Les Cahiers du GERAD ISSN: 0711–2440

RipQP: A multi-precision regularized predictor-corrector method for convex quadratic optimization

G. Leconte, D. Orban

G–2021–03

Janvier 2021
Revised: July 2024

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2021
– Bibliothèque et Archives Canada, 2021

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2021
– Library and Archives Canada, 2021

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

RipQP: A multi-precision regularized predictor-corrector method for convex quadratic optimization

Geoffroy Leconte

Dominique Orban

GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Qc), Canada, H3T 1J4

geoffroy.leconte@polymtl.ca
dominique.orban@gerad.ca

Janvier 2021
Revised: July 2024
Les Cahiers du GERAD
G–2021–03

Abstract : We describe the implementation of RipQP, an interior-point algorithm for convex quadratic optimization. Our Julia implementation is open source, and accommodates computations in multiple floating-point systems. In particular, it can be initialized in a low-precision system, such as Float32, as a form of warm start, and gradually transition through higher-precision systems until it reaches the prescribed accuracy. On platforms with hardware for various floating-point systems, our strategy results in savings in terms of time, number of normalized iterations, and energy expended during the solve. When we only dispose of double-precision hardware and we want to solve problems in a higher floating-point system such as quadruple precision, RipQP can perform some operations in double precision, while still maintaining satisfying stopping criteria.

Keywords : Interior-point methods, convex quadratic optimization, multi-precision

Résumé : Nous présentons RipQP, un algorithme de points intérieurs pour l'optimisation quadratique convexe écrit en Julia, libre de droit, dont le code source est libre d'accès, et qui est capable d'effectuer des opérations dans plusieurs précisions de calcul. En particulier, RipQP peut être initialisé dans une précision de calcul faible telle que la simple précision, pour ensuite transitionner graduellement vers des précisions de calcul plus élevées. Sur des plateformes pouvant effectuer nativement des calculs dans plusieurs systèmes en virgule flottante, notre méthode permet d'économiser du temps de calcul et de réduire l'énergie émise pour la résolution du problème. Si la plateforme utilisée dispose uniquement de la double précision de manière native, et que nous souhaitons résoudre des problèmes dans une précision plus élevée telle que la quadruple précision, RipQP peut effectuer certaines opérations en double précision, tout en maintenant des critères d'arrêt satisfaisants.

Acknowledgements: Research of G. Leconte is supported by an IVADO Undergraduate Research Scholarship. Research of D. Orban is partially supported by an NSERC Discovery Grant.

1 Introduction

We describe an efficient implementation of an interior-point algorithm named RipQP for the convex quadratic problem

$$\underset{x}{\text{minimize}}\ c^T x + \tfrac{1}{2} x^T Q x \quad \text{subject to}\ Ax = b,\ \ell \le x \le u, \tag{1}$$

where $c \in \mathbb{R}^n$, $Q = Q^T \in \mathbb{R}^{n\times n}$ is positive semi-definite, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $\ell \in \mathbb{R}^n$, $u \in \mathbb{R}^n$, and inequalities are understood elementwise. Although there already exist several commercial and freely-available implementation of this type of method, ours has a number of distinguishing features, including:

1. treatment of free variables, weak convexity and rank-deficient constraints by way of exact primal-dual regularization (Friedlander and Orban, 2012);
2. effective scaling strategy based on that of Ruiz (2001);
3. our algorithm is entirely implemented in the Julia language (Bezanson et al., 2017) with no compiled or operating-system-specific dependencies, and therefore is portable, is compiled on the fly, and does not require a separate compiler;
4. a generic implementation that can be executed in any floating-point arithmetic supported by Julia;
5. a multi-precision heuristic attempting to save time, computations and energy on appropriate platforms;
6. double precision efficiency competitive with that of the commercial libraries CPLEX (IBM ILOG CPLEX Optimization Studio), Gurobi (Gurobi Optimization, LLC, 2023) and Xpress (FICO Xpress Optimization);
7. modular linear algebra for solving a linear system that is often considered as the most expensive operation in interior-point algorithms;
8. our software is open source and freely available from `https://github.com/JuliaSmoothOptimizers/RipQP.jl`.

For simplicity of exposition, we assume that all elements of ℓ and u are finite. Accommodating absence of bounds is easily achieved by deleting rows and/or columns from matrices involving those bounds. If linear inequality constraints $b_\ell \le Ax \le b_u$ are present, our implementation adds slack variables $Ax - t = 0$ and the bounds $b_\ell \le t \le b_u$ to recover the form (1).

The dual of (1) is

$$\begin{aligned} &\underset{x,y,s_\ell,s_u}{\text{maximize}}\ y^T b + s_\ell^T \ell - s_u^T u - \tfrac{1}{2} x^T Q x \\ &\text{subject to}\ -Qx + A^T y + s_\ell - s_u = c,\ s_\ell \ge 0,\ s_u \ge 0, \end{aligned} \tag{2}$$

where $y \in \mathbb{R}^m$, $s_\ell \in \mathbb{R}^n$ and $s_u \in \mathbb{R}^n$ are vectors of Lagrange multipliers associated with the equality constraints, lower bounds, and upper bounds of (1), respectively. A solution (x, y, s_ℓ, s_u) of (1)–(2), when one exists, satisfies the Karush-Kuhn-Tucker (KKT) conditions

$$F(x, y, s_\ell, s_u) := \begin{bmatrix} -Qx + A^T y + s_\ell - s_u - c \\ Ax - b \\ S_\ell (x - \ell) \\ S_u (u - x) \end{bmatrix} = 0, \tag{3a}$$

$$s_\ell \ge 0, \quad s_u \ge 0, \quad \ell \le x \le u, \tag{3b}$$

where $S_\ell := \operatorname{diag}(s_\ell)$ and $S_u := \operatorname{diag}(s_u)$. The conditions (3) can be generalized to problems with infinite bounds in ℓ and u by removing the associated components of x and columns of S_ℓ and/or S_u.

The main computational cost of RIPQP consists in solving the Newton system

$$\begin{bmatrix} -Q & A^T & I & -I \\ A & 0 & 0 & 0 \\ S_\ell & 0 & X-L & 0 \\ S_u & 0 & 0 & X-U \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s_\ell \\ \Delta s_u \end{bmatrix} = - \begin{bmatrix} r_c \\ r_b \\ \xi_\ell \\ \xi_u \end{bmatrix} \tag{4}$$

at each iteration, where $X := \text{diag}(x)$, $L := \text{diag}(\ell)$, and $U := \text{diag}(u)$. This system can be symmetrized, and some variables may be eliminated, in order to keep the computation time reasonable. One of the modular advantages of RIPQP is that it allows the user to choose an equivalent formulation of (4) to solve.

A reformulation of (4) that is often used in convex quadratic optimization is

$$\begin{bmatrix} -(Q+D) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}, \tag{5}$$

where $D = S_\ell (X-L)^{-1} + S_u (U-X)^{-1}$. This system is referred as the K_2 formulation (Greif et al., 2014; Ghannad et al., 2021), and may be numerically unstable in the sense that its condition number grows without bound as convergence occurs (Greif et al., 2014). If A is rank deficient, (4) and (5) are singular, which complicates the computation of the search direction. As a remedy, we use primal-dual regularization such as that of Friedlander and Orban (2012):

$$\begin{bmatrix} -(Q+\rho I) & A^T & I & -I \\ A & \delta I & 0 & 0 \\ S_\ell & 0 & X-L & 0 \\ S_u & 0 & 0 & X-U \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s_\ell \\ \Delta s_u \end{bmatrix} = - \begin{bmatrix} r_c \\ r_b \\ \xi_\ell \\ \xi_u \end{bmatrix}, \tag{K3}$$

where the iteration-dependent positive regularization parameters ρ and δ have small values (typically close to the square root of machine precision). The system (5) then becomes

$$\begin{bmatrix} -(Q+D+\rho I) & A^T \\ A & \delta I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}. \tag{K2}$$

This system is symmetric quasi-definite (SQD), and may be solved in several ways (Orban and Arioli, 2017). RIPQP implements two of the most efficient methods used to solve (K2): a signed Cholesky factorization and a preconditioned Krylov method.

Most implementations used to solve (1) perform computations entirely in double floating-point arithmetic. On platforms with hardware facilities to work in a lower precision, such as single precision floating-point arithmetic, lower-precision computations are performed faster and expend less energy than higher-precision computations, as observed by Haidar et al. (2020). Moreover, given that some applications require solving (1) to high accuracy, or that some problems are inherently difficult to solve numerically, it can be necessary to perform computations in higher floating-point systems (Ma et al., 2017).

Similarly to the Julia interior-point solver for linear problems Tulip.jl from Tanneau et al. (2021), our implementation can be employed in any floating-point system supported by Julia, including half (on moderate-size problems), single, double and quadruple precision, with precision-dependent stopping criteria. One of its interesting features is that it is also able to solve problems by starting in a low precision and transitioning to a higher precision as the algorithm gets closer to a solution. This strategy has similarities with the quadruple precision simplex solver DQQ from Ma et al. (2017), but is adapted to an interior-point solver. In addition, when solving (K2) in a precision p, RIPQP can perform costly operations in a lower precision $q < p$, while maintaining an accuracy worthy of precision p.

The Julia programming language is convenient for writing generic type-stable algorithms thanks to its multiple dispatch features, and allowed us to write efficient type-stable functions that can work with

any floating-point system. Indeed, a specialized version of each function is compiled according to the type of its arguments, so that, given the current precision of the data used to solve our problem, the function compiled with the corresponding floating-point system is used. For example, when solving (1) in precision p with some initial iterations in precision $q < p$, the functions used in both precisions are compiled in precision p and q.

2 Two different methods to solve the convex quadratic problem

Interior-point methods require solving one or several linear systems such as (4) at each iteration to find a search direction $(\Delta x, \Delta y, \Delta s_\ell, \Delta s_u)$. If several systems need to be solved per iteration, they have the same matrix, and a factorization approach is efficient as the factorization can be computed once and reused. RipQP provides two different methods to compute steps, which should be chosen by taking into account the solver used to solve the linear systems.

2.1 The predictor-corrector method

The first method implemented within RipQP is the classic predictor-corrector method of Mehrotra (1992).

We define the primal and dual residuals as

$$r_b := Ax - b, \tag{6a}$$
$$r_c := -Qx + A^T y + s_\ell - s_u - c. \tag{6b}$$

The predictor step, or affine-scaling step, solves

$$\begin{bmatrix} -(Q+\rho I) & A^T & I & -I \\ A & \delta I & 0 & 0 \\ S_\ell & 0 & X-L & 0 \\ -S_u & 0 & 0 & U-X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{aff}} \\ \Delta y^{\text{aff}} \\ \Delta {s_\ell}^{\text{aff}} \\ \Delta {s_u}^{\text{aff}} \end{bmatrix} = - \begin{bmatrix} r_c \\ r_b \\ S_\ell(x-\ell) \\ S_u(u-x) \end{bmatrix}. \tag{7}$$

We then compute a steplength along the primal and dual search directions that preserves sufficient strict feasibility:

$$\alpha_{\text{pri}}^{\text{aff}} = \arg\max\{\alpha \in (0,1] \mid x + \alpha \Delta x^{\text{aff}} \in [\ell + \tau(x-\ell), u - \tau(u-x)]\} \tag{8a}$$
$$\alpha_{\text{dual}}^{\text{aff}} = \arg\max\{\alpha \in (0,1] \mid s_\ell + \alpha \Delta {s_\ell}^{\text{aff}} \geq \tau s_\ell \text{ and } s_u + \alpha \Delta {s_u}^{\text{aff}} \geq \tau s_u\} \tag{8b}$$

for a user-defined $\tau \in (0,1)$.

The duality gap is defined as

$$\mu = \frac{s_\ell^T(x-\ell) + s_u^T(u-x)}{2n}, \tag{9}$$

and must converge to zero to satisfy (3).

The second step is the centering/corrector step, and is achieved by computing the duality gap that would result from the predictor step, and setting the centering parameter σ according to the heuristic of Mehrotra (1992):

$$x^{\text{aff}} := x + \alpha_{\text{pri}}^{\text{aff}} \Delta x^{\text{aff}} \tag{10a}$$
$$(y^{\text{aff}}, s_\ell^{\text{aff}}, s_u^{\text{aff}}) := (y, s_\ell, s_u) + \alpha_{\text{dual}}^{\text{aff}} (\Delta y^{\text{aff}}, \Delta s_\ell^{\text{aff}}, \Delta s_u^{\text{aff}}) \tag{10b}$$
$$\sigma := (\mu^{\text{aff}}/\mu)^3, \tag{10c}$$

where μ^{aff} is the duality measure of $(x^{\text{aff}}, y^{\text{aff}}, s_\ell^{\text{aff}}, s_u^{\text{aff}})$. The centering/corrector is then computed by solving

$$\begin{bmatrix} -(Q+\rho I) & A^T & I & -I \\ A & \delta I & 0 & 0 \\ S_\ell & 0 & X-L & 0 \\ -S_u & 0 & 0 & U-X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{cc}} \\ \Delta y^{\text{cc}} \\ \Delta s_\ell^{\text{cc}} \\ \Delta s_u^{\text{cc}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sigma\mu e - \Delta S_\ell^{\text{aff}^T} \Delta x^{\text{aff}} \\ \sigma\mu e + \Delta S_u^{\text{aff}^T} \Delta x^{\text{aff}} \end{bmatrix}. \tag{11}$$

The overall search direction is the combined step

$$(\Delta x, \Delta y, \Delta s_\ell, \Delta s_u) = (\Delta x, \Delta y, \Delta s_\ell, \Delta s_u)^{\text{aff}} + (\Delta x, \Delta y, \Delta s_\ell, \Delta s_u)^{\text{cc}}. \tag{12}$$

2.2 The infeasible path-following method

The second method implemented within RipQP is the infeasible path-following method, similar to that of Kojima et al. (1993). The search direction is computed by solving the linear system

$$\begin{bmatrix} -(Q+\rho I) & A^T & I & -I \\ A & \delta I & 0 & 0 \\ S_\ell & 0 & X-L & 0 \\ -S_u & 0 & 0 & U-X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s_\ell \\ \Delta s_u \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ \sigma\mu e - S_\ell(x-\ell) \\ \sigma\mu e + S_u(u-x) \end{bmatrix}. \tag{13}$$

The centering parameter σ is computed using the heuristic from Vanderbei and Shanno (1999):

$$\xi = \min\left(\frac{\min_i s_{\ell i}(x_i - \ell_i)}{s_\ell^T(x-\ell)}, \frac{\min_i s_{ui}(u_i - x_i)}{s_u^T(u-x)} \right) \tag{14a}$$

$$\sigma = \gamma \min\left((1-r)\frac{1-\xi}{\xi}, 2 \right)^3, \tag{14b}$$

where we set $r = 0.999$ and $\gamma = 0.05$. The objective of (14) is to follow the central path by trying to preserve uniformity of the complementarity products $s_{\ell,i}(x_i - \ell_i)$ and $s_{u,i}(u_i - x_i)$, $i \in \{1, \ldots, n\}$, while reducing μ. We do not further expand on this matter, since numerous details about the central path already exist in the litterature (Wright, 1997), (Nocedal and Wright, 2006, chapter 14).

3 The different formulations to solve at each iteration

The Newton system (K3), though its condition number is bounded under strict complementarity, is rarely solved directly because of its size and of the fact that it is not symmetric. However, it can be easily symmetrized by multiplying the last two block rows by S_ℓ^{-1} and S_u^{-1}, respectively, to get the K_{3S} system, which has the drawback of having an unbounded condition number. Forsgren, Greif et al. (2014) and Ghannad et al. (2021) use a symmetric variant of K_{3S} with a bounded condition number.

The system that is often favored to solve convex quadratic problems is the augmented system (K2), which is sometimes referred to as the augmented system. Ghannad et al. (2021) also present a variant of (K2) with bounded condition number:

$$\begin{bmatrix} -X_{\ell u}^{1/2}(Q+D+\rho I)X_{\ell u}^{1/2} & X_{\ell u}^{1/2}A^T \\ AX_{\ell u}^{1/2} & \delta I \end{bmatrix} \begin{bmatrix} X_{\ell u}^{-1/2}\Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} X_{\ell u}^{1/2} f_x \\ f_y \end{bmatrix}, \tag{K2.5}$$

where $X_{\ell u} = (X-L)(U-X)$. The system (K2.5) can be viewed as a form of diagonal scaling of (K2).

When solving linear problems, the normal equations (K1) are often used:

$$(A(Q+D+\rho I)^{-1}A^T + \delta I)\Delta y = f_y + A(Q+D+\rho I)^{-1} f_x, \tag{K1}$$

because of their smaller size and their symmetric positive definiteness. All the formulations presented in this section are implemented in RipQP to be used with a Krylov method, but we only present results with (K2) in this paper, since it is smaller than K_3, K_{3S}, $K_{3.5}$, and because (K1) would require factorizing $Q + D + \rho I$ at each interior-point iteration, which would be costly for general Q. Even though (K2.5) is usable within RipQP with a factorization algorithm and with Krylov methods, we did not see significant improvements with factorization algorithms compared to (K2), and we decided not to include it in our experiments.

4 A multi-precision algorithm

Thanks to the benefits of Julia's multiple dispatch, RipQP is able to solve a problem in any floating-point system given that the stopping criteria are set accordingly to the precision of solve. We usually use tolerances close to or greater than the machine epsilon of the required precision (our stopping criteria are described in the beginning of Section 5.5). However, half-precision (and lower) solves may not be robust since there can quickly be some overflow errors.

On compatible hardware, Haidar et al. (2020) indicate that, when dividing by a factor 2 the precision of the floating-point system, operations can be up to 2 times faster and consume up to 4 times less energy. That is why we might wish to reduce the cost of performing operations that do not require high accuracy by using multi-precision techniques. RipQP uses several strategies to benefit from computations in lower-precision systems.

4.1 Warm-start in a lower-precision floating-point system

As the interior-point iterates approach a solution, the condition number of (K2) becomes increasingly high, which leads to inaccurate solves. That is why we might want to lower the precision of the initialization and early iterations of RipQP, since there are the easiest to compute. Once numerical issues are encountered, usually when performing the factorization of an ill-conditionned matrix, or when some residuals of the interior-point algorithm are sufficiently small, RipQP converts the data of the interior-point algorithm to a higher floating-point system to solve the systems with large condition number. This mechanism is described in Algorithm 1 The criteria for increasing the precision are described in Section 5.5.

Algorithm 1 Solving a quadratic problem in precision p with RipQP warm-started in a lower precision q (multi-precision mode).

Require: Q, A, c, c_0, b, ℓ, u defined in (1) in precision p.
1: Duplicate Q, A, c, c_0, b, ℓ, u in precision q.
2: Solve (1) in precision q until a relaxed stopping criterion is reached.
3: Convert the current point (x, y, s_ℓ, s_u) and all the necessary storage to precision p.
4: Solve (1) in precision p, bypassing the initialization procedure by choosing (x, y, s_ℓ, s_u) as a starting point.

4.2 Solving systems with a Krylov method preconditioned by a lower-precision factorization

Another way to perform operations in a lower floating-point system that is implemented within RipQP is to solve (K2) with a Krylov method preconditioned by a factorization in a lower precision. Algorithm 2 is a simplified version of that of Amestoy et al. (2023), used in two different floating-point systems, and is similar to that of Arioli and Duff (2008). This can be seen as a form of iterative refinement (Arioli et al., 1989). It consists of solving (K2) in precision p with the (approximate) signed Cholesky factorization in precision $q < p$:

$$K_2 \approx L_2 D_2 L_2^T, \tag{15}$$

where L_2 is lower triangular with a diagonal of ones, and D_2 is diagonal. Since we use a factorization in precision $q < p$ to precondition the Krylov solver in precision p, this preconditioner is expected to be cheap enough that the total cost of this method is lower than the factorization-only solve in precision p.

Algorithm 2 Solve (K2) in precision p preconditioned by a signed Cholesky factorization in precision q, where $q < p$.

