

# QOCO: A Quadratic Objective Conic Optimizer with Custom Solver Generation

Govind M. Chari\*, Behçet Açıkmeşe†

University of Washington, Seattle, WA 98195, USA

August 19, 2025

## Abstract

Second-order cone programs (SOCPs) with quadratic objective functions are common in optimal control and other fields. Most SOCP solvers which use interior-point methods are designed for linear objectives and convert quadratic objectives into linear ones via slack variables and extra constraints, despite the computational advantages of handling quadratic objectives directly. In applications like model-predictive control and online trajectory optimization, these SOCPs have known sparsity structures and require rapid solutions. When solving these problems, most solvers use sparse linear algebra routines, which introduce computational overhead and hinder performance. In contrast, custom linear algebra routines can exploit the known sparsity structure of problem data and be significantly faster. This work makes two key contributions: (1) the development of QOCO, an open-source C-based solver for quadratic objective SOCPs, and (2) the introduction of QOCOGEN, an open-source custom solver generator for quadratic objective SOCPs, which generates a solver written in C that leverages custom linear algebra. Both implement a primal-dual interior-point method with Mehrotra's predictor-corrector. Our benchmarks show that QOCO is faster and more robust than many commonly used solvers, and solvers generated by QOCOGEN are significantly faster than QOCO and are free of dynamic memory allocation making them an attractive option for real-time optimization on resource-constrained embedded systems.

## 1 Introduction

We consider the quadratic objective second-order cone program (SOCP)

$$\begin{aligned} & \underset{x}{\text{minimize}} && \frac{1}{2}x^\top Px + c^\top x \\ & \text{subject to} && Gx \preceq_{\mathcal{K}} h \\ & && Ax = b, \end{aligned} \tag{1}$$

with optimization variable  $x \in \mathbb{R}^n$ . The cost is defined by the positive semidefinite matrix  $P = P^\top \succeq 0$  and vector  $c \in \mathbb{R}^n$ . The constraints are defined by matrices  $A \in \mathbb{R}^{p \times n}$  and  $G \in \mathbb{R}^{m \times n}$  and vectors  $b \in \mathbb{R}^p$  and  $h \in \mathbb{R}^m$ . The generalized inequality  $\preceq_{\mathcal{K}}$  denotes membership in a closed, proper cone  $\mathcal{K}$ , i.e.  $h - Gx \in \mathcal{K}$ . We restrict  $\mathcal{K}$  to be the Cartesian product

$$\mathcal{K} := \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_K,$$

where  $\mathcal{C}_i$  is either the non-negative orthant or a second-order cone. Each cone  $\mathcal{C}_i$  corresponds to a subset of constraints, inducing a partition on  $G$  and  $h$ , i.e.,  $h_i - G_i x \in \mathcal{C}_i$  for  $i = 1, \dots, K$ . Throughout this work, we assume that Problem 1 is **feasible** and has a **bounded** optimal objective.

---

\*Ph.D. Student, William E. Boeing Department of Aeronautics & Astronautics; [gchari@uw.edu](mailto:gchari@uw.edu)

†Professor, William E. Boeing Department of Aeronautics & Astronautics; [behcet@uw.edu](mailto:behcet@uw.edu)

Problem 1 appears in various applications, including model predictive control (MPC) [1], network flow optimization [2], portfolio optimization [3, 4], robust optimization [5, 6], and filter design [7], among others. Additionally, the solution of SOCPs is used as a subroutine in algorithms for nonconvex optimization such as sequential convex programming (SCP) [8, 9, 10, 11].

Problem 1 can be reformulated as a SOCP with a linear objective by introducing a slack variable  $t$ ,

$$\begin{aligned} & \underset{x,t}{\text{minimize}} \quad t + c^\top x \\ & \text{subject to} \quad \left\| \begin{bmatrix} t - \frac{1}{2} \\ P^{1/2}x \end{bmatrix} \right\|_2 \leq t + \frac{1}{2} \\ & \quad Gx \preceq_{\mathcal{K}} h \\ & \quad Ax = b. \end{aligned}$$

If  $P$  is large (has many rows and columns), then computing  $P^{1/2}$  can be prohibitively expensive, making this reformulation impractical. Solving this reformulation can also be slow if the matrix square root  $P^{1/2}$  has significantly more nonzero elements than  $P$ . Thus it is advantageous for a SOCP solver to natively handle quadratic objective functions without resorting to the linear objective reformulation.

Some SOCP solvers that can solve Problem 1 directly include CLARABEL [12], GUROBI [13], COSMO [14], and SCS [15]. However, COSMO and SCS implement operator-splitting methods. These methods are preferred for large-scale problems due to their low per-iteration cost but can struggle with problem scaling, conditioning, and achieving high-accuracy solutions [16]. For modestly sized SOCPs, an interior-point method (IPM) is preferred, due to its robustness to scaling and conditioning of the problem, and ability to converge to high accuracy solutions [17, Chapter 1]. One of the few open-source IPMs that can directly solve Problem 1 is CLARABEL. In addition to solving SOCPs, CLARABEL can also handle semidefinite programs, as well as problems involving exponential cones, power cones, and generalized power cones. Additionally, it supports infeasibility detection. However, CLARABEL is written in Rust. Although Rust is memory-safe due to its ownership system and can be used in embedded systems, its ecosystem is far less mature than that of C, and for legacy systems, such as aerospace software, C is still preferred.

Many engineering applications including linear MPC, nonlinear MPC [18], portfolio backtesting, sequential quadratic programming (SQP) [19, 20], and SCP, require solving SOCPs with a fixed, known sparsity structures in  $P, A, G$  and a constant cone  $\mathcal{K}$ . These problems often arise in real-time settings, where computational efficiency is critical. For example, sequential convex programming (SCP) iteratively solves a sequence of SOCPs with quadratic objectives and identical sparsity patterns to find stationary points of nonconvex problems. SCP has been widely applied in aerospace trajectory optimization, where real-time performance is essential. Notable applications include powered-descent guidance [21, 22, 23], in-space rendezvous [24, 25], and hypersonic entry guidance [26, 27]. SCP is also the solution method used by NASA's SPLICE program for lunar landing guidance [28, 29, 30, 31].

Most general-purpose SOCP solvers, such as CLARABEL [12], COSMO [14], ECOS [32], GUROBI [13], MOSEK [33], and SCS rely on sparse linear algebra routines. Although sparse linear algebra allows the aforementioned solvers to solve SOCPs regardless of their sparsity structures, it can be quite slow due to the extra overhead of determining where the nonzero elements are before performing floating point operations, which leads to more CPU instructions being issued, more memory accesses, and more cache misses. However, when the sparsity structure of the problem matrices is known beforehand, it is possible to generate a custom solver that implements customized linear algebra routines tailored to the sparsity structure of the problem. Specifically, in sparse linear algebra matrices are stored as three arrays: a row/column index array, column/row pointer array, and data array. Before performing a floating point operation on a nonzero element in the matrix, sparse linear algebra routines must first index into the pointer and index arrays to determine the location of the element in the data array. In customized linear algebra, the index and pointer arrays do not exist, and we directly index into the data array, eliminating the overhead of sparse linear algebra, resulting in significantly faster operations.

We discuss this further in Section 4.1.

A *custom solver generator* is a tool that takes the sparsity structure of an optimization problem as an input and outputs a solver (typically in C) optimized for that specific sparsity structure without relying on sparse linear algebra. Two notable custom solver generators are CVXGEN [34] and BSOCP [35, 36]. CVXGEN, which is academically licensed but not open-source, solves quadratic programs (QPs) rather than SOCPs (i.e.  $\mathcal{K}$  in Problem 1 is the non-negative orthant) and BSOCP, which is neither academically licensed nor open source, solves linear objective SOCPs (i.e.  $P = 0$  in Problem 1). Both have demonstrated substantial speed improvements over general-purpose solvers and have had a significant impact on aerospace trajectory optimization. For instance, after the development of lossless convexification [37, 38], BSOCP was flight-tested on Masten’s Xombie vehicle to validate G-FOLD, a powered-descent guidance algorithm based on lossless convexification [39]. Later, CVXGEN was used by SpaceX for landing their Falcon 9 boosters [40].

A seemingly related but distinct tool is CVXPYGEN [41]. It leverages CVXPY to parse the optimization problem to the standard form for solvers and generates C code for updating problem data and retrieving solutions. CVXPYGEN then wraps a backend solver without (or with very little) modification to its source code. Because it customizes only the parsing and solution retrieval while relying entirely on an existing solver, we do not classify CVXPYGEN as a custom solver generator.

## 1.1 Contribution

This work makes two key contributions: (1) the development of QOCO, an open-source C-based solver for quadratic objective SOCPs, and (2) the introduction of QOCOGEN, an open-source custom solver generator for quadratic objective SOCPs which generates a custom solver written in C<sup>1</sup>.

We demonstrate that QOCO outperforms most commercial and open-source solvers in terms of both speed and robustness, and QOCO<sub>custom</sub> (the name of solvers generated by QOCOGEN) is significantly faster than QOCO. Both solvers are easy to use since QOCO and QOCOGEN can be called from CVXPY [42, 43] and CVXPYGEN [41] respectively, allowing users to formulate optimization problems in a natural way following from math rather than manually converting the problem to the solver-required standard form. Additionally, QOCO can be called from C/C++, Matlab, and Python, and QOCOGEN can be called from Python. Both implement a primal-dual interior point method with Mehrotra’s predictor-corrector [44]. However, they both use a variety of numerical enhancements to quickly and robustly solve the linear system that arises in the step direction computation. Specifically, we use an  $LDL^\top$  factorization along with the Approximate Minimum Degree (AMD) [45, 46] heuristic to permute the coefficient matrix, minimizing fill-in for the factor  $L$ . We also apply static and dynamic regularization to the coefficient matrix to ensure that the matrix is invertible and the factorization succeeds. After solving the linear system, we apply iterative refinement to ensure that the original, unregularized system was solved to high accuracy, further enhancing numerical stability and robustness.

## 1.2 Outline

Section 2 introduces the primal-dual interior point method implemented in QOCO and QOCOGEN. Section 3 explains the techniques used to ensure a stable factorization of the KKT matrix, which is essential to compute the step direction. Section 4 discusses the advantages of custom linear algebra over sparse linear algebra, and how QOCOGEN generates custom solvers which exploit the known sparsity structure of problem data. Finally, Section 5 presents extensive numerical results comparing QOCO and QOCO<sub>custom</sub> to existing solvers.

## 1.3 Notation

We use  $(a, b)$  to denote the concatenation of vectors  $a \in \mathbb{R}^a$  and  $b \in \mathbb{R}^b$ , resulting in a vector in  $\mathbb{R}^{a+b}$ . The non-negative orthant in  $\mathbb{R}^l$  is denoted by  $\mathbb{R}_+^l = \{u \in \mathbb{R}^l \mid u_i \geq 0\}$ , and for any  $u \in \mathbb{R}_+^l$ , we use  $u_i$  to refer to its  $i^{\text{th}}$  element. The  $q$ -dimensional second-order cone is defined as  $\mathcal{Q}^q = \{(u_0, u_1) \in \mathbb{R} \times \mathbb{R}^{q-1} \mid$

---

<sup>1</sup>Both are available on GitHub at <https://github.com/qoco-org>

$\|u_1\|_2 \leq u_0\}$ , where any  $u \in \mathcal{Q}^q$  is written as  $u = (u_0, u_1)$  with  $u_0 \in \mathbb{R}$  and  $u_1 \in \mathbb{R}^{q-1}$ . Finally, when a vector  $x$  lies in the Cartesian product of cones, i.e.,  $x \in \mathcal{C}_1 \times \mathcal{C}_2 \times \dots \times \mathcal{C}_K$ , we denote  $x_i$  as its  $i^{\text{th}}$  subvector belonging to cone  $\mathcal{C}_i$ .

## 2 Primal-dual interior point method

In this section, we describe the primal-dual interior point algorithm we implement in QOCO and QOCO-GEN, as outlined in Algorithm 1. This algorithm is equivalent to the `coneqp` algorithm outlined in [47] and follows a derivation similar to those presented in [48, Chapter 6], [49, 50].

If strong duality holds for Problem 1, the optimal primal-dual solution  $(x^*, s^*, y^*, z^*)$  satisfies the Karush-Kuhn-Tucker (KKT) conditions [51, Chapter 5], which can be written as

$$Px + c + A^\top y + G^\top z = 0 \quad (2a)$$

$$Ax = b \quad (2b)$$

$$Gx + s = h \quad (2c)$$

$$s_i^\top z_i = 0 \text{ for } i = 1, \dots, K \quad (2d)$$

$$(s, z) \in \mathcal{K} \times \mathcal{K}. \quad (2e)$$

The primal-dual interior point method applies a modified Newton's method to Equations (2a) - (2d), and a line search to satisfy Equation (2e). The modifications to Newton's method correct for linearization errors in the Newton step and bias the search directions towards the interior of  $\mathcal{K}$ . This prevents the iterates from prematurely approaching the boundary of the cone  $\mathcal{K}$ , enabling longer steps without violating Equation (2e). It also ensures iterates do not converge to spurious solutions that satisfy Equations (2a) - (2d), but not Equation (2e) [50].

### 2.1 Central path derivation

To understand how the algorithm biases the search directions towards the interior of  $\mathcal{K}$ , we first introduce the concept of the *central path*.

Problem 1 can be equivalently rewritten as

$$\begin{aligned} & \underset{x}{\text{minimize}} && \frac{1}{2} x^\top Px + c^\top x + \mathcal{I}_{\mathcal{K}}(h - Gx) \\ & \text{subject to} && Ax = b, \end{aligned}$$

where  $\mathcal{I}_{\mathcal{K}}$ , the indicator function of the cone  $\mathcal{K}$ , is defined as

$$\mathcal{I}_{\mathcal{K}}(u) = \begin{cases} 0, & \text{for } u \in \mathcal{K} \\ \infty, & \text{otherwise.} \end{cases}$$

We then replace the indicator function,  $\mathcal{I}_{\mathcal{K}}(u)$ , with a smooth barrier function,  $\phi_{\mathcal{K}}(u)$ , which approaches  $\infty$  as its argument approaches the boundary of the cone.

We use the barrier function

$$\phi_{\mathcal{K}}(u) = \sum_{i=1}^K \phi_i(u_i),$$

where  $\phi_i$  is the barrier function for  $\mathcal{C}_i$ . The barrier function for the non-negative orthant and the second-order cone are

$$\phi_i(u) = \begin{cases} -\sum_{j=1}^l \log u_j, & \text{for } \mathcal{C}_i = \mathbb{R}_+^l \\ -(1/2) \log(u_0^2 - u_1^\top u_1), & \text{for } \mathcal{C}_i = \mathcal{Q}^q, \end{cases}$$

where  $\log$  is the natural logarithm.

Their gradients are

$$\nabla \phi_i(u) = \begin{cases} (-1/u_1, \dots, -1/u_l), & \text{for } \mathcal{C}_i = \mathbb{R}_+^l \\ -(u^\top J u)^{-1} J u, & \text{for } \mathcal{C}_i = \mathcal{Q}^q, \end{cases} \quad (3)$$

where

$$J = \begin{bmatrix} 1 & 0 \\ 0 & -I_{q-1} \end{bmatrix}.$$

After replacing the indicator function with the barrier function, we obtain

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2} x^\top P x + c^\top x + \tau \sum_{i=1}^K \phi_i(h_i - G_i x) \\ \text{subject to} \quad & Ax = b, \end{aligned} \quad (4)$$

where  $\tau > 0$  is a scalar. Denoting the optimal solution of Problem 4 as  $x^*(\tau)$ , it can be shown that as  $\tau \rightarrow 0$ ,  $x^*(\tau) \rightarrow x^*$ .

The KKT condition for Problem 4 are

$$\begin{aligned} Px + c + A^\top y - \tau \sum_{i=1}^K G_i^\top \nabla \phi_i(h_i - G_i x) &= 0 \\ Ax &= b \\ h - Gx &\in \mathcal{K}. \end{aligned}$$

If we define  $s = h - Gx$  and  $z_i = -\tau \nabla \phi_i(h_i - G_i x)$  for  $i = 1, \dots, K$ , we can rewrite the KKT conditions as

$$Px + c + A^\top y + G^\top z = 0 \quad (5a)$$

$$Ax = b \quad (5b)$$

$$Gx + s = h \quad (5c)$$

$$z_i = -\tau \nabla \phi_i(s_i) \text{ for } i = 1, \dots, K \quad (5d)$$

$$(s, z) \in \mathcal{K} \times \mathcal{K}, \quad (5e)$$

where the condition  $z \in \mathcal{K}$  arises because if  $u \in \mathcal{K}$ , then  $-\nabla \phi(u) \in \mathcal{K}$  [47].

It is desirable to write Equation (5d), in a form where  $s_i$  and  $z_i$  appear symmetrically. To this end, we define the *Jordan product* [52, 49, 47], a commutative and linear operation, for the non-negative orthant and second-order cone as

$$u \circ v = \begin{cases} (u_1 v_1, \dots, u_l v_l), & \text{for } \mathcal{C}_i = \mathbb{R}_+^l \\ (u^\top v, u_0 v_1 + v_0 u_1), & \text{for } \mathcal{C}_i = \mathcal{Q}^q, \end{cases}$$

and the Jordan product for  $\mathcal{K}$  as

$$u \circ v = (u_1 \circ v_1, u_2 \circ v_2, \dots, u_K \circ v_K), \quad (6)$$

where  $u_i, v_i \in \mathcal{C}_i$ .

The identity element  $e_i$  for cone  $\mathcal{C}_i$  is defined as

$$e_i = \begin{cases} (1, 1, \dots, 1), & \text{for } \mathcal{C}_i = \mathbb{R}_+^l \\ (1, 0, \dots, 0), & \text{for } \mathcal{C}_i = \mathcal{Q}^q, \end{cases}$$

and the identity element for  $\mathcal{K}$  is

$$\mathbf{e} = (e_1, \dots, e_K). \quad (7)$$

Taking the Jordan product with  $s_i$  on both sides of Equation (5d) and substituting Equation (3), we obtain

$$Px + c + A^\top y + G^\top z = 0 \quad (8a)$$

$$Ax = b \quad (8b)$$

$$Gx + s = h \quad (8c)$$

$$s_i \circ z_i = \tau e_i \text{ for } i = 1, \dots, K \quad (8d)$$

$$(s, z) \in \mathcal{K} \times \mathcal{K}. \quad (8e)$$

We then rewrite Equation (8d) by stacking  $s_i$  and  $z_i$  and using Equations (6) and (7) to obtain

$$Px + c + A^\top y + G^\top z = 0 \quad (9a)$$

$$Ax = b \quad (9b)$$

$$Gx + s = h \quad (9c)$$

$$s \circ z = \tau \mathbf{e} \quad (9d)$$

$$(s, z) \in \mathcal{K} \times \mathcal{K}. \quad (9e)$$

The *central path* is defined as the trajectory of points parameterized by  $\tau > 0$ , satisfying Equations (9a) - (9e). We denote the central path with the tuple  $(x^*(\tau), s^*(\tau), y^*(\tau), z^*(\tau))$ . From the above analysis, we see that the central path equations given by Equations (9a) - (9e) are equivalent to the optimality conditions for the barrier formulation of Problem 1 given by Problem 4. It can be shown that as  $\tau \rightarrow 0$ ,  $(x^*(\tau), s^*(\tau), y^*(\tau), z^*(\tau)) \rightarrow (x^*, s^*, y^*, z^*)$ .

The primal-dual interior-point method applies Newton's method to move towards the central path rather than directly towards points satisfying Equation (2d). Since the central path lies within the cone, maintaining proximity to it allows the algorithm to take longer steps without violating Equation (2e) [50].

## 2.2 Nesterov-Todd scaling

If Newton's method were applied directly to Equations (5a) - (5d), the coefficient matrix of the resulting linear system would lack symmetry (see Appendix A). As a result, it would be necessary to store the entire matrix, rather than only the upper or lower triangular portion. Additionally, solving

this system would require matrix factorizations for non-symmetric matrices, which are generally more computationally expensive than those for symmetric matrices.

To derive a symmetric linear system, we introduce the following change of variables

$$\tilde{s} = W^{-\top} s, \quad \tilde{z} = Wz,$$

where we choose  $W$  such that the above transformation preserves cone membership and leaves the central path equations unchanged

$$s \in \mathcal{K} \iff \tilde{s} \in \mathcal{K}, \quad z \in \mathcal{K} \iff \tilde{z} \in \mathcal{K}, \quad s \circ z = \tau e \iff \tilde{s} \circ \tilde{z} = \tau e. \quad (10)$$

Using this transformation, the central path equations can be equivalently expressed as

$$Px + c + A^\top y + G^\top z = 0 \quad (11a)$$

$$Ax = b \quad (11b)$$

$$Gx + s = h \quad (11c)$$

$$(W^{-\top} s) \circ (Wz) = \tau e \quad (11d)$$

$$(s, z) \in \mathcal{K} \times \mathcal{K}. \quad (11e)$$

There are many matrices,  $W$ , which satisfy Equation (10). In particular, we use the *Nesterov-Todd scaling* matrix [53, 54]. This scaling matrix is determined based on the current iterates  $s_k$  and  $z_k$  and the unique point  $w$  that satisfies

$$\nabla^2 \phi_{\mathcal{K}}(w) s_k = z_k, \quad (12)$$

where  $\nabla^2 \phi_{\mathcal{K}}(w)$  is the Hessian of the barrier function evaluated at  $w$ . Since the barrier function is strictly convex,  $\nabla^2 \phi_{\mathcal{K}}(w)$  is positive definite.

From this, we compute  $W_k$  as

$$\nabla^2 \phi_{\mathcal{K}}(w)^{-1} = W_k^\top W_k. \quad (13)$$

A key property that follows from Equations (12) and (13) is

$$W_k^{-\top} s_k = W_k z_k. \quad (14)$$

This property allows us to define

$$\lambda_k = W_k^{-\top} s_k = W_k z_k. \quad (15)$$

For more details on how to compute the scaling point  $w$  and the scaling matrix  $W_k$ , see [47].

### 2.3 Computing search directions

Given the current iterate  $(x_k, s_k, y_k, z_k)$  we define the *duality measure* as

$$\mu_k = s_k^\top z_k / m, \quad (16)$$

which represents the average violation of the complementary slackness condition given by Equation (2d). To compute a search direction for updating the current iterate, we apply Mehrotra's predictor-corrector method [44]. This scheme computes a Newton step with three key objectives: reducing the duality measure, maintaining proximity to the central path, and correcting for the linearization error introduced during this step.

Applying Newton's method to Equation (11) first requires a linearization of the central path equations around the current iterate  $(x_k, s_k, y_k, z_k)$ . This is done by substituting  $(x, s, y, z)$  with  $(x_k + \Delta x, s_k + \Delta s, y_k + \Delta y, z_k + \Delta z)$  and ignoring the higher order term  $(W^{-\top} \Delta s) \circ (W \Delta z)$ . This results in a linear system of the form

$$P \Delta x + A^\top \Delta y + G^\top \Delta z = -r_x \quad (17a)$$

$$A \Delta x = -r_y \quad (17b)$$

$$G \Delta x + \Delta s = -r_z \quad (17c)$$

$$(W_k^{-\top} s_k) \circ (W_k \Delta z) + (W_k^{-\top} \Delta s) \circ (W_k z_k) = -r_s. \quad (17d)$$

Here,  $(r_x, r_y, r_z, r_s)$  are residual vectors that will be defined later.

Using Equation (15), we can rewrite the system as

$$P \Delta x + A^\top \Delta y + G^\top \Delta z = -r_x \quad (18a)$$

$$A \Delta x = -r_y \quad (18b)$$

$$G \Delta x + \Delta s = -r_z \quad (18c)$$

$$\lambda_k \circ (W_k \Delta z + W_k^{-\top} \Delta s) = -r_s. \quad (18d)$$

To obtain a symmetric system, we can eliminate  $\Delta s$  and rewrite the equations in matrix form as

$$\begin{bmatrix} P & A^\top & G^\top \\ A & 0 & 0 \\ G & 0 & -W_k^\top W_k \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} -r_x \\ -r_y \\ -r_z + W_k^\top (\lambda_k \setminus r_s) \end{bmatrix} \quad (19a)$$

$$\Delta s = -r_z - G \Delta x, \quad (19b)$$

where the operator  $\setminus$  represents the inverse of the Jordan product  $\circ$ , meaning that  $x \setminus (x \circ y) = y$ .

To account for the higher order term we ignored when forming Equation (17), we apply Mehrotra's predictor-corrector [44]. This method consists of a predictor step, where a search direction is computed by taking a Newton step on Equation (11) with  $\tau = 0$ , followed by a combined step. The combined step incorporates centering and correction terms that will help keep the next iterate close to the central path while compensating for the linearization error incurred by ignoring the  $(W^{-\top} \Delta s) \circ (W \Delta z)$  term when linearizing Equation (11). Note that the predictor step is used to estimate this linearization error. Figure 1 illustrates Mehrotra's predictor-corrector.

To compute the predictor step, also called the *affine scaling direction*,  $(\Delta x_a, \Delta s_a, \Delta y_a, \Delta z_a)$ , we solve Equation (19) with the residual vectors

$$r_x = Px_k + c + A^\top y_k + G^\top z_k \quad (20a)$$

$$r_y = Ax_k - b \quad (20b)$$

$$r_z = Gx_k + s_k - h \quad (20c)$$

$$r_s = \lambda_k \circ \lambda_k. \quad (20d)$$

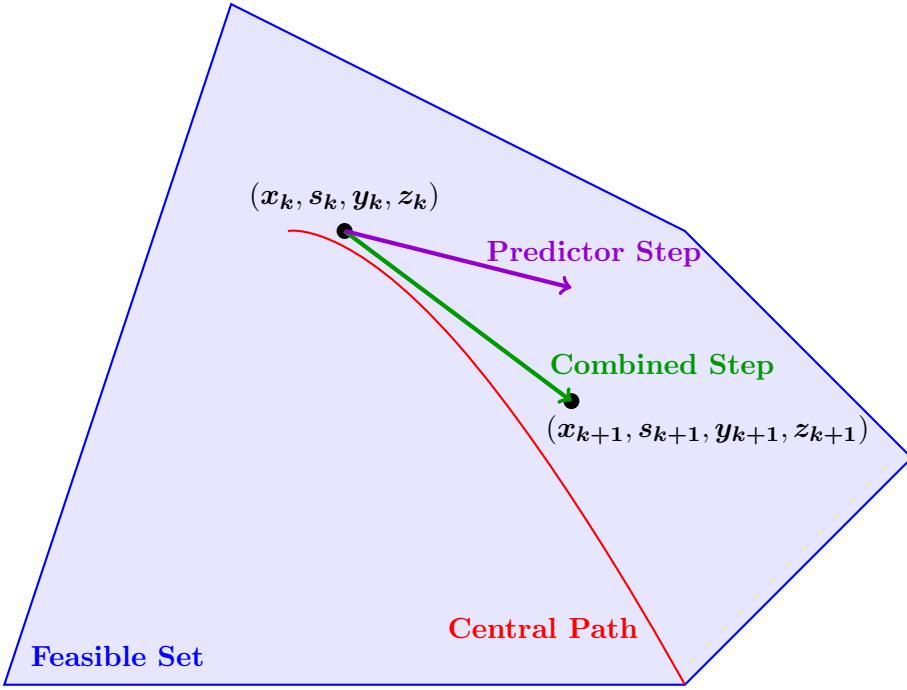


Figure 1: Mehrotra’s predictor-corrector.

Since taking a full step along the affine scaling direction can violate the constraint  $(s, z) \in \mathcal{K} \times \mathcal{K}$ , we compute the centering parameter  $\sigma \in [0, 1]$  as

$$\begin{aligned}\alpha &= \sup\{\alpha \in [0, 1] \mid (s_k, z_k) + \alpha(\Delta s_a, \Delta z_a) \in \mathcal{K} \times \mathcal{K}\} \\ \rho &= \frac{(s_k + \alpha \Delta s_a)^\top (z_k + \alpha \Delta z_a)}{s_k^\top z_k} \\ \sigma &= \max\{0, \min\{1, \rho\}^3\}.\end{aligned}$$

The centering parameter balances the need to reduce the duality measure while maintaining proximity to the central path, allowing for a longer step in the next iteration [50].

Finally, we compute the combined direction, which includes both predictor and corrector information. This direction,  $(\Delta x, \Delta s, \Delta y, \Delta z)$  is obtained by solving Equation (19) with the residual vectors

$$r_x = Px_k + c + A^\top y_k + G^\top z_k \quad (21a)$$

$$r_y = Ax_k - b \quad (21b)$$

$$r_z = Gx_k + s_k - h \quad (21c)$$

$$r_s = \lambda_k \circ \lambda_k + (W_k^{-\top} \Delta s_a) \circ (W_k \Delta z_a) - \sigma \mu_k e \quad (21d)$$

This direction moves the next iterate closer to  $(x^*(\sigma \mu_k), s^*(\sigma \mu_k), y^*(\sigma \mu_k), z^*(\sigma \mu_k))$ , a point on the central path where the duality measure is reduced by a factor of  $\sigma$ . Note that the residual vector  $r_s$  has two additional terms when compared to Equation (20d):  $(W_k^{-\top} \Delta s_a) \circ (W_k \Delta z_a)$  and  $-\sigma \mu_k e$ . The former is to correct for linearization error in Newton’s method, and the latter is to maintain proximity to the central path and arises when applying Newton’s method to Equation (9d) with  $\tau = \sigma \mu_k$ .

We then compute the next iterate by performing a line search to ensure  $(s_{k+1}, z_{k+1}) \in \mathcal{K} \times \mathcal{K}$ . Specifically, we update the iterate as

$$(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1}) = (x_k, s_k, y_k, z_k) + \alpha(\Delta x, \Delta s, \Delta y, \Delta z)$$

where

$$\alpha = \sup \left\{ \alpha \in [0, 1] \mid (s_k, z_k) + \frac{\alpha}{0.99} (\Delta s, \Delta z) \in \mathcal{K} \times \mathcal{K} \right\}$$

The factor of 0.99 ensures that  $(s_{k+1}, z_{k+1})$  remains strictly in the interior of  $\mathcal{K} \times \mathcal{K}$ .

## 2.4 Initialization

To initialize the primal-dual interior point method, we follow the approach outlined in [47]. The initialization procedure consists of two steps. First, we find  $(x, s, y, z)$  that satisfy Equations (9a) - (9c). If  $s$  and  $z$  do not satisfy Equation (9e), a correction is applied to ensure they are in  $\mathcal{K}$ .

We begin by solving the optimization problem

$$\begin{aligned} & \underset{x}{\text{minimize}} && \frac{1}{2} x^\top P x + c^\top x + \|Gx - h\|_2^2 \\ & \text{subject to} && Ax = b. \end{aligned}$$

Since this is an equality-constrained quadratic program, the KKT conditions correspond to the linear system

$$\begin{bmatrix} P & A^\top & G^\top \\ A & 0 & 0 \\ G & 0 & -I \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -c \\ b \\ h \end{bmatrix}. \quad (22)$$

We then set  $s = -z$ . This results in the tuple  $(x, s, y, z)$  satisfying Equations (9a) - (9c), but  $s$  and  $z$  may not necessarily satisfy Equation (9e).

We then initialize  $x_0$  and  $y_0$  as  $x_0 = x$ ,  $y_0 = y$ , and initialize  $s_0$  as

$$s_0 = \begin{cases} s, & \text{if } s \in \mathcal{K} \\ s + (1 + \alpha_s)\mathbf{e} & \text{otherwise,} \end{cases}$$

where  $\alpha_s = \inf\{\alpha \mid s + \alpha\mathbf{e} \in \mathcal{K}\}$  and  $\mathbf{e}$  is defined in Equation (7). We initialize  $z_0$  as

$$z_0 = \begin{cases} z, & \text{if } z \in \mathcal{K} \\ z + (1 + \alpha_z)\mathbf{e}, & \text{otherwise} \end{cases}$$

where  $\alpha_z = \inf\{\alpha \mid z + \alpha\mathbf{e} \in \mathcal{K}\}$ . Note that  $\alpha_s$  and  $\alpha_z$  are the minimum scalings of  $\mathbf{e}$  that need to be added to  $s$  and  $z$  respectively to move them to the boundary of  $\mathcal{K}$ , but we scale  $\mathbf{e}$  by  $(1 + \alpha_s)$  and  $(1 + \alpha_z)$  to ensure that  $s_0$  and  $z_0$  are strictly in the interior of  $\mathcal{K}$ .

---

**Algorithm 1** Primal-dual interior point method

---

**Ensure:** Optimal solution  $(x^*, s^*, y^*, z^*)$

```
1: Initialization:
2:   Solve KKT system (22) for  $x, y, z$ 
3:    $x_0 \leftarrow x, y_0 \leftarrow y$ 
4:    $s \leftarrow -z$ 
5:   if  $s \in \mathcal{K}$  then
6:      $s_0 \leftarrow s$ 
7:   else
8:      $\alpha_s \leftarrow \inf\{\alpha \mid s + \alpha e \in \mathcal{K}\}$ 
9:      $s_0 \leftarrow s + (1 + \alpha_s)e$ 
10:  end if
11:  if  $z \in \mathcal{K}$  then
12:     $z_0 \leftarrow z$ 
13:  else
14:     $\alpha_z \leftarrow \inf\{\alpha \mid z + \alpha e \in \mathcal{K}\}$ 
15:     $s_0 \leftarrow z + (1 + \alpha_z)e$ 
16:  end if
17:  repeat
18:    Compute duality measure:
19:     $\mu_k \leftarrow s_k^\top z_k / m$ 
20:    Compute Nesterov-Todd scaling:
21:      Construct  $W_k$ 
22:      Calculate  $\lambda_k = W_k^{-\top} s_k = W_k z_k$ 
23:    Compute affine direction  $(\Delta x_a, \Delta s_a, \Delta y_a, \Delta z_a)$ :
24:      Solve KKT system (19) with residuals (20)
25:    Calculate centering parameter:
26:       $\alpha \leftarrow \sup\{\alpha \in [0, 1] \mid (s_k, z_k) + \alpha(\Delta s_a, \Delta z_a) \in \mathcal{K}\}$ 
27:       $\rho \leftarrow \frac{(s_k + \alpha \Delta s_a)^\top (z_k + \alpha \Delta z_a)}{s_k^\top z_k}$ 
28:       $\sigma \leftarrow \max\{0, \min\{1, \rho\}^3\}$ 
29:    Compute combined direction  $(\Delta x, \Delta s, \Delta y, \Delta z)$ :
30:      Solve (19) with residuals (21)
31:    Compute step-size:
32:       $\alpha \leftarrow \sup\{\alpha \in [0, 1] \mid (s_k, z_k) + \frac{\alpha}{0.99}(\Delta s, \Delta z) \in \mathcal{K}\}$ 
33:    Update iterates:
34:       $x_{k+1} \leftarrow x_k + \alpha \Delta x$ 
35:       $s_{k+1} \leftarrow s_k + \alpha \Delta s$ 
36:       $y_{k+1} \leftarrow y_k + \alpha \Delta y$ 
37:       $z_{k+1} \leftarrow z_k + \alpha \Delta z$ 
38:  until Stopping criteria (28) satisfied.
```

---

### 3 Algorithm implementation

This section describes the implementation of the primal-dual interior point method in QOCO and QOCOGEN. We focus on three key aspects: the matrix factorization used to solve the system in (19), numerical techniques for ensuring a robust factorization and solution of (19), and the stopping criteria. These aspects apply to both QOCO and QOCOGEN, but while both use the same underlying matrix factorization, their implementations are different. These differences will be discussed in Section 4.1.

#### 3.1 Linear system solve

The coefficient matrix in (19), which we will refer to as the Karush-Kuhn-Tucker (KKT) matrix is given as

$$K = \begin{bmatrix} P & A^\top & G^\top \\ A & 0 & 0 \\ G & 0 & -W_k^\top W_k \end{bmatrix}. \quad (23)$$

Algorithm 1 requires solving the linear system  $K\xi = r$  for two different residual vectors  $r$ . Since  $K$  remains constant for both solves, we factorize it once and then apply two backsolves.

The KKT matrix (23) is symmetric but indefinite, meaning it has both positive and negative eigenvalues. A common approach for factoring such matrices is the Bunch-Parlett factorization [55], which factors  $K$  as

$$\Pi K \Pi^\top = LDL^\top, \quad (24)$$

where  $\Pi$  is a permutation matrix,  $L$  is a lower triangular matrix, and  $D$  is a block diagonal matrix with  $1 \times 1$  and  $2 \times 2$  blocks. However,  $\Pi$  must be selected during runtime with knowledge of the data in  $K$ , as the factorization may not exist for all permutations. Additionally,  $K$  may not be invertible if, for example,  $A$  is not full row rank.

**Static regularization** To ensure a numerically stable factorization that exists for all permutations  $\Pi$  (allowing us to select  $\Pi$  based solely on the sparsity pattern of  $K$ ), we apply *static regularization* [56, 12, 34] to the KKT matrix

$$\hat{K} = \left[ \begin{array}{c|cc} P & A^\top & G^\top \\ \hline A & 0 & 0 \\ G & 0 & -W_k^\top W_k \end{array} \right] + \left[ \begin{array}{c|cc} \epsilon_s I & 0 & 0 \\ \hline 0 & -\epsilon_s I & 0 \\ 0 & 0 & -\epsilon_s I \end{array} \right]. \quad (25)$$

In QOCO and QOCOGEN, we use  $\epsilon_s = 10^{-8}$ .

Now  $\hat{K}$  is a *quasidefinite* matrix, which is a special case of a symmetric indefinite matrix where the  $(1, 1)$  block is positive definite and the  $(2, 2)$  block is negative definite [57].

For such matrices, the  $LDL^\top$  factorization

$$\Pi K \Pi^\top = LDL^\top, \quad (26)$$

exists for any permutation  $\Pi$ , and  $D$  will be a diagonal matrix with known signs for its diagonal elements [57].

