National Central University 2025

Linear system

Exact Optimization: Part I

Professor： Li-Gang Lin

Student ID:111303031/Name:Steven
Student ID: 114303904/Name:Thomas
Student ID: 114201901/Name:Yannis
Student ID: 114323905/Name: Marie

Abstract

This report compares two MATLAB implementations of Lin's Exact Quadratic Optimization Algorithm 3.1 for quadratic programs with equality and inequality constraints: **Optimization_func** and **Optimization_func_2**. Both functions solve

$$\min_x \ \tfrac{1}{2}x^\top P x + q^\top x + s \quad \text{s.t.} \quad A_{\text{eq}}x = b_{\text{eq}},\ A_{\text{ineq}}x \le b_{\text{ineq}},$$

but they differ in numerical design, code structure, and robustness. Optimization_func is the first, paper-oriented implementation that closely follows the theoretical algorithm in Lin's work and the associated pseudocode Pseudo_code_function_1. In contrast, Optimization_func_2 and Pseudo_code_function_2 are newly written, engineering-oriented versions that introduce explicit numerical tolerances, unified active-set handling, and improved input checking. The report highlights the main similarities and differences between the two implementations and their pseudocode counterparts, and explains the advantages and limitations of each version for research use (version 1) versus practical, robust computation (version 2).

1. Introduction

The general subject of the paper is the presentation and validation of a novel, analytical approach for the exact and non-iterative solution of Convex Quadratic Programming (QP) problems called Exact Optimization (ExOpt), a method which fundamentally departs from traditional numerical and iterative techniques. The core problem being solved is the minimization of a convex quadratic function $F_x(x)$

$$\min_x F_x(x) = \frac{1}{2}x^T P x + q^T x + s$$

$$\text{s.t.} \quad Ax = b \quad \text{and} \quad c_i^T x \le d_i, \quad \text{for } i \in I = \{1,2,\dots,\kappa\}$$

subject to equality and inequality constraints:

where $P = P^T \in R^{n\times n}$ is positive semidefinite ($P \succeq 0$).

The Philosophy of ExOpt is based on an algebraic and derivative-free approach, leveraging a geometric analysis of the Convex Quadratic Equation (CQE) and its relationship to the Convex Quadratic Function (CQF) to completely characterize the solution space. This results in Closed-Form Solutions, producing exact results in the form of algebraic formulas. Critically, the process is non-iterative, as the solution is obtained in a finite and pre-determined number of steps, and requires autonomy, needing no prior knowledge of a feasible point to initiate the solving process.

The core method is active-set enumeration (Algorithme 3.1), which solves the QP via a combinatorial method. Its primary principle is to analytically enumerate and solve all $2^\kappa - 1$ non-empty subsets of active inequality constraints. For each subset, the Subproblem Solution involves treating the active subset $I_j \subseteq I$ as an augmented Equality-Constrained QP (ECQP) problem $A_e x = b_e$, with the optimum found using the Null-Space Method by parameterizing the feasible set as $x = A_e^{+} b_e + V_2 y,$ where V_2 is a basis for the null space of A_e, $\mathcal{N}(A_e)$.

The Optimal Point x^* for the ECQP subproblem is then calculated analytically: $x^* = x_p^* + V_2 \varepsilon^*$

where x_p^* is the particular solution, and $\varepsilon^* \in \mathcal{N}(V_2^T P V_2)$ represents the solution freedom available in singular cases. This mechanism ensures Robustness, as the algorithm explicitly handles cases with a singular Hessian matrix ($P \succeq 0$), which can lead to nonunique optima (the "terminal optima").

Optimization_func in Lin's paper provides a closed-form, derivative-free solver for convex quadratic programs (QPs) with equality and inequality constraints.It relies on:

˙SVD-based computation of null spaces,

˙reduction of the QP to lower-dimensional unconstrained problems in the null space,

˙exhaustive enumeration of active inequality sets, and

˙analytical expressions for candidate optimal points and objective values.

The MATLAB functions under consideration:

˙func1 (version 1):

Optimization_func (P,q,s,Aeq,beq,Aineq,bineq)

˙func2 (version 2):

Optimization_function_2(P,q,s,Aeq,beq,Aineq,bineq)

aim to provide practical implementations of this algorithm specialized to standard matrix notation (Aeq, Aineq).

