An Operator Splitting Method for Large-Scale CVaR-Constrained Quadratic Programs

Eric Luxenberg*[1], David Pérez-Piñeiro*[1,2], Steven Diamond[1], and Stephen Boyd[1]

[1]Department of Electrical Engineering, Stanford University
[2]Department of Chemical Engineering, Norwegian University of Science and Technology

March 7, 2026

Abstract

We introduce a fast and scalable method for solving quadratic programs with conditional value-at-risk (CVaR) constraints. While these problems can be formulated as standard quadratic programs, the number of variables and constraints grows linearly with the number of scenarios, making general-purpose solvers impractical for large-scale problems. Our method combines operator splitting with a specialized $O(m \log m)$ algorithm for projecting onto CVaR constraints, where m is the number of scenarios. The method alternates between solving a linear system and performing parallel projections: onto CVaR constraints using our specialized algorithm and onto box constraints with a closed-form solution. Numerical examples from several application domains demonstrate that our method outperforms general-purpose solvers by several orders of magnitude on problems with up to millions of scenarios. Our method is implemented in an open-source package called CVQP.

*Equal contribution.

Contents

1 Introduction **3**
- 1.1 Conditional value-at-risk (CVaR) . . . 3
- 1.2 CVaR-constrained quadratic programs . . . 4
- 1.3 Related work . . . 5
- 1.4 Outline . . . 6

2 Solution via ADMM **6**
- 2.1 ADMM . . . 6
- 2.2 Evaluating the updates . . . 8
- 2.3 Summary . . . 8

3 CVaR projection preliminaries **9**
- 3.1 Sorted projection . . . 9
- 3.2 Optimality conditions . . . 10
- 3.3 Algorithm motivation . . . 11

4 CVaR projection algorithm **11**
- 4.1 Decrease step . . . 12
- 4.2 Complexity . . . 14
- 4.3 Correctness . . . 14

5 Numerical experiments **15**
- 5.1 Experimental setup . . . 15
- 5.2 CVaR projection . . . 15
- 5.3 Portfolio optimization . . . 16
- 5.4 Quantile regression . . . 18

6 Extensions and variations **19**

1 Introduction

Many applications in science and engineering require controlling the risk of extreme outcomes. The *conditional value-at-risk* (CVaR) is the expected value of losses exceeding a given quantile. Unlike variance, it penalizes only downside risk; unlike value-at-risk, it is coherent and convex. Convexity means that optimization problems involving CVaR can be solved exactly and efficiently.

Many practical applications, from portfolio optimization to quantile regression, can be formulated as quadratic programs with CVaR constraints. While these problems are convex and can be reformulated as standard quadratic programs, the number of variables and constraints grows linearly with the number of scenarios. For problems with many scenarios, general-purpose solvers become prohibitively slow or fail entirely. To address this challenge, we develop a fast and scalable method for solving quadratic programs with CVaR constraints.

We present two main contributions. First, we develop an $O(m \log m)$ algorithm for projecting onto CVaR constraints, where m is the number of scenarios. Building on this algorithm, our second contribution is an operator splitting method for solving large-scale CVaR-constrained quadratic programs. The method alternates between solving a linear system and performing parallel projections: one onto the CVaR constraints using our specialized algorithm and another onto box constraints with a simple closed-form solution. Numerical examples from several application domains demonstrate that our method outperforms general-purpose solvers by several orders of magnitude on problems with up to millions of scenarios.

1.1 Conditional value-at-risk (CVaR)

The conditional value-at-risk (CVaR) at level β is a risk measure that captures the expected value over the worst $(1-\beta)$ fraction of outcomes of a real-valued random variable. For a random variable X representing losses (where larger values are worse), we first define the value-at-risk (VaR) at level β as

$$\psi_\beta(X) = \inf\{x \mid \mathbb{P}(X \leq x) \geq \beta\}.$$

CVaR at level β is defined as the expected value of all losses exceeding VaR at level β,

$$\phi_\beta(X) = \mathbf{E}[X \mid X \geq \psi_\beta(X)].$$

At the same level β, CVaR provides an upper bound on VaR.

Sample CVaR. For a finite set of samples $z_1, \ldots, z_m \in \mathbf{R}$ representing the distribution of losses, Rockafellar and Uryasev [6] showed that the CVaR at level β, denoted $\phi_\beta : \mathbf{R}^m \to \mathbf{R}$, can be computed as

$$\phi_\beta(z) = \inf_{\alpha \in \mathbf{R}} \left\{ \alpha + \frac{1}{(1-\beta)m} \sum_{i=1}^{m} (z_i - \alpha)_+ \right\},$$

where $(z-\alpha)_+ = \max\{z - \alpha, 0\}$ is the positive part of $z - \alpha$. A CVaR constraint at level β can be represented by a set of linear inequality constraints,

$$\{z \mid \phi_\beta(z) \leq \kappa\} = \left\{ z \;\middle|\; \begin{array}{l} \alpha + \frac{1}{(1-\beta)m} \sum_{i=1}^{m} y_i \leq \kappa, \\ z_i - \alpha \leq y_i,\ 0 \leq y_i, \quad i = 1, \ldots, m \end{array} \right\}.$$

1.2 CVaR-constrained quadratic programs

This work presents a customized solver for the *CVaR-constrained quadratic program* (CVQP),

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & \phi_\beta(Ax) \leq \kappa, \quad l \leq Bx \leq u, \end{array} \tag{1}$$

where $x \in \mathbf{R}^n$ is the decision variable. The objective function is defined by a positive semidefinite matrix $P \in \mathbf{S}_+^n$ and a vector $q \in \mathbf{R}^n$. The CVaR constraint is defined by quantile level $\beta \in (0, 1)$, threshold $\kappa \in \mathbf{R}$, and matrix $A \in \mathbf{R}^{m \times n}$. Additional linear inequality constraints are defined by a matrix $B \in \mathbf{R}^{p \times n}$, and vectors $l \in (\mathbf{R} \cup \{-\infty\})^p$, $u \in (\mathbf{R} \cup \{\infty\})^p$. Since P is positive semidefinite and the constraints are convex, the CVQP is a convex optimization problem. We focus on the case where $p \ll m$, *i.e.*, the number of additional constraints is much smaller than the number of scenarios.

