consists in updating the dual variables y^k and z^k obtained from (15) only when the corresponding primal infeasibility (denoted by p^{k+1} hereafter) is small enough. More precisely, we use a second sequence of tolerances denoted by ϵ^k_{ext} (which we also tune within the BCL strategy) and update the dual variables only when $p^{k+1} \leq \epsilon^k_{ext}$, where p^{k+1} denotes the primal infeasibility as follows:

$$p^{k+1} := \max(\|Ax^{k+1}-b\|_\infty, \|[Cx^{k+1}-u]_+\|_\infty). \tag{17}$$

It remains to explain how the BCL strategy chooses appropriate values for the hyper-parameters ϵ^k, ϵ^k_{ext}, μ_i and μ_e. As for the update of the dual variables, it proceeds in two stages:

- **If** $p^{k+1} < \epsilon^k_{ext}$**:** the primal infeasibility is good enough, we thus keep the constraint penalization parameters as is.
- **Otherwise:** the primal infeasibility is too large, we thus increase quadratic penalizations terms on the constraints for the subsequent proximal subproblems (15).

Concerning the accuracy parameters ϵ^k and ϵ^k_{ext}, the update rules are more technical and the motivation underlying those choices is to ensure global convergence: an geometric-decay type update when primal infeasibility is good enough, and see [4, Lemma 4.1] for when the infeasibility is too large. The detailed strategy is summarized in Algorithm 1.

Remark 1 (Global convergence)**.** *The BCL strategy [4, Algorithm 1 and Algorithm 2] was originally developed using an augmented Lagrangian, as opposed to a proximal augmented Lagrangian in our case. As a consequence, some of the original convergence guarantees from [4] do not hold anymore. For this reason, we introduced a safeguard parameter* $k_{\max} \in \mathbb{N}$ *enforcing the algorithm to ultimately always accept the candidates multipliers from the PMM updates* (15)*. This simple trick allows to inherit some nice properties from PMM that include convergence under mild assumptions [36, Theorem 4].*

In the next section, we review the last missing piece in our approach: a method for approximately solving the intermediary proximal subproblems (15).

IV. Solving the proximal subproblems

For obtaining an approximate solution to (15), we introduce a primal-dual merit function, originating and extending Gill and Robinson's primal-dual augmented Lagrangian (PDAL) [11] to the case of inequality constraints. We show that associated semi-smooth Newton steps involve linear systems, whose structure is well-conditioned and similar to the one presented for the equality constrained case (10).

In the next subsections, we first review the classic primal PMM (IV-A), before providing details on the proposed primal-dual approach (IV-B).

A. A primal semi-smooth approach

One possible instantiation of (15)-(16) consists in maximizing the PAL analytically w.r.t. the dual variables (y,z) and

minimizing it approximately w.r.t. the primal variable x:

$$\left\{\begin{array}{ll} x^{k+1}\approx_{\epsilon^k} & \arg\min_x \Phi_\rho^k(x), \\ y^{k+1} = & y^k + \frac{1}{\mu_e}(Ax^{k+1}-b), \\ z^{k+1} = & [z^k + \frac{1}{\mu_i}(Cx^{k+1}-u)]_+, \end{array}\right. \tag{18}$$

where $\approx_{\epsilon^k}$ now stands for x^{k+1} to be an ϵ^k-approximate solution in the following sense:

$$\|\nabla_x \Phi_\rho^k(x^{k+1})\|_\infty \leq \epsilon^k. \tag{19}$$

The PAL function Φ_ρ^k is semi-smooth [27]. Hence its unique minimum can be found in finite time using a semi-smooth Newton method with exact line-search (see, e.g., convergence proof in [40, Theorem 3] and algorithm in [15, Section IV.C]). Practically speaking, the semi-smooth Newton method is initialized at $\hat{x}^{(0)} := x^k$ and generates a sequence $\hat{x}^{(1)}, \hat{x}^{(2)}, \ldots$ via the update rule:

$$\hat{x}^{(l+1)} = \hat{x}^{(l)} + \alpha^* dx, \tag{20}$$

where the step-size α^* is computed via an exact line-search:

$$\alpha^* := \underset{\alpha\geq 0}{\arg\min}\, \Phi_\rho^k(\hat{x}^{(l)} + \alpha dx), \tag{21}$$

(note that $\alpha \longrightarrow \Phi_\rho^k(\hat{x}^{(l)} + \alpha dx)$ is a continuous piecewise quadratic function with a finite number of breaking points), and where dx is found by solving a linear system of equations:

$$\begin{bmatrix} H+\rho I & A^T & C^T_{I_k(\hat{x}^{(l)})} \\ A & -\mu_e I & 0 \\ C_{I_k(\hat{x}^{(l)})} & 0 & -\mu_i I \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} = \begin{bmatrix} -\nabla_x \Phi_\rho^k(\hat{x}^{(l)}) \\ 0 \\ 0 \end{bmatrix}, \tag{22}$$

where $I_k(\hat{x}^{(l)}) := \{i \in [1, n_i] \,|\, C_i\hat{x}^{(l)} - u_i + z_i^k\mu_i \geq 0\}$ refers to the active set of the current subproblem, and $C_{I_k(\hat{x}^{(l)})}$ corresponds to a reduced version of the matrix C containing the active rows indexed by $\mathcal{I}_k(\hat{x}^{(l)})$.

This iterative process is repeated until reaching the accuracy requirement (19); that is, as soon as $\|\nabla\Phi_\rho^k(\hat{x}^{(l)})\|_\infty \leq \epsilon^k$ it outputs $x^{k+1} \leftarrow \hat{x}^{(l)}$ as an approximate solution.

One drawback of such strategy comes from the right hand side term $\nabla_x\Phi_\rho^k(x)$. Indeed, similarly to (5), $\nabla_x\Phi_\rho^k(x)$ gathers quadratics involving the constraint matrices A and C, which ill-conditioning impact the whole accuracy in linear system (22). It can lead to numerical saturation effects with ill-conditioned problems when high precision is required. For this reason, we have decided to investigate other merit functions in order to solve better conditioned linear systems. It leads to our primal-dual approach.