1: Compute a signed Cholesky factorization of (K2) in precision q using the dynamic regularization procedure from Altman and Gondzio (1999).
2: Solve (K2) with the computed signed Cholesky factorization in precision q to get an initial solution $\begin{bmatrix}\Delta x_0 & \Delta y_0\end{bmatrix}^T$.
3: Solve (K2) in precision p using a Krylov method starting from $\begin{bmatrix}\Delta x_0 & \Delta y_0\end{bmatrix}^T$, and preconditioned by the computed signed Cholesky factorization in precision q.

Sometimes, the regularization parameters ρ and δ in (K2) at a given interior-point iteration are too small to be able to compute the factorization in precision q without numerical issues (zero elements in the diagonal of D). To avoid these issues, Saunders and Tomlin (1996) mention that we should have $\rho\delta \gg \epsilon_M$ for linear problems, where ϵ_M is the machine epsilon relative to the precision of the factorization. That is why we use the dynamic regularization procedure described by Altman and Gondzio (1999) to avoid these numerical issues. When the factorization computes a $D_{2i,i}$ whose absolute value is too close to zero, the procedure updates $D_{2i,i} \leftarrow D_{2i,i} \pm \epsilon_f$, where ϵ_f is a small parameter, usually close to the square root of the machine precision q. The sign of the update is determined by the index i of the pivot. n first pivots are negative, and m pivots are positive. The dynamic regularization acts as a second form of regularization in precision q, but it is only used on pivots that are close to zero, which has minimal effect on the matrix K_2.

It is also possible to scale K_2 so that its rows and columns have an infinity norm of 1, using the algorithm of Ruiz (2001). This scaling is employed by Amestoy et al. (2021) to prevent overflow issues when factorizing a matrix in half-precision.

RipQP is also able to use a limited-memory signed Cholesky factorization such as that of Orban (2015). This limited-memory factorization can be computed in a lower precision system so that it is usable with the multi-precision strategy, and allows to reduce even more the cost of computing the preconditioner, at the expense of performing potentially more Krylov iterations.

4.3 Combining approaches

Thanks to the modular nature of RipQP, it is possible to combine the techniques presented in Section 4.1 and Section 4.2. This is achieved by using up to three different *SolverParams*, a type used to define the algorithm used to solve the interior-point system. Each *SolverParams* enables the user to choose the floating-point system used for the solve, as well as parameters specific to the solver used. For exemple, it is possible to create a *SolverParams* that instructs RipQP to solve the interior-point system with GMRES used on (K2) in double precision with a signed Cholesky preconditioner in single precision. Any combination of up to three *SolverParams* can be used within Algorithm 3, as long as a conversion functions between adjacent *SolverParams* is implemented.

Algorithm 3 Solving a *QuadraticModel* using three different *SolverParams*. Each *SolverParams* may be in a different floating-point system, given that the floating-point precision of the algorithm increases or stays the same when transitioning from one *SolverParams* to another.

Require: a *QuadraticModel* qm.
1: Solve qm using the *SolverParams* sp until some criteria tol_1 are satisfied, or something fails in the solve,
2: get the current point $x_1, y_1, s_{\ell,1}, s_{u,1}$ and convert the floating-point system of the data used for the solves if needed,
3: Solve qm using the *SolverParams* $sp2$ starting from $x_1, y_1, s_{\ell,1}, s_{u,1}$, until some criteria tol_2 are satisfied, or something fails in the solve,
4: get the current point $x_2, y_2, s_{\ell,2}, s_{u,2}$ and convert the floating-point system of the data used for the solves if needed,
5: Solve qm using the *SolverParams* $sp3$ starting from $x_2, y_2, s_{\ell,2}, s_{u,2}$, until the global criteria are satisfied.

```
stats = ripqp(
qm, # QuadraticModel in Double64 (quadruple precision with DoubleFloats.
    jl)
mode = :multi,
# solve K2 in Float64 with GMRES and Float64 preconditioner
sp = K2KrylovParams(
kmethod = :gmres,
preconditioner = LDL(T = Float64, pos = :R),
ρ_min = 1.0e-8,
δ_min = 1.0e-8,
mem = 50,
itmax = 50,
),
# solve K2 in Double64 with GMRES and Float64 preconditioner
sp2 = K2KrylovParams{Double64}(
kmethod = :gmres,
preconditioner = LDL(T = Float64, pos = :R),
ρ_min = Double64(1.0e-12),
δ_min = Double64(1.0e-12),
mem = 20,
itmax = 20,
),
# solve K2 in Double64 with LDL factorization
sp3 = K2LDLParams{Double64}(
ρ_min = Double64(1.0e-15),
δ_min = Double64(1.0e-15),
),
)
```

Figure 1: Example of configuration of RipQP to solve a *QuadraticModel* in quadruple precision using three *SolverParams*. We reduce the regularization values as we use a more precise solver.

5 Implementation details

5.1 Modeling

RipQP uses the Julia package QuadraticModels.jl (Orban et al., 2020b) to define quadratic problems of the form

$$\underset{x}{\text{minimize}}\ c_0 + c^T x + \tfrac{1}{2} x^T Q x \quad \text{subject to}\ \ell con \leq Ax \leq ucon,\ \ell \leq x \leq u. \tag{16}$$

Internally, RipQP transforms the model to have the form (1) by adding slack variables. Once a QuadraticModel, the datatype used to represent (16), has been created, it can be passed directly to RipQP.

5.1.1 Sparse coordinate format

The default input type for Q and A in the presolve procedure of QuadraticModels.jl (detailed in Section 5.2) is the sparse coordinate format (COO), as implemented in the Julia package SparseMatricesCOO.jl. We chose this format because it is the format used in QPS files, a common storage type for linear and quadratic models. The QPS format is used to represent all the quadratic models that we tested in this paper, and can be read easily with the Julia package QPSReader.jl (Tanneau et al., 2020).

Once the presolve has completed, it is possible to transform A and Q to another format. For example, when using the pure-Julia signed Cholesky factorization LDLFactorizations.jl (Orban and contributors, 2020), we work with arrays in compressed sparse column (CSC) format.

5.1.2 Linear operators

It is also possible to create a QuadraticModel with Q and A defined as abstract linear operators (for instance, the linear operator K is defined by the function $x \mapsto Kx$), via the Julia package

LinearOperators.jl (Orban et al., 2020a). With this QuadraticModel, the interior-point method should be based on a Krylov method, since these methods only require operator-vector products. However, solving a quadratic problem using linear operators might be more difficult because Krylov methods often require an effective preconditioner to solve a linear system, especially in the last interior-point iterations. An improvement to this functionality would be to compute matrix-free preconditioners constructed from a linear operator.

5.2 Presolve and scaling

QuadraticModels.jl contains a rudimentary presolve procedure (currently in development) that implements some of the operations described by Gould and Toint (2004). It is also inspired by the package MathOptPresolve.jl, which contains presolve routines for linear problems. However, our implementation is adapted to quadratic problems, with the storage format required by our modeling API. Our presolve contains the operations described in Algorithm 4.

Algorithm 4 Presolve

1: Remove empty rows of A.
2: Remove singleton rows of A.
3: Remove linearly unconstrained variables (empty columns of A) that occur linearly in the objective function by fixing them at one of their bound, chosen by the sign of the corresponding component of c, and detect if the problem is unbounded.
4: Remove singleton columns of A whose associated variables have infinite bounds and occur linearly in the objective. This also removes the associated rows corresponding to the row indices of the nonzero elements of the removed columns.
5: Remove free rows of A (row i is free if $lcon_i = -\infty$ and $ucon_i = +\infty$).
6: Remove fixed variables.

Once the model has been presolved, RipQP scales it so that $\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix}$ has rows and columns with unit ℓ_∞-norm using the scaling algorithm of Ruiz (2001).

5.3 Starting point

Our algorithm to compute a starting point first solves

$$\begin{bmatrix} -(Q+\rho I) & A^T \\ A & \delta I \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}, \tag{17}$$

and sets

$$\tilde{s}_\ell = Q\tilde{x} - A^T\tilde{y} + c \tag{18a}$$
$$\tilde{s}_u = -(Q\tilde{x} - A^T\tilde{y} + c). \tag{18b}$$

As mentioned by Friedlander and Orban (2012), since (17) has the same sparsity pattern as (K2), which is solved at each interior-point iteration, if we use a factorization to solve the interior-point system, its symbolic analysis can be performed only once.

Next, we adapt the starting point procedure of Mehrotra (1992) to two-sided bounds. We introduce $\delta_{x_\ell} = \max\left(-\frac{3}{2}\min(\tilde{x}-\ell), \epsilon_M^{1/4}\right)$, $\delta_{x_u} = \max\left(-\frac{3}{2}\min(u-\tilde{x}), \epsilon_M^{1/4}\right)$, $\delta_{s_\ell} = \max\left(-\frac{3}{2}\min(s_\ell), \epsilon_M^{1/4}\right)$, and $\delta_{s_u} = \max\left(-\frac{3}{2}\min(s_u), \epsilon_M^{1/4}\right)$, where ϵ_M is the machine precision of the initial precision, and set

$$\tilde{\delta}_{x_\ell} = \delta_{x_\ell} + \tfrac{1}{2}\frac{\tilde{s}_\ell^T(\tilde{x} + \delta_{x_\ell} - \ell)}{\sum_i (s_{\ell i} + \delta_{s_\ell})} \tag{19a}$$
$$\tilde{\delta}_{x_u} = \delta_{x_u} + \tfrac{1}{2}\frac{\tilde{s}_u^T(u - \tilde{x} - \delta_{x_u})}{\sum_i (s_{ui} + \delta_{s_u})} \tag{19b}$$

$$\tilde{\delta}_{s_\ell} = \delta_{s_\ell} + \tfrac{1}{2}\frac{\tilde{s}_\ell^T(\tilde{x} + \delta_{x_\ell} - \ell)}{\sum_i(\tilde{x}_i + \delta_{x_\ell} - \ell_i)} \tag{19c}$$

$$\tilde{\delta}_{s_u} = \delta_{s_u} + \tfrac{1}{2}\frac{\tilde{s}_u^T(u - \tilde{x} - \delta_{x_u})}{\sum_i(u_i - \tilde{x}_i - \delta_{x_u})}. \tag{19d}$$

Finally, we set the initial values

$$x_0 = \tilde{x} + \tilde{\delta}_{x_\ell} - \tilde{\delta}_{x_u} \tag{20a}$$

$$s_{\ell,0} = \tilde{s}_\ell + \tilde{\delta}_{s_\ell} \tag{20b}$$

$$s_{u,0} = \tilde{s}_u + \tilde{\delta}_{s_u}. \tag{20c}$$

This procedure ensures $s_{\ell,0} \geq 0$ and $s_{u,0} \geq 0$. However, if for some $i \in \{1, \ldots, n\}$ we have $-\infty < \ell_i \leq u_i < +\infty$, then we could obtain $x_{0,i} \notin (\ell_i, u_i)$. In this case, we set $x_{0,i}$ close to the bound it was violating, and leave $s_{\ell,0}$ and $s_{u,0}$ unchanged. For example, if $x_{0,i} \leq \ell_i$, we set $x_{0,i} = \ell_i + \epsilon_M^{1/4}$.

5.4 Updating the regularization

When solving (K2) with a signed Cholesky factorization (without the dynamic regularization described in Section 4.2), some problems require lower regularization values to converge to the required tolerances. We detect that a lower regularization is needed when the current point becomes too close to its bounds, i.e., $x - \ell$ and/or $u - x$ have indices that are too close to zero, or when μ is below the machine precision ϵ_M. The latter case should not occur if the current point is not close to the central path, defined by $r_b = 0$, $r_c = 0$ and $s_{\ell,i}(x_i - \ell_i) = s_{u,i}(u_i - x_i) = \tau > 0$ for all $i \in \{1, ..., n\}$, and is therefore a good indicator that we should adjust the regularization values.

But when the regularization values are too low, the factorization is likely to fail. When it does, we decide to increase the regularization values and to recompute the factorization. When the factorization fails more than 10 times, we decide that the problem cannot be solved to the required tolerances, and we return the current point.

5.5 Stopping criteria and transition between floating-point systems

We define the relative primal-dual gap as

$$pdd := \frac{\left|c^T x + x^T Q x - y^T b - s_\ell^T \ell + s_u^T u\right|}{1 + \left|c^T x + \frac{1}{2} x^T Q x\right|}. \tag{21}$$

A quadratic problem is deemed to have been solved once r_c and r_b defined in (6), and pdd defined in (21), are smaller than some user-defined tolerances.

In multi-precision mode, the transition between several floating-point systems is also based on these residuals, but with higher tolerances. In addition, the transition is performed if the signed Cholesky factorization (used without dynamic regularization) fails, or if the current point becomes too close to its bounds. As we know that the n first diagonal elements of (K2) increase when, for a given index i, $x_i - \ell_i \to 0$ or $u_i - x_i \to 0$, and gets close to zero when $Q_{i,i} = 0$, $s_{\ell,i} \to 0$ and $s_{u,i} \to 0$, we decide that a point is too close from its bounds if the absolute value of the greatest diagonal element of K_2 is greater than $\frac{1}{\epsilon_M}$, and the absolute value of the smallest diagonal element is smaller than $\frac{1}{10}$. These additional transition criteria are similar to those of Section 5.4, but instead of modifying the regularization values, we switch to a higher-accuracy floating-point system.

6 Numerical results

We present some results of various tests showing the performance and multi-precision features of RipQP (Orban and Leconte, 2022) with (K2). The tests can easily be reproduced using the package RipQPBenchmarks.jl, available from https://github.com/geoffroyleconte/RipQPBenchmarks.jl.

6.1 Time comparisons with other solvers in double precision

In this section, we compare RipQP using (K2) with LDLFactorizations.jl to commercial solvers with a single thread. The problems used for the comparisons are the 114 linear problems from the Netlib dataset (Netlib), and the 138 convex quadratic problems from Maros and Mészáros (1999). All solvers are used with their respective barrier algorithm implementation, and crossover disabled. The stopping criteria for the solvers are 10^{-6} for the primal and dual residuals, and 10^{-8} for the relative primal-dual gap. All the other parameters of the above solvers are left to their default value. However, since each solver performs its own scaling, the primal and dual tolerances are not exactly the same. Nevertheless, the results in this section give a good indication as to the performance of RipQP compared to commercial solvers.

The solvers used are:

- CPLEX (IBM ILOG CPLEX Optimization Studio), via the Julia interface CPLEX.jl and QuadraticModelsCPLEX.jl on top of it,
- Gurobi (Gurobi Optimization, LLC, 2023) via the Julia interface Gurobi.jl and QuadraticModelsGurobi.jl on top of it,
- Xpress (FICO Xpress Optimization), via the Julia interface Xpress.jl and QuadraticModelsXpress.jl on top of it,

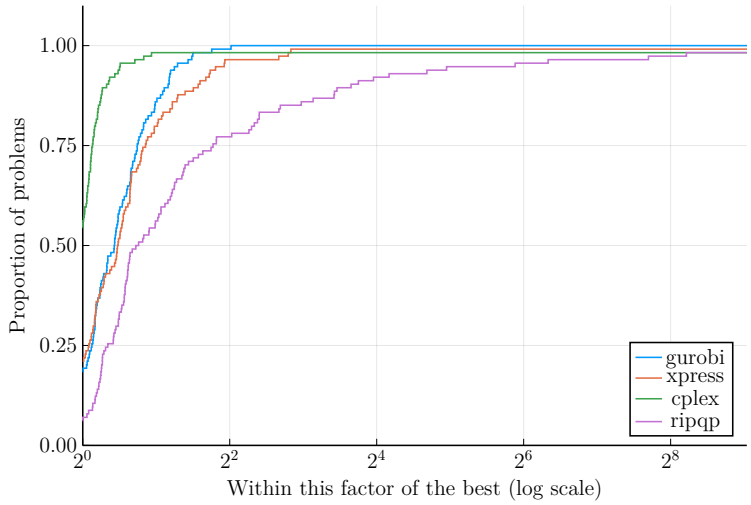

Figure 2: Time performance profile on Netlib problems between RipQP and some commercial solvers.

We can see from Figure 3 that CPLEX is the fastest solver, followed closely by RipQP which is faster than Gurobi and Xpress on most of the quadratic problems. RipQP is also the most robust on Maros and Meszaros problems. However, Figure 2 shows that it is slower than the other solvers on linear problems. This may be due to the fact that these solvers use the smaller system (K1) instead of (K2) to solve linear problems (solving (K1) is more difficult for quadratic problems when Q is not diagonal because $Q + D + \rho I$ has to be inverted at each iteration). Moreover, each solver has its own presolve algorithm, and that of RipQP is probably the least efficient. RipQP fails on the two Netlib problems *PILOT-JA* (factorization breakdown) and *PILOT-WE* (reaches the maximum number of iterations because the step sizes become increasingly small). These failures are linked to the factorization of (K2) and may be prevented by changing the factorization algorithm (see Section 6.2) and/or changing the default initial and minimal regularization values (but this might lead to failures on other problems).

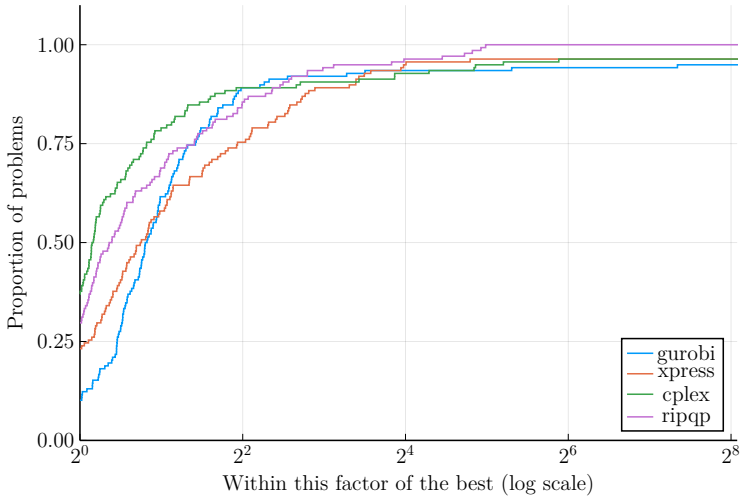

Figure 3: Time performance profile on Maros and Meszaros problems between RipQP and some commercial solvers.

6.2 Tests of different factorization algorithms

In this section we present results with the different factorization algorithms available within RipQP when solving a problem in factorization-only mode.

RipQP solves by default the augmented system (K2) with the signed Cholesky factorization package LDLFactorizations.jl (Orban and contributors, 2020). We also added interfaces to other factorization packages, which can be used if they are imported before importing RipQP:

- the CHOLMOD factorization from *SuiteSparse* (Chen et al., 2008),
- HSL_MA57 (Duff, 2004) with all default parameters and the SQD setting, and HSL_MA97 (Hogg and Scott, 2011) with all default parameters via the Julia interface HSL.jl (Orban and contributors, 2021a),
- QDLDL.jl, a Julia implementation of the factorization QDLDL used in the OSQP solver (Stellato et al., 2020).

A comparison of the performance of these factorization algorithms is shown in Figure 4 and Figure 5.

We can see that the performance of LDLFactorizations.jl and QDLDL.jl (two pure-julia factorizations) are similar. We also see that the choice of the factorization algorithm has an effect on the robustness of RipQP, for example with MA97, all the Netlib problems are solved. On most of the problems shown in Figure 4 and Figure 5, MA57 is the slowest algorithm. However, Table 1 shows that when solving the problems that are solved the slowest with LDLFactorizations.jl, it becomes most of the time the fastest algorithm.

6.3 Multiple centrality corrections

We implemented the centrality corrections from Gondzio (1996), with a slight modification on the maximum number of corrections to perform described in Algorithm 5. We show in Figure 6 and Figure 7 the comparisons of RipQP using (K2) and LDLFactorizations.jl with and without centrality corrections. Centrality corrections seem to be most interesting on quadratic problems, but the difference is barely noticeable. Even though the number of iterations is almost always lower with centrality corrections, the time to perform the additional solves often compensates for the iterations savings.