A judicious choice of  $\Pi$  can minimize the *fill-in* for the factor  $L$ . Fill-in is the introduction of non-zero elements in positions of a matrix where they previously did not exist. Excessive fill-in can lead to several negative consequences, including increased storage requirements (since more non-zero elements must be stored in memory) and increased computational cost (due to more floating-point operations). Finding an optimal  $\Pi$  to minimize fill-in is NP-complete [58], but heuristic methods such as the Approximate Minimum Degree (AMD) ordering can effectively compute near-optimal permutations [45]. In QOCO and QOCOGEN, we use Tim Davis' implementation of the AMD ordering [46] to determine  $\Pi$ .

After computing  $\Pi$ , the  $LDL^\top$  factorization consists of two phases: a symbolic and numeric factorization. The symbolic factorization determines the elimination tree, a data structure which encodes dependencies between elements in the matrix during the factorization process, and the sparsity pattern of the factor  $L$ . It only requires the sparsity pattern of the regularized KKT matrix  $\hat{K}$ , which remains unchanged throughout Algorithm 1. Therefore, the symbolic factorization needs to be computed only once. The numeric factorization calculates the numerical values of  $L$  and  $D$ , which depend on the values of the nonzero elements of  $\hat{K}$ . Since these values change at each iteration of Algorithm 1, the numeric factorization must be computed at every step.

To compute the  $LDL^\top$  factorization, we use a modified version of QDLDL [56] with *dynamic regularization* for QOCO, and a custom  $LDL^\top$  routine for QOCOGEN which we will discuss in Section 4.1.

**Dynamic regularization** Although the  $LDL^\top$  factorization must theoretically exist for  $\hat{K}$ , due to the nature of floating-point arithmetic it is possible for diagonal elements  $D_{ii}$  to be rounded to zero or be very close to zero leading to divide-by-zero errors and the factorization failing. To alleviate this and factor  $\hat{K}$  in a stable manner, we apply *dynamic regularization* as given by Algorithm 2, which bounds the magnitude of  $D_{ii}$  away from zero, by some  $\epsilon_d > 0$ , and corrects for its sign if necessary [12, 34, 32]. In QOCO and QOCOGEN, we use  $\epsilon_d = 10^{-8}$ .

---

**Algorithm 2** Dynamic regularization

---

```

1: Given:  $\epsilon_d > 0$ 
2: if  $D_{ii}$  should be positive then
3:   if  $D_{ii} < 0$  then
4:      $D_{ii} = \epsilon_d$ 
5:   else
6:      $D_{ii} = D_{ii} + \epsilon_d$ 
7:   end if
8: else if  $D_{ii}$  should be negative then
9:   if  $D_{ii} > 0$  then
10:     $D_{ii} = -\epsilon_d$ 
11:   else
12:      $D_{ii} = D_{ii} - \epsilon_d$ 
13:   end if
14: end if
```

---

**Iterative refinement** After static and dynamic regularization we end up solving the linear system  $\hat{K}\xi = r$  rather than system  $K\xi = r$ . To get a solution to the latter system, we apply *iterative refinement* [59, Section 2.5.1]. This process corrects the solution by iteratively reducing the residual error,  $\|r - K\xi\|$ , ensuring higher numerical accuracy without requiring an additional factorization. It involves iteratively solving the system

$$\hat{K}\Delta\xi^k = r - K\xi^k, \quad (27)$$

and updating  $\xi^{k+1} = \xi^k + \Delta\xi^k$ , where  $\xi^0$  solves the first linear system (i.e.  $\hat{K}\xi^0 = r$ ). Note that since we have already computed the factorization of  $\hat{K}$ , one iteration of Equation (27) only requires backsolves and no matrix factorization.

### 3.2 Stopping criterion

Algorithm 1 terminates when three conditions are met: primal feasibility (28a), stationarity (28b), and complementary slackness (28c). We use both absolute and relative tolerances to ensure robustness across different problem scales. Our stopping criteria is met if the following conditions hold:

$$\left\| \begin{bmatrix} Ax_k - b \\ Gx_k + s_k - h \end{bmatrix} \right\|_\infty \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max \{ \|Ax_k\|_\infty, \|b\|_\infty, \|Gx_k\|_\infty, \|h\|_\infty, \|s_k\|_\infty \} \quad (28a)$$

$$\|Px_k + A^\top y_k + G^\top z_k + c\|_\infty \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max \{ \|Px_k\|_\infty, \|A^\top y_k\|_\infty, \|G^\top z_k\|_\infty, \|c\|_\infty \} \quad (28b)$$

$$|s_k^\top z_k| \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max \{ 1, |p_k|, |d_k| \}, \quad (28c)$$

where the primal and dual objectives are

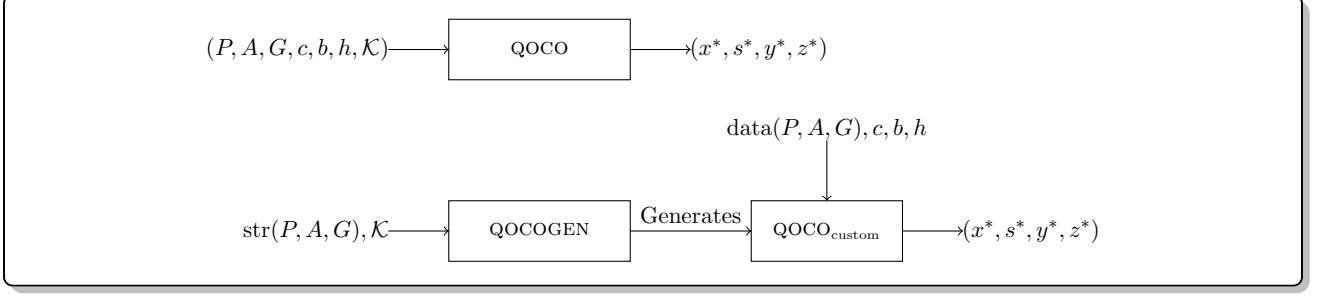


Figure 2: Usage of QOCO, QOCOGEN, and QOCO<sub>custom</sub>.

$$p_k := \frac{1}{2}x_k^\top Px_k + c^\top x_k, \quad d_k := -\frac{1}{2}x_k^\top Px_k - b^\top y_k - h^\top z_k.$$

## 4 Sparsity exploiting custom solver

This section discusses QOCOGEN, which generates an implementation of Algorithm 1 called QOCO<sub>custom</sub>. Unlike generic solvers which use sparse linear algebra, QOCO<sub>custom</sub> uses a custom  $LDL^\top$  factorization and other tailored linear algebra routines, which exploit the known sparsity structure of problem data, to solve Problem 1 significantly faster.

While QOCO can solve instances of Problem 1 with any sparsity pattern for  $P$ ,  $A$ , and  $G$  and cone  $\mathcal{K}$ , the use of custom linear algebra in QOCO<sub>custom</sub> restricts it to instances with the same sparsity structure for  $P$ ,  $A$ , and  $G$  and optimizes over the same cone  $\mathcal{K}$ . This makes QOCO<sub>custom</sub> useful for applications that repeatedly solve optimization problems with identical sparsity structures, such as sequential convex programming [8].

Figure 2 illustrates the usage of QOCO, QOCOGEN, and QOCO<sub>custom</sub>. In the figure, the matrices  $(P, A, G)$  contain two pieces of information: their sparsity patterns  $\text{str}(P, A, G)$  (the location of nonzero elements) and the values of the nonzero elements  $\text{data}(P, A, G)$ . We see that QOCO takes in  $(P, A, G)$ , which contains both the sparsity patterns and nonzero elements for the matrices, and the rest of the problem data and returns the optimal solution  $(x^*, s^*, y^*, z^*)$ . In contrast, QOCOGEN only takes in the sparsity patterns of  $(P, A, G)$  and cone  $\mathcal{K}$ , and then generates QOCO<sub>custom</sub>. This instance of QOCO<sub>custom</sub> can then solve optimization problems where the sparsity patterns of  $(P, A, G)$  do not change, but the nonzero values for  $(P, A, G)$  and the vector data  $(c, b, h)$  can change.

Although QOCO avoids dynamic memory allocation during the solution process, it still relies on `setup` and `cleanup` functions for dynamic memory allocation and deallocation before and after solving the problem. In contrast, QOCO<sub>custom</sub> exclusively uses static memory allocation, making it well-suited for resource-constrained embedded systems, where dynamic memory allocation can lead to non-deterministic behavior and memory fragmentation.

### 4.1 Custom linear algebra

A key difference between QOCO<sub>custom</sub> and QOCO is that QOCO<sub>custom</sub> uses custom linear algebra routines rather than sparse linear algebra. The main motivation for custom routines is to speed up the  $LDL^\top$  factorization of the regularized KKT matrix, which is the most computationally expensive step of Algorithm 1. Typically, the KKT matrix is factored using a sparse linear algebra routine that stores the matrix in either *compressed sparse column* (CSC) or *compressed sparse row* (CSR) format. However, these sparse routines involve more than just the floating-point operations required for factorization. They also incur additional overhead to locate nonzero elements and their positions in the matrix. For example, in a CSC matrix, accessing data first requires indexing into the column pointer and row index arrays. This results in more CPU instructions, increased memory accesses, and a higher likelihood of cache misses. We provide empirical evidence supporting these claims in Appendix B.

```

1 # Adata stores the nonzeros of A in column-major format, i.e.
2 # Adata[0] = a00
3 # Adata[1] = a20
4 # Adata[2] = a02
5 def custom_matvec(y, Adata, x):
6     y[0] = Adata[0]*x[0] + Adata[2]*x[2]
7     y[1] = 0
8     y[2] = Adata[1]*x[0]

```

Listing 1: Custom matrix-vector multiplication.

When the sparsity structure of the KKT matrix is known *a priori* it is possible to write a custom  $LDL^\top$  factorization routine which only contains code to perform the necessary floating-point operations and data accesses, eliminating the additional overhead that typical sparse linear algebra routines have. This makes the custom  $LDL^\top$  factorization significantly faster than a sparse  $LDL^\top$  factorization. We also note that since the AMD ordering we employ to compute the permutation matrix  $\Pi$  only depends on the sparsity pattern of the KKT matrix, we can determine  $\Pi$  at the time of code generation, so the generated code does not include the code to compute the AMD ordering.

A concrete example of a custom linear algebra routine for matrix-vector multiplication is given by Listing 1, where the sparsity pattern of  $A$  is fixed and defined in Equation (29). Note that the custom routine only includes the necessary floating-point operations. Although Listing 1 depicts Python code, the custom linear algebra routines in  $\text{QOCO}_{\text{custom}}$  are in C.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \underbrace{\begin{bmatrix} a_{00} & 0 & a_{02} \\ 0 & 0 & 0 \\ a_{20} & 0 & 0 \end{bmatrix}}_A \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \quad (29)$$

A drawback of these custom routines is that the amount of code generated increases with the problem size. As a result, the time to generate and compile  $\text{QOCO}_{\text{custom}}$  can become burdensome as the problem sizes get very large.

## 4.2 Code generation

$\text{QOCOGEN}$  is written in Python and example code for generating  $\text{QOCO}_{\text{custom}}$  for a problem family defined by Equation (30) is given by Listing 2. The generated solver  $\text{QOCO}_{\text{custom}}$  is customized to the sparsity pattern of the matrices  $P, A, G$  which are passed to the function `qocogen.generate_solver()` as sparse `scipy` matrices.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x_1^2 + x_2^2 + x_3^2 + x_4^2 \\ \text{subject to} \quad & x_1 + x_2 = 1 \\ & x_2 + x_3 = 1 \\ & x_1 \geq 0 \\ & \sqrt{x_3^2 + x_4^2} \leq x_2 \end{aligned} \quad (30)$$

The file tree for  $\text{QOCO}_{\text{custom}}$  is given by Figure 3. The file `qoco_custom.c` contains the `qoco_custom_solve()` function which implements Algorithm 1, `1d1.c` contains the custom  $LDL^\top$  factorization of the regularized KKT matrix, `CMakeLists.txt` is the CMake configuration file which automates the build process [60], `runttest.c` gives an example of calling  $\text{QOCO}_{\text{custom}}$  with default data and times the resulting solve, and the remaining files contain various helper functions needed to implement Algorithm 1.

Listing 3 provides an example of calling  $\text{QOCO}_{\text{custom}}$  from C, updating the problem data, and solving it again.

```

1 import qocogen
2 import numpy as np
3 from scipy import sparse
4
5 # Define problem data
6 P = sparse.diags([2, 2, 2, 0], 0).tocsc()
7
8 c = np.array([0, 0, 0, 1])
9 G = -sparse.identity(4).tocsc()
10 h = np.zeros(4)
11 A = sparse.csc_matrix([[1, 1, 0, 0], [0, 1, 1, 0]])
12 b = np.array([1, 1])
13
14 l = 1
15 n = 4
16 m = 4
17 p = 2
18 nsoc = 1
19 q = np.array([3])
20
21 # Generate custom solver.
22 qocogen.generate_solver(n, m, p, P, c, A, b, G, h, l, nsoc, q)

```

Listing 2: Generating a custom solver with QOCOGEN.

```

1 #include "qoco_custom.h"
2
3 int main() {
4     // Instantiate workspace.
5     Workspace work;
6
7     // Set default settings, but settings can be modified with
8     // work.settings.setting_name = ...
9     set_default_settings(&work);
10
11    // Loads the default data for the problem
12    // (i.e. the data passed to the qocogen.generate() call).
13    load_data(&work);
14
15    // Solve original problem.
16    qoco_custom_solve(&work);
17    printf("\nobj: %f", work.sol.obj);
18
19    // Can modify non-zero elements of P,A,G and c, b, h.
20    update_P(&work, Pnew);
21    update_A(&work, Anew);
22    update_G(&work, Gnew);
23    update_c(&work, cnew);
24    update_b(&work, bnew);
25    update_h(&work, hnew);
26
27    // Solve updated problem.
28    qoco_custom_solve(&work);
29    printf("\nobj: %f", work.sol.obj);
30}

```

Listing 3: Calling QOCO<sub>custom</sub> from C/C++.

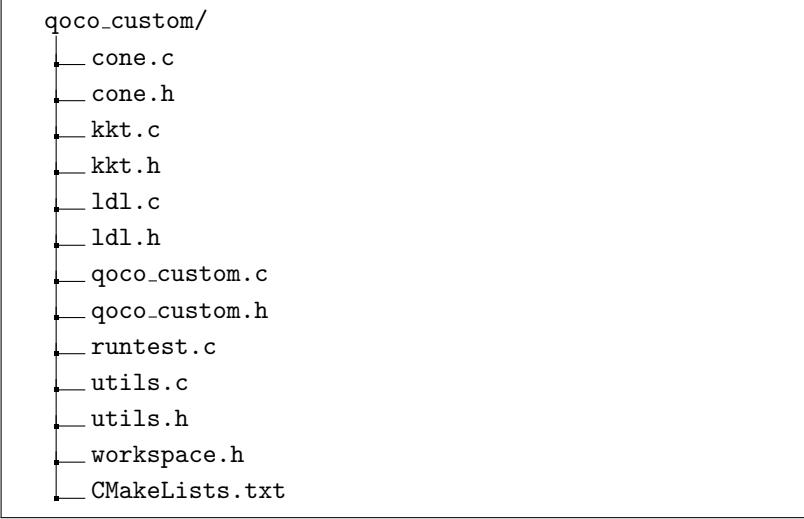


Figure 3: `QOCOcustom` file structure.

## 5 Numerical results

We benchmark QOCO and `QOCOcustom` against several conic interior-point solvers <sup>2</sup>: the open-source solvers ECOS [32] and CLARABEL [12], the commercial solvers GUROBI [13] and MOSEK [33], and against the academically licensed custom solver generator CVXGEN which generates a primal-dual interior point method for quadratic programs. We do not test against the custom solver generator BSOCOP since it is not academically licensed. For all solvers, we use their default settings but set the tolerances  $\epsilon_{abs} = \epsilon_{rel} = 10^{-7}$ .

In our experiments, we define problem size as the total number of nonzero elements in  $A$ ,  $G$ , and in the upper half of  $P$ . We evaluate our solvers on a set of benchmark problems we developed, a set of model-predictive control problems from [61], the Maros–Mészáros problems [62], and least-squares problems derived from the SuiteSparse repository [63]. To evaluate the solvers’ performance, we use performance profiles [64] and the shifted geometric mean, both of which are frequently used to assess solver performance [12, 32, 56, 14, 65].

All numerical results were generated on a desktop computer with an AMD Ryzen 9 7950X3D processor running at 4.2 GHz and with 128 GB of RAM. Our benchmarks are written in Python and we use CVXPY [42, 43] to interface with the solvers.

We provide detailed numeric results in Appendix D, which includes iteration counts and solve times for each solver on all problems we ran. Because Gurobi does not report iteration counts via the CVXPY interface for SOCPs, these values are omitted from our results.

**Shifted geometric mean** We use the normalized shifted geometric mean to assign a scalar value to the performance of a solver on a test set of problems. The shifted geometric mean of  $N$  runtimes for solver  $s$  is computed as

$$g_s = \left[ \prod_{p=1}^N (t_{s,p} + k) \right]^{1/N} - k,$$

where  $t_{s,p}$  is the runtime of solver  $s$  on problem  $p$  and  $k = 1$  is the shift. If the solver does not find an optimal solution, then the runtime is set to 10 seconds for benchmark problems, 0.5 seconds for the MPC problems, 1200 seconds for the Maros–Mészáros problems, and 600 seconds for the SuiteSparse

---

<sup>2</sup>Our benchmarks are publicly available at <https://github.com/qoco-org/qoco-benchmarks>

least-squares problems. These times were chosen to be roughly an order of magnitude higher than the slowest successful solve on the respective problem set.

We then define the normalized shifted geometric mean for solver  $s$  as

$$r_s = g_s / \min_s g_s,$$

so the solver with the lowest shifted geometric mean will have a normalized shifted geometric mean of 1.00.

**Performance profiles** We also plot the relative and absolute performance profiles to compare solvers. We first define the relative performance ratio of solver  $s$  on problem  $p$  as

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

The relative performance profile then plots  $f_s^r(\tau)$  where

$$f_s^r(\tau) = \frac{1}{N} \sum_{p=1}^N \mathcal{I}_{\leq \tau}(u_{s,p}),$$

and  $\mathcal{I}_{\leq \tau}(z) = 1$  if  $z \leq \tau$  and  $\mathcal{I}_{\leq \tau}(z) = 0$  otherwise. If the relative performance profile passes through the point  $(x, y)$  for solver  $s$ , this means that solver  $s$  solves a fraction  $y$  of the problems within a factor of  $x$  of the fastest solver.

The absolute performance profile plots  $f_s^a(\tau)$  where

$$f_s^a(\tau) = \frac{1}{N} \sum_{p=1}^N \mathcal{I}_{\leq \tau}(t_{s,p}).$$

If the absolute performance profile passes through the point  $(x, y)$  for solver  $s$ , this means that solver  $s$  solves a fraction  $y$  of the problems within  $x$  time. For both relative and absolute profiles, the profile of the highest-performing solver will lie above the profiles of the other solvers.

## 5.1 Benchmark problems

We solve optimization problems from five problem classes: robust Kalman filtering, group lasso regression [66], the losslessly convexified powered-descent guidance problem [37], Markowitz portfolio optimization [3], and the oscillating masses control problem [67]. The first three of these are SOCPs and the last two are QPs. For each of the five classes, we consider 10 different problem sizes, generating 20 unique problem instances per size, for a total of 1000 distinct optimization problems. Details on the problems we solve can be found in Appendix C. For these problems, we plot runtime as a function of problem size in addition to reporting performance profiles and shifted geometric means. We test QOCO on all problems, but only test QOCO<sub>custom</sub> on the smaller problems, as code generation and compile times become prohibitively long for the larger instances.

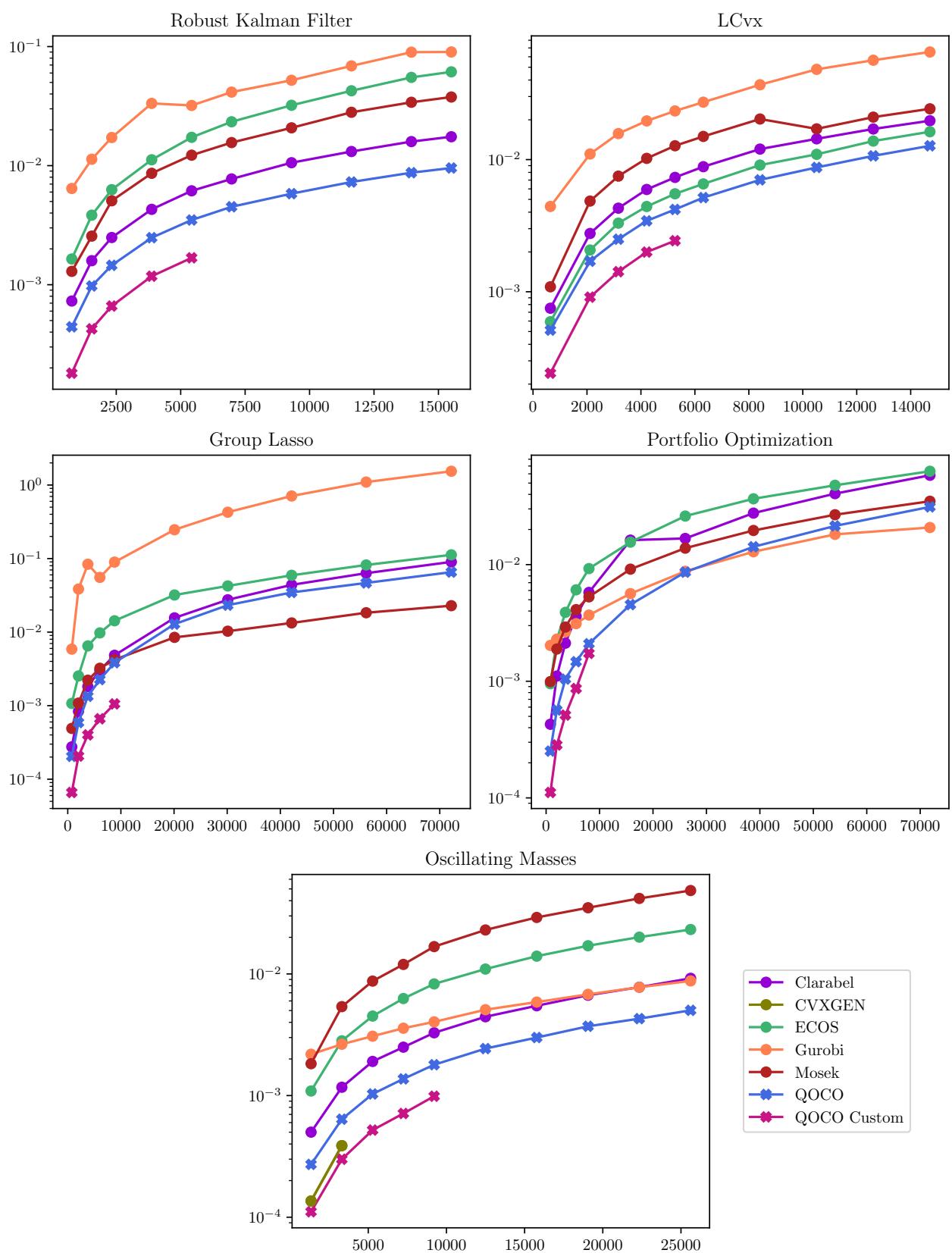
For each solver, we solve each optimization problem 100 times and record the minimum runtime, which includes both setup and solve time. Since the computer runs various background processes that can artificially inflate execution time, we conduct 100 runs and report the minimum execution time, thereby providing a more accurate measure of the solver's performance.

Although the oscillating masses and portfolio optimization problems are QPs and can be solved with CVXGEN, we limit testing to the two smaller oscillating masses problems. For larger instances, CVXGEN was unable to generate code due to the size of the problems. For the portfolio optimization problems, we do not test CVXGEN due to the sparsity pattern of the factor matrix  $F$  in Problem 33. To specify

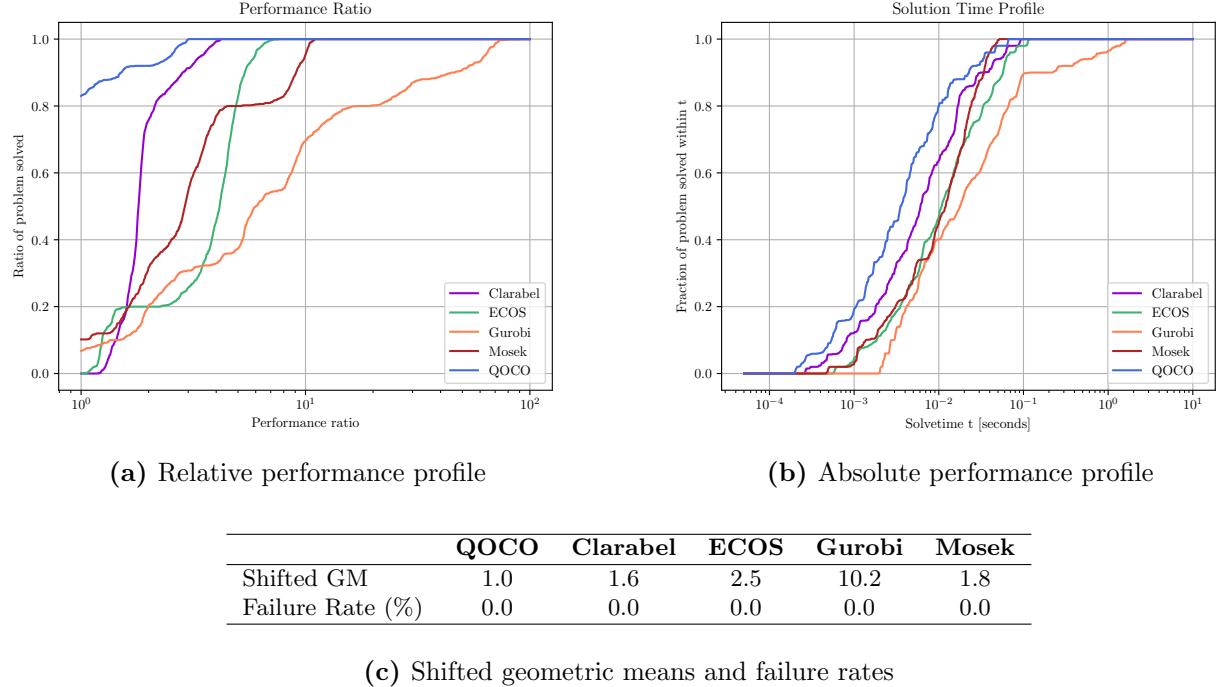
the sparsity pattern of  $F$  in CVXGEN, we must manually provide the locations of nonzero elements into the web interface, which would have been prohibitively expensive, since  $F$  has hundreds of nonzero elements. In contrast, to generate code with QOCOGEN, we pass in problem data as sparse `scipy` matrices rather than manually specifying sparsity patterns. Although we can specify  $F$  as a dense matrix and only populate the nonzero elements for CVXGEN, this would severely hinder its performance and result in an unfair comparison.

When computing normalized shifted geometric mean, relative performance profiles, and absolute performance profiles, all solvers must be tested on the same set of problems. Since we test QOCO<sub>custom</sub> on only half of the problems we provide two versions of the aforementioned metrics. The first version, given by Figure 5, includes CLARABEL, ECOS, GUROBI, MOSEK, and QOCO evaluated on all problems. The second version, given by Figure 6, also includes QOCO<sub>custom</sub>, but the solvers are evaluated on the half of the benchmark problems which QOCO<sub>custom</sub> is tested on.

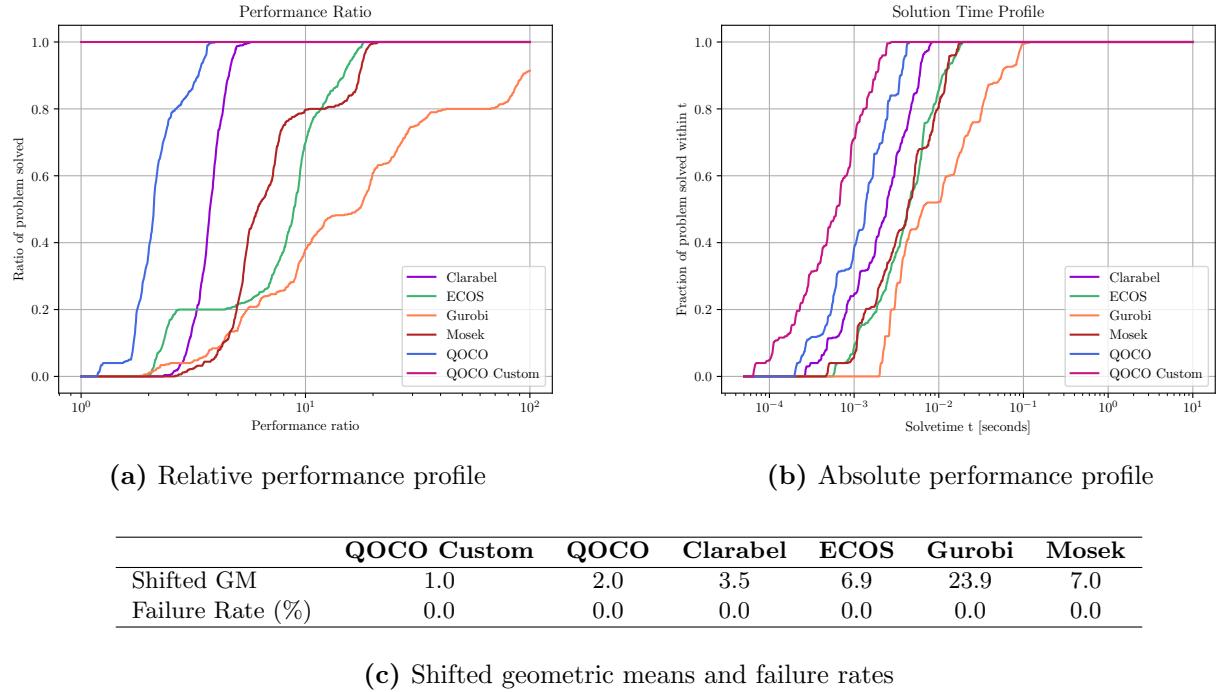
Figures 4, 5, and 6 show that QOCO<sub>custom</sub> is the fastest solver on our benchmark problems by a significant margin and is as performant as CVXGEN on the 40 smallest oscillating mass problems. This latter result is unsurprising, as QOCO<sub>custom</sub> uses an identical algorithm to CVXGEN when solving QPs and both utilize custom linear algebra routines. Additionally, QOCO is the fastest open-source generic solver on all of the problems, although it is slower than Mosek on the group lasso problems and slower than Gurobi on some of the largest portfolio optimization problems. However, we notice that if we disable presolve for Mosek and Gurobi, QOCO is faster on these problems as well.



**Figure 4: Solvetime in seconds vs problem size for benchmark problems**



**Figure 5: Performance profiles for all benchmark problems**



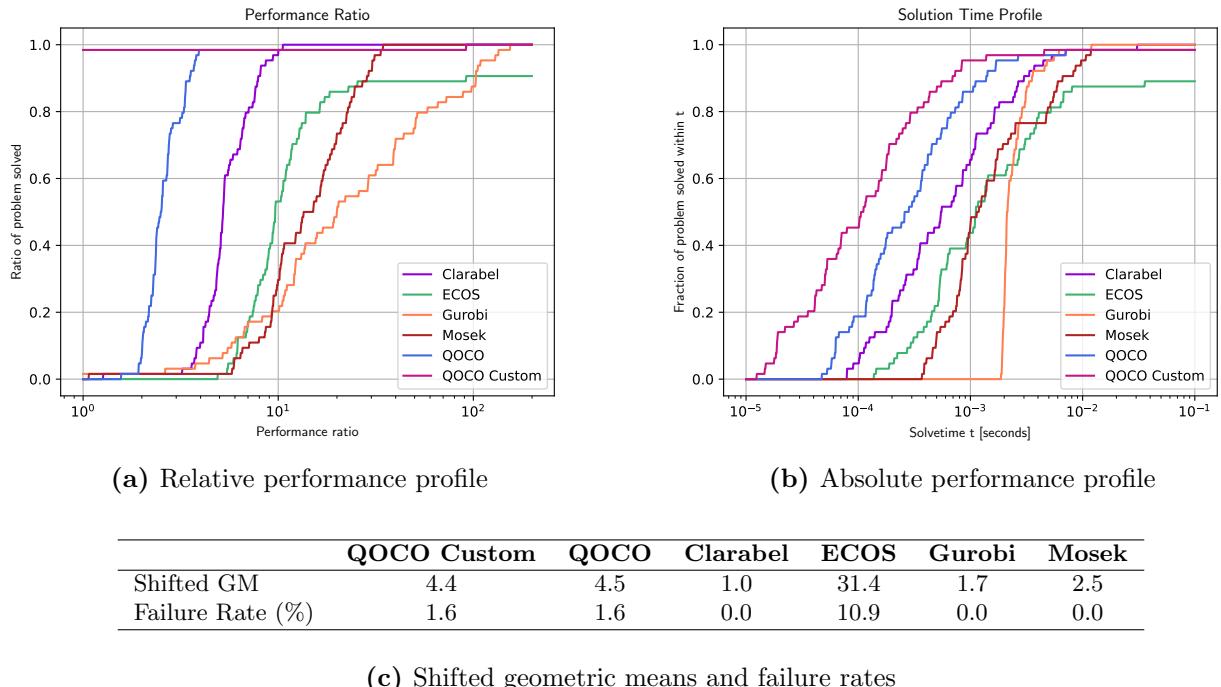
**Figure 6: Performance profiles for benchmark problems with custom solver**

## 5.2 Model-predictive control problems

We consider a set of 64 model-predictive control problems taken from [61] and also used by [12]. Note that we exclude the eight “nonlinear chain” problems as they were too large to generate custom solvers for. Similarly to the benchmark problems, we solve each MPC problem 100 times and record the minimum runtime. These problems take the following form

$$\begin{aligned}
& \underset{x,y,u}{\text{minimize}} && \sum_{k=0}^{\tau} \begin{bmatrix} y_k - y_k^r \\ u_k - u_k^r \end{bmatrix}^\top \begin{bmatrix} Q & S \\ S^\top & R \end{bmatrix} \begin{bmatrix} y_k - y_k^r \\ u_k - u_k^r \end{bmatrix} + (x_\tau - x_\tau^r)^\top P(x_\tau - x_\tau^r) \\
& \text{subject to} && x_0 = x_{\text{init}} \\
& && x_{k+1} = Ax_k + Bu_k + f_k \\
& && y_k = Cx_k + Du_k + e_k \\
& && d_{\min} \leq Mx_k + Nu_k \leq d_{\max} \\
& && u_{\min} \leq u_k \leq u_{\max} \\
& && y_{\min} \leq y_k \leq y_{\max} \\
& && d_{\min}^\tau \leq Tx_\tau \leq d_{\max}^\tau.
\end{aligned}$$

Figure 7 shows that CLARABEL, GUROBI, and MOSEK solve all problems successfully, QOCO and QOCO<sub>custom</sub> each fail on only one, and ECOS fails on seven. Among the 63 problems solved by both QOCO and QOCO<sub>custom</sub>, QOCO<sub>custom</sub> is the fastest solver by a wide margin, followed by QOCO.



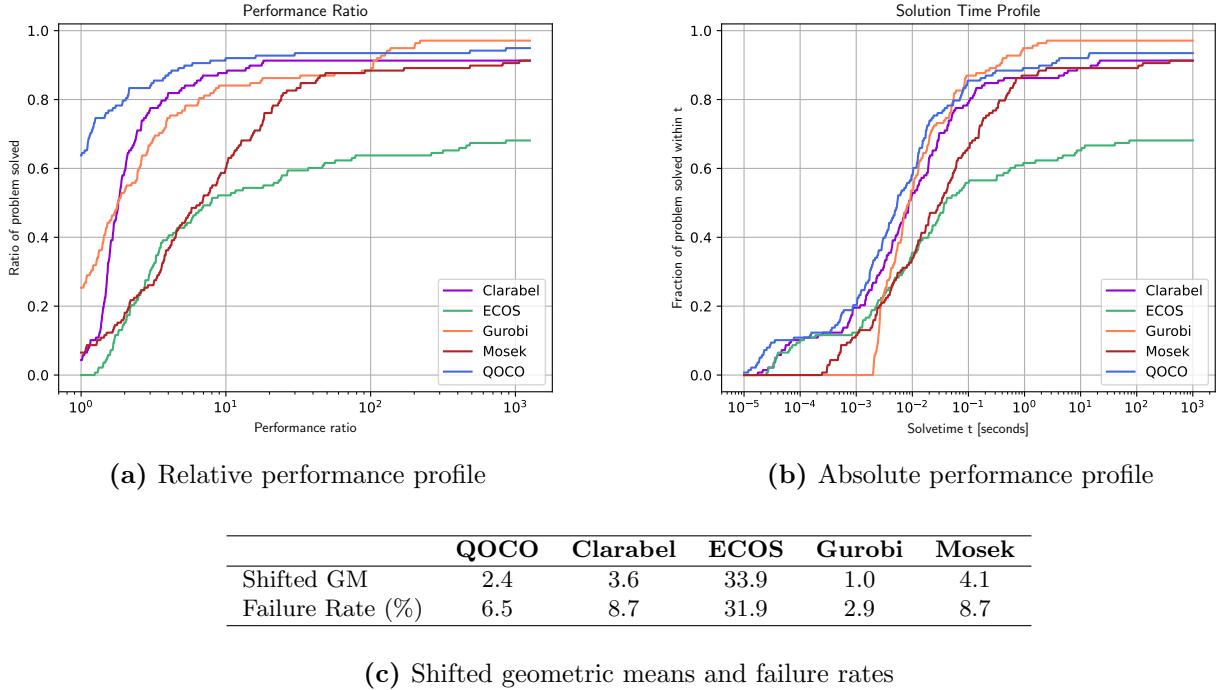
**Figure 7: Performance profiles for model-predictive control problems**

### 5.3 Maros–Mészáros problems

The Maros–Mészáros problems are a set of 138 challenging QPs that include a wide range of problem sizes (from 3 to around 300,000 variables) and contain very difficult problems that have ill-conditioning and poor scaling [62]. These properties make the Maros–Mészáros problems a good test set to assess the robustness of solvers. We test QOCO but not QOCO<sub>custom</sub> on these problems since many problems are far too large to generate code for. Additionally, we run each problem once rather than the 100 runs we do for the benchmark and MPC problems since many of the Maros–Mészáros problems are extremely large, and running them 100 times per solver is extremely cumbersome.

