Numerical Optimization Project: Analyzing ADMM and IRWA for Quadratic Programming

Hu Xinyue Zhuang Sihan Gao Shenghan

Abstract

Quadratic Programming (QP) serves as a fundamental optimization problem in various domains, including finance, machine learning, and engineering. This paper investigates two QP solution methodologies: the Iterative Reweighted Algorithm (IRWA) and the Alternating Direction Method of Multipliers (ADMM). We address key challenges in these algorithms, including ill-conditioning, inconsistent penalty rates, and scaling issues, by incorporating ϵ^{-1}-Threshold and scaling techniques. Extensive evaluations on generated QP instances reveal the complementary strengths of IRWA and ADMM, with IRWA demonstrating robustness in complex scenarios and ADMM excelling in computational efficiency. Our results underscore the importance of penalty synchronization and numerical stability in improving convergence rates and solution accuracy. The proposed enhancements make these methods more effective for a wide range of QP applications, bridging the gap between theory and real-world problem-solving.

1 Introduction

1.1 Background

1.1.1 The Importance of Optimization Problems

Optimization problems are integral to scientific and engineering fields, underpinning solutions in machine learning, economic modeling, and industrial design. These problems often translate real-world challenges into mathematical formulations, enabling data-driven decision-making and system optimization.

Quadratic Programming (QP), a fundamental optimization class, features a quadratic objective function and linear constraints, making it versatile and widely applicable. Key applications include:

- Finance: Portfolio optimization minimizes risks under budget constraints to determine optimal investments.
- Machine Learning: Support Vector Machines (SVMs) use QP to maximize classification margins.
- Engineering: Resource scheduling and path planning often rely on QP for objective modeling.

These use cases underscore the practical and theoretical importance of QP in optimization research.

Preprint. Under review.

1.1.2 Mathematical Characteristics of Quadratic Programming

A standard QP formulation is:

$$\min_{x} \phi(x) = g^\top x + \frac{1}{2}x^\top Hx \quad \text{s.t.} Ax + b \in C$$

where:

- $x \in \mathbb{R}^n$: decision variable.
- $g \in \mathbb{R}^n$: linear term coefficients.
- $H \in \mathbb{R}^{n\times n}$: symmetric quadratic term matrix.
- $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$: constraint coefficients.
- C: feasible region (equality and inequality constraints).

Convex vs. Non-Convex Problems:

- **Convex:** Positive semi-definite H ensures global solutions and efficient algorithms.
- **Non-Convex:** Negative eigenvalues in H introduce local minima, complicating global optimization.

This structure defines QP's theoretical challenges and its broad practical relevance.

1.2 Research Objectives

This project aims to explore Quadratic Programming by implementing and evaluating two optimization algorithms: Iterative Reweighting Algorithm (IRWA) and Alternating Direction Augmented Lagrangian (ADAL). The focus lies on their design, analysis, and performance in solving QP problems.

1.2.1 Literature Review

The initial phase involves a comprehensive review of existing QP methodologies. By examining state-of-the-art solvers, their benchmarks, and relevant theoretical insights, this research seeks to identify effective approaches and uncover potential areas for improvement. This review will inform the design and implementation of new or modified algorithms.

1.2.2 Algorithm Implementation

The implementation involves translating the theoretical principles of IRWA and ADAL into practical algorithms:

- **IRWA:** Focuses on iterative reAnement by reweighting penalty terms to approximate the solution of exact penalty subproblems.
- **ADAL:** Utilizes an augmented Lagrangian framework to decouple constraints, solving subproblems in alternating directions while updatingmultipliers for constraint satisfaction.

1.2.3 Core Objectives

The core objectives are:

- **Algorithm Implementation:**
  - Translate the theoretical principles of IRWA and ADAL into practical algorithms.
  - Explore their behavior under various conditions, including convex, non-convex, and infeasible cases.
- **Performance Analysis:** The project evaluates the algorithms on success rate, convergence rate, tolerance handling, scalability and solution quality.

By addressing these goals, this project seeks to contribute to the advancement of robust and versatile QP solvers capable of meeting diverse computational demands.

1.3 Structure of the Report

This report outlines the objectives, methodology, and findings of the project. The structure is as follows:

- **Introduction:** Background and objectives of QP research, including its mathematical foundation.
- **Literature Review:** Summary of existing QP solvers, their characteristics, strengths, and limitations.
- **Algorithm Implementation:** Details on IRWA and ADAL, including their mathematical formulations and implementation steps.
- **Test, Evaluation, and Discussion:** Experimental setup, results on various QP instances, along with analysis, algorithm comparison, and potential improvements.
- **Conclusion:** Key findings and broader implications for QP research.
- **Reference**
- **Appendix**

2 Review of Quadratic Programming Solvers

Quadratic programming (QP) solvers are computational tools used to optimize quadratic objective functions subject to linear constraints. These solvers have broad applications in fields like finance, machine learning, control systems, and engineering. Modern QP solvers focus on achieving efficiency, robustness, scalability, and versatility. Iterative algorithms such as interior-point methods, active-set methods, and augmented Lagrangian techniques are central to these solvers.

2.1 Quadratic Programming Solver Methods

2.1.1 Interior-Point Methods

Interior-point methods are widely used for solving QP problems due to their efficiency and scalability. These methods solve the primal-dual KKT (Karush-Kuhn-Tucker) conditions iteratively by transforming the inequality constraints into logarithmic barrier terms:

$$\min_{\mathbf{x}} \quad g^\top \mathbf{x} + \frac{1}{2}\mathbf{x}^\top H\mathbf{x} - \mu \sum_{i=1}^{m} \log(b - A\mathbf{x}),$$

where $\mu > 0$ is the barrier parameter. As μ approaches zero, the solution converges to the optimal point.

The main advantages of interior-point methods are:

- Efficient handling of large-scale QP problems with many constraints.
- Strong theoretical convergence guarantees for convex problems.

However, they require solving large linear systems at each iteration, which can be computationally expensive.

2.1.2 Active-Set Methods

Active-set methods iteratively adjust the set of active constraints, solving a sequence of equality-constrained subproblems. At each step, the algorithm identifies whether a constraint should be added to or removed from the active set based on the KKT conditions.