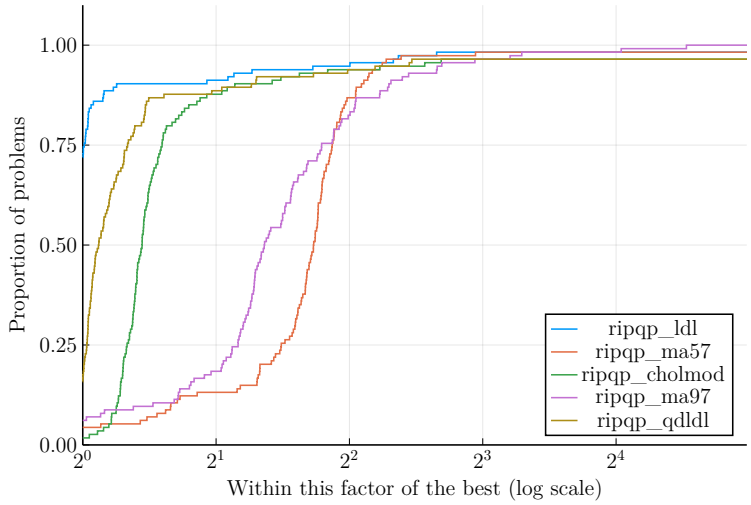

Figure 4: Time performance profile on Netlib problems with LDLFactorization.jl (*ripqp_ldl*), QDLDL.jl (*ripqp_qdldl*), MA57 (*ripqp_ma57*) and MA97 (*ripqp_ma97*) via HSL.jl, and CHOLMOD from SuiteSparse.jl (*ripqp_cholmod*).

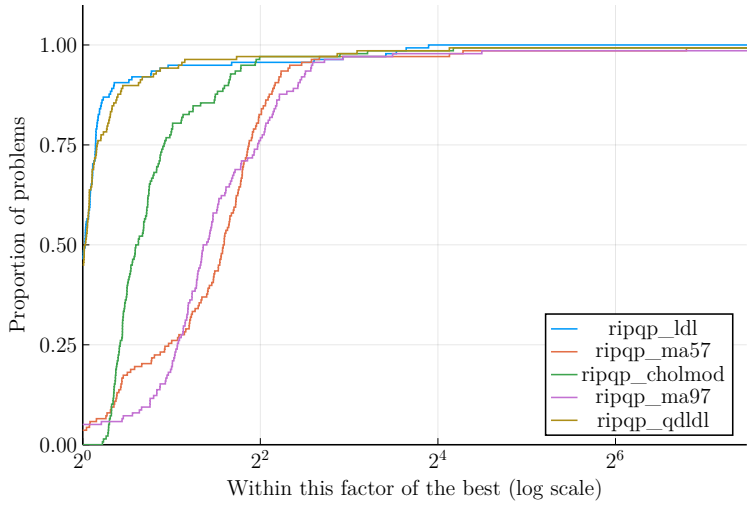

Figure 5: Time performance profile on Maros and Meszaros problems with LDLFactorization.jl (*ripqp_ldl*), QDLDL.jl (*ripqp_qdldl*), MA57 (*ripqp_ma57*) and MA97 (*ripqp_ma97*) via HSL.jl, and CHOLMOD from SuiteSparse.jl (*ripqp_cholmod*).

Table 1: Time comparison between RipQP with LDLFactorizations.jl and MA57 on the 9 Netlib problems that are solved the slowest.

problem	**time** *ripqp_ldl* **(s)**	**time** *ripqp_ma57* **(s)**
DFL001	8.8e+01	4.2e+01
KEN-18	2.6e+01	2.9e+01
OSA-60	1.6e+01	3.8e+01
PDS-06	1.0e+01	9.0e+00
PDS-10	6.9e+01	2.6e+01
PDS-20	7.6e+02	1.6e+02
PILOT87	1.0e+01	6.8e+00
QAP12	5.1e+01	1.9e+01
QAP15	4.3e+02	7.7e+01

Algorithm 5 Computation of the maximal number of centrality corrections $kmax$.

Once the starting point is computed according to the procedure described in Section 5.3, compute $r_{f/s}$, the factorization time divided by the solve time of the initial system.
if $r_{f/s} \leq 10$ **then**
    $kmax = 0$
else if $10 < r_{f/s} \leq 30$ **then**
    $kmax = 1$
else if $30 < r_{f/s} \leq 50$ **then**
    $kmax = 2$
else
    $kmax = 3$
end if

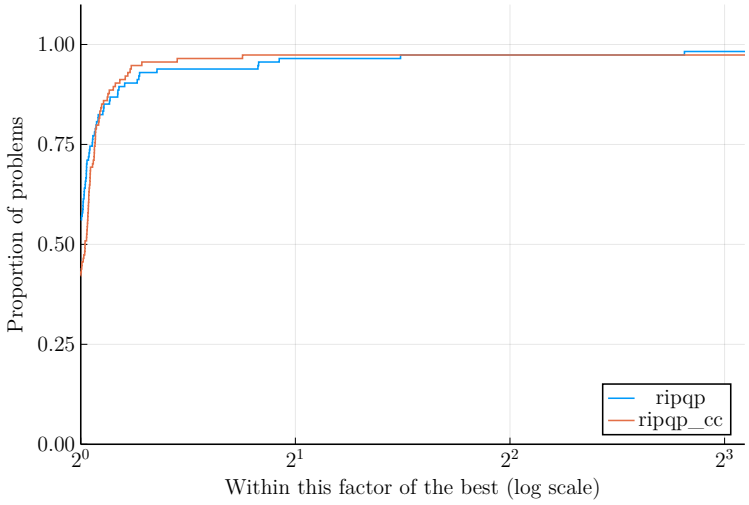

Figure 6: Time performance profile on Netlib problems with (*ripqp_cc*) and without (*ripqp*) centrality corrections.

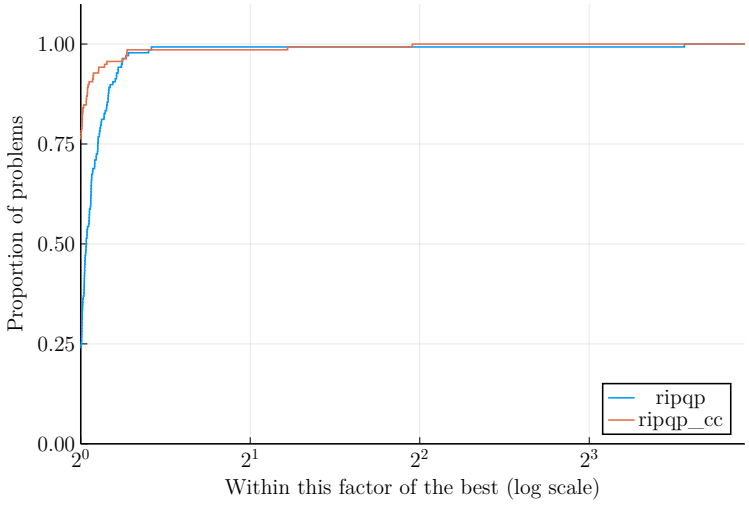

Figure 7: Time performance profile on Maros and Meszaros problems with (*ripqp_cc*) and without (*ripqp*) centrality corrections.

6.4 Multi-precision vs mono-precision

In this section, we compare the mono-precision mode of RipQP using a factorization in *Float64* to RipQP in multi-precision mode, starting the solve in *Float32*, and then transitioning to *Float64*. All

the tests are made with the predictor-corrector algorithm, and the linear system to solve at each iteration is (K2). The results that we present are a simulation of what would be the benefits of using a multi-precision algorithm on a platform with hardware facilities that can perform operations in *Float32* and *Float64*. As stated by Haidar et al. (2020), dividing by two the precision of the computations might be two times faster, and might save up to four times the energy required by the computations. That is why, in our experiment, the performance profiles are displayed so that an iteration in *Float32* counts as one iteration, and an iteration in *Float64* counts as four iterations. We call the performance profiles with this measure energy performance profiles.

Figure 8 and Figure 9 show the energy performance profiles between the mono-precision and multi-precision modes. The criteria used to transition from single to double floating-point arithmetic specified in Section 5.5 are 10^{-2} for *pdd*, 10^{-4} for r_b and r_c, and a maximum number of 40 iterations. When iterating in *Float32*, the regularization parameters ρ and δ of (K2) have to be set to a higher value so that the factorization does not fail. However, some problems require very low regularization values

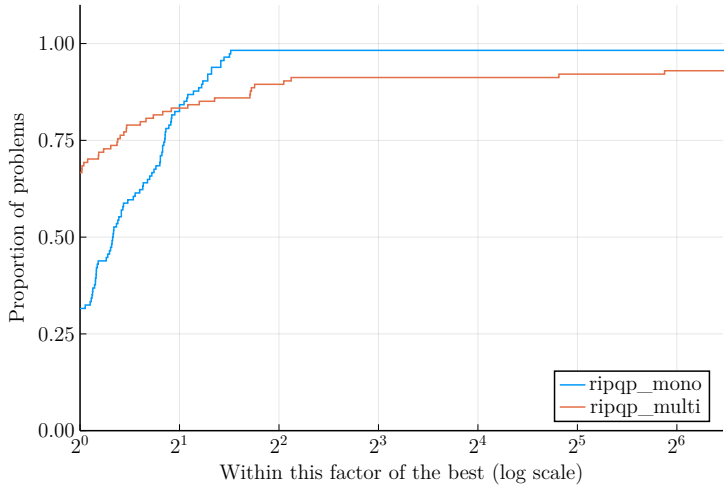

Figure 8: Energy performance profile using LDLFactorizations.jl on Netlib problems with RipQP in *Float64* only (*ripqp_mono*) and RipQP in *Float32* then *Float64* (*ripqp_multi*).

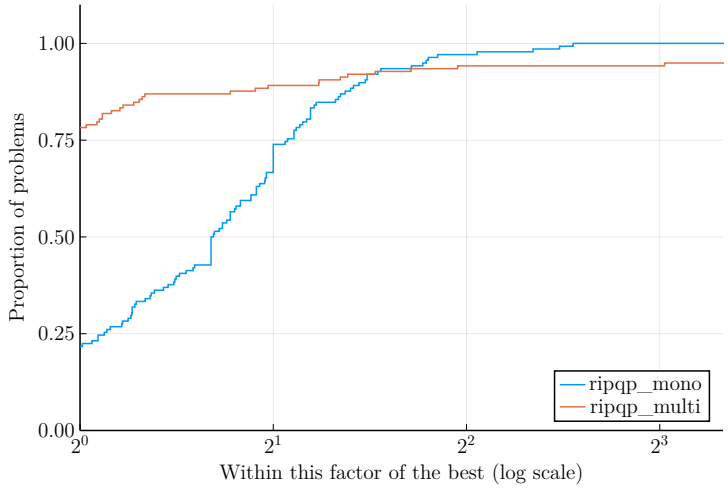

Figure 9: Energy performance profile using LDLFactorizations.jl on Maros and Meszaros problems with RipQP in *Float64* only (*ripqp_mono*) and RipQP in *Float32* then *Float64* (*ripqp_multi*).

to converge, which is why they are solved slower with the multi-precision mode, require more energy and sometimes get stuck close to a bound or lead to factorization breakdowns. This could be avoided by reducing the maximum number of iterations in *Float32* for these difficult problems, but the energy gains would be slower on the other problems. We show in Appendix A and Appendix B the detailed results of this comparison, including the number of iterations performed in *Float32* and *Float64*.

We also made experiments with the time to solve problems, using MA57 from HSL.jl. This factorization can work in *Float32* and *Float64*, and solves in *Float32* are about two times faster than the solves in *Float64*.

Figure 10 and Figure 11 are made with the same transition criteria to switch precision as in Figure 8 and Figure 9. However, since the benefits of multi-precision here are only of a factor two (instead of a factor four), we need to transition faster to *Float64*. Profiles 12 and 13 show results with *ripqp_ma57_multi*2 which uses less strict transition criteria: 10^{-2} for the primal and dual residuals, and 10^{0} for the relative primal-dual gap. With this setting, we can see that the multi-precision mode is faster than the mono-precision on more than 75% of the Netlib problems, and more than 60% of the Maros and Meszaros problems.

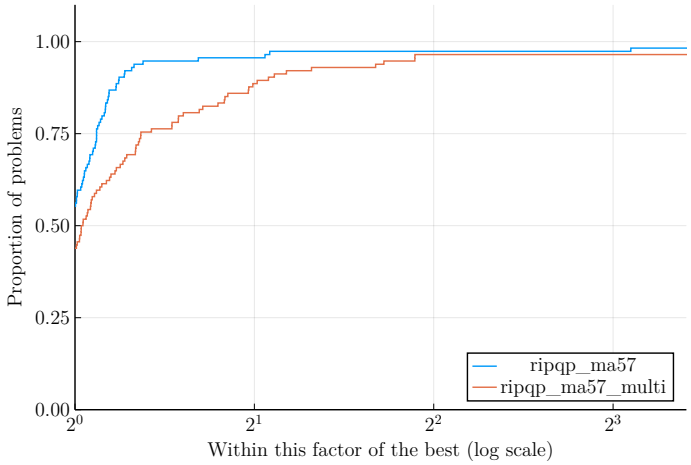

Figure 10: Time performance profile using MA57 on Netlib problems with RipQP in *Float64* only (*ripqp_ma57*) and RipQP in *Float32* then *Float64* (*ripqp_ma57_multi*).

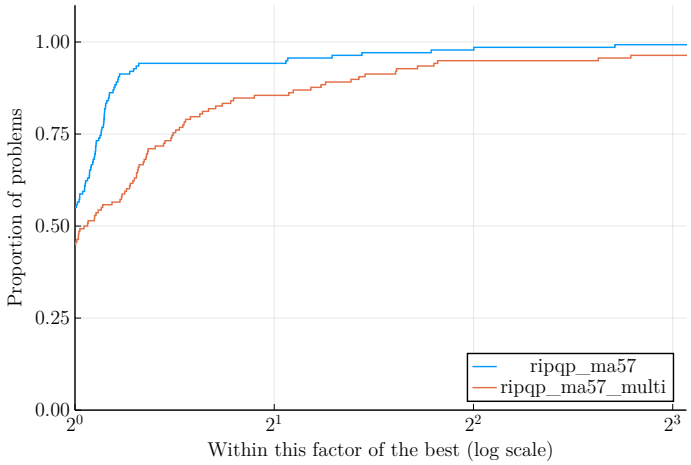

Figure 11: Time performance profile using MA57 on Maros and Meszaros problems with RipQP in *Float64* only (*ripqp_ma57*) and RipQP in *Float32* then *Float64* (*ripqp_ma57_multi*).

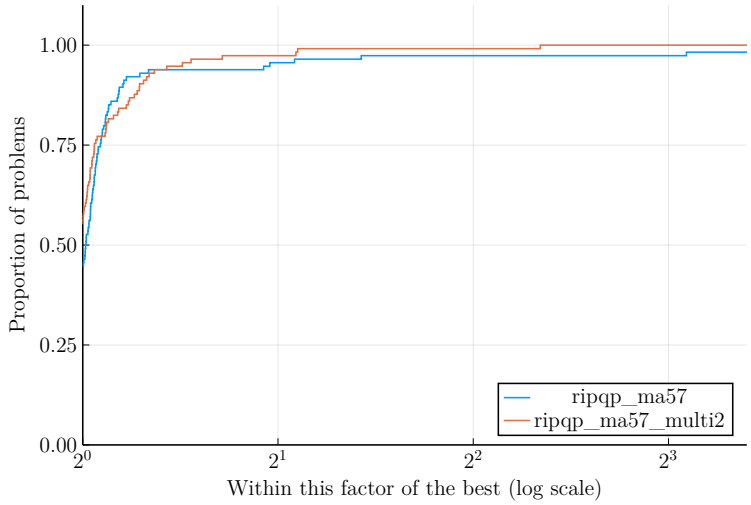

Figure 12: Time performance profile using MA57 on Netlib problems with RipQP in *Float64* only (*ripqp_ma57*) and RipQP in *Float32* then *Float64* with softer transition tolerances (*ripqp_ma57_multi2*).

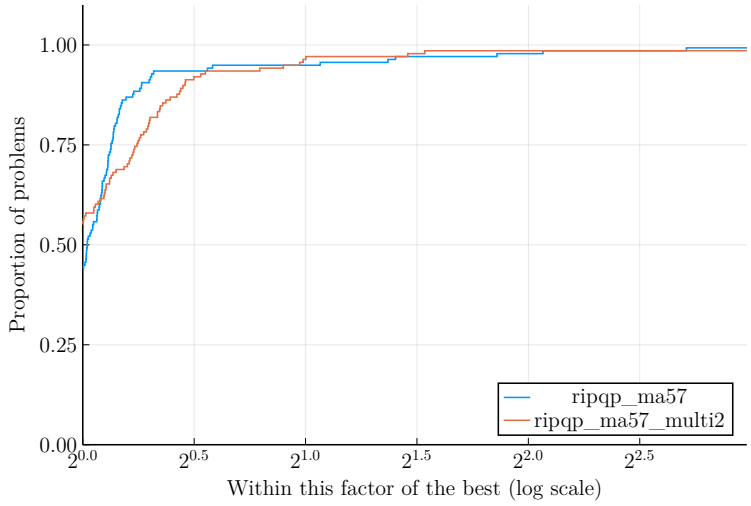

Figure 13: Time performance profile using MA57 on Maros and Meszaros problems with RipQP in *Float64* only (*ripqp_ma57*) and RipQP in *Float32* then *Float64* with softer transition tolerances (*ripqp_ma57_multi2*).

Future work could seek to improve the transition criteria when trying to decrease solve times. We show energy performance profiles between RipQP with MA57 in mono-precision, multi-precision and multi-precision with softer transition tolerances in Figure 14 and Figure 15. Because the measure of performance is different, these profiles are mostly in favor of *ripqp_ma57_multi*.

6.5 Preconditioning the augmented system with a lower-precision factorization

In this section, we compare the results of the multi-precision mode shown in Section 6.4 with an algorithm using a Krylov method preconditioned by a lower-precision factorization as in Algorithm 2. The goal is to improve the solve of a linear system by a lower-precision factorization using a form of iterative refinement.

Algorithm 6 is based upon Algorithm 2. In its second step, the use of GMRES in *Float64* with a signed Cholesky factorization in *Float64* is only useful when (K2) becomes too badly conditionned. Most of the time, only 0 or 1 iteration are needed. Even though (K2) is symmetric, we prefer using

GMRES instead of MINRES because of the loss of orthogonality due to its poor condition number. Moreover, using MINRES would force us to modify our preconditioner to be positive-definite by changing the pivots (computed by the signed Cholesky factorization) so that they are all positive. This operation can be performed easily at no computational cost, but it has the drawback to require more iterations of the Krylov method.

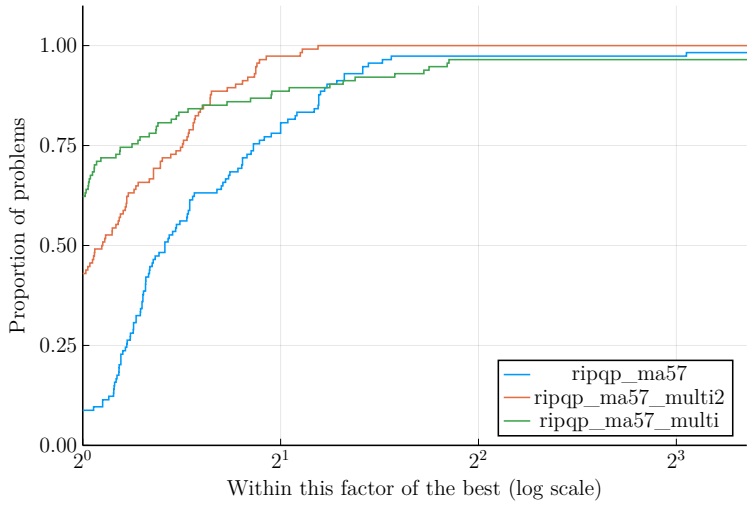

Figure 14: Energy performance profile using MA57 on Netlib problems with RipQP in *Float64* only (*ripqp_ma57*), RipQP in *Float32* then *Float64* (*ripqp_ma57_multi*), and RipQP in *Float32* then *Float64* with softer transition tolerances (*ripqp_ma57_multi2*).