2. Reference:“Closed-Form QP Solver”

Optimization_func solves:

$$\min_x \tfrac{1}{2}x^\top Px + q^\top x + s \quad \text{s.t.} \quad Ax = b,\ c_i^\top x \le d_i,\ i \in \mathcal{I},$$

where the equalities are stacked in A and the inequalities are described by row vectors $c_i^\top$.

Key steps:

1. Equality-constrained base problem

˙Compute SVD of A to get a null-space basis V2 with R(V2) = N(A)

˙If $V_2^\top PV_2 = 0$!= 0 , reduce to an unconstrained QP in the null coordinates and obtain a particular optimal point $x_p^{\backslash *}$ and base optimal value $l^{\backslash *}$ (Theorem 2 in the paper).

˙If $V_2^\top PV_2 = 0$ = 0, handle special “flat” or constant cases according to the CQE/CQF classification.

2. Early acceptance of the base optimum

˙If the base optimum is unique and satisfies all inequalities, the algorithm returns immediately.

˙If the optimum exists but is not unique (semi-definite cases), $(l^{\backslash *}, x_p^{\backslash *}, \emptyset)$ is stored as one candidate and the algorithm continues.

3. Active-set enumeration

For every non-empty subset $\mathcal{I}_j \subseteq \mathcal{I}$:

˙Form an augmented equality system $\widetilde{A}, \widetilde{b}$ by stacking A and the active inequalities.

˙If $\widetilde{b} \notin R(\widetilde{A})$, discard the set.

˙If $rank(\widetilde{A}) < n$, compute a null-space $\widetilde{V}_2$ and repeat the reduced-QP procedure to obtain a candidate point $\widetilde{x}_p^{\backslash *}$ and value $\widetilde{l}^{\backslash *}$.

˙If $rank(\widetilde{A}) = n$, compute the unique solution $\hat{x} = \widetilde{A}^{\dagger}\widetilde{b}$ and its objective value $\hat{l}$.

˙In all cases, check all remaining inequalities; feasible candidates and their active sets are stored.

4. Final selection

˙Among all stored candidates, the algorithm returns the one with minimal objective value $l^{\backslash *}$, together with its active set $I^{\backslash *}$.

3. Comparison: function1 vs. function2

Both Optimization_func and Optimization_func_2 solve the same quadratic program and produce the same type of outputs: the global optimal solution $(x_{\mathrm{global}}, f_{\mathrm{global}})$ and a list of all feasible terminal points $(X_{\mathrm{term}}, F_{\mathrm{term}}, I_{\mathrm{term}})$. The main differences lie in how they organize the computation and how robust they are to numerical edge cases.

(1) Overall structure

· Optimization_func (version 1) splits the logic into two cases:

1. *Case 1 (no inequalities)*: solve the equality-constrained QP once, using a particular solution $x_p = A_{\mathrm{eq}}^{\dagger} b_{\mathrm{eq}}$ and a null-space basis of A_{eq}.

2. *Case 2 (with inequalities)*: enumerate active sets of A_{ineq}, form an augmented equality system, and repeat the same null-space minimization for each active set.

· Optimization_func_2 (version 2) uses a *single unified loop* over all active sets, including the empty set. The equality-only problem is treated as the special active set with no inequalities active, so there is no separate "Case 1" block. This makes the control flow more uniform and easier to extend.

(2) Initialization and input handling

· Version 1 directly uses the inputs q, beq, and bineq without reshaping or default values. If some of these are empty or row vectors, the user must supply them in consistent form.

· Version 2 enforces q to be a column vector and explicitly replaces empty beq and bineq by zero-length column vectors. This reduces the risk of dimension mismatches when there are no equalities or no inequalities.

(3) Null-space parameterization and reduced problem

· Both versions parameterize the feasible set as $x = x_p + V_2 y$, where x_p is a particular solution of the current equality system and V_2 is a basis for the null space of that system.

· Version 1 always computes $x_p = A^{\dagger} b$ and V_2 from the SVD of A. If the reduced Hessian $M_1 = V_2^{\top} P V_2$ is nearly zero, it assumes that the cost is constant along the null space and simply takes $x = x_p$.