This approach is particularly useful for:

- Problems with a small number of constraints relative to the problem size.
- Situations where the active set changes infrequently, leading to fewer iterations.

2.1.3 First-Order Methods: Alternating Direction Method of Multipliers (ADMM)

First-order methods are a class of algorithms used for solving optimization problems with a focus on efficiency and scalability. The Alternating Direction Method of Multipliers (ADMM) is one such method, well-suited for distributed optimization. ADMM decomposes the original problem into smaller subproblems that can be solved more efficiently in parallel.

For quadratic programming (QP) problems, ADMM splits the optimization variables $\mathbf{x}$ and reformulates the problem as:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \quad & g^\top \mathbf{x} + \frac{1}{2}\mathbf{x}^\top H \mathbf{x} \\ \text{s.t.} \quad & \mathbf{x} = \mathbf{z}, \quad A\mathbf{z} \leq \mathbf{b}. \end{aligned}$$

The method alternates between minimizing $\mathbf{x}$ and $\mathbf{z}$ while updating the Lagrange multipliers to enforce the constraints. The primary benefits of ADMM include:

- Scalability for problems with separable structures, enabling the decomposition of complex problems into simpler subproblems.

- Parallelization potential, which makes ADMM suitable for large-scale or distributed systems, where each subproblem can be solved independently.

ADMM is particularly useful in applications involving sparse data, large datasets, and when the problem structure allows decomposition into smaller, easier-to-solve subproblems.

2.1.4 Comparison of QP Solvers

Each solver has distinct strengths and weaknesses, as summarized in Table 1. Interior-point methods are robust for dense problems, while active-set methods excel in sparsity scenarios. ADMM provides flexibility in distributed settings but may require careful parameter tuning.

Table 1: Comparison of QP Solvers

Method	**Advantages**	**Disadvantages**
Interior-Point	Robust, scalable, strong convergence guarantees	High computational cost for dense problems
Active-Set	Efficient for sparse or small-scale problems	May be slow for problems with many constraints
ADMM	Scalable, suitable for distributed optimization	Requires parameter tuning, slow convergence

2.2 Modern QP Solvers

2.2.1 Evolution of Quadratic Programming Solvers

1983	The Goldfarb-Idnani (quadprog) is introduced.
1990s	QPOASES is developed at Freiburg University.
2000	MOSEK (Denmark) is released.
2013	CVXOPT is launched by Eindhoven University of Technology.
2013	Gurobi Optimization Inc. releases Gurobi solver.
2014	ECOS (Boyd Team, Stanford) becomes available.
2016	OSQP is introduced by Imperial College London.
2017	qpSWIFT is developed by Aydin Idil.
2018	HiGHS solver is developed at the University of Leeds.
2019	DAQP is launched by ETH Zurich.
2020	PIQP by Tobias Koch Team is released.
2021	Clarabel (Boyd Team) becomes available.
2021	ProxQP is launched by Inria and ENS Paris Team.
2022	SCS and HPIPM are introduced.

The timeline of QP solver development, as illustrated in figure 1, highlights the progression of solver technologies over the decades.

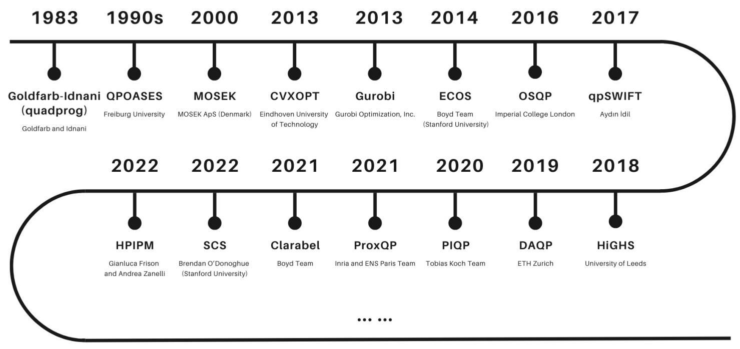

Figure 1: Timeline of QP Solvers Development

These trends underline the dynamic evolution of QP solvers, catering to diverse optimization challenges across commercial, academic, and embedded contexts:

- Commercial Solvers: Early solvers such as Gurobi and MOSEK emerged in the late 20th and early 21st centuries, offering robust solutions for industrial-scale optimization.
- Open-Source Solvers: The rise of open-source solvers, including OSQP, HiGHS, and SCS, began after 2010, driven by the need for academic transparency and meeting industrial requirements.
- Embedded and Real-Time Optimization: Recent years have witnessed a surge in solvers like DAQP and HPIPM, designed specifically for embedded systems and real-time applications.

2.2.2 Classification and Solver Characteristics

QP solvers are categorized based on their supported problem types, scale, and underlying algorithms:

- **Problem Types:**
  - Most solvers (e.g., OSQP, ECOS) target convex problems, ensuring global solutions.
  - Solvers like Gurobi and CPLEX handle non-convex cases, often at higher computational costs.
- **Problem Scale:**
  - Small-scale problems are addressed efficiently by solvers such as QPOASES.
  - Large-scale sparse problems are handled by HiGHS and Gurobi, optimized for industrial-grade applications.
- **Algorithmic Types:**
  - **Interior-Point Methods:** Focus on high-precision optimization (e.g., CVXOPT, MOSEK).
  - **Active-Set Methods:** Excel in solving dense, small-scale problems (e.g., QPOASES).
  - **Augmented Lagrangian Methods:** Balance dense and sparse problem handling (e.g., ProxQP).
  - **Other Methods:** Leverage specialized techniques like Douglas–Rachford splitting for sparse matrices (e.g., OSQP).