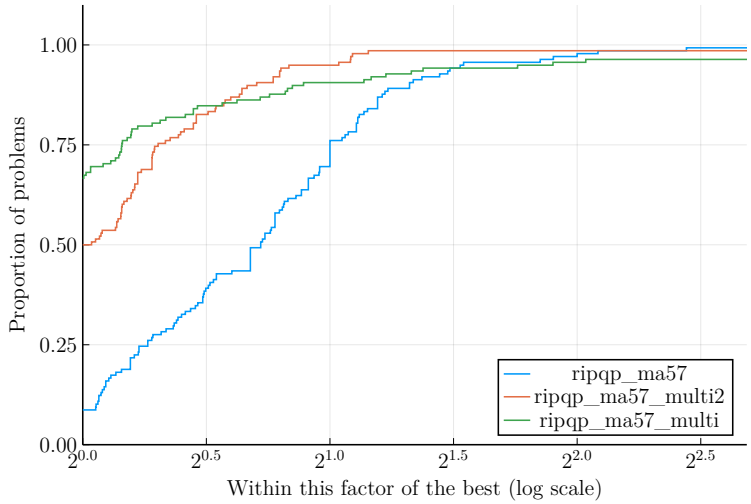

Figure 15: Energy performance profile using MA57 on Maros and Meszaros problems with RipQP in *Float64* only (*ripqp_ma57*), RipQP in *Float32* then *Float64* (*ripqp_ma57_multi*), and RipQP in *Float32* then *Float64* with softer transition tolerances (*ripqp_ma57_multi2*).

When computing the energy performance profiles, we made the assumption that the cost of the GMRES iterations could be neglected. Therefore, the profiles only monitor the energy of the factorizations, but we can expect that the savings we observed for the factorizations are sufficient to compensate for this additional costs, since we only perform a few GMRES iterations. The savings cannot be verified with *MA57* (which is about two times faster in single precision) because it does not support dynamic regularization. However, in Section 6.7, we will show results using a similar algo-

rithm to solve problems in quadruple precision that benefit from savings in the number of operations performed in quadruple precision (since it is not supported natively by the processor that we used).

Algorithm 6 ***ripqp_multifact.***

1: Run the predictor-corrector method using Algorithm 2 where $p = Float64$ and $q = Float32$, using GMRES with a memory and a maximum number of iterations of 10. We set the initial absolute and relative tolerances for the residuals of GMRES to $atol = rtol = 10^{-2}$, and we divide this value by 10 at every interior-point iteration until we reach $atol = rtol = 10^{-8}$.
2: Run the predictor-corrector method using Algorithm 2 where $p = Float64$ and $q = Float64$, using GMRES with a memory and a maximum number of iterations of 5. We start from the current values of $atol$ and $rtol$, and we divide them by 10 at every interior-point iteration until we reach $atol = rtol = 10^{-10}$.

Figure 16 and Figure 17 show that Algorithm 6 is efficient to reduce the number of interior-point iterations. Using the equilibration scaling to scale (K2) does not change much the performance of the algorithm.

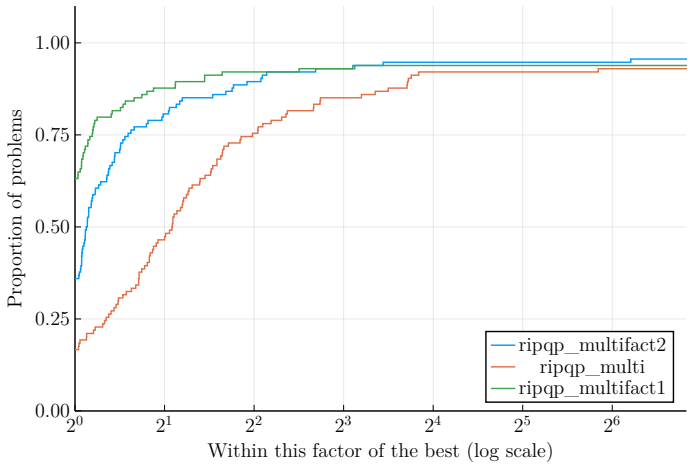

Figure 16: Energy performance profile on Netlib problems with RipQP in *Float32* then *Float64* (*ripqp_multi*), RipQP with Algorithm 6 (*ripqp_multifact1*), and RipQP with Algorithm 6 and equilibration scaling of (K2) (*ripqp_multifact2*).

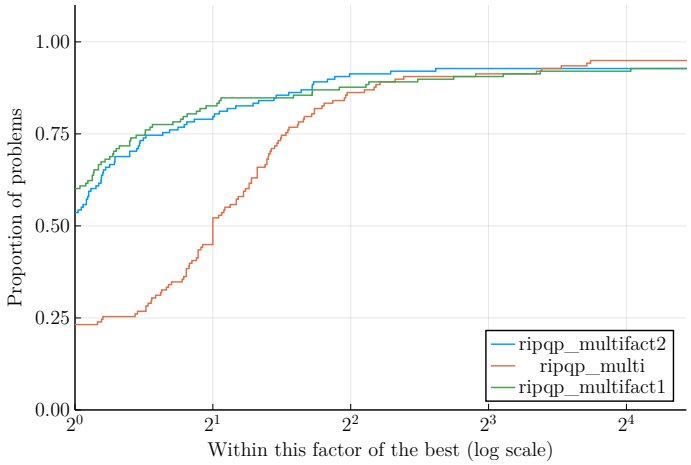

Figure 17: Energy performance profile on Maros and Meszaros problems with RipQP in *Float32* then *Float64* (*ripqp_multi*), RipQP with Algorithm 6 (*ripqp_multifact1*), and RipQP with Algorithm 6 and equilibration scaling of (K2) (*ripqp_multifact2*).

6.6 Using a limited-memory factorization as a preconditioner

Instead of using a signed Cholesky factorization preconditioner in a lower precision system as presented in Section 6.5, it is possible to use a limited-memory factorization. Orban (2015) presents a limited-memory signed Cholesky factorization that we used for our experiments. The code is available in the Julia package LimitedLDLFactorizations.jl (Orban and contributors, 2021b).

Algorithm 7 is a variant of Algorithm 6 where the first step in double precision preconditioned by a single precision factorization uses a limited-memory signed Cholesky factorization with a memory of 20 instead of a complete signed Cholesky factorization.

Algorithm 7 ***ripqp_multifact_lldl.***

1: Run the predictor-corrector method using Algorithm 2 with a limited-memory version of the signed Cholesky factorization, where $p = Float64$ and $q = Float32$, using GMRES with a memory and a maximum number of iterations of 10. The memory of the limited-memory factorization is set to 20.
2: Run the predictor-corrector method using Algorithm 2 where $p = Float64$ and $q = Float64$, using GMRES with a memory and a maximum number of iterations of 5.

We can deduce from Figure 18 and Figure 19 that the number of iterations is higher with the limited-memory factorization. However, Figure 20 and Figure 21 show that the limited-memory factorization is faster despite the additional number of iterations. In addition, we point out that our limited-memory signed Cholesky factorization works with a symmetric matrix represented by its lower triangle, whereas our signed Cholesky factorization works with a symmetric matrix represented by its upper triangle. As a consequence, the conversion between the solvers requires some additional memory and time to transpose (K2). This time could be reduced by writing a signed Cholesky factorization that works with a symmetric matrix represented by its lower triangle.

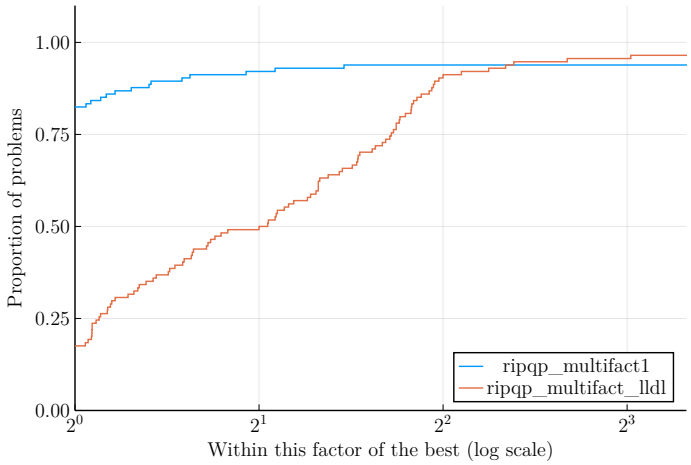

Figure 18: Energy performance profile on Netlib problems with Algorithm 6 and LDLFactorizations.jl (*ripqp_multifact1*), and with Algorithm 7 and LimitedLDLFactorizations.jl (*ripqp_multifact_lldl*).

6.7 Solving difficult problems in quadruple precision

We also tested the multi-precision features of RipQP when solving the difficult problems of Ma et al. (2017). Those problems require iterations in quadruple precision to reach satisfactory residuals. However, the early iterations of the interior-point method can still be performed in a lower floating-point system. The platform used to solve these problems does not support quadruple precision natively, so we had to maximize the operations in double precision to lower the solve time. For this purpose, we used Algorithm 8, which has a double precision warm-start (D) that uses more GMRES iterations than the previous algorithms. That is why we chose to perform only one (K2) solve with the infeasible path-following method, instead of two solves with the predictor-corrector method.

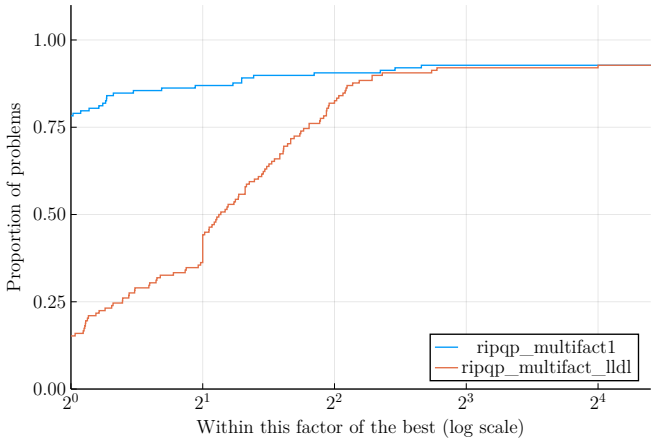

Figure 19: Energy performance profile on Maros and Meszaros problems with Algorithm 6 and LDLFactorizations.jl (*ripqp_multifact1*), and with Algorithm 7 and LimitedLDLFactorizations.jl (*ripqp_multifact_lldl*).

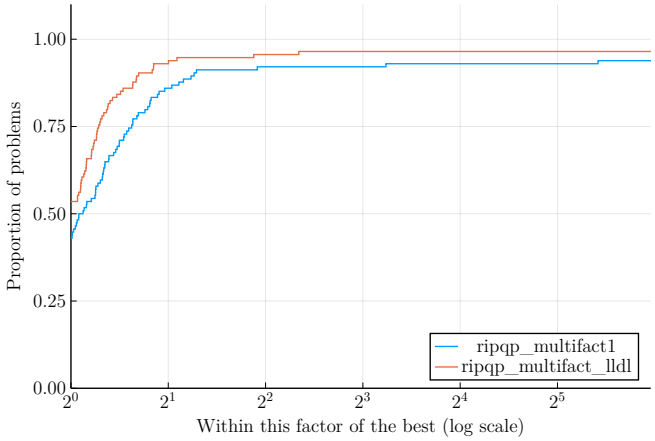

Figure 20: Time performance profile on Netlib problems with Algorithm 6 and LDLFactorizations.jl (*ripqp_multifact1*), and with Algorithm 7 and LimitedLDLFactorizations.jl (*ripqp_multifact_lldl*).

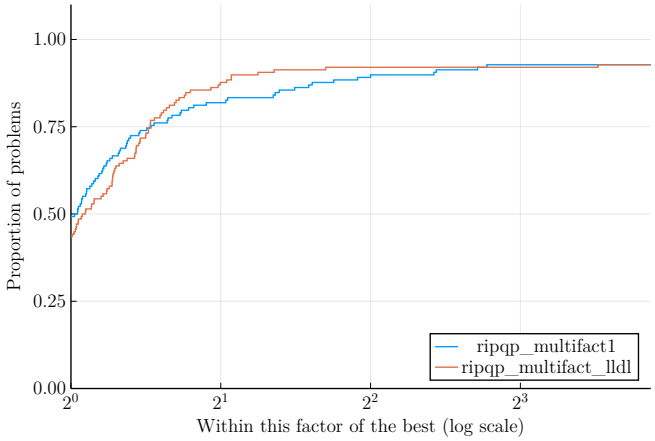

Figure 21: Time performance profile on Maros and Meszaros problems with Algorithm 6 and LDLFactorizations.jl (*ripqp_multifact1*), and with Algorithm 7 and LimitedLDLFactorizations.jl (*ripqp_multifact_lldl*).

Table 2 shows the choice of the parameters of Algorithm 8 for the solvers *multiquad*1 and *multiquad*2 using respectively the presolve algorithms from CPLEX and from RipQP. Since the presolve of CPLEX is more efficient than that of RipQP, we allow more iterations for *multiquad*1 in double precision with fewer GMRES iterations. However, the drawback of the presolve of CPLEX is that we cannot use its postsolve to find the original solution.

Algorithm 8 Algorithm *multiquad* to solve difficult problems in quadruple precision.

(D) Run the infeasible path-following method using Algorithm 2 with GMRES, where $p = Float64$ and $q = Float64$,
(Q) Run the predictor-corrector method using Algorithm 2 with GMRES, where $p = Float128$ and $q = Float128$,

Table 2: Parameters of Algorithm 8 used for RipQP in mode *multiquad*1 and *multiquad*2. The regularization values are the values of ρ and δ is (K2).

solver name	*multiquad*1		*multiquad*2	
presolve algorithm	CPLEX		RipQP	
step	D	Q	D	Q
maximum GMRES iterations	50	5	100	5
minimal regularization values	10^{-16}	10^{-16}	10^{-16}	10^{-16}
maximum number of IPM iterations	200	/	100	/

Table 3 shows that Algorithm 8 is the most efficient method to solve difficult problems in quadruple precision. The problems were declared solved when the relative primal-dual gap and the primal and dual residuals normalized by their respective initial value are lower than 10^{-16}. The last problem *GlcAerWT* is not solved by the *mono* and *multi* modes before the maximal time limit. In comparison, Ma et al. (2017) had solve times of 385 seconds for *TMA_ME*, 12600 seconds for *GlcAerWT*, and 16200 seconds for *GlcAlift* with their multi-precision simplex algorithm *DQQ* to reach the smallest residuals that they could get. For clarity, we show their results in Table 4. Even though these benchmarks were not computed with the same computer, this gives us a good order of magnitude of the efficiency of our method. Table 5 indicates the smallest residuals that we were able to reach with algorithm *multiquad*2. These residuals only require a few more iterations than the number of iterations shown in Table 3, and are only slightly higher than the residuals obtained by Ma et al. (2017).

Table 3: Results of RipQP in *mono* precision (*Float128* only), in *multi* precision mode (*Float64* then *Float128*), and in *multiquad1* and *multiquad2* modes described in Algorithm 8 with the parameters in Table 2. All the solvers use the presolve of CPLEX, except *multiquad2*. *obj* is the primal objective, *pdd* is the relative primal-dual gap, *pfeas* and *dfeas* are the primal and dual residuals.

Problem	solver	time (s)	iter64	iter128	obj	pdd	pfeas	dfeas
TMA_ME	*multiquad*1	5.5e+01	83	31	8.7e−07	1.3e−17	3.7e−21	1.3e−22
	multi	7.9e+01	35	54	8.7e−07	5.1e−19	3.5e−23	1.3e−20
	mono	1.4e+02	0	95	8.7e−07	4.4e−20	3.1e−26	3.5e−22
	*multiquad*2	2.2e+02	100	68	8.7e−07	6.9e−18	1.2e−19	1.0e−21
GlcAlift	*multiquad*1	8.7e+02	200	49	−7.0e+05	2.5e−18	1.6e−17	7.8e−18
	multi	1.9e+03	98	161	−7.0e+05	9.1e−18	9.8e−25	1.7e−19
	mono	3.5e+03	0	301	−7.0e+05	8.6e−18	1.2e−25	2.1e−17
	*multiquad*2	3.0e+03	100	196	−7.0e+05	3.5e−18	2.6e−18	1.5e−18
GlcAerWT	*multiquad*1	1.1e+03	200	65	−7.0e+05	1.0e−18	5.1e−17	7.0e−18
	multi	2.0e+04	100	1763	−7.0e+05	2.3e−07	7.2e−14	5.9e−08
	mono	2.0e+04	0	1766	−7.0e+05	4.5e−06	4.1e−26	5.3e−08
	*multiquad*2	8.3e+03	100	587	−7.0e+05	2.6e−19	1.8e−17	1.5e−18

Table 4: Smallest residuals reached with DQQ (Ma et al., 2017).

problem	pfeas	dfeas
TMA_ME	0	10^{-32}
GlcAlift	10^{-18}	10^{-23}
GlcAerWT	10^{-21}	10^{-22}

Table 5: Smallest residuals that we can reach with RipQP using algorithm $multiquad2$. *itertot* is the total number of iterations (it is not an energy measure: $itertot = iter64 + iter128$). The first 100 iterations are performed in *Float64*, and the others in *Float128*.

problem	ppd	pfeas	dfeas	itertot
TMA_ME	10^{-27}	10^{-29}	10^{-25}	174
GlcAlift	10^{-21}	10^{-25}	10^{-21}	319
GlcAerWT	10^{-20}	10^{-25}	10^{-20}	709

7 Discussion

We have presented in this paper several uses of our Julia solver RipQP. In particular, the results on convex quadratic problems show that it is competitive with commercial solvers. The modular aspect of our algorithm allows the usage of several factorization algorithms and formulations. Other factorization algorithms could be incorporated in a few lines of code.

The multi-precision features should allow the user to save about twice the required energy for the solves when using suitable hardware. These features already show some benefits regarding the time to solve difficult problems in quadruple precision, and extends the work done by Ma et al. (2017) for their multi-precision simplex algorithm to an interior-point method. When disposing from a factorization such as MA57, whose speed is increased by a factor two when changing the precision from *Float64* to *Float32*, we have also shown that the time to solve the quadratic problem in double precision is generally similar or slightly better in multi-precision. A faster factorization does not automatically translates into a lot of speedups of the algorithm, because the factorization is less accurate in single precision, and therefore the search directions are also less accurate, which leads to more interior-point iterations. That is why we experimented with several transition criteria from single to double precision, so that the single precision iterations do not add too many additional iterations compared to solving the quadratic problem only in double precision.

There are a lot of ideas that we could not explore in order to remain concise. Here is a list of some that could be interesting for future work:

- Use the formulation (K2.5) that has a bounded condition number. Since Krylov methods perform poorly on badly conditioned matrices, this should improve the solves with GMRES.
- Analyze the solves of all the interior-point systems described in Section 3 with several Krylov methods and preconditionners.
- Try to solve difficult problems in quadruple precision using a limited-memory factorization.
- Use strategies with (K1) when solving linear problems (which would require writing operations to keep AA^T as sparse as possible).
- Incorporate operations in half-precision (*Float16*). Overflow and underflow generally occur very quickly when using interior-point methods in half-precision, but we could try to incorporate some strategies at the very beginning of the algorithm, (before the condition number of (K2) deteriorates), using for example some strategies described by Amestoy et al. (2021).
- Saunders and Tenenblat (2006) and Weber et al. (2019) present algorithms that performs successive solves of refined linear are convex quadratic problems respectively, that we could try to implement by changing the precision of each solve.