We assume the CVQP problem (1) is feasible and that P, A, and B have zero common nullspace, which ensures that the problem is bounded. This condition also implies that the matrix $M = P + \rho(A^T A + B^T B)$ is positive definite for any $\rho > 0$, which is needed for the ADMM linear system solve described in §2. In practice the condition holds when P is positive definite or when A has full column rank.

CVaR terms in the objective. The CVQP formulation extends to problems with CVaR terms in the objective. Consider the problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x + \phi_\beta(Ax) \\ \text{subject to} & l \leq Bx \leq u. \end{array}$$

Using the translation-equivariance property of CVaR (i.e., $\phi_\beta(X + c) = \phi_\beta(X) + c$), we introduce an auxiliary variable t and reformulate as

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x + t \\ \text{subject to} & \phi_\beta(Ax - t) \leq 0, \quad l \leq Bx \leq u. \end{array}$$

Defining the decision variable as the pair (x, t), this is an instance of the standard CVQP form (1).

Quadratic program solvers. By using the linear inequality representation of the CVaR constraint, the CVQP can be reformulated as a standard quadratic program and solved with any QP solver. However, when the number of scenarios m is large, this approach becomes prohibitively slow.

Applications. CVaR-constrained quadratic programs arise in many domains. In portfolio optimization, x represents portfolio weights and Ax gives portfolio losses across m return scenarios; the CVaR constraint limits the expected loss in the worst $(1 - \beta)$ fraction of scenarios, while the quadratic objective captures risk-adjusted returns [7]. In supply chain planning, x includes order quantities and routing decisions, with scenarios modeling demand uncertainty and supplier disruptions; CVaR constraints bound extreme total cost realizations [10, 12]. Similar formulations appear in network flow problems with stochastic capacities [14], facility location and disaster response logistics [13], energy systems with uncertain generation and prices [9], and radiation treatment planning, where CVaR constraints replace intractable dose-volume constraints [8]. In control and statistical learning, CVaR provides tractable convex approximations to chance constraints [15, 17]. For a comprehensive review, see [16].

1.3 Related work

CVaR constraints were introduced by Rockafellar and Uryasev [6] and are now standard in risk-constrained optimization [7]. Our method builds on operator splitting [11], which decomposes the problem into simpler subproblems: a linear system solve and projections.

The CVaR projection algorithm adds to a family of finite-termination projection methods, including the classical simplex projection [3] and isotonic regression [1]. Concurrently with the development of our CVaR projection algorithm, Roth and Cui released a preprint [18], later published as [20], containing two finite termination algorithms for CVaR projection attaining the same complexity. After corresponding

with the authors, we were unable to find any equivalence between their algorithms and ours.

Roth and Cui [19] also proposed a second-order computational framework for CVaR-constrained optimization, using a semismooth Newton-based augmented Lagrangian method. As a second-order method, it can achieve superlinear convergence near a solution, while ADMM is a first-order method with an $O(1/k)$ convergence rate. On the other hand, each ADMM iteration is cheap (a single backsolve and parallel projections after an initial factorization), and the method is simple to implement. In our experience, ADMM reaches moderate accuracy in few iterations, which is sufficient for most applications.

1.4 Outline

This paper is organized as follows. In §2, we present an ADMM-based solution method for the CVQP. §3 establishes theoretical foundations for projecting onto CVaR constraints, which we use to develop our $O(m \log m)$ projection algorithm in §4. In §5, we present benchmark results comparing our method against state-of-the-art solvers on portfolio optimization and quantile regression problems with up to millions of scenarios. §6 discusses possible extensions to our method.

2 Solution via ADMM

2.1 ADMM

We solve the CVQP by reformulating the problem and applying the alternating direction method of multipliers (ADMM). By introducing auxiliary variables $z \in \mathbf{R}^m$ and $\tilde{z} \in \mathbf{R}^p$, we can rewrite problem (1) as

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x + I_{\mathcal{C}}(z) + I_{[l,u]}(\tilde{z}) \\ \text{subject to} & Ax = z, \quad Bx = \tilde{z}, \end{array}$$

where $\mathcal{C} = \{z \mid \phi_\beta(z) \leq \kappa\}$. The auxiliary variables z and $\tilde{z}$ decouple the CVaR constraint from the box constraints, so that each can be handled separately in the ADMM updates below. Here, $I_{\mathcal{C}}$ and $I_{[l,u]}$ are the indicator functions for the sets $\mathcal{C}$ and $\{\tilde{z} \mid l \leq \tilde{z} \leq u\}$, respectively, given by

$$I_{\mathcal{C}}(z) = \begin{cases} 0 & z \in \mathcal{C} \\ \infty & \text{otherwise} \end{cases}, \quad I_{[l,u]}(\tilde{z}) = \begin{cases} 0 & l \leq \tilde{z} \leq u \\ \infty & \text{otherwise} \end{cases}.$$

We use the scaled dual variables $u = (1/\rho)y$ and $\tilde{u} = (1/\rho)\tilde{y}$, where $\rho > 0$ is a hyperparameter and $y \in \mathbf{R}^m$ and $\tilde{y} \in \mathbf{R}^p$ are the vectors of dual variables associated with the equality constraints $Ax = z$ and $Bx = \tilde{z}$, respectively. We use ADMM with over-relaxation (see [4, 5] for analysis), with over-relaxation parameter $\alpha \in (0, 2)$.

