Formatting Instructions For NeurIPS 2024

Shenghan Gao
ShanghaiTech University
2021533103
gaoshh1@shanghaitech.edu.cn

Mokai Pan
ShanghaiTech University
2021533106
panmk@shanghaitech.edu.cn

Hongjin Wang
ShanghaiTech University
2021533014
wanghj1@shanghaitech.edu.cn

Abstract

This report investigates and implements two key algorithms, Iterative Reweighting Algorithm (IRWA) and Alternating Direction Augmented Lagrangian (ADAL), to solve quadratic programming (QP) problems with both equality and inequality linear constraints. IRWA transforms the original problem into a sequence of weighted convex subproblems, while ADAL alternates between optimizing subproblems and updating Lagrange multipliers to achieve global convergence.

1 Introduction

Quadratic programming (QP) plays a crucial role in numerous optimization applications, ranging from control systems to machine learning. A standard QP problem minimizes a quadratic objective function subject to linear equality and inequality constraints. Traditional solvers often struggle with large-scale or ill-conditioned problems, motivating the development of advanced algorithms.

2 Backgrounds

Quadratic programming is widely used in engineering optimization, economics, support vector machines, etc. Its particularity lies in the fact that the quadratic nature of the objective function makes the problem mathematically sound, while the linear constraints make it easy to handle. We have searched the Web for some popular and advanced solvers for quadratic programming that are currently available, and we will have a basic look at some of those solvers.

2.1 ProxQP

ProxQP [1], introduced in September 2023, is a quadratic programming solver initially designed for robotics applications. It has demonstrated comparable performance to leading solvers across generic QP problems, proving its versatility as a reliable off-the-shelf tool for broader applications. The algorithm is based on primal-dual augmented Lagrangian techniques and ensures global convergence while solving convex QPs efficiently. Notably, PROXQP addresses primal infeasibility by solving the closest primal-feasible QP.

Meanwhile, the algorithm incorporates enhancements from the proximal method of multipliers (PMM) and primal-dual proximal methods (PDPMM), improving convergence through penalty functions and proximal terms. Implemented within the PROXSUITE library, it supports C++, Julia, and Python, achieving efficiency across sparse, dense, and generic QPs. Its robustness has been showcased in standard robotics and control experiments, particularly in closed-loop model predictive control scenarios.

This solver exemplifies advancements in QP solvers, blending theoretical rigor with practical efficiency for real-world applications.

2.2 OSQP

OSQP [2], short for 'Operator Splitting Quadratic Program', is a versatile, general-purpose QP solver written in C and designed to be library-free. The solver is based on the Alternating Direction Method of Multipliers (ADMM) and delivers high-accuracy solutions using a novel splitting technique. This technique involves solving a quasi-definite linear system, which ensures solvability regardless of problem data. Unlike traditional methods, OSQP does not require strict convexity of the cost function or linear independence of constraints.

The algorithm performs a single matrix factorization when parameters are fixed, reusing it across iterations to enhance efficiency. Occasionally, the factorization is updated to improve convergence. To refine solution accuracy, OSQP applies "solution polishing," where active constraints are identified in the final iteration, and an auxiliary equality-constrained QP is solved. This approach produces solutions comparable to or better than interior-point methods.

OSQP also supports warm-starting and efficient factorization reuse, making it particularly effective for parametric QPs. Combining ideas from active-set methods, interior-point methods, and first-order techniques, OSQP has proven its adaptability and efficiency in solving a wide range of problems.

2.3 MOSEK

MOSEK [3] is a high-performance optimization solver designed to address a variety of problem types, including quadratic programming. Known for its efficiency, robustness, and versatility, it is widely applied in both academic research and industrial scenarios. MOSEK supports the following problem types:

- **Convex Quadratic Programming (Convex QP):** Solves problems with convex quadratic objectives and linear constraints.
- **Non-convex Quadratic Programming (Non-convex QP):** Handles specific non-convex QPs, often reformulated as conic optimization problems.
- **Quadratically Constrained Quadratic Programming (QCQP):** Addresses problems with quadratic constraints.
- **Large-scale Sparse QP:** Efficiently solves sparse, high-dimensional problems.