A Detailed results in mono-precision

Table 6: RipQP results in mono-precision mode on Netlib problems.

Name	n	m	status	objective	pdd	pfeas	dfeas	time (s)	iter tot	iter64	iter32
25FV47	1571	821	*first_order*	5.5e+03	5.4e−10	6.7e−10	2.5e−12	1.5e−01	28	28	0
80BAU3B	9799	2262	*first_order*	9.9e+05	2.5e−09	1.8e−05	3.6e−12	3.8e−01	47	47	0
ADLITTLE	97	56	*first_order*	2.3e+05	6.2e−11	9.0e−13	6.2e−11	2.0e−03	14	14	0
AFIRO	32	27	*first_order*	−4.6e+02	7.4e−11	1.5e−13	9.7e−16	6.0e−04	10	10	0
AGG	163	488	*first_order*	−3.6e+07	4.2e−10	2.1e−08	8.1e−08	1.0e−02	23	23	0
AGG2	302	516	*first_order*	−2.0e+07	2.6e−09	2.3e−10	3.4e−05	5.6e−02	23	23	0
AGG3	302	516	*first_order*	1.0e+07	1.2e−10	1.8e−08	2.0e−10	5.3e−02	22	22	0
BANDM	472	305	*first_order*	−1.6e+02	8.5e−10	1.9e−11	9.5e−12	8.4e−03	17	17	0
BEACONFD	262	173	*first_order*	3.4e+04	5.5e−10	1.9e−10	3.9e−09	4.6e−03	12	12	0
BLEND	83	74	*first_order*	−3.1e+01	4.5e−11	2.1e−13	5.6e−11	2.1e−03	14	14	0
BNL1	1175	643	*first_order*	2.0e+03	9.5e−09	2.2e−08	2.7e−12	4.8e−02	36	36	0
BNL2	3489	2324	*first_order*	1.8e+03	1.4e−11	2.8e−10	5.0e−14	5.8e−01	44	44	0
BOEING1	384	351	*first_order*	−3.4e+02	2.2e−10	8.4e−12	5.8e−11	1.9e−02	26	26	0
BOEING2	143	166	*first_order*	−3.2e+02	1.9e−10	7.3e−12	5.9e−11	6.5e−03	22	22	0
BORE3D	315	233	*first_order*	1.4e+03	9.3e−10	2.0e−09	8.8e−11	4.0e−03	18	18	0
BRANDY	249	220	*first_order*	1.5e+03	2.0e−09	7.4e−10	2.5e−11	8.8e−03	19	19	0
CAPRI	353	271	*first_order*	2.7e+03	1.0e−10	1.1e−10	1.2e−13	7.6e−02	24	24	0
CRE-A	4067	3516	*first_order*	2.4e+07	2.2e−09	3.6e−08	2.2e−09	1.3e−01	29	29	0
CRE-B	72447	9648	*first_order*	2.3e+07	3.6e−11	2.2e−09	8.5e−11	3.2e+00	47	47	0
CRE-C	3678	3068	*first_order*	2.5e+07	7.1e−09	5.9e−09	7.3e−08	1.1e−01	29	29	0
CRE-D	69980	8926	*first_order*	2.4e+07	2.7e−09	2.9e−09	7.2e−08	2.5e+00	47	47	0
CYCLE	2857	1903	*first_order*	−5.2e+00	1.8e−09	1.4e−10	1.9e−12	4.8e−01	25	25	0
CZPROB	3523	929	*first_order*	2.2e+06	1.1e−09	3.9e−07	1.4e−12	5.3e−02	35	35	0
D2Q06C	5167	2171	*first_order*	1.2e+05	1.6e−11	6.4e−09	6.0e−09	1.1e+00	35	35	0
D6CUBE	6184	415	*first_order*	3.2e+02	5.3e−11	1.1e−08	2.7e−15	3.6e−01	23	23	0
DEGEN2	534	444	*first_order*	−1.4e+03	8.1e−11	2.1e−11	1.7e−11	3.0e−02	14	14	0
DEGEN3	1818	1503	*first_order*	−9.9e+02	1.7e−10	3.1e−11	2.8e−11	3.8e−01	18	18	0
DFL001	12230	6071	*first_order*	1.1e+07	5.7e−09	8.4e−07	1.5e−08	8.1e+01	44	44	0
E226	282	223	*first_order*	−1.2e+01	1.2e−09	3.7e−09	1.7e−08	2.0e−02	30	30	0
ETAMACRO	688	400	*first_order*	−7.6e+02	3.7e−09	4.6e−12	5.9e−08	4.0e−02	31	31	0
FFFFF800	854	524	*first_order*	5.6e+05	2.6e−11	7.0e−10	7.0e−08	7.4e−02	35	35	0
FINNIS	614	497	*first_order*	1.7e+05	4.2e−09	2.0e−09	4.4e−12	2.3e−02	36	36	0
FIT1D	1026	24	*first_order*	−9.1e+03	4.5e−10	1.1e−10	6.7e−13	2.7e−02	20	20	0
FIT1P	1677	627	*first_order*	9.1e+03	2.6e−10	7.0e−11	2.8e−11	2.5e−02	13	13	0
FIT2D	10500	25	*first_order*	−6.8e+04	2.0e−11	6.9e−10	3.4e−13	3.0e−01	24	24	0
FIT2P	13525	3000	*first_order*	6.8e+04	9.7e−09	7.3e−10	3.0e−10	2.0e−01	18	18	0
FORPLAN	421	161	*first_order*	−6.6e+02	1.5e−09	5.5e−12	3.6e−10	2.1e−02	33	33	0
GANGES	1681	1309	*first_order*	−1.1e+05	3.4e−09	5.1e−05	6.8e−12	5.2e−02	26	26	0
GFRD-PNC	1092	616	*first_order*	6.9e+06	5.7e−09	3.0e−06	1.6e−10	1.6e−02	22	22	0
GREENBEA	5405	2392	*first_order*	−7.3e+07	1.3e−09	1.3e−03	5.8e−09	2.9e+00	267	267	0
GREENBEB	5405	2392	*first_order*	−4.3e+06	1.1e−09	1.1e−07	5.1e−10	7.6e−01	81	81	0
GROW15	645	300	*first_order*	−1.1e+08	1.6e−09	8.3e−07	1.6e−13	1.5e−02	14	14	0
GROW22	946	440	*first_order*	−1.6e+08	1.0e−09	1.3e−06	5.8e−14	2.5e−02	15	15	0
GROW7	301	140	*first_order*	−4.8e+07	5.8e−11	4.0e−09	2.8e−14	7.5e−03	14	14	0
ISRAEL	142	174	*first_order*	−9.0e+05	3.5e−09	5.0e−10	3.7e−10	9.5e−03	23	23	0
KB2	41	43	*first_order*	−1.7e+03	5.8e−11	1.2e−09	5.3e−09	1.5e−03	18	18	0
KEN-07	3602	2426	*first_order*	−6.8e+08	4.5e−09	1.4e−09	3.9e−09	3.2e−02	16	16	0
KEN-11	21349	14694	*first_order*	−7.0e+09	1.1e−09	9.6e−10	1.1e−08	4.4e−01	26	26	0
KEN-13	42659	28632	*first_order*	−1.0e+10	7.3e−10	2.3e−08	5.3e−11	1.5e+00	30	30	0
KEN-18	154699	105127	*first_order*	−5.2e+10	5.1e−11	1.8e−06	3.4e−09	2.2e+01	42	42	0
LOTFI	308	153	*first_order*	−2.5e+01	1.5e−11	1.9e−09	1.8e−15	6.5e−03	21	21	0
MAROS-R7	9408	3136	*first_order*	1.5e+06	5.8e−09	9.7e−11	2.5e−12	7.7e+00	17	17	0
MAROS	1443	846	*first_order*	−5.8e+04	1.0e−09	5.8e−08	3.3e−08	6.1e−02	25	25	0
MODSZK1	1620	687	*first_order*	3.2e+02	1.3e−09	1.1e−10	9.1e−13	2.4e−02	27	27	0
NESM	2923	662	*first_order*	1.4e+07	5.4e−09	3.7e−08	1.5e−11	1.0e−01	37	37	0
OSA-07	23949	1118	*first_order*	5.4e+05	8.8e−11	2.2e−09	5.8e−12	6.8e−01	20	20	0
OSA-14	52460	2337	*first_order*	1.1e+06	2.8e−09	1.3e−09	2.7e−11	3.2e+00	31	31	0
OSA-30	100024	4350	*first_order*	2.1e+06	2.1e−09	6.2e−09	5.5e−11	7.6e+00	42	42	0
OSA-60	232966	10280	*first_order*	4.0e+06	7.3e−09	2.0e−08	3.8e−11	1.6e+01	30	30	0
PDS-02	7535	2953	*first_order*	2.9e+10	3.0e−10	4.1e−08	1.1e−08	2.3e−01	38	38	0
PDS-06	28655	9881	*first_order*	2.8e+10	6.7e−09	8.1e−08	5.0e−08	9.5e+00	44	44	0
PDS-10	48763	16558	*first_order*	2.7e+10	2.4e−09	3.2e−07	3.7e−08	5.6e+01	51	51	0
PDS-20	105728	33874	*first_order*	2.4e+10	4.5e−11	3.1e−07	2.2e−09	6.8e+02	63	63	0
PEROLD	1376	625	*first_order*	−9.4e+03	3.0e−12	1.5e−05	3.0e−10	1.5e−01	39	39	0
PILOT-JA	1988	940	*unknown*	−6.1e+03	1.3e−05	4.7e−10	6.3e−05	1.4e+00	211	211	0
PILOT-WE	2789	722	*max_iter*	−2.7e+06	1.3e−02	7.6e−02	9.3e+00	1.5e+00	800	800	0
PILOT	3652	1441	*first_order*	−5.6e+02	8.1e−09	4.4e−07	2.2e−11	2.2e+00	49	49	0
PILOT4	1000	410	*first_order*	−2.6e+03	1.6e−09	1.3e−07	2.4e−11	4.4e−02	34	34	0
PILOT87	4883	2030	*first_order*	3.0e+02	6.7e−09	3.7e−09	9.8e−11	8.9e+00	47	47	0
PILOTNOV	2172	975	*first_order*	−4.5e+03	5.8e−11	3.2e−10	1.0e−10	3.7e−01	61	61	0
QAP12	8856	3192	*first_order*	5.2e+02	1.0e−09	7.4e−12	8.3e−11	4.4e+01	18	18	0
QAP15	22275	6330	*first_order*	1.0e+03	8.1e−09	1.5e−11	2.1e−10	3.9e+02	22	22	0
QAP8	1632	912	*first_order*	2.0e+02	9.4e−11	7.3e−13	9.8e−14	5.3e−01	9	9	0
RECIPELP	180	91	*first_order*	−2.7e+02	2.5e−09	7.5e−11	2.4e−10	2.0e−03	10	10	0
SC105	103	105	*first_order*	−5.2e+01	3.7e−10	2.2e−12	5.2e−11	1.5e−03	11	11	0
SC205	203	205	*first_order*	−5.2e+01	2.5e−09	1.0e−11	6.1e−11	3.4e−03	15	15	0
SC50A	48	50	*first_order*	−6.5e+01	4.0e−10	4.1e−12	9.3e−12	7.6e−04	9	9	0
SC50B	48	50	*first_order*	−7.0e+01	5.7e−09	6.2e−12	1.1e−09	6.7e−04	8	8	0
SCAGR25	500	471	*first_order*	−1.5e+07	1.5e−10	1.8e−09	2.9e−09	6.3e−03	17	17	0
SCAGR7	140	129	*first_order*	−2.3e+06	3.9e−09	2.5e−11	9.9e−09	1.8e−03	13	13	0
SCFXM1	457	330	*first_order*	1.8e+04	1.2e−11	2.2e−09	3.4e−14	1.4e−02	23	23	0
SCFXM2	914	660	*first_order*	3.7e+04	3.1e−11	4.6e−09	1.2e−13	3.0e−02	25	25	0
SCFXM3	1371	990	*first_order*	5.5e+04	2.2e−09	3.9e−07	9.9e−12	4.4e−02	25	25	0
SCORPION	358	388	*first_order*	1.9e+03	1.4e−10	5.5e−12	1.1e−12	4.0e−03	12	12	0
SCRS8	1169	490	*first_order*	9.0e+02	8.1e−11	1.3e−12	1.4e−12	1.4e−02	19	19	0
SCSD1	760	77	*first_order*	8.7e+00	2.2e−10	1.7e−13	2.4e−13	4.4e−03	8	8	0
SCSD6	1350	147	*first_order*	5.1e+01	1.6e−09	1.8e−13	1.5e−14	8.6e−03	10	10	0
SCSD8	2750	397	*first_order*	9.1e+02	6.3e−10	5.6e−13	1.4e−12	1.7e−02	10	10	0
SCTAP1	480	300	*first_order*	1.4e+03	6.8e−10	1.1e−12	9.1e−14	6.7e−03	15	15	0
SCTAP2	1880	1090	*first_order*	1.7e+03	1.1e−11	1.1e−12	3.9e−14	2.8e−02	14	14	0
SCTAP3	2480	1480	*first_order*	1.4e+03	3.8e−09	1.1e−11	1.2e−12	3.8e−02	15	15	0
SEBA	1028	515	*first_order*	1.6e+04	2.6e−11	4.7e−12	2.2e−13	1.4e−02	20	20	0
SHARE1B	225	117	*first_order*	−7.7e+04	1.1e−10	5.2e−08	1.6e−07	5.3e−03	24	24	0
SHARE2B	79	96	*first_order*	−4.2e+02	1.4e−10	5.1e−12	5.1e−10	2.4e−03	13	13	0
SHELL	1775	536	*first_order*	1.2e+09	3.9e−09	6.9e−07	1.6e−10	1.8e−02	24	24	0
SHIP04L	2118	402	*first_order*	1.8e+06	2.9e−10	1.3e−09	1.8e−11	1.6e−02	14	14	0
SHIP04S	1458	402	*first_order*	1.8e+06	2.0e−10	3.6e−09	3.0e−12	1.2e−02	16	16	0
SHIP08L	4283	778	*first_order*	1.9e+06	1.2e−09	2.2e−10	3.6e−12	3.3e−02	20	20	0
SHIP08S	2387	778	*first_order*	1.9e+06	6.3e−09	2.4e−08	2.0e−09	1.7e−02	18	18	0

Continued on next page

Name	n	m	status	objective	pdd	pfeas	dfeas	time (s)	iter tot	iter64	iter32
SHIP12L	5427	1151	*first_order*	1.5e+06	1.3e−11	3.3e−11	7.3e−12	3.9e−02	17	17	0
SHIP12S	2763	1151	*first_order*	1.5e+06	2.6e−11	1.3e−11	1.6e−11	1.8e−02	15	15	0
SIERRA	2036	1227	*first_order*	1.5e+07	2.5e−09	4.3e−09	8.6e−11	4.3e−02	22	22	0
STAIR	467	356	*first_order*	−2.5e+02	6.1e−10	8.6e−09	1.3e−11	1.9e−02	19	19	0
STANDATA	1075	359	*first_order*	1.3e+03	2.5e−10	6.0e−13	7.7e−12	8.7e−03	12	12	0
STANDGUB	1184	361	*first_order*	1.3e+03	2.5e−10	6.0e−13	7.7e−12	8.7e−03	12	12	0
STANDMPS	1075	467	*first_order*	1.4e+03	3.0e−09	1.4e−11	1.4e−11	1.6e−02	20	20	0
STOCFOR1	111	117	*first_order*	−4.1e+04	2.7e−11	6.9e−11	4.4e−13	2.0e−03	13	13	0
STOCFOR2	2031	2157	*first_order*	−3.9e+04	4.3e−09	3.1e−08	1.2e−11	4.7e−02	19	19	0
STOCFOR3	15695	16675	*first_order*	−4.0e+04	1.9e−09	5.0e−08	2.7e−11	7.3e−01	37	37	0
TRUSS	8806	1000	*first_order*	4.6e+05	2.3e−09	4.9e−11	5.9e−10	1.4e−01	17	17	0
TUFF	587	333	*first_order*	2.9e−01	5.5e−11	1.7e−10	1.2e−13	2.5e−02	25	25	0
VTP-BASE	203	198	*first_order*	1.3e+05	1.6e−11	3.6e−11	1.8e−12	1.5e−03	14	14	0
WOOD1P	2594	244	*first_order*	1.4e+00	3.9e−11	3.7e−10	5.0e−14	1.2e−01	16	16	0
WOODW	8405	1098	*first_order*	1.3e+00	3.9e−11	2.0e−09	2.7e−14	1.5e−01	27	27	0

Table 7: RipQP results in mono-precision mode on Maros and Meszaros problems.