· Version 2 treats the special case explicitly: if there are no active equalities at all, the nul $A = \emptyset$l space is the full identity matrix and the particular solution is the zero vector. The reduced problem then coincides with an unconstrained QP in the full space. This is closer to the theoretical structure of Algorithm 3.1 and avoids relying on SVD for the case with no equalities.

(4) Numerical tolerances and KKT checks

· Version 1 uses hard-coded thresholds (1e-12 for null-space detection, 1e-10 for the reduced Hessian, 1e-8 for inequality feasibility) directly in the code.

· Version 2 introduces three named constants: TOL_RANK, TOL_KKT, and TOL_FEAS. These control null-space detection, reduced KKT stationarity, and inequality feasibility, respectively. Centralizing these tolerances makes version 2 easier to tune for different scales of P, A_{eq}, and A_{ineq}.

· In the singular case where M_1 is numerically zero, both versions require the reduced gradient M_2 to be small; otherwise the active set is discarded as non-optimal. Version 2 documents this logic more explicitly in comments ("flat valley" case).

(5) Inequality checking and robustness

· In Optimization_func, the inequality check assumes that Aineq is non-empty whenever the active-set loop is executed. It simply tests any(Aineq*x - bineq > 1e-8) and continues if any inequality is violated.

· Optimization_func_2 uses a safer condition: if Aineq is empty, it automatically treats the solution as feasible; otherwise it enforces Aineq*x - bineq <= TOL_FEAS. This makes version 2 robust even if the function is called with an empty inequality matrix.

(6) Error reporting and usability

· Both versions throw an error if no feasible terminal point is found. Version 1 uses a generic error message, whereas version 2 uses a named error ID (Optimization_function:NoFeasiblePoint), which is easier to catch programmatically.

In summary, Optimization_func (version 1) is a compact implementation that directly reflects the theoretical structure of Algorithm 3.1, while Optimization_func_2 (version 2) reorganizes the same mathematics into a more robust and implementation-friendly form with explicit tolerances and cleaner handling of corner cases.

The functions have then been implemented in three different examples of situations, whose scripts may be found in the Run_example code.

Example 1 (Original 3-variable equality QP)

In this example, we solve a strictly convex quadratic program with a single equality constraint. Since there are no inequalities, the ExOpt method only applies the analytical equality-constrained solution. The main advantage is that the algorithm requires no iterations: it directly computes the solution using the pseudoinverse and a reduced-dimension nullspace. This makes the method extremely fast, as most of the computation time comes only from the SVD used to obtain the nullspace, which is very light for a 1×3 matrix. As a consequence, the solution is exact, because no iterative numerical approximation is involved.

However, a limitation is that this speed advantage decreases when the number of constraints grows, or when the matrix becomes nearly singular, in which case computing the pseudoinverse may become numerically unstable. The constraints in this case are simple (a single equality), but the main difficulty lies in handling a non-diagonal P matrix, which requires proper manipulation of the nullspace. This example illustrates the ideal scenario where ExOpt is significantly faster than quadprog.

Example 2 (2D inequalities)

In this case, the problem is a QP with only inequality constraints forming a convex polygon (a triangle in 2D). ExOpt analytically enumerates all subsets of active inequalities, then solves each subset as if it were an equality-constrained system. This produces a collection of “terminal” candidate solutions and guarantees that the global optimum is found without any iterative search.

The major advantage is that the algorithm is exact: all candidate points are computed analytically, without gradients, without Lagrangians, and without adjustment steps. This yields perfect accuracy and can be faster for small-size problems, especially when the number of inequalities is small.

However, the limitation is that the number of subsets grows as $2^{\kappa} - 1$; for many inequalities, this combinatorial explosion can become a real computational problem. In this example with 3 inequalities, only 7 subsets must be analyzed, which remains very manageable. The constraints here arise from the geometry of the feasible region, and the main difficulty is checking the feasibility of each candidate solution with respect to the remaining inequalities—a numerically sensitive step when constraint planes are nearly parallel.