Figure 8 shows that the proprietary solver Gurobi successfully solved the largest fraction of the problems followed by QOCO, CLARABEL, MOSEK, and ECOS. We can also see that for the majority of the problems, QOCO was the fastest solver, whereas MOSEK and ECOS tended to be the slowest. The latter result is unsurprising as MOSEK and ECOS can only handle linear objectives and the reformulation of the

quadratic objective into a second-order cone likely introduces significant fill-in, slowing these solvers down. We also observe that for many of the largest problems, Gurobi is the fastest solver, due to multithreading in its matrix factorization.



**Figure 8: Performance profiles for Maros–Mészáros problems**

## 5.4 SuiteSparse least-squares problems

Finally, we consider a set of 23 least-squares problems  $Ax \approx b$ , where the matrix  $A \in \mathbb{R}^{m \times n}$  is drawn from the SuiteSparse Matrix Collection [63]. Following [56] and [12], we get an approximate solution to each least-squares problem by solving a Huber regression and lasso regression problem, which can both be formulated as constrained quadratic programs [51]. These optimization problems are quite large with some having over a million optimization variables, so we do not test  $\text{QOCO}_{\text{custom}}$  due to the prohibitively long code-generation time. Additionally, we run each problem once rather than 100 times.

The Huber regression problem is given by

$$\underset{x}{\text{minimize}} \quad \sum_{i=1}^m \phi(a_i^\top x - b_i),$$

where  $a_i^\top$  is the  $i^{\text{th}}$  row of  $A$  and  $\phi : \mathbb{R} \rightarrow \mathbb{R}$  is defined as

$$\phi(z) = \begin{cases} z^2 & \text{if } |z| \leq \delta \\ \delta(2|z| - \delta) & \text{if } |z| > \delta \end{cases},$$

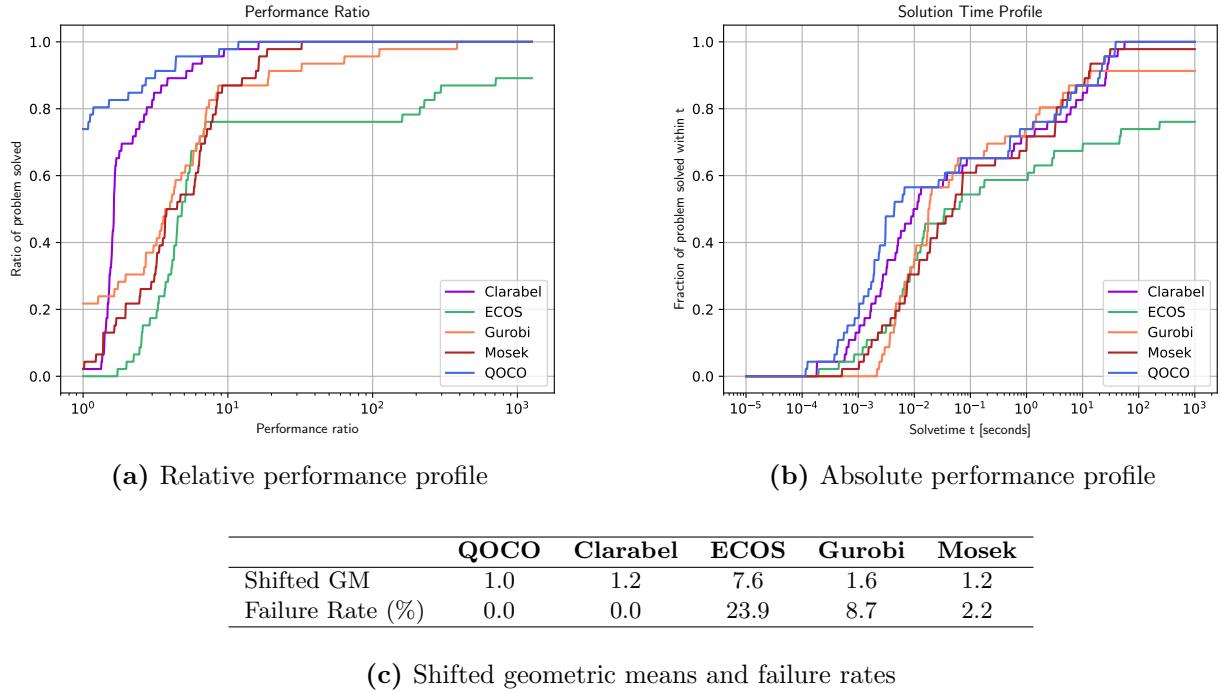
where we use  $\delta = 1$ .

The lasso regression problem is given by

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

where we set  $\lambda = \|A^\top b\|_\infty$ .

Figure 9 shows that QOCO and CLARABEL successfully solve all problems, with QOCO being the fastest on most instances. As with the Maros–Mészáros problems, we observe that Gurobi is the fastest on some of the largest problems, likely due to its multithreaded matrix factorization.



**Figure 9: Performance profiles for SuiteSparse least-squares problems**

## 6 Conclusion

We have presented QOCO, a C-based solver, and QOCOGEN, a custom solver generator for quadratic objective second-order cone programs (SOCPs). Both implement primal-dual interior-point methods, with QOCOGEN generating custom solvers written in C, called  $\text{QOCO}_{\text{custom}}$ , that exploit the sparsity pattern of problems to gain a computational advantage. We demonstrate that QOCO is robust and faster than most of its competitors and  $\text{QOCO}_{\text{custom}}$  is significantly faster than QOCO, making it a useful tool for real-time applications, such as model-predictive control, trajectory optimization, and other domains where computational efficiency is critical.

Since both QOCO and QOCOGEN are open-source, they are accessible to users in both academia and industry. QOCO and QOCOGEN are integrated with CVXPY and CVXPYGEN respectively, allowing users to formulate optimization problems in a natural way following from math rather than manually converting the problem to the solver-required standard form. This integration enhances usability of our solvers.

Documentation and installation instructions for QOCO and QOCOGEN are available at <https://qoco-org.github.io/qoco/index.html>.

**Acknowledgments** This research was supported by ONR grant N00014-20-1-2288, AFOSR grant FA9550-20-1-0053, and Blue Origin, LLC. Government sponsorship is acknowledged. The authors would like to thank Danylo Malyuta for his review of this paper.

## References

- [1] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.
- [2] L. R. Ford and D. R. Fulkerson, “A suggested computation for maximal multi-commodity network flows,” *Management Science*, vol. 5, no. 1, pp. 97–101, 1958.
- [3] H. Markowitz, “Portfolio selection,” *J. Finance*, vol. 7, p. 77, Mar. 1952.
- [4] M. S. Lobo, M. Fazel, and S. Boyd, “Portfolio optimization with linear and fixed transaction costs,” *Annals of Operations Research*, vol. 152, pp. 341–365, 2007.
- [5] A. Ben-Tal and A. Nemirovski, “Robust convex optimization,” *Mathematics of Operations Research*, vol. 23, no. 4, pp. 769–805, 1998.
- [6] A. Ben-Tal and A. Nemirovski, “Robust solutions of uncertain linear programs,” *Operations Research Letters*, vol. 25, no. 1, pp. 1–13, 1999.
- [7] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, “Applications of second-order cone programming,” *Linear algebra and its applications*, vol. 284, no. 1-3, pp. 193–228, 1998.
- [8] D. Malyuta, T. P. Reynolds, M. Szmuk, T. Lew, R. Bonalli, M. Pavone, and B. Açıkmeşe, “Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently,” *IEEE Control Systems Magazine*, vol. 42, no. 5, pp. 40–113, 2022.
- [9] D. Drusvyatskiy and A. S. Lewis, “Error bounds, quadratic growth, and linear convergence of proximal methods,” *Mathematics of Operations Research*, vol. 43, no. 3, pp. 919–948, 2018.
- [10] T. Lipp and S. Boyd, “Variations and extension of the convex–concave procedure,” *Optimization and Engineering*, vol. 17, pp. 263–287, 2016.
- [11] A. L. Yuille and A. Rangarajan, “The concave-convex procedure (cccp),” *Advances in neural information processing systems*, vol. 14, 2001.
- [12] P. J. Goulart and Y. Chen, “Clarabel: An interior-point solver for conic programs with quadratic objectives,” 2024.
- [13] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2023.
- [14] M. Garstka, M. Cannon, and P. Goulart, “Cosmo: A conic operator splitting method for convex conic problems,” *Journal of Optimization Theory and Applications*, vol. 190, p. 779–810, Aug. 2021.
- [15] B. O’Donoghue, “Operator splitting for a homogeneous embedding of the linear complementarity problem,” *SIAM Journal on Optimization*, vol. 31, pp. 1999–2023, 08 2021.
- [16] A. Themelis and P. Patrinos, “Supermann: a superlinearly convergent algorithm for finding fixed points of nonexpansive operators,” *IEEE Transactions on Automatic Control*, vol. 64, no. 12, pp. 4875–4890, 2019.
- [17] E. K. Ryu and W. Yin, *Large-Scale Convex Optimization: Algorithms and Analyses via Monotone Operators*. Cambridge University Press, Nov. 2022.
- [18] J. Rawlings, E. Meadows, and K. Muske, “Nonlinear model predictive control: A tutorial and survey,” *IFAC Proceedings Volumes*, vol. 27, no. 2, pp. 185–197, 1994. IFAC Symposium on Advanced Control of Chemical Processes, Kyoto, Japan, 25-27 May 1994.
- [19] J. Nocedal and S. Wright, *Numerical optimization*, pp. 1–664. Springer Series in Operations Research and Financial Engineering, Springer Nature, 2006.
- [20] P. T. Boggs and J. W. Tolle, “Sequential quadratic programming,” *Acta numerica*, vol. 4, pp. 1–51, 1995.

- [21] M. Szmuk, T. P. Reynolds, and B. Açıkmese, “Successive convexification for real-time six-degree-of-freedom powered descent guidance with state-triggered constraints,” *Journal of Guidance, Control, and Dynamics*, vol. 43, p. 1399–1413, Aug. 2020.
- [22] A. G. Kamath, P. Elango, Y. Yu, S. Mceowen, G. M. Chari, J. M. Carson III, and B. Açıkmese, “Real-time sequential conic optimization for multi-phase rocket landing guidance,” *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 3118–3125, 2023. 22nd IFAC World Congress.
- [23] G. M. Chari, A. G. Kamath, P. Elango, and B. Acikmese, “Fast monte carlo analysis for 6-dof powered-descent guidance via gpu-accelerated sequential convex programming,” in *AIAA SCITECH 2024 Forum*, American Institute of Aeronautics and Astronautics, Jan. 2024.
- [24] A. Berning, E. Burnett, and S. Bieniawski, “Chance-constrained, drift-safe guidance for spacecraft rendezvous,” in *AAS Rocky Mountain Guidance, Navigation and Control Conference*, AAS, 2023.
- [25] G. M. Chari and B. Açıkmese, “Spacecraft rendezvous guidance via factorization-free sequential convex programming using a first-order method,” *arXiv preprint arXiv:2402.04561*, 2024.
- [26] S. Mceowen, A. G. Kamath, P. Elango, T. Kim, S. C. Buckner, and B. Acikmese, “High-accuracy 3-dof hypersonic reentry guidance via sequential convex programming,” in *AIAA scitech 2023 forum*, p. 0300, 2023.
- [27] S. Mceowen, D. J. Calderone, A. Tiwary, J. S. Zhou, T. Kim, P. Elango, and B. Acikmese, “Auto-tuned primal-dual successive convexification for hypersonic reentry guidance,” in *AIAA SCITECH 2025 Forum*, p. 1317, 2025.
- [28] J. M. Carson, M. M. Munk, R. R. Sostaric, J. N. Estes, F. Amzajerdian, J. B. Blair, D. K. Rutishauser, C. I. Restrepo, A. M. Dwyer-Cianciolo, G. Chen, and T. Tse, *The SPLICE Project: Continuing NASA Development of GN&C Technologies for Safe and Precise Landing*.
- [29] T. P. Reynolds, M. Szmuk, D. Malyuta, M. Mesbahi, B. Açıkmese, and J. M. Carson, “Dual quaternion-based powered descent guidance with state-triggered constraints,” *Journal of Guidance, Control, and Dynamics*, vol. 43, p. 1584–1599, Sept. 2020.
- [30] A. G. Kamath, P. Elango, T. Kim, S. Mceowen, Y. Yu, J. M. Carson, M. Mesbahi, and B. Acikmese, *Customized Real-Time First-Order Methods for Onboard Dual Quaternion-based 6-DoF Powered-Descent Guidance*.
- [31] G. Mendeck and W. May, “Space technology mission directorate-game changing development program-fy23 splice annual review presentation,” in *NASA Game Changing Development Annual Program Review*, 2023.
- [32] A. Domahidi, E. Chu, and S. Boyd, “ECOS: An SOCP solver for embedded systems,” in *European Control Conference (ECC)*, pp. 3071–3076, 2013.
- [33] M. ApS, *MOSEK Optimization Suite 10.2.6*, 2024.
- [34] J. Mattingley and S. Boyd, “Cvxgen: A code generator for embedded convex optimization,” *Optimization and Engineering*, vol. 13, pp. 1–27, 2012.
- [35] D. Dueri, J. Zhang, and B. Açıkmese, “Automated custom code generation for embedded, real-time second order cone programming,” *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 1605–1612, 2014. 19th IFAC World Congress.
- [36] D. Dueri, B. Açıkmese, D. P. Scharf, and M. W. Harris, “Customized real-time interior-point methods for onboard powered-descent guidance,” *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 197–212, 2017.
- [37] B. Acikmese and S. R. Ploen, “Convex programming approach to powered descent guidance for mars landing,” *Journal of Guidance Control and Dynamics*, vol. 30, pp. 1353–1366, Sept. 2007.

- [38] B. Acikmese, J. M. Carson, and L. Blackmore, “Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem,” *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2104–2113, 2013.
- [39] D. P. Scharf, B. Açıkmese, D. Dueri, J. Benito, and J. Casoliva, “Implementation and experimental demonstration of onboard powered-descent guidance,” *Journal of Guidance, Control, and Dynamics*, vol. 40, p. 213–229, Feb. 2017.
- [40] L. Blackmore, “Autonomous precision landing of space rockets,” in *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2016 Symposium*, vol. 46, pp. 15–20, 2016.
- [41] M. Schaller, G. Banjac, S. Diamond, A. Agrawal, B. Stellato, and S. Boyd, “Embedded code generation with cvxpy,” *IEEE Control Systems Letters*, vol. 6, pp. 2653–2658, 2022.
- [42] S. Diamond and S. Boyd, “CVXPY: A Python-embedded modeling language for convex optimization,” *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.
- [43] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, “A rewriting system for convex optimization problems,” *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.
- [44] S. Mehrotra, “On the implementation of a primal-dual interior point method,” *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.
- [45] P. R. Amestoy, T. A. Davis, and I. S. Duff, “An approximate minimum degree ordering algorithm,” *SIAM Journal on Matrix Analysis and Applications*, vol. 17, p. 886–905, Oct. 1996.
- [46] T. A. Davis, “Algorithm 849: A concise sparse cholesky factorization package,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 4, pp. 587–591, 2005.
- [47] L. Vandenberghe, “The cvxopt linear and quadratic cone program solvers,” *Online: <http://cvxopt.org/documentation/coneprog.pdf>*, 2010.
- [48] A. Ben-Tal and A. Nemirovski, *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. Society for Industrial and Applied Mathematics, Jan. 2001.
- [49] F. Alizadeh and D. Goldfarb, “Second-order cone programming,” *Mathematical programming*, vol. 95, no. 1, pp. 3–51, 2003.
- [50] S. J. Wright, *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, Jan. 1997.
- [51] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, Mar. 2004.
- [52] J. Faraut and A. Korányi, *Analysis on symmetric cones*. Oxford university press, 1994.
- [53] Y. E. Nesterov and M. J. Todd, “Self-scaled barriers and interior-point methods for convex programming,” *Mathematics of Operations research*, vol. 22, no. 1, pp. 1–42, 1997.
- [54] Y. E. Nesterov and M. J. Todd, “Primal-dual interior-point methods for self-scaled cones,” *SIAM Journal on optimization*, vol. 8, no. 2, pp. 324–364, 1998.
- [55] J. R. Bunch and B. N. Parlett, “Direct methods for solving symmetric indefinite systems of linear equations,” *SIAM Journal on Numerical Analysis*, vol. 8, no. 4, pp. 639–655, 1971.
- [56] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, “OSQP: an operator splitting solver for quadratic programs,” *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.
- [57] R. J. Vanderbei, “Symmetric quasidefinite matrices,” *SIAM Journal on Optimization*, vol. 5, no. 1, pp. 100–113, 1995.
- [58] M. Yannakakis, “Computing the minimum fill-in is np-complete,” *SIAM Journal on Algebraic Discrete Methods*, vol. 2, p. 77–79, Mar. 1981.

- [59] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. USA: Cambridge University Press, 3 ed., 2007.
- [60] Kitware, Inc., *CMake: Cross-Platform Build System*, 2024.
- [61] D. Kouzoupias, A. Zanelli, H. Peyrl, and H. J. Ferreau, “Towards proper assessment of qp algorithms for embedded model predictive control,” in *2015 European Control Conference (ECC)*, pp. 2609–2616, IEEE, 2015.
- [62] I. Maros and C. Mészáros, “A repository of convex quadratic programming problems,” *Optimization Methods and Software*, vol. 11, p. 671–681, Jan. 1999.
- [63] T. A. Davis and Y. Hu, “The university of florida sparse matrix collection,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.
- [64] E. D. Dolan and J. J. Moré, “Benchmarking optimization software with performance profiles,” *Mathematical Programming*, vol. 91, p. 201–213, Jan. 2002.
- [65] R. Schwan, Y. Jiang, D. Kuhn, and C. N. Jones, “PIQP: A proximal interior-point quadratic programming solver,” in *2023 62nd IEEE Conference on Decision and Control (CDC)*, pp. 1088–1093, 2023.
- [66] M. Yuan and Y. Lin, “Model selection and estimation in regression with grouped variables,” *Journal of the Royal Statistical Society Series B*, vol. 68, pp. 49–67, 02 2006.
- [67] Y. Wang and S. Boyd, “Fast model predictive control using online optimization,” *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.
- [68] CVXPY, “Robust kalman filtering for vehicle tracking.” [https://www.cvxpy.org/examples/applications/robust\\_kalman.html](https://www.cvxpy.org/examples/applications/robust_kalman.html). Accessed: 2024-11-16.

## A Nonsymmetric Newton system

In this section we will show that Newton steps applied to Equations (5a), (5b), (5c), (5d) result in a nonsymmetric linear system which is less efficient to store and factor than the symmetric system presented in Equation (19).

We must first introduce the arrow matrix, a matrix that will allow us to rewrite the Jordan product, defined in Equation (6), as a matrix-vector multiplication. The arrow matrix can be written for vectors in the non-negative orthant and second-order cone as

$$\text{Arw}(x) = \begin{cases} \text{diag}(x) & \text{if } x \in \mathbb{R}_+^l \\ \begin{bmatrix} x_0 & x_1^\top \\ x_1 & x_0 I \end{bmatrix} & \text{if } x \in \mathcal{Q}^q. \end{cases}$$

For a vector in cone  $\mathcal{K}$  where

$$\mathcal{K} = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_K,$$

and  $\mathcal{C}_i$  is the non-negative orthant or a second-order cone, we can write the arrow matrix as

$$\text{Arw}(x) = \text{blkdiag}(\text{Arw}(x_1), \dots, \text{Arw}(x_K)) \quad \text{where } x_i \in \mathcal{C}_i,$$

Note that if  $x$  is in the interior of  $\mathcal{K}$ , then  $\text{Arw}(x)$  is positive definite and thus is invertible.

We can then write the Jordan product for two vectors  $u, v \in \mathcal{K}$  as

$$x \circ y = \text{Arw}(x)y,$$

We now linearize Equations (9a) - (9e) about the current iterate,  $(x_k, s_k, y_k, z_k)$ , write the Jordan products using the arrow matrix and get the linear system

$$\begin{bmatrix} P & 0 & A^\top & G^\top \\ 0 & \text{Arw}(z_k) & 0 & \text{Arw}(s_k) \\ A & 0 & 0 & 0 \\ G & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} -r_x \\ -r_s \\ -r_y \\ -r_z \end{bmatrix}$$

We can attempt to symmetrize the coefficient matrix by multiplying the second equation by  $\text{Arw}(s_k)^{-1}$  to get

$$\begin{bmatrix} P & 0 & A^\top & G^\top \\ 0 & \text{Arw}(s_k)^{-1}\text{Arw}(z_k) & 0 & I \\ A & 0 & 0 & 0 \\ G & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} -r_x \\ -\text{Arw}(s_k)^{-1}r_s \\ -r_y \\ -r_z \end{bmatrix}. \quad (31)$$

However, the coefficient matrix above will only be symmetric if  $\text{Arw}(s_k)^{-1}\text{Arw}(z_k)$  is symmetric. This can only be ensured if  $\mathcal{K}$  only consisted of the non-negative orthant and no second-order cones. In other words, the system in (31) would only be symmetric if Problem 1 was a quadratic program rather than a second-order cone program. Even if we were to eliminate  $\Delta s$  from Equation (31), we would still not have a symmetric linear system.

## B Custom LDL factorization performance

This section provides empirical evidence supporting our claims in Sections 1 and 4.1. Specifically, we show that sparse linear algebra incurs extra overhead from identifying and locating nonzero elements, leading to more CPU instructions, memory accesses and cache misses. In contrast, a custom implementation that hardcodes the exact memory accesses and floating point operations reduces these inefficiencies. We specifically compare QDLDL (used in QOCO) against our custom  $LDL^\top$  factorization (used in QOCO<sub>custom</sub>), since the most expensive operation in a generic implementation of Algorithm 1 is the  $LDL^\top$  factorization. To compare the two matrix factorization implementations, we use Linux `perf`, a profiling tool that provides CPU performance metrics, which will allow us to quantify the computational overhead of sparse versus custom factorizations.

As a representative example of a matrix we would factor, we consider the KKT matrix associated with the Linear Quadratic Regulator (LQR). The LQR problem is

$$\begin{aligned} \underset{x,u}{\text{minimize}} \quad & \frac{1}{2} \left( \sum_{k=1}^T x_k^\top Q_k x_k + \sum_{k=1}^{T-1} u_k^\top R_k u_k \right) \\ \text{subject to} \quad & x_{k+1} = A_k x_k + B_k u_k \quad \forall k \in [1, T-1] \\ & x_1 = x_{\text{init}}, \end{aligned}$$

and its associated KKT matrix is

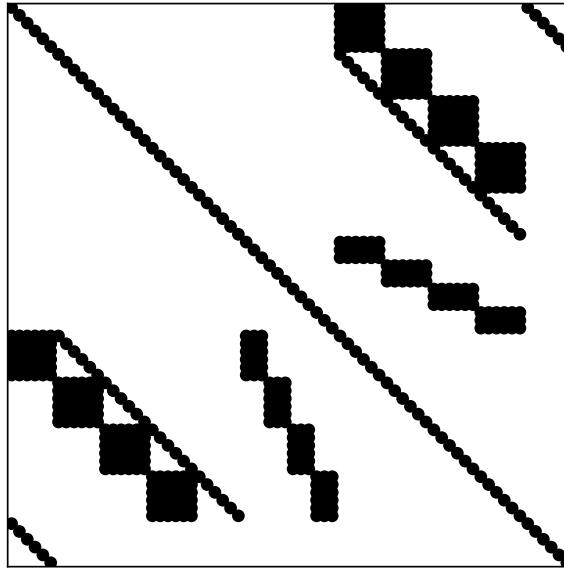
$$K = \begin{bmatrix} P & H^\top \\ H & -\epsilon I \end{bmatrix},$$

where

$$P = \text{blkdiag}(Q_1, Q_2, \dots, Q_T, R_1, R_2, \dots, R_{T-1})$$

$$H = \begin{bmatrix} A_1 & -I & & B_1 & & \\ & A_2 & -I & & B_2 & \\ & & \ddots & & & \ddots \\ & & & A_{N-1} & -I & \\ I & & & & & B_{N-1} \end{bmatrix}.$$

Notice that we apply static regularization to ensure that the  $(2, 2)$  block of  $K$  is negative definite, making  $K$  quasidefinite. The sparsity pattern of  $K$  is depicted by Figure 10.



**Figure 10: Sparsity pattern of LQR KKT matrix for  $T=5$**

For simplicity, we assume time-invariant system and cost matrices, i.e.,  $Q_k = R_k = I$  and fixed matrices  $A \in \mathbb{R}^{6 \times 6}$ ,  $B \in \mathbb{R}^{6 \times 3}$ . We choose the elements of  $A$  and  $B$  randomly and consider time horizons  $T$ :  $\{5, 15, 50, 75, 100\}$ . The actual elements of these matrices do not matter much, as our goal with these experiments is to understand the CPU behavior which (to first order) is unaffected by the value of elements in the matrices. To get accurate profiling results, we run each matrix factorization 1 million times to get the average runtime, number of instructions, number of L1 data cache loads, number of L1 cache misses, number of branches, and number of branch mispredictions.

From Table 1, we can see that the custom  $LDL^\top$  factorization is about four times faster than the sparse factorization because it does much less work: issuing fewer instructions, performing fewer memory accesses, and incurring fewer L1 cache misses. Although QDLDL has a far more branches than our custom  $LDL^\top$  implementation, the branches are quite predictable, and pipeline stalls are avoided.

By hardcoding the exact memory accesses and floating point operations, our custom  $LDL^\top$  factorization eliminates much of the overhead associated with sparse matrix operations. This substantial reduction in memory accesses and instructions leads to a significantly faster factorization.

**Table 1: Perf results for qdldl and custom  $LDL^\top$  factorization**

Size	Runtime (us)		Instructions		L1 cache loads		L1 cache misses		Branches		Branch misses	
	qdldl	custom	qdldl	custom	qdldl	custom	qdldl	custom	qdldl	custom	qdldl	custom
5	2.103	0.435	36650	4858	14049	1851	0	0	5419	5	1	0
15	7.507	1.777	132796	18053	50550	6442	572	27	19124	8	4	0
50	27.077	6.117	469289	62046	179709	21468	2539	1204	67081	17	11	1
75	38.848	8.485	709731	93912	279033	32211	3735	1828	101347	24	14	2
100	49.577	12.894	950101	125747	365981	43559	5567	2459	135597	35	18	5

## C Problem classes

In this section, we describe the five problem classes considered in our numerical experiments, along with various problem sizes and instances within each problem class. We use the notation  $\mathcal{U}(a, b)$  to denote a uniform distribution on the interval  $[a, b]$ ,  $\mathcal{N}(\mu, \sigma)$  for a Gaussian distribution with mean  $\mu$  and standard deviation  $\sigma$ ,  $0_{m \times n}$  for the matrix of all zeros in  $\mathbb{R}^{m \times n}$ , and  $I_n$  for the identity matrix in  $\mathbb{R}^{n \times n}$ .

### C.1 Robust Kalman filter

We consider the robust Kalman filtering problem for vehicle tracking, as outlined in [68]. This problem is a quadratic objective second-order cone program (SOCP) and takes the form

$$\begin{aligned} & \underset{x_k, w_k, v_k}{\text{minimize}} \quad \sum_{k=0}^{N-1} (\|w_k\|_2^2 + \tau \phi_\rho(v_k)) \\ & \text{subject to} \quad x_{k+1} = Ax_k + Bw_k \quad \forall k \in [0, N-1] \\ & \quad y_k = Cx_k + v_k \quad \forall k \in [0, N-1], \end{aligned} \tag{32}$$

where  $\phi_\rho$  is the Huber loss function

$$\phi_\rho(z) = \begin{cases} \|z\|_2^2 & \|z\|_2 \leq \rho \\ 2\rho\|z\|_2 - \rho^2 & \|z\|_2 > \rho. \end{cases}$$

In this problem,  $x_k \in \mathbb{R}^4$  represents the state of the vehicle at timestep  $k$ ,  $w_k \in \mathbb{R}^2$  is an unknown force vector acting on the vehicle at timestep  $k$ ,  $v_k \in \mathbb{R}^2$  is the measurement noise vector at timestep  $k$ , and  $y_k \in \mathbb{R}^2$  represents a noisy measurement of the vehicle's state at timestep  $k$ . The matrices  $A$  and  $B$  are known dynamics matrices for the system, and  $C$  is the known observation matrix. To reconstruct the vehicle's state trajectory, we solve Problem 32.

We use the system matrices

$$\begin{aligned} A &= \begin{bmatrix} 1 & 0 & (1 - \frac{\gamma}{2}\Delta t)\Delta t & 0 \\ 0 & 1 & 0 & (1 - \frac{\gamma}{2}\Delta t)\Delta t \\ 0 & 0 & 1 - \gamma\Delta t & 0 \\ 0 & 0 & 0 & 1 - \gamma\Delta t \end{bmatrix} \\ B &= \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \\ C &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \end{aligned}$$

which correspond to a two-dimensional double integrator model with linear velocity drag and unit mass, where the input is a force and we observe the position of the vehicle. We use  $\gamma = 0.05$  for the velocity drag parameter,  $\Delta t$  for the discretization time, and  $\rho = 2$  and  $\tau = 2$  for the Huber loss function.

**Problem sizes** To generate different problem sizes, we vary the number of timesteps  $N$ . We consider the following values for  $N$ :  $\{25, 50, 75, 125, 175, 225, 300, 375, 450, 500\}$ . For each of these problem sizes, we set  $\Delta t = T/(N-1)$ , where  $T = 50$  is the time horizon.

**Problem instances** For each problem size, we generate 20 random problem instances, corresponding to 20 different observation trajectories  $y_t$ . To do this, we first randomly generate the force and noise vectors  $w_t$  and  $v_t$ , respectively, for  $t \in [0, N - 1]$ . Then, we forward-simulate an initial state of  $x_0 = [0 \ 0 \ 0 \ 0]$  through the dynamics defined in the constraints of Problem 32. Each element of  $w_t$  and  $v_t$  is drawn from a standard Gaussian distribution  $\mathcal{N}(0, 1)$ . Additionally, for 20 % of the elements of  $v_t$ , we introduce outlier noise by drawing elements from  $\mathcal{N}(0, 20)$ , to simulate noise with a larger magnitude.

## C.2 Lossless convexification

The losslessly convexified powered-descent guidance problem is an optimal control problem that aims to compute a soft landing trajectory for a rocket while respecting relevant constraints. One important constraint is the lower thrust bound on the engine, which is nonconvex. However, it has been shown that this constraint admits a *lossless* convex relaxation [37, 38]. This means that the solution to the relaxed convex problem guarantees a solution to the original nonconvex problem. The losslessly convexified problem can be written as an SOCP

$$\begin{aligned} & \underset{x, z, u, \sigma}{\text{minimize}} && -z_T \\ & \text{subject to} && x_{k+1} = Ax_k + Bu_k + g \quad \forall k \in [0, T-1] \\ & && z_{k+1} = z_k - \alpha\sigma_k \Delta t \quad \forall k \in [0, T-1] \\ & && \|u_k\|_2 \leq \sigma_k \quad \forall k \in [0, T-1] \\ & && \log(m_{\text{wet}} - \alpha\rho_2 k \Delta t) \leq z_k \leq \log(m_{\text{wet}} - \alpha\rho_1 k \Delta t) \quad \forall k \in [0, T-1] \\ & && \mu_{1,k} \left[ 1 - [z_k - z_{0,k}] + \frac{[z_k - z_{0,k}]^2}{2} \right] \leq \sigma_k \leq \mu_{2,k} [1 - (z_k - z_{0,k})] \quad \forall k \in [0, T-1] \\ & && e_3^\top u_k \geq \sigma_k \cos(\theta_{\max}) \quad \forall k \in [0, T-1] \\ & && x_0 = x_{\text{init}}, \ z_0 = \log(m_{\text{wet}}), \ z_T \geq \log(m_{\text{dry}}), \end{aligned}$$

where  $x_k \in \mathbb{R}^6$  is the position and velocity of the rocket at timestep  $k$ ,  $u_k \in \mathbb{R}^3$  is the thrust per unit mass at timestep  $k$ ,  $z_k \in \mathbb{R}$  is the natural logarithm of the rocket's mass at timestep  $k$ , and  $\sigma_k \in \mathbb{R}$  is a slack variable at timestep  $k$ . Further we define  $z_{0,k} := \log(m_{\text{wet}} - \alpha\rho_2 k \Delta t)$ ,  $\mu_{1,k} := \rho_1 e^{-z_0}$ , and  $\mu_{2,k} := \rho_2 e^{-z_0}$ . The parameter  $\rho_1 \in \mathbb{R}$  is the minimum thrust of the engine,  $\rho_2 \in \mathbb{R}$  is the maximum thrust of the engine,  $\alpha \in \mathbb{R}$  is a mass depletion parameter that describes how efficient the engine is,  $m_{\text{wet}} \in \mathbb{R}$  is the sum of the dry mass of the rocket and the initial fuel mass,  $\theta_{\max} \in \mathbb{R}$  is the maximum angle from vertical that the thrust vector can make,  $x_{\text{init}} \in \mathbb{R}^6$  is the initial state of the rocket,  $\Delta t \in \mathbb{R}$  is the discretization time, and  $e_3$  is the third canonical basis vector in  $\mathbb{R}^3$ . The dynamics matrices  $A$  and  $B$ , and the vector  $g$ , are defined as

$$\begin{aligned} A &= \begin{bmatrix} I_3 & \Delta t(I_3) \\ 0_{3 \times 3} & I_3 \end{bmatrix} \\ B &= \begin{bmatrix} \frac{1}{2}\Delta t^2(I_3) \\ \Delta t(I_3) \end{bmatrix} \\ g &= [0 \ 0 \ -0.5g_0\Delta t^2 \ 0 \ 0 \ -g_0\Delta t]. \end{aligned}$$

For this problem class we use the parameters

$$g_0 = 9.807, \ \rho_1 = 100, \ \rho_2 = 500, \ m_{\text{dry}} = 25, \ m_{\text{wet}} = 35, \ \theta_{\max} = \pi/4, \ \alpha = 0.001.$$

**Problem sizes** To generate different problem sizes, we vary the number of timesteps  $T$ . The discretization time is computed as  $\Delta t = t_f/(T - 1)$ , where  $t_f = 20$ . We consider the following values for  $T$ :  $\{15, 50, 75, 100, 125, 150, 200, 250, 300, 350\}$ .

**Problem instances** For each problem size (i.e., for each number of timesteps), we generate 20 random problem instances, which involves generating a random initial state  $x_{\text{init}}$ . We generate  $x_{\text{init}}$  as

$$x_{\text{init}} \sim \begin{bmatrix} \mathcal{U}(-10, 10) \\ \mathcal{U}(-10, 10) \\ \mathcal{U}(200, 400) \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

### C.3 Group lasso

The group lasso problem is a quadratic objective SOCP. It is a variation of the least-squares regression problem, where the regression variables are partitioned into disjoint groups, and a regularization term is added to encourage group sparsity. Let  $x = [x^{(1)}, x^{(2)}, \dots, x^{(N)}]$  represent the partitioning of the regression variables, where each  $x^{(i)}$  is a group of variables. We can write the problem as

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2^2 + \lambda \sum_{i=1}^N \|x^{(i)}\|_2.$$

This can be written as an SOCP by introducing a slack variable  $t \in \mathbb{R}^N$ , and an auxiliary variable  $y \in \mathbb{R}^m$  to avoid computing  $A^\top A$  and maintain sparsity in the Hessian of the objective function. The problem is now

$$\begin{aligned} & \underset{x, t}{\text{minimize}} \quad \|y\|_2^2 + \lambda \sum_{i=1}^N t_i \\ & \text{subject to} \quad \|x^{(i)}\|_2 \leq t_i \quad \forall i \in [1, N] \\ & \quad y = Ax - b, \end{aligned}$$

where  $A \in \mathbb{R}^{m \times n}$ ,  $b \in \mathbb{R}^m$ .

**Problem sizes** To generate different problem sizes, we vary the number of groups  $N$ . We consider the following values for  $N$ :  $\{1, 2, 3, 4, 5, 8, 10, 12, 14, 16\}$ .

**Problem instances** For each problem size (i.e., for each number of groups) we generate 20 random problem instances, corresponding to the generation of the matrix  $A$  and the vector  $b$ . We set  $A$  to have  $m = 250N$  rows and  $n = 10N$  columns, where  $x$  is partitioned into  $N$  groups, each containing ten regression variables. The matrix  $A$  is sparse, with 10% of its elements nonzero, and the nonzero elements are drawn from the uniform distribution  $\mathcal{U}(0, 1)$ .

To generate  $b$ , we first generate  $\hat{x} \in \mathbb{R}^n$  and partition it into  $N$  groups of ten variables. Half of the groups are set to zero, and the remaining elements are drawn from the standard Gaussian distribution  $\mathcal{N}(0, 1)$ . Next, we generate an error vector  $e \in \mathbb{R}^m$  with elements drawn from  $\mathcal{N}(0, 1/n)$ . Finally, we compute  $b = A\hat{x} + e$ . Note that each problem instance will have a different sparsity pattern for  $A$ , which requires us to generate a new custom solver for each instance.