B. A primal-dual approach

In this section, we present a primal-dual approach to solve (15). We introduce a primal-dual merit function, originating and extending Gill and Robinson PDAL to the case of inequality constraints [11]. The associated semi-smooth Newton steps involve better-conditioned linear systems with a structure similar to the one presented in (10).

a) Optimality conditions: A triplet (x, y, z) is a solution to (11) if and only if [6]:

$$\left\{\begin{array}{l} Hx + g + \rho(x - x^k) + A^Ty + C^Tz = 0, \\ \mu_e y - (Ax - b + \mu_e y^k) = 0, \\ \mu_i z - [Cx - u + \mu_i z^k]_+ = 0. \end{array}\right. \tag{23}$$

Hence we define $r^k(x, y, z)$ as:

$$r^k(x,y,z) := \begin{bmatrix} Hx + g + \rho(x - x^k) + A^Ty + C^Tz \\ \mu_e y - (Ax - b + \mu_e y^k) \\ \mu_i z - [Cx - u + \mu_i z^k]_+ \end{bmatrix}. \tag{24}$$

b) Generalized primal-dual augmented Lagrangian: Re-injecting appropriately second and third equations of (23) in the first one, one can notice, that (23) is equivalent to (25):

$$\left\{\begin{array}{c} Hx + g + \rho(x - x^k) + A^T(\frac{1}{\mu_e}(Ax - b) + y^k))+ \\ C^T[\frac{1}{\mu_i}(Cx - u) + z^k]_+ + (Ax - b - \mu_e(y - y^k)) \\ +([Cx - u + \mu_i z^k]_+ - \mu_i z) = 0, \\ \mu_e y - (Ax - b + \mu_e y^k) = 0, \\ \mu_i z - [Cx - u + \mu_i z^k]_+ = 0. \end{array}\right. \tag{25}$$

Condition (25) correspond to KKT conditions for the following primal-dual merit function $\mathcal{M}_{\mu,\rho}^k$:

$$\begin{aligned} \mathcal{M}_{\mu,\rho}^k(x,y,z) := \Phi_\rho^k(x) &+ \frac{1}{2\mu_e}\|Ax - b - \mu_e(y - y^k)\|_2^2 \\ &+ \frac{1}{2\mu_i}\|[Cx - u + \mu_i z^k]_+ - \mu_i z\|_2^2. \end{aligned} \tag{26}$$

This merit function is strictly convex and continuously differentiable. Hence, as for Φ_ρ^k in (IV-A), its unique minimum can be found in finite time using a semi-smooth Newton method with exact line-search [40, Theorem 3]. Thanks to the equivalence between (23) and (25), r^k can consequently be used as a suitable stopping criterion for measuring solution required inexactness:

$$\|r^k(x^{k+1}, y^{k+1}, z^{k+1})\|_\infty \leq \epsilon^k. \tag{27}$$

Remark 2. *Gill and Robinson introduced a generalized PDAL function [11] $(x, y) \rightarrow \mathcal{G}_{\rho,\mu}^k(x, y)$ for first tackling equality constrained problems. In the presence of inequalities, this PDAL function can be framed in its equality constrained form introducing a slack variable s satisfying the new equality constraint:*

$$Cx - u - s = 0. \tag{28}$$

The minimization of $\mathcal{G}_{\rho,\mu}^k$ w.r.t. variables x, y, z and s commutes. Considering the problem structure and following ideas from [6], it can be shown that s and z can be directly deduced as functions of x or z. Consequently, the ordering of variables used to minimize $\mathcal{G}_{\rho,\mu}^k$ defines different merit functions which may inherit from useful features of $\mathcal{G}_{\rho,\mu}$ in order to build an iterative procedure for solving (15)*. For example, minimizing $\mathcal{G}_{\rho,\mu}^k$ w.r.t. the slack variable s and re-injecting its optimal value s (depending of x and z) in the PDAL allows obtaining the merit function used in QPDO solver [6]. Another possibility, consists in minimizing $\mathcal{G}_{\rho,\mu}^k$ w.r.t. z, y and then to s*

and to re-inject all optimal values found (which are functions of x). In the latter case one retrieves then the PAL merit function introduced first by Rockafellar [36]. In the sequel, in order to get better conditioned linear systems, we decide to follow the same strategy but to relax optimal y and z values found, which leads to the new merit function $\mathcal{M}^k_{\rho,\mu}$.

c) Primal-dual Newton semi-smooth steps: A semi-smooth Newton step applied to $\mathcal{M}^k_{\mu,\rho}$, and initiated at $(\hat{x}^{(0)}, \hat{y}^{(0)}, \hat{z}^{(0)}) = (x^k, y^k, z^k)$ involves finding for $l \geq 0$ $dw := (dx, dy, dz)$ such that:

$$\nabla^2 \mathcal{M}^k_{\mu,\rho}(\hat{x}^{(l)}, \hat{y}^{(l)}, \hat{z}^{(l)}) dw + \nabla \mathcal{M}^k_{\mu,\rho}(\hat{x}^{(l)}, \hat{y}^{(l)}, \hat{z}^{(l)}) = 0, \tag{29}$$

where $\nabla^2 \mathcal{M}^k_{\mu,\rho}(\hat{x}^{(l)}, \hat{y}^{(l)}, \hat{z}^{(l)})$ stands for an element of its generalized Hessian [35, section 23]. It reads equivalently:

$$\begin{cases} \begin{bmatrix} H + \rho I + \frac{1}{\mu_e} A^T A + \frac{1}{\mu_i} (\hat{C}^{(l)})^T \hat{C}^{(l)} & -A^T & -(\hat{C}^{(l)})^T \\ -A & \mu_e I & 0 \\ -\hat{C}^{(l)} & 0 & \mu_i I \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} \\ = \begin{bmatrix} -(\nabla_x \Phi_\rho(\hat{x}^{(l)}) + A^T(\frac{1}{\mu_e}(A\hat{x}^{(l)} - d) + y_k - \hat{y}^{(l)}) \\ + C^T([\frac{1}{\mu_i}(C\hat{x}^{(l)} - u) + z_k]_+ - \hat{z}^{(l)}) \\ A\hat{x}^{(l)} - d - \mu_e(\hat{y}^{(l)} - y_k) \\ [C\hat{x}^{(l)} - u + \mu_e z_k]_+ - \mu_i \hat{z}^{(l)} \end{bmatrix}, \end{cases} \tag{30}$$