For notational convenience we define the linear system parameters

$$M = P + \rho(A^TA + B^TB), \quad p^k = q - \rho A^T(z^k - u^k) - \rho B^T(\tilde{z}^k - \tilde{u}^k).$$

Our assumption on the common nullspace of P, A, and B implies that M is positive definite. The ADMM updates are

$$\begin{aligned}
x^{k+1} &:= \operatorname*{argmin}_x \left(\frac{1}{2}x^TMx + (p^k)^Tx\right) && (2)\\
z^{k+1/2} &:= \alpha Ax^{k+1} + (1-\alpha)z^k && (3)\\
\tilde{z}^{k+1/2} &:= \alpha Bx^{k+1} + (1-\alpha)\tilde{z}^k && (4)\\
z^{k+1} &:= \Pi_{\mathcal{C}}\left(z^{k+1/2} + u^k\right) && (5)\\
\tilde{z}^{k+1} &:= \Pi_{[l,u]}\left(\tilde{z}^{k+1/2} + \tilde{u}^k\right) && (6)\\
u^{k+1} &:= u^k + z^{k+1/2} - z^{k+1} && (7)\\
\tilde{u}^{k+1} &:= \tilde{u}^k + \tilde{z}^{k+1/2} - \tilde{z}^{k+1}. && (8)
\end{aligned}$$

The operator $\Pi_{\mathcal{C}}$ is the (Euclidean) projection onto $\mathcal{C}$ and $\Pi_{[l,u]}$ is the projection onto the set $[l, u] \subset \mathbf{R}^p$. The over-relaxation, dual updates, and box projection are all trivial operations; the linear system solve requires one cached factorization of M. The main computational bottleneck is the CVaR projection (5), which we address in §3–4.

It is well known that this ADMM algorithm converges [2, 4], with a worst-case $O(1/k)$ convergence rate, though linear convergence is commonly observed in practice for quadratic programs [11].

Dynamic penalty parameter. While ADMM converges for any fixed penalty parameter $\rho > 0$, the convergence rate can be improved by adaptively updating ρ. A simple and effective update scheme adjusts ρ based on the relative magnitudes of the primal and dual residuals. When the primal residual is μ times larger than the dual residual, we multiply ρ by a factor $\tau > 1$; when the dual residual is μ times larger than the primal residual, we divide ρ by τ. Since ρ appears in the matrix M, each update requires re-factorizing the linear system. To balance computational cost with convergence benefits, we perform these updates at fixed intervals of T iterations.

2.2 Evaluating the updates

The updates (3), (4), (7), and (8) are trivial to implement. The update (6) is also a very simple clipping operation:

$$\tilde{z}_i^{k+1} = \begin{cases} l_i & v_i < l_i \\ v_i & l_i \leq v_i \leq u_i \\ u_i & v_i > u_i \end{cases}$$

where $v = \tilde{z}^{k+1/2} + u^k$.

The update (2) can be expressed as $x^{k+1} = -M^{-1}p^k$, which requires the solution of a linear system of equations with positive definite coefficient matrix M. We factorize M once (for example, a sparse Cholesky or LDL^T factorization) so that subsequent solves require only the backsolve. This can substantially improve the efficiency of this step; when M is dense, for example, the first factorization costs $O(n^3)$ flops, while subsequent solves require only $O(n^2)$ flops. (When M is sparse the speedup from caching the factorization is smaller than n, but still very significant.)

That leaves only the CVaR projection (5), for which we develop an $O(m \log m)$ algorithm in the next two sections.

2.3 Summary

For convenience we summarize the complete CVQP method in Algorithm 1. The CVaR projection is the main computational bottleneck and uses the $O(m \log m)$ algorithm described in §4.

Algorithm 1 CVQP solver

1: Factorize $M = P + \rho(A^TA + B^TB)$.
2: **repeat**
3: Solve linear system (2).
4: Over-relax (3)–(4).
5: Project onto CVaR constraint (5); see §4.
6: Clip to box constraints (6).
7: Update dual variables (7)–(8).
8: Update ρ and re-factorize M if needed.
9: **until** primal and dual residuals below tolerance.

3 CVaR projection preliminaries

We now present preliminaries that motivate our efficient algorithm for evaluating the projection operator $\Pi_{\mathcal{C}}$, *i.e.*, for solving the problem

$$\begin{array}{ll} \text{minimize} & \|v - z\|_2^2 \\ \text{subject to} & \phi_\beta(z) \leq \kappa, \end{array} \tag{9}$$

with variable $z \in \mathbf{R}^m$.

The CVaR constraint can be expressed equivalently using the *sum of k largest components* function,

$$f_k(z) = \sum_{i=1}^{k} z_{[i]}, \tag{10}$$

where $z_{[1]} \geq z_{[2]} \geq \cdots \geq z_{[m]}$ are the components of z in nonincreasing order. This is a convex function that sums the k largest elements of a vector (the pointwise maximum of $\binom{m}{k}$ linear functions). Setting $k = (1-\beta)m$ and evaluating the infimum over α in the Rockafellar-Uryasev formula (the minimizer is $\alpha = z_{[k]}$), we obtain $\phi_\beta(z) = (1/k)f_k(z)$. The constraint $\phi_\beta(z) \leq \kappa$ is therefore equivalent to $f_k(z) \leq \kappa k$. We rewrite problem (9) as

$$\begin{array}{ll} \text{minimize} & \|v - z\|_2^2 \\ \text{subject to} & f_k(z) \leq d, \end{array} \tag{11}$$

where $d = \kappa k$. We assume $(1-\beta)m \in \mathbf{N}$ (and round up to the nearest integer if not).

We will use the formulation (11) to derive our CVaR projection algorithm. While this is a computationally tractable convex optimization problem, we seek closed form or computationally efficient ways to evaluate this projection operator. Since merely evaluating f_k with $k = (1-\beta)m$ has a complexity of $O(m \log m)$, we desire a similar complexity for the projection operator.