### C.4 Portfolio optimization

We consider the Markowitz portfolio optimization problem with a no-short-selling constraint [3, 51]. This problem is the QP

$$\begin{aligned}
& \underset{x}{\text{minimize}} && \gamma x^\top \Sigma x - \mu^\top x \\
& \text{subject to} && \sum_{i=1}^n x_i = 1 \\
& && x_i \geq 0 \quad \forall i \in [1, n],
\end{aligned}$$

where  $x \in \mathbb{R}^n$  is the allocation vector, with  $x_i$  representing the percentage of resources invested in asset  $i$ ,  $\mu \in \mathbb{R}^n$  is the vector of expected returns, and  $\Sigma \in \mathbb{S}_+^n$  is the covariance matrix of asset returns.

Typically, the return covariance matrix  $\Sigma$  is approximated as the sum of a low-rank matrix and a diagonal matrix. Thus, we write  $\Sigma = FF^\top + D$ , where  $F \in \mathbb{R}^{n \times k}$  is a factor matrix of rank  $k$ . We then add the auxiliary variable  $y$  and the constraint  $y = F^\top x$  to avoid computing  $FF^\top$  and maintain sparsity in the Hessian of the objective function. The final problem can be rewritten as

$$\begin{aligned}
& \underset{x,y}{\text{minimize}} && x^\top Dx + y^\top y - \gamma^{-1} \mu^\top x \\
& \text{subject to} && y = F^\top x \\
& && \sum_{i=1}^n x_i = 1 \\
& && x_i \geq 0 \quad \forall i \in [1, n].
\end{aligned} \tag{33}$$

**Problem sizes** To generate different problem sizes, we vary the number of factors,  $k$ . We consider the following values for  $k$ :  $\{2, 4, 6, 8, 10, 15, 20, 25, 30, 35\}$ .

**Problem instances** For each problem size (i.e., each number of factors), we generate 20 random problem instances. This corresponds to generating the matrices  $D$ ,  $F$ , and the vector  $\mu$ . We set the number of assets to  $n = 100k$ , and for all instances, we take  $\gamma = 1$ . The diagonal elements of  $D$  are drawn from  $\mathcal{N}(0, 1)$ , and the elements of  $\mu$  are drawn from  $\mathcal{N}(0, \sqrt{k})$ . The matrix  $F$  is a sparse matrix, with 50% of its elements being nonzero, and its nonzero elements are drawn from  $\mathcal{N}(0, 1)$ . Note that each problem instance will have a different sparsity pattern for  $F$ , which requires us to generate a new custom solver for each instance.

## C.5 Oscillating masses

The oscillating masses problem is an optimal control problem involving a system of  $N$  masses in one dimension, where each mass is connected to its neighboring mass by a spring, and the outermost two masses are attached to a fixed wall. The goal is to apply forces to each mass to bring the system to equilibrium [67]. This problem can be formulated as a QP

$$\begin{aligned}
& \underset{x,u}{\text{minimize}} && \frac{1}{2} \left( \sum_{k=0}^T x_k^\top Q x_k + \sum_{k=0}^{T-1} u_k^\top R u_k \right) \\
& \text{subject to} && x_{k+1} = Ax_k + Bu_k \quad \forall k \in [0, T-1] \\
& && x_0 = x_{\text{init}} \\
& && \|x_k\|_\infty \leq x_{\max} \quad \forall k \in [0, T] \\
& && \|u_k\|_\infty \leq u_{\max} \quad \forall k \in [0, T-1].
\end{aligned}$$

where  $x_k \in \mathbb{R}^{2N}$  is the position and velocity of the masses at timestep  $k$ ,  $u_k \in \mathbb{R}^N$  is force applied to each mass at timestep  $k$ , and  $Q \in \mathbb{S}_+^{2N}$  and  $R \in \mathbb{S}_+^N$  penalize deviation from equilibrium and control effort respectively.

The dynamics matrices  $A$  and  $B$  are derived from an exact discretization of the continuous-time linear dynamics, using a zero-order hold on control input

$$A = e^{A_c \Delta t}$$

$$B = A_c^{-1}(A - I_{2N})B_c,$$

where

$$A_c = \begin{bmatrix} 0_{N \times N} & I_N \\ L_N & 0_{N \times N} \end{bmatrix}$$

$$B_c = \begin{bmatrix} 0_{N \times N} \\ I_N \end{bmatrix}.$$

Here,  $L_N \in \mathbb{R}^{N \times N}$  is a symmetric tridiagonal matrix with  $-2$  on the main diagonal and  $1$  on the subdiagonal and superdiagonal.

For this problem class, we use the parameters

$$N = 4, \Delta t = 0.25, x_{\max} = 2, u_{\max} = 5.$$

**Problem sizes** To generate different problem sizes, we vary the number of timesteps  $T$ . We consider the following values for  $T$ :  $\{8, 20, 32, 44, 56, 76, 96, 116, 136, 156\}$ .

**Problem instances** For each problem size (i.e., each number of timesteps), we generate 20 random problem instances. This involves generating a random initial state  $x_{\text{init}}$ , as well as the matrices  $Q$  and  $R$ . We draw each element of  $x_{\text{init}}$  from  $\mathcal{N}(0, 1)$ , then clip the elements such that they lie within the interval  $[-0.9x_{\max}, 0.9x_{\max}]$ . This ensures that the initial state satisfies the state constraints, making the resulting problem feasible. The matrices  $Q$  and  $R$  are generated as diagonal matrices, with each diagonal element drawn from the uniform distribution  $\mathcal{U}(0, 10)$ .

## D Detailed numeric results

**Table 2: Iterations and solver runtimes for robust Kalman filter problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
ROBUST_KALMAN_FILTER_N_25_I_0	775	12	15	-	9	9	9	0.00082	0.00148	0.00618	0.00123	0.00043	0.00018
ROBUST_KALMAN_FILTER_N_25_I_1	775	10	15	-	10	9	9	0.00071	0.00154	0.00652	0.00129	0.00046	0.00018
ROBUST_KALMAN_FILTER_N_25_I_2	775	11	18	-	11	10	10	0.00077	0.00175	0.00701	0.00134	0.00047	0.00020
ROBUST_KALMAN_FILTER_N_25_I_3	775	10	17	-	10	9	9	0.00072	0.00171	0.00616	0.00129	0.00045	0.00018
ROBUST_KALMAN_FILTER_N_25_I_4	775	11	15	-	11	9	9	0.00077	0.00148	0.00630	0.00137	0.00043	0.00018
ROBUST_KALMAN_FILTER_N_25_I_5	775	10	14	-	9	8	8	0.00071	0.00145	0.00659	0.00124	0.00041	0.00016
ROBUST_KALMAN_FILTER_N_25_I_6	775	11	15	-	10	9	9	0.00076	0.00152	0.00656	0.00130	0.00045	0.00018
ROBUST_KALMAN_FILTER_N_25_I_7	775	10	17	-	11	9	9	0.00071	0.00168	0.00728	0.00136	0.00045	0.00018
ROBUST_KALMAN_FILTER_N_25_I_8	775	10	15	-	9	9	9	0.00071	0.00147	0.00614	0.00123	0.00044	0.00018
ROBUST_KALMAN_FILTER_N_25_I_9	775	9	18	-	11	9	9	0.00066	0.00182	0.00633	0.00133	0.00045	0.00018
ROBUST_KALMAN_FILTER_N_25_I_10	775	11	16	-	11	10	10	0.00077	0.00161	0.00668	0.00135	0.00048	0.00020
ROBUST_KALMAN_FILTER_N_25_I_11	775	12	19	-	11	10	10	0.00084	0.00185	0.00647	0.00134	0.00048	0.00020
ROBUST_KALMAN_FILTER_N_25_I_12	775	10	17	-	10	9	9	0.00071	0.00166	0.00576	0.00128	0.00045	0.00018
ROBUST_KALMAN_FILTER_N_25_I_13	775	8	18	-	10	8	8	0.00061	0.00182	0.00687	0.00129	0.00040	0.00016
ROBUST_KALMAN_FILTER_N_25_I_14	775	10	17	-	10	9	9	0.00071	0.00166	0.00582	0.00129	0.00044	0.00018
ROBUST_KALMAN_FILTER_N_25_I_15	775	10	16	-	9	8	8	0.00069	0.00162	0.00623	0.00123	0.00039	0.00016
ROBUST_KALMAN_FILTER_N_25_I_16	775	10	17	-	9	9	9	0.00072	0.00164	0.00665	0.00124	0.00043	0.00017
ROBUST_KALMAN_FILTER_N_25_I_17	775	9	17	-	10	9	9	0.00065	0.00175	0.00625	0.00130	0.00043	0.00018
ROBUST_KALMAN_FILTER_N_25_I_18	775	11	16	-	10	9	9	0.00076	0.00155	0.00579	0.00130	0.00044	0.00018
ROBUST_KALMAN_FILTER_N_25_I_19	775	11	19	-	10	10	10	0.00077	0.00186	0.00706	0.00129	0.00047	0.00020
ROBUST_KALMAN_FILTER_N_50_I_0	1550	13	18	-	11	12	12	0.00174	0.00364	0.01170	0.00272	0.00108	0.00048
ROBUST_KALMAN_FILTER_N_50_I_1	1550	14	19	-	11	12	12	0.00185	0.00362	0.01123	0.00275	0.00110	0.00048
ROBUST_KALMAN_FILTER_N_50_I_2	1550	11	20	-	10	10	10	0.00146	0.00393	0.01189	0.00268	0.00095	0.00041
ROBUST_KALMAN_FILTER_N_50_I_3	1550	11	19	-	9	10	10	0.00148	0.00375	0.01069	0.00246	0.00092	0.00040
ROBUST_KALMAN_FILTER_N_50_I_4	1550	11	20	-	9	10	10	0.00149	0.00381	0.01077	0.00245	0.00092	0.00040
ROBUST_KALMAN_FILTER_N_50_I_5	1550	11	19	-	9	10	10	0.00149	0.00377	0.01188	0.00243	0.00092	0.00041
ROBUST_KALMAN_FILTER_N_50_I_6	1550	12	19	-	10	11	11	0.00164	0.00373	0.01144	0.00260	0.00098	0.00045
ROBUST_KALMAN_FILTER_N_50_I_7	1550	10	17	-	8	9	9	0.00137	0.00347	0.01058	0.00228	0.00086	0.00037
ROBUST_KALMAN_FILTER_N_50_I_8	1550	11	20	-	9	10	10	0.00150	0.00386	0.01149	0.00257	0.00093	0.00040
ROBUST_KALMAN_FILTER_N_50_I_9	1550	12	24	-	9	10	10	0.00159	0.00463	0.01121	0.00246	0.00093	0.00040
ROBUST_KALMAN_FILTER_N_50_I_10	1550	11	21	-	9	9	9	0.00150	0.00410	0.01281	0.00245	0.00087	0.00037
ROBUST_KALMAN_FILTER_N_50_I_11	1550	11	19	-	9	10	10	0.00152	0.00377	0.01052	0.00246	0.00098	0.00040
ROBUST_KALMAN_FILTER_N_50_I_12	1550	12	18	-	9	11	11	0.00161	0.00358	0.01095	0.00244	0.00099	0.00044
ROBUST_KALMAN_FILTER_N_50_I_13	1550	13	21	-	12	12	12	0.00173	0.00411	0.01202	0.00287	0.00107	0.00048
ROBUST_KALMAN_FILTER_N_50_I_14	1550	13	19	-	12	12	12	0.00176	0.00377	0.01116	0.00286	0.00110	0.00049
ROBUST_KALMAN_FILTER_N_50_I_15	1550	11	19	-	10	10	10	0.00150	0.00379	0.01168	0.00259	0.00096	0.00041
ROBUST_KALMAN_FILTER_N_50_I_16	1550	12	21	-	9	10	10	0.00162	0.00414	0.01058	0.00243	0.00092	0.00040
ROBUST_KALMAN_FILTER_N_50_I_17	1550	13	21	-	10	11	11	0.00172	0.00398	0.01104	0.00255	0.00101	0.00044
ROBUST_KALMAN_FILTER_N_50_I_18	1550	11	18	-	9	10	10	0.00152	0.00356	0.01133	0.00243	0.00094	0.00041
ROBUST_KALMAN_FILTER_N_50_I_19	1550	13	19	-	11	12	12	0.00175	0.00372	0.01152	0.00270	0.00107	0.00048
ROBUST_KALMAN_FILTER_N_75_I_0	2325	13	18	-	12	11	11	0.00257	0.00556	0.01736	0.00523	0.00147	0.00068
ROBUST_KALMAN_FILTER_N_75_I_1	2325	12	19	-	10	10	10	0.00236	0.00565	0.01971	0.00477	0.00135	0.00062
ROBUST_KALMAN_FILTER_N_75_I_2	2325	11	21	-	11	10	10	0.00227	0.00648	0.01652	0.00496	0.00140	0.00061
ROBUST_KALMAN_FILTER_N_75_I_3	2325	13	21	-	13	11	11	0.00258	0.00616	0.01908	0.00540	0.00146	0.00068
ROBUST_KALMAN_FILTER_N_75_I_4	2325	13	22	-	12	12	12	0.00261	0.00666	0.01768	0.00523	0.00158	0.00073
ROBUST_KALMAN_FILTER_N_75_I_5	2325	12	23	-	12	11	11	0.00240	0.00680	0.01752	0.00521	0.00149	0.00068

**Table 2: Iterations and solver runtimes for robust Kalman filter problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
ROBUST_KALMAN_FILTER_N_75_L_6	2325	12	21	-	10	10	10	0.00241	0.00646	0.01624	0.00471	0.00142	0.00061
ROBUST_KALMAN_FILTER_N_75_L_7	2325	12	24	-	11	10	10	0.00236	0.00723	0.01604	0.00488	0.00140	0.00063
ROBUST_KALMAN_FILTER_N_75_L_8	2325	12	21	-	11	10	10	0.00235	0.00619	0.01583	0.00499	0.00135	0.00063
ROBUST_KALMAN_FILTER_N_75_L_9	2325	12	20	-	11	10	10	0.00241	0.00601	0.01868	0.00502	0.00136	0.00062
ROBUST_KALMAN_FILTER_N_75_L_10	2325	14	23	-	13	12	12	0.00276	0.00668	0.01850	0.00548	0.00159	0.00074
ROBUST_KALMAN_FILTER_N_75_L_11	2325	13	21	-	11	11	11	0.00254	0.00644	0.01732	0.00497	0.00147	0.00068
ROBUST_KALMAN_FILTER_N_75_L_12	2325	12	22	-	12	11	11	0.00246	0.00644	0.01831	0.00520	0.00149	0.00067
ROBUST_KALMAN_FILTER_N_75_L_13	2325	13	21	-	11	11	11	0.00256	0.00626	0.01602	0.00493	0.00144	0.00067
ROBUST_KALMAN_FILTER_N_75_L_14	2325	13	19	-	11	11	11	0.00258	0.00577	0.01731	0.00497	0.00150	0.00068
ROBUST_KALMAN_FILTER_N_75_L_15	2325	14	22	-	13	12	12	0.00275	0.00643	0.01746	0.00544	0.00161	0.00074
ROBUST_KALMAN_FILTER_N_75_L_16	2325	13	22	-	11	11	11	0.00256	0.00649	0.01592	0.00489	0.00147	0.00068
ROBUST_KALMAN_FILTER_N_75_L_17	2325	12	21	-	11	10	10	0.00237	0.00628	0.01567	0.00495	0.00137	0.00061
ROBUST_KALMAN_FILTER_N_75_L_18	2325	13	20	-	12	11	11	0.00256	0.00593	0.01666	0.00523	0.00147	0.00067
ROBUST_KALMAN_FILTER_N_75_L_19	2325	12	20	-	11	10	10	0.00236	0.00578	0.01685	0.00494	0.00141	0.00062
ROBUST_KALMAN_FILTER_N_125_L_0	3875	13	22	-	14	11	11	0.00427	0.01094	0.03237	0.00975	0.00244	0.00116
ROBUST_KALMAN_FILTER_N_125_L_1	3875	14	23	-	13	12	12	0.00459	0.01191	0.03259	0.00893	0.00264	0.00127
ROBUST_KALMAN_FILTER_N_125_L_2	3875	13	20	-	12	11	11	0.00432	0.01002	0.03394	0.00857	0.00247	0.00117
ROBUST_KALMAN_FILTER_N_125_L_3	3875	13	21	-	12	10	10	0.00424	0.01075	0.03132	0.00857	0.00225	0.00109
ROBUST_KALMAN_FILTER_N_125_L_4	3875	13	20	-	11	10	10	0.00428	0.01025	0.03491	0.00816	0.00226	0.00107
ROBUST_KALMAN_FILTER_N_125_L_5	3875	13	26	-	12	12	12	0.00430	0.01322	0.03353	0.00852	0.00264	0.00127
ROBUST_KALMAN_FILTER_N_125_L_6	3875	14	21	-	13	12	12	0.00458	0.01053	0.03500	0.00932	0.00266	0.00129
ROBUST_KALMAN_FILTER_N_125_L_7	3875	13	21	-	13	12	12	0.00436	0.01074	0.03184	0.00902	0.00267	0.00127
ROBUST_KALMAN_FILTER_N_125_L_8	3875	11	23	-	11	10	10	0.00369	0.01136	0.03438	0.00813	0.00226	0.00107
ROBUST_KALMAN_FILTER_N_125_L_9	3875	12	23	-	11	10	10	0.00402	0.01188	0.03560	0.00808	0.00227	0.00107
ROBUST_KALMAN_FILTER_N_125_L_10	3875	15	24	-	13	12	12	0.00484	0.01217	0.03564	0.00894	0.00264	0.00124
ROBUST_KALMAN_FILTER_N_125_L_11	3875	13	24	-	11	11	11	0.00436	0.01190	0.03224	0.00818	0.00245	0.00116
ROBUST_KALMAN_FILTER_N_125_L_12	3875	13	23	-	13	11	11	0.00436	0.01151	0.03240	0.00891	0.00246	0.00117
ROBUST_KALMAN_FILTER_N_125_L_13	3875	11	21	-	11	10	10	0.00375	0.01088	0.03264	0.00812	0.00225	0.00106
ROBUST_KALMAN_FILTER_N_125_L_14	3875	14	23	-	12	11	11	0.00460	0.01119	0.03209	0.00857	0.00247	0.00119
ROBUST_KALMAN_FILTER_N_125_L_15	3875	13	21	-	12	11	11	0.00426	0.01063	0.03336	0.00861	0.00249	0.00117
ROBUST_KALMAN_FILTER_N_125_L_16	3875	12	21	-	12	11	11	0.00404	0.01096	0.03084	0.00858	0.00249	0.00117
ROBUST_KALMAN_FILTER_N_125_L_17	3875	13	23	-	12	12	12	0.00425	0.01165	0.03524	0.00862	0.00269	0.00125
ROBUST_KALMAN_FILTER_N_125_L_18	3875	13	21	-	12	11	11	0.00430	0.01095	0.03311	0.00846	0.00249	0.00116
ROBUST_KALMAN_FILTER_N_125_L_19	3875	14	21	-	12	12	12	0.00452	0.01066	0.03371	0.00850	0.00265	0.00126
ROBUST_KALMAN_FILTER_N_175_L_0	5425	13	23	-	13	11	11	0.00593	0.01662	0.03626	0.01213	0.00344	0.00164
ROBUST_KALMAN_FILTER_N_175_L_1	5425	13	24	-	13	11	11	0.00608	0.01747	0.03183	0.01210	0.00348	0.00163
ROBUST_KALMAN_FILTER_N_175_L_2	5425	13	24	-	14	11	11	0.00597	0.01745	0.03231	0.01318	0.00341	0.00162
ROBUST_KALMAN_FILTER_N_175_L_3	5425	13	22	-	12	11	11	0.00595	0.01584	0.03097	0.01173	0.00340	0.00163
ROBUST_KALMAN_FILTER_N_175_L_4	5425	13	22	-	12	11	11	0.00597	0.01597	0.03450	0.01165	0.00344	0.00164
ROBUST_KALMAN_FILTER_N_175_L_5	5425	14	26	-	13	11	11	0.00642	0.01877	0.03103	0.01214	0.00344	0.00166
ROBUST_KALMAN_FILTER_N_175_L_6	5425	15	22	-	13	12	12	0.00680	0.01587	0.03132	0.01206	0.00372	0.00180
ROBUST_KALMAN_FILTER_N_175_L_7	5425	14	25	-	13	11	11	0.00642	0.01788	0.03193	0.01218	0.00341	0.00166
ROBUST_KALMAN_FILTER_N_175_L_8	5425	14	24	-	13	11	11	0.00636	0.01728	0.03399	0.01199	0.00339	0.00166
ROBUST_KALMAN_FILTER_N_175_L_9	5425	14	24	-	14	12	12	0.00639	0.01703	0.03370	0.01262	0.00367	0.00179
ROBUST_KALMAN_FILTER_N_175_L_10	5425	13	24	-	12	11	11	0.00604	0.01771	0.03132	0.01146	0.00337	0.00168
ROBUST_KALMAN_FILTER_N_175_L_11	5425	13	24	-	14	12	12	0.00600	0.01756	0.03318	0.01306	0.00368	0.00177

**Table 2: Iterations and solver runtimes for robust Kalman filter problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
ROBUST_KALMAN_FILTER_N_175_L_12	5425	14	23	-	14	12	12	0.00636	0.01663	0.03131	0.01259	0.00388	0.00176
ROBUST_KALMAN_FILTER_N_175_L_13	5425	12	26	-	13	10	10	0.00559	0.01881	0.03124	0.01259	0.00313	0.00150
ROBUST_KALMAN_FILTER_N_175_L_14	5425	13	22	-	12	10	10	0.00599	0.01625	0.02931	0.01162	0.00319	0.00153
ROBUST_KALMAN_FILTER_N_175_L_15	5425	14	26	-	13	13	13	0.00641	0.01890	0.03223	0.01215	0.00396	0.00194
ROBUST_KALMAN_FILTER_N_175_L_16	5425	12	22	-	12	11	11	0.00566	0.01615	0.03234	0.01155	0.00344	0.00165
ROBUST_KALMAN_FILTER_N_175_L_17	5425	14	25	-	13	11	11	0.00644	0.01818	0.03067	0.01220	0.00341	0.00167
ROBUST_KALMAN_FILTER_N_175_L_18	5425	13	25	-	12	12	12	0.00610	0.01829	0.03079	0.01211	0.00365	0.00178
ROBUST_KALMAN_FILTER_N_175_L_19	5425	13	24	-	15	11	11	0.00604	0.01730	0.03026	0.01383	0.00338	0.00164
ROBUST_KALMAN_FILTER_N_225_L_0	6975	13	24	-	12	11	-	0.00769	0.02292	0.03904	0.01513	0.00441	-
ROBUST_KALMAN_FILTER_N_225_L_1	6975	13	24	-	12	11	-	0.00768	0.02281	0.03951	0.01509	0.00436	-
ROBUST_KALMAN_FILTER_N_225_L_2	6975	13	26	-	13	12	-	0.00775	0.02437	0.03929	0.01586	0.00472	-
ROBUST_KALMAN_FILTER_N_225_L_3	6975	12	23	-	12	10	-	0.00729	0.02152	0.04337	0.01506	0.00401	-
ROBUST_KALMAN_FILTER_N_225_L_4	6975	13	25	-	13	11	-	0.00771	0.02463	0.04028	0.01566	0.00462	-
ROBUST_KALMAN_FILTER_N_225_L_5	6975	13	25	-	12	11	-	0.00769	0.02433	0.04326	0.01502	0.00453	-
ROBUST_KALMAN_FILTER_N_225_L_6	6975	13	26	-	13	11	-	0.00777	0.02557	0.04165	0.01581	0.00439	-
ROBUST_KALMAN_FILTER_N_225_L_7	6975	13	24	-	14	11	-	0.00774	0.02269	0.04116	0.01626	0.00455	-
ROBUST_KALMAN_FILTER_N_225_L_8	6975	13	24	-	12	11	-	0.00770	0.02280	0.03837	0.01508	0.00461	-
ROBUST_KALMAN_FILTER_N_225_L_9	6975	12	25	-	13	11	-	0.00718	0.02402	0.03773	0.01582	0.00451	-
ROBUST_KALMAN_FILTER_N_225_L_10	6975	14	26	-	13	12	-	0.00819	0.02443	0.04265	0.01576	0.00476	-
ROBUST_KALMAN_FILTER_N_225_L_11	6975	14	25	-	14	12	-	0.00824	0.02341	0.04128	0.01714	0.00497	-
ROBUST_KALMAN_FILTER_N_225_L_12	6975	12	25	-	11	10	-	0.00712	0.02437	0.04371	0.01434	0.00404	-
ROBUST_KALMAN_FILTER_N_225_L_13	6975	13	24	-	13	11	-	0.00762	0.02290	0.04400	0.01571	0.00451	-
ROBUST_KALMAN_FILTER_N_225_L_14	6975	13	24	-	15	11	-	0.00778	0.02273	0.04209	0.01791	0.00471	-
ROBUST_KALMAN_FILTER_N_225_L_15	6975	14	24	-	12	10	-	0.00818	0.02301	0.04042	0.01514	0.00451	-
ROBUST_KALMAN_FILTER_N_225_L_16	6975	13	24	-	12	11	-	0.00771	0.02254	0.04902	0.01508	0.00452	-
ROBUST_KALMAN_FILTER_N_225_L_17	6975	14	23	-	12	11	-	0.00817	0.02096	0.04042	0.01522	0.00446	-
ROBUST_KALMAN_FILTER_N_225_L_18	6975	13	25	-	13	11	-	0.00774	0.02364	0.04153	0.01579	0.00436	-
ROBUST_KALMAN_FILTER_N_225_L_19	6975	13	25	-	13	12	-	0.00775	0.02357	0.04061	0.01564	0.00471	-
ROBUST_KALMAN_FILTER_N_300_L_0	9300	14	23	-	13	11	-	0.01091	0.03031	0.05073	0.02093	0.00585	-
ROBUST_KALMAN_FILTER_N_300_L_1	9300	13	25	-	13	11	-	0.01028	0.03182	0.05255	0.02074	0.00591	-
ROBUST_KALMAN_FILTER_N_300_L_2	9300	14	25	-	12	10	-	0.01087	0.03232	0.05360	0.01995	0.00539	-
ROBUST_KALMAN_FILTER_N_300_L_3	9300	13	25	-	12	11	-	0.01032	0.03302	0.05028	0.01985	0.00587	-
ROBUST_KALMAN_FILTER_N_300_L_4	9300	13	23	-	13	11	-	0.01033	0.03029	0.05402	0.02151	0.00584	-
ROBUST_KALMAN_FILTER_N_300_L_5	9300	13	25	-	13	11	-	0.01028	0.03236	0.05103	0.02079	0.00621	-
ROBUST_KALMAN_FILTER_N_300_L_6	9300	13	24	-	12	11	-	0.01024	0.03124	0.05269	0.01998	0.00599	-
ROBUST_KALMAN_FILTER_N_300_L_7	9300	14	25	-	13	11	-	0.01096	0.03146	0.05782	0.02094	0.00612	-
ROBUST_KALMAN_FILTER_N_300_L_8	9300	13	25	-	13	10	-	0.01023	0.03290	0.05050	0.02072	0.00536	-
ROBUST_KALMAN_FILTER_N_300_L_9	9300	14	26	-	12	11	-	0.01093	0.03401	0.05052	0.01978	0.00599	-
ROBUST_KALMAN_FILTER_N_300_L_10	9300	13	24	-	12	11	-	0.01030	0.03104	0.04965	0.02001	0.00583	-
ROBUST_KALMAN_FILTER_N_300_L_11	9300	13	25	-	14	11	-	0.01033	0.03223	0.04701	0.02167	0.00578	-
ROBUST_KALMAN_FILTER_N_300_L_12	9300	14	26	-	13	11	-	0.01110	0.03346	0.05214	0.02155	0.00584	-
ROBUST_KALMAN_FILTER_N_300_L_13	9300	13	25	-	14	11	-	0.01037	0.03264	0.05290	0.02281	0.00585	-
ROBUST_KALMAN_FILTER_N_300_L_14	9300	14	25	-	13	11	-	0.01104	0.03283	0.05176	0.02094	0.00585	-
ROBUST_KALMAN_FILTER_N_300_L_15	9300	13	27	-	13	11	-	0.01018	0.03389	0.05460	0.02070	0.00577	-
ROBUST_KALMAN_FILTER_N_300_L_16	9300	16	26	-	14	12	-	0.01230	0.03374	0.05565	0.02153	0.00629	-
ROBUST_KALMAN_FILTER_N_300_L_17	9300	13	23	-	12	10	-	0.01023	0.02993	0.05101	0.01985	0.00539	-

**Table 2: Iterations and solver runtimes for robust Kalman filter problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
ROBUST_KALMAN_FILTER_N_300_L_18	9300	13	24	-	12	10	-	0.01019	0.03166	0.05298	0.01995	0.00538	-
ROBUST_KALMAN_FILTER_N_300_L_19	9300	13	25	-	14	11	-	0.01033	0.03166	0.05133	0.02138	0.00581	-
ROBUST_KALMAN_FILTER_N_375_L_0	11625	13	26	-	12	11	-	0.01295	0.04382	0.06739	0.02676	0.00735	-
ROBUST_KALMAN_FILTER_N_375_L_1	11625	13	26	-	14	10	-	0.01291	0.04279	0.06874	0.02889	0.00674	-
ROBUST_KALMAN_FILTER_N_375_L_2	11625	12	26	-	12	11	-	0.01215	0.04399	0.06708	0.02649	0.00731	-
ROBUST_KALMAN_FILTER_N_375_L_3	11625	13	26	-	13	11	-	0.01293	0.04319	0.06396	0.02766	0.00730	-
ROBUST_KALMAN_FILTER_N_375_L_4	11625	13	24	-	15	11	-	0.01297	0.04012	0.06814	0.03156	0.00728	-
ROBUST_KALMAN_FILTER_N_375_L_5	11625	12	23	-	12	10	-	0.01214	0.03820	0.06889	0.02659	0.00704	-
ROBUST_KALMAN_FILTER_N_375_L_6	11625	14	25	-	13	12	-	0.01376	0.04189	0.06935	0.02769	0.00789	-
ROBUST_KALMAN_FILTER_N_375_L_7	11625	14	25	-	13	11	-	0.01381	0.04206	0.07491	0.02782	0.00727	-
ROBUST_KALMAN_FILTER_N_375_L_8	11625	13	26	-	14	11	-	0.01294	0.04346	0.07001	0.02881	0.00724	-
ROBUST_KALMAN_FILTER_N_375_L_9	11625	14	27	-	13	11	-	0.01389	0.04595	0.06740	0.02766	0.00733	-
ROBUST_KALMAN_FILTER_N_375_L_10	11625	14	25	-	12	11	-	0.01390	0.04195	0.07056	0.02656	0.00729	-
ROBUST_KALMAN_FILTER_N_375_L_11	11625	12	26	-	13	10	-	0.01212	0.04352	0.06386	0.02762	0.00665	-
ROBUST_KALMAN_FILTER_N_375_L_12	11625	13	24	-	14	11	-	0.01294	0.03993	0.06879	0.02891	0.00729	-
ROBUST_KALMAN_FILTER_N_375_L_13	11625	13	27	-	12	11	-	0.01298	0.04403	0.06671	0.02637	0.00735	-
ROBUST_KALMAN_FILTER_N_375_L_14	11625	14	26	-	14	11	-	0.01387	0.04313	0.06759	0.02902	0.00731	-
ROBUST_KALMAN_FILTER_N_375_L_15	11625	14	26	-	13	11	-	0.01382	0.04325	0.07478	0.02744	0.00736	-
ROBUST_KALMAN_FILTER_N_375_L_16	11625	13	27	-	13	12	-	0.01301	0.04496	0.06681	0.02751	0.00791	-
ROBUST_KALMAN_FILTER_N_375_L_17	11625	13	26	-	15	11	-	0.01300	0.04334	0.06621	0.03148	0.00726	-
ROBUST_KALMAN_FILTER_N_375_L_18	11625	13	24	-	13	11	-	0.01283	0.03984	0.06962	0.02743	0.00733	-
ROBUST_KALMAN_FILTER_N_375_L_19	11625	14	25	-	14	11	-	0.01394	0.04152	0.07587	0.02906	0.00733	-
ROBUST_KALMAN_FILTER_N_450_L_0	13950	13	26	-	14	11	-	0.01558	0.05342	0.08442	0.03470	0.00877	-
ROBUST_KALMAN_FILTER_N_450_L_1	13950	13	27	-	13	11	-	0.01559	0.05502	0.08003	0.03476	0.00872	-
ROBUST_KALMAN_FILTER_N_450_L_2	13950	13	26	-	13	11	-	0.01556	0.05468	0.08639	0.03301	0.00879	-
ROBUST_KALMAN_FILTER_N_450_L_3	13950	13	28	-	14	10	-	0.01539	0.05612	0.09875	0.03590	0.00811	-
ROBUST_KALMAN_FILTER_N_450_L_4	13950	13	31	-	13	11	-	0.01551	0.06150	0.08927	0.03265	0.00875	-
ROBUST_KALMAN_FILTER_N_450_L_5	13950	14	26	-	14	11	-	0.01651	0.05324	0.09395	0.03425	0.00880	-
ROBUST_KALMAN_FILTER_N_450_L_6	13950	14	29	-	14	11	-	0.01651	0.05747	0.08958	0.03447	0.00876	-
ROBUST_KALMAN_FILTER_N_450_L_7	13950	13	27	-	15	11	-	0.01555	0.05310	0.08318	0.03585	0.00877	-
ROBUST_KALMAN_FILTER_N_450_L_8	13950	14	28	-	14	12	-	0.01658	0.05595	0.08977	0.03436	0.00967	-
ROBUST_KALMAN_FILTER_N_450_L_9	13950	13	29	-	13	10	-	0.01544	0.05851	0.08958	0.03259	0.00811	-
ROBUST_KALMAN_FILTER_N_450_L_10	13950	14	27	-	14	10	-	0.01656	0.05439	0.10099	0.03602	0.00807	-
ROBUST_KALMAN_FILTER_N_450_L_11	13950	12	25	-	13	10	-	0.01446	0.05188	0.09591	0.03300	0.00833	-
ROBUST_KALMAN_FILTER_N_450_L_12	13950	15	26	-	13	12	-	0.01738	0.05263	0.09965	0.03307	0.00944	-
ROBUST_KALMAN_FILTER_N_450_L_13	13950	13	26	-	14	11	-	0.01559	0.05297	0.09955	0.03434	0.00880	-
ROBUST_KALMAN_FILTER_N_450_L_14	13950	13	26	-	13	10	-	0.01553	0.05320	0.07826	0.03279	0.00811	-
ROBUST_KALMAN_FILTER_N_450_L_15	13950	14	28	-	13	10	-	0.01669	0.05794	0.08767	0.03325	0.00813	-
ROBUST_KALMAN_FILTER_N_450_L_16	13950	14	29	-	14	11	-	0.01654	0.05936	0.08538	0.03584	0.00873	-
ROBUST_KALMAN_FILTER_N_450_L_17	13950	13	27	-	13	11	-	0.01547	0.05590	0.08448	0.03308	0.00878	-
ROBUST_KALMAN_FILTER_N_450_L_18	13950	14	26	-	13	13	-	0.01660	0.05390	0.08806	0.03296	0.01010	-
ROBUST_KALMAN_FILTER_N_450_L_19	13950	13	25	-	14	11	-	0.01542	0.05115	0.09033	0.03430	0.00879	-
ROBUST_KALMAN_FILTER_N_500_L_0	15500	14	25	-	12	11	-	0.01843	0.05906	0.10041	0.03531	0.00977	-
ROBUST_KALMAN_FILTER_N_500_L_1	15500	13	25	-	14	11	-	0.01730	0.05758	0.09165	0.03863	0.00972	-
ROBUST_KALMAN_FILTER_N_500_L_2	15500	13	25	-	12	10	-	0.01713	0.05923	0.09384	0.03503	0.00894	-
ROBUST_KALMAN_FILTER_N_500_L_3	15500	13	26	-	12	11	-	0.01729	0.06229	0.08355	0.03555	0.00964	-

**Table 2: Iterations and solver runtimes for robust Kalman filter problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
ROBUST_KALMAN_FILTER_N_500_L_4	15500	13	29	-	14	11	-	0.01718	0.06716	0.08791	0.03857	0.00981	-
ROBUST_KALMAN_FILTER_N_500_L_5	15500	13	25	-	13	10	-	0.01721	0.05794	0.08527	0.03686	0.00897	-
ROBUST_KALMAN_FILTER_N_500_L_6	15500	14	29	-	14	11	-	0.01838	0.06719	0.08700	0.03852	0.00976	-
ROBUST_KALMAN_FILTER_N_500_L_7	15500	13	25	-	13	11	-	0.01732	0.05852	0.09098	0.03673	0.00975	-
ROBUST_KALMAN_FILTER_N_500_L_8	15500	14	27	-	14	11	-	0.01850	0.06306	0.09047	0.03830	0.00987	-
ROBUST_KALMAN_FILTER_N_500_L_9	15500	13	30	-	13	10	-	0.01717	0.06862	0.08998	0.03689	0.00901	-
ROBUST_KALMAN_FILTER_N_500_L_10	15500	14	26	-	13	11	-	0.01861	0.05992	0.09209	0.03657	0.00973	-
ROBUST_KALMAN_FILTER_N_500_L_11	15500	13	28	-	13	11	-	0.01721	0.06351	0.08248	0.03656	0.00975	-
ROBUST_KALMAN_FILTER_N_500_L_12	15500	13	26	-	14	10	-	0.01726	0.06046	0.08330	0.03811	0.00894	-
ROBUST_KALMAN_FILTER_N_500_L_13	15500	12	25	-	13	10	-	0.01613	0.05756	0.08654	0.03681	0.00897	-
ROBUST_KALMAN_FILTER_N_500_L_14	15500	13	26	-	14	11	-	0.01735	0.05895	0.09669	0.04051	0.00978	-
ROBUST_KALMAN_FILTER_N_500_L_15	15500	14	27	-	15	11	-	0.01839	0.06165	0.09525	0.04201	0.00972	-
ROBUST_KALMAN_FILTER_N_500_L_16	15500	13	26	-	14	11	-	0.01729	0.05945	0.09317	0.03859	0.00977	-
ROBUST_KALMAN_FILTER_N_500_L_17	15500	13	27	-	14	11	-	0.01714	0.06225	0.09272	0.03811	0.00970	-
ROBUST_KALMAN_FILTER_N_500_L_18	15500	13	26	-	14	11	-	0.01720	0.05936	0.08589	0.03816	0.00980	-
ROBUST_KALMAN_FILTER_N_500_L_19	15500	13	28	-	14	11	-	0.01728	0.06322	0.09300	0.03831	0.00972	-