Name	n	m	status	objective	pdd	pfeas	dfeas	time (s)	iter tot	iter64	iter32
AUG2D	20200	10000	*first_order*	1.7e+06	6.4e−12	5.5e−12	2.2e−13	1.8e+00	5	5	0
AUG2DC	20200	10000	*first_order*	1.8e+06	8.0e−12	6.8e−12	2.7e−13	1.6e−01	5	5	0
AUG2DCQP	20200	10000	*first_order*	6.5e+06	9.8e−11	4.8e−12	1.5e−05	3.4e−01	12	12	0
AUG2DQP	20200	10000	*first_order*	6.2e+06	4.4e−09	7.1e−13	1.9e−05	3.5e−01	12	12	0
AUG3D	3873	1000	*first_order*	5.5e+02	4.7e−11	4.2e−11	2.6e−11	1.0e−01	3	3	0
AUG3DC	3873	1000	*first_order*	7.7e+02	1.9e−10	1.5e−10	6.7e−11	1.8e−02	3	3	0
AUG3DCQP	3873	1000	*first_order*	9.9e+02	5.7e−10	2.7e−14	4.4e−08	5.1e−02	12	12	0
AUG3DQP	3873	1000	*first_order*	6.8e+02	1.5e−09	4.0e−15	1.3e−15	5.3e−02	12	12	0
BOYD1	93261	18	*first_order*	−6.2e+07	8.8e−09	2.9e−02	9.2e−03	1.8e+00	30	30	0
BOYD2	93263	186531	*first_order*	2.1e+01	1.6e−10	7.8e−03	1.8e−08	1.6e+01	106	106	0
CONT-050	2597	2401	*first_order*	−4.6e+00	3.1e−09	3.4e−13	2.4e−11	1.4e−01	12	12	0
CONT-100	10197	9801	*first_order*	−4.6e+00	4.2e−09	8.2e−13	5.2e−11	1.2e+00	12	12	0
CONT-101	10197	10098	*first_order*	2.0e−01	2.1e−10	1.8e−14	7.4e−13	1.4e+00	14	14	0
CONT-200	40397	39601	*first_order*	−4.7e+00	2.3e−09	5.4e−13	1.3e−11	1.2e+01	13	13	0
CONT-201	40397	40198	*first_order*	1.9e−01	3.8e−10	1.7e−14	6.1e−12	1.4e+01	14	14	0
CONT-300	90597	90298	*first_order*	1.9e−01	2.4e−10	8.9e−15	2.8e−12	7.8e+01	16	16	0
CVXQP1_L	10000	5000	*first_order*	1.1e+08	9.0e−10	7.2e−10	1.0e−02	6.2e+01	12	12	0
CVXQP1_M	1000	500	*first_order*	1.1e+06	3.6e−10	3.3e−11	7.0e−05	1.6e−01	12	12	0
CVXQP1_S	100	50	*first_order*	1.2e+04	8.4e−10	2.9e−10	1.5e−07	1.5e−03	6	6	0
CVXQP2_L	10000	2500	*first_order*	8.2e+07	3.3e−10	1.5e−11	2.9e−03	4.1e+01	11	11	0
CVXQP2_M	1000	250	*first_order*	8.2e+05	2.0e−11	4.2e−12	2.8e−05	7.5e−02	9	9	0
CVXQP2_S	100	25	*first_order*	8.1e+03	8.6e−10	2.0e−12	3.5e−06	1.5e−03	8	8	0
CVXQP3_L	10000	7500	*first_order*	1.2e+08	1.9e−09	1.1e−10	1.9e−03	6.9e+01	12	12	0
CVXQP3_M	1000	750	*first_order*	1.4e+06	4.0e−11	2.5e−10	2.7e−06	2.3e−01	16	16	0
CVXQP3_S	100	75	*first_order*	1.2e+04	1.7e−11	5.7e−12	2.2e−08	2.2e−03	9	9	0
DPKLO1	133	77	*first_order*	3.7e−01	2.7e−10	1.1e−10	1.6e−10	2.1e−03	3	3	0
DTOC3	14999	9998	*first_order*	2.4e+02	3.5e−12	8.0e−15	9.9e−15	1.0e−01	6	6	0
DUAL1	85	1	*first_order*	3.5e−02	9.2e−10	0.0e+00	4.7e−07	7.8e−03	9	9	0
DUAL2	96	1	*first_order*	3.4e−02	5.3e−09	1.1e−16	7.1e−07	4.5e−03	7	7	0
DUAL3	111	1	*first_order*	1.4e−01	4.2e−09	8.9e−16	1.8e−06	7.6e−03	9	9	0
DUAL4	75	1	*first_order*	7.5e−01	7.3e−09	2.2e−16	4.8e−13	2.8e−03	8	8	0
DUALC1	9	215	*first_order*	6.2e+03	5.9e−10	6.8e−13	1.1e−08	3.6e−03	10	10	0
DUALC2	7	229	*first_order*	3.6e+03	1.4e−11	9.1e−13	9.0e−11	2.9e−03	8	8	0
DUALC5	8	278	*first_order*	4.3e+02	1.1e−11	2.3e−13	5.7e−09	3.2e−03	7	7	0
DUALC8	8	503	*first_order*	1.8e+04	4.4e−10	5.8e−13	1.6e−07	5.7e−03	7	7	0
EXDATA	3000	3001	*first_order*	−1.4e+02	1.7e−09	1.8e−13	1.6e−07	1.6e+01	12	12	0
GENHS28	10	8	*first_order*	9.3e−01	1.4e−09	1.5e−07	3.9e−07	3.2e−04	2	2	0
GOULDQP2	699	349	*first_order*	1.8e−04	1.5e−09	3.7e−15	3.2e−07	2.3e−02	74	74	0
GOULDQP3	699	349	*first_order*	2.1e+00	3.9e−11	3.3e−15	7.4e−12	6.3e−03	15	15	0
HS118	15	17	*first_order*	6.6e+02	1.1e−11	1.4e−14	1.4e−11	4.8e−04	12	12	0
HS21	2	1	*first_order*	−1.0e+02	1.3e−11	0.0e+00	1.2e−10	2.1e−04	7	7	0
HS268	5	5	*first_order*	3.8e−09	8.1e−09	1.4e−14	5.5e−12	3.5e−04	20	20	0
HS35	3	1	*first_order*	1.1e−01	3.3e−10	1.2e−14	8.5e−11	2.1e−04	7	7	0
HS35MOD	3	1	*first_order*	2.5e−01	2.5e−09	1.7e−13	8.5e−16	2.4e−04	12	12	0
HS51	5	3	*first_order*	0.0e+00	7.9e−10	3.6e−10	2.3e−10	1.8e−04	3	3	0
HS52	5	3	*first_order*	5.3e+00	1.2e−09	6.5e−10	7.3e−10	1.9e−04	3	3	0
HS53	5	3	*first_order*	4.1e+00	3.2e−09	3.7e−13	6.2e−10	2.1e−04	4	4	0
HS76	4	3	*first_order*	−4.7e+00	2.2e−10	1.8e−13	2.1e−10	2.4e−04	6	6	0
HUES-MOD	10000	2	*first_order*	3.5e+07	1.3e−09	6.8e−12	9.1e−08	3.7e−02	9	9	0
HUESTIS	10000	2	*first_order*	3.5e+11	9.9e−12	3.6e−12	5.0e−04	4.4e−02	11	11	0
KSIP	20	1001	*first_order*	5.8e−01	1.5e−09	7.9e−12	3.4e−08	4.0e−02	14	14	0
LASER	1002	1000	*first_order*	2.4e+06	5.5e−12	3.0e−10	4.6e−07	8.3e−02	134	134	0
LISWET1	10002	10000	*first_order*	3.6e+01	1.0e−09	4.4e−16	2.1e−10	2.1e−01	30	30	0
LISWET10	10002	10000	*first_order*	4.9e+01	6.3e−09	5.6e−16	4.2e−10	5.0e−01	77	77	0
LISWET11	10002	10000	*first_order*	5.0e+01	1.1e−09	6.7e−16	8.3e−08	5.6e−01	89	89	0
LISWET12	10002	10000	*first_order*	1.7e+03	2.0e−09	7.5e−16	1.2e−09	1.1e+00	179	179	0
LISWET2	10002	10000	*first_order*	2.5e+01	3.1e−09	5.7e−14	2.5e−10	2.0e−01	20	20	0
LISWET3	10002	10000	*first_order*	2.5e+01	3.7e−09	5.1e−11	3.6e−07	1.3e−01	15	15	0
LISWET4	10002	10000	*first_order*	2.5e+01	1.2e−09	1.1e−11	3.2e−07	1.8e−01	22	22	0
LISWET5	10002	10000	*first_order*	2.5e+01	3.2e−09	6.6e−11	6.6e−07	1.5e−01	17	17	0
LISWET6	10002	10000	*first_order*	2.5e+01	6.4e−09	1.1e−11	1.0e−06	7.1e−01	107	107	0
LISWET7	10002	10000	*first_order*	5.0e+02	9.2e−09	7.8e−16	2.3e−09	2.2e−01	30	30	0
LISWET8	10002	10000	*first_order*	7.1e+02	1.2e−09	4.5e−16	1.1e−07	6.2e−01	96	96	0
LISWET9	10002	10000	*first_order*	2.0e+03	6.3e−09	2.6e−15	9.3e−10	6.9e−01	109	109	0
LOTSCHD	12	7	*first_order*	2.4e+03	1.0e−10	1.8e−12	1.8e−11	4.1e−04	8	8	0
MOSARQP1	2500	700	*first_order*	−9.5e+02	1.0e−09	7.4e−12	2.5e−06	2.2e−02	9	9	0
MOSARQP2	900	600	*first_order*	−1.6e+03	7.5e−09	3.8e−12	8.0e−06	2.0e−02	10	10	0
POWELL20	10000	10000	*first_order*	5.2e+10	1.0e−10	3.6e−10	1.7e−07	2.5e−01	31	31	0
PRIMAL1	325	85	*first_order*	−3.5e−02	1.6e−11	4.4e−13	4.8e−10	1.1e−02	10	10	0
PRIMAL2	649	96	*first_order*	−3.4e−02	1.5e−10	1.3e−13	3.8e−10	1.4e−02	8	8	0
PRIMAL3	745	111	*first_order*	−1.4e−01	7.3e−10	1.1e−11	1.6e−08	4.6e−02	10	10	0
PRIMAL4	1489	75	*first_order*	−7.5e−01	6.4e−09	4.2e−11	3.2e−15	3.2e−02	10	10	0
PRIMALC1	230	9	*first_order*	−6.2e+03	7.2e−11	4.3e−08	2.2e−08	2.7e−03	10	10	0
PRIMALC2	231	7	*first_order*	−3.6e+03	9.2e−11	1.8e−11	1.8e−09	2.1e−03	8	8	0
PRIMALC5	287	8	*first_order*	−4.3e+02	3.8e−09	1.4e−11	7.3e−09	2.3e−03	7	7	0
PRIMALC8	520	8	*first_order*	−1.8e+04	1.6e−10	7.6e−10	6.0e−09	4.1e−03	8	8	0
Q25FV47	1571	820	*first_order*	1.4e+07	4.2e−09	4.6e−08	1.5e−02	5.3e−01	27	27	0
QADLITTL	97	56	*first_order*	4.8e+05	9.3e−09	1.6e−08	5.4e−04	1.6e−03	11	11	0
QAFIRO	32	27	*first_order*	−1.6e+00	6.1e−11	1.6e−14	4.0e−11	5.6e−04	10	10	0
QBANDM	472	305	*first_order*	1.6e+04	9.6e−09	1.1e−09	7.5e−05	8.5e−03	18	18	0
QBEACONF	262	173	*first_order*	1.6e+05	6.6e−10	1.2e−10	1.6e−09	4.9e−03	15	15	0
QBORE3D	315	233	*first_order*	3.1e+03	4.1e−10	2.0e−09	8.8e−11	3.9e−03	18	18	0

Continued on next page

Name	n	m	status	objective	pdd	pfeas	dfeas	time (s)	iter tot	iter64	iter32
QBRANDY	249	220	*first_order*	2.8e+04	8.2e−11	8.2e−10	6.5e−09	7.8e−03	17	17	0
QCAPRI	353	271	*first_order*	6.7e+07	6.7e−09	4.6e−08	9.1e−06	1.4e−02	30	30	0
QE226	282	223	*first_order*	2.1e+02	5.2e−09	2.4e−12	2.4e−07	1.2e−02	19	19	0
QETAMACR	688	400	*first_order*	8.7e+04	5.4e−09	4.3e−10	5.3e−04	4.1e−02	26	26	0
QFFFFF80	854	524	*first_order*	8.7e+05	6.0e−11	1.3e−08	1.4e−06	7.6e−02	31	31	0
QFORPLAN	421	161	*first_order*	7.5e+09	9.1e−09	2.5e−06	1.8e−02	1.4e−02	23	23	0
QGFRDXPN	1092	616	*first_order*	1.0e+11	1.7e−09	2.8e−05	2.4e−04	1.3e−02	21	21	0
QGROW15	645	300	*first_order*	−1.0e+08	6.9e−09	1.7e−08	5.0e−01	2.2e−01	17	17	0
QGROW22	946	440	*first_order*	−1.5e+08	2.4e−09	1.7e−08	2.3e−01	3.2e−02	20	20	0
QGROW7	301	140	*first_order*	−4.3e+07	5.0e−09	1.0e−08	1.4e−01	9.3e−03	17	17	0
QISRAEL	142	174	*first_order*	2.5e+07	8.8e−09	2.0e−06	2.4e−03	1.1e−02	24	24	0
QPCBLEND	83	74	*first_order*	−7.8e−03	3.0e−10	5.7e−12	2.2e−07	2.2e−03	16	16	0
QPCBOEI1	384	351	*first_order*	1.2e+07	7.1e−09	4.8e−08	5.4e−03	2.0e−02	27	27	0
QPCBOEI2	143	166	*first_order*	8.2e+06	2.1e−09	1.2e−07	5.6e−03	9.7e−03	36	36	0
QPCSTAIR	467	356	*first_order*	6.2e+06	2.7e−11	6.6e−12	3.4e−04	2.1e−02	21	21	0
QPILOTNO	2172	975	*first_order*	4.7e+06	8.7e−09	1.0e−05	1.5e−08	5.9e−01	94	94	0
QPTEST	2	2	*first_order*	4.4e+00	5.5e−10	2.7e−15	8.3e−13	3.8e−04	10	10	0
QRECIPE	180	91	*first_order*	−2.7e+02	8.6e−10	1.9e−11	5.8e−12	2.6e−03	16	16	0
QSC205	203	205	*first_order*	−5.8e−03	5.3e−11	1.5e−11	5.6e−10	3.3e−03	12	12	0
QSCAGR25	500	471	*first_order*	2.0e+08	1.3e−09	4.5e−10	8.7e−06	7.3e−03	17	17	0
QSCAGR7	140	129	*first_order*	2.7e+07	7.7e−10	2.5e−10	1.1e−06	2.3e−03	17	17	0
QSCFXM1	457	330	*first_order*	1.7e+07	8.9e−09	5.8e−07	1.3e−02	1.6e−02	23	23	0
QSCFXM2	914	660	*first_order*	2.8e+07	4.1e−09	5.3e−06	2.1e−04	3.9e−02	30	30	0
QSCFXM3	1371	990	*first_order*	3.1e+07	6.5e−10	7.3e−06	7.4e−11	5.5e−02	29	29	0
QSCORPIO	358	388	*first_order*	1.9e+03	5.4e−10	2.5e−11	1.8e−08	4.7e−03	13	13	0
QSCRS8	1169	490	*first_order*	9.0e+02	5.0e−11	3.4e−08	6.7e−10	1.8e−02	21	21	0
QSCSD1	760	77	*first_order*	8.7e+00	2.2e−10	1.6e−13	8.6e−11	5.9e−03	8	8	0
QSCSD6	1350	147	*first_order*	5.1e+01	1.6e−10	1.2e−13	5.4e−08	1.4e−02	12	12	0
QSCSD8	2750	397	*first_order*	9.4e+02	4.7e−10	9.5e−12	1.0e−06	2.6e−02	11	11	0
QSCTAP1	480	300	*first_order*	1.4e+03	6.9e−11	5.9e−12	1.4e−09	8.3e−03	16	16	0
QSCTAP2	1880	1090	*first_order*	1.7e+03	1.1e−11	1.1e−12	1.7e−10	3.5e−02	14	14	0
QSCTAP3	2480	1480	*first_order*	1.4e+03	5.9e−11	9.9e−13	9.0e−11	5.1e−02	16	16	0
QSEBA	1028	515	*first_order*	8.1e+07	7.9e−10	4.5e−09	1.1e−04	2.0e−02	26	26	0
QSHARE1B	225	117	*first_order*	7.2e+05	3.5e−09	2.4e−08	1.7e−05	5.2e−03	22	22	0
QSHARE2B	79	96	*first_order*	1.2e+04	9.8e−10	3.5e−11	1.8e−07	2.6e−03	14	14	0
QSHELL	1775	536	*first_order*	1.6e+12	1.6e−09	7.2e−08	1.5e−01	4.2e−02	30	30	0
QSHIP04L	2118	402	*first_order*	2.4e+06	2.6e−10	6.8e−10	2.7e−07	1.8e−02	15	15	0
QSHIP04S	1458	402	*first_order*	2.4e+06	1.1e−10	7.9e−10	5.0e−08	1.1e−02	12	12	0
QSHIP08L	4283	778	*first_order*	2.4e+06	1.1e−09	1.7e−10	1.8e−04	3.4e−01	12	12	0
QSHIP08S	2387	778	*first_order*	2.4e+06	8.2e−10	6.8e−10	5.7e−04	5.1e−02	12	12	0
QSHIP12L	5427	1151	*first_order*	3.0e+06	1.3e−09	7.8e−09	6.3e−05	4.5e−01	14	14	0
QSHIP12S	2763	1151	*first_order*	3.1e+06	2.0e−09	1.6e−09	2.2e−05	5.3e−02	14	14	0
QSIERRA	2036	1227	*first_order*	2.4e+07	8.5e−11	1.7e−08	5.5e−07	4.6e−02	21	21	0
QSTAIR	467	356	*first_order*	8.0e+06	2.8e−10	1.1e−06	1.8e−04	2.7e−02	22	22	0
QSTANDAT	1075	359	*first_order*	6.4e+03	1.1e−09	2.1e−12	6.5e−08	1.0e−02	11	11	0
S268	5	5	*first_order*	3.8e−09	8.1e−09	1.4e−14	5.5e−12	4.1e−04	20	20	0
STADAT1	2001	3999	*first_order*	−2.9e+07	5.8e−12	3.7e−09	3.6e−08	4.0e−01	188	188	0
STADAT2	2001	3999	*first_order*	−3.3e+01	8.2e−11	1.9e−11	1.2e−11	1.3e−01	55	55	0
STADAT3	4001	7999	*first_order*	−3.6e+01	2.0e−09	7.0e−10	1.6e−10	2.9e−01	58	58	0
STCQP1	4097	2052	*first_order*	1.6e+05	1.2e−09	0.0e+00	3.6e−05	1.7e−02	8	8	0
STCQP2	4097	2052	*first_order*	2.2e+04	5.3e−10	0.0e+00	3.5e−06	3.2e−02	9	9	0
TAME	2	1	*first_order*	0.0e+00	4.6e−11	3.5e−14	7.5e−13	2.7e−04	4	4	0
UBH1	18009	12000	*first_order*	1.1e+00	5.9e−10	4.5e−13	3.0e−11	1.4e−01	13	13	0
VALUES	202	1	*first_order*	−1.4e+00	4.4e−09	2.8e−17	1.4e−06	5.3e−03	12	12	0
YAO	2002	2000	*first_order*	2.0e+02	1.9e−09	3.1e−15	6.8e−11	6.9e−02	53	53	0
ZECEVIC2	2	2	*first_order*	−4.1e+00	1.7e−10	2.5e−14	2.5e−10	3.7e−04	7	7	0

B Detailed results in multi-precision

Table 8: RipQP results in multi-precision mode on Netlib problems.

Name	n	m	status	objective	pdd	pfeas	dfeas	time (s)	iter tot	iter32	iter64
25FV47	1571	821	*first_order*	5.5e+03	6.9e−09	2.5e−08	6.7e−11	1.4e−01	27	14	13
80BAU3B	9799	2262	*max_iter*	6.8e+25	1.6e+07	4.0e+07	1.2e+163	1.7e+01	800	5	795
ADLITTLE	97	56	*first_order*	2.3e+05	1.8e−09	2.7e−08	9.4e−08	2.4e−03	15	12	3
AFIRO	32	27	*first_order*	−4.6e+02	2.4e−09	2.6e−08	1.2e−09	6.6e−04	10	8	2
AGG	163	488	*first_order*	−3.6e+07	8.1e−10	2.4e−08	2.2e−06	1.3e−02	31	9	22
AGG2	302	516	*first_order*	−2.0e+07	9.3e−10	1.1e−05	1.5e−05	1.1e−01	50	20	30
AGG3	302	516	*first_order*	1.0e+07	7.7e−10	1.3e−07	3.6e−08	9.3e−02	43	21	22
BANDM	472	305	*first_order*	−1.6e+02	1.1e−09	2.5e−09	9.0e−09	8.6e−03	18	12	6
BEACONFD	262	173	*first_order*	3.4e+04	4.1e−10	2.2e−07	9.8e−09	5.3e−03	14	4	10
BLEND	83	74	*first_order*	−3.1e+01	1.0e−11	1.9e−11	2.2e−10	2.0e−03	14	11	3
BNL1	1175	643	*first_order*	2.0e+03	4.7e−09	1.2e−08	5.4e−09	9.9e−02	80	5	75
BNL2	3489	2324	*first_order*	1.8e+03	8.2e−09	1.2e−08	3.1e−12	5.8e−01	44	5	39
BOEING1	384	351	*first_order*	−3.4e+02	3.6e−10	8.3e−08	1.3e−07	3.2e−02	50	5	45
BOEING2	143	166	*first_order*	−3.2e+02	1.9e−10	1.5e−11	5.1e−07	1.8e−02	70	26	44
BORE3D	315	233	*first_order*	1.4e+03	3.2e−09	3.8e−10	1.5e−05	1.1e−02	68	12	56
BRANDY	249	220	*first_order*	1.5e+03	6.1e−10	7.5e−09	4.9e−09	1.0e−02	21	14	7
CAPRI	353	271	*first_order*	2.7e+03	3.5e−09	1.1e−09	3.5e−10	1.9e−02	43	15	28
CRE-A	4067	3516	*first_order*	2.4e+07	2.5e−11	9.7e−09	2.2e−07	1.3e−01	31	15	16
CRE-B	72447	9648	*first_order*	2.3e+07	1.7e−11	3.3e−09	2.9e−09	2.6e+00	46	21	25
CRE-C	3678	3068	*first_order*	2.5e+07	1.5e−11	1.5e−10	1.3e−07	1.1e−01	33	14	19
CRE-D	69980	8926	*first_order*	2.4e+07	5.7e−09	5.6e−08	6.2e−06	2.0e+00	42	18	24
CYCLE	2857	1903	*first_order*	−5.2e+00	2.4e−10	5.3e−10	1.9e−13	2.0e−01	25	7	18
CZPROB	3523	929	*first_order*	2.2e+06	6.0e−09	6.4e−08	3.7e−08	5.2e−02	37	12	25
D2Q06C	5167	2171	*first_order*	1.2e+05	1.1e−10	2.2e−07	3.7e−09	1.1e+00	35	5	30
D6CUBE	6184	415	*first_order*	3.2e+02	1.4e−09	3.3e−08	1.6e−11	3.6e−01	25	16	9
DEGEN2	534	444	*first_order*	−1.4e+03	8.4e−11	1.0e−11	1.3e−10	2.8e−02	14	8	6
DEGEN3	1818	1503	*first_order*	−9.9e+02	3.0e−10	2.1e−07	7.1e−12	5.6e−01	28	10	18
DFL001	12230	6071	*max_time*	3.1e+07	6.0e+05	1.0e+00	1.8e+04	1.2e+03	625	12	613
E226	282	223	*first_order*	−1.2e+01	5.8e−11	2.1e−11	7.1e−12	2.9e−02	23	16	7
ETAMACRO	688	400	*first_order*	−7.6e+02	5.7e−09	3.8e−06	9.8e−06	5.1e−02	29	5	24
FFFFF800	854	524	*max_iter*	5.6e+05	1.7e−02	5.8e−05	1.5e+02	1.5e+00	800	15	785
FINNIS	614	497	*first_order*	1.7e+05	2.2e−09	6.1e−08	1.0e−08	1.6e−02	27	5	22
FIT1D	1026	24	*first_order*	−9.1e+03	2.0e−09	6.2e−08	3.6e−11	2.6e−02	21	13	8
FIT1P	1677	627	*first_order*	9.1e+03	6.3e−11	5.1e−10	2.3e−09	3.1e−02	13	8	5
FIT2D	10500	25	*first_order*	−6.8e+04	5.1e−09	4.6e−08	2.0e−11	2.8e−01	23	13	10
FIT2P	13525	3000	*first_order*	6.8e+04	1.7e−11	1.1e−12	5.5e−10	2.1e−01	19	11	8