3.1 Sorted projection

Let $P \in \mathbf{R}^{m \times m}$ be a permutation matrix that sorts v in descending order, *i.e.*, $(Pv)_1 \geq (Pv)_2 \geq \cdots \geq (Pv)_m$. We will denote the sorted vector Pv as v'. Since for any $z \in \mathbf{R}^m$,

$$\|v - z\|_2^2 = \|Pv - Pz\|_2^2, \qquad f_k(z) = f_k(Pz),$$

the solution to the projection problem is given by

$$\begin{aligned} z^\star &= \operatorname*{argmin}_{f_k(z)\le d} \|v-z\|_2^2 \\ &= \operatorname*{argmin}_{f_k(Pz)\le d} \|Pv-Pz\|_2^2 \\ &= P^T\left(\operatorname*{argmin}_{f_k(u)\le d} \|v'-u\|_2^2\right). \end{aligned}$$

The last equality is the change of variable $u = Pz$. In words, we can sort v in descending order, project the sorted vector v', and unsort the result to obtain the projection of v onto the set $\{z \mid f_k(z) \le d\}$.

3.2 Optimality conditions

We characterize the optimality conditions of the projection problem; these conditions will be used in §4.3 to prove the correctness of the algorithm. Let $\mathcal{A} = \{a \in \{0,1\}^m \mid \mathbf{1}^T a = k\}$ be the set of all $\binom{m}{k}$ k-hot indicator vectors. Since $f_k(z) = \max_{a\in\mathcal{A}} a^T z$, the constraint $f_k(z) \le d$ is equivalent to $a^T z \le d$ for all $a \in \mathcal{A}$. We can therefore rewrite problem (11) in the equivalent form

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|v-z\|_2^2 \\ \text{subject to} & a_i^T z \le d, \quad i = 1,\ldots,\binom{m}{k}, \end{array}$$

where the subscript i enumerates the elements of $\mathcal{A}$.

The KKT conditions tell us a z, λ pair is optimal if and only if

$$v - z = \sum_{i=1}^{\binom{m}{k}} \lambda_i a_i,$$

and

$$a_i^T z \le d, \quad \lambda_i \ge 0, \quad \left(a_i^T z < d \implies \lambda_i = 0\right), \qquad i = 1,\ldots,\binom{m}{k}.$$

The optimality conditions motivate an algorithm that solves the KKT system. We will define $z = v - \Delta$, where Δ is the vector we remove from v to get the projection. The KKT conditions above imply that Δ is a nonnegative combination of the vectors a_i for which $a_i^T z = d$, *i.e.*, the indicators of the largest k entries

of z. (Note that there are potentially several such vectors in the presence of ties: consider the largest 2 entries in the vector $(2, 1, 1, 0)$.) Thus, if we can construct a vector Δ that is a nonnegative combination of largest-k indicator vectors, such that $f_k(v - \Delta) = d$, then we have found the projection $z = v - \Delta$.

3.3 Algorithm motivation

We describe at a high level the motivation behind our projection algorithm. We pre-process by sorting and work with the sorted vector v', since from §3.1 we know that sort-project-unsort is the same as projecting. We then proceed by incrementally constructing Δ until the sum of the k largest elements of $v' - \Delta$ is equal to d.

We want to begin decreasing elements of v until the sum of the k largest elements is equal to d. Intuitively, we should decrease the largest k elements without modifying the remaining elements, since they do not affect the sum. We begin by reducing the largest elements uniformly until either the constraint is satisfied, or until the kth largest element ties the next element. We reduce each of the k largest elements by the same amount because we are penalized by the sum of squared changes.

If reducing the largest k elements causes elements k and $k+1$ to tie, it is intuitive that the tied elements should be further reduced by the same amount; otherwise a reduction would be applied to an element not in the largest k. Since we also want to uniformly decrease the largest elements preceding the tie, we need to decide the ratio of these two reductions. Given this ratio, we reduce the untied and tied entries until either the constraint is satisfied or the next tie occurs.

We will need to determine this ratio at each step of the algorithm, since at a given step of the algorithm, the altered vector $v' - \Delta$ will be partitioned into three regions: first the largest $n_u \in \mathbf{N}$ *"untied"* elements, then the $n_t \in \mathbf{N}$ *"tied"* elements, and lastly the remaining unaltered elements:

$$v' - \Delta = \left(\underbrace{v'_1 \geq \cdots \geq v'_{n_u}}_{\text{untied}} \;\geq\; \underbrace{v'_{n_u+1} = \cdots = v'_{n_u+n_t}}_{\text{tied}} \;\geq\; \underbrace{v'_{n_u+n_t+1} \geq \cdots \geq v'_m}_{\text{unaltered}} \right).$$

The core insight of the algorithm is deriving the ratio of reducing the untied and tied elements, and is derived in §4.

4 CVaR projection algorithm

With the preliminaries in place, we now present our CVaR projection algorithm.

Pre- and post-processing. The CVaR projection algorithm is applied to the vector v sorted in descending order as described in §3.1. The complexity of sorting v and then unsorting the projection is $O(m \log m)$. The sorting and unsorting are in practice carried out with indices; the permutation matrix P is never explicitly formed.

Algorithm state. The algorithm's state is

- j: the iterate number,
- n_u: the number of untied entries,
- n_t: the number of tied entries,
- η: the decrease to the untied entries,
- S: the sum of the largest k entries,
- a_t: the value of the tied entries,
- a_u: the value of the last untied entry,
- a_e: the value of first unaltered entry.

The algorithm is initialized with $n_u = k$, $n_t = 0$, $\eta = 0$, $a_t = -\infty$, $S = \sum_{i=1}^{k} v'_i$, $a_u = v'_k$, and $a_e = v'_{k+1}$. The algorithm repeats the *decrease step* described in §4.1 until $S = d$, after which the projection of the sorted v',

$$z = \left(\underbrace{v'_1 - \eta, \ldots, v'_{n_u} - \eta}_{n_u}, \ \underbrace{a_t, \ldots, a_t}_{n_t}, \ v'_{n_u+n_t+1}, \ldots, v'_m \right)$$

is formed, unsorted, and returned.