2.2.3 Key Solvers and Their Applications

The following table 2 summarizes key QP solvers and their applications:

Table 2: Characteristics of Major QP Solvers

Solver	**Algorithm Type**	**Supported Problem Type**	**Matrix Features**	**Application Scenarios**
OSQP	Douglas–Rachford	Convex Problems	Sparse	Embedded systems, real-time control
Gurobi	Interior Point + Mixed Integer	Convex and Non-Convex Problems	Sparse	Financial optimization, engineering
HiGHS	Active Set	Convex Problems	Sparse	Industrial-scale optimization
ProxQP	Augmented Lagrangian	Convex Problems	Dense and Sparse	High-precision optimization
CVXOPT	Interior Point	Convex Problems	Dense	Academic research, simulations
QPOASES	Active Set	Convex Problems	Dense	Small-scale real-time optimization
MOSEK	Interior Point	Convex Problems	Sparse	High-precision financial optimization
Clarabel	Interior Point	Convex Problems	Sparse	Control, Optimization in Sparse Systems
DAQP	Active Set	Convex Problems	Dense	Real-time Optimization, Robotics
ECOS	Interior Point	Convex Problems	Sparse	Financial Optimization, Control
HPIPM	Interior Point	Convex Problems	Dense	Embedded Optimization, MPC
PIQP	Proximal Interior Point	Convex Problems	Dense and Sparse	Quadratic Programming, Portfolio Optimization
qpSWIFT	Interior Point	Convex Problems	Sparse	Small-scale optimization, research
quadprog	Goldfarb-Idnani	Convex Problems	Dense	Academic research, prototyping
SCS	Douglas–Rachford	Convex Problems	Sparse	Large-scale optimization, control

2.3 Performance and Benchmarks

2.3.1 Performance Metrics

Evaluating QP solvers involves metrics such as:

- **Success Rate**: Proportion of solved problems, reflecting robustness.

- **Runtime Efficiency**: Solution time, critical for real-time/large-scale use.
- **Stability**: Ability to converge reliably under varying conditions, such as ill-conditioned problems or different numerical tolerances, measured through:
  - Primal Residual: Maximum error on equality and inequality constraints at the returned solution.
  - Dual Residual: Maximum error on the dual feasibility condition at the returned solution.
  - Duality Gap: Measure of the gap between primal and dual solutions.

2.3.2 Performance Comparison Based on Benchmarks

The Maros-Meszaros dataset benchmarks solver performance across 138 problems, including:

- Dense problems (62 instances): Test computational efficiency with small, dense matrices.
- Sparse problems: Assess scalability for large-scale scenarios.

Figure 2 shows solver performance on the dense subset with high accuracy. The x-axis represents runtime (log scale), and the y-axis indicates the number of problems solved.

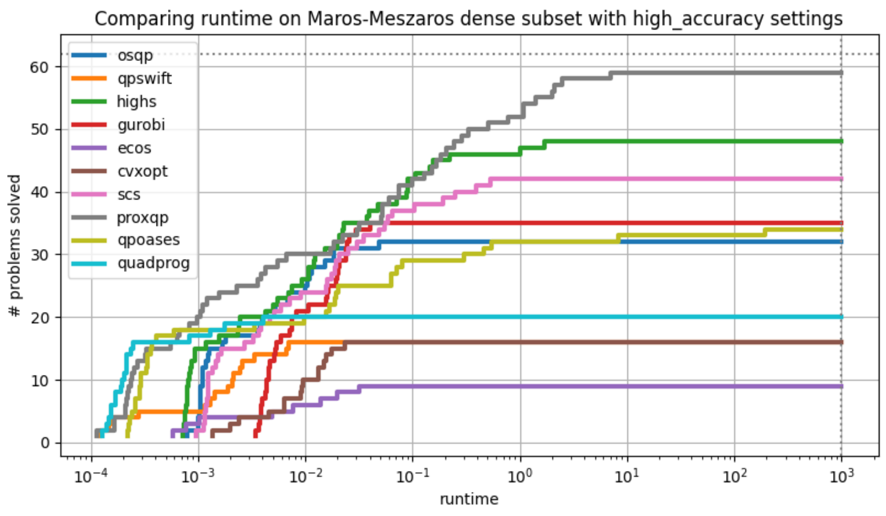

Figure 2: Solver performance on Maros-Meszaros dense subset under high-accuracy settings

2.3.3 Comprehensive Performance Analysis

The Table 3 below shows the performance comparison of different QP solvers based on Benchmark tests:

Table 3: Performance Comparison of Solvers Based on Benchmark Tests

Solver	Success Rate	Computation Time	Stability Metrics
OSQP	High for convex problems	Fast for sparse problems	Stable primal and dual residuals
Gurobi	Very high (convex + nonconvex)	Fast with multithreaded support	Robust across all metrics
HiGHS	Moderate	Fast for sparse problems	Reliable for large-scale convex problems
ProxQP	High for precision modes	Moderate	Excellent numerical stability
CVXOPT	High for small-medium size	Slow	Stable for dense problems

- **Success Rate:**
  - Gurobi demonstrates nearly perfect success across problem types, excelling in both convex and non-convex cases.
  - OSQP excels in sparse and convex scenarios, suitable for real-time applications.

  - ProxQP performs well in high-accuracy modes but requires more runtime for complex problems.
- **Runtime Efficiency:**
  - OSQP and HiGHS are highly efficient for sparse problems, ideal for large-scale applications.
  - Gurobi, with multithreaded support, achieves superior runtime across diverse problem sizes.
  - ProxQP balances precision and runtime, particularly effective for dense problems.
- **Stability and Scalability:**
  - Gurobi and ProxQP maintain robust convergence under varying conditions.
  - OSQP shows reliable performance in embedded systems requiring fast convergence.
  - CVXOPT struggles with large-scale or highly complex problems.

3 Algorithm Implementation

3.1 IRWA

3.1.1 Origin Algorithm

The Iterative Reweighted Algorithm (IRWA) is a penalty-based optimization method designed to solve quadratic programming (QP) problems through the approximation of exact penalty subproblems. The algorithm iteratively minimizes a reweighted quadratic objective, updating weights and relaxation vectors dynamically to handle constraints effectively. IRWA is especially suited for problems where constraints are challenging to satisfy or infeasibility is a concern.

Key characteristics of IRWA include:

- **Dynamic Weight Adjustment:** Iterative updates of weight vectors ensure constraint satisfaction while refining the solution.
- **Exact Penalty Approximation:** The use of reweighted penalties closely approximates the original constrained problem.
- **Scalability and Robustness:** Effective for handling high-dimensional and challenging QP problems.