**Table 3: Iterations and solver runtimes for lossless convexification problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
LCVX_N_15_I_0	644	8	7	-	9	9	9	0.00074	0.00060	0.00438	0.00111	0.00049	0.00023
LCVX_N_15_I_1	644	8	7	-	9	9	9	0.00072	0.00060	0.00428	0.00110	0.00049	0.00024
LCVX_N_15_I_2	644	10	7	-	9	10	10	0.00085	0.00062	0.00458	0.00099	0.00055	0.00025
LCVX_N_15_I_3	644	8	7	-	9	10	10	0.00073	0.00057	0.00442	0.00107	0.00052	0.00026
LCVX_N_15_I_4	644	9	7	-	9	9	9	0.00078	0.00062	0.00472	0.00109	0.00049	0.00023
LCVX_N_15_I_5	644	10	7	-	10	9	9	0.00084	0.00060	0.00446	0.00115	0.00049	0.00023
LCVX_N_15_I_6	644	7	7	-	9	10	10	0.00068	0.00058	0.00417	0.00109	0.00053	0.00025
LCVX_N_15_I_7	644	9	7	-	9	10	10	0.00078	0.00060	0.00450	0.00108	0.00053	0.00025
LCVX_N_15_I_8	644	8	7	-	10	9	9	0.00074	0.00059	0.00444	0.00115	0.00048	0.00023
LCVX_N_15_I_9	644	8	7	-	9	10	10	0.00073	0.00057	0.00458	0.00109	0.00053	0.00025
LCVX_N_15_I_10	644	8	7	-	9	9	9	0.00072	0.00060	0.00441	0.00109	0.00049	0.00023
LCVX_N_15_I_11	644	7	7	-	9	9	9	0.00067	0.00059	0.00413	0.00104	0.00049	0.00023
LCVX_N_15_I_12	644	9	7	-	9	10	10	0.00087	0.00058	0.00439	0.00109	0.00054	0.00026
LCVX_N_15_I_13	644	7	7	-	9	9	9	0.00069	0.00060	0.00426	0.00107	0.00050	0.00023
LCVX_N_15_I_14	644	8	7	-	9	10	10	0.00074	0.00058	0.00441	0.00109	0.00054	0.00025
LCVX_N_15_I_15	644	8	7	-	10	10	10	0.00080	0.00059	0.00453	0.00117	0.00054	0.00026
LCVX_N_15_I_16	644	7	7	-	9	9	9	0.00066	0.00058	0.00415	0.00108	0.00049	0.00023
LCVX_N_15_I_17	644	8	7	-	9	10	10	0.00072	0.00060	0.00455	0.00108	0.00054	0.00025
LCVX_N_15_I_18	644	8	7	-	9	9	9	0.00073	0.00061	0.00441	0.00110	0.00050	0.00023
LCVX_N_15_I_19	644	9	7	-	9	10	10	0.00079	0.00060	0.00462	0.00108	0.00053	0.00025
LCVX_N_50_I_0	2114	9	8	-	13	10	10	0.00282	0.00210	0.01095	0.00445	0.00169	0.00091
LCVX_N_50_I_1	2114	9	7	-	16	10	10	0.00266	0.00183	0.01104	0.00504	0.00171	0.00091
LCVX_N_50_I_2	2114	9	7	-	15	10	10	0.00269	0.00190	0.01103	0.00486	0.00170	0.00092
LCVX_N_50_I_3	2114	10	9	-	16	10	10	0.00298	0.00238	0.01180	0.00504	0.00170	0.00091
LCVX_N_50_I_4	2114	8	7	-	15	10	10	0.00248	0.00185	0.01045	0.00484	0.00170	0.00092
LCVX_N_50_I_5	2114	8	7	-	13	10	10	0.00246	0.00187	0.00975	0.00448	0.00169	0.00091
LCVX_N_50_I_6	2114	9	7	-	16	10	10	0.00268	0.00186	0.01130	0.00502	0.00171	0.00091
LCVX_N_50_I_7	2114	8	8	-	13	10	10	0.00247	0.00224	0.01052	0.00445	0.00171	0.00091
LCVX_N_50_I_8	2114	10	8	-	16	10	10	0.00281	0.00218	0.01099	0.00502	0.00169	0.00090
LCVX_N_50_I_9	2114	10	8	-	16	10	10	0.00284	0.00220	0.01120	0.00504	0.00168	0.00093
LCVX_N_50_I_10	2114	10	8	-	16	10	10	0.00283	0.00219	0.01113	0.00502	0.00171	0.00092
LCVX_N_50_I_11	2114	10	8	-	15	10	10	0.00298	0.00209	0.01104	0.00487	0.00170	0.00090
LCVX_N_50_I_12	2114	9	7	-	15	10	10	0.00262	0.00187	0.01087	0.00487	0.00172	0.00092
LCVX_N_50_I_13	2114	9	8	-	14	10	10	0.00280	0.00205	0.01061	0.00464	0.00170	0.00091
LCVX_N_50_I_14	2114	10	7	-	16	10	10	0.00282	0.00186	0.01116	0.00506	0.00169	0.00091
LCVX_N_50_I_15	2114	9	8	-	15	10	10	0.00263	0.00215	0.01134	0.00482	0.00173	0.00090
LCVX_N_50_I_16	2114	9	7	-	15	10	10	0.00267	0.00188	0.01109	0.00485	0.00172	0.00089
LCVX_N_50_I_17	2114	10	9	-	16	10	10	0.00298	0.00240	0.01185	0.00501	0.00169	0.00091
LCVX_N_50_I_18	2114	9	8	-	13	10	10	0.00267	0.00211	0.01076	0.00447	0.00172	0.00090
LCVX_N_50_I_19	2114	12	9	-	17	10	10	0.00335	0.00235	0.01167	0.00525	0.00168	0.00091
LCVX_N_75_I_0	3164	10	8	-	16	10	10	0.00423	0.00324	0.01509	0.00753	0.00247	0.00141
LCVX_N_75_I_1	3164	10	9	-	17	10	10	0.00441	0.00350	0.01635	0.00782	0.00248	0.00142
LCVX_N_75_I_2	3164	10	8	-	15	10	10	0.00447	0.00309	0.01605	0.00729	0.00249	0.00145
LCVX_N_75_I_3	3164	8	8	-	15	10	10	0.00371	0.00332	0.01549	0.00725	0.00249	0.00141
LCVX_N_75_I_4	3164	10	9	-	17	10	10	0.00442	0.00351	0.01552	0.00779	0.00250	0.00145
LCVX_N_75_I_5	3164	9	8	-	16	10	10	0.00396	0.00327	0.01546	0.00753	0.00250	0.00143

**Table 3: Iterations and solver runtimes for lossless convexification problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
LCVX_N_75_1_6	3164	8	8	-	12	10	10	0.00365	0.00323	0.01358	0.00646	0.00249	0.00141
LCVX_N_75_1_7	3164	11	9	-	17	10	10	0.00471	0.00349	0.01546	0.00779	0.00251	0.00143
LCVX_N_75_1_8	3164	10	8	-	15	10	10	0.00446	0.00310	0.01592	0.00723	0.00249	0.00142
LCVX_N_75_1_9	3164	11	9	-	17	10	10	0.00474	0.00348	0.01640	0.00778	0.00252	0.00136
LCVX_N_75_1_10	3164	10	8	-	16	10	10	0.00419	0.00322	0.01604	0.00751	0.00250	0.00142
LCVX_N_75_1_11	3164	8	8	-	16	10	10	0.00366	0.00315	0.01535	0.00751	0.00247	0.00141
LCVX_N_75_1_12	3164	11	9	-	17	10	10	0.00479	0.00350	0.01714	0.00779	0.00248	0.00141
LCVX_N_75_1_13	3164	9	8	-	13	10	10	0.00421	0.00300	0.01504	0.00671	0.00248	0.00141
LCVX_N_75_1_14	3164	10	9	-	17	10	10	0.00455	0.00353	0.01655	0.00779	0.00248	0.00140
LCVX_N_75_1_15	3164	10	9	-	16	10	10	0.00442	0.00352	0.01661	0.00752	0.00250	0.00147
LCVX_N_75_1_16	3164	10	8	-	16	10	10	0.00447	0.00308	0.01539	0.00752	0.00250	0.00142
LCVX_N_75_1_17	3164	11	9	-	17	10	10	0.00473	0.00352	0.01640	0.00779	0.00251	0.00142
LCVX_N_75_1_18	3164	10	8	-	17	10	10	0.00419	0.00326	0.01578	0.00779	0.00248	0.00141
LCVX_N_75_1_19	3164	9	8	-	16	10	10	0.00384	0.00309	0.01508	0.00753	0.00261	0.00141
LCVX_N_100_1_0	4214	9	8	-	16	10	10	0.00550	0.00424	0.01873	0.00996	0.00341	0.00200
LCVX_N_100_1_1	4214	10	9	-	16	10	10	0.00612	0.00462	0.01938	0.01000	0.00336	0.00196
LCVX_N_100_1_2	4214	12	9	-	17	11	11	0.00680	0.00450	0.02016	0.01029	0.00365	0.00214
LCVX_N_100_1_3	4214	10	9	-	15	10	10	0.00618	0.00471	0.01943	0.01001	0.00344	0.00190
LCVX_N_100_1_4	4214	8	8	-	15	10	10	0.00454	0.00416	0.01976	0.00965	0.00335	0.00195
LCVX_N_100_1_5	4214	11	9	-	17	10	10	0.00655	0.00469	0.01987	0.01032	0.00338	0.00195
LCVX_N_100_1_6	4214	10	9	-	16	10	10	0.00596	0.00461	0.01895	0.01004	0.00337	0.00198
LCVX_N_100_1_7	4214	11	9	-	18	11	11	0.00625	0.00447	0.01985	0.01070	0.00365	0.00210
LCVX_N_100_1_8	4214	9	8	-	15	10	10	0.00530	0.00422	0.01903	0.01003	0.00333	0.00195
LCVX_N_100_1_9	4214	9	8	-	16	10	10	0.00495	0.00414	0.01832	0.00998	0.00339	0.00198
LCVX_N_100_1_10	4214	10	8	-	16	10	10	0.00580	0.00405	0.01931	0.01000	0.00335	0.00196
LCVX_N_100_1_11	4214	10	8	-	16	10	10	0.00579	0.00397	0.02012	0.00996	0.00346	0.00205
LCVX_N_100_1_12	4214	10	9	-	17	10	10	0.00606	0.00464	0.01984	0.01033	0.00337	0.00194
LCVX_N_100_1_13	4214	10	9	-	18	11	11	0.00593	0.00445	0.01944	0.01071	0.00366	0.00215
LCVX_N_100_1_14	4214	11	9	-	17	11	11	0.00628	0.00446	0.02044	0.01027	0.00366	0.00211
LCVX_N_100_1_15	4214	10	9	-	17	10	10	0.00579	0.00468	0.01965	0.01036	0.00340	0.00201
LCVX_N_100_1_16	4214	10	8	-	17	10	10	0.00582	0.00421	0.02082	0.01032	0.00333	0.00197
LCVX_N_100_1_17	4214	12	9	-	17	10	10	0.00682	0.00464	0.01991	0.01035	0.00343	0.00197
LCVX_N_100_1_18	4214	10	8	-	18	10	10	0.00577	0.00422	0.01906	0.01075	0.00337	0.00192
LCVX_N_100_1_19	4214	11	9	-	17	10	10	0.00662	0.00464	0.02034	0.01031	0.00339	0.00194
LCVX_N_125_1_0	5264	11	9	-	18	11	11	0.00777	0.00565	0.02290	0.01312	0.00445	0.00273
LCVX_N_125_1_1	5264	11	9	-	17	10	10	0.00765	0.00577	0.02467	0.01285	0.00417	0.00236
LCVX_N_125_1_2	5264	10	9	-	18	10	10	0.00760	0.00562	0.02316	0.01321	0.00415	0.00242
LCVX_N_125_1_3	5264	11	9	-	17	10	10	0.00762	0.00579	0.02365	0.01275	0.00418	0.00240
LCVX_N_125_1_4	5264	9	8	-	15	10	10	0.00653	0.00504	0.02218	0.01198	0.00422	0.00239
LCVX_N_125_1_5	5264	11	9	-	18	10	10	0.00807	0.00563	0.02226	0.01319	0.00417	0.00239
LCVX_N_125_1_6	5264	11	9	-	16	10	10	0.00767	0.00584	0.02496	0.01285	0.00425	0.00245
LCVX_N_125_1_7	5264	11	8	-	18	10	10	0.00750	0.00525	0.02348	0.01317	0.00415	0.00244
LCVX_N_125_1_8	5264	9	8	-	16	10	10	0.00606	0.00519	0.02307	0.01232	0.00418	0.00238
LCVX_N_125_1_9	5264	10	9	-	15	10	10	0.00779	0.00588	0.02401	0.01235	0.00415	0.00241
LCVX_N_125_1_10	5264	9	8	-	17	10	10	0.00613	0.00523	0.02227	0.01273	0.00413	0.00240
LCVX_N_125_1_11	5264	8	8	-	13	10	10	0.00577	0.00520	0.02197	0.01101	0.00413	0.00246

**Table 3: Iterations and solver runtimes for lossless convexification problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
LCVX_N_125_l_12	5264	9	8	-	15	10	10	0.00727	0.00507	0.02199	0.01193	0.00418	0.00241
LCVX_N_125_l_13	5264	11	8	-	18	10	10	0.00708	0.00529	0.02405	0.01320	0.00408	0.00245
LCVX_N_125_l_14	5264	9	8	-	18	10	10	0.00612	0.00515	0.02389	0.01320	0.00415	0.00238
LCVX_N_125_l_15	5264	11	9	-	18	10	10	0.00801	0.00569	0.02199	0.01310	0.00414	0.00238
LCVX_N_125_l_16	5264	11	9	-	17	10	10	0.00773	0.00589	0.02420	0.01281	0.00416	0.00243
LCVX_N_125_l_17	5264	11	9	-	17	11	11	0.00797	0.00561	0.02357	0.01277	0.00454	0.00263
LCVX_N_125_l_18	5264	11	9	-	18	10	10	0.00804	0.00554	0.02353	0.01319	0.00417	0.00238
LCVX_N_125_l_19	5264	11	9	-	17	10	10	0.00829	0.00588	0.02513	0.01278	0.00413	0.00238
LCVX_N_150_l_0	6314	10	8	-	19	10	-	0.00803	0.00623	0.02961	0.01603	0.00504	-
LCVX_N_150_l_1	6314	11	9	-	18	11	-	0.00992	0.00671	0.02842	0.01561	0.00542	-
LCVX_N_150_l_2	6314	9	8	-	15	10	-	0.00822	0.00598	0.02509	0.01408	0.00498	-
LCVX_N_150_l_3	6314	10	9	-	16	10	-	0.00937	0.00697	0.02804	0.01450	0.00513	-
LCVX_N_150_l_4	6314	9	8	-	15	10	-	0.00835	0.00600	0.02466	0.01403	0.00505	-
LCVX_N_150_l_5	6314	11	10	-	17	11	-	0.00955	0.00737	0.02743	0.01504	0.00544	-
LCVX_N_150_l_6	6314	8	8	-	13	10	-	0.00682	0.00627	0.02362	0.01299	0.00504	-
LCVX_N_150_l_7	6314	9	8	-	17	10	-	0.00737	0.00600	0.02651	0.01505	0.00503	-
LCVX_N_150_l_8	6314	11	9	-	17	10	-	0.00936	0.00723	0.02915	0.01502	0.00526	-
LCVX_N_150_l_9	6314	12	11	-	20	11	-	0.01045	0.00825	0.02870	0.01658	0.00561	-
LCVX_N_150_l_10	6314	11	9	-	18	10	-	0.00950	0.00699	0.02729	0.01550	0.00506	-
LCVX_N_150_l_11	6314	10	8	-	16	10	-	0.00888	0.00601	0.02707	0.01451	0.00507	-
LCVX_N_150_l_12	6314	10	8	-	16	10	-	0.00899	0.00605	0.02689	0.01458	0.00507	-
LCVX_N_150_l_13	6314	12	9	-	18	11	-	0.01040	0.00675	0.02879	0.01552	0.00547	-
LCVX_N_150_l_14	6314	9	8	-	15	10	-	0.00795	0.00596	0.02549	0.01397	0.00506	-
LCVX_N_150_l_15	6314	10	8	-	17	10	-	0.00881	0.00600	0.02704	0.01508	0.00508	-
LCVX_N_150_l_16	6314	11	9	-	18	10	-	0.00982	0.00678	0.02729	0.01558	0.00507	-
LCVX_N_150_l_17	6314	9	8	-	18	10	-	0.00799	0.00606	0.02714	0.01560	0.00505	-
LCVX_N_150_l_18	6314	9	8	-	17	10	-	0.00740	0.00613	0.02876	0.01510	0.00520	-
LCVX_N_150_l_19	6314	11	9	-	17	10	-	0.00956	0.00722	0.02700	0.01508	0.00503	-
LCVX_N_200_l_0	8414	10	9	-	17	11	-	0.01181	0.00899	0.03711	0.02025	0.00735	-
LCVX_N_200_l_1	8414	10	9	-	17	11	-	0.01206	0.00902	0.03977	0.02033	0.00738	-
LCVX_N_200_l_2	8414	9	9	-	15	11	-	0.01093	0.00935	0.03452	0.01897	0.00721	-
LCVX_N_200_l_3	8414	12	9	-	18	11	-	0.01347	0.00911	0.03906	0.02090	0.00734	-
LCVX_N_200_l_4	8414	12	9	-	17	11	-	0.01426	0.00914	0.03792	0.02017	0.00733	-
LCVX_N_200_l_5	8414	11	9	-	17	10	-	0.01247	0.00955	0.03944	0.02029	0.00674	-
LCVX_N_200_l_6	8414	11	8	-	17	10	-	0.01161	0.00814	0.03592	0.02027	0.00672	-
LCVX_N_200_l_7	8414	11	9	-	18	11	-	0.01345	0.00907	0.03841	0.02110	0.00722	-
LCVX_N_200_l_8	8414	10	8	-	17	10	-	0.01160	0.00820	0.03577	0.02033	0.00677	-
LCVX_N_200_l_9	8414	9	8	-	17	10	-	0.01001	0.00822	0.03583	0.02019	0.00668	-
LCVX_N_200_l_10	8414	11	12	-	18	11	-	0.01301	0.01216	0.03920	0.02107	0.00734	-
LCVX_N_200_l_11	8414	9	8	-	16	10	-	0.01107	0.00824	0.03434	0.01956	0.00672	-
LCVX_N_200_l_12	8414	10	8	-	16	10	-	0.01063	0.00807	0.03451	0.01957	0.00675	-
LCVX_N_200_l_13	8414	10	8	-	17	10	-	0.01078	0.00811	0.03646	0.02030	0.00678	-
LCVX_N_200_l_14	8414	11	11	-	18	11	-	0.01295	0.01127	0.03857	0.02098	0.00729	-
LCVX_N_200_l_15	8414	11	9	-	18	10	-	0.01246	0.00952	0.03749	0.02083	0.00690	-
LCVX_N_200_l_16	8414	9	8	-	18	10	-	0.01008	0.00839	0.03665	0.02101	0.00669	-
LCVX_N_200_l_17	8414	12	9	-	18	11	-	0.01344	0.00907	0.03711	0.02103	0.00728	-

**Table 3: Iterations and solver runtimes for lossless convexification problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
LCVX_N_200_l_18	8414	9	8	-	14	10	-	0.01000	0.00806	0.03299	0.01820	0.00669	-
LCVX_N_200_l_19	8414	12	10	-	17	11	-	0.01413	0.01004	0.03575	0.02032	0.00730	-
LCVX_N_250_l_0	10514	10	9	-	11	11	-	0.01406	0.01104	0.05053	0.01684	0.00924	-
LCVX_N_250_l_1	10514	11	10	-	12	11	-	0.01524	0.01236	0.05496	0.01762	0.00913	-
LCVX_N_250_l_2	10514	11	9	-	12	11	-	0.01541	0.01128	0.05248	0.01766	0.00915	-
LCVX_N_250_l_3	10514	9	8	-	11	10	-	0.01237	0.01002	0.04312	0.01670	0.00848	-
LCVX_N_250_l_4	10514	10	8	-	11	10	-	0.01348	0.00992	0.04709	0.01676	0.00847	-
LCVX_N_250_l_5	10514	11	8	-	12	10	-	0.01450	0.01018	0.04506	0.01763	0.00902	-
LCVX_N_250_l_6	10514	9	8	-	11	10	-	0.01251	0.01000	0.04508	0.01683	0.00844	-
LCVX_N_250_l_7	10514	11	8	-	11	10	-	0.01454	0.01008	0.04595	0.01692	0.00840	-
LCVX_N_250_l_8	10514	12	12	-	12	11	-	0.01624	0.01454	0.05261	0.01774	0.00911	-
LCVX_N_250_l_9	10514	9	8	-	11	10	-	0.01245	0.01014	0.04380	0.01679	0.00844	-
LCVX_N_250_l_10	10514	12	9	-	12	10	-	0.01566	0.01161	0.04630	0.01759	0.00843	-
LCVX_N_250_l_11	10514	11	9	-	11	10	-	0.01453	0.01193	0.04860	0.01690	0.00836	-
LCVX_N_250_l_12	10514	12	9	-	12	11	-	0.01674	0.01127	0.05007	0.01760	0.00929	-
LCVX_N_250_l_13	10514	11	8	-	11	10	-	0.01452	0.01019	0.04785	0.01683	0.00843	-
LCVX_N_250_l_14	10514	10	9	-	11	11	-	0.01442	0.01137	0.05002	0.01687	0.00919	-
LCVX_N_250_l_15	10514	9	8	-	11	10	-	0.01243	0.01009	0.04669	0.01690	0.00848	-
LCVX_N_250_l_16	10514	10	8	-	10	10	-	0.01351	0.01007	0.04663	0.01604	0.00848	-
LCVX_N_250_l_17	10514	10	8	-	11	10	-	0.01347	0.01012	0.04441	0.01690	0.00842	-
LCVX_N_250_l_18	10514	13	9	-	12	11	-	0.01775	0.01133	0.05219	0.01766	0.00908	-
LCVX_N_250_l_19	10514	10	9	-	11	10	-	0.01341	0.01172	0.05027	0.01685	0.00841	-
LCVX_N_300_l_0	12614	11	11	-	12	11	-	0.01898	0.01659	0.06157	0.02122	0.01100	-
LCVX_N_300_l_1	12614	11	9	-	12	10	-	0.01757	0.01430	0.05777	0.02123	0.01015	-
LCVX_N_300_l_2	12614	10	11	-	12	11	-	0.01742	0.01652	0.05705	0.02147	0.01101	-
LCVX_N_300_l_3	12614	12	10	-	13	12	-	0.01939	0.01514	0.05686	0.02239	0.01189	-
LCVX_N_300_l_4	12614	10	8	-	12	10	-	0.01617	0.01223	0.05519	0.02120	0.01002	-
LCVX_N_300_l_5	12614	12	10	-	13	11	-	0.01980	0.01521	0.06007	0.02236	0.01102	-
LCVX_N_300_l_6	12614	10	8	-	11	10	-	0.01609	0.01220	0.05390	0.02013	0.01013	-
LCVX_N_300_l_7	12614	9	8	-	11	10	-	0.01491	0.01226	0.05744	0.02019	0.01013	-
LCVX_N_300_l_8	12614	10	9	-	11	11	-	0.01631	0.01384	0.05907	0.02027	0.01105	-
LCVX_N_300_l_9	12614	11	9	-	12	11	-	0.01744	0.01359	0.05865	0.02138	0.01101	-
LCVX_N_300_l_10	12614	13	10	-	13	11	-	0.02084	0.01507	0.05715	0.02225	0.01098	-
LCVX_N_300_l_11	12614	10	9	-	11	11	-	0.01641	0.01386	0.05330	0.02026	0.01103	-
LCVX_N_300_l_12	12614	11	9	-	12	11	-	0.01768	0.01376	0.05674	0.02117	0.01097	-
LCVX_N_300_l_13	12614	9	9	-	11	11	-	0.01500	0.01384	0.05338	0.02027	0.01108	-
LCVX_N_300_l_14	12614	10	9	-	12	10	-	0.01625	0.01423	0.05383	0.02125	0.01011	-
LCVX_N_300_l_15	12614	11	8	-	11	10	-	0.01745	0.01226	0.05440	0.02019	0.01017	-
LCVX_N_300_l_16	12614	9	8	-	11	10	-	0.01497	0.01204	0.05560	0.02032	0.01035	-
LCVX_N_300_l_17	12614	10	8	-	12	10	-	0.01610	0.01207	0.05475	0.02112	0.01015	-
LCVX_N_300_l_18	12614	9	8	-	11	10	-	0.01488	0.01205	0.05439	0.02027	0.01017	-
LCVX_N_300_l_19	12614	10	10	-	11	11	-	0.01721	0.01514	0.05864	0.02016	0.01100	-
LCVX_N_350_l_0	14714	9	9	-	12	11	-	0.01761	0.01590	0.06407	0.02445	0.01288	-
LCVX_N_350_l_1	14714	11	10	-	12	11	-	0.02119	0.01799	0.06574	0.02439	0.01290	-
LCVX_N_350_l_2	14714	11	9	-	12	11	-	0.02053	0.01606	0.06838	0.02432	0.01327	-
LCVX_N_350_l_3	14714	10	8	-	12	10	-	0.01901	0.01428	0.06508	0.02446	0.01181	-

**Table 3: Iterations and solver runtimes for lossless convexification problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
LCVX_N_350_l_4	14714	9	8	-	11	10	-	0.01757	0.01428	0.06324	0.02319	0.01181	-
LCVX_N_350_l_5	14714	11	9	-	12	11	-	0.02034	0.01593	0.06593	0.02441	0.01277	-
LCVX_N_350_l_6	14714	10	10	-	11	11	-	0.01905	0.01783	0.06727	0.02322	0.01285	-
LCVX_N_350_l_7	14714	11	9	-	12	11	-	0.02053	0.01591	0.06480	0.02435	0.01284	-
LCVX_N_350_l_8	14714	10	8	-	12	10	-	0.01895	0.01422	0.06198	0.02463	0.01183	-
LCVX_N_350_l_9	14714	11	9	-	12	11	-	0.02035	0.01588	0.06730	0.02448	0.01286	-
LCVX_N_350_l_10	14714	9	8	-	11	11	-	0.01758	0.01444	0.06150	0.02327	0.01288	-
LCVX_N_350_l_11	14714	12	10	-	13	12	-	0.02271	0.01792	0.06807	0.02549	0.01384	-
LCVX_N_350_l_12	14714	12	10	-	13	12	-	0.02284	0.01772	0.06639	0.02558	0.01382	-
LCVX_N_350_l_13	14714	11	10	-	13	11	-	0.02159	0.01788	0.06887	0.02561	0.01290	-
LCVX_N_350_l_14	14714	10	11	-	11	11	-	0.02021	0.01889	0.06600	0.02322	0.01288	-
LCVX_N_350_l_15	14714	11	9	-	12	11	-	0.02037	0.01601	0.06846	0.02443	0.01286	-
LCVX_N_350_l_16	14714	11	9	-	12	11	-	0.02056	0.01593	0.06633	0.02416	0.01288	-
LCVX_N_350_l_17	14714	8	8	-	11	10	-	0.01600	0.01423	0.06098	0.02317	0.01178	-
LCVX_N_350_l_18	14714	9	11	-	11	11	-	0.01766	0.01929	0.06235	0.02325	0.01283	-
LCVX_N_350_l_19	14714	10	8	-	12	10	-	0.01902	0.01435	0.06182	0.02450	0.01180	-

**Table 4: Iterations and solver runtimes for group lasso problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
GROUP_LASSO_N_1_I_0	761	4	11	-	4	6	6	0.00025	0.00111	0.00564	0.00047	0.00020	0.00007
GROUP_LASSO_N_1_I_1	761	4	11	-	5	6	6	0.00027	0.00109	0.00580	0.00049	0.00020	0.00007
GROUP_LASSO_N_1_I_2	761	4	10	-	5	6	6	0.00027	0.00098	0.00597	0.00049	0.00021	0.00007
GROUP_LASSO_N_1_I_3	761	4	12	-	5	6	6	0.00027	0.00122	0.00577	0.00050	0.00020	0.00007
GROUP_LASSO_N_1_I_4	761	4	11	-	5	6	6	0.00027	0.00110	0.00545	0.00050	0.00020	0.00007
GROUP_LASSO_N_1_I_5	761	4	11	-	5	6	6	0.00026	0.00113	0.00610	0.00048	0.00020	0.00007
GROUP_LASSO_N_1_I_6	761	4	11	-	5	6	6	0.00027	0.00109	0.00588	0.00049	0.00022	0.00006
GROUP_LASSO_N_1_I_7	761	5	10	-	5	6	6	0.00030	0.00099	0.00549	0.00049	0.00021	0.00007
GROUP_LASSO_N_1_I_8	761	4	12	-	5	6	6	0.00026	0.00115	0.00552	0.00051	0.00020	0.00006
GROUP_LASSO_N_1_I_9	761	5	10	-	5	6	6	0.00030	0.00098	0.00571	0.00048	0.00021	0.00007
GROUP_LASSO_N_1_I_10	761	4	9	-	5	6	6	0.00027	0.00091	0.00631	0.00050	0.00020	0.00007
GROUP_LASSO_N_1_I_11	761	4	11	-	4	6	6	0.00027	0.00112	0.00654	0.00047	0.00021	0.00007
GROUP_LASSO_N_1_I_12	761	5	9	-	5	6	6	0.00030	0.00090	0.00634	0.00050	0.00021	0.00006
GROUP_LASSO_N_1_I_13	761	4	11	-	5	6	6	0.00026	0.00108	0.00587	0.00049	0.00020	0.00006
GROUP_LASSO_N_1_I_14	761	4	12	-	5	6	6	0.00026	0.00116	0.00571	0.00049	0.00020	0.00007
GROUP_LASSO_N_1_I_15	761	4	12	-	5	6	6	0.00027	0.00109	0.00592	0.00049	0.00020	0.00006
GROUP_LASSO_N_1_I_16	761	4	11	-	5	6	6	0.00027	0.00107	0.00588	0.00049	0.00020	0.00007
GROUP_LASSO_N_1_I_17	761	4	12	-	5	6	6	0.00027	0.00121	0.00575	0.00050	0.00020	0.00007
GROUP_LASSO_N_1_I_18	761	5	11	-	5	6	6	0.00030	0.00111	0.00571	0.00050	0.00020	0.00006
GROUP_LASSO_N_1_I_19	761	5	10	-	4	6	6	0.00030	0.00098	0.00601	0.00047	0.00020	0.00006
GROUP_LASSO_N_2_I_0	2022	6	11	-	7	7	7	0.00082	0.00257	0.03673	0.00107	0.00057	0.00021
GROUP_LASSO_N_2_I_1	2022	7	11	-	7	7	7	0.00088	0.00254	0.03584	0.00108	0.00056	0.00020
GROUP_LASSO_N_2_I_2	2022	5	11	-	7	7	7	0.00074	0.00260	0.03842	0.00111	0.00057	0.00020
GROUP_LASSO_N_2_I_3	2022	7	12	-	7	8	8	0.00090	0.00278	0.04030	0.00108	0.00062	0.00023
GROUP_LASSO_N_2_I_4	2022	6	11	-	7	7	7	0.00081	0.00254	0.03843	0.00109	0.00059	0.00021
GROUP_LASSO_N_2_I_5	2022	7	11	-	8	8	8	0.00089	0.00249	0.03584	0.00112	0.00061	0.00023
GROUP_LASSO_N_2_I_6	2022	6	11	-	7	7	7	0.00082	0.00256	0.03675	0.00109	0.00057	0.00021
GROUP_LASSO_N_2_I_7	2022	6	10	-	7	7	7	0.00081	0.00239	0.03766	0.00107	0.00062	0.00020
GROUP_LASSO_N_2_I_8	2022	6	10	-	7	7	7	0.00082	0.00239	0.03624	0.00107	0.00058	0.00021
GROUP_LASSO_N_2_I_9	2022	6	11	-	7	7	7	0.00082	0.00241	0.05126	0.00109	0.00056	0.00020
GROUP_LASSO_N_2_I_10	2022	6	11	-	7	6	6	0.00084	0.00250	0.03732	0.00107	0.00059	0.00017
GROUP_LASSO_N_2_I_11	2022	7	11	-	7	7	7	0.00089	0.00242	0.03560	0.00106	0.00071	0.00020
GROUP_LASSO_N_2_I_12	2022	6	10	-	7	7	7	0.00081	0.00236	0.04354	0.00108	0.00057	0.00020
GROUP_LASSO_N_2_I_13	2022	7	13	-	7	7	7	0.00090	0.00305	0.03834	0.00109	0.00058	0.00021
GROUP_LASSO_N_2_I_14	2022	6	12	-	7	7	7	0.00082	0.00271	0.03596	0.00109	0.00058	0.00020
GROUP_LASSO_N_2_I_15	2022	6	11	-	7	7	7	0.00082	0.00249	0.03711	0.00108	0.00056	0.00020
GROUP_LASSO_N_2_I_16	2022	6	11	-	7	7	7	0.00082	0.00256	0.03639	0.00108	0.00060	0.00020
GROUP_LASSO_N_2_I_17	2022	6	11	-	7	7	7	0.00082	0.00251	0.03742	0.00109	0.00061	0.00021
GROUP_LASSO_N_2_I_18	2022	7	11	-	7	7	7	0.00090	0.00249	0.04392	0.00109	0.00055	0.00021
GROUP_LASSO_N_2_I_19	2022	6	10	-	7	7	7	0.00082	0.00236	0.03721	0.00109	0.00056	0.00020
GROUP_LASSO_N_3_I_0	3783	7	12	-	8	8	8	0.00177	0.00642	0.08580	0.00233	0.00135	0.00041
GROUP_LASSO_N_3_I_1	3783	7	12	-	8	8	8	0.00180	0.00665	0.09602	0.00229	0.00138	0.00042
GROUP_LASSO_N_3_I_2	3783	6	11	-	7	7	7	0.00164	0.00624	0.05644	0.00218	0.00133	0.00037
GROUP_LASSO_N_3_I_3	3783	6	11	-	6	7	7	0.00166	0.00655	0.09255	0.00223	0.00130	0.00037
GROUP_LASSO_N_3_I_4	3783	7	12	-	7	8	8	0.00183	0.00650	0.09112	0.00219	0.00136	0.00042
GROUP_LASSO_N_3_I_5	3783	6	12	-	7	7	7	0.00166	0.00618	0.08814	0.00218	0.00125	0.00037