4.1 Decrease step

Define $\delta(n_u, n_t, k) \in \mathbf{R}^m$ as

$$\delta(n_u, n_t, k) = \left(\underbrace{\frac{n_t}{k - n_u}, \ldots, \frac{n_t}{k - n_u}}_{n_u}, \ \underbrace{1, \ldots, 1}_{n_t}, \ \underbrace{0, \ldots, 0}_{m - n_u - n_t} \right),$$

for $n_u < k$, and

$$\delta(n_u, n_t, k) = \left(\underbrace{1, \ldots, 1}_{k}, \ \underbrace{0, \ldots, 0}_{m-k} \right)$$

when $n_u = k, n_t = 0$.

Note that

$$\delta(n_u, n_t, k) = \frac{1}{\binom{n_t - 1}{k - n_u - 1}} \sum_{\delta \in \mathcal{I}(n_u, n_t, k)} \delta, \tag{12}$$

where

$$\mathcal{I}(n_u, n_t, k) = \left\{ \delta \in \{0,1\}^m \;\middle|\; \begin{array}{ll} \delta_i = 1 & i \leq n_u \\ \delta_i = 0 & i > n_u + n_t \\ \sum_{i=1}^m \delta_i = k & \end{array} \right\}$$

is the set of k-hot indicator vectors whose first n_u entries are equal to 1 and all $1's$ occurring in the first $n_u + n_t$ entries.

Scaling the step. Let

$$z(s) = z - s\delta(n_u, n_t, k)$$

denote the vector z reduced by an s-multiple of $\delta(n_u, n_t, k)$. The scaling s is chosen as the largest $s > 0$ such that $z(s)$ is still sorted and the sum of its largest k entries is greater than or equal to d. We describe explicitly how to compute s using the algorithm state. The three cases below correspond to the scaling (1) causing a tie between the untied and tied blocks, (2) causing a tie between the tied and unaltered blocks, and (3) satisfying the sum constraint. We compute the scaling using the algorithm state, which allows us to avoid modifying z at each step at cost $O(m)$.

(1) We find the value $s_1 \in \mathbf{R}$ such that the value of the final untied entry is equal to the value of the first tied entry, *i.e.*, $z(s_1)_{n_u} = z(s_1)_{n_u+1}$. If n_u is 0, there are no untied entries, and $s_1 = \infty$. Otherwise, $s_1 = (a_t - a_u) \,/\, \left(1 - \frac{n_t}{k - n_u}\right)$.

(2) We find the value $s_2 \in \mathbf{R}$ such that the value of the last tied entry is equal to the value of the first unaltered entry, *i.e.*, $z(s_2)_{n_u+n_t} = z(s_2)_{n_u+n_t+1}$. If $n_u + n_t = m$, there are no unaltered entries, and $s_2 = \infty$. Otherwise, define a_p as $\frac{n_t}{k-n_u}$ if $n_t = 0$ and otherwise $a_p = 1$. Define a_f as a_u if $n_t = 0$ and otherwise $a_f = a_t$. Then, $s_2 = \frac{a_e - a_f}{-a_p}$.

(3) Let $s_3 \in \mathbf{R}_+$ be the scaling such that the sum of the first k entries of $z(s_3)$ is equal to d, *i.e.*, $s_3 = \frac{S-d}{w}$, where $w = \sum_i \delta(n_u, n_t, k)_i = n_u(\frac{n_t}{k-n_u}) + n_t$.

Let $s_0 = \min\{s_1, s_2, s_3\}$. The update is then

$$z \leftarrow z - \delta_j,$$

where $\delta_j = s_0\delta(n_u, n_t, k)$.

Updating the state. Instead of explicitly carrying out this vector subtraction at every step (which would have a complexity of $O(m)$ per iterate), the constant size algorithm state is updated to reflect the change in the vector. The updates are given by

$$\begin{array}{ll} j & \leftarrow j+1 \\ n_u & \leftarrow \max\{n_u-1,0\} \text{ if } s_0 = s_1 \text{ else } n_u \\ n_t & \leftarrow \min\{n_t+1, m\} \\ \eta & \leftarrow \eta + s_0 \frac{n_t}{k-n_u} \\ S & \leftarrow S - s_0\left(\frac{n_t}{k-n_u} n_u + (k-n_u)\right) \\ a_t & \leftarrow a_t - s_0 \text{ if } a_t > -\infty \text{ else } z_{k+1} \\ a_u & \leftarrow z_{n_u} - \eta \text{ if } n_u > 0 \text{ else } a_u \\ a_e & \leftarrow z_{n_u+n_t+1} \text{ if } n_u + n_t < m \text{ else } a_e \end{array}$$

4.2 Complexity

At each step, n_u either decreases by 1, or remains the same. At each step, n_t increases by 1. If the algorithm hasn't terminated after m steps, then there are m tied entries, thus the subsequent decrease step will reduce the sum to the desired value, and the algorithm will terminate. Each step of the algorithm has constant complexity, and thus the algorithm (with sorted input) has complexity $O(m)$. Therefore, including the sorting and unsorting, the total complexity of the algorithm is $O(m \log m)$.

4.3 Correctness

At termination, the algorithm returns $z = v' - \sum_t \delta_j$. Note that by (12), each δ_j is a nonnegative combination of largest-k indicator vectors. Thus, the algorithm returns z which satisfies

$$v' - z = \sum_{i=1}^{\binom{m}{k}} \lambda_i a_i, \qquad \lambda_i \geq 0, \quad i = 1, \ldots, \binom{m}{k}. \tag{13}$$

Since the decrease step maintains the monotonicity of v', all previous largest-k indicators are still valid indicators of the largest k entries of z. Thus, the termination criterion provides that

$$a_i^T z \leq d, \quad \left(a_i^T z < d \implies \lambda_i = 0\right), \qquad i = 1, \ldots, \binom{m}{k}. \tag{14}$$

However, (13) and (14) are exactly the KKT conditions for the projection problem described in §3.2. Therefore, the algorithm correctly computes the projection of v' onto the set $\{z \mid f_k(z) \leq d\}$.

5 Numerical experiments

We compare our CVaR projection algorithm and our CVQP solver with two general-purpose solvers, Mosek and Clarabel. We first benchmark the projection algorithm on synthetic instances, then benchmark the CVQP solver on portfolio optimization and quantile regression problems. Code is available at https://github.com/cvxgrp/cvqp.