where $I_k(x)$ is the active set of the current subproblem at x:

$$I_k(x) = \{i \in [1, n_i] | Cx - u + \mu_i z^k \geq 0\}, \tag{31}$$

and $\hat{C}^{(l)}$ is a short-hand for denoting the generalized Jacobian of $[Cx - u + \mu_i z^k]_+$ at $x^{(l)}$:

$$\hat{C}^{(l)}_i := \begin{cases} C_i & \text{if } (Cx^{(l)} - u + \mu_i z^k)_i \geq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{32}$$

where C_i denotes the ith row of C. Remarking that for inactive constraints, i.e., $i \notin I_k(\hat{x}^{(l)})$, we have:

$$dz_i = -(\hat{z}^{(l)})_i, \tag{33}$$

the linear system (30) can hence be equivalently formulated as:

$$\begin{cases} \begin{bmatrix} H + \rho I & A^T & C^T_{I_k(\hat{x}^{(l)})} \\ A & -\mu_e I & 0 \\ C_{I_k(\hat{x}^{(l)})} & 0 & -\mu_i I_{I_k(\hat{x}^{(l)})} \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz_{I_k(\hat{x}^{(l)})} \end{bmatrix} = \\ - \begin{bmatrix} H\hat{x}^{(l)} + g + \rho(\hat{x}^{(l)} - x^k) + A^T \hat{y}^{(l)} + C^T_{I_k(\hat{x}^{(l)})} \hat{z}^{(l)}_{I_k(\hat{x}^{(l)})} \\ A\hat{x}^{(l)} - d - \mu_e(\hat{y}^{(l)} - y_k) \\ ([C\hat{x}^{(l)} - u + \mu_i z_k]_+ - \mu_i \hat{z}^{(l)})_{I_k(\hat{x}^{(l)})} \end{bmatrix} \\ dz_{I^c_k(\hat{x}^{(l)}))} = -\hat{z}^{(l)}_{I^c_k(\hat{x}^{(l)}))}, \end{cases} \tag{34}$$

where $I^c_k(x)$ is defined as the set of inactive constraints at x. Eq. (34) has the same structure as (10) in the equality constrained case. The right hand side does not contain any term involving square matrices. The main differences come from the active set introduced by $\mathcal{M}^k_{\rho,\mu}$, which filters the only constraints that need to be taken into account in (34).

d) Primal-dual line-search procedure: Once a semi-smooth Newton step (dx, dy, dz) has been obtained, the exact line-search procedure consists in finding the unique α^* such that:

$$\alpha^* = \underset{\alpha \geq 0}{\arg\min} \mathcal{M}^k_{\rho,\mu}(\hat{x}^{(l)} + \alpha dx, \hat{y}^{(l)} + \alpha dy, \hat{z}^{(l)} + \alpha dz). \tag{35}$$

Similarly to (21) the function $\alpha \to \mathcal{M}^k_{\rho,\mu}(\hat{x}^{(l)} + \alpha dx, \hat{y}^{(l)} + \alpha dy, \hat{z}^{(l)} + \alpha dz)$ is a continuous piecewise quadratic function with a finite number of breaking points. Hence, one can compute α^* exactly. Finally, the primal-dual semi-smooth Newton method initiated at $(\hat{x}^{(0)}, \hat{y}^{(0)}, \hat{z}^{(0)}) = (x^k, y^k, z^k)$ generates a sequence $(\hat{x}^l, \hat{y}^l, \hat{z}^l)$ via the update rule:

$$\begin{aligned} \hat{x}^{(l+1)} &= \hat{x}^{(l)} + \alpha^* dx, \\ \hat{y}^{(l+1)} &= \hat{y}^{(l)} + \alpha^* dy, \\ \hat{z}^{(l+1)} &= \hat{z}^{(l)} + \alpha^* dz. \end{aligned}$$

V. Detailed approach

The complete ProxQP procedure is provided in Algorithm 1. It contains a few additional practical enhancements, which are now detailed.

Preconditioning. ProxQP contains a preconditioning strategy, enhancing the overall numerical stability and convergence of the optimization process. The preconditioner used in our current implementation is often referred to as the Ruiz equilibration [37]; see, e.g., [39, Algorithm 2].

Initialization. Motivated by the fact that equality constraints are always active at an optimal solution, we use the following initialization for primal and dual variables:

$$\begin{bmatrix} x^{\text{init}} \\ y^{\text{init}} \end{bmatrix} = \begin{bmatrix} H + \rho I & A^T \\ A & \mu_e I \end{bmatrix}^{-1} \begin{bmatrix} -g \\ b \end{bmatrix} \text{ and } z^{\text{init}} = 0, \tag{36}$$

which corresponds to the (primal-dual) solution to the corresponding QP where the inequality constraints were removed. For the same reason, the default initial penalization is larger for equality constraints than it is for inequality constraints. The choice $\mu_e = \mu_i/100$ appears to perform well.

Cold restarts. To solve hard QPs to high precision, we use an additional cold restart strategy. The idea consists in resetting the penalization parameters μ_e and μ_i as soon as both primal and dual feasibility appears to stall while the current values for μ_e and μ_i are smaller than a certain threshold.

VI. Results

, In this section, we detail the software implementation of Algorithm 1 and then benchmark the practical performance of this new solver by comparing it to existing state-of-the-art solvers on various problems ranging from randomly generated problems to the harder Maros-Mészàros QPs [24], which includes a few classic robotic problems.