Continued on next page

Name	n	m	status	objective	pdd	pfeas	dfeas	time (s)	iter tot	iter32	iter64
FORPLAN	421	161	*first_order*	−6.6e+02	5.7e−09	4.9e−08	2.2e−09	3.2e−02	45	10	35
GANGES	1681	1309	*first_order*	−1.1e+05	5.9e−09	3.1e−05	6.8e−09	5.3e−02	27	5	22
GFRD-PNC	1092	616	*first_order*	6.9e+06	1.1e−09	2.2e−06	1.5e−09	1.7e−02	25	7	18
GREENBEA	5405	2392	*first_order*	−7.3e+07	3.7e−10	3.3e−06	1.8e−10	3.8e+00	366	5	361
GREENBEB	5405	2392	*first_order*	−4.3e+06	5.4e−09	1.7e−06	7.7e−10	4.2e−01	42	7	35
GROW15	645	300	*first_order*	−1.1e+08	2.0e−11	2.4e−07	5.6e−12	4.4e−02	47	15	32
GROW22	946	440	*first_order*	−1.6e+08	7.1e−10	3.1e−06	4.6e−10	7.4e−02	61	16	45
GROW7	301	140	*first_order*	−4.8e+07	2.5e−10	4.2e−08	4.9e−12	2.4e−02	57	13	44
ISRAEL	142	174	*first_order*	−9.0e+05	7.4e−11	2.5e−08	4.6e−09	9.6e−03	24	15	9
KB2	41	43	*first_order*	−1.7e+03	4.0e−09	3.7e−08	1.2e−06	1.5e−03	20	14	6
KEN-07	3602	2426	*first_order*	−6.8e+08	1.9e−09	1.7e−11	4.0e−06	3.6e−02	21	11	10
KEN-11	21349	14694	*first_order*	−7.0e+09	6.5e−11	1.1e−11	8.8e−08	5.0e−01	30	14	16
KEN-13	42659	28632	*first_order*	−1.0e+10	2.0e−09	7.4e−08	2.4e−10	1.5e+00	28	5	23
KEN-18	154699	105127	*first_order*	−5.2e+10	2.2e−11	4.7e−08	7.0e−10	2.4e+01	41	5	36
LOTFI	308	153	*first_order*	−2.5e+01	2.6e−09	9.3e−10	2.8e−09	6.9e−03	23	6	17
MAROS-R7	9408	3136	*first_order*	1.5e+06	4.1e−09	4.6e−09	5.3e−09	1.1e+01	24	14	10
MAROS	1443	846	*first_order*	−5.8e+04	1.0e−09	7.4e−07	7.5e−10	7.0e−02	27	11	16
MODSZK1	1620	687	*first_order*	3.2e+02	1.9e−10	4.7e−11	4.5e−13	2.3e−02	23	3	20
NESM	2923	662	*first_order*	1.4e+07	8.4e−09	3.1e−06	2.4e−09	1.1e−01	37	5	32
OSA-07	23949	1118	*first_order*	5.4e+05	1.8e−09	5.9e−09	2.0e−10	1.0e+00	32	8	24
OSA-14	52460	2337	*first_order*	1.1e+06	3.4e−09	8.8e−08	7.4e−10	2.9e+00	29	8	21
OSA-30	100024	4350	*first_order*	2.1e+06	2.8e−09	4.9e−07	4.5e−11	5.5e+00	33	9	24
OSA-60	232966	10280	*first_order*	4.0e+06	5.9e−09	1.8e−07	7.2e−10	2.7e+01	49	10	39
PDS-02	7535	2953	*first_order*	2.9e+10	1.2e−10	1.7e−08	1.4e−07	2.0e−01	31	5	26
PDS-06	28655	9881	*first_order*	2.8e+10	9.7e−09	1.8e−08	1.0e−05	1.2e+01	43	5	38
PDS-10	48763	16558	*first_order*	2.7e+10	2.6e−09	2.6e−08	3.2e−07	1.1e+02	50	10	40
PDS-20	105728	33874	*max_time*	2.4e+10	1.5e−04	7.3e−04	3.0e−03	1.2e+03	45	10	35
PEROLD	1376	625	*first_order*	−9.4e+03	4.2e−09	2.4e−07	2.8e−09	6.1e−01	165	5	160
PILOT-JA	1988	940	*unknown*	−6.1e+03	2.5e−03	1.5e−05	7.4e−03	5.2e−01	68	5	63
PILOT-WE	2789	722	*unknown*	−2.7e+06	1.6e−06	1.7e−05	1.6e−01	2.6e−01	119	5	114
PILOT	3652	1441	*first_order*	−5.6e+02	5.9e−09	1.6e−05	3.8e−09	2.4e+00	51	5	46
PILOT4	1000	410	*max_iter*	−1.1e+03	1.1e−01	6.7e−04	3.0e+00	9.1e−01	800	5	795
PILOT87	4883	2030	*first_order*	3.0e+02	9.2e−09	9.1e−07	2.6e−08	1.0e+01	52	5	47
PILOTNOV	2172	975	*unknown*	−4.5e+03	6.7e−04	1.0e−05	2.8e−09	2.3e−01	31	5	26
QAP12	8856	3192	*first_order*	5.2e+02	3.0e−10	2.7e−08	4.6e−10	1.1e+02	40	8	32
QAP15	22275	6330	*first_order*	1.0e+03	1.3e−09	1.0e−10	7.1e−07	4.5e+02	23	9	14
QAP8	1632	912	*first_order*	2.0e+02	3.2e−11	1.2e−12	5.9e−12	5.8e−01	9	6	3
RECIPELP	180	91	*first_order*	−2.7e+02	9.3e−09	2.9e−08	1.4e−08	3.3e−03	10	7	3
SC105	103	105	*first_order*	−5.2e+01	7.9e−09	4.7e−10	3.6e−08	5.5e−02	653	9	644
SC205	203	205	*first_order*	−5.2e+01	1.5e−09	2.7e−10	3.0e−13	6.9e−02	434	16	418
SC50A	48	50	*first_order*	−6.5e+01	3.2e−11	2.3e−13	1.7e−11	2.2e−03	21	9	12
SC50B	48	50	*first_order*	−7.0e+01	3.1e−10	1.8e−08	7.8e−11	9.8e−04	11	9	2
SCAGR25	500	471	*first_order*	−1.5e+07	6.2e−11	4.9e−09	5.7e−08	1.1e−02	18	11	7
SCAGR7	140	129	*first_order*	−2.3e+06	5.5e−10	6.6e−08	1.8e−06	2.2e−03	14	8	6
SCFXM1	457	330	*first_order*	1.8e+04	4.6e−11	8.5e−09	1.4e−11	2.7e−02	43	15	28
SCFXM2	914	660	*first_order*	3.7e+04	1.0e−09	2.7e−08	4.5e−10	3.3e−02	24	9	15
SCFXM3	1371	990	*first_order*	5.5e+04	4.3e−09	7.1e−07	1.2e−09	4.9e−02	24	10	14
SCORPION	358	388	*first_order*	1.9e+03	1.4e−11	6.6e−12	1.2e−09	8.5e−03	26	8	18
SCRS8	1169	490	*first_order*	9.0e+02	3.4e−09	1.2e−08	2.0e−08	1.9e−02	26	9	17
SCSD1	760	77	*first_order*	8.7e+00	2.9e−10	4.0e−11	7.6e−10	4.9e−03	8	6	2
SCSD6	1350	147	*first_order*	5.1e+01	2.4e−09	1.3e−11	9.4e−12	1.2e−02	10	6	4
SCSD8	2750	397	*first_order*	9.0e+02	3.8e−10	4.7e−09	4.1e−09	1.9e−02	10	8	2
SCTAP1	480	300	*first_order*	1.4e+03	3.1e−11	5.4e−12	3.0e−12	7.3e−03	16	11	5
SCTAP2	1880	1090	*first_order*	1.7e+03	1.0e−11	2.3e−11	1.9e−11	2.8e−02	14	11	3
SCTAP3	2480	1480	*first_order*	1.4e+03	9.5e−09	5.0e−08	1.0e−08	4.2e−02	15	13	2
SEBA	1028	515	*first_order*	1.6e+04	1.1e−11	5.1e−12	6.5e−11	1.5e−02	19	14	5
SHARE1B	225	117	*first_order*	−7.7e+04	2.4e−09	1.1e−06	5.3e−08	2.1e−02	112	10	102
SHARE2B	79	96	*first_order*	−4.2e+02	1.0e−11	1.1e−10	1.5e−10	2.8e−03	14	9	5
SHELL	1775	536	*first_order*	1.2e+09	3.0e−09	8.0e−07	4.0e−10	2.0e−02	25	8	17
SHIP04L	2118	402	*first_order*	1.8e+06	1.3e−09	5.5e−09	5.7e−10	1.7e−02	14	8	6
SHIP04S	1458	402	*first_order*	1.8e+06	2.1e−09	1.0e−08	1.5e−08	1.2e−02	15	8	7
SHIP08L	4283	778	*first_order*	1.9e+06	1.9e−09	3.0e−10	1.3e−09	3.8e−02	23	6	17
SHIP08S	2387	778	*first_order*	1.9e+06	6.3e−11	1.6e−10	1.5e−06	2.4e−02	25	9	16
SHIP12L	5427	1151	*first_order*	1.5e+06	2.0e−10	3.4e−10	4.8e−10	7.4e−02	31	12	19
SHIP12S	2763	1151	*first_order*	1.5e+06	5.0e−10	1.2e−11	6.2e−06	5.2e−02	57	10	47
SIERRA	2036	1227	*first_order*	1.5e+07	1.4e−09	6.5e−06	1.1e−05	5.9e−02	28	10	18
STAIR	467	356	*first_order*	−2.5e+02	2.2e−10	1.5e−11	1.6e−11	2.4e−02	25	14	11
STANDATA	1075	359	*first_order*	1.3e+03	3.3e−09	2.3e−07	4.4e−09	9.3e−03	12	10	2
STANDGUB	1184	361	*first_order*	1.3e+03	3.3e−09	2.3e−07	4.4e−09	9.3e−03	12	10	2
STANDMPS	1075	467	*first_order*	1.4e+03	1.5e−10	2.7e−12	1.8e−12	1.5e−02	19	12	7
STOCFOR1	111	117	*first_order*	−4.1e+04	1.1e−11	8.3e−11	2.8e−11	2.7e−03	17	7	10
STOCFOR2	2031	2157	*first_order*	−3.9e+04	3.0e−09	2.4e−08	4.1e−10	5.4e−02	22	9	13
STOCFOR3	15695	16675	*first_order*	−4.0e+04	9.9e−11	8.5e−08	1.7e−10	7.7e−01	35	6	29
TRUSS	8806	1000	*first_order*	4.6e+05	7.1e−10	1.7e−07	2.1e−07	1.5e−01	18	16	2
TUFF	587	333	*first_order*	2.9e−01	3.2e−11	1.1e−10	1.3e−12	3.2e−02	32	12	20
VTP-BASE	203	198	*first_order*	1.3e+05	3.8e−09	7.9e−10	2.5e−06	2.5e−03	24	10	14
WOOD1P	2594	244	*first_order*	1.4e+00	2.1e−10	2.2e−10	2.5e−12	1.2e−01	16	11	5
WOODW	8405	1098	*first_order*	1.3e+00	2.3e−11	9.0e−11	3.4e−12	1.5e−01	27	19	8

Table 9: RipQP results in multi-precision mode on Maros and Meszaros problems.

Name	n	m	status	objective	pdd	pfeas	dfeas	time (s)	iter tot	iter32	iter64
AUG2D	20200	10000	*first_order*	1.7e+06	4.1e−10	3.5e−10	1.3e−11	1.7e−01	6	4	2
AUG2DC	20200	10000	*first_order*	1.8e+06	5.3e−10	4.5e−10	1.6e−11	1.9e−01	6	4	2
AUG2DCQP	20200	10000	*first_order*	6.5e+06	2.9e−12	9.5e−12	3.0e−06	3.8e−01	14	10	4
AUG2DQP	20200	10000	*first_order*	6.2e+06	2.8e−09	1.6e−12	2.5e−10	3.3e−01	12	8	4
AUG3D	3873	1000	*first_order*	5.5e+02	2.1e−09	4.8e−07	3.0e−08	9.9e−03	3	2	1
AUG3DC	3873	1000	*first_order*	7.7e+02	4.5e−09	4.2e−07	1.2e−07	1.9e−02	3	2	1
AUG3DCQP	3873	1000	*first_order*	9.9e+02	5.7e−10	2.8e−14	4.4e−08	5.3e−02	12	7	5
AUG3DQP	3873	1000	*first_order*	6.8e+02	1.5e−09	1.8e−15	1.1e−14	3.8e−01	12	6	6
BOYD1	93261	18	*first_order*	−6.2e+07	7.1e−10	6.4e−02	1.7e−03	1.8e+00	31	5	26
BOYD2	93263	186531	*first_order*	2.1e+01	3.8e−09	2.3e−01	5.7e−08	1.2e+01	74	5	69
CONT-050	2597	2401	*first_order*	−4.6e+00	1.7e−09	2.5e−12	7.9e−10	2.7e−01	23	12	11
CONT-100	10197	9801	*unknown*	−4.6e+00	4.5e−04	3.4e−09	1.7e−03	4.4e+00	41	6	35
CONT-101	10197	10098	*max_iter*	2.3e−01	6.5e−01	5.0e−08	6.4e−02	7.5e+01	800	10	790
CONT-200	40397	39601	*max_iter*	−4.7e+00	1.5e−02	3.7e−08	5.4e−03	7.5e+02	800	6	794
CONT-201	40397	40198	*max_iter*	2.6e−01	5.5e−01	1.5e−07	2.5e−01	7.5e+02	800	7	793
CONT-300	90597	90298	*max_time*	5.3e−01	2.6e+00	9.9e−08	2.4e−01	1.2e+03	306	7	299
CVXQP1_L	10000	5000	*first_order*	1.1e+08	5.6e−12	4.3e−13	2.8e−04	6.8e+02	63	22	41
CVXQP1_M	1000	500	*first_order*	1.1e+06	1.0e−09	6.1e−11	3.0e−05	7.8e−01	30	23	7
CVXQP1_S	100	50	*first_order*	1.2e+04	2.0e−11	5.1e−09	6.9e−07	3.9e−03	7	5	2