Algorithm 1 IRWA

```
Require: Initial values $x^0$, $\epsilon^0 > 0$, parameters $\eta \in (0,1)$, $\gamma > 0$, $M > 0$, $\text{tol}_{\text{prim}}$, $\text{tol}_{\text{dual}}$, and
    maximum iterations $k_{\max}$
1: Initialize iteration counter $k = 0$
2: repeat
3:
     $$w_i(x,\epsilon) := \begin{cases} \left(|a_i^\top x + b_i|^2 + \epsilon_i^2\right)^{-1/2}, & i \in \mathcal{I}_1, \\ \left(\max\{(a_i^\top x + b_i), 0\}^2 + \epsilon_i^2\right)^{-1/2}, & i \in \mathcal{I}_2, \end{cases} \quad (1)$$
4:     $W_k = \text{diag}(w_i^k)$
5:     $x^{k+1} \leftarrow \arg\min_x \; g^\top x + \frac{1}{2}x^\top H x + \frac{1}{2}(Ax+b)^\top W_k (Ax+b)$
6:     $r_i^k = (1 - v_i^k)(A_i x^k + b_i)$
7:     $q_i^k = A_i(x^{k+1} - x^k)$
8:     if $|q_i^k| \le M \cdot \left((r_i^k)^2 + \epsilon_k^2\right)^{0.5+\gamma}$ then
9:         $\epsilon_i^{k+1} \leftarrow \eta \cdot \epsilon_i^k$
10:    end if
11: until termination criterion is satisfied
```

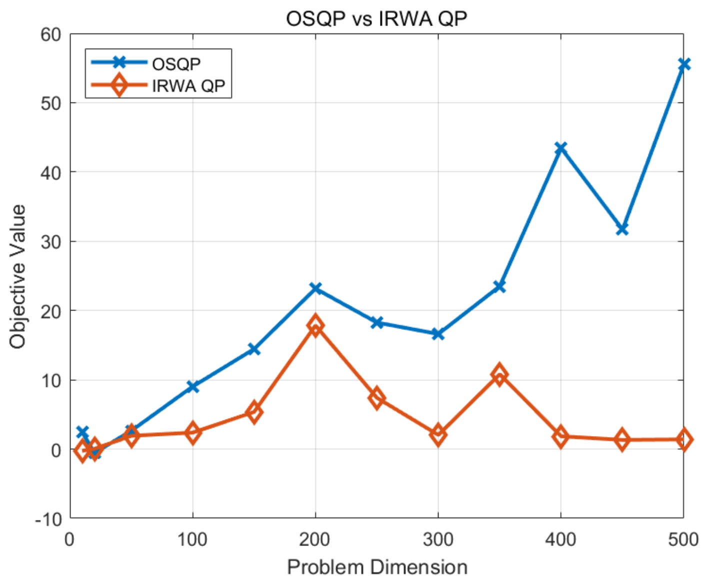

Figure 3: OSQP vs origin IRWA

This figure shows a comparison of the original IRWA algorithm and the OSQP optimal solution. The difference between IRWA and OSQPacross varying dimensions are highlighted. We can also find that optimal value are always lower than those of OSQP.

3.1.2 IRWA with ϵ^{-1}-Threshold

We observed that the rate at which equality and inequality constraints were penalized to reach the feasible domain was not consistent, which led to the original algorithm producing lower objective values than the OSQP solver. This inconsistency arises due to penalty weights not being on the same scale, which encourages the algorithm to violate equality constraints(which achieves the feasible domain slower and thus has much smaller penalty weights) to achieve a lower objective value. To handle this problem, we implemented the ϵ^{-1}-Threshold to ensure proper penalty synchronization.

Methodology

- **Alignment**: The ϵ^{-1}-threshold serves as a lower bound for the penalty weights and thus aligns the weights across equality and inequality constraints.

  The ϵ^{-1}-threshold method is performed after computing 1 and the weight w_i for each iteration is updated as:

$$w_i \leftarrow \max\left(w_i, \epsilon^{-1}\right) \tag{2}$$

  This ensures consistent penalty weighting scale between equality and inequality constraints.

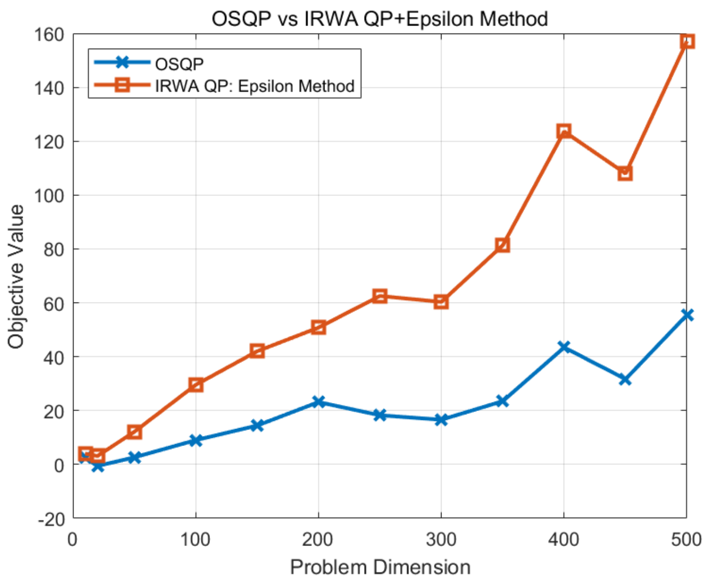

Figure 4: OSQP vs IRWA+Epsilon Method

3.1.3 IRWA with ϵ^{-1}-Threshold and Scaling on

We observed that when ϵ drops quickly, (i.e. the penalty weights rises fast), the linear system becomes ill-conditioned and the algorithm terminated before reaching the optimal point. To address this, we performed a matrix scaling method.[1]

Methodology

- **Scaling Factors**: The Frobenius norm of A_1 (equality constraint matrix) is used to compute the scaling factor for equality constraints, while the Frobenius norm of A_2 (inequality constraint matrix) is used for inequality constraints:

$$\text{scale\_eq} = \frac{1}{\|A_1\|_F}, \quad \text{scale\_ineq} = \frac{1}{\|A_2\|_F}.$$

- **Scaling Matrix**: A diagonal scaling matrix is constructed using the computed scaling factors:

$$S = \text{diag}([\text{scale\_eq} \cdot \mathbf{1}_{m_{\text{eq}}}; \text{scale\_ineq} \cdot \mathbf{1}_{m_{\text{ineq}}}]),$$

  where m_{eq} and m_{ineq} are the number of equality and inequality constraints, respectively.