Software implementation. Our solver, referred to as ProxQP, is implemented in C++ and is extensively rooted on the generic-purpose Eigen library [14] for linear algebra. The current implementation is tailored for dense matrix operations,

Algorithm 1: ProxQP

Inputs:
- initial states: x^0, y^0, z^0,
- initial parameters: ϵ_{ext}^0, ϵ^0, ϵ_{abs}, ρ, μ_e, $\mu_i > 0$
- hyper-parameters: $\mu_f < 1, \alpha_{bcl} \in (0,1), \beta_{bcl} \in (0,1)$, $k_{\max} \in \mathbb{N}$, $\mu_{i,\min}, \mu_{e,\min} > 0$.

Initialization:
- preconditioning (see Section V)
- optional initialization (see Section V) of x^0,y^0, z^0.

while *Stopping criterion* (2) *not satisfied* **do**
  Compute $(\hat{x}, \hat{y}, \hat{z})$ satisfying (16) (ϵ^k-approximation to proximal subproblem (15)) using Section IV-B;
  $x^{k+1} = \hat{x}$
  if $p^{k+1} < \epsilon_{ext}^k$ *OR* $k \geq k_{\max}$ **then**
    $\epsilon^{k+1} = \epsilon^k \mu_i$, $\epsilon_{ext}^{k+1} = \epsilon_{ext}^k \mu_i^{\beta_{bcl}}$
    $y^{k+1} = \hat{y}$, $z^{k+1} = \hat{z}$
  else
    $\mu_i \leftarrow \max(\mu_{i,\min}, \mu_f \mu_i)$
    $\mu_e \leftarrow \max(\mu_{e,\min}, \mu_f \mu_e)$
    $\epsilon^{k+1} = \epsilon^0 \mu_i$, $\epsilon_{ext}^{k+1} = \epsilon_{ext}^0 \mu_i^{\alpha_{bcl}}$
    $y^{k+1} = y^k$, $z^{k+1} = z^k$
  end
  $k \leftarrow k+1$
  Apply cold restart if conditions are met
end
Output: A (x^k, y^k, z^k) satisfying the ϵ_{abs}-approximation criterion (2) for problem (QP).

and leverages recent CPU architectures providing advanced vectorization mechanisms. Our new solver if freely available at https://github.com/Simple-Robotics/proxsuite and comes with an easy-to-use interface inspired from OSQP [39].

Additionally, we have developed a dedicated LDLT Cholesky factorization to explicitly account for the specific features of the proposed approach. In particular, this Cholesky factorization implements advanced update routines to efficiently accounting for the change of active sets when solving (22) and (22), while lowering the overall memory footprint to maximize the performances. The new LDLT Cholesky is freely available within the same repository https://github.com/Simple-Robotics/proxsuite and will be hopefully integrated in Eigen.

Contrary to other solvers such as OSQP, qpSWIFT or qpOASES, our current implementations only relies on dense linear algebra routines (we plan to deeply exploit the sparsity of sparse problems in future works). Overall, as highlighted by the subsequent results, ProxQP performs better than modern sparse solvers over relatively sparse problems ranging from small to medium size (up to $d = 1000$) while remaining competitive for larger dimensions.

Benchmark scenarios. We benchmark our implementation on different types of QPs: equality and inequality constrained QPs, degenerate QPs with only inequality constraints, non-strictly convex QPs with only inequalities, as well as hard QPs from the Maros-Mészàros dataset [24] with different levels of sparsity and dimensions. We also benchmark on typical QPs encountered in inverse kinematic and dynamic robotic problems. The benchmarks are available at https://github.com/Bambade/proxqp_benchmark with an easy-to-use interface inspired from the one proposed by OSQP [39].

Our tests were carried in the following conditions:
- The level of accuracy required for termination is set arbitrarily to $\epsilon_{abs} = 10^{-9}$ (see criterion in (2)) so that to show across all these experiments which solvers manage to be the most accurate, the fastest, and the most robust within a large variety of QPs.
- A time limit of 1000 seconds has also been set as in OSQP API [39] for benchmarks (if a problem is not solved by a solver at such precision its "solving" time is set to this value in order to draw test plots).
- For all the tests, we used the same set of parameters for Algorithm 1: $\mu_e = 10^{-3}$, $\mu_i = 0.1$, $\mu_f = 0.1$, $\rho = 10^{-6}$, $\alpha_{\text{bcl}} = 0.1$, $\beta_{\text{bcl}} = 0.9$, $\mu_{i,\min} = 10^{-8}$, $\mu_{e,\min} = 10^{-9}$. Finally, ϵ^0 and ϵ^0_{ext} follow the initialization procedure proposed in [29, Algorithm 17.4], the safeguard $k_{\max} = 10^6$ (not used in practice).

Remark 3 (Accuracy choice)**.** *Choosing a low ϵ_{abs} has a few non-negligible advantages when comparing solvers. Indeed, the fact a solver might not reach every desired level of accuracy (within the available finite precision limits) is typically due to either (i) an algorithm that has a (very) slow convergence, or (ii) somehow "inappropriate" underlying subroutines (including linear algebra ones) with a limited working precision/stability range, worsening the effect of finite precision. Hence, such low accuracy level reveals which solvers provide on the same time (i) an algorithm whose capabilities allow to reach high accuracy and (ii) a set of underlying numerical routines allowing to reach high precisions in reasonable times.*

Benchmark setup. Benchmarks are performed with a standard laptop equipped with a relatively old CPU (Core i5 - 5300U - 5th Generation @ 2,3 GHz processor) and, for some specific benchmarks, also with a more recent CPU (Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz) typically equipping clusters, illustrating potential gains of using a more recent architecture.

A. Random problems

Concerning random problems, three different levels of sparsity (0.15, 0.5, and 1) were used for generating the random matrices H, A and C in the spirit of the benchmark API from OSQP [39]. The dimensions of those problems were picked from $d = 10$ to $d = 1000$. For each set of parameters (sparsity levels and dimensions), we generated 5 problem instances with different random seeds and averaged the running time of each algorithm on 10 consecutive runs (to limit the impact of loading costs). The results are provided in Figure 1 using bar plots (including the median, the minimal and maximal execution timings). We also report base statistics in Table I.