**Table 4: Iterations and solver runtimes for group lasso problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
GROUP_LASSO_N_3_1_6	3783	6	12	-	7	7	7	0.00166	0.00679	0.09434	0.00219	0.00130	0.00037
GROUP_LASSO_N_3_1_7	3783	7	12	-	8	8	8	0.00179	0.00663	0.08940	0.00209	0.00136	0.00042
GROUP_LASSO_N_3_1_8	3783	8	12	-	8	8	8	0.00193	0.00672	0.09324	0.00204	0.00141	0.00042
GROUP_LASSO_N_3_1_9	3783	7	12	-	8	7	7	0.00179	0.00674	0.08115	0.00227	0.00129	0.00037
GROUP_LASSO_N_3_1_10	3783	7	12	-	7	8	8	0.00179	0.00644	0.08538	0.00194	0.00141	0.00042
GROUP_LASSO_N_3_1_11	3783	8	12	-	9	8	8	0.00207	0.00663	0.08846	0.00241	0.00138	0.00042
GROUP_LASSO_N_3_1_12	3783	8	12	-	9	8	8	0.00197	0.00664	0.09559	0.00238	0.00136	0.00042
GROUP_LASSO_N_3_1_13	3783	7	12	-	7	7	7	0.00183	0.00669	0.08609	0.00218	0.00130	0.00037
GROUP_LASSO_N_3_1_14	3783	8	12	-	8	8	8	0.00194	0.00646	0.05551	0.00227	0.00135	0.00042
GROUP_LASSO_N_3_1_15	3783	8	12	-	8	8	8	0.00195	0.00625	0.08523	0.00226	0.00143	0.00042
GROUP_LASSO_N_3_1_16	3783	8	12	-	7	8	8	0.00194	0.00618	0.05406	0.00218	0.00135	0.00042
GROUP_LASSO_N_3_1_17	3783	7	11	-	7	7	7	0.00179	0.00600	0.08719	0.00220	0.00127	0.00037
GROUP_LASSO_N_3_1_18	3783	7	12	-	7	7	7	0.00179	0.00651	0.08314	0.00234	0.00128	0.00037
GROUP_LASSO_N_3_1_19	3783	7	12	-	7	8	8	0.00182	0.00658	0.08637	0.00226	0.00133	0.00042
GROUP_LASSO_N_4_1_0	6044	7	11	-	8	8	8	0.00300	0.00908	0.05446	0.00337	0.00234	0.00070
GROUP_LASSO_N_4_1_1	6044	7	11	-	7	7	7	0.00297	0.00955	0.05064	0.00300	0.00221	0.00061
GROUP_LASSO_N_4_1_2	6044	7	11	-	8	8	8	0.00303	0.00940	0.04993	0.00319	0.00207	0.00069
GROUP_LASSO_N_4_1_3	6044	6	11	-	7	7	7	0.00282	0.00981	0.05742	0.00330	0.00219	0.00060
GROUP_LASSO_N_4_1_4	6044	6	10	-	7	7	7	0.00278	0.00832	0.06131	0.00313	0.00219	0.00063
GROUP_LASSO_N_4_1_5	6044	7	11	-	8	8	8	0.00303	0.00939	0.05877	0.00314	0.00232	0.00069
GROUP_LASSO_N_4_1_6	6044	8	11	-	8	9	9	0.00316	0.00966	0.05485	0.00325	0.00246	0.00078
GROUP_LASSO_N_4_1_7	6044	7	11	-	7	8	8	0.00307	0.00955	0.05600	0.00312	0.00230	0.00069
GROUP_LASSO_N_4_1_8	6044	7	12	-	8	7	7	0.00310	0.01008	0.05623	0.00344	0.00220	0.00061
GROUP_LASSO_N_4_1_9	6044	8	12	-	8	8	8	0.00319	0.01056	0.05563	0.00322	0.00231	0.00068
GROUP_LASSO_N_4_1_10	6044	7	12	-	7	7	7	0.00299	0.01032	0.06346	0.00307	0.00221	0.00061
GROUP_LASSO_N_4_1_11	6044	7	11	-	7	7	7	0.00299	0.00932	0.05491	0.00310	0.00218	0.00062
GROUP_LASSO_N_4_1_12	6044	8	12	-	10	8	8	0.00323	0.01024	0.05190	0.00378	0.00230	0.00068
GROUP_LASSO_N_4_1_13	6044	8	12	-	9	8	8	0.00319	0.01032	0.05939	0.00336	0.00233	0.00070
GROUP_LASSO_N_4_1_14	6044	7	11	-	8	8	8	0.00301	0.00938	0.05642	0.00331	0.00231	0.00068
GROUP_LASSO_N_4_1_15	6044	7	11	-	7	7	7	0.00298	0.00909	0.05730	0.00309	0.00218	0.00061
GROUP_LASSO_N_4_1_16	6044	7	12	-	6	8	8	0.00298	0.01051	0.05445	0.00311	0.00231	0.00068
GROUP_LASSO_N_4_1_17	6044	7	12	-	7	7	7	0.00300	0.01066	0.05494	0.00304	0.00219	0.00062
GROUP_LASSO_N_4_1_18	6044	7	12	-	8	8	8	0.00315	0.01057	0.04919	0.00324	0.00233	0.00068
GROUP_LASSO_N_4_1_19	6044	7	11	-	8	8	8	0.00299	0.00936	0.05176	0.00323	0.00230	0.00071
GROUP_LASSO_N_5_1_0	8805	7	12	-	8	8	8	0.00469	0.01341	0.08319	0.00421	0.00384	0.00104
GROUP_LASSO_N_5_1_1	8805	7	12	-	7	8	8	0.00470	0.01325	0.10216	0.00416	0.00379	0.00106
GROUP_LASSO_N_5_1_2	8805	7	12	-	8	8	8	0.00476	0.01535	0.08730	0.00363	0.00379	0.00104
GROUP_LASSO_N_5_1_3	8805	7	12	-	9	8	8	0.00485	0.01440	0.07491	0.00442	0.00378	0.00104
GROUP_LASSO_N_5_1_4	8805	7	12	-	8	8	8	0.00493	0.01482	0.08799	0.00419	0.00376	0.00104
GROUP_LASSO_N_5_1_5	8805	8	12	-	8	8	8	0.00504	0.01447	0.08363	0.00418	0.00378	0.00104
GROUP_LASSO_N_5_1_6	8805	8	13	-	10	9	9	0.00498	0.01613	0.10808	0.00407	0.00419	0.00116
GROUP_LASSO_N_5_1_7	8805	8	13	-	10	9	9	0.00500	0.01470	0.08271	0.00449	0.00397	0.00119
GROUP_LASSO_N_5_1_8	8805	7	12	-	8	8	8	0.00473	0.01473	0.09001	0.00415	0.00381	0.00105
GROUP_LASSO_N_5_1_9	8805	8	12	-	9	9	9	0.00499	0.01269	0.07729	0.00438	0.00396	0.00115
GROUP_LASSO_N_5_1_10	8805	8	12	-	9	8	8	0.00496	0.01467	0.08892	0.00432	0.00383	0.00103
GROUP_LASSO_N_5_1_11	8805	7	12	-	8	8	8	0.00476	0.01321	0.09225	0.00420	0.00383	0.00105

**Table 4: Iterations and solver runtimes for group lasso problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
GROUP_LASSO_N_5_1_12	8805	7	12	-	8	8	8	0.00463	0.01325	0.09461	0.00419	0.00390	0.00103
GROUP_LASSO_N_5_1_13	8805	7	12	-	8	8	8	0.00468	0.01545	0.08324	0.00420	0.00380	0.00102
GROUP_LASSO_N_5_1_14	8805	8	13	-	8	8	8	0.00505	0.01502	0.08284	0.00418	0.00384	0.00107
GROUP_LASSO_N_5_1_15	8805	6	11	-	8	7	7	0.00439	0.01318	0.11825	0.00425	0.00364	0.00092
GROUP_LASSO_N_5_1_16	8805	8	13	-	9	8	8	0.00503	0.01559	0.09084	0.00435	0.00376	0.00104
GROUP_LASSO_N_5_1_17	8805	7	12	-	7	8	8	0.00493	0.01263	0.08314	0.00426	0.00381	0.00105
GROUP_LASSO_N_5_1_18	8805	8	12	-	8	8	8	0.00500	0.01342	0.09374	0.00423	0.00381	0.00104
GROUP_LASSO_N_5_1_19	8805	7	12	-	8	8	8	0.00469	0.01493	0.08515	0.00421	0.00381	0.00106
GROUP_LASSO_N_8_1_0	20088	8	12	-	9	8	-	0.01510	0.03081	0.23046	0.00891	0.01268	-
GROUP_LASSO_N_8_1_1	20088	8	12	-	10	9	-	0.01562	0.03244	0.22238	0.00864	0.01326	-
GROUP_LASSO_N_8_1_2	20088	8	11	-	9	8	-	0.01589	0.03120	0.25512	0.00829	0.01277	-
GROUP_LASSO_N_8_1_3	20088	8	12	-	9	9	-	0.01565	0.03183	0.25449	0.00827	0.01317	-
GROUP_LASSO_N_8_1_4	20088	8	13	-	9	9	-	0.01555	0.03495	0.24709	0.00835	0.01317	-
GROUP_LASSO_N_8_1_5	20088	8	12	-	10	9	-	0.01568	0.03265	0.24470	0.00763	0.01316	-
GROUP_LASSO_N_8_1_6	20088	7	11	-	9	8	-	0.01558	0.02985	0.23378	0.00829	0.01273	-
GROUP_LASSO_N_8_1_7	20088	8	12	-	9	8	-	0.01546	0.03117	0.24632	0.00847	0.01270	-
GROUP_LASSO_N_8_1_8	20088	8	13	-	10	9	-	0.01594	0.03331	0.24449	0.00891	0.01317	-
GROUP_LASSO_N_8_1_9	20088	8	12	-	10	8	-	0.01574	0.03230	0.24570	0.00866	0.01274	-
GROUP_LASSO_N_8_1_10	20088	8	12	-	10	8	-	0.01627	0.03130	0.25514	0.00867	0.01271	-
GROUP_LASSO_N_8_1_11	20088	8	12	-	9	8	-	0.01566	0.03195	0.24611	0.00844	0.01276	-
GROUP_LASSO_N_8_1_12	20088	7	11	-	9	8	-	0.01504	0.03068	0.22589	0.00866	0.01274	-
GROUP_LASSO_N_8_1_13	20088	8	12	-	8	8	-	0.01627	0.03083	0.25352	0.00849	0.01269	-
GROUP_LASSO_N_8_1_14	20088	7	12	-	9	8	-	0.01506	0.03242	0.24988	0.00830	0.01274	-
GROUP_LASSO_N_8_1_15	20088	8	12	-	10	8	-	0.01586	0.03222	0.28708	0.00933	0.01270	-
GROUP_LASSO_N_8_1_16	20088	8	12	-	10	8	-	0.01544	0.03166	0.25367	0.00892	0.01281	-
GROUP_LASSO_N_8_1_17	20088	7	12	-	9	8	-	0.01509	0.03256	0.26745	0.00820	0.01271	-
GROUP_LASSO_N_8_1_18	20088	7	12	-	8	8	-	0.01508	0.03457	0.22807	0.00801	0.01269	-
GROUP_LASSO_N_8_1_19	20088	8	11	-	9	8	-	0.01641	0.03025	0.24183	0.00822	0.01267	-
GROUP_LASSO_N_10_1_0	30110	8	12	-	10	8	-	0.02756	0.04259	0.42095	0.01052	0.02289	-
GROUP_LASSO_N_10_1_1	30110	8	13	-	10	8	-	0.02695	0.04466	0.49732	0.01039	0.02288	-
GROUP_LASSO_N_10_1_2	30110	8	13	-	10	8	-	0.02863	0.04405	0.38538	0.01061	0.02286	-
GROUP_LASSO_N_10_1_3	30110	8	12	-	9	8	-	0.02738	0.04291	0.44772	0.01012	0.02288	-
GROUP_LASSO_N_10_1_4	30110	8	12	-	10	9	-	0.02784	0.04340	0.39140	0.01026	0.02376	-
GROUP_LASSO_N_10_1_5	30110	8	12	-	10	8	-	0.02721	0.04283	0.44021	0.01029	0.02308	-
GROUP_LASSO_N_10_1_6	30110	8	13	-	11	9	-	0.02714	0.04510	0.38498	0.01081	0.02469	-
GROUP_LASSO_N_10_1_7	30110	8	12	-	10	8	-	0.02728	0.03974	0.43310	0.01084	0.02287	-
GROUP_LASSO_N_10_1_8	30110	8	12	-	9	8	-	0.02705	0.04384	0.46175	0.00990	0.02397	-
GROUP_LASSO_N_10_1_9	30110	7	12	-	9	8	-	0.02639	0.04041	0.47321	0.00964	0.02276	-
GROUP_LASSO_N_10_1_10	30110	8	13	-	10	9	-	0.02693	0.04484	0.38864	0.01007	0.02359	-
GROUP_LASSO_N_10_1_11	30110	7	12	-	8	8	-	0.02591	0.04340	0.40515	0.00960	0.02277	-
GROUP_LASSO_N_10_1_12	30110	8	12	-	9	8	-	0.02906	0.03989	0.39042	0.00981	0.02378	-
GROUP_LASSO_N_10_1_13	30110	8	12	-	9	8	-	0.02754	0.04088	0.38960	0.00992	0.02323	-
GROUP_LASSO_N_10_1_14	30110	8	12	-	10	8	-	0.02794	0.04135	0.47359	0.01084	0.02279	-
GROUP_LASSO_N_10_1_15	30110	8	13	-	10	8	-	0.02995	0.04443	0.45111	0.01077	0.02382	-
GROUP_LASSO_N_10_1_16	30110	8	12	-	9	8	-	0.02747	0.04256	0.45569	0.01025	0.02287	-
GROUP_LASSO_N_10_1_17	30110	8	12	-	10	8	-	0.02945	0.04246	0.40807	0.01069	0.02267	-

**Table 4: Iterations and solver runtimes for group lasso problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
GROUP_LASSO_N_10_I_18	30110	8	12	-	9	8	-	0.02665	0.04161	0.40308	0.01116	0.02278	-
GROUP_LASSO_N_10_I_19	30110	7	11	-	8	8	-	0.02629	0.03774	0.43661	0.00941	0.02381	-
GROUP_LASSO_N_12_I_0	42132	8	12	-	9	8	-	0.04325	0.05675	0.72594	0.01384	0.03507	-
GROUP_LASSO_N_12_I_1	42132	8	12	-	8	8	-	0.04418	0.05847	0.72517	0.01283	0.03495	-
GROUP_LASSO_N_12_I_2	42132	7	12	-	8	7	-	0.04241	0.05995	0.72523	0.01339	0.03310	-
GROUP_LASSO_N_12_I_3	42132	8	13	-	8	8	-	0.04294	0.06362	0.71941	0.01290	0.03478	-
GROUP_LASSO_N_12_I_4	42132	8	12	-	8	9	-	0.04232	0.05920	0.80056	0.01300	0.03589	-
GROUP_LASSO_N_12_I_5	42132	8	12	-	9	8	-	0.04281	0.05680	0.73030	0.01386	0.03468	-
GROUP_LASSO_N_12_I_6	42132	8	12	-	8	8	-	0.04788	0.06034	0.65356	0.01307	0.03501	-
GROUP_LASSO_N_12_I_7	42132	8	12	-	8	8	-	0.04578	0.06083	0.67306	0.01335	0.03442	-
GROUP_LASSO_N_12_I_8	42132	8	12	-	8	8	-	0.04329	0.06219	0.66407	0.01339	0.03478	-
GROUP_LASSO_N_12_I_9	42132	8	12	-	8	8	-	0.04575	0.05707	0.72486	0.01280	0.03471	-
GROUP_LASSO_N_12_I_10	42132	8	12	-	8	8	-	0.04532	0.05731	0.64343	0.01296	0.03480	-
GROUP_LASSO_N_12_I_11	42132	8	12	-	8	8	-	0.04423	0.05731	0.58129	0.01348	0.03428	-
GROUP_LASSO_N_12_I_12	42132	8	12	-	8	8	-	0.04310	0.06025	0.68214	0.01284	0.03404	-
GROUP_LASSO_N_12_I_13	42132	8	12	-	8	8	-	0.04308	0.06026	0.80281	0.01309	0.03423	-
GROUP_LASSO_N_12_I_14	42132	8	12	-	8	8	-	0.04317	0.05748	0.69917	0.01355	0.03446	-
GROUP_LASSO_N_12_I_15	42132	8	12	-	9	8	-	0.04401	0.05937	0.65879	0.01412	0.03489	-
GROUP_LASSO_N_12_I_16	42132	8	12	-	8	8	-	0.04318	0.05796	0.73036	0.01336	0.03458	-
GROUP_LASSO_N_12_I_17	42132	8	12	-	8	8	-	0.04254	0.06040	0.66171	0.01297	0.03405	-
GROUP_LASSO_N_12_I_18	42132	8	12	-	9	8	-	0.04516	0.06173	0.86410	0.01419	0.03433	-
GROUP_LASSO_N_12_I_19	42132	8	12	-	8	8	-	0.04368	0.05717	0.72283	0.01353	0.03530	-
GROUP_LASSO_N_14_I_0	56154	7	11	-	8	8	-	0.06224	0.07874	1.23566	0.01759	0.04544	-
GROUP_LASSO_N_14_I_1	56154	8	12	-	8	8	-	0.06245	0.08160	0.96851	0.01818	0.04671	-
GROUP_LASSO_N_14_I_2	56154	8	12	-	9	8	-	0.06498	0.08289	0.97849	0.01848	0.04671	-
GROUP_LASSO_N_14_I_3	56154	8	12	-	8	8	-	0.06662	0.08547	1.17808	0.01798	0.04580	-
GROUP_LASSO_N_14_I_4	56154	8	12	-	8	8	-	0.06497	0.08176	1.22698	0.01762	0.04588	-
GROUP_LASSO_N_14_I_5	56154	7	11	-	7	8	-	0.06280	0.07908	1.09989	0.01686	0.04782	-
GROUP_LASSO_N_14_I_6	56154	8	13	-	8	9	-	0.06354	0.08527	1.09452	0.01801	0.04769	-
GROUP_LASSO_N_14_I_7	56154	8	13	-	8	9	-	0.06345	0.08543	1.11042	0.01851	0.04802	-
GROUP_LASSO_N_14_I_8	56154	8	12	-	8	8	-	0.06339	0.08132	0.99903	0.01885	0.04614	-
GROUP_LASSO_N_14_I_9	56154	7	11	-	8	8	-	0.06288	0.07812	1.26237	0.01756	0.04631	-
GROUP_LASSO_N_14_I_10	56154	7	11	-	8	8	-	0.06023	0.07890	1.10284	0.01826	0.04638	-
GROUP_LASSO_N_14_I_11	56154	8	12	-	8	8	-	0.06586	0.08050	0.99469	0.01772	0.04630	-
GROUP_LASSO_N_14_I_12	56154	7	12	-	8	8	-	0.06166	0.08194	1.15584	0.01847	0.04567	-
GROUP_LASSO_N_14_I_13	56154	8	12	-	8	8	-	0.06373	0.08154	1.07342	0.02029	0.04599	-
GROUP_LASSO_N_14_I_14	56154	7	12	-	7	8	-	0.06258	0.08137	1.00195	0.01702	0.04789	-
GROUP_LASSO_N_14_I_15	56154	7	12	-	7	8	-	0.06407	0.08213	1.11372	0.01982	0.04643	-
GROUP_LASSO_N_14_I_16	56154	8	12	-	8	8	-	0.06230	0.08161	1.06719	0.01917	0.04592	-
GROUP_LASSO_N_14_I_17	56154	8	13	-	9	8	-	0.06332	0.08538	1.08016	0.01906	0.04597	-
GROUP_LASSO_N_14_I_18	56154	7	12	-	8	8	-	0.06121	0.08230	0.97765	0.01952	0.04903	-
GROUP_LASSO_N_14_I_19	56154	8	12	-	8	8	-	0.06337	0.08094	1.21798	0.01775	0.04637	-
GROUP_LASSO_N_16_I_0	72176	7	12	-	8	8	-	0.08818	0.10965	1.53172	0.02212	0.06506	-
GROUP_LASSO_N_16_I_1	72176	8	12	-	10	8	-	0.09073	0.11368	1.73110	0.02489	0.06656	-
GROUP_LASSO_N_16_I_2	72176	7	13	-	8	8	-	0.08744	0.11966	1.39775	0.02242	0.06552	-
GROUP_LASSO_N_16_I_3	72176	7	12	-	8	8	-	0.08782	0.11273	1.61282	0.02303	0.06462	-

**Table 4: Iterations and solver runtimes for group lasso problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
GROUP_LASSO_N_16_I_4	72176	7	12	-	8	8	-	0.09297	0.11399	1.57281	0.02254	0.06636	-
GROUP_LASSO_N_16_I_5	72176	7	12	-	8	8	-	0.08922	0.11064	1.39139	0.02248	0.06505	-
GROUP_LASSO_N_16_I_6	72176	8	12	-	8	8	-	0.09090	0.11306	1.59569	0.02503	0.06585	-
GROUP_LASSO_N_16_I_7	72176	7	12	-	8	8	-	0.08715	0.11313	1.59741	0.02194	0.06630	-
GROUP_LASSO_N_16_I_8	72176	7	12	-	8	8	-	0.08660	0.10653	1.56979	0.02195	0.06473	-
GROUP_LASSO_N_16_I_9	72176	8	12	-	8	8	-	0.09179	0.11243	1.58186	0.02176	0.06512	-
GROUP_LASSO_N_16_I_10	72176	8	12	-	8	8	-	0.09056	0.11126	1.44625	0.02328	0.06416	-
GROUP_LASSO_N_16_I_11	72176	8	12	-	9	8	-	0.09036	0.11116	1.60883	0.02410	0.06535	-
GROUP_LASSO_N_16_I_12	72176	8	12	-	8	8	-	0.09242	0.11283	1.61328	0.02212	0.06555	-
GROUP_LASSO_N_16_I_13	72176	8	12	-	9	8	-	0.09120	0.11232	1.57203	0.02442	0.06505	-
GROUP_LASSO_N_16_I_14	72176	8	12	-	8	8	-	0.09242	0.11370	1.44119	0.02223	0.06555	-
GROUP_LASSO_N_16_I_15	72176	8	12	-	8	8	-	0.09026	0.11228	1.41785	0.02290	0.06527	-
GROUP_LASSO_N_16_I_16	72176	8	12	-	8	8	-	0.09227	0.11340	1.45983	0.02243	0.06665	-
GROUP_LASSO_N_16_I_17	72176	7	12	-	8	8	-	0.08756	0.11138	1.42790	0.02237	0.06617	-
GROUP_LASSO_N_16_I_18	72176	8	12	-	8	8	-	0.09079	0.11191	1.61307	0.02328	0.06484	-
GROUP_LASSO_N_16_I_19	72176	7	12	-	8	8	-	0.09151	0.11219	1.59887	0.02206	0.06481	-

**Table 5: Iterations and solver runtimes for portfolio optimization problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
PORTFOLIO_N_2_1_0	804	9	12	14	11	9	9	0.00041	0.00087	0.00205	0.00100	0.00025	0.00012
PORTFOLIO_N_2_1_1	804	9	14	13	11	8	8	0.00041	0.00096	0.00200	0.00101	0.00024	0.00010
PORTFOLIO_N_2_1_2	804	8	12	12	11	8	8	0.00038	0.00089	0.00204	0.00100	0.00024	0.00010
PORTFOLIO_N_2_1_3	804	8	15	16	11	8	8	0.00038	0.00110	0.00203	0.00101	0.00024	0.00011
PORTFOLIO_N_2_1_4	804	9	14	15	9	8	8	0.00041	0.00099	0.00207	0.00092	0.00024	0.00011
PORTFOLIO_N_2_1_5	804	10	14	14	13	9	9	0.00046	0.00097	0.00203	0.00110	0.00026	0.00011
PORTFOLIO_N_2_1_6	804	10	18	11	10	10	10	0.00045	0.00133	0.00198	0.00096	0.00028	0.00013
PORTFOLIO_N_2_1_7	804	8	10	11	9	7	7	0.00037	0.00078	0.00202	0.00091	0.00021	0.00009
PORTFOLIO_N_2_1_8	804	9	17	13	9	9	9	0.00041	0.00134	0.00203	0.00091	0.00025	0.00011
PORTFOLIO_N_2_1_9	804	16	13	15	12	9	9	0.00064	0.00090	0.00203	0.00104	0.00026	0.00011
PORTFOLIO_N_2_1_10	804	10	13	17	13	10	10	0.00045	0.00089	0.00203	0.00110	0.00027	0.00013
PORTFOLIO_N_2_1_11	804	10	12	14	13	12	12	0.00045	0.00087	0.00205	0.00109	0.00031	0.00015
PORTFOLIO_N_2_1_12	804	10	11	15	11	10	10	0.00044	0.00084	0.00206	0.00101	0.00027	0.00013
PORTFOLIO_N_2_1_13	804	9	12	13	11	8	8	0.00041	0.00088	0.00202	0.00100	0.00023	0.00010
PORTFOLIO_N_2_1_14	804	9	14	14	11	9	9	0.00040	0.00101	0.00201	0.00103	0.00029	0.00011
PORTFOLIO_N_2_1_15	804	9	11	14	11	8	8	0.00041	0.00081	0.00204	0.00103	0.00023	0.00010
PORTFOLIO_N_2_1_16	804	10	14	12	10	8	8	0.00044	0.00101	0.00204	0.00095	0.00024	0.00011
PORTFOLIO_N_2_1_17	804	8	11	11	8	8	8	0.00038	0.00084	0.00198	0.00085	0.00024	0.00010
PORTFOLIO_N_2_1_18	804	9	13	12	11	9	9	0.00041	0.00097	0.00202	0.00102	0.00025	0.00011
PORTFOLIO_N_2_1_19	804	10	11	12	9	8	8	0.00045	0.00083	0.00203	0.00092	0.00023	0.00010
PORTFOLIO_N_4_1_0	2008	10	13	14	11	9	9	0.00110	0.00194	0.00230	0.00184	0.00056	0.00028
PORTFOLIO_N_4_1_1	2008	10	14	14	12	11	11	0.00109	0.00212	0.00231	0.00191	0.00062	0.00034
PORTFOLIO_N_4_1_2	2008	11	15	14	12	10	10	0.00116	0.00268	0.00232	0.00189	0.00059	0.00031
PORTFOLIO_N_4_1_3	2008	11	16	15	13	9	9	0.00116	0.00227	0.00230	0.00198	0.00055	0.00027
PORTFOLIO_N_4_1_4	2008	9	17	13	11	8	8	0.00100	0.00249	0.00229	0.00182	0.00049	0.00025
PORTFOLIO_N_4_1_5	2008	11	13	14	12	9	9	0.00117	0.00202	0.00230	0.00193	0.00057	0.00027
PORTFOLIO_N_4_1_6	2008	10	14	13	11	9	9	0.00106	0.00210	0.00226	0.00186	0.00055	0.00028
PORTFOLIO_N_4_1_7	2008	11	15	13	11	9	9	0.00112	0.00221	0.00228	0.00185	0.00057	0.00028
PORTFOLIO_N_4_1_8	2008	11	13	12	10	9	9	0.00113	0.00192	0.00227	0.00176	0.00055	0.00028
PORTFOLIO_N_4_1_9	2008	11	16	14	13	10	10	0.00116	0.00245	0.00230	0.00200	0.00065	0.00030
PORTFOLIO_N_4_1_10	2008	11	18	16	14	10	10	0.00118	0.00256	0.00231	0.00209	0.00058	0.00031
PORTFOLIO_N_4_1_11	2008	9	15	13	11	9	9	0.00099	0.00230	0.00228	0.00183	0.00059	0.00028
PORTFOLIO_N_4_1_12	2008	12	14	12	10	9	9	0.00122	0.00209	0.00227	0.00176	0.00056	0.00027
PORTFOLIO_N_4_1_13	2008	10	15	16	12	9	9	0.00106	0.00239	0.00231	0.00192	0.00054	0.00027
PORTFOLIO_N_4_1_14	2008	10	17	15	14	9	9	0.00108	0.00240	0.00231	0.00208	0.00055	0.00028
PORTFOLIO_N_4_1_15	2008	11	12	13	11	10	10	0.00114	0.00190	0.00226	0.00183	0.00059	0.00031
PORTFOLIO_N_4_1_16	2008	11	17	14	12	10	10	0.00116	0.00277	0.00231	0.00192	0.00059	0.00031
PORTFOLIO_N_4_1_17	2008	9	13	12	10	9	9	0.00101	0.00194	0.00225	0.00174	0.00054	0.00028
PORTFOLIO_N_4_1_18	2008	11	14	14	13	9	9	0.00117	0.00204	0.00231	0.00197	0.00055	0.00027
PORTFOLIO_N_4_1_19	2008	9	19	14	11	8	8	0.00099	0.00298	0.00231	0.00181	0.00052	0.00025
PORTFOLIO_N_6_1_0	3612	13	16	14	11	11	11	0.00223	0.00405	0.00261	0.00277	0.00112	0.00058
PORTFOLIO_N_6_1_1	3612	14	16	14	10	8	8	0.00233	0.00418	0.00267	0.00270	0.00093	0.00043
PORTFOLIO_N_6_1_2	3612	12	14	15	12	10	10	0.00211	0.00345	0.00269	0.00294	0.00111	0.00053
PORTFOLIO_N_6_1_3	3612	15	18	15	13	10	10	0.00247	0.00431	0.00268	0.00302	0.00105	0.00052
PORTFOLIO_N_6_1_4	3612	12	14	14	13	11	11	0.00208	0.00370	0.00264	0.00304	0.00112	0.00058
PORTFOLIO_N_6_1_5	3612	11	13	12	10	9	9	0.00198	0.00332	0.00257	0.00266	0.00098	0.00048

**Table 5: Iterations and solver runtimes for portfolio optimization problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
PORTFOLIO_N_6_1_6	3612	10	15	13	11	9	9	0.00186	0.00390	0.00261	0.00278	0.00100	0.00048
PORTFOLIO_N_6_1_7	3612	12	14	14	12	9	9	0.00212	0.00340	0.00262	0.00292	0.00100	0.00047
PORTFOLIO_N_6_1_8	3612	12	16	15	13	9	9	0.00210	0.00398	0.00269	0.00303	0.00098	0.00048
PORTFOLIO_N_6_1_9	3612	13	15	15	14	10	10	0.00222	0.00392	0.00271	0.00317	0.00109	0.00053
PORTFOLIO_N_6_1_10	3612	11	15	14	12	10	10	0.00195	0.00369	0.00266	0.00287	0.00105	0.00052
PORTFOLIO_N_6_1_11	3612	11	14	13	11	9	9	0.00196	0.00359	0.00261	0.00279	0.00100	0.00047
PORTFOLIO_N_6_1_12	3612	12	15	16	11	11	11	0.00211	0.00410	0.00265	0.00277	0.00115	0.00058
PORTFOLIO_N_6_1_13	3612	11	15	14	12	10	10	0.00197	0.00369	0.00266	0.00288	0.00105	0.00053
PORTFOLIO_N_6_1_14	3612	13	16	14	13	10	10	0.00226	0.00394	0.00262	0.00301	0.00107	0.00054
PORTFOLIO_N_6_1_15	3612	12	17	14	12	10	10	0.00216	0.00417	0.00265	0.00289	0.00105	0.00053
PORTFOLIO_N_6_1_16	3612	11	17	14	12	9	9	0.00200	0.00421	0.00265	0.00289	0.00097	0.00048
PORTFOLIO_N_6_1_17	3612	13	15	16	15	10	10	0.00222	0.00358	0.00274	0.00323	0.00109	0.00052
PORTFOLIO_N_6_1_18	3612	12	20	15	14	10	10	0.00210	0.00528	0.00264	0.00313	0.00106	0.00053
PORTFOLIO_N_6_1_19	3612	13	13	14	11	9	9	0.00217	0.00344	0.00264	0.00278	0.00100	0.00047
PORTFOLIO_N_8_1_0	5616	11	15	14	11	12	12	0.00332	0.00572	0.00302	0.00373	0.00166	0.00098
PORTFOLIO_N_8_1_1	5616	12	15	14	13	11	11	0.00353	0.00585	0.00306	0.00405	0.00166	0.00089
PORTFOLIO_N_8_1_2	5616	12	16	14	12	12	12	0.00346	0.00602	0.00308	0.00391	0.00163	0.00098
PORTFOLIO_N_8_1_3	5616	13	15	16	14	11	11	0.00371	0.00564	0.00317	0.00423	0.00146	0.00095
PORTFOLIO_N_8_1_4	5616	13	16	15	12	11	11	0.00368	0.00639	0.00312	0.00389	0.00156	0.00093
PORTFOLIO_N_8_1_5	5616	12	15	13	13	10	10	0.00345	0.00587	0.00306	0.00405	0.00141	0.00084
PORTFOLIO_N_8_1_6	5616	14	17	13	12	10	10	0.00386	0.00634	0.00303	0.00390	0.00136	0.00086
PORTFOLIO_N_8_1_7	5616	12	14	16	15	9	9	0.00340	0.00550	0.00324	0.00435	0.00131	0.00077
PORTFOLIO_N_8_1_8	5616	13	17	15	14	11	11	0.00370	0.00649	0.00310	0.00423	0.00156	0.00089
PORTFOLIO_N_8_1_9	5616	14	18	14	13	10	10	0.00393	0.00621	0.00308	0.00405	0.00147	0.00085
PORTFOLIO_N_8_1_10	5616	13	18	16	15	10	10	0.00374	0.00629	0.00325	0.00439	0.00136	0.00087
PORTFOLIO_N_8_1_11	5616	12	18	15	15	10	10	0.00345	0.00668	0.00315	0.00436	0.00142	0.00085
PORTFOLIO_N_8_1_12	5616	11	15	15	12	9	9	0.00325	0.00565	0.00313	0.00390	0.00136	0.00073
PORTFOLIO_N_8_1_13	5616	14	17	15	13	10	10	0.00389	0.00647	0.00316	0.00406	0.00147	0.00083
PORTFOLIO_N_8_1_14	5616	12	17	15	14	10	10	0.00349	0.00627	0.00313	0.00423	0.00138	0.00085
PORTFOLIO_N_8_1_15	5616	14	18	15	15	10	10	0.00389	0.00672	0.00305	0.00437	0.00147	0.00084
PORTFOLIO_N_8_1_16	5616	14	17	16	14	12	12	0.00387	0.00617	0.00315	0.00422	0.00168	0.00099
PORTFOLIO_N_8_1_17	5616	12	15	15	13	9	9	0.00347	0.00575	0.00308	0.00404	0.00134	0.00075
PORTFOLIO_N_8_1_18	5616	12	14	14	13	10	10	0.00352	0.00551	0.00308	0.00404	0.00142	0.00083
PORTFOLIO_N_8_1_19	5616	12	16	16	14	10	10	0.00350	0.00603	0.00318	0.00420	0.00141	0.00083
PORTFOLIO_N_10_1_0	8020	13	18	14	13	9	9	0.00581	0.01020	0.00366	0.00525	0.00185	0.00149
PORTFOLIO_N_10_1_1	8020	12	19	13	13	10	10	0.00556	0.01172	0.00357	0.00521	0.00200	0.00166
PORTFOLIO_N_10_1_2	8020	14	18	14	12	10	10	0.00613	0.00946	0.00364	0.00502	0.00198	0.00169
PORTFOLIO_N_10_1_3	8020	14	16	15	14	10	10	0.00606	0.00823	0.00372	0.00544	0.00199	0.00165
PORTFOLIO_N_10_1_4	8020	12	16	15	13	9	9	0.00546	0.00860	0.00371	0.00523	0.00186	0.00149
PORTFOLIO_N_10_1_5	8020	11	15	14	12	11	11	0.00518	0.00977	0.00360	0.00504	0.00218	0.00180
PORTFOLIO_N_10_1_6	8020	14	17	16	14	11	11	0.00605	0.00913	0.00381	0.00543	0.00216	0.00181
PORTFOLIO_N_10_1_7	8020	14	15	15	13	12	12	0.00602	0.00865	0.00370	0.00524	0.00235	0.00194
PORTFOLIO_N_10_1_8	8020	13	16	17	14	11	11	0.00580	0.00873	0.00385	0.00544	0.00217	0.00179
PORTFOLIO_N_10_1_9	8020	12	15	15	12	11	11	0.00557	0.00826	0.00372	0.00501	0.00217	0.00177
PORTFOLIO_N_10_1_10	8020	13	19	14	14	11	11	0.00582	0.00975	0.00364	0.00540	0.00215	0.00182
PORTFOLIO_N_10_1_11	8020	14	18	16	16	10	10	0.00606	0.00962	0.00376	0.00583	0.00198	0.00167

**Table 5: Iterations and solver runtimes for portfolio optimization problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
PORTFOLIO_N_10_L_12	8020	13	18	14	13	12	12	0.00575	0.00940	0.00364	0.00523	0.00235	0.00200
PORTFOLIO_N_10_L_13	8020	13	13	13	13	9	9	0.00570	0.00734	0.00357	0.00523	0.00187	0.00150
PORTFOLIO_N_10_L_14	8020	13	18	14	13	10	10	0.00587	0.00992	0.00366	0.00527	0.00199	0.00163
PORTFOLIO_N_10_L_15	8020	12	16	15	12	11	11	0.00554	0.00864	0.00368	0.00502	0.00218	0.00178
PORTFOLIO_N_10_L_16	8020	13	18	16	13	12	12	0.00580	0.00995	0.00379	0.00522	0.00236	0.00192
PORTFOLIO_N_10_L_17	8020	13	17	15	14	10	10	0.00585	0.00932	0.00372	0.00542	0.00200	0.00168
PORTFOLIO_N_10_L_18	8020	13	18	15	13	10	10	0.00576	0.00967	0.00370	0.00525	0.00198	0.00164
PORTFOLIO_N_10_L_19	8020	13	15	15	15	12	12	0.00570	0.00853	0.00370	0.00564	0.00236	0.00194
PORTFOLIO_N_15_L_0	15780	13	18	14	13	11	-	0.01637	0.01628	0.00539	0.00874	0.00454	-
PORTFOLIO_N_15_L_1	15780	13	18	16	14	11	-	0.01608	0.01650	0.00567	0.00908	0.00456	-
PORTFOLIO_N_15_L_2	15780	13	16	16	16	11	-	0.01628	0.01516	0.00572	0.00959	0.00456	-
PORTFOLIO_N_15_L_3	15780	12	18	16	15	12	-	0.01570	0.02050	0.00563	0.00941	0.00490	-
PORTFOLIO_N_15_L_4	15780	13	18	16	13	10	-	0.01600	0.01653	0.00561	0.00872	0.00422	-
PORTFOLIO_N_15_L_5	15780	13	15	15	13	10	-	0.01613	0.01388	0.00554	0.00884	0.00418	-
PORTFOLIO_N_15_L_6	15780	14	15	17	14	11	-	0.01664	0.01377	0.00577	0.00905	0.00460	-
PORTFOLIO_N_15_L_7	15780	13	16	16	15	10	-	0.01615	0.01428	0.00562	0.00936	0.00424	-
PORTFOLIO_N_15_L_8	15780	14	19	18	16	11	-	0.01667	0.01589	0.00614	0.00980	0.00455	-
PORTFOLIO_N_15_L_9	15780	13	14	15	14	10	-	0.01606	0.01471	0.00551	0.00904	0.00421	-
PORTFOLIO_N_15_L_10	15780	14	17	15	14	10	-	0.01686	0.01465	0.00549	0.00904	0.00422	-
PORTFOLIO_N_15_L_11	15780	13	17	15	13	11	-	0.01626	0.01497	0.00552	0.00880	0.00455	-
PORTFOLIO_N_15_L_12	15780	14	21	16	17	11	-	0.01670	0.01898	0.00574	0.00994	0.00458	-
PORTFOLIO_N_15_L_13	15780	13	16	15	14	12	-	0.01601	0.01464	0.00550	0.00911	0.00501	-
PORTFOLIO_N_15_L_14	15780	12	17	16	14	12	-	0.01563	0.01455	0.00574	0.00902	0.00497	-
PORTFOLIO_N_15_L_15	15780	15	17	16	15	13	-	0.01715	0.01484	0.00562	0.00931	0.00534	-
PORTFOLIO_N_15_L_16	15780	12	16	16	14	11	-	0.01557	0.01435	0.00568	0.00906	0.00459	-
PORTFOLIO_N_15_L_17	15780	13	15	16	13	11	-	0.01612	0.01491	0.00569	0.00872	0.00458	-
PORTFOLIO_N_15_L_18	15780	13	17	15	13	10	-	0.01614	0.01608	0.00555	0.00880	0.00418	-
PORTFOLIO_N_15_L_19	15780	12	17	15	15	10	-	0.01582	0.01613	0.00559	0.00937	0.00421	-
PORTFOLIO_N_20_L_0	26040	13	15	15	14	11	-	0.01608	0.02261	0.00864	0.01355	0.00848	-
PORTFOLIO_N_20_L_1	26040	13	19	15	15	10	-	0.01652	0.02820	0.00860	0.01399	0.00780	-
PORTFOLIO_N_20_L_2	26040	13	17	18	14	13	-	0.01644	0.02987	0.00915	0.01361	0.00985	-
PORTFOLIO_N_20_L_3	26040	13	16	15	14	11	-	0.01514	0.02325	0.00849	0.01360	0.00855	-
PORTFOLIO_N_20_L_4	26040	15	19	14	15	13	-	0.01828	0.03341	0.00854	0.01400	0.00992	-
PORTFOLIO_N_20_L_5	26040	13	16	16	17	10	-	0.01646	0.02098	0.00871	0.01484	0.00782	-
PORTFOLIO_N_20_L_6	26040	14	20	15	15	11	-	0.01747	0.02746	0.00835	0.01398	0.00852	-
PORTFOLIO_N_20_L_7	26040	15	17	16	15	11	-	0.01858	0.02532	0.00888	0.01407	0.00862	-
PORTFOLIO_N_20_L_8	26040	13	18	14	13	11	-	0.01628	0.02561	0.00837	0.01308	0.00845	-
PORTFOLIO_N_20_L_9	26040	14	17	15	15	11	-	0.01726	0.02707	0.00880	0.01410	0.00843	-
PORTFOLIO_N_20_L_10	26040	14	18	16	14	11	-	0.01711	0.02505	0.00872	0.01356	0.00849	-
PORTFOLIO_N_20_L_11	26040	13	16	17	14	13	-	0.01621	0.02322	0.00890	0.01350	0.00990	-
PORTFOLIO_N_20_L_12	26040	15	17	17	16	11	-	0.01829	0.02644	0.00895	0.01436	0.00851	-
PORTFOLIO_N_20_L_13	26040	13	15	15	13	10	-	0.01599	0.02926	0.00906	0.01311	0.00777	-
PORTFOLIO_N_20_L_14	26040	13	18	18	15	10	-	0.01616	0.02533	0.00919	0.01403	0.00775	-
PORTFOLIO_N_20_L_15	26040	13	16	16	14	11	-	0.01636	0.02529	0.00865	0.01381	0.00854	-
PORTFOLIO_N_20_L_16	26040	13	17	16	14	12	-	0.01634	0.02396	0.00867	0.01358	0.00933	-
PORTFOLIO_N_20_L_17	26040	13	18	16	16	12	-	0.01627	0.02555	0.00881	0.01447	0.00929	-