- **Transformation**: The constraint matrix A and vector b are scaled as:

$$A_{\text{scaled}} = S \cdot A, \quad b_{\text{scaled}} = S \cdot b.$$

  Similarly, the weight vector w and other associated terms are scaled within each iteration.

The scaling approach ensures:

1. Improved Numerical Stability: The rescaled problem reduces sensitivity to **ill-conditioned** matrices, particularly in A_1 and A_2.
2. Faster Convergence: The iterative solution benefits from a more balanced problem structure, leading to stable and efficient updates.

Combining ϵ^{-1}-Threshold with Scaling, we obtain the final algorithm:

Algorithm 2 IRWA with Scaling and ϵ^{-1}-Threshold

```
Require: Initial values $x^0$, $\epsilon^0 > 0$, parameters $\eta \in (0,1)$, $\gamma > 0$, $M > 0$, tol$_{\text{prim}}$, tol$_{\text{dual}}$, and
    maximum iterations $k_{\max}$, $A_1$, $A_2$ ,$b_1$, $b_2$, $g$,$H$
 1: scale_eq $= \frac{1}{\|A_1\|_F}$,    scale_ineq $= \frac{1}{\|A_2\|_F}$
 2: $S = \text{diag}([\text{scale\_eq} \cdot \mathbf{1}_{m_{\text{eq}}}; \text{scale\_ineq} \cdot \mathbf{1}_{m_{\text{ineq}}}])$
 3: $A_{\text{scaled}} = S \cdot A$,    $b_{\text{scaled}} = S \cdot b$
 4: repeat
 5:
        $$w_i(x,\epsilon) := \begin{cases} \left(|a_i^\top x + b_i|^2 + \epsilon_i^2\right)^{-1/2}, & i \in \mathcal{I}_1, \\ \left(\max\{(a_i^\top x + b_i), 0\}^2 + \epsilon_i^2\right)^{-1/2}, & i \in \mathcal{I}_2, \end{cases}$$
 6:     $w_i \leftarrow \max(w_i, \epsilon^{-1})$
 7:     $W_k = \text{diag}(w_i^k)$
 8:     $x^{k+1} \leftarrow \arg\min_x \; g^\top x + \frac{1}{2} x^\top H x + \frac{1}{2}(A_{\text{scaled}} x + b_{\text{scaled}})^\top W_k (A_{\text{scaled}} x + b_{\text{scaled}})$
 9:     $r_i^k = (1 - v_i^k)(A_{\text{scaled},i} x^k + b_{\text{scaled},i})$
10:     $q_i^k = A_{\text{scaled},i}(x^{k+1} - x^k)$
11:     if $|q_i^k| \le M \cdot \left((r_i^k)^2 + \epsilon_k^2\right)^{0.5+\gamma}$, $\forall i$ then
12:         $\epsilon_i^{k+1} \leftarrow \eta \cdot \epsilon_i^k$
13:     end if
14: until termination criterion is satisfied
```

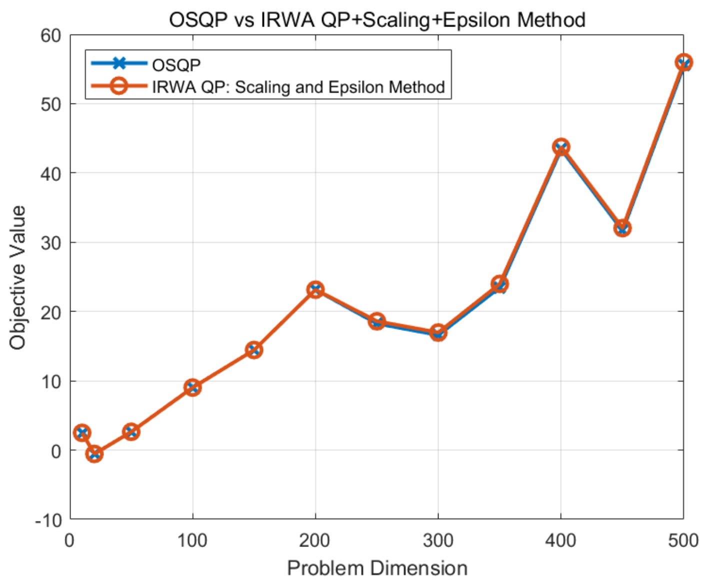

Figure 5: OSQP vs IRWA+Scaling+Epsilon

Figure 5 illustrates the comparison between OSQP and IRWA with the combined Scaling and ϵ^{-1}-Threshold methods.

- The combined method achieves objective values almost identical to OSQP across all problem dimensions, indicating its effectiveness in maintaining optimality.
- The method demonstrates consistent performance, avoiding divergence or instability even for high-dimensional problems.
- Using only ϵ^{-1}-Threshold or Scaling independently leads to deviations from the OSQP baseline, the combined approach successfully balances penalty synchronization and numerical stability.

These results confirm that incorporating both Scaling and ϵ^{-1}-Threshold into IRWA enhances its robustness and makes it highly competitive with state-of-the-art solvers like OSQP.

3.1.4 Comparison Test

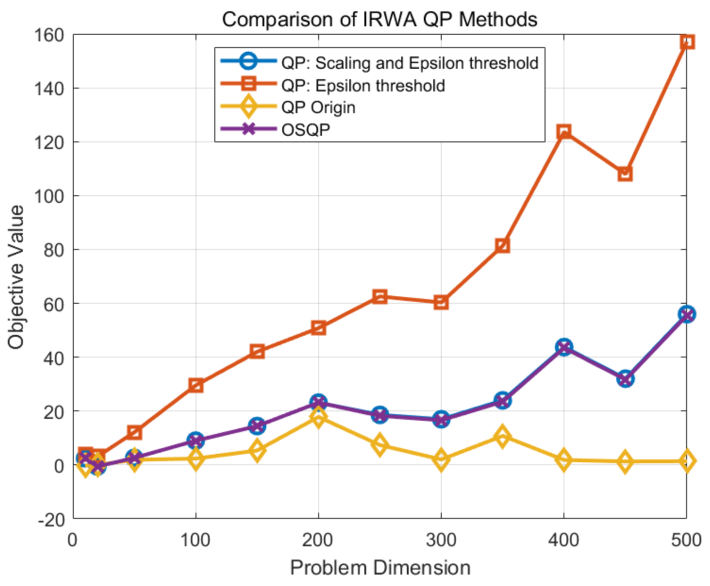

Figure 6: Conclusion of IRWA QP and enhance methods

The results in Figure 6 show that combining the ϵ^{-1}-Threshold and Scaling methods significantly improves the IRWA algorithm:

1. The combined method achieves objective values nearly identical to those of OSQP across all dimensions, ensuring optimality and accuracy.