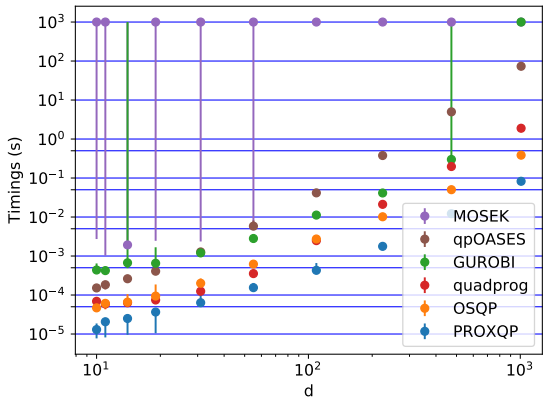

Fig. 1. Solving times for sparse equality and inequality constrained QPs (Section VI-A), using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. For each set of executions, the median is shown with a dot.

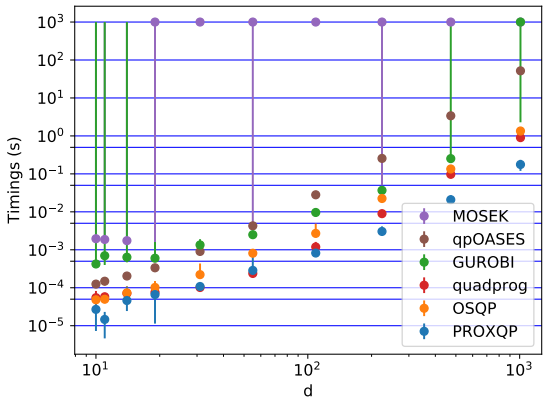

Fig. 2. Solving times for sparse degenerate pure inequality constrained QPs (Section VI-B), using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. For each set of executions, the median is shown with a dot.

	osqp	proxqp	gurobi	mosek	qpoases	quadprog
SGM [s]	4.6	1.0	1146.9	40891.4	305.6	18.8
FR (%)	0	0	16	80	0	0

TABLE I
SHIFTED GEOMETRIC MEANS (SGM) AND FAILURE RATES (FR) FOR SPARSE EQUALITY AND INEQUALITY CONSTRAINED QPS (SECTION VI-A).

In particular, Figure 1 shows that even in a sparse configuration (we recall that our solver uses a dense back-end), our solver is around 4 to 5 times faster than OSQP (the second best solver from our test-bed) for examples of dimensions $d \approx 50$ (which is representative of typical robotic applications). When dimension grows to $d \approx 1000$, the speed-up reaches almost an order of magnitude.

The failure rates observed in Table I for MOSEK and GUROBI solvers come from the fact they are not able to reach the desired precision $\epsilon_{\text{abs}} = 10^{-9}$: solutions proposed are not accurate enough. Consequently, it also impacts their geometric means.

B. Degenerate pure inequality-constrained problems

Figure 2 provides the results of our numerical experiments when generating pure inequality-constrained QPs where the matrix H is positive definite but for which LICQ conditions are no longer satisfied (by duplicating the constraints), a common type of degeneracy. We observe from Figure 2 that for problems of dimensions $d \approx 50$ our solver is about 3 times faster than OSQP (the second best solver in this set of experiments).

	osqp	proxqp	gurobi	mosek	qpoases	quadprog
SGM [s]	7.1	1.0	461.2	16065.0	118.5	3.7
FR (%)	0	0	14	76	0	0

TABLE II
SHIFTED GEOMETRIC MEANS (SGM) AND FAILURE RATES (FR) FOR SPARSE DEGENERATE PURE INEQUALITY CONSTRAINED QPS (SECTION VI-B).

	osqp	proxqp	gurobi	mosek	qpoases	quadprog
SGM [s]	2.0	1.0	917.2	6446.9	70.1	NA
FR (%)	0	0	34	72	0	100

TABLE III
SHIFTED GEOMETRIC MEANS (SGM) AND FAILURE RATES (FR) FOR SPARSE PURE INEQUALITY CONSTRAINED QPS WITH NON-STRICTLY POSITIVE DEFINITE HESSIAN (SECTION VI-C).

We report base statistics in Table II. The failure rates observed for MOSEK and GUROBI solvers come from the fact, again, they are not able reaching the desired precision $\epsilon_{\text{abs}} = 10^{-9}$, their outputted solutions being not accurate enough.

C. Non-strongly convex problems

As before, in the spirit of the OSQP [39], we generate random QPs for which the Hessian H is not strictly positive definite. We can see in Figure 3 that when matrices have 15% of sparsity, OSQP and ProxQP have a similar speed for $d \leq 200$. For higher dimension, we observe that ProxQP is approximately 1.8 times faster than OSQP. When sparsity is about 50%, one can see in Figure 4 ProxQP is about 1.8 to 2 times faster for $d \approx 50$. When $d \approx 1000$, ProxQP is about four times faster.

One can see on Figure 5 that when executed on a more modern computer (see Benchmark setup), performance gains are higher, i.e., for $d \geq 200$ our solver is 2 to 3 times faster than the second best solver (it was about about 1.8 times faster before).

We report base statistics in Table III. The failure rates observed for MOSEK and GUROBI solvers come from the fact, again, their solutions not being precise enough.

D. Maros-Mészàros problems

The Maros-Mészàros test set [24] is composed of 138 "hard" QPs. Most of them are sparse and ill-conditioned problems, and they contain up to 90597 variables and 180895 constraints. About 83% of the problems have a sparsity level

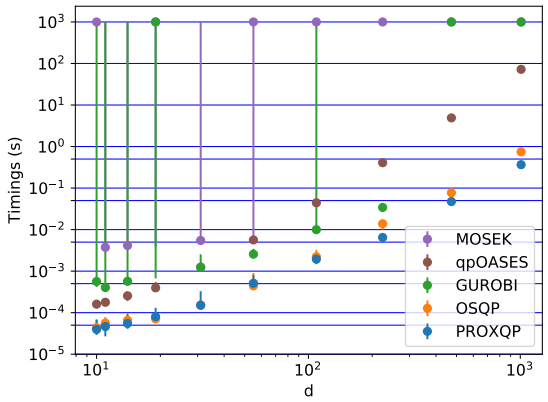

Fig. 3. Solving times for sparse pure inequality constrained QPs with non-strictly positive definite Hessian (Section VI-C), using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. For each set of executions, the median is shown with a dot.