MOSEK incorporates advanced techniques, including:

- Interior-point method
- Conic optimization reformulation
- Simplex method
- Parallel computing

These methods ensure numerical stability, making MOSEK particularly effective for ill-conditioned or noisy problems. It is capable of handling optimization problems involving millions of variables and constraints, utilizing sparse problem structures to minimize computation time and memory usage.

Rather than being limited to a single algorithm, MOSEK functions as a flexible toolbox, offering multiple techniques to adapt to different optimization challenges. For instance, the Interior-Point Method provides robust solutions for various quadratic programming applications.

2.4 Recent Recovery: BLACK-BOX QUADRATIC PROGRAMMING SOLVERS

In recent years, many deep learning methods have integrated optimization problems as layers in neural networks. However, incorporating these optimization layers requires computing derivatives of the solution with respect to problem parameters, which existing black-box solvers often lack. To address this limitation, dQP [4], introduced in December 2024, provides a modular framework that enables differentiable interfaces for any QP solver.

The key theoretical insight of dQP is its use of the active constraint set at the QP optimum to efficiently compute derivatives. By leveraging this relationship, the framework supports seamless integration into neural networks and bi-level optimization tasks. Importantly, dQP requires only the primal solution for differentiation, making it computationally efficient.

The implementation supports over 15 state-of-the-art QP solvers, transforming them into differentiable layers for immediate use in learning tasks. Benchmarks demonstrate dQP's scalability and effectiveness across dense, sparse, and structured QP problems, as well as novel applications like bi-level geometry optimization.

3 Methods

QP problems a type of optimization problem where the objective function is quadratic, and the constraints are linear and can be formulated as follows:

$$\begin{aligned} &\min_x \varphi(x), \quad \text{where} \quad \varphi(x) := g^T x + \frac{1}{2} x^T H x \\ &\text{s.t.} \quad Ax + b \in \mathcal{C} \end{aligned} \tag{1}$$

And a simple version is

$$\begin{aligned} &\min_x \; g^T x + \frac{1}{2} x^T H x \\ &\text{s.t. } a_i^T x + b_i = 0, \forall i \in \mathcal{I}_1 := \{1, 2, \cdots, l\} \\ &\quad a_i^T x + b_i \le 0, \forall i \in \mathcal{I}_2 := \{l+1, \cdots, m\} \end{aligned} \tag{2}$$

where $x \in \mathbb{R}^n$ is the decision variable, $g \in \mathbb{R}^n$ is a vector, $H \in \mathbb{R}^{n \times n}$ is a symmetric matrix, $A \in \mathbb{R}^{m \times n}$ is a matrix, $b \in \mathbb{R}^m$ is a vector, $\mathcal{C} \subset \mathbb{R}^m$ is a nonempty, closed set, for any $i \in \mathcal{I} := \mathcal{I}_1 \cap \mathcal{I}_2$, $a_i^T \in \mathbb{R}^n$ is the i-th row of A and $b_i \in \mathbb{R}$ is the i-th element of b.

As to the problem in the form of (2), we can define its related exact penalty subproblems as follows:

$$\min_{x \in \mathcal{X}} \quad J(x), \quad \text{where} \quad J(x) := g^T x + \frac{1}{2} x^T H x + M_1 \sum_{i \in \mathcal{I}_1} |a_i^T x + b_i| + M_2 \sum_{i \in \mathcal{I}_2} \max\{a_i^T x + b_i, 0\} \tag{3}$$

where M_1 and M_2 are two individual penalty coefficients which can be changed according to different problems. In most situations, we take $M_1 = 1$ and $M_2 = 1$.

The following two algorithms are all based on the exact penalty subproblems (3).