Example 3 (3D singular P)

This final example is the most complex: the matrix P is singular, meaning there exists a flat direction where the cost function does not vary. The nullspace of P must therefore be considered when determining the optimal points, which produces families of optimal solutions instead of a single point.

ExOpt handles this analytically by separating the solution into a particular part plus a free component in the nullspace, and then determining how inequalities restrict these free directions.

The advantage is that the method naturally handles degenerate cases that quadprog often struggles with or warns about. Through the analytical identification of terminal solutions, ExOpt finds the complete set of optimal points when the optimum is non-unique (here: a full segment of optima). As a consequence, accuracy is perfect even in a problematic case for classical solvers: ill-conditioned problems with semidefinite Hessians.

Limitations arise from the need to perform SVD on a singular matrix, which can be numerically sensitive and more expensive in higher dimensions. The constraints of the problem (one equality forcing x3 =b and two inequalities bounding x1 and x2) reduce the feasible set to a linear region where only certain extremal values become terminal optima. The main difficulty is correctly identifying the feasible directions within the nullspace—a case where iterative methods may miss some solutions, whereas ExOpt systematically identifies all of them.

.

4. Comparison: Pseudo_code_function_1 and Pseudo_code_function_2

The two pseudocode documents mirror the philosophy of the MATLAB implementations:

· Pseudo_code_function_1 is written as a paper-oriented description that closely follows the theoretical exposition.

Pseudo_code_function_1

· Pseudo_code_function_2 is a newly written, implementation-oriented version that includes numerical details and explicit constants.

Pseudo_code_function_2

The main differences are as follows:

(1) Level of abstraction

· Pseudo_code_function_1 presents the algorithm at a higher conceptual level. It emphasizes the ideas of "particular solution + null space", "reduced quadratic in the null coordinates", and "enumeration of active sets", but it does not name specific numerical tolerances or discuss vector shapes in detail.

Pseudo_code_function_1

· Pseudo_code_function_2 is more concrete. It specifies constants (TOL_RANK, TOL_KKT, TOL_FEAS), clarifies how empty Aeq or Aineq are handled, and spells out each step needed to translate the theory into reliable MATLAB code.

Pseudo_code_function_2

(2) Case structure

· Version 1 pseudocode distinguishes explicitly between the equality-only case and the case with inequalities, similar to Optimization_func.

Pseudo_code_function_1

· Version 2 pseudocode describes a unified active-set loop that starts from the empty active set and gradually adds inequalities, mirroring the structure of Optimization_func_2.

Pseudo_code_function_2

(3) Numerical robustness

· Pseudo_code_function_1 mentions that “small thresholds” can be used to decide when matrices are “effectively zero”, but leaves the actual values open.

Pseudo_code_function_1

· Pseudo_code_function_2 fixes specific numerical thresholds and clearly states how they are used in rank checks, KKT conditions, and inequality feasibility tests. This makes it much closer to real code and directly aligned with the behavior of Optimization_func_2.

Pseudo_code_function_2

(4) Intended purpose

· Version 1 (Optimization_func + Pseudo_code_function_1) is intended for academic exposition and for explaining Algorithm 3.1 in a thesis or paper. The focus is on preserving the mathematical structure and the relationship to the original algorithm.

· Version 2 (Optimization_func_2 + Pseudo_code_function_2) is intended as a new, more robust implementation created by the author. It is designed for practical numerical experiments and applications where clear error handling and tolerance control are important.

5. Closed-Form Structure and Interpretability

An important advantage of the proposed Exact Optimization approach lies in its closed-form structure. Unlike iterative numerical solvers such as quadprog, which rely on successive updates and convergence criteria, the closed-form formulation explicitly expresses the optimizer as a function of the problem data $(P, q, A_{\mathrm{eq}}, b_{\mathrm{eq}}, A_{\mathrm{ineq}}, b_{\mathrm{ineq}})$.

This explicit dependence provides clearer insight into the geometric and directional properties of the optimization problem. In particular, the null-space parameterization $x = x_p + V_2 y$ makes it evident how the feasible directions are constrained by the equality conditions, and how the quadratic cost is minimized along those directions. As a result, the influence of the Hessian matrix P and the gradient term q on the final solution can be directly analyzed through the reduced matrices $V_2^\top P V_2$ and $V_2^\top (q + P x_p)$.