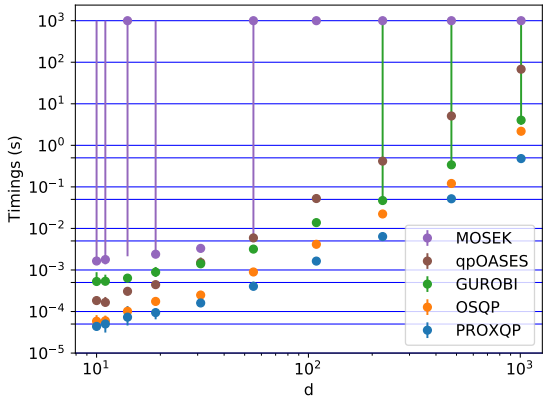

Fig. 4. Solving times for pure inequality constrained QPs with non-strictly positive definite Hessian and 50% of sparsity (Section VI-C), using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. For each set of executions, the median is shown with a dot.

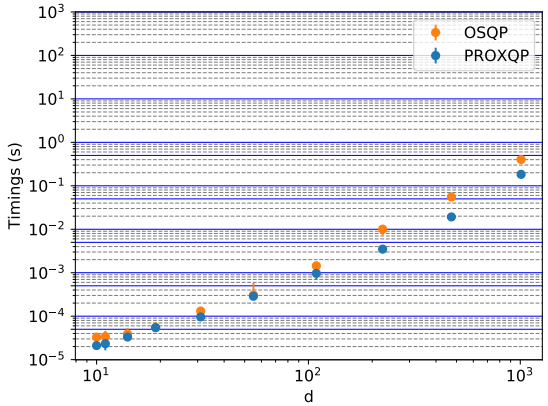

Fig. 5. Solving times for sparse pure inequality constrained QPs with non-strictly positive definite Hessian (Section VI-C), using an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz. For each set of executions, the median is shown with a dot.

	osqp	proxqp	gurobi	mosek	qpoases	quadprog
SGM [s]	13.4	1.0	47.6	164.5	4.7	96.4
FR (%)	32.3	4.8	58.1	83.9	14.5	72.6

TABLE IV
SHIFTED GEOMETRIC MEANS (SGM) AND FAILURE RATE (FR) FOR MAROS-MÉSZÀROS PROBLEMS (SECTION VI-D).

lower than 10% for the Hessians H, as well as for the constraint matrices A and C.

As we have currently implemented only a dense version of our solver, we restrict the benchmark to problems whose dimensions (constraints and variables) are below or equal to 10^3; that is, a subset of 62 problems of the Maros-Mészàros test set (about 45% of the set), for which two thirds have a sparsity level no larger than 20%.

On this set, we measure the ability of the different solvers to tackle those ill-conditioned problems to high accuracy within the predefined runtime limit (1000 seconds). We report a few statistics for each solver under consideration below, including the *failure rate* (FR), the *sifted geometric means* (SGM) as well as the *performance profile* (see details in, e.g., [39, 16]).

a) Failure rates: Failure rates mostly come from saturation effects (time limit reached mostly for OSQP, or outputted solutions being not precise enough for others, in some cases primal or dual infeasibility is detected [39, Section 5.1]). These saturations come at different precision levels. Concerning ProxQP, all proposed solutions are outputted within the time limit of 1000 seconds. However, for three of them, unscaling the scaled proposed solutions through reversed equilibration procedure [39, Section 5.1]) makes them finally close to $\epsilon_{\text{abs}} = 10^{-9}$ but just above the threshold (around $\approx 1.1 \times 10^{-9}$ for example, whereas the scaled solutions satisfy the scaled stopping criterion strictly below 10^{-9}). For other solvers, the situation can be close in some cases, but it has also been observed that saturation effects appears often sooner at higher precision levels ($\approx 10^{-7}$ for MOSEK or GUROBI solvers for example).

b) Shifted geometric means: Let $t_{s,p}$ denote the time required for solver s to solve problem p. The shifted geometric mean $\hat{t}_s$ of the runtimes for solver s on problem set $\mathcal{P}$ is

$$\begin{aligned} \hat{t}_s &:= |\mathcal{P}| \sqrt{\prod_{p \in \mathcal{P}} (t_{s,p} + \zeta)} - \zeta \\ &= e^{\frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \log(t_{s,p} + \zeta)} - \zeta. \end{aligned} \tag{37}$$

The second formulation is used in practice to prevent overflow when computing the product. In this paper, runtimes are expressed in seconds, and a shift of $\zeta = 10$ is used, as in the OSQP API for benchmarks [39]. As in [39], we employ the convention that when a solver s fails to solve a problem p (within the time limit of 1000 seconds), the corresponding $t_{s,p}$ is set to the time limit for the computation. We report these experimental results in Table IV for the Maros-Mészàros set.

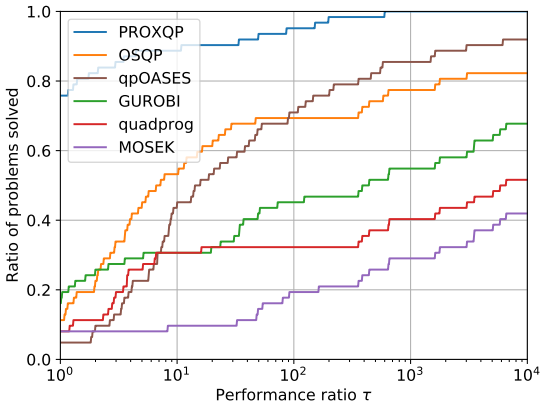

Fig. 6. Performance profiles on small to medium-sized Maros-Mészàros problems (see Section VI-D), using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. Target accuracy $\epsilon_{abs} = 10^{-9}$ (see criterion in (2)) and time limit set to 1000 seconds. The higher the better.