3.1 Iterative Reweighting Algorithm (IRWA)

The Iterative Reweighting Algorithm is used to solve optimization problems with non-convex objectives. It transforms the original problem into a sequence of weighted convex subproblems. IR addresses optimization problems of the following form:

$$\min_x f(x) + \phi(x), \tag{4}$$

where $f(x)$ is a convex function, and $\phi(x)$ is a non-convex regularization term. Examples include:

$$\phi(x) = \|x\|_p \quad (0 < p < 1), \quad \text{or} \quad \phi(x) = \sum_i |x_i|. \tag{5}$$

With constraints, the problem extends to:

$$\begin{aligned} &\min_x f(x) + \phi(x), \\ &\text{s.t.} \quad A_1 x + b_1 = 0, \quad A_2 x + b_2 \le 0. \end{aligned} \tag{6}$$

The key steps of the algorithm are as follows:

1. Weighted Approximation: Replace the non-convex term with a weighted convex approximation:

$$\phi(x) \approx \sum_i w_i(x) f(x_i), \tag{7}$$

where $w_i(x)$ are weights and $f(x_i)$ is convex.

2. Weight Update Rule: Update weights iteratively:

$$w_i^{(k+1)} = g(x_i^{(k)}). \tag{8}$$

3. Convex Optimization: Solve the weighted convex problem at each iteration:

$$\min_x f(x) + \sum_i w_i^{(k)} f(x_i). \tag{9}$$

We concluded the IRWA algorithm stated in the paper [5] as the following pseudocode:

Algorithm 1 Iterative Reweighting Algorithm (IRWA)

```
1:  Initialization: Choose $x^{(0)}$, $\epsilon^{(0)}$, $\eta \in (0,1)$, $\gamma > 0$, $M > 0$, penalty factor $M_1, M_2 = 1$ and
    tolerances $\sigma, \sigma' \geq 0$. Set $k = 0$.
2:  while stopping criteria not met do
3:      Step 1: Solve weighted subproblem.
4:      Solve:
            $\min_{x \in X} \quad \hat{G}_{(x^{(k)}, \epsilon^{(k)})}(x)$
            Linear system: $(H + A^T W A)x + (g + A^T W v) = 0$
5:      Step 2: Update relaxation vector.
6:      Compute $q_i^{(k)}$ and $r_i^{(k)}$. Update $\epsilon^{(k+1)}$ based on:
            $\epsilon^{(k+1)} = \begin{cases} \in (0, \eta\epsilon^{(k)}] & \text{if condition satisfied} \\ \epsilon^{(k)} & \text{otherwise} \end{cases}$
7:      Step 3: Check stopping criteria.
8:      Stop if $\|x^{(k+1)} - x^{(k)}\|_2 \leq \sigma$ and $\|\epsilon^{(k)}\|_2 \leq \sigma'$.
9:      Update $k \leftarrow k + 1$.
10: end while
```

3.2 Alternating Direction Augmented Lagrangian (ADAL)

The Alternating Direction Augmented Lagrangian (ADAL or ADMM) is an algorithm for distributed convex optimization, particularly effective for problems with separable structures. It alternates between optimizing subproblems and updating Lagrange multipliers to achieve global convergence. The algorithm addresses optimization problems of the following form:

$$\begin{aligned} &\min_{x,z} f(x) + g(z), \\ &\text{s.t.} \quad Ax + Bz = c, \end{aligned} \tag{10}$$

where $f(x)$ and $g(z)$ are convex functions, and A, B, c represent the constraint matrices and vector.

The key steps of the algorithm are as follows:

1. Augmented Lagrangian: Define the augmented Lagrangian as:

$$L_\rho(x, z, y) = f(x) + g(z) + y^\top (Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2, \tag{11}$$

where y is the Lagrange multiplier, and $\rho > 0$ is the penalty parameter.

2. Alternating Updates:

As for x-update:

$$x^{(k+1)} = \arg\min_x \left\{ f(x) + \frac{\rho}{2}\|Ax + Bz^{(k)} - c + \frac{1}{\rho}y^{(k)}\|_2^2 \right\}. \tag{12}$$

As for z-update:

$$z^{(k+1)} = \arg\min_z \left\{ g(z) + \frac{\rho}{2}\|Ax^{(k+1)} + Bz - c + \frac{1}{\rho}y^{(k)}\|_2^2 \right\}. \tag{13}$$

As for y-update:

$$y^{(k+1)} = y^{(k)} + \rho(Ax^{(k+1)} + Bz^{(k+1)} - c). \tag{14}$$

3. Iterative Steps: Alternate the updates until convergence.

We concluded the ADAL algorithm stated in the paper [5] as the following pseudocode:

Algorithm 2 Alternating Direction Augmented Lagrangian (ADAL)

```
1: **Initialization:** Choose $x^{(0)}$, $p^{(0)}$, $u^{(0)}$, $\mu > 0$, penalty factor $M_1, M_2 = 1$ and tolerances $\sigma, \sigma'' \geq 0$. Set $k = 0$.
2: **while** stopping criteria not met **do**
3:     **Step 1:** Solve Lagrangian subproblems.
4:     Solve for $x^{(k+1)}$:
           $\min_x \quad L(x, p^{(k)}, u^{(k)})$
5:     Solve for $p^{(k+1)}$:
           $\min_p \quad L(x^{(k+1)}, p, u^{(k)})$
6:     **Step 2:** Update dual variables.
           $u^{(k+1)} = u^{(k)} + \frac{1}{\mu}\left(Ax^{(k+1)} + b - p^{(k+1)}\right)$
7:     **Step 3:** Check stopping criteria.
8:     Stop if $\|x^{(k+1)} - x^{(k)}\|_2 \leq \sigma$ and $\sup_{i \in I}\{|a_i^T x^{(k+1)} + b_i - p_i^{(k+1)}|\} \leq \sigma''$.
9:     Update $k \leftarrow k + 1$.
10: **end while**
```

3.3 Bayesian Optimization

In our experiments, we observed that the selection of M_1 and M_2 plays a critical role in the performance of both IRWA and ADAL, highlighting the importance of constraints in the optimization process. To identify the optimal values for M_1 and M_2, we employ Gaussian Process Bayesian Optimization (GPBO).

GPBO [6] is a powerful probabilistic optimization technique designed to efficiently find the minimum (or maximum) of an expensive, black-box objective function. It combines the flexibility of Gaussian Processes (GP) for modeling the objective function with the decision-making framework of Bayesian Optimization (BO).

For our task, we define the objective function f as the sum of the output from the meta-algorithm and the two constraint losses:

$$\text{loss} = F + \text{loss}_1 \times 1000 + \text{loss}_2 \times 1000,$$

where F represents the result of the meta-algorithm (e.g., ADAL), loss_1 is the loss associated with the constraint $A_1x + b_1 = 0$, and loss_2 corresponds to the loss of the constraint $A_2x + b_2 \leq 0$. Specifically, we define:

$$\text{loss}_1 = \sum_{i \in \mathcal{I}_1} \left\| a_i^T x + b_i \right\|_1,$$

$$\text{loss}_2 = \sum_{i \in \mathcal{I}_2} \max\{a_i^T x + b_i, 0\}.$$

Algorithm 3 Gaussian Process Bayesian Optimization (GPBO)

```
1: **Input:** Initial point $M_0 = \{(M_{1,0}, M_{2,0})\}$, Loss function $loss$
2: **Initialize:**
3:     Place a Gaussian process prior on $f$
4:     Observe $f$ at $M_0$
5:     Set the iteration counter $t = 0$.
6: **while** $t < T$ **do**
7:     **Step 1:** Let $M_t$ be a minimizer of $f$:
           $M_t = \arg\min_M f$
8:     **Step 2:** Evaluate the objective function at $M_t$ to obtain $y_t$:
           $y_t = \text{loss}(M_t)$
9:     **Step 3:** Update the dataset:
           $M_t = M_{t-1} \cup M_t, y_t = y_{t-1} \cup y_t$
       .
10:    **Step 4:** Update the posterior probability distribution on $f$ where $\mu$ is the mean of $y$ and $\sigma$ is
       the covariance matrix of $y$
           $f \sim \mathcal{N}(\mu, \sigma)$
11:    Increment $t = t + 1$.
12: **end while**
13: **Output:** Best solution found: $\hat{M} = \arg\min_M loss$.
```

4 Experiments

Our experiments design refer to the experimental part of [5]: For the matrices in the problem 3, we randomly generated A and satisfies that the first half of A corresponds to equation constraints and the second half corresponds to inequality constraints. Each matrix A is obtained by first randomly choosing a mean and variance from the integers on the interval $[1, 10]$ with equal probability and secondly generating the elements of A according to a normal distribution with the mean and variance chosen before. Similarly, each of the vectors b and g are constructed by randomly choosing integers on the intervals $[-100, 100]$ for the mean and $[1, 100]$ for the variance and generating the elements through normal distribution. The matrix H took the form of $H = 0.1I + LL^T$ where the elements of $L \in \mathbb{R}^{n\times n}$ are chosen from a normal distribution having mean 1 and variance 2, which helps promise H to be semi-positive definite.