5.1 Experimental setup

All experiments were run on a Google Cloud n1-highmem-32 instance (32 vCPUs, 208 GB RAM), with a two-hour time limit per solve. For our CVQP solver we use ADMM penalty parameter $\rho^{(0)} = 10^{-2}$, over-relaxation parameter $\alpha = 1.7$, and update ρ adaptively with parameters $\mu = 10$ and $\tau = 2$. We stop when primal and dual residuals satisfy absolute tolerance 10^{-4} and relative tolerance 10^{-3}. Mosek and Clarabel are called through CVXPY with default settings. When we tighten our tolerances to 10^{-6}, objective values agree with both solvers to about four significant figures.

5.2 CVaR projection

We generate random CVaR projection problems with vectors $v \in \mathbf{R}^m$ having entries uniformly distributed on $[0, 1]$, with $k = (1 - \beta)m$ and $\beta = 0.95$. For each v, we set the constraint threshold to $d = \eta f_k(v)$ in problem (11), where $\eta \in (0, 1)$ controls problem difficulty (larger η gives easier problems), and project v onto the convex set $\{z \mid f_k(z) \leq d\}$.

Results. Figure 1 shows solve times versus number of scenarios m for $\eta = 0.5$, averaged over 10 random instances. Our method solves these problems in 0.51 ms at $m = 10^4$ and 1.98 s at $m = 10^7$, more than two orders of magnitude faster than both Mosek and Clarabel across all tested sizes. Mosek does not solve instances with $m \geq 3 \times 10^6$ within the time limit.

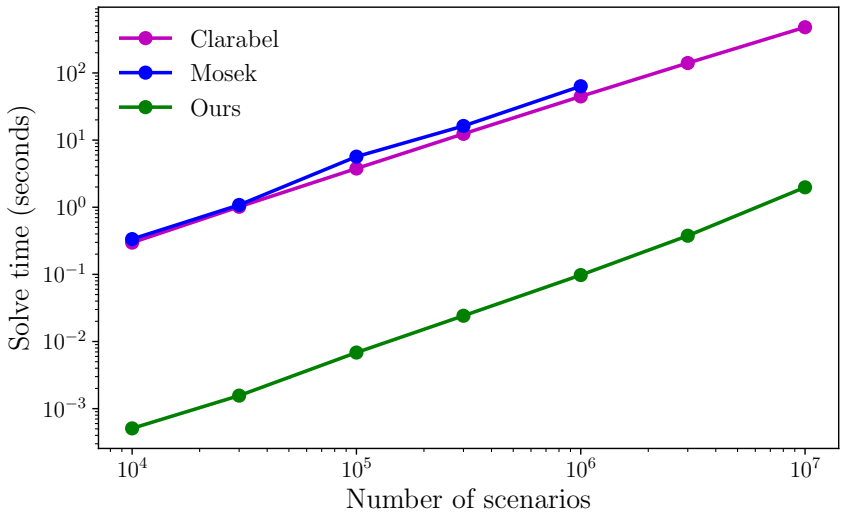

Figure 1: CVaR projection solve times, with $\eta = 0.5$.

5.3 Portfolio optimization

We consider a portfolio optimization problem with n assets and m return scenarios. The matrix $R \in \mathbf{R}^{m\times n}$ contains asset returns, where R_{ij} is the return of asset j in scenario i. The mean return vector $\mu \in \mathbf{R}^n$ and covariance matrix Σ are given by

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} R_{ij}, \quad \Sigma = \frac{1}{m}\sum_{i=1}^{m}(R_i - \mu)(R_i - \mu)^T.$$

The portfolio optimization problem is

$$\begin{array}{ll} \text{minimize} & -\mu^T x + \frac{\gamma}{2} x^T \Sigma x \\ \text{subject to} & \mathbf{1}^T x = 1 \\ & x \geq 0 \\ & \phi_\beta(-Rx) \leq \kappa, \end{array}$$

with variable $x \in \mathbf{R}^n$ (the portfolio weights), risk aversion parameter $\gamma > 0$, and a CVaR constraint bounding the expected loss in the worst $(1-\beta)$ fraction of scenarios.

CVQP form. The problem maps to CVQP form as

$$P = \gamma\Sigma, \qquad q = -\mu, \quad A = -R,$$

$$B = \begin{bmatrix} \mathbf{1}^T \\ I \end{bmatrix}, \quad l = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad u = \begin{bmatrix} 1 \\ \infty \end{bmatrix}.$$

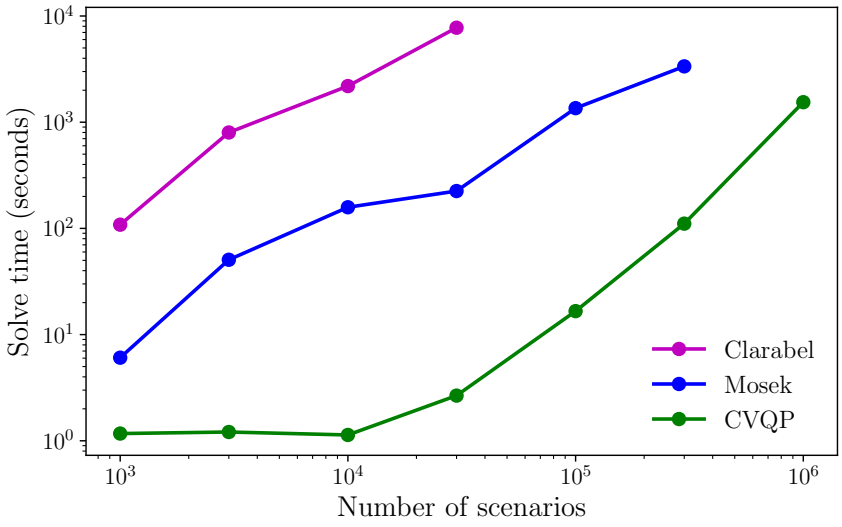

Figure 2: Portfolio optimization solve times, with $n = 2000$ assets.