Continued on next page

Name	n	m	status	objective	pdd	pfeas	dfeas	time (s)	iter tot	iter32	iter64
CVXQP2_L	10000	2500	*first_order*	8.2e+07	1.3e−10	5.6e−11	5.2e−03	2.6e+02	18	9	9
CVXQP2_M	1000	250	*first_order*	8.2e+05	7.5e−11	3.0e−11	2.6e−05	3.1e−01	15	7	8
CVXQP2_S	100	25	*first_order*	8.1e+03	1.1e−09	2.8e−09	1.5e−06	4.1e−03	8	6	2
CVXQP3_L	10000	7500	*first_order*	1.2e+08	5.1e−10	7.7e−12	1.4e−02	5.2e+02	44	21	23
CVXQP3_M	1000	750	*first_order*	1.4e+06	5.6e−10	1.0e−12	3.9e−04	2.0e+00	140	13	127
CVXQP3_S	100	75	*first_order*	1.2e+04	1.1e−10	6.0e−11	5.7e−07	2.6e−03	13	8	5
DPKLO1	133	77	*first_order*	3.7e−01	8.8e−09	2.0e−06	9.6e−08	2.8e−03	3	2	1
DTOC3	14999	9998	*first_order*	2.4e+02	3.5e−10	9.3e−13	5.6e−13	1.0e−01	10	7	3
DUAL1	85	1	*first_order*	3.5e−02	1.6e−09	1.1e−12	6.8e−07	4.3e−03	9	5	4
DUAL2	96	1	*first_order*	3.4e−02	5.2e−09	1.3e−10	7.1e−07	4.5e−03	7	5	2
DUAL3	111	1	*first_order*	1.4e−01	4.2e−09	4.9e−14	1.7e−06	7.6e−03	9	5	4
DUAL4	75	1	*first_order*	7.5e−01	7.2e−09	1.2e−13	4.8e−10	2.9e−03	8	5	3
DUALC1	9	215	*first_order*	6.2e+03	1.5e−10	4.4e−09	1.7e−05	4.0e−03	12	9	3
DUALC2	7	229	*first_order*	3.6e+03	7.3e−10	2.1e−07	3.4e−06	2.8e−03	8	6	2
DUALC5	8	278	*first_order*	4.3e+02	1.0e−11	8.0e−11	7.3e−09	3.3e−03	7	4	3
DUALC8	8	503	*first_order*	1.8e+04	6.0e−10	1.6e−04	6.3e−02	5.7e−03	7	6	1
EXDATA	3000	3001	*first_order*	−1.4e+02	3.4e−10	1.4e−13	1.2e−08	1.6e+01	13	8	5
GENHS28	10	8	*first_order*	9.3e−01	2.9e−14	1.9e−13	2.4e−13	3.9e−04	4	2	2
GOULDQP2	699	349	*first_order*	1.8e−04	8.9e−10	5.1e−14	2.0e−08	7.5e−03	17	5	12
GOULDQP3	699	349	*first_order*	2.1e+00	8.5e−09	1.5e−12	9.1e−09	6.9e−03	17	9	8
HS118	15	17	*first_order*	6.6e+02	5.0e−11	1.0e−12	6.2e−11	5.8e−04	12	8	4
HS21	2	1	*first_order*	−1.0e+02	1.1e−11	5.1e−10	2.6e−10	2.4e−04	6	4	2
HS268	5	5	*first_order*	1.1e−11	1.5e−11	1.8e−14	7.3e−12	4.4e−04	26	21	5
HS35	3	1	*first_order*	1.1e−01	1.1e−10	7.6e−11	3.0e−10	2.4e−04	7	5	2
HS35MOD	3	1	*first_order*	2.5e−01	2.5e−09	2.8e−12	1.9e−12	2.6e−04	12	6	6
HS51	5	3	*first_order*	−8.9e−16	3.3e−13	2.0e−13	1.7e−13	2.2e−04	4	2	2
HS52	5	3	*first_order*	5.3e+00	6.1e−09	7.5e−10	1.3e−07	2.1e−04	3	2	1
HS53	5	3	*first_order*	4.1e+00	3.2e−09	2.0e−13	3.5e−10	2.4e−04	4	2	2
HS76	4	3	*first_order*	−4.7e+00	1.5e−10	1.3e−10	1.8e−10	2.7e−04	6	4	2
HUES-MOD	10000	2	*first_order*	3.5e+07	1.3e−09	3.9e−12	9.4e−08	3.5e−02	9	6	3
HUESTIS	10000	2	*first_order*	3.5e+11	9.8e−12	1.0e−11	5.0e−04	4.1e−02	11	6	5
KSIP	20	1001	*first_order*	5.8e−01	7.6e−10	2.2e−12	1.8e−08	4.1e−02	13	6	7
LASER	1002	1000	*first_order*	2.4e+06	1.7e−11	1.6e−09	4.3e−07	2.4e−02	36	5	31
LISWET1	10002	10000	*first_order*	3.6e+01	2.4e−10	8.9e−16	1.5e−09	2.2e−01	32	9	23
LISWET10	10002	10000	*first_order*	4.9e+01	4.6e−09	1.0e−15	8.3e−10	2.7e−01	37	9	28
LISWET11	10002	10000	*first_order*	5.0e+01	5.5e−10	2.7e−14	4.1e−07	2.8e−01	37	9	28
LISWET12	10002	10000	*first_order*	1.7e+03	6.0e−09	2.0e−15	1.9e−09	2.8e−01	42	9	33
LISWET2	10002	10000	*first_order*	2.5e+01	4.4e−09	1.0e−13	5.8e−08	2.1e−01	31	9	22
LISWET3	10002	10000	*first_order*	2.5e+01	6.6e−09	1.3e−09	2.7e−06	1.7e−01	23	9	14
LISWET4	10002	10000	*first_order*	2.5e+01	4.3e−10	9.4e−11	7.1e−07	1.8e−01	25	9	16
LISWET5	10002	10000	*first_order*	2.5e+01	8.3e−10	1.5e−10	5.6e−07	1.7e−01	22	9	13
LISWET6	10002	10000	*first_order*	2.5e+01	7.2e−12	3.7e−12	6.2e−08	1.8e−01	25	9	16
LISWET7	10002	10000	*first_order*	5.0e+02	1.0e−10	5.6e−16	1.9e−09	2.0e−01	28	9	19
LISWET8	10002	10000	*first_order*	7.1e+02	7.8e−09	3.9e−14	1.7e−07	2.4e−01	36	9	27
LISWET9	10002	10000	*first_order*	2.0e+03	5.1e−09	2.2e−15	1.9e−09	2.9e−01	38	9	29
LOTSCHD	12	7	*first_order*	2.4e+03	1.0e−10	2.9e−10	2.1e−10	6.6e−04	8	5	3
MOSARQP1	2500	700	*first_order*	−9.5e+02	1.0e−09	1.0e−09	2.5e−06	2.2e−02	9	6	3
MOSARQP2	900	600	*first_order*	−1.6e+03	7.5e−09	2.4e−11	8.0e−06	2.0e−02	10	5	5
POWELL20	10000	10000	*first_order*	5.2e+10	1.9e−11	1.7e−10	6.1e−08	3.0e−01	31	23	8
PRIMAL1	325	85	*first_order*	−3.5e−02	2.0e−09	2.7e−12	3.3e−08	1.0e−02	9	6	3
PRIMAL2	649	96	*first_order*	−3.4e−02	2.0e−11	9.8e−14	1.1e−10	1.4e−02	8	4	4
PRIMAL3	745	111	*first_order*	−1.4e−01	6.9e−10	5.0e−12	1.4e−08	4.5e−02	10	5	5
PRIMAL4	1489	75	*first_order*	−7.5e−01	6.6e−09	2.5e−12	3.2e−12	3.2e−02	10	5	5
PRIMALC1	230	9	*first_order*	−6.2e+03	7.5e−09	1.1e−07	5.6e−08	2.4e−03	10	7	3
PRIMALC2	231	7	*first_order*	−3.6e+03	9.7e−10	3.6e−05	3.4e−07	1.8e−03	8	6	2
PRIMALC5	287	8	*first_order*	−4.3e+02	2.0e−09	2.5e−09	4.1e−09	3.0e−03	12	6	6
PRIMALC8	520	8	*first_order*	−1.8e+04	1.1e−11	1.6e−07	3.1e−10	4.2e−03	9	6	3
Q25FV47	1571	820	*first_order*	1.4e+07	2.8e−09	2.4e−06	1.5e−02	5.2e−01	26	14	12
QADLITTL	97	56	*first_order*	4.8e+05	9.0e−09	6.5e−06	5.6e−05	1.7e−03	13	11	2
QAFIRO	32	27	*first_order*	−1.6e+00	2.7e−10	3.9e−09	2.4e−10	5.4e−04	10	8	2
QBANDM	472	305	*first_order*	1.6e+04	2.6e−09	1.4e−08	1.4e−04	1.0e−02	21	13	8
QBEACONF	262	173	*first_order*	1.6e+05	6.4e−11	1.9e−09	5.5e−09	5.7e−03	19	7	12
QBORE3D	315	233	*first_order*	3.1e+03	1.4e−09	3.8e−10	1.5e−05	9.9e−03	68	12	56
QBRANDY	249	220	*first_order*	2.8e+04	1.5e−10	1.2e−08	5.1e−07	7.8e−03	16	13	3
QCAPRI	353	271	*first_order*	6.7e+07	9.1e−09	8.1e−07	1.2e−05	1.6e−02	35	7	28
QE226	282	223	*first_order*	2.1e+02	1.4e−09	4.1e−10	8.3e−08	1.2e−02	20	15	5
QETAMACR	688	400	*first_order*	8.7e+04	5.6e−09	1.0e−09	4.1e−04	5.9e−02	39	12	27
QFFFFF80	854	524	*first_order*	8.7e+05	6.1e−11	1.4e−07	1.5e−06	8.2e−02	34	11	23
QFORPLAN	421	161	*first_order*	7.5e+09	9.5e−09	8.0e−07	5.7e−02	1.5e−02	25	7	18
QGFRDXPN	1092	616	*first_order*	1.0e+11	8.8e−10	4.1e−06	3.7e−07	1.4e−02	21	5	16
QGROW15	645	300	*first_order*	−1.0e+08	1.2e−09	2.7e−06	1.4e−01	4.6e−02	53	13	40
QGROW22	946	440	*first_order*	−1.5e+08	7.5e−09	1.8e−05	2.4e−01	1.1e−01	77	33	44
QGROW7	301	140	*first_order*	−4.3e+07	6.1e−09	1.3e−05	1.4e−01	3.1e−02	76	36	40
QISRAEL	142	174	*first_order*	2.5e+07	8.9e−09	2.8e−07	2.1e−03	1.1e−02	25	10	15
QPCBLEND	83	74	*first_order*	−7.8e−03	1.7e−09	1.4e−12	1.0e−06	2.0e−03	16	5	11
QPCBOEI1	384	351	*first_order*	1.2e+07	9.0e−09	1.2e−07	5.4e−03	1.9e−02	27	5	22
QPCBOEI2	143	166	*first_order*	8.2e+06	3.2e−09	3.2e−08	1.7e−02	1.2e−02	49	6	43
QPCSTAIR	467	356	*first_order*	6.2e+06	2.5e−11	3.3e−10	4.1e−04	2.3e−02	24	12	12
QPILOTNO	2172	975	*unknown*	4.7e+06	1.5e−06	4.5e−03	1.2e−06	2.8e−01	42	5	37
QPTEST	2	2	*first_order*	4.4e+00	8.4e−10	4.6e−14	3.0e−12	3.9e−04	10	5	5
QRECIPE	180	91	*first_order*	−2.7e+02	1.7e−09	5.8e−08	2.3e−10	2.2e−03	15	9	6
QSC205	203	205	*first_order*	−5.8e−03	2.4e−10	6.8e−13	1.2e−11	2.5e−03	11	7	4
QSCAGR25	500	471	*first_order*	2.0e+08	8.3e−10	3.3e−08	3.0e−06	6.3e−03	17	11	6
QSCAGR7	140	129	*first_order*	2.7e+07	4.2e−12	1.8e−08	9.9e−07	2.1e−03	18	10	8
QSCFXM1	457	330	*first_order*	1.7e+07	2.6e−09	1.9e−07	2.4e−04	1.5e−02	24	13	11
QSCFXM2	914	660	*first_order*	2.8e+07	7.1e−09	4.0e−06	1.0e−04	4.0e−02	33	16	17
QSCFXM3	1371	990	*first_order*	3.1e+07	1.1e−09	3.5e−06	1.2e−04	5.3e−02	29	10	19
QSCORPIO	358	388	*first_order*	1.9e+03	1.4e−10	4.0e−13	1.5e−08	4.2e−03	13	5	8
QSCRS8	1169	490	*first_order*	9.0e+02	4.1e−09	8.3e−10	7.6e−07	2.0e−02	31	10	21
QSCSD1	760	77	*first_order*	8.7e+00	1.6e−10	7.2e−11	6.8e−10	4.9e−03	8	6	2
QSCSD6	1350	147	*first_order*	5.1e+01	4.1e−11	2.1e−10	4.6e−08	1.2e−02	13	11	2
QSCSD8	2750	397	*first_order*	9.4e+02	4.8e−10	2.0e−11	8.9e−07	2.2e−02	11	8	3
QSCTAP1	480	300	*first_order*	1.4e+03	4.0e−09	1.0e−07	1.3e−07	7.6e−03	17	15	2
QSCTAP2	1880	1090	*first_order*	1.7e+03	3.3e−09	3.1e−08	8.1e−08	3.9e−02	14	12	2
QSCTAP3	2480	1480	*first_order*	1.4e+03	6.1e−09	1.4e−07	2.7e−07	5.1e−02	16	14	2
QSEBA	1028	515	*first_order*	8.1e+07	3.5e−09	1.1e−09	3.2e−04	1.7e−02	25	5	20
QSHARE1B	225	117	*first_order*	7.2e+05	4.1e−11	3.9e−08	8.6e−08	1.1e−02	63	29	34
QSHARE2B	79	96	*first_order*	1.2e+04	4.5e−09	1.5e−07	1.1e−04	4.2e−03	30	8	22
QSHELL	1775	536	*first_order*	1.6e+12	9.2e−09	4.1e−07	5.6e−01	3.8e−02	30	9	21
QSHIP04L	2118	402	*first_order*	2.4e+06	6.8e−09	4.8e−11	3.5e−06	1.6e−02	16	8	8
QSHIP04S	1458	402	*first_order*	2.4e+06	1.5e−10	3.6e−12	2.1e−07	1.2e−02	18	9	9
QSHIP08L	4283	778	*first_order*	2.4e+06	4.9e−10	1.5e−11	1.2e−04	4.9e−01	19	10	9
QSHIP08S	2387	778	*first_order*	2.4e+06	6.0e−11	3.8e−11	8.3e−06	6.5e−02	18	9	9
QSHIP12L	5427	1151	*first_order*	3.0e+06	1.1e−09	8.7e−12	3.3e−04	1.1e+00	39	8	31
QSHIP12S	2763	1151	*first_order*	3.1e+06	3.5e−10	1.1e−11	2.9e−04	1.0e−01	35	10	25

Continued on next page

Name	n	m	status	objective	pdd	pfeas	dfeas	time (s)	iter tot	iter32	iter64
QSIERRA	2036	1227	*first_order*	2.4e+07	5.6e−11	2.2e−06	1.6e−06	6.7e−02	34	10	24
QSTAIR	467	356	*first_order*	8.0e+06	7.0e−09	8.2e−06	3.4e−03	3.2e−02	28	13	15
QSTANDAT	1075	359	*first_order*	6.4e+03	8.0e−09	1.6e−07	2.2e−06	8.3e−03	11	9	2
S268	5	5	*first_order*	1.1e−11	1.5e−11	1.8e−14	7.3e−12	4.5e−04	26	21	5
STADAT1	2001	3999	*first_order*	−2.9e+07	2.1e−09	1.0e−07	1.1e−06	1.5e−01	75	5	70
STADAT2	2001	3999	*first_order*	−3.3e+01	2.3e−09	4.6e−10	2.8e−11	4.9e−02	19	5	14
STADAT3	4001	7999	*first_order*	−3.6e+01	1.1e−10	2.0e−10	4.1e−11	1.1e−01	22	7	15
STCQP1	4097	2052	*first_order*	1.6e+05	1.2e−09	0.0e+00	3.6e−05	1.4e−02	8	5	3
STCQP2	4097	2052	*first_order*	2.2e+04	5.1e−10	0.0e+00	3.4e−06	2.9e−02	9	6	3
TAME	2	1	*first_order*	0.0e+00	5.1e−11	3.8e−14	7.8e−13	2.8e−04	4	2	2
UBH1	18009	12000	*max_iter*	NaN	NaN	NaN	NaN	5.0e+00	800	40	760
VALUES	202	1	*first_order*	−1.4e+00	6.7e−10	2.5e−10	8.8e−09	4.9e−03	12	10	2
YAO	2002	2000	*first_order*	2.0e+02	6.6e−10	9.1e−15	5.2e−10	3.2e−02	23	10	13
ZECEVIC2	2	2	*first_order*	−4.1e+00	4.6e−10	2.6e−11	9.5e−10	3.3e−04	7	5	2

References

A. Altman and J. Gondzio. Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization. Optim. Method Softw., 11(1–4):275–302, Jan. 1999.

P. Amestoy, A. Buttari, N. Higham, J.-Y. L'Excellent, T. Mary, and B. Vieuble. Five-Precision GMRES-based iterative refinement. Technical Report hal-03190686, 2021.

P. Amestoy, A. Buttari, N. J. Higham, J.-Y. L'Excellent, T. Mary, and B. Vieublé. Combining sparse approximate factorizations with mixed-precision iterative refinement. ACM Trans. Math. Software, 49(1), Mar 2023.

M. Arioli and I. Duff. Using FGMRES to obtain backward stability in mixed precision. ETNA, 33:31–44, 2008.

M. Arioli, J. W. Demmel, and I. S. Duff. Solving Sparse Linear Systems with Sparse Backward Error. SIAM J. Matrix Anal. Appl., 10(2):165–190, Apr. 1989.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A Fresh Approach to Numerical Computing. SIAM Rev., 59(1):65–98, Jan. 2017.

Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. ACM Trans. Math. Software, 35(3), oct 2008.

I. S. Duff. Ma57—a code for the solution of sparse symmetric definite and indefinite systems. ACM Trans. Math. Software, 30(2):118–144, jun 2004.

FICO Xpress Optimization. Xpress solver. https://www.fico.com/en/products/fico-xpress-solver.

A. Forsgren. Inertia-controlling factorizations for optimization algorithms. 43(1):91–107.

M. P. Friedlander and D. Orban. A primal-dual regularized interior-point method for convex quadratic programs. Math. Program. Comp., 4(1):71–107, 2012.

A. Ghannad, D. Orban, and M. A. Saunders. Linear systems arising in interior methods for convex optimization: A symmetric formulation with bounded condition number. Optim. Method Softw., pages 1–26, Oct. 2021.

J. Gondzio. Multiple centrality corrections in a primal-dual method for linear programming. Comput. Optim. Appl., 6(2):137–156, Sept. 1996.

N. Gould and P. L. Toint. Preprocessing for quadratic programming. Math. Program., 100(1), May 2004.

C. Greif, E. Moulding, and D. Orban. Bounds on Eigenvalues of Matrices Arising from Interior-Point Methods. SIAM J. Optim., 24(1):49–83, Jan. 2014.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. https://www.gurobi.com, 2023.

A. Haidar, H. Bayraktar, S. Tomov, J. Dongarra, and N. J. Higham. Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 476(2243):20200110, Nov. 2020.

J. Hogg and J. A. Scott. Hsl_ma97 : a bit-compatible multifrontal code for sparse symmetric systems. Rutherford Appleton Laboratory Technical Reports, 2011.

IBM ILOG CPLEX Optimization Studio. V12: User's Manual for CPLEX. https://www.ibm.com/products/ilog-cplex-optimization-studio.

M. Kojima, N. Megiddo, and S. Mizuno. A primal—dual infeasible-interior-point algorithm for linear programming. 61(1):263–280, 1993.

D. Ma, L. Yang, R. M. T. Fleming, I. Thiele, B. O. Palsson, and M. A. Saunders. Reliable and efficient solution of genome-scale models of Metabolism and macromolecular Expression. Scientific Reports, 7(1):40863, Feb. 2017.

I. Maros and C. Mészáros. A repository of convex quadratic programming problems. Optim. Method Softw., 11:671–681, 1999.

S. Mehrotra. On the Implementation of a Primal-Dual Interior Point Method. SIAM J. Optim., 2(4):575–601, Nov. 1992.

Netlib. The Netlib linear problems dataset. https://netlib.org/lp/.

J. Nocedal and S. J. Wright. Numerical Optimization. Springer Series in Operations Research. Springer, New York, 2nd edition, 2006.

D. Orban. Limited-memory LDL factorization of symmetric quasi-definite matrices with application to constrained optimization. Numer. Algor., 70(1):9–41, Sept. 2015.

D. Orban and M. Arioli. Iterative Solution of Symmetric Quasi-Definite Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, Apr. 2017.

D. Orban and contributors. LDLFactorizations.jl: Factorization of symmetric matrices. June 2020.

D. Orban and contributors. HSL.jl: A Julia interface to the HSL mathematical software library. https://github.com/JuliaSmoothOptimizers/HSL.jl, March 2021a.

D. Orban and contributors. LimitedLDLFactorizations.jl: Limited-memory LDL factorization. https://github.com/JuliaSmoothOptimizers/LimitedLDLFactorizations.jl, January 2021b.

D. Orban and G. Leconte. RipQP.jl: Regularized interior point solver for quadratic problems. Nov. 2022.

D. Orban, A. S. Siqueira, and contributors. LinearOperators.jl. https://github.com/JuliaSmoothOptimizers/LinearOperators.jl, September 2020a.

D. Orban, A. S. Siqueira, and contributors. QuadraticModels.jl: Data structures for linear and quadratic optimization problems based on NLPModels.jl. https://github.com/JuliaSmoothOptimizers/QuadraticModels.jl, December 2020b.

D. Ruiz. A Scaling Algorithm to Equilibrate Both Rows and Columns Norms in Matrices. Technical Report RAL-TR-2001-034, Rutherford Appleton Lab., Oxon, UK, 2001.

M. A. Saunders and L. Tenenblat. The Zoom strategy for accelerating and warm-starting interior methods, 2006.

M. A. Saunders and J. A. Tomlin. Solving regularized linear programs using barrier methods and KKT systems, volume 10064. T. J. Watson Research Center, 1996.

B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. Math. Program. Comp., 12(4):637–672, 2020.

M. Tanneau, D. Orban, and contributors. QPSReader.jl: A reader for linear and quadratic optimization problems in MPS, QPS, and SIF formats. https://github.com/JuliaSmoothOptimizers/QPSReader.jl, December 2020.

M. Tanneau, M. F. Anjos, and A. Lodi. Design and implementation of a modular interior-point solver for linear optimization. Math. Program. Comp., Feb. 2021.

R. Vanderbei and D. Shanno. An interior-point algorithm for nonconvex nonlinear programming. Comput. Optim. Appl., 13:231–252, Jan. 1999.

T. Weber, S. Sager, and A. Gleixner. Solving quadratic programs to high precision using scaled iterative refinement. Math. Program. Comp., 11(3):421–455, Sept. 2019.

S. J. Wright. Primal-Dual Interior-Point Methods. SIAM, 1997.