For the input superparameters for the IRWA algorithm, we chose $\eta = 0.6$, $M = 10^4$, $\gamma = \frac{1}{6}$, $\epsilon^{(0)} = 2000 \times [1, 1, ..., 1]^T$, $x^{(0)} = [0, 0, ..., 0]^T$, $\sigma = \sigma' = 10^{-5}$. For the ADAL algorithm, we chose $\mu = 1$, $u^{(0)} = [0, 0, ..., 0]^T$, $x^{(0)} = [0, 0, ..., 0]^T$, $\sigma = 10^{-5}$.

4.1 Experiment 1: Effect of Problem Scale on Algorithm Performance

In this experiment, we investigate the impact of problem size on the performance of the ADAL and IRWA algorithms. We first evaluate the base versions of both algorithms with $n = 2, 4, \ldots, 100$ constraints, where the number of equality constraints equals the number of inequality constraints, and x represents the n-dimensional decision space.

The relative error for each algorithm is calculated as follows:

$$\text{Error} = \log\left(\frac{|f - f^*|}{|f^*|}\right)$$

where f is the computed objective value, and f^* is the optimal objective value.

The results are shown in Figure 1. We observed that for ADAL, approximately 100% of the solutions were infeasible, while for IRWA, about 28% of the solutions were infeasible.

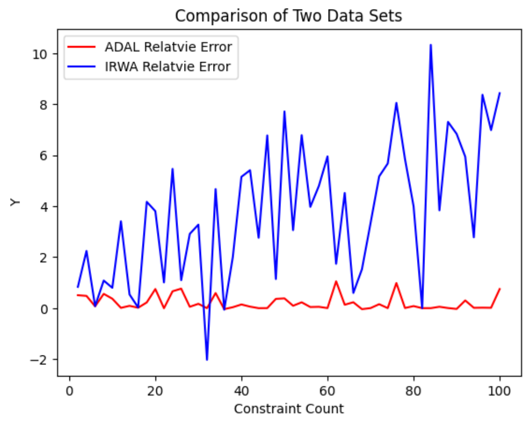

Figure 1: Performance of ADAL and IRWA with increasing problem size.

To address the infeasibility issue, we introduced Bayesian Optimization to determine the optimal penalty for constraint violations. Due to time constraints, the Bayesian version was tested with $n = 2, 4, 8, 16, 32, 64$ constraints. The results are shown in Figure 2.

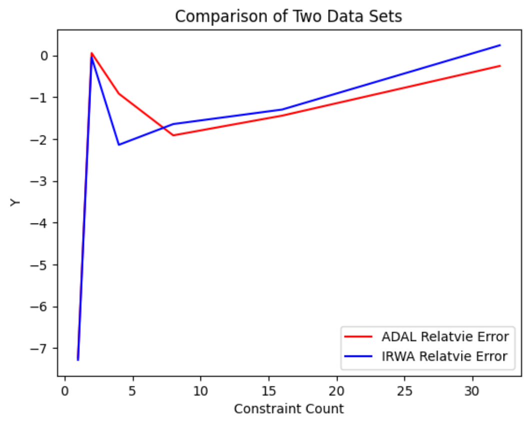

Figure 2: Performance with Bayesian Optimization for constraint handling.

From these results, we again observed that for ADAL, approximately 28% of the solutions remained infeasible, which was consistent with the IRWA performance.

We found that the original versions of the ADAL and IRWA algorithms performed poorly, with IRWA exhibiting greater overall volatility and a larger trend of difference as n increased. However, both the Bayesian-optimized ARAL and IRWA demonstrated improved performance, although the differences still increased as n grew up.