2. ϵ^{-1}-Threshold alone results in higher objective values, while Scaling alone underestimates the optimal values. The combined approach effectively balances penalties for better performance.

3. The combined method remains robust and consistent as problem dimensions increase, avoiding divergence or instability.

This demonstrates that the combined ϵ^{-1}-Threshold and scaling approach is reliable and comparable to OSQP, offering both stability and optimality for QP problems.

3.2 ADMM

3.2.1 Origin Algorithm

The Accelerated Distributed Augmented Lagrangian (ADAL) is a variant of the ADMM algorithm designed to address convergence issues in multi-block optimization problems under Jacobian decomposition[2]. ADAL introduces a correction step that modifies only the primal variables to ensure convergence in a broader set of scenarios.

Conjugate Gradient (CG) is used to solve the linear system in each iteration, making the method scalable to large problems.

Algorithm 3 ADMM

```
Require: initial values $x^0, z^0, y^0$ and parameters $\rho > 0, \sigma > 0, \alpha \in (0, 2)$
1: repeat
2:     $(\tilde{x}^{k+1}, \nu^{k+1}) \leftarrow$ solve linear system
           $\begin{bmatrix} P+\sigma I & A^T \\ A & -\rho^{-1}I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \rho^{-1}y^k \end{bmatrix}$
3:     $\tilde{z}^{k+1} \leftarrow z^k + \rho^{-1}(\nu^{k+1} - y^k)$
4:     $x^{k+1} \leftarrow \alpha\tilde{x}^{k+1} + (1-\alpha)x^k$
5:     $z^{k+1} \leftarrow \Pi\big(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k + \rho^{-1}y^k\big)$
6:     $y^{k+1} \leftarrow y^k + \rho\big(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k - z^{k+1}\big)$
7: until termination criterion is satisfied
```

3.2.2 Adaptive Parameter for ADMM

Adaptive Parameter improve ADMM algorithm by dynamically compute penalty parameter ρ based on the current residuals. This ensures better convergence properties and stability, particularly for problems with varying scales or challenging constraints.

It mainly involves the enhancement of dynamic Penalty Adjustment: The penalty parameter ρ is computed adaptively in each iteration based on the ratio of primal and dual residuals.

Methodology

- The adaptive adjustment of ρ is based on the norms of the scaled primal and dual residuals:

$$\text{primal\_res} = \frac{\|Ax - z\|_\infty}{\|z\|_\infty + \|Ax\|_\infty + \text{tol}}, \quad \text{dual\_res} = \frac{\|Px + q + A^T y\|_\infty}{\|q\|_\infty + \|A^T y\|_\infty + \|Px\|_\infty + \text{tol}}.$$

  Here, ρ is updated as:

$$\rho = \min\left(\max\left(\rho \cdot \sqrt{\frac{\text{primal\_res}}{\text{dual\_res}}}, \rho_{\min}\right), \rho_{\max}\right).$$

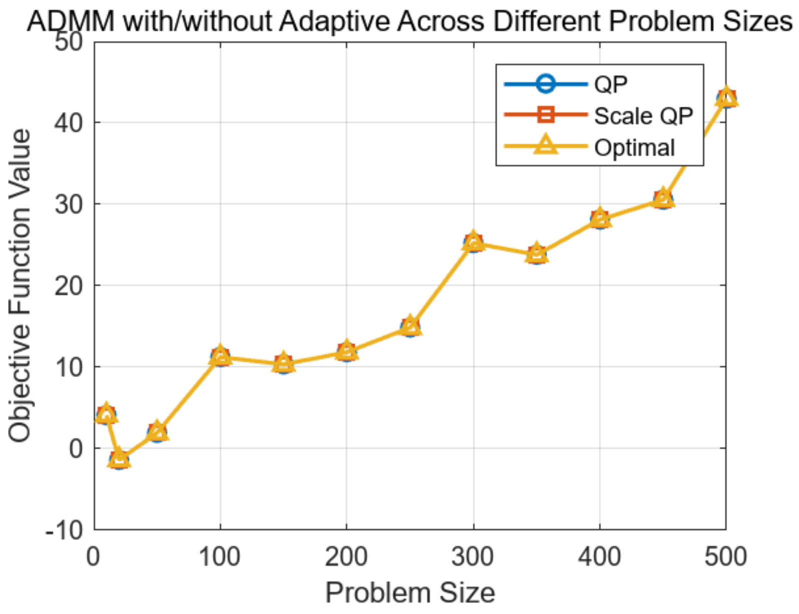

Figure 7

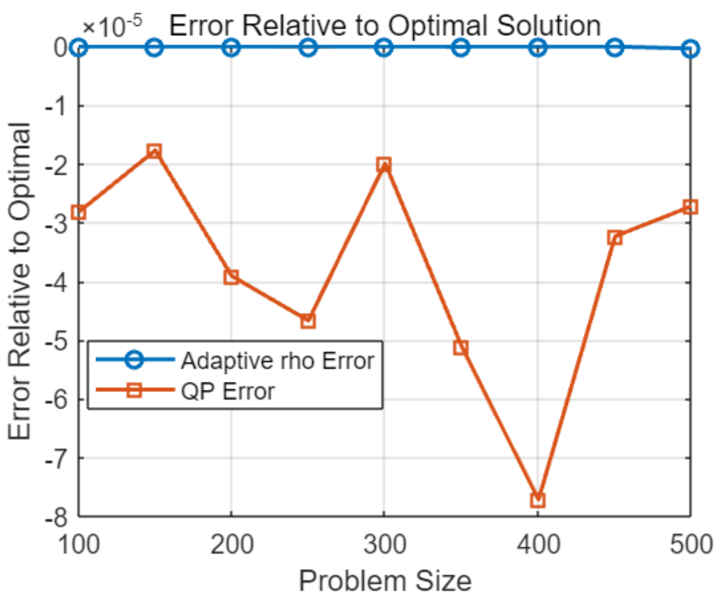

Figure 8: Error Relative to Optimal Solution Using Adaptive Rho

Figure 8 demonstrates the error relative to the optimal solution. The adaptive method could reduce the error compared to the standard fixed penalty approach, showcasing its effectiveness in improving accuracy and stability.