c) Performance profiles: We report performance profiles on Figure 6. Performance profiles correspond to the fraction of problems solved as a function of certain runtime (measured in terms of a multiple of the runtime of the fastest solver for that problem). More precisely, if $\mathcal{S}$ is the set of solvers tested:

$$r_{s,p} := \frac{t_{s,p}}{\min_{s\in\mathcal{S}} t_{s,p}}, \tag{38}$$

denotes the performance ratio for solver s with respect to problem p. The fraction of problems $q_s(\tau)$ solved by s within a multiple τ of the best runtime, is then given by

$$q_s(\tau) = \frac{1}{|\mathcal{P}|} \sum_{p\in\mathcal{P}, r_{s,p}\leq\tau} 1. \tag{39}$$

Figure 6 shows the plot of $q_s(\tau)$ curves for all the solvers s of our test-bed, when executed in our subset of Maros-Mészàros problems defined above. It turns out that our solver always depicts the best performance profile, with only 3 unsolved problems (see more details on it in paragraph VI-D0a), while other solvers appear to be less efficient and output more unsolved problems: either they reach the 10^3[s] time-out or are unable to satisfy the desired accuracy.

E. Inverse kinematics and dynamics

Finally, we have benchmarked ProxQP against OSQP (the second best solver of our previous tests bed) on the typical inverse kinematics and dynamics settings. We consider the task of controlling the center of mass of the TALOS humanoid robot [38] (which should remain within the convex supporting polygon), keeping the two feet on the ground while also moving both arms to reach two random target placements. We use the Pinocchio library [2] to compute the kinematic and dynamic quantities associated to each task. ProxQP solves the QP problems in $24 \pm 7\,\mu$s, while OSQP takes $167 \pm 93\,\mu$s. For the inverse dynamic task, ProxQP solves it in $25 \pm 6\,\mu$s while OSQP takes $441 \pm 193\,\mu$s. Such performances may lead to inverse kinematics or inverse dynamics controllers to be run at very high frequency, an important feature for torque-controlled robots [17] for instance.

VII. Conclusion

In this work, we have proposed a new algorithm for solving generic QPs together with its numerical C++ implementation, motivated by robotic applications. We notably propose to combine the bounded constraint Lagrangian (BCL) globalization strategy [4] with the solving of (primal-dual) Karush-Kuhn-Tucker conditions associated to QP. The intermediary proximal subproblems are solved via a primal semi-smooth Newton method, potentially initiated at an educated guess that allows to ultimately not requiring any Newton iteration when the iterates get close enough to an optimal solution. We highlight the numerical efficiency of our method on various sets of standard QPs ranging from randomly generated problems strongly inspired by the benchmark suite [39] to a subset of the Maros-Mészàros problems [24]. Our solver turns out to perform better than modern solvers of the literature on small to medium-size problems ($d \leq 10^3$), while remaining competitive for larger dimensions. As future work, we plan to extend our solver to also support the case of large-dimensional sparse problems for enabling its use in larger-scale problem setups ($d \gg 10^3$). This will make ProxQP a suitable, efficient and reliable QP solver to operate on modern optimization problems in robotics and beyond.

We also plan to generalize the exploitation of the primal-dual augmented Lagrangian techniques to the context of nonlinear problems (e.g. nonlinear optimal control of robotic systems for solving locomotion and manipulation tasks, etc.) following the approach proposed in [7] and recently extended in [20] and [21]. From a software perspective, we plan to extend our contribution to account for sparse and matrix-free methods for large QP problems ($d > 1000$).

Acknowledgments

This work was partly funded by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute), the European Research Council (grant SEQUOIA 724063) and the Louis Vuitton ENS Chair on Artificial Intelligence.

References

[1] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN 0521833787. URL https://web.stanford.edu/~boyd/cvxbook/.

[2] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard. The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *2019 IEEE/SICE International Symposium on System Integration (SII)*, pages 614–619. IEEE, 2019. URL https://ieeexplore.ieee.org/document/8700380.

[3] J. Carpentier, R. Budhiraja, and N. Mansard. Proximal and sparse resolution of constrained dynamic equations.

In *Robotics: Science and Systems 2021*, 2021. URL http://www.roboticsproceedings.org/rss17/p017.pdf.

[4] A. R. Conn, N. I. M. Gould, and Ph. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991. doi: 10.1137/0728030. URL https://doi.org/10.1137/0728030.

[5] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Lancelot a Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992. ISBN 9783662122112 3662122111. URL http://dx.doi.org/10.1007/978-3-662-12211-2.

[6] A. De Marchi. On a primal-dual Newton proximal method for convex quadratic programs. *Computational Optimization and Applications*, 81(2):369–395, 2022. ISSN 0926-6003, 1573-2894. doi: 10.1007/s10589-021-00342-y. URL https://link.springer.com/10.1007/s10589-021-00342-y.

[7] S. El Kazdadi, J. Carpentier, and J. Ponce. Equality Constrained Differential Dynamic Programming. In *2021 International Conference on Robotics and Automation (ICRA)*, 2021. URL https://hal.inria.fr/hal-03184203/file/equality-constrained-ddp.pdf.

[8] A. Escande, N. Mansard, and P.-B. Wieber. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028, 2014. URL https://gepettoweb.laas.fr/uploads/Publications/2014_escande_ijrr.pdf.

[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*, 6(4):327–363, 2014. URL http://mpc.zib.de/archive/2014/4/Ferreau2014_Article_QpOASESAParametricActive-setAl.pdf.

[10] E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software (TOMS)*, 29(1):58–81, 2003. URL https://pages.cs.wisc.edu/~swright/papers/p58-m_gertz.pdf.

[11] Ph. E. Gill and Daniel P. Robinson. A primal-dual augmented Lagrangian. *Computational Optimization and Applications*, 51(1):1–25, 2021. ISSN 0926-6003, 1573-2894. doi: 10.1007/s10589-010-9339-1. URL http://link.springer.com/10.1007/s10589-010-9339-1.

[12] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27:1–33, 1983. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.521.6352&rep=rep1&type=pdf.

[13] N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD user documentation. *GALAHAD*, 2017. URL https://dl.acm.org/doi/pdf/10.1145/962437.962438.