4.2 Experiment 2: parameter sensitivity of IRWA algorithm

In Experiment 2, we examined the sensitivity of the IRWA algorithm's performance with respect to varying values of the penalty parameter, η, across different problem sizes, n. The results, summarized in the Table 1, reveal notable trends in the behavior of the algorithm as η changes.

For smaller problem sizes, such as $n = 2$ and $n = 4$, the performance of the IRWA algorithm remained relatively stable, with objective values hovering around 0.83 and 2.25, respectively, regardless of the variations in η. This suggests that, for these smaller problem sizes, the choice of η does not significantly affect the algorithm's efficiency or outcome.

However, as the problem size increases, the sensitivity to η becomes more pronounced. For $n = 8$, we observe that the performance is fairly stable for small values of η (1e1 and 1e2), but there is a

Table 1: Performance of the algorithm with varying numbers of constraints n and penalty factors η. The table shows the error values for different combinations of n (number of constraints) and η (penalty factor), demonstrating the algorithm's sensitivity to changes in these parameters.

		η				
		1e1	1e2	1e3	1e4	1e5
n	2	0.83	0.83	0.83	0.83	0.83
	4	2.24	2.24	2.24	2.24	2.24
	8	1.04	1.04	3.16	1.017	1.04
	16	1.63	0.06	2.91	2.78	1.29
	32	1.17	1.83	1.40	2.46	2.05
	64	4.19	3.99	5.64	3.57	4.76

sharp increase in the objective value when η reaches 1e3, which later stabilizes as η increases further. This pattern is indicative of potential instability or poor convergence when the penalty term becomes too large. In contrast, for $n = 16$, the performance fluctuates drastically, with the objective value dropping to an unusually low value of 0.06187873 at $\eta = 1e2$, before rising again with higher values of η. This suggests that, for intermediate problem sizes, the algorithm may experience significant variation in performance depending on the chosen penalty parameter.

For larger problem sizes, such as $n = 32$ and $n = 64$, the IRWA algorithm shows more consistent but slightly higher objective values, ranging from 1.17 to 4.77. These results indicate that as the problem scale increases, a larger η might lead to slower or less stable convergence, as evidenced by the increasing variability in performance across different values of η. The algorithm appears to require careful tuning of the penalty parameter to avoid suboptimal performance, particularly as the problem size grows.

In conclusion, the performance of the IRWA algorithm is highly sensitive to the choice of η, with smaller problem sizes exhibiting less sensitivity than larger ones. For larger n, careful tuning of η is crucial to achieve optimal performance and avoid potential instability.

4.3 Experiment 3: parameter sensitivity of ADAL algorithm.

In our third experiment, we investigated the sensitivity of the ADAL algorithm to the adjustment of the penalty parameter μ. The table presents the error values for different values of n (the number of constraints) and varying penalty factors μ (ranging from $1e1$ to $1e5$).

From Table 2, we can observe that the algorithm's performance is highly sensitive to the choice of μ, particularly when the number of constraints is small. For example, with $n = 2$, the error starts at 5.06×10^{-1} for $\mu = 1e1$ and decreases dramatically as μ increases, approaching near-zero values (-4.89×10^{-5}) for $\mu = 1e5$. This trend suggests that larger values of μ improve the algorithm's performance by reducing the error significantly.

For larger values of n, such as $n = 8$ and $n = 16$, the performance improvement becomes less pronounced, but the error still decreases as μ increases. For instance, for $n = 8$, the error improves from 5.60×10^{-1} to -3.00×10^{-3} as μ increases from $1e1$ to $1e5$. A similar trend is observed for $n = 16$, where the error drops from 1.66×10^{-2} to 1.64×10^{-5}.

Interestingly, at $n = 32$, the performance does not improve significantly for larger values of μ. The error stabilizes, suggesting that beyond a certain penalty value, the algorithm's sensitivity to μ diminishes. Finally, for $n = 64$, the performance still improves, but more gradually, with the error decreasing from 1.34×10^{-1} to 1.61×10^{-3} as μ increases.

Overall, the results demonstrate that the ADAL algorithm benefits from increasing values of μ up to a certain point. However, as the number of constraints grows, the improvement becomes less dramatic, suggesting that the penalty parameter μ has diminishing returns in terms of performance for larger problem sizes.