4 Test and Evaluation

4.1 Experimental Design

To evaluate the performance of the IRWA and ADMM algorithms, we designed an experimental setup involving the generation of Quadratic Programming (QP) problems. These problems require the matrix H to be positive semi-definite for convex optimization. We generated H by creating a random symmetric matrix and ensuring its positive definiteness through the addition of an identity matrix scaled by a positive constant. This process is mathematically described as:

$$H = H_{\text{random}} + \alpha I_n,$$

where H_{random} is a randomly generated symmetric matrix, I_n represents the identity matrix of size $n \times n$, and $\alpha > 0$ ensures all eigenvalues of H are non-negative, satisfying the convexity condition. This approach ensures that the resulting QP problems are well-posed and suitable for evaluating the performance of optimization algorithms.

The generated QP problems were solved using both the IRWA and ADMM algorithms. The performance of these algorithms was evaluated based on three primary metrics:

1. **Number of Iterations**: Measures the convergence speed of the algorithm.
2. **Constrained Residual and Duality Gap**: Quantifies the accuracy and feasibility of the solution.
3. **Computation Time**: Captures the efficiency of the algorithm in solving QP problems of varying sizes.

4.1.1 First Experiment

In the first experiment, we randomly generated 500 QP problems with different dimensions of the constraint matrix A, specifically:

$$A \in \mathbb{R}^{8\times 20}, \quad A \in \mathbb{R}^{100\times 200}, \quad A \in \mathbb{R}^{200\times 500}.$$

These dimensions were chosen to test the scalability and performance of the algorithms under a variety of problem sizes. For each iteration, we computed the percentage of solved problems to evaluate the convergence behavior of IRWA and ADMM.

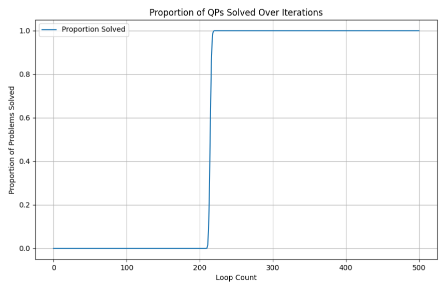

Figure 9: IRWA: Percentage of Problems Solved Over Iterations

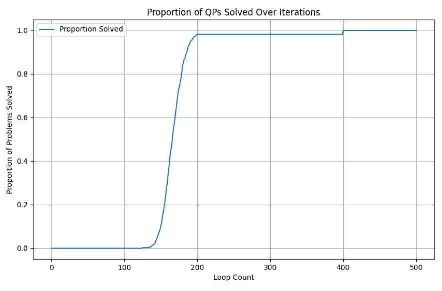

Figure 10: ADMM: Percentage of Problems Solved Over Iterations

The experiment reveals key insights into the behavior of both algorithms. While ADMM demonstrates faster initial progress, IRWA eventually achieves comparable performance but requires more iterations to reach the same level of feasibility.

Figure 9 provides a clear view of IRWA's gradual convergence process, where the percentage of solved problems remains minimal in the initial iterations but experiences a steep rise after approximately 200 iterations. In contrast, Figure 10 underscores ADMM's ability to solve problems more quickly in the early stages. The rapid increase in the percentage of solved problems, beginning at around 100 iterations, showcases ADMM's effectiveness in leveraging distributed computation to address constraints and achieve feasibility efficiently.

In conclusion, these results underline the complementary strengths of IRWA and ADMM, with IRWA favoring accuracy and long-term stability and ADMM prioritizing speed and early feasibility. This experiment provides valuable guidance on selecting the appropriate algorithm for different problem scenarios.

4.1.2 Second Experiment

In this experiment, we compare the performance of IRWA and ADMM against the optimal solution of OSQP. The objective is to measure the relative deviation of the objective function values obtained by IRWA and ADMM from the optimal solution, as well as the differences between normal and accelerated IRWA methods.

The results are calculated as the relative differences in the objective function values. It is represented as:

$$\frac{\text{optimal value} - \text{optimal value of OSQP}}{\text{optimal value of OSQP}} \times 100$$

This allows us to evaluate the effectiveness of acceleration techniques in improving the performance of IRWA. Additionally, the deviations of both IRWA and ADMM from OSQP's optimal solution provide insights into the trade-offs between different methods in terms of solution accuracy.

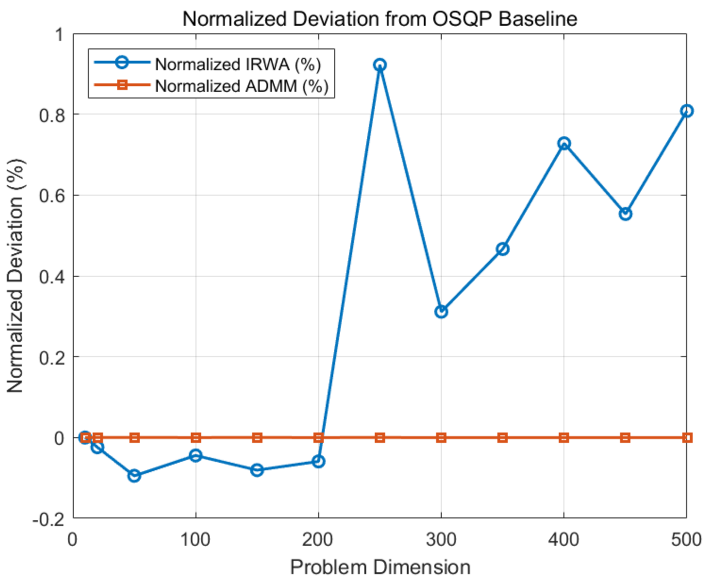

Figure 11

The Figure 11 shows that IRWA method exhibits larger fluctuations in normalized deviation compared to ADMM. While ADMM consistently maintains a normalized deviation close to 0%, demonstrating its robustness and stability across all problem dimensions.