Problem instances. We generate return scenarios using a two-component Gaussian mixture model

$$R_i \sim \omega\mathcal{N}(\nu\mathbf{1}, I) + (1-\omega)\mathcal{N}(-\nu\mathbf{1}, \sigma^2 I), \quad i = 1, \ldots, m,$$

where ω is the probability of normal market conditions, ν is the mean return per asset (positive in normal conditions, negative in stress periods), and $\sigma > 1$ scales the volatility during stress periods. We use parameters

$$\begin{aligned} \omega &= 0.8, \quad \nu = 0.2, \quad \gamma = 1, \\ \sigma &= 2, \quad \beta = 0.95, \quad \kappa = 0.3. \end{aligned}$$

Results. Figure 2 shows solve times versus number of scenarios m for problems with $n = 2000$ assets, averaged over 3 random instances. Our method is faster than both general-purpose solvers by one to three orders of magnitude. Mosek does not solve instances with $m \geq 10^6$ within the time limit, and Clarabel does not solve instances with $m \geq 10^5$. Our method solves instances with up to one million scenarios in under 26 minutes.

5.4 Quantile regression

Given m data points with features $u_i \in \mathbf{R}^n$ and responses $y_i \in \mathbf{R}$, the τ-quantile regression problem is

$$\text{minimize} \quad \frac{1}{m}\sum_{i=1}^{m} \rho_\tau(y_i - x^T u_i - x_0),$$

with variables $x \in \mathbf{R}^n$ and $x_0 \in \mathbf{R}$, where $\rho_\tau(z) = \tau(z)_+ + (1-\tau)(z)_-$ is the tilted ℓ_1 penalty and $\tau \in (0,1)$ is the quantile level. The asymmetric penalty weighs underpredictions $\tau/(1-\tau)$ times more than overpredictions, so the optimal prediction is the conditional τ-quantile of y given u, rather than the conditional mean.

CVaR reformulation. Defining the feature matrix $U \in \mathbf{R}^{m\times n}$ with rows u_i^T, the response vector $y \in \mathbf{R}^m$, and $\bar{u} = \frac{1}{m}\sum_{i=1}^m u_i$, we use the identity $\rho_\tau(z) = (1-\tau)(-z) + (z)_+$ to write the objective as

$$(1-\tau)\left(x^T\bar{u} + x_0 - \bar{y}\right) + \frac{1}{m}\sum_{i=1}^{m}(y_i - u_i^T x - x_0)_+,$$

where $\bar{y} = \frac{1}{m}\sum_{i=1}^m y_i$. The terms involving x_0 can be written as

$$(1-\tau)\left[x_0 + \frac{1}{(1-\tau)m}\sum_{i=1}^{m}(y_i - u_i^T x - x_0)_+\right],$$

which is $(1-\tau)$ times the sample CVaR formula evaluated at $\alpha = x_0$. Minimizing over x_0 yields the infimum in the CVaR definition. Since $\bar{y}$ is constant and $(1-\tau) > 0$, the problem reduces to

$$\text{minimize} \quad x^T\bar{u} + \phi_\tau(-Ux + y),$$

with variable $x \in \mathbf{R}^n$. Introducing an epigraph variable t for the CVaR term gives

$$\begin{array}{ll} \text{minimize} & x^T\bar{u} + t \\ \text{subject to} & \phi_\tau(-Ux+y) \le t. \end{array}$$

CVQP form. We introduce an auxiliary variable s with the constraint $s = 1$, and define $\tilde{x} = (x, t, s) \in \mathbf{R}^{n+2}$. The CVQP parameters, with CVaR level $\beta = \tau$, are

$$P = 0, \qquad q = \begin{bmatrix}\bar{u}\\ 1 \\ 0\end{bmatrix}, \quad A = \begin{bmatrix}-U & -\mathbf{1} & y\end{bmatrix},$$

$$B = \begin{bmatrix}0^T & 0 & 1\end{bmatrix}, \quad l = 1, \qquad u = 1.$$

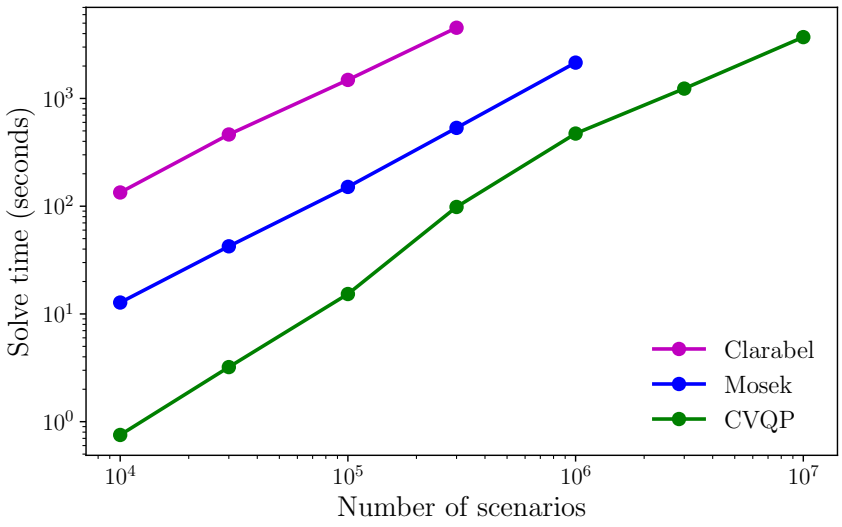

Figure 3: Quantile regression solve times, with $n = 500$ features.

Problem instances. We generate features $u_i \in \mathbf{R}^n$ with independent standard normal entries, a vector with entries $\beta_j \sim \mathcal{N}(0, 1/(1+j))$, and responses $y_i = u_i^T\beta + \epsilon_i$, where ϵ_i follows a t-distribution with 5 degrees of freedom scaled by 0.1. We use quantile level $\tau = 0.9$.