Table 2: Performance of the algorithm with varying numbers of constraints n and penalty factors μ. The table shows the error values for different combinations of n (number of constraints) and μ (penalty factor), demonstrating the algorithm's sensitivity to changes in these parameters.

		μ				
		1e1	1e2	1e3	1e4	1e5
n	2	5.06e-01	-4.76e-02	-4.88e-03	-4.89e-04	-4.89e-05
	4	4.80e-01	1.22e-01	1.15e-01	1.14e-01	1.10e-01
	8	5.60e-01	-1.03e+00	-2.54e-01	-2.95e-02	-3.00e-03
	16	1.66e-02	4.38e-03	1.23e-03	1.59e-04	1.64e-05
	32	-9.72e-04	1.69e-04	1.29e-04	5.80e-05	7.92e-06
	64	1.34e-01	3.63e-02	1.69e-02	5.39e-03	1.61e-03

5 Conclusions

In this study, we explored and implemented two prominent algorithms, the Iterative Reweighting Algorithm (IRWA) and the Alternating Direction Augmented Lagrangian (ADAL), to address quadratic programming (QP) problems characterized by both equality and inequality constraints. Our investigation revealed several key insights into the performance and sensitivity of these algorithms under varying problem scales and parameter settings.

Performance Comparison: Through extensive experimentation, we observed that both IRWA and ADAL exhibit distinct performance profiles as the problem size increases. IRWA demonstrated a higher degree of volatility and a greater tendency for infeasibility in larger-scale problems compared to ADAL. Conversely, ADAL initially suffered from a high rate of infeasibility in smaller problems, which was mitigated through the application of Bayesian Optimization for parameter tuning. This highlights the inherent trade-offs between the two algorithms in terms of scalability and reliability.

Parameter Sensitivity: Our experiments underscored the critical role of penalty parameters (M_1, M_2, and μ) in the convergence and accuracy of both IRWA and ADAL. IRWA showed significant sensitivity to the penalty parameter η, especially as the number of constraints increased, necessitating careful tuning to maintain performance stability. Similarly, ADAL's performance improved consistently with higher values of μ, though the benefits plateaued for larger problem sizes. The successful application of Gaussian Process Bayesian Optimization (GPBO) to identify optimal penalty parameters underscores the potential of advanced hyperparameter tuning techniques in enhancing algorithmic performance.

Bayesian Optimization Integration: Incorporating GPBO proved effective in reducing infeasibility rates and improving objective accuracy for both algorithms. This integration not only facilitated more robust performance across different problem scales but also demonstrated the feasibility of automating parameter tuning processes in optimization algorithms, thereby reducing the manual effort required for calibration.

Implications and Future Work: The findings suggest that while IRWA and ADAL are both viable for solving constrained QP problems, their effectiveness is highly contingent on appropriate parameter settings, which vary with problem size and complexity. Future research could explore adaptive mechanisms for dynamic parameter adjustment, further enhancing the scalability and robustness of these algorithms. Additionally, extending the framework to incorporate other state-of-the-art solvers, such as the recently introduced dQP, could provide a more comprehensive understanding of solver performance in diverse application domains.

In conclusion, this work contributes to the optimization literature by providing a comparative analysis of IRWA and ADAL in the context of constrained QP problems, highlighting the importance of parameter tuning and the benefits of Bayesian Optimization in achieving reliable and accurate solutions. These insights pave the way for more efficient and scalable optimization strategies in various engineering and machine learning applications.

References

[1] PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond. In *RSS 2022 - Robotics: Science and Systems*, New York, United States, 2022.

[2] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[3] Erling D. Andersen and Knud D. Andersen. *The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm*, pages 197–232. Springer US, Boston, MA, 2000.

[4] Connor W. Magoon, Fengyu Yang, Noam Aigerman, and Shahar Z. Kovalsky. Differentiation through black-box quadratic programming solvers, 2024.

[5] James V. Burke, Frank E. Curtis, Hao Wang, and Jiashan Wang. Iterative reweighted linear least squares for exact penalty subproblems on product sets. *SIAM Journal on Optimization*, 25(1):261–294, 2015.

[6] Peter I. Frazier. A tutorial on bayesian optimization, 2018.