4.1.3 Third Experiment

This experiment evaluates the computation time of IRWA, ADMM, and OSQP (baseline) for problem sizes ranging from 20 to 500 dimensions. The average computation time is computed over 50 runs for each size. Figure 12 shows the computation time for IRWA and ADMM, while Figure 13 compares their runtime ratios relative to OSQP.

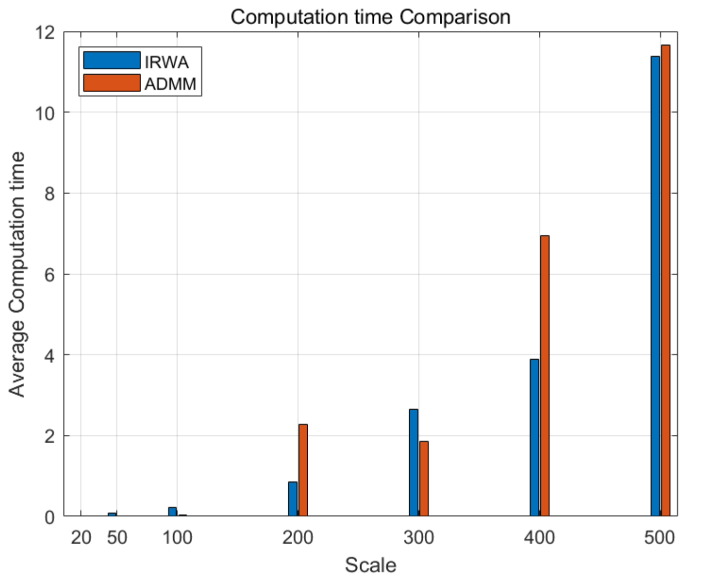

Figure 12: Computation time across different problem sizes for IRWA and ADMM

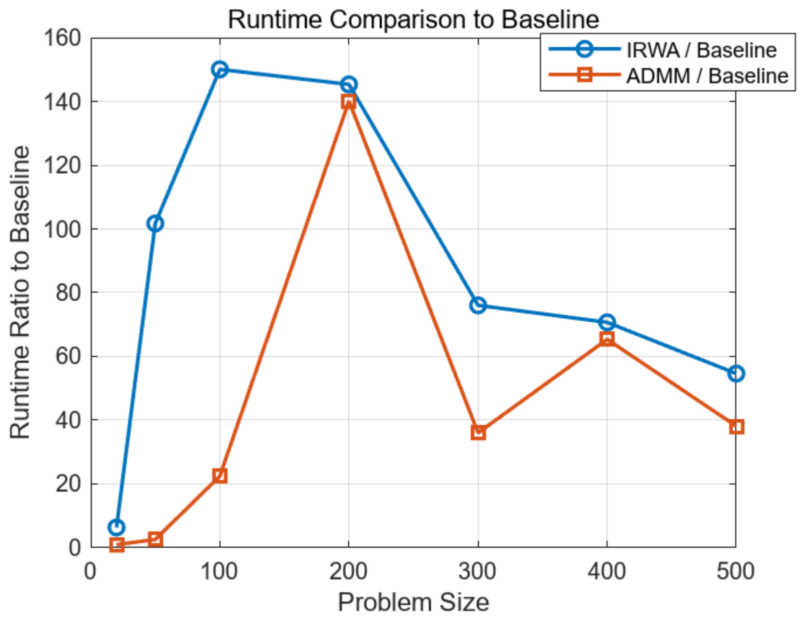

Figure 13: Comparison of runtime ratio to baseline (OSQP) for IRWA and ADMM

Analysis of Results

- **Computation Time Trends**: As shown in Figure 12, ADMM consistently achieves faster computation times than IRWA, particularly for medium-sized problems (200–300 dimensions). However, IRWA demonstrates more stable performance as problem sizes grow, albeit with a higher computation time overall.
- **Runtime Ratio to Baseline**: Figure 13 illustrates the runtime ratio relative to OSQP. IRWA exhibits a sharp peak for problem sizes around 200, indicating increased overhead during scaling adjustments. ADMM maintains a lower runtime ratio but shows instability for medium-sized problems, potentially due to convergence challenges.
- **Large-Scale Performance**: For large problem sizes (above 400 dimensions), both IRWA and ADMM show improved relative efficiency, converging closer to the OSQP baseline, indicating better scalability for high-dimensional problems.

Conclusions ADMM demonstrates better overall efficiency and is well-suited for small to medium-sized problems. However, its performance instability for certain scales highlights the need for further optimization. IRWA, while slower, provides stable and reliable performance, making it more suitable for scenarios requiring robustness in handling complex constraints.

References

[1] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, Dec 2020.

[2] Yu Yang, Xiaohong Guan, Qing-Shan Jia, Liang Yu, Bolun Xu, and Costas J. Spanos. A survey of admm variants for distributed optimization: Problems, algorithms and features, 2022.

A Appendix / supplemental material

- Wiki: Quadratic programming.
  Available at: `https://en.wikipedia.org/wiki/Quadratic_programming`
- PyPI: qpsolvers_benchmark. Benchmark for quadratic programming solvers available in Python.

Available at: https://pypi.org/project/qpsolvers_benchmark/ (accessed on December 12, 2024)

- GitHub: maros_meszaros_qpbenchmark. Benchmark datasets and test results for QP solvers. Available at: https://github.com/qpsolvers/maros_meszaros_qpbenchmark (accessed on December 12, 2024).
- Burke, J. V., Curtis, F. E., Wang, H., and Wang, J. (2015). Iterative Reweighted Linear Least Squares for Exact Penalty Subproblems on Product Sets. SIAM Journal on Optimization, 25(1), 261–294.
  Available at: https://www.siam.org/journals/siopt/25-1/95023.html