**Table 5: Iterations and solver runtimes for portfolio optimization problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
PORTFOLIO_N_20_L18	26040	13	18	16	14	10	-	0.01645	0.02772	0.00881	0.01352	0.00777	-
PORTFOLIO_N_20_L19	26040	14	16	16	15	11	-	0.01738	0.02570	0.00892	0.01392	0.00846	-
PORTFOLIO_N_25_L0	38800	15	18	17	17	12	-	0.02906	0.03832	0.01357	0.02078	0.01460	-
PORTFOLIO_N_25_L1	38800	13	19	17	15	12	-	0.02572	0.04047	0.01364	0.01964	0.01463	-
PORTFOLIO_N_25_L2	38800	14	15	15	15	11	-	0.02703	0.03219	0.01285	0.01948	0.01357	-
PORTFOLIO_N_25_L3	38800	15	20	15	16	12	-	0.02894	0.03989	0.01286	0.02016	0.01476	-
PORTFOLIO_N_25_L4	38800	13	16	15	15	11	-	0.02520	0.03309	0.01289	0.01982	0.01353	-
PORTFOLIO_N_25_L5	38800	14	15	15	15	12	-	0.02751	0.03296	0.01305	0.01962	0.01489	-
PORTFOLIO_N_25_L6	38800	14	18	14	14	11	-	0.02750	0.03763	0.01266	0.01904	0.01364	-
PORTFOLIO_N_25_L7	38800	14	19	15	16	13	-	0.02732	0.03533	0.01281	0.02014	0.01590	-
PORTFOLIO_N_25_L8	38800	14	17	15	15	11	-	0.02728	0.03293	0.01274	0.01960	0.01364	-
PORTFOLIO_N_25_L9	38800	14	19	15	15	12	-	0.02739	0.03600	0.01290	0.01954	0.01477	-
PORTFOLIO_N_25_L10	38800	14	18	14	14	11	-	0.02736	0.03734	0.01236	0.01896	0.01347	-
PORTFOLIO_N_25_L11	38800	15	21	15	16	11	-	0.02910	0.04072	0.01281	0.02027	0.01354	-
PORTFOLIO_N_25_L12	38800	15	18	17	14	12	-	0.02923	0.03431	0.01344	0.01905	0.01463	-
PORTFOLIO_N_25_L13	38800	16	19	15	16	11	-	0.03070	0.03879	0.01290	0.02019	0.01364	-
PORTFOLIO_N_25_L14	38800	14	16	13	13	10	-	0.02719	0.03314	0.01209	0.01832	0.01244	-
PORTFOLIO_N_25_L15	38800	13	16	17	15	11	-	0.02563	0.03328	0.01350	0.01982	0.01352	-
PORTFOLIO_N_25_L16	38800	14	19	14	15	12	-	0.02742	0.05195	0.01246	0.01948	0.01469	-
PORTFOLIO_N_25_L17	38800	14	15	14	13	12	-	0.02716	0.03272	0.01235	0.01838	0.01469	-
PORTFOLIO_N_25_L18	38800	15	18	16	16	12	-	0.02915	0.03600	0.01317	0.02032	0.01499	-
PORTFOLIO_N_25_L19	38800	14	17	15	15	12	-	0.02670	0.03615	0.01267	0.01962	0.01478	-
PORTFOLIO_N_30_L0	54060	15	17	18	16	11	-	0.04222	0.04377	0.01902	0.02735	0.02026	-
PORTFOLIO_N_30_L1	54060	15	18	18	17	11	-	0.04154	0.04821	0.01899	0.02813	0.02007	-
PORTFOLIO_N_30_L2	54060	14	20	17	15	14	-	0.03917	0.05034	0.01822	0.02646	0.02502	-
PORTFOLIO_N_30_L3	54060	14	20	17	15	12	-	0.03929	0.05799	0.01819	0.02625	0.02192	-
PORTFOLIO_N_30_L4	54060	14	17	18	15	13	-	0.03964	0.04580	0.01906	0.02649	0.02396	-
PORTFOLIO_N_30_L5	54060	14	18	15	16	11	-	0.03956	0.04614	0.01716	0.02728	0.02006	-
PORTFOLIO_N_30_L6	54060	12	19	16	16	11	-	0.03474	0.04898	0.01781	0.02729	0.02011	-
PORTFOLIO_N_30_L7	54060	14	17	16	16	11	-	0.03936	0.04394	0.01761	0.02717	0.02021	-
PORTFOLIO_N_30_L8	54060	15	17	17	15	12	-	0.04215	0.04182	0.01836	0.02652	0.02186	-
PORTFOLIO_N_30_L9	54060	14	18	17	17	12	-	0.03966	0.04980	0.01842	0.02831	0.02199	-
PORTFOLIO_N_30_L10	54060	13	18	16	15	11	-	0.03722	0.04368	0.01767	0.02632	0.02027	-
PORTFOLIO_N_30_L11	54060	14	18	16	16	12	-	0.03977	0.04644	0.01754	0.02713	0.02191	-
PORTFOLIO_N_30_L12	54060	14	18	16	15	11	-	0.03923	0.04913	0.01778	0.02649	0.02034	-
PORTFOLIO_N_30_L13	54060	15	21	17	16	12	-	0.04164	0.05595	0.01808	0.02713	0.02175	-
PORTFOLIO_N_30_L14	54060	16	19	17	15	11	-	0.04419	0.05431	0.01838	0.02643	0.02015	-
PORTFOLIO_N_30_L15	54060	14	19	17	15	12	-	0.03961	0.04777	0.01842	0.02656	0.02187	-
PORTFOLIO_N_30_L16	54060	15	17	18	15	14	-	0.04202	0.04420	0.01883	0.02649	0.02490	-
PORTFOLIO_N_30_L17	54060	16	16	17	14	11	-	0.04433	0.04140	0.01831	0.02563	0.02025	-
PORTFOLIO_N_30_L18	54060	15	19	16	14	12	-	0.04180	0.05000	0.01772	0.02550	0.02201	-
PORTFOLIO_N_30_L19	54060	16	18	16	15	11	-	0.04432	0.04734	0.01777	0.02642	0.02010	-
PORTFOLIO_N_35_L0	71820	15	19	18	19	12	-	0.05816	0.05931	0.02139	0.03880	0.03173	-
PORTFOLIO_N_35_L1	71820	15	20	17	16	12	-	0.05906	0.06253	0.02106	0.03483	0.03089	-
PORTFOLIO_N_35_L2	71820	14	18	15	14	12	-	0.05513	0.06219	0.01980	0.03284	0.03091	-
PORTFOLIO_N_35_L3	71820	15	20	16	17	13	-	0.05894	0.06915	0.02002	0.03579	0.03306	-

**Table 5: Iterations and solver runtimes for portfolio optimization problems**

Problem	Size	Iterations						Solver Runtime (s)					
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM
PORTFOLIO_N_35_L4	71820	16	19	16	15	13	-	0.06250	0.06503	0.02030	0.03380	0.03310	-
PORTFOLIO_N_35_L5	71820	14	19	17	15	13	-	0.05460	0.06351	0.02085	0.03382	0.03323	-
PORTFOLIO_N_35_L6	71820	15	18	18	15	12	-	0.05876	0.05876	0.02086	0.03374	0.03071	-
PORTFOLIO_N_35_L7	71820	17	17	17	15	12	-	0.06557	0.05745	0.02104	0.03375	0.03093	-
PORTFOLIO_N_35_L8	71820	15	20	17	17	11	-	0.05911	0.06865	0.02056	0.03577	0.02855	-
PORTFOLIO_N_35_L9	71820	14	17	19	17	12	-	0.05430	0.05969	0.02247	0.03597	0.03085	-
PORTFOLIO_N_35_L10	71820	15	18	17	16	12	-	0.05799	0.05773	0.02076	0.03484	0.03083	-
PORTFOLIO_N_35_L11	71820	15	20	16	16	12	-	0.05800	0.06867	0.02054	0.03480	0.03071	-
PORTFOLIO_N_35_L12	71820	15	18	16	15	11	-	0.05866	0.06290	0.01991	0.03373	0.02864	-
PORTFOLIO_N_35_L13	71820	15	20	16	17	12	-	0.05746	0.06915	0.02026	0.03583	0.03091	-
PORTFOLIO_N_35_L14	71820	16	20	18	18	13	-	0.06144	0.06523	0.02127	0.03676	0.03324	-
PORTFOLIO_N_35_L15	71820	15	19	16	15	11	-	0.05826	0.06382	0.02012	0.03400	0.02846	-
PORTFOLIO_N_35_L16	71820	14	18	19	16	13	-	0.05486	0.06249	0.02178	0.03498	0.03310	-
PORTFOLIO_N_35_L17	71820	13	17	16	14	12	-	0.05057	0.05973	0.02016	0.03373	0.03074	-
PORTFOLIO_N_35_L18	71820	16	19	18	17	13	-	0.06121	0.06841	0.02141	0.03582	0.03322	-
PORTFOLIO_N_35_L19	71820	16	17	18	16	12	-	0.06150	0.05831	0.02125	0.03476	0.03118	-

**Table 6: Iterations and solver runtimes for oscillating masses problems**

Problem	Size	Iterations							Solver Runtime (s)						
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN
OSCILLATING_MASSES_N_8_1_0	1344	5	13	11	11	5	5	7	0.00049	0.00108	0.00222	0.00187	0.00027	0.00011	0.00014
OSCILLATING_MASSES_N_8_1_1	1344	5	14	8	10	5	5	6	0.00048	0.00117	0.00218	0.00181	0.00028	0.00012	0.00013
OSCILLATING_MASSES_N_8_1_2	1344	5	17	9	11	5	5	6	0.00048	0.00139	0.00216	0.00189	0.00026	0.00011	0.00013
OSCILLATING_MASSES_N_8_1_3	1344	5	14	8	11	5	5	6	0.00047	0.00115	0.00217	0.00190	0.00026	0.00011	0.00012
OSCILLATING_MASSES_N_8_1_4	1344	5	12	10	11	5	5	7	0.00049	0.00099	0.00219	0.00189	0.00029	0.00011	0.00015
OSCILLATING_MASSES_N_8_1_5	1344	5	12	8	11	5	5	6	0.00047	0.00100	0.00216	0.00190	0.00026	0.00011	0.00013
OSCILLATING_MASSES_N_8_1_6	1344	10	13	12	11	7	7	10	0.00083	0.00110	0.00223	0.00188	0.00036	0.00015	0.00020
OSCILLATING_MASSES_N_8_1_7	1344	5	13	9	10	5	5	6	0.00048	0.00104	0.00219	0.00178	0.00027	0.00011	0.00012
OSCILLATING_MASSES_N_8_1_8	1344	5	14	9	11	4	4	6	0.00048	0.00112	0.00217	0.00188	0.00023	0.00009	0.00012
OSCILLATING_MASSES_N_8_1_9	1344	5	15	8	11	5	5	6	0.00048	0.00121	0.00219	0.00153	0.00027	0.00011	0.00013
OSCILLATING_MASSES_N_8_1_10	1344	5	14	9	11	5	5	6	0.00048	0.00111	0.00221	0.00190	0.00026	0.00011	0.00013
OSCILLATING_MASSES_N_8_1_11	1344	5	13	8	13	5	5	6	0.00048	0.00105	0.00217	0.00210	0.00028	0.00011	0.00013
OSCILLATING_MASSES_N_8_1_12	1344	5	14	9	12	5	5	7	0.00048	0.00113	0.00220	0.00203	0.00027	0.00011	0.00014
OSCILLATING_MASSES_N_8_1_13	1344	5	12	8	12	5	5	6	0.00048	0.00097	0.00216	0.00160	0.00027	0.00011	0.00012
OSCILLATING_MASSES_N_8_1_14	1344	5	12	8	9	5	5	6	0.00048	0.00098	0.00215	0.00138	0.00026	0.00011	0.00013
OSCILLATING_MASSES_N_8_1_15	1344	5	13	8	11	5	5	7	0.00048	0.00109	0.00218	0.00156	0.00026	0.00011	0.00015
OSCILLATING_MASSES_N_8_1_16	1344	6	14	10	11	6	6	8	0.00055	0.00117	0.00220	0.00190	0.00031	0.00013	0.00016
OSCILLATING_MASSES_N_8_1_17	1344	5	13	10	12	5	5	6	0.00047	0.00105	0.00221	0.00199	0.00027	0.00011	0.00014
OSCILLATING_MASSES_N_8_1_18	1344	5	12	9	11	4	4	6	0.00048	0.00100	0.00218	0.00190	0.00024	0.00009	0.00013
OSCILLATING_MASSES_N_8_1_19	1344	5	12	10	11	5	5	6	0.00048	0.00100	0.00221	0.00187	0.00026	0.00011	0.00013
OSCILLATING_MASSES_N_20_1_0	3312	5	15	9	12	5	5	7	0.00110	0.00300	0.00265	0.00541	0.00062	0.00029	0.00039
OSCILLATING_MASSES_N_20_1_1	3312	5	14	9	12	5	5	7	0.00117	0.00275	0.00266	0.00523	0.00061	0.00029	0.00041
OSCILLATING_MASSES_N_20_1_2	3312	5	15	9	12	5	5	7	0.00115	0.00290	0.00266	0.00538	0.00065	0.00029	0.00042
OSCILLATING_MASSES_N_20_1_3	3312	7	14	10	13	7	7	8	0.00145	0.00274	0.00270	0.00561	0.00080	0.00039	0.00046
OSCILLATING_MASSES_N_20_1_4	3312	7	14	10	14	7	7	9	0.00151	0.00275	0.00271	0.00579	0.00077	0.00039	0.00051
OSCILLATING_MASSES_N_20_1_5	3312	5	14	8	13	5	5	6	0.00113	0.00272	0.00262	0.00553	0.00062	0.00029	0.00035
OSCILLATING_MASSES_N_20_1_6	3312	5	16	8	11	5	5	7	0.00114	0.00308	0.00259	0.00510	0.00062	0.00029	0.00040
OSCILLATING_MASSES_N_20_1_7	3312	5	14	9	12	5	5	6	0.00115	0.00278	0.00265	0.00517	0.00062	0.00029	0.00035
OSCILLATING_MASSES_N_20_1_8	3312	5	13	8	12	5	5	7	0.00116	0.00259	0.00260	0.00530	0.00064	0.00029	0.00041
OSCILLATING_MASSES_N_20_1_9	3312	5	16	9	13	5	5	6	0.00113	0.00306	0.00266	0.00562	0.00061	0.00029	0.00035
OSCILLATING_MASSES_N_20_1_10	3312	5	14	8	12	5	5	6	0.00114	0.00274	0.00262	0.00547	0.00062	0.00029	0.00034
OSCILLATING_MASSES_N_20_1_11	3312	5	14	10	12	5	5	7	0.00114	0.00276	0.00270	0.00532	0.00064	0.00029	0.00041
OSCILLATING_MASSES_N_20_1_12	3312	5	15	9	14	5	5	7	0.00118	0.00287	0.00263	0.00580	0.00062	0.00029	0.00041
OSCILLATING_MASSES_N_20_1_13	3312	5	14	8	12	5	5	6	0.00112	0.00268	0.00262	0.00486	0.00061	0.00029	0.00036
OSCILLATING_MASSES_N_20_1_14	3312	5	15	9	12	5	5	6	0.00111	0.00289	0.00265	0.00552	0.00063	0.00029	0.00035
OSCILLATING_MASSES_N_20_1_15	3312	5	13	8	11	5	5	6	0.00113	0.00260	0.00259	0.00510	0.00062	0.00029	0.00036
OSCILLATING_MASSES_N_20_1_16	3312	5	14	8	12	5	5	6	0.00113	0.00276	0.00259	0.00510	0.00063	0.00029	0.00034
OSCILLATING_MASSES_N_20_1_17	3312	5	15	10	13	5	5	7	0.00115	0.00294	0.00266	0.00533	0.00062	0.00029	0.00040
OSCILLATING_MASSES_N_20_1_18	3312	5	15	8	13	5	5	6	0.00113	0.00280	0.00261	0.00560	0.00063	0.00029	0.00035
OSCILLATING_MASSES_N_20_1_19	3312	5	15	10	14	5	5	7	0.00109	0.00298	0.00267	0.00543	0.00061	0.00029	0.00039
OSCILLATING_MASSES_N_32_1_0	5280	5	14	9	13	5	5	-	0.00179	0.00420	0.00309	0.00885	0.00097	0.00048	-
OSCILLATING_MASSES_N_32_1_1	5280	5	14	8	13	5	5	-	0.00181	0.00434	0.00306	0.00881	0.00096	0.00049	-
OSCILLATING_MASSES_N_32_1_2	5280	5	15	8	13	5	5	-	0.00178	0.00456	0.00303	0.00882	0.00095	0.00048	-
OSCILLATING_MASSES_N_32_1_3	5280	7	13	9	13	6	6	-	0.00226	0.00398	0.00308	0.00925	0.00110	0.00057	-
OSCILLATING_MASSES_N_32_1_4	5280	9	15	10	15	8	8	-	0.00285	0.00470	0.00318	0.00958	0.00137	0.00078	-
OSCILLATING_MASSES_N_32_1_5	5280	5	15	8	10	5	5	-	0.00177	0.00457	0.00306	0.00761	0.00097	0.00049	-

**Table 6: Iterations and solver runtimes for oscillating masses problems**

Problem	Size	Iterations							Solver Runtime (s)						
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN
OSCILLATING_MASSES_N_32_l_6	5280	6	14	7	13	6	6	-	0.00197	0.00419	0.00299	0.00929	0.00110	0.00058	-
OSCILLATING_MASSES_N_32_l_7	5280	6	15	9	13	6	6	-	0.00202	0.00450	0.00307	0.00865	0.00113	0.00056	-
OSCILLATING_MASSES_N_32_l_8	5280	5	16	9	13	5	5	-	0.00182	0.00486	0.00311	0.00885	0.00097	0.00048	-
OSCILLATING_MASSES_N_32_l_9	5280	5	14	8	12	5	5	-	0.00178	0.00429	0.00306	0.00839	0.00097	0.00048	-
OSCILLATING_MASSES_N_32_l_10	5280	6	15	10	13	5	5	-	0.00204	0.00462	0.00323	0.00882	0.00098	0.00049	-
OSCILLATING_MASSES_N_32_l_11	5280	5	13	8	12	5	5	-	0.00179	0.00395	0.00307	0.00842	0.00098	0.00049	-
OSCILLATING_MASSES_N_32_l_12	5280	5	15	8	11	5	5	-	0.00177	0.00441	0.00305	0.00849	0.00098	0.00049	-
OSCILLATING_MASSES_N_32_l_13	5280	5	15	9	13	5	5	-	0.00179	0.00464	0.00314	0.00887	0.00115	0.00050	-
OSCILLATING_MASSES_N_32_l_14	5280	6	16	9	13	6	6	-	0.00199	0.00482	0.00312	0.00895	0.00110	0.00058	-
OSCILLATING_MASSES_N_32_l_15	5280	5	16	8	12	5	5	-	0.00179	0.00486	0.00305	0.00842	0.00096	0.00049	-
OSCILLATING_MASSES_N_32_l_16	5280	5	15	9	12	5	5	-	0.00179	0.00459	0.00311	0.00846	0.00097	0.00049	-
OSCILLATING_MASSES_N_32_l_17	5280	5	15	8	12	5	5	-	0.00175	0.00460	0.00302	0.00850	0.00102	0.00049	-
OSCILLATING_MASSES_N_32_l_18	5280	5	15	8	12	5	5	-	0.00179	0.00471	0.00298	0.00845	0.00100	0.00049	-
OSCILLATING_MASSES_N_32_l_19	5280	5	15	8	13	5	5	-	0.00181	0.00465	0.00304	0.00916	0.00098	0.00049	-
OSCILLATING_MASSES_N_44_l_0	7248	5	15	9	11	5	5	-	0.00237	0.00623	0.00356	0.01125	0.00132	0.00069	-
OSCILLATING_MASSES_N_44_l_1	7248	6	16	10	12	6	6	-	0.00277	0.00660	0.00365	0.01121	0.00148	0.00083	-
OSCILLATING_MASSES_N_44_l_2	7248	6	15	11	14	6	6	-	0.00282	0.00623	0.00376	0.01247	0.00156	0.00081	-
OSCILLATING_MASSES_N_44_l_3	7248	5	14	8	12	5	5	-	0.00247	0.00604	0.00350	0.01175	0.00134	0.00070	-
OSCILLATING_MASSES_N_44_l_4	7248	5	16	8	13	5	5	-	0.00241	0.00676	0.00348	0.01229	0.00138	0.00070	-
OSCILLATING_MASSES_N_44_l_5	7248	5	14	8	13	5	5	-	0.00245	0.00587	0.00349	0.01222	0.00133	0.00070	-
OSCILLATING_MASSES_N_44_l_6	7248	6	15	9	14	6	6	-	0.00287	0.00642	0.00361	0.01279	0.00150	0.00082	-
OSCILLATING_MASSES_N_44_l_7	7248	5	15	9	12	5	5	-	0.00243	0.00638	0.00355	0.01170	0.00146	0.00070	-
OSCILLATING_MASSES_N_44_l_8	7248	5	15	9	12	5	5	-	0.00245	0.00643	0.00359	0.01171	0.00134	0.00069	-
OSCILLATING_MASSES_N_44_l_9	7248	5	15	9	12	5	5	-	0.00245	0.00624	0.00359	0.01178	0.00130	0.00069	-
OSCILLATING_MASSES_N_44_l_10	7248	5	15	9	13	5	5	-	0.00252	0.00618	0.00358	0.01230	0.00133	0.00069	-
OSCILLATING_MASSES_N_44_l_11	7248	5	14	9	13	5	5	-	0.00240	0.00600	0.00358	0.01233	0.00136	0.00068	-
OSCILLATING_MASSES_N_44_l_12	7248	5	16	9	12	5	5	-	0.00241	0.00651	0.00356	0.01173	0.00133	0.00070	-
OSCILLATING_MASSES_N_44_l_13	7248	5	14	8	12	5	5	-	0.00246	0.00607	0.00352	0.01169	0.00133	0.00069	-
OSCILLATING_MASSES_N_44_l_14	7248	5	16	8	13	5	5	-	0.00249	0.00677	0.00352	0.01231	0.00133	0.00069	-
OSCILLATING_MASSES_N_44_l_15	7248	5	14	10	13	5	5	-	0.00241	0.00592	0.00366	0.01204	0.00133	0.00071	-
OSCILLATING_MASSES_N_44_l_16	7248	5	16	9	13	5	5	-	0.00245	0.00671	0.00357	0.01214	0.00132	0.00068	-
OSCILLATING_MASSES_N_44_l_17	7248	5	15	9	13	5	5	-	0.00244	0.00638	0.00362	0.01167	0.00138	0.00069	-
OSCILLATING_MASSES_N_44_l_18	7248	5	14	9	13	5	5	-	0.00245	0.00584	0.00352	0.01216	0.00134	0.00071	-
OSCILLATING_MASSES_N_44_l_19	7248	5	14	9	12	5	5	-	0.00248	0.00594	0.00366	0.01185	0.00137	0.00068	-
OSCILLATING_MASSES_N_56_l_0	9216	5	15	9	13	5	5	-	0.00312	0.00795	0.00405	0.01711	0.00171	0.00092	-
OSCILLATING_MASSES_N_56_l_1	9216	5	16	9	12	5	5	-	0.00303	0.00856	0.00404	0.01631	0.00167	0.00092	-
OSCILLATING_MASSES_N_56_l_2	9216	5	16	10	11	5	5	-	0.00303	0.00849	0.00416	0.01550	0.00189	0.00093	-
OSCILLATING_MASSES_N_56_l_3	9216	5	15	8	12	5	5	-	0.00319	0.00797	0.00396	0.01628	0.00172	0.00092	-
OSCILLATING_MASSES_N_56_l_4	9216	5	15	9	13	5	5	-	0.00315	0.00786	0.00409	0.01698	0.00167	0.00093	-
OSCILLATING_MASSES_N_56_l_5	9216	7	16	8	13	7	7	-	0.00407	0.00848	0.00392	0.01688	0.00217	0.00125	-
OSCILLATING_MASSES_N_56_l_6	9216	5	14	9	12	5	5	-	0.00313	0.00751	0.00410	0.01632	0.00189	0.00092	-
OSCILLATING_MASSES_N_56_l_7	9216	5	19	8	13	5	5	-	0.00312	0.01001	0.00397	0.01723	0.00169	0.00096	-
OSCILLATING_MASSES_N_56_l_8	9216	5	14	9	12	5	5	-	0.00312	0.00765	0.00404	0.01606	0.00170	0.00093	-
OSCILLATING_MASSES_N_56_l_9	9216	6	16	9	13	6	6	-	0.00354	0.00851	0.00406	0.01712	0.00192	0.00110	-
OSCILLATING_MASSES_N_56_l_10	9216	5	19	9	13	5	5	-	0.00313	0.01006	0.00401	0.01689	0.00169	0.00093	-
OSCILLATING_MASSES_N_56_l_11	9216	5	14	10	12	5	5	-	0.00311	0.00758	0.00416	0.01613	0.00168	0.00095	-

**Table 6: Iterations and solver runtimes for oscillating masses problems**

Problem	Size	Iterations							Solver Runtime (s)						
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN
OSCILLATING_MASSES_N_56_1_12	9216	5	15	8	12	5	5	-	0.00311	0.00801	0.00397	0.01629	0.00171	0.00092	-
OSCILLATING_MASSES_N_56_1_13	9216	5	14	8	12	5	5	-	0.00311	0.00760	0.00394	0.01614	0.00169	0.00092	-
OSCILLATING_MASSES_N_56_1_14	9216	5	15	8	12	5	5	-	0.00311	0.00783	0.00399	0.01622	0.00171	0.00095	-
OSCILLATING_MASSES_N_56_1_15	9216	6	14	9	13	6	6	-	0.00357	0.00749	0.00407	0.01692	0.00193	0.00109	-
OSCILLATING_MASSES_N_56_1_16	9216	5	16	8	13	5	5	-	0.00311	0.00849	0.00394	0.01687	0.00167	0.00093	-
OSCILLATING_MASSES_N_56_1_17	9216	6	15	9	14	6	6	-	0.00353	0.00801	0.00404	0.01801	0.00191	0.00110	-
OSCILLATING_MASSES_N_56_1_18	9216	5	16	8	13	5	5	-	0.00312	0.00879	0.00395	0.01696	0.00168	0.00091	-
OSCILLATING_MASSES_N_56_1_19	9216	7	16	10	16	7	7	-	0.00422	0.00866	0.00424	0.01921	0.00216	0.00128	-
OSCILLATING_MASSES_N_76_1_0	12496	5	16	9	12	5	-	-	0.00417	0.01169	0.00509	0.02337	0.00229	-	-
OSCILLATING_MASSES_N_76_1_1	12496	6	15	10	12	5	-	-	0.00491	0.01090	0.00521	0.02242	0.00231	-	-
OSCILLATING_MASSES_N_76_1_2	12496	5	15	9	12	5	-	-	0.00425	0.01106	0.00518	0.02213	0.00238	-	-
OSCILLATING_MASSES_N_76_1_3	12496	5	14	8	12	6	-	-	0.00430	0.01017	0.00494	0.02243	0.00265	-	-
OSCILLATING_MASSES_N_76_1_4	12496	5	14	8	12	5	-	-	0.00418	0.01023	0.00488	0.02254	0.00231	-	-
OSCILLATING_MASSES_N_76_1_5	12496	5	15	9	14	5	-	-	0.00427	0.01097	0.00511	0.02418	0.00245	-	-
OSCILLATING_MASSES_N_76_1_6	12496	5	15	8	13	5	-	-	0.00424	0.01141	0.00492	0.02355	0.00258	-	-
OSCILLATING_MASSES_N_76_1_7	12496	5	15	9	14	5	-	-	0.00419	0.01078	0.00506	0.02419	0.00257	-	-
OSCILLATING_MASSES_N_76_1_8	12496	6	14	9	12	6	-	-	0.00488	0.01022	0.00507	0.02243	0.00262	-	-
OSCILLATING_MASSES_N_76_1_9	12496	5	14	9	13	5	-	-	0.00423	0.01017	0.00509	0.02297	0.00228	-	-
OSCILLATING_MASSES_N_76_1_10	12496	6	15	9	11	6	-	-	0.00466	0.01093	0.00511	0.02109	0.00264	-	-
OSCILLATING_MASSES_N_76_1_11	12496	5	14	8	12	5	-	-	0.00423	0.01031	0.00494	0.02259	0.00233	-	-
OSCILLATING_MASSES_N_76_1_12	12496	7	15	10	13	7	-	-	0.00543	0.01121	0.00517	0.02368	0.00294	-	-
OSCILLATING_MASSES_N_76_1_13	12496	7	17	11	14	6	-	-	0.00550	0.01210	0.00542	0.02445	0.00259	-	-
OSCILLATING_MASSES_N_76_1_14	12496	5	15	10	14	5	-	-	0.00424	0.01108	0.00518	0.02432	0.00224	-	-
OSCILLATING_MASSES_N_76_1_15	12496	5	16	9	13	5	-	-	0.00425	0.01168	0.00506	0.02324	0.00231	-	-
OSCILLATING_MASSES_N_76_1_16	12496	5	15	9	11	5	-	-	0.00427	0.01089	0.00507	0.02136	0.00233	-	-
OSCILLATING_MASSES_N_76_1_17	12496	5	15	9	13	5	-	-	0.00429	0.01105	0.00502	0.02345	0.00230	-	-
OSCILLATING_MASSES_N_76_1_18	12496	5	16	9	12	5	-	-	0.00421	0.01152	0.00504	0.02265	0.00228	-	-
OSCILLATING_MASSES_N_76_1_19	12496	5	15	9	12	5	-	-	0.00420	0.01093	0.00503	0.02258	0.00228	-	-
OSCILLATING_MASSES_N_96_1_0	15776	5	15	8	13	5	-	-	0.00537	0.01378	0.00572	0.02968	0.00287	-	-
OSCILLATING_MASSES_N_96_1_1	15776	6	16	9	13	6	-	-	0.00618	0.01448	0.00589	0.02978	0.00333	-	-
OSCILLATING_MASSES_N_96_1_2	15776	5	16	8	12	5	-	-	0.00528	0.01466	0.00580	0.02827	0.00288	-	-
OSCILLATING_MASSES_N_96_1_3	15776	5	17	8	13	5	-	-	0.00530	0.01562	0.00575	0.02965	0.00287	-	-
OSCILLATING_MASSES_N_96_1_4	15776	5	13	10	11	5	-	-	0.00526	0.01187	0.00609	0.02489	0.00291	-	-
OSCILLATING_MASSES_N_96_1_5	15776	5	15	8	14	5	-	-	0.00535	0.01372	0.00577	0.03095	0.00290	-	-
OSCILLATING_MASSES_N_96_1_6	15776	5	14	9	12	5	-	-	0.00536	0.01284	0.00593	0.02815	0.00332	-	-
OSCILLATING_MASSES_N_96_1_7	15776	5	15	9	13	5	-	-	0.00543	0.01424	0.00603	0.02947	0.00290	-	-
OSCILLATING_MASSES_N_96_1_8	15776	5	16	8	13	5	-	-	0.00552	0.01457	0.00577	0.02981	0.00289	-	-
OSCILLATING_MASSES_N_96_1_9	15776	5	15	8	11	5	-	-	0.00536	0.01356	0.00574	0.02750	0.00293	-	-
OSCILLATING_MASSES_N_96_1_10	15776	6	14	10	13	6	-	-	0.00589	0.01277	0.00609	0.02968	0.00327	-	-
OSCILLATING_MASSES_N_96_1_11	15776	6	15	9	12	6	-	-	0.00608	0.01380	0.00583	0.02834	0.00331	-	-
OSCILLATING_MASSES_N_96_1_12	15776	5	17	9	13	5	-	-	0.00527	0.01555	0.00592	0.02948	0.00289	-	-
OSCILLATING_MASSES_N_96_1_13	15776	5	15	8	13	5	-	-	0.00531	0.01377	0.00574	0.02977	0.00293	-	-
OSCILLATING_MASSES_N_96_1_14	15776	5	17	10	13	5	-	-	0.00534	0.01547	0.00612	0.02916	0.00288	-	-
OSCILLATING_MASSES_N_96_1_15	15776	6	15	9	12	6	-	-	0.00603	0.01406	0.00591	0.02810	0.00328	-	-
OSCILLATING_MASSES_N_96_1_16	15776	5	14	8	13	5	-	-	0.00543	0.01295	0.00574	0.02947	0.00290	-	-
OSCILLATING_MASSES_N_96_1_17	15776	5	16	8	14	5	-	-	0.00529	0.01427	0.00579	0.03088	0.00290	-	-