[14] G. Guennebaud, B. Jacob, et al. Eigen v3, 2010. URL http://eigen.tuxfamily.org.

[15] B. Hermans, A. Themelis, and P. Patrinos. QPALM: A Newton-type Proximal Augmented Lagrangian Method for Quadratic Programs. *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019. URL https://lirias.kuleuven.be/retrieve/544118.

[16] B. Hermans, A. Themelis, and P. Patrinos. QPALM: A Proximal Augmented Lagrangian Method for Nonconvex Quadratic Programs, 2021. URL https://link.springer.com/article/10.1007/s12532-022-00218-0.

[17] A. Herzog, N. Rotella, S. Mason, F. Grimminger, S. Schaal, and L. Righetti. Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. *Autonomous Robots*, 40(3):473–491, 2016. URL http://www.cs.cmu.edu/~cga/z/Herzog_AURO_2016.pdf.

[18] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969. URL https://link.springer.com/article/10.1007/BF00927673.

[19] B. Houska, H. J. Ferreau, and M. Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011. URL https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.493.2356&rep=rep1&type=pdf.

[20] W. Jallet, N. Mansard, and J. Carpentier. Implicit differential dynamic programming. In *2022 International Conference on Robotics and Automation (ICRA)*.

[21] W. Jallet, A. Bambade, N. Mansard, and J. Carpentier. Constrained differential dynamic programming: A primal-dual augmented lagrangian approach. 2022. URL https://hal.archives-ouvertes.fr/hal-03597630/document.

[22] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous robots*, 40(3):429–455, 2016. URL https://www.cs.cmu.edu/~cga/z/Kuindersma_AURO_2016.pdf.

[23] D. B. Leineweber, I. Bauer, H. G. Bock, and J. P. Schlöder. An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. part 1: theoretical aspects. *Computers & Chemical Engineering*, 27(2):157–166, 2003. URL https://www.sciencedirect.com/science/article/abs/pii/S0098135402001588.

[24] I. Maros and C. Mészaros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1-4):671–681, 1999. URL http://www.doc.ic.ac.uk/rr2000/DTR97-6.pdf.

[25] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2 (4):575–601, 1992.

[26] C. Mészáros. The bpmpd interior point solver for convex quadratic problems. *Optimization Methods and Software*, 11(1-4):431–449, 1999. URL https://www.tandfonline.com/doi/abs/10.1080/10556789908805758.

[27] R. Mifflin. Semismooth and semiconvex functions in constrained optimization. *Siam Journal on Control*, 15:957–972, 1977. URL https://web.archive.org/web/20170923012726id_/http://pure.iiasa.ac.at/524/1/RR-76-021.pdf.

[28] Mosek. MOSEK optserver documentation. *Mosek*, 2022. URL https://docs.mosek.com/9.3/opt-server.pdf.

[29] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer series in operations research and financial engineering. Springer, 2nd edition, 2006. URL https://www.math.uci.edu/~qnie/Publications/NumericalOptimization.pdf.

[30] Gurobi Optimization. GUROBI optimizer reference manual. *Gurobi Optimization*, 2020. URL https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/9.0/refman.pdf.

[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*, 4(4):3355–3362, 2019. URL https://ieeexplore.ieee.org/abstract/document/8754693.

[32] B. Plancher, Z. Manchester, and S. Kuindersma. Constrained unscented dynamic programming. In *2017 International Conference on Intelligent Robots and Systems (IROS)*, pages 5674–5680, 2017. URL https://agile.seas.harvard.edu/files/agile/files/constrained-udp.pdf.

[33] M. J. D. Powell. A method for nonlinear constraints in minimization problems. *Optimization*, pages 283–298, 1969. URL https://www.semanticscholar.org/paper/A-method-for-nonlinear-constraints-in-minimization-Powell/192818e804f5b014dcf4d678795856594fb969b8.

[34] S. Redon, A. Kheddar, and S. Coquillart. Gauss least constraints principle and rigid body simulations. In *2002 International Conference on Robotics and Automation (ICRA)*, volume 1, pages 517–522. IEEE, 2002. URL https://hal.archives-ouvertes.fr/hal-01147672/document.

[35] R. T. Rockafellar. *Convex analysis*. Princeton Mathematical Series. Princeton University Press, Princeton, N. J., 1970. URL https://www.degruyter.com/document/doi/10.1515/9781400873173/html.

[36] R. T. Rockafellar. Augmented Lagrangians and Applications of the Proximal Point Algorithm in Convex Programming. *Mathematics of Operations Research*, 1(2):97–116, 1976. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.298.6206&rep=rep1&type=pdf.

[37] D. Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical report, Rutherford Appleton Laboratory, 2001. URL https://cds.cern.ch/record/585592/files/CM-P00040415.pdf.

[38] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, A. Del Prete, P. Souères, N. Mansard, F. Lamiraux, J-P. Laumond, et al. Talos: A new humanoid research platform targeted for industrial applications. In *IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017. URL https://homepages.laas.fr/ostasse/drupal/sites/homepages.laas.fr.ostasse/files/ICHR17_0084_MS_0.pdf.

[39] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12 (4):637–672, 2020. URL https://web.stanford.edu/~boyd/papers/pdf/osqp.pdf.

[40] J. Sun. On piecewise quadratic Newton and trust region problems. *Mathematical Programming*, 76:451–467, 1997. URL https://link.springer.com/article/10.1007/BF02614393.

[41] Y. Tassa, N. Mansard, and E. Todorov. Control-limited differential dynamic programming. In *2014 International Conference on Robotics and Automation (ICRA)*, pages 1168–1175, 2014. URL https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.648.5032&rep=rep1&type=pdf.

[42] P.-B. Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *2006 International Conference on Humanoid Robots*, pages 137–142. IEEE, 2006. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.411.6286&rep=rep1&type=pdf.

[43] S. J. Wright. Primal-dual interior-point methods. *SIAM*, 1997. URL https://archive.siam.org/books/swright/.

[44] A. Wächter and L.T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. URL http://users.iems.northwestern.edu/~4er/MehrotraNomination/Ref2ImplPrimalDualInterior.pdf.