Results. Figure 3 shows solve times versus number of scenarios m for quantile regression problems with $n = 500$ features, averaged over 3 random instances. Our method is faster than both Mosek and Clarabel across all tested sizes, by a factor ranging from about 5 to over 100. Mosek does not solve instances with $m > 10^6$ within the time limit, and Clarabel does not solve instances with $m \geq 3 \times 10^5$. Our method scales to $m = 10^7$ scenarios.

6 Extensions and variations

Our method extends to more general problems and admits several practical variations.

More general constraints on $\tilde{z}$. When projection onto a set $\mathcal{C}$ is efficient (via a closed-form solution or a fast algorithm), we can directly handle constraints of the form $\tilde{z} \in \mathcal{C}$ in place of the box constraints $l \leq \tilde{z} \leq u$.

Extensions to the x-update. The method extends to additional constraints on x or nonquadratic objective functions. The x-update then becomes

$$\begin{array}{ll} \text{minimize} & f(x) + (\rho/2)\|Ax - z^k + u^k\|_2^2 + (\rho/2)\|Bx - \tilde{z}^k + \tilde{u}^k\|_2^2 \\ \text{subject to} & Gx \in \mathcal{D}, \end{array}$$

where f is any convex function (not just quadratic), $G \in \mathbf{R}^{p' \times n}$, and $\mathcal{D} \subseteq \mathbf{R}^{p'}$ is a convex set. Using a standard solver with warm-starting, the per-iteration cost remains manageable, though typically higher than solving the original linear system.

Equilibration. Equilibration as a preprocessing step often improves ADMM convergence, particularly for problems with poorly scaled data.

Warm starting. Two effective strategies for choosing the initial point are (1) solving with a reduced set of scenarios and using that solution to initialize the full problem, and (2) using a quadratic approximation of the CVaR constraint to compute an initial point. Similarly, the sorting step in the CVaR projection can be warm-started using the sorted order from the previous ADMM iteration. Since the projection input changes only slightly between iterations, the sorted order is nearly preserved, and an adaptive sorting algorithm can exploit this to reduce the per-iteration cost in practice.

Acknowledgments

David Pérez-Piñeiro received support from the Research Council of Norway and the HighEFF Research Centre (Centre for Environmentally Friendly Energy Research, Grant No. 257632).

References

[1] R. Barlow and H. Brunk, "The isotonic regression problem and its dual," *Journal of the American Statistical Association*, vol. 67, no. 337, pp. 140–147, 1972.

[2] D. Gabay, "Applications of the method of multipliers to variational inequalities," in *Augmented Lagrangian methods: applications to the numerical solution of boundary-value problems*, ser. Studies in Mathematics and Its Applications, M. Fortin and R. Glowinski, Eds., vol. 15, Elsevier, 1983, pp. 299–331.

[3] E. Gafni and D. Bertsekas, "Two-metric projection methods for constrained optimization," *SIAM Journal on Control and Optimization*, vol. 22, no. 6, pp. 936–964, 1984.

[4] J. Eckstein and D. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 55, pp. 293–318, 1992.

[5] J. Eckstein, "Parallel alternating direction multiplier decomposition of convex programs," *Journal of Optimization Theory and Applications*, vol. 80, no. 1, pp. 39–62, 1994.

[6] R. T. Rockafellar, S. Uryasev, *et al.*, "Optimization of conditional value-at-risk," *Journal of Risk*, vol. 2, pp. 21–42, 2000.

[7] P. Krokhmal, J. Palmquist, and S. Uryasev, "Portfolio optimization with conditional value-at-risk objective and constraints," *Journal of Risk*, vol. 4, pp. 43–68, 2002.

[8] H. E. Romeijn, R. K. Ahuja, J. F. Dempsey, A. Kumar, and J. G. Li, "A novel linear programming approach to fluence map optimization for intensity modulated radiation therapy treatment planning," *Physics in Medicine & Biology*, vol. 48, no. 21, pp. 3521–3542, 2003.

[9] J. Doege, P. Schiltknecht, and H.-J. Lüthi, "Risk management of power portfolios and valuation of flexibility," *OR Spectrum*, vol. 28, no. 2, pp. 267–287, 2006.

[10] J.-y. Gotoh and Y. Takano, "Newsvendor solutions via conditional value-at-risk minimization," *European Journal of Operational Research*, vol. 179, no. 1, pp. 80–96, 2007.

[11] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[12] T. Sawik, "Selection of supply portfolio under disruption risks," *Omega*, vol. 39, no. 2, pp. 194–208, 2011.

[13] N. Noyan, "Risk-averse two-stage stochastic programming with an application to disaster management," *Computers & Operations Research*, vol. 39, no. 3, pp. 541–559, 2012.

[14] A. Sorokin, V. Boginski, A. Nahapetyan, and P. M. Pardalos, "Computational risk management techniques for fixed charge network flow problems with uncertain arc failures," *Journal of Combinatorial Optimization*, vol. 25, no. 1, pp. 99–122, 2013.

[15] B. P. Van Parys, D. Kuhn, P. J. Goulart, and M. Morari, "Distributionally robust control of constrained stochastic systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 2, pp. 430–442, 2015.

[16] C. Filippi, G. Guastaroba, and M. G. Speranza, "Conditional value-at-risk beyond finance: A survey," *International Transactions in Operational Research*, vol. 27, no. 3, pp. 1277–1319, 2020.

[17] Y. Laguel, K. Pillutla, J. Malick, and Z. Harchaoui, "Superquantiles at work: Machine learning applications and efficient subgradient computation," *Set-Valued and Variational Analysis*, vol. 29, no. 4, pp. 967–996, 2021.

[18] J. Roth and Y. Cui, *On O(n) algorithms for projection onto the top-k-sum sublevel set*, 2023.

[19] J. Roth and Y. Cui, *Fast computation of superquantile-constrained optimization through implicit scenario reduction*, 2024.

[20] J. Roth and Y. Cui, "On O(n) algorithms for projection onto the top-k-sum sublevel set," *Mathematical Programming Computation*, pp. 1–42, 2025.