**Table 6: Iterations and solver runtimes for oscillating masses problems**

Problem	Size	Iterations							Solver Runtime (s)						
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN
OSCILLATING_MASSES_N_96_L_18	15776	5	14	9	13	5	-	-	0.00517	0.01306	0.00603	0.02977	0.00290	-	-
OSCILLATING_MASSES_N_96_L_19	15776	5	16	8	13	5	-	-	0.00527	0.01481	0.00577	0.02987	0.00295	-	-
OSCILLATING_MASSES_N_116_L_0	19056	5	15	9	13	5	-	-	0.00645	0.01652	0.00678	0.03539	0.00354	-	-
OSCILLATING_MASSES_N_116_L_1	19056	6	14	9	12	6	-	-	0.00723	0.01589	0.00676	0.03201	0.00404	-	-
OSCILLATING_MASSES_N_116_L_2	19056	5	15	10	12	5	-	-	0.00653	0.01675	0.00712	0.03463	0.00351	-	-
OSCILLATING_MASSES_N_116_L_3	19056	5	16	8	12	5	-	-	0.00637	0.01775	0.00654	0.03461	0.00347	-	-
OSCILLATING_MASSES_N_116_L_4	19056	5	15	8	12	5	-	-	0.00637	0.01692	0.00664	0.03405	0.00351	-	-
OSCILLATING_MASSES_N_116_L_5	19056	6	16	8	13	6	-	-	0.00724	0.01791	0.00661	0.03601	0.00395	-	-
OSCILLATING_MASSES_N_116_L_6	19056	6	14	9	11	6	-	-	0.00670	0.01542	0.00674	0.03227	0.00398	-	-
OSCILLATING_MASSES_N_116_L_7	19056	5	15	9	14	5	-	-	0.00663	0.01652	0.00677	0.03753	0.00347	-	-
OSCILLATING_MASSES_N_116_L_8	19056	7	17	9	14	7	-	-	0.00864	0.01920	0.00681	0.03718	0.00454	-	-
OSCILLATING_MASSES_N_116_L_9	19056	5	15	10	13	5	-	-	0.00646	0.01671	0.00698	0.03599	0.00350	-	-
OSCILLATING_MASSES_N_116_L_10	19056	5	14	9	12	5	-	-	0.00640	0.01547	0.00678	0.03438	0.00354	-	-
OSCILLATING_MASSES_N_116_L_11	19056	5	16	9	13	5	-	-	0.00628	0.01727	0.00673	0.03629	0.00349	-	-
OSCILLATING_MASSES_N_116_L_12	19056	5	16	9	12	5	-	-	0.00635	0.01787	0.00674	0.03427	0.00355	-	-
OSCILLATING_MASSES_N_116_L_13	19056	5	15	11	12	6	-	-	0.00641	0.01696	0.00725	0.03417	0.00401	-	-
OSCILLATING_MASSES_N_116_L_14	19056	6	15	10	13	6	-	-	0.00744	0.01671	0.00700	0.03591	0.00397	-	-
OSCILLATING_MASSES_N_116_L_15	19056	5	16	9	13	5	-	-	0.00639	0.01772	0.00676	0.03526	0.00353	-	-
OSCILLATING_MASSES_N_116_L_16	19056	5	14	9	13	5	-	-	0.00639	0.01565	0.00678	0.03580	0.00362	-	-
OSCILLATING_MASSES_N_116_L_17	19056	5	16	8	13	5	-	-	0.00646	0.01823	0.00662	0.03592	0.00346	-	-
OSCILLATING_MASSES_N_116_L_18	19056	5	15	8	11	6	-	-	0.00638	0.01739	0.00661	0.03253	0.00402	-	-
OSCILLATING_MASSES_N_116_L_19	19056	5	16	9	13	5	-	-	0.00655	0.01813	0.00678	0.03584	0.00350	-	-
OSCILLATING_MASSES_N_136_L_0	22336	5	16	9	13	5	-	-	0.00746	0.02089	0.00775	0.04277	0.00412	-	-
OSCILLATING_MASSES_N_136_L_1	22336	5	15	9	12	5	-	-	0.00758	0.01966	0.00779	0.04064	0.00407	-	-
OSCILLATING_MASSES_N_136_L_2	22336	5	14	10	12	5	-	-	0.00765	0.01879	0.00805	0.04135	0.00411	-	-
OSCILLATING_MASSES_N_136_L_3	22336	5	16	9	12	5	-	-	0.00758	0.02064	0.00774	0.04057	0.00413	-	-
OSCILLATING_MASSES_N_136_L_4	22336	5	15	8	13	5	-	-	0.00747	0.02013	0.00759	0.04242	0.00409	-	-
OSCILLATING_MASSES_N_136_L_5	22336	5	16	9	13	5	-	-	0.00752	0.02073	0.00777	0.03960	0.00425	-	-
OSCILLATING_MASSES_N_136_L_6	22336	5	15	12	14	5	-	-	0.00764	0.01978	0.00859	0.04500	0.00409	-	-
OSCILLATING_MASSES_N_136_L_7	22336	5	15	10	13	5	-	-	0.00744	0.01972	0.00795	0.04260	0.00411	-	-
OSCILLATING_MASSES_N_136_L_8	22336	5	15	8	13	5	-	-	0.00744	0.01993	0.00757	0.04299	0.00404	-	-
OSCILLATING_MASSES_N_136_L_9	22336	5	14	9	12	5	-	-	0.00746	0.01840	0.00776	0.04081	0.00414	-	-
OSCILLATING_MASSES_N_136_L_10	22336	6	14	10	13	6	-	-	0.00844	0.01862	0.00804	0.04146	0.00469	-	-
OSCILLATING_MASSES_N_136_L_11	22336	6	15	8	13	6	-	-	0.00843	0.01963	0.00749	0.04219	0.00471	-	-
OSCILLATING_MASSES_N_136_L_12	22336	5	15	8	12	5	-	-	0.00752	0.01976	0.00757	0.04099	0.00449	-	-
OSCILLATING_MASSES_N_136_L_13	22336	5	15	8	14	5	-	-	0.00773	0.02007	0.00764	0.04463	0.00408	-	-
OSCILLATING_MASSES_N_136_L_14	22336	5	16	8	13	5	-	-	0.00747	0.02072	0.00768	0.04340	0.00417	-	-
OSCILLATING_MASSES_N_136_L_15	22336	6	15	9	12	6	-	-	0.00835	0.01984	0.00774	0.04062	0.00481	-	-
OSCILLATING_MASSES_N_136_L_16	22336	6	16	9	13	6	-	-	0.00833	0.02041	0.00778	0.04251	0.00472	-	-
OSCILLATING_MASSES_N_136_L_17	22336	6	16	9	11	5	-	-	0.00830	0.02048	0.00773	0.03879	0.00411	-	-
OSCILLATING_MASSES_N_136_L_18	22336	5	17	9	12	5	-	-	0.00749	0.02231	0.00781	0.04124	0.00415	-	-
OSCILLATING_MASSES_N_136_L_19	22336	6	16	8	12	6	-	-	0.00835	0.02071	0.00741	0.04067	0.00473	-	-
OSCILLATING_MASSES_N_156_L_0	25616	6	15	10	13	7	-	-	0.01005	0.02309	0.00922	0.04915	0.00601	-	-
OSCILLATING_MASSES_N_156_L_1	25616	6	14	8	13	6	-	-	0.00959	0.02071	0.00842	0.04941	0.00539	-	-
OSCILLATING_MASSES_N_156_L_2	25616	7	20	11	14	5	-	-	0.01111	0.02926	0.00921	0.05136	0.00469	-	-
OSCILLATING_MASSES_N_156_L_3	25616	5	16	10	14	5	-	-	0.00860	0.02390	0.00897	0.05125	0.00467	-	-

**Table 6: Iterations and solver runtimes for oscillating masses problems**

Problem	Size	Iterations							Solver Runtime (s)						
		CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CVXGEN
OSCILLATING_MASSES_N_156_L4	25616	6	16	10	12	6	-	-	0.00953	0.02348	0.00909	0.04725	0.00544	-	-
OSCILLATING_MASSES_N_156_L5	25616	5	16	9	12	5	-	-	0.00858	0.02381	0.00855	0.04722	0.00475	-	-
OSCILLATING_MASSES_N_156_L6	25616	5	15	10	13	5	-	-	0.00871	0.02267	0.00897	0.04866	0.00466	-	-
OSCILLATING_MASSES_N_156_L7	25616	5	16	8	13	5	-	-	0.00842	0.02388	0.00832	0.04869	0.00468	-	-
OSCILLATING_MASSES_N_156_L8	25616	5	15	9	12	5	-	-	0.00860	0.02269	0.00877	0.04664	0.00473	-	-
OSCILLATING_MASSES_N_156_L9	25616	5	15	9	14	5	-	-	0.00869	0.02238	0.00865	0.05132	0.00481	-	-
OSCILLATING_MASSES_N_156_L10	25616	6	15	9	13	6	-	-	0.00988	0.02311	0.00865	0.04906	0.00544	-	-
OSCILLATING_MASSES_N_156_L11	25616	5	17	9	12	5	-	-	0.00842	0.02520	0.00880	0.04729	0.00471	-	-
OSCILLATING_MASSES_N_156_L12	25616	5	15	9	12	5	-	-	0.00859	0.02292	0.00881	0.04701	0.00471	-	-
OSCILLATING_MASSES_N_156_L13	25616	5	14	10	12	5	-	-	0.00872	0.02105	0.00898	0.04681	0.00473	-	-
OSCILLATING_MASSES_N_156_L14	25616	6	16	8	12	6	-	-	0.00954	0.02290	0.00847	0.04685	0.00539	-	-
OSCILLATING_MASSES_N_156_L15	25616	5	14	8	13	5	-	-	0.00860	0.02131	0.00842	0.04894	0.00474	-	-
OSCILLATING_MASSES_N_156_L16	25616	6	13	7	11	6	-	-	0.00952	0.01997	0.00811	0.04467	0.00543	-	-
OSCILLATING_MASSES_N_156_L17	25616	5	17	10	13	5	-	-	0.00869	0.02538	0.00900	0.04941	0.00476	-	-
OSCILLATING_MASSES_N_156_L18	25616	7	16	11	13	7	-	-	0.01143	0.02377	0.00915	0.04932	0.00598	-	-
OSCILLATING_MASSES_N_156_L19	25616	5	15	9	13	5	-	-	0.00878	0.02218	0.00868	0.04924	0.00471	-	-

**Table 7: Iterations and solver runtimes for mpc problems**

Problem	Size	Iterations						Solver Runtime (s)					
		QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK
FORCESEXAMPLE_3	189	9	9	9	17	10	9	0.00009	0.00003	0.00014	0.00029	0.00194	0.00050
SPACECRAFT_1	1137	9	9	9	16	10	13	0.00039	0.00018	0.00074	0.00110	0.00217	0.00167
DCMOTOR_4	5514	8	8	8	17	13	17	0.00170	0.00084	0.00370	0.00677	0.00459	0.00823
PENDULUM_3	155	5	5	5	10	7	9	0.00005	0.00001	0.00008	0.00014	0.00191	0.00039
FORCESEXAMPLE_2	170	7	7	7	13	12	9	0.00006	0.00002	0.00010	0.00021	0.00188	0.00046
BINARYDISTILLATIONCOLUMN_2	2936	9	9	9	17	12	8	0.00085	0.00043	0.00164	0.00269	0.00288	0.00251
QUADCOPTER_2	3172	7	7	7	13	14	11	0.00105	0.00050	0.00180	0.00274	0.00353	0.00469
PENDULUM_1	435	6	6	5	12	10	14	0.00012	0.00004	0.00020	0.00041	0.00209	0.00091
SHELL_3	739	10	10	12	18	13	14	0.00045	0.00018	0.00085	0.00139	0.00218	0.00113
FIORDOSEXAMPLE_3	119	10	10	8	-	10	12	0.00006	0.00002	0.00009	-	0.00191	0.00039
FIORDOSEXAMPLE_1	119	8	8	7	12	13	10	0.00005	0.00001	0.00008	0.00014	0.00193	0.00037
AIRCRAFT_11	636	11	11	12	23	9	14	0.00026	0.00010	0.00054	0.00097	0.00203	0.00099
TRIPLEINVERTEDPENDULUM_3	2919	8	8	8	19	14	12	0.00079	0.00038	0.00299	0.00406	0.00317	0.00515
ROBOTARM_1	276	5	5	5	-	9	14	0.00008	0.00002	0.00018	-	0.00196	0.00060
AIRCRAFT_12	636	11	11	12	23	9	14	0.00026	0.00011	0.00055	0.00093	0.00202	0.00100
QUADCOPTER_5	3072	-	-	26	-	40	20	-	-	0.00705	-	0.00557	0.00596
DCMOTOR_2	1114	8	8	9	17	11	15	0.00036	0.00015	0.00099	0.00135	0.00235	0.00140
SPRINGMASS_1	21239	10	10	12	24	17	40	0.00706	0.00452	0.03067	0.03561	0.01192	0.10089
AIRCRAFT_10	584	8	8	9	15	10	13	0.00020	0.00008	0.00041	0.00055	0.00204	0.00081
DOUBLEINVERTEDPENDULUM_1	502	5	5	6	13	10	16	0.00012	0.00004	0.00031	0.00054	0.00211	0.00095
SPRINGMASS_3	2144	5	5	7	21	10	21	0.00045	0.00019	0.00107	0.00307	0.00247	0.00548
ROBOTARM_2	1056	10	10	10	-	13	21	0.00042	0.00019	0.00142	-	0.00230	0.00194
BALLONPLATE_1	398	7	7	6	10	12	13	0.00014	0.00005	0.00023	0.00038	0.00200	0.00081
AIRCRAFT_13	3116	14	14	15	25	11	19	0.00141	0.00073	0.00334	0.00523	0.00276	0.00965
AIRCRAFT_3	464	7	7	7	14	16	12	0.00015	0.00005	0.00025	0.00048	0.00209	0.00070
NONLINEARCSTR_2	1164	9	9	10	17	11	11	0.00037	0.00016	0.00085	0.00112	0.00221	0.00164
QUADCOPTER_6	1678	7	7	7	15	13	12	0.00055	0.00025	0.00103	0.00207	0.00265	0.00176
QUADCOPTER_3	7912	7	7	7	13	14	12	0.00266	0.00138	0.00446	0.00676	0.00612	0.01181
AIRCRAFT_4	504	8	8	9	15	10	11	0.00018	0.00007	0.00035	0.00052	0.00203	0.00076
TOYEXAMPLE_1	199	6	6	7	18	9	11	0.00006	0.00002	0.00013	0.00031	0.00194	0.00054
SHELL_1	739	6	6	7	15	11	10	0.00028	0.00011	0.00056	0.00116	0.00216	0.00096
HELICOPTER_2	1051	6	6	5	23	12	18	0.00032	0.00012	0.00068	0.00213	0.00256	0.00230
AIRCRAFT_2	524	8	8	9	17	10	13	0.00017	0.00007	0.00035	0.00053	0.00202	0.00085
BALLONPLATE_2	398	7	7	6	12	10	14	0.00014	0.00005	0.00023	0.00044	0.00201	0.00083
TRIPLEINVERTEDPENDULUM_2	2919	7	7	7	19	13	14	0.00073	0.00034	0.00265	0.00393	0.00312	0.00589
FORCESEXAMPLE_1	171	7	7	7	13	12	9	0.00006	0.00002	0.00010	0.00022	0.00198	0.00046
FIORDOSEXAMPLE_2	139	7	7	8	15	11	10	0.00006	0.00002	0.00012	0.00019	0.00193	0.00040
DCMOTOR_6	1114	6	6	7	17	11	11	0.00030	0.00012	0.00110	0.00134	0.00236	0.00119
BINARYDISTILLATIONCOLUMN_1	2936	9	9	9	17	12	8	0.00085	0.00042	0.00162	0.00262	0.00285	0.00252
DCMOTOR_1	564	7	7	8	15	10	14	0.00018	0.00006	0.00048	0.00062	0.00207	0.00084
BALLONPLATE_4	658	8	8	9	15	11	16	0.00024	0.00010	0.00053	0.00091	0.00208	0.00126
DOUBLEINVERTEDPENDULUM_3	542	5	5	6	15	11	15	0.00013	0.00005	0.00033	0.00065	0.00208	0.00094
QUADCOPTER_1	1592	7	7	7	12	13	9	0.00054	0.00023	0.00092	0.00136	0.00256	0.00137
DCMOTOR_5	1114	8	8	12	18	15	15	0.00036	0.00015	0.00161	0.00143	0.00241	0.00140
SHELL_2	1469	7	7	7	16	11	10	0.00062	0.00027	0.00113	0.00342	0.00269	0.00174
DCMOTOR_3	5514	8	8	10	21	15	20	0.00168	0.00083	0.00532	0.00803	0.00478	0.00868

**Table 7: Iterations and solver runtimes for mpc problems**

Problem	Size	Iterations						Solver Runtime (s)					
		QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	QOCO_CUSTOM	CLARABEL	ECOS	GUROBI	MOSEK
HELICOPTER_3	536	5	5	5	22	10	18	0.00015	0.00004	0.00034	0.00105	0.00210	0.00125
QUADCOPTER_4	3172	10	10	10	15	15	16	0.00139	0.00069	0.00258	0.00383	0.00359	0.00554
DOUBLEINVERTEDPENDULUM_2	502	8	8	9	16	12	18	0.00016	0.00007	0.00041	0.00063	0.00210	0.00101
TRIPLEINVERTEDPENDULUM_1	2739	7	7	7	19	13	12	0.00067	0.00029	0.00250	0.00369	0.00305	0.00489
SPRINGMASS_2	2159	6	6	7	21	10	21	0.00050	0.00022	0.00110	0.00303	0.00266	0.00665
TOYEXAMPLE_5	1909	9	9	11	-	17	16	0.00062	0.00029	0.00151	-	0.00312	0.00473
TOYEXAMPLE_3	199	6	6	12	-	8	15	0.00007	0.00002	0.00019	-	0.00193	0.00065
HELICOPTER_1	201	4	4	4	21	8	13	0.00006	0.00001	0.00011	0.00034	0.00198	0.00049
SPACECRAFT_2	1137	9	9	9	16	10	13	0.00042	0.00018	0.00073	0.00110	0.00214	0.00168
FORCESEXAMPLE_4	369	7	7	7	15	12	10	0.00012	0.00004	0.00020	0.00051	0.00208	0.00076
TOYEXAMPLE_2	389	6	6	6	18	10	11	0.00012	0.00004	0.00020	0.00061	0.00207	0.00080
BALLONPLATE_3	398	7	7	7	14	9	14	0.00014	0.00005	0.00026	0.00052	0.00198	0.00084
TOYEXAMPLE_4	959	9	9	10	-	14	18	0.00034	0.00014	0.00071	-	0.00242	0.00217
NONLINEARCSTR_3	294	7	7	8	16	9	10	0.00009	0.00003	0.00019	0.00027	0.00190	0.00050
NONLINEARCSTR_1	1164	9	9	10	17	11	11	0.00037	0.00016	0.00085	0.00112	0.00222	0.00163
SPRINGMASS_4	4279	7	7	7	22	10	23	0.00111	0.00055	0.00240	0.00637	0.00332	0.01068
AIRCRAFT_1	504	8	8	9	15	10	11	0.00018	0.00007	0.00036	0.00053	0.00202	0.00074
PENDULUM_2	432	8	8	8	13	14	15	0.00014	0.00005	0.00027	0.00045	0.00206	0.00095

**Table 8: Iterations and solver runtimes for Maros–Mészáros problems**

Problem	Size	Iterations					Solver Runtime (s)				
		QOCO	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	CLARABEL	ECOS	GUROBI	MOSEK
LASER	9216	10	10	-	21	24	0.00295	0.00418	-	0.02075	0.03384
QPCBOEI1	4743	21	15	-	22	44	0.00557	0.00623	-	0.00675	0.04344
HUESTIS	39968	7	8	-	14	-	0.01247	0.02049	-	0.01075	-
QBANDM	3007	21	20	30	21	17	0.00283	0.00447	0.00984	0.00504	0.00617
QSC205	775	15	18	17	17	22	0.00085	0.00158	0.00146	0.00349	0.00221
LISWET11	40002	-	-	-	40	31	-	-	-	0.13571	0.27070
LISWET8	40002	-	-	-	417	57	-	-	-	1.72564	0.69070
AUG2DC	60200	0	0	-	2	12	0.01347	0.01958	-	0.02400	0.23210
QSEBA	6548	30	29	-	20	34	0.00622	0.01452	-	0.00404	0.01449
AUG2D	59800	0	0	-	2	14	0.00958	0.01984	-	0.02289	0.31375
QGROW15	7365	18	20	17	17	22	0.00529	0.00997	0.00961	0.00758	0.01993
CONT-200	319196	13	10	30	12	24	3.08655	4.03513	8.48776	0.39624	1.77756
LISWET1	40002	-	-	-	27	45	-	-	-	0.09196	0.50075
ZECEVIC2	9	6	7	8	8	7	0.00003	0.00005	0.00004	0.00255	0.00106
QFFFFF80	8997	93	25	45	26	26	0.06986	0.02945	0.42187	0.01826	0.09688
AUG2DCQP	80400	11	12	-	17	-	0.08231	0.13210	-	0.05561	-
QPILOTNO	16258	38	31	39	26	29	0.07808	0.09577	0.32104	0.03651	0.11626
CVXQP2_M	6733	13	9	30	12	16	0.02790	0.02295	10.49264	0.02690	0.70517
POWELL20	30000	38	-	-	60	-	0.05336	-	-	0.20098	-
STADAT2	15997	17	44	-	18	21	0.00891	0.04415	-	0.01993	0.04982
QPCBOEI2	1623	40	19	-	27	35	0.00236	0.00273	-	0.00413	0.00854
PRIMAL3	22292	11	8	18	18	8	0.01258	0.01602	0.03150	0.01333	0.01050
QSHIP08S	21178	14	15	27	20	17	0.01986	0.02778	1.55641	0.02175	0.15395
DUALC2	1645	18	10	13	9	15	0.00053	0.00056	0.00085	0.00207	0.00080
LISWET4	40002	10	22	-	13	20	0.01496	0.05909	-	0.05490	0.15414
QPCSTAIR	4872	25	22	-	21	34	0.00806	0.01108	-	0.01309	0.02036
PRIMALC2	2076	24	15	30	10	12	0.00120	0.00119	0.00306	0.00213	0.00196
LISWET6	40002	9	17	-	13	21	0.01726	0.05086	-	0.04968	0.17387
QSHIP04L	8506	14	15	25	17	22	0.00711	0.00944	0.01949	0.00936	0.01942
QE226	3824	18	24	22	19	18	0.00422	0.00677	0.02736	0.00802	0.01922
PRIMAL4	17520	9	9	16	16	9	0.01190	0.01330	0.02602	0.00949	0.01691
QSCSD8	13844	11	10	15	14	10	0.00644	0.00955	0.05031	0.01281	0.03718
QSCORPIO	1824	15	11	13	15	14	0.00122	0.00165	0.00236	0.00320	0.00257
CONT-201	290393	12	10	-	12	-	3.68995	5.09101	-	0.37022	-
GOULDQP2	3142	9	12	20	11	7	0.00177	0.00315	0.56048	0.00697	0.04653
LISWET10	40002	-	-	-	125	20	-	-	-	0.40801	0.15746
HS268	40	5	10	15	12	15	0.00003	0.00007	0.00009	0.00301	0.00053
QBORE3D	1834	18	28	24	11	15	0.00198	0.00440	0.00724	0.00253	0.00180
QSHIP12L	83825	18	15	32	18	29	0.20070	0.19269	72.93811	0.05007	2.43260
QSHIP04S	5866	14	14	25	16	20	0.00358	0.00600	0.01307	0.00542	0.01263
QSHARE1B	1415	28	28	43	21	26	0.00190	0.00319	0.00469	0.00467	0.02870
QGFRDXPN	3889	24	20	-	19	-	0.00537	0.00846	-	0.00708	-
CONT-050	19796	10	9	24	10	18	0.03774	0.04240	0.10226	0.01759	0.08744
MOSARP1	8467	9	10	24	16	12	0.00509	0.00884	0.03554	0.01028	0.01954
BOYD2	517049	68	-	-	90	-	1.79598	-	-	0.87993	-
QSCAGR25	2182	17	19	34	18	31	0.00173	0.00351	0.00796	0.00450	0.01722

**Table 8: Iterations and solver runtimes for Maros–Mészáros problems**

Problem	Size	Iterations					Solver Runtime (s)				
		QOCO	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	CLARABEL	ECOS	GUROBI	MOSEK
HUES-MOD	39968	13	12	-	13	18	0.02211	0.02751	-	0.01053	0.07958
YAO	8007	-	-	-	28	34	-	-	-	0.01528	0.05277
CONT-101	72693	10	10	-	11	80	0.30443	0.43120	-	0.08288	1.72033
HS53	24	5	6	9	8	6	0.00003	0.00005	0.00007	0.00261	0.00047
CVXQP1_M	7482	13	10	-	14	22	0.04193	0.03792	-	0.05341	0.45150
STCQP2	48135	7	8	25	10	11	0.07536	0.10185	0.51387	0.03917	0.04279
QGROW22	10930	22	25	20	19	24	0.00963	0.01915	0.01957	0.01047	0.04234
HS35	11	5	7	10	9	9	0.00002	0.00005	0.00003	0.00266	0.00999
STCQP1	48135	7	7	28	9	11	0.03430	0.04184	0.31809	0.00664	0.02806
STADAT1	15997	20	15	-	-	-	0.01204	0.01224	-	-	-
PRIMAL1	6140	12	9	17	18	11	0.00330	0.00377	0.00911	0.00539	0.00803
DUAL3	6441	9	11	13	15	7	0.00287	0.00433	0.00484	0.00864	0.00370
HS21	8	9	8	8	0	11	0.00002	0.00003	0.00003	0.00205	0.00034
QBEACONF	3664	12	-	29	11	14	0.00182	-	0.00648	0.00268	0.00234
QISRAEL	3109	26	25	30	21	22	0.00340	0.00542	0.00873	0.01089	0.01127
LISWET2	40002	9	17	-	15	20	0.01736	0.04994	-	0.05463	0.15266
DTOC3	49994	4	6	-	2	14	0.01034	0.02675	-	0.01440	0.13182
GOULDQP3	3840	6	7	25	8	16	0.00103	0.00196	3.92135	0.00670	0.17622
PRIMAL2	8691	11	8	16	17	7	0.00437	0.00506	0.01278	0.00645	0.00473
TAME	7	4	5	5	6	4	0.00002	0.00004	0.00003	0.00259	0.00034
PRIMALC5	2860	20	13	14	11	11	0.00141	0.00145	0.00216	0.00210	0.00236
AUG3DC	10419	0	0	14	2	7	0.00201	0.00302	0.03647	0.00928	0.01731
AUG3DCQP	14292	9	10	29	16	10	0.01278	0.01802	0.08718	0.01828	0.02658
QSCFXM2	7228	32	30	37	31	37	0.01037	0.01937	0.04069	0.01564	0.06499
VALUES	4428	11	13	-	-	-	0.00165	0.00226	-	-	-
QAFIRO	121	15	13	15	12	14	0.00015	0.00021	0.00018	0.00265	0.00661
QETAMACR	7761	26	19	26	21	21	0.05323	0.04463	0.94189	0.01268	0.06093
QSCSD6	7070	14	13	16	15	13	0.00372	0.00569	0.02306	0.00688	0.04069
QSTAIR	5423	23	33	32	20	38	0.01015	0.01912	0.02920	0.00891	0.03350
PRIMALC1	2514	25	16	27	11	14	0.00216	0.00148	0.00321	0.00279	0.00321
QSHELL	40488	28	36	-	38	26	0.09449	0.12115	-	0.07993	0.18460
CVXQP2_L	67483	22	9	-	15	23	14.21961	8.60789	-	0.47604	390.09870
QSCFXM1	3779	26	25	36	23	35	0.00385	0.00677	0.02216	0.00790	0.03538
DUAL1	3813	10	12	14	15	12	0.00173	0.00259	0.00275	0.00672	0.00236
UBH1	66033	7	17	-	6	11	0.02002	0.07995	-	0.01160	0.05267
DUALC5	2276	10	10	18	11	12	0.00047	0.00071	0.00116	0.00244	0.00067
LISWET9	40002	-	-	-	53	-	-	-	-	0.63997	
Q25FV47	71470	27	25	33	27	41	0.16469	0.17414	4.81007	0.08488	0.33967
PRIMALC8	5182	20	12	14	9	13	0.00247	0.00326	0.00375	0.00220	0.00417
QPCBLEND	657	16	16	17	19	20	0.00055	0.00088	0.00115	0.00355	0.00547
LOTSCHD	72	7	8	18	10	13	0.00004	0.00006	0.00012	0.00201	0.00053
QSCAGR7	585	15	15	29	18	49	0.00042	0.00081	0.00128	0.00274	0.00534
QSCRS8	4472	23	31	33	26	24	0.00487	0.01325	0.01523	0.00647	0.01403
CVXQP1_S	734	8	8	12	11	12	0.00050	0.00076	0.01201	0.00416	0.00402
QADLITTL	567	11	13	17	12	16	0.00037	0.00070	0.00128	0.00330	0.00193
CVXQP2_S	660	8	9	12	12	10	0.00035	0.00058	0.01285	0.00294	0.00394

**Table 8: Iterations and solver runtimes for Maros–Mészáros problems**

Problem	Size	Iterations					Solver Runtime (s)				
		QOCO	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	CLARABEL	ECOS	GUROBI	MOSEK
QRECIPE	988	18	16	19	19	18	0.00111	0.00149	0.00185	0.00347	0.00222
QSCSD1	3893	11	10	12	13	9	0.00251	0.00348	0.00695	0.00614	0.00966
GENHS28	43	0	0	9	1	5	0.00001	0.00002	0.00006	0.00262	0.01276
QSCTAP1	2325	15	10	10	0	12	0.00173	0.00191	0.00247	0.00246	0.01201
QBRANDY	2462	19	18	27	17	14	0.00299	0.00430	0.00576	0.00514	0.00510
HS52	14	0	0	7	2	6	0.00001	0.00003	0.00004	0.00266	0.00030
HS51	14	0	0	7	2	7	0.00001	0.00002	0.00004	0.00201	0.00025
CVXQP3_S	808	9	10	13	12	14	0.00052	0.00086	0.01384	0.00406	0.00455
CVXQP3_L	82481	-	10	-	18	24	-	21.84589	-	2.47151	106.68357
STADAT3	31997	16	54	28	26	22	0.01792	0.10851	0.07749	0.04597	0.07562
S268	40	5	10	15	12	15	0.00002	0.00005	0.00007	0.00239	0.00047
QSTANDAT	5030	24	20	11	18	18	0.00456	0.00757	0.03674	0.00550	0.05588
EXDATA	1137750	16	21	19	11	13	4.14068	5.91543	41.02600	0.93205	0.88089
QGROW7	3550	19	21	16	17	21	0.00292	0.00498	0.00441	0.00494	0.01086
DUAL4	3024	8	11	13	14	13	0.00105	0.00190	0.00228	0.00526	0.00334
QFORPLAN	5591	28	21	-	23	-	0.00554	0.00757	-	0.00635	-
QSCTAP3	12401	16	11	11	0	10	0.01539	0.01249	0.06928	0.00262	0.06155
QSHARE2B	828	19	15	24	18	21	0.00060	0.00086	0.00130	0.00313	0.00202
HS35MOD	12	9	11	11	9	13	0.00002	0.00004	0.00003	0.00203	0.00031
DUAL2	4796	8	10	13	13	8	0.00144	0.00237	0.00345	0.00495	0.00281
MOSARP2	4775	8	9	27	16	13	0.00384	0.00567	0.02281	0.00931	0.01393
QPTEST	10	5	7	9	11	10	0.00002	0.00003	0.00003	0.00204	0.00030
AUG3D	9219	0	0	13	2	8	0.00148	0.00230	0.03595	0.00389	0.03126
QSIERRA	11557	22	26	38	18	30	0.01250	0.02299	0.04104	0.01296	0.05868
LISWET5	40002	8	9	-	13	24	0.01327	0.03012	-	0.05090	0.20424
LISWET3	40002	9	16	-	13	20	0.01723	0.05153	-	0.04864	0.15546
LISWET12	40002	-	-	-	-	51	-	-	-	-	0.61876
CVXQP3_M	8231	23	11	-	14	24	0.09018	0.05447	-	0.08596	0.55480
HS118	108	11	11	9	12	10	0.00008	0.00012	0.00010	0.00250	0.00051
QSHIP12S	28344	18	15	32	17	19	0.02393	0.02954	7.88479	0.01928	0.33531
DUALC1	1998	23	11	29	12	14	0.00088	0.00079	0.00184	0.00230	0.00084
CVXQP1_L	74982	-	10	-	13	34	-	20.36505	-	1.39803	127.34169
LISWET7	40002	11	-	-	58	46	0.01622	-	-	0.23969	0.53076
CONT-100	79596	11	9	22	12	20	0.27128	0.33981	0.67966	0.08552	0.35265
QSCFXM3	10369	31	32	35	29	43	0.01834	0.02726	0.06242	0.01993	0.10596
DPKLO1	1652	0	0	13	2	5	0.00016	0.00025	0.00173	0.00276	0.00111
K SIP	18871	14	14	21	17	11	0.00483	0.00848	0.00887	0.01248	0.00428
DUALC8	4076	14	9	27	10	17	0.00104	0.00143	0.00446	0.00241	0.00098
QSHIP08L	52050	14	15	28	19	17	0.09698	0.12749	11.94011	0.04549	0.47082
AUG3DQP	13092	11	11	31	18	17	0.01532	0.01932	0.08252	0.01974	0.10652
AUG2DQP	80000	11	13	-	19	-	0.07272	0.13726	-	0.05924	-
HS76	20	6	6	10	9	7	0.00002	0.00004	0.00004	0.00232	0.00054
QSCTAP2	9371	14	10	10	0	9	0.00771	0.00859	0.02937	0.00252	0.06080
BOYD1	745507	37	-	-	30	-	0.93186	-	-	0.31081	-
QCAPRI	3147	40	31	-	29	21	0.00544	0.00852	-	0.00698	0.01174
CONT-300	653093	12	10	-	12	-	13.65596	15.11321	-	0.84948	-

**Table 9: Iterations and solver runtimes for SuiteSparse problems**

Problem	Size	Iterations					Solver Runtime (s)				
		QOCO	CLARABEL	ECOS	GUROBI	MOSEK	QOCO	CLARABEL	ECOS	GUROBI	MOSEK
NYPA_MARAGAL_6_HUBER	686479	6	6	13	-	65	18.85022	26.29638	47.81607	-	30.89082
NYPA_MARAGAL_6_LASSO	620812	7	7	-	13	13	20.08794	27.58199	-	1.69537	3.33290
PEREYRA_LANDMARK_HUBER	1650512	8	8	16	9	11	0.75585	1.39359	3.07564	1.48089	4.40789
PEREYRA_LANDMARK_LASSO	1301568	7	7	-	12	24	0.50615	0.80743	-	1.37256	0.99517
NYPA_MARAGAL_1_HUBER	458	5	5	10	7	8	0.00012	0.00018	0.00045	0.00237	0.00102
NYPA_MARAGAL_1_LASSO	354	8	8	12	10	10	0.00011	0.00018	0.00020	0.00217	0.00051
HB_ASH608_HUBER	5472	8	8	14	11	12	0.00192	0.00315	0.00993	0.00690	0.01230
HB_ASH608_LASSO	3184	7	7	21	9	11	0.00102	0.00163	0.00458	0.00759	0.00376
ANSYS_DELOR64K_HUBER	1105173	6	7	7	0	3	1.30160	2.34456	2.86475	0.41221	1.00721
ANSYS_DELOR64K_LASSO	7922958	7	7	-	17	20	6.12221	9.98326	-	4.06191	13.62744
HB_ASH85_HUBER	1118	5	5	6	6	3	0.00038	0.00057	0.00084	0.00264	0.00222
HB_ASH85_LASSO	1033	7	7	16	9	10	0.00042	0.00061	0.00120	0.00295	0.00153
NYPA_MARAGAL_3_HUBER	30221	6	7	12	9	10	0.02718	0.03827	0.07005	0.04743	0.12791
NYPA_MARAGAL_3_LASSO	25211	7	6	15	8	10	0.03499	0.03233	0.06423	0.04115	0.05463
HB_WELL1033_HUBER	11963	6	6	11	7	9	0.00307	0.00518	0.01379	0.01955	0.02565
HB_WELL1033_LASSO	8078	6	13	-	8	27	0.00185	0.00573	-	0.00885	0.01698
NYPA_MARAGAL_7_HUBER	1528452	6	9	37	-	-	36.98824	55.93122	236.53158	-	-
NYPA_MARAGAL_7_LASSO	1400483	7	6	-	13	11	38.76392	41.96500	-	4.47169	5.46443
HB_JLLC1033_HUBER	11950	6	6	11	6	9	0.00307	0.00676	0.01472	0.01855	0.02622
HB_JLLC1033_LASSO	8065	6	19	32	9	43	0.00231	0.00883	0.01289	0.00981	0.03614
NYPA_MARAGAL_4_HUBER	40467	6	6	12	8	55	0.06713	0.08642	0.14732	0.05997	0.74862
NYPA_MARAGAL_4_LASSO	34783	6	6	17	7	9	0.06302	0.07162	0.17837	0.05356	0.07373
HB_ASH219_HUBER	1971	7	7	12	10	11	0.00064	0.00105	0.00312	0.00369	0.00461
HB_ASH219_LASSO	1216	8	8	18	9	11	0.00043	0.00068	0.00142	0.00359	0.00127
NYPA_MARAGAL_2_HUBER	8242	6	7	12	9	11	0.00304	0.00456	0.01130	0.01764	0.01897
NYPA_MARAGAL_2_LASSO	6867	6	6	14	8	11	0.00243	0.00328	0.00613	0.01660	0.00786
HB_ASH331_HUBER	2979	8	8	12	10	12	0.00104	0.00169	0.00456	0.00451	0.00679
HB_ASH331_LASSO	1740	7	7	20	10	10	0.00055	0.00089	0.00226	0.00467	0.00174
HB_JLLC1850_HUBER	21586	6	6	14	8	9	0.00669	0.01189	0.03405	0.02046	0.05253
HB_JLLC1850_LASSO	15184	6	13	-	7	42	0.00443	0.01323	-	0.01773	0.07218
HB_ASH958_HUBER	8622	8	8	14	10	12	0.00302	0.00499	0.01556	0.01024	0.01923
HB_ASH958_LASSO	5000	7	7	23	9	12	0.00154	0.00250	0.00853	0.01077	0.00546
ANSYS_DELOR295K_HUBER	4471461	7	7	8	-	4	3.12931	5.19758	10.03436	-	3.17687
ANSYS_DELOR295K_LASSO	10288503	7	7	-	12	33	7.50801	12.16215	-	12.36872	24.15621
ANSYS_DELOR338K_HUBER	6614251	7	7	-	0	4	4.05127	6.12029	-	0.92594	3.42096
ANSYS_DELOR338K_LASSO	8446303	7	6	-	15	23	5.27505	7.54937	-	13.85264	13.07142
NYPA_MARAGAL_8_HUBER	1540899	6	6	13	-	16	22.18652	25.05274	45.98680	-	10.81397
NYPA_MARAGAL_8_LASSO	1675147	7	7	-	12	14	24.59839	29.16281	-	5.67463	7.79855
HB_ABB313_HUBER	3748	8	8	14	11	12	0.00124	0.00205	0.00550	0.00456	0.00744
HB_ABB313_LASSO	2887	6	6	22	8	11	0.00085	0.00129	0.00467	0.00434	0.00264
NYPA_MARAGAL_5_HUBER	125669	6	7	14	14	82	0.47005	0.60018	1.05444	0.17352	3.24096
NYPA_MARAGAL_5_LASSO	115679	7	7	22	14	11	0.51054	0.54681	1.35915	0.19943	0.27376
HB_ASH292_HUBER	4252	6	6	13	8	8	0.00171	0.00263	0.00664	0.00551	0.01193
HB_ASH292_LASSO	3960	8	8	24	9	15	0.00195	0.00273	0.00825	0.00678	0.00735
HB_WELL1850_HUBER	21705	6	6	14	7	9	0.00622	0.00968	0.03297	0.01690	0.04740
HB_WELL1850_LASSO	15303	6	12	-	8	40	0.00432	0.01136	-	0.01770	0.07075