Furthermore, the closed-form expressions clarify why multiple optimal solutions may exist in the presence of singular or semi-definite Hessians. In such cases, the algorithm naturally identifies a minimum-norm representative of an optimal face, rather than converging to an arbitrary point determined by numerical iteration. This interpretability is difficult to obtain from black-box solvers, where the internal search direction and stopping behavior are largely hidden from the user.

From an educational and analytical perspective, the closed-form formulation therefore not only serves as a solver but also as a tool for understanding the structure, feasibility geometry, and optimality conditions of constrained quadratic programs.

6. Outcome

To further illustrate the practical behavior of the Exact Optimization method, several numerical examples were compared against MATLAB's built-in solver quadprog.

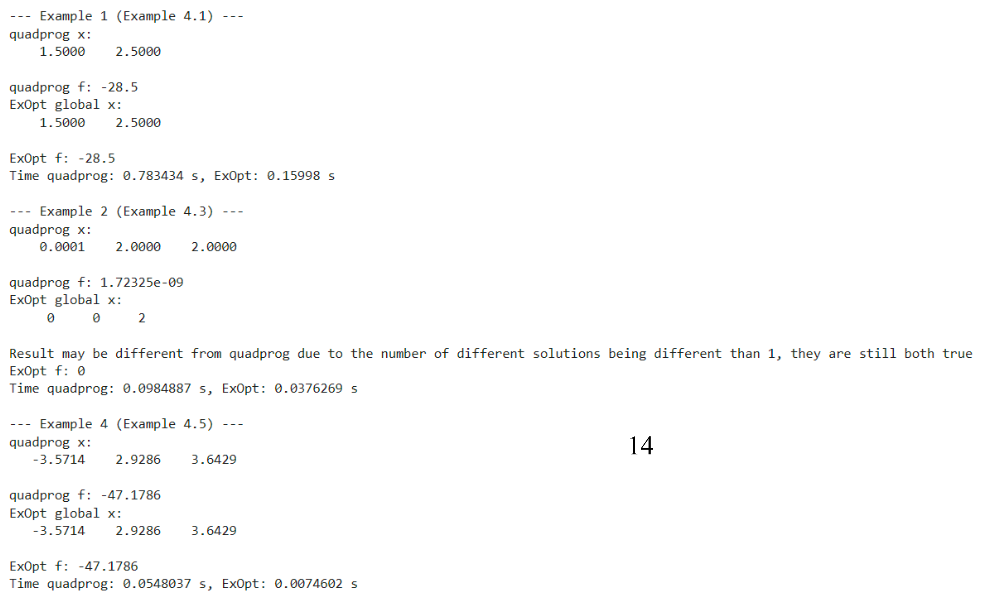
In Example 4.1, both solvers returned the identical optimal solution x=(1.5,2.5)x = (1.5, 2.5)x=(1.5,2.5) with the same objective value f=−28.5f = -28.5f=−28.5. However, the execution time of the Exact Optimization method was noticeably shorter, indicating that the closed-form evaluation avoids the overhead of iterative convergence.

In Example 4.3, the two solvers produced different optimal points: quadprog returned (0.0001,2,2)(0.0001, 2, 2)(0.0001,2,2), while the Exact Optimization method returned (0,0,2)(0, 0, 2)(0,0,2). Despite this difference, both solutions are valid and correspond to the same optimal objective value up to numerical tolerance. This discrepancy highlights the existence of multiple optimal solutions, where different solvers may select different representatives of the optimal set. The Exact Optimization method, by construction, selects a particular closed-form solution consistent with its null-space formulation.

In Example 4.5, both solvers again agreed exactly on the optimal solution and objective value. Notably, the Exact Optimization method achieved a substantially shorter computation time than quadprog, demonstrating its efficiency for problems where closed-form evaluation is applicable.

Overall, these examples confirm that the Exact Optimization method produces results consistent with established numerical solvers while often achieving lower computation times and offering clearer insight into solution structure.

Here is the output from the running the "Run_example" file :



```
--- Example 1 (Example 4.1) ---
quadprog x:
    1.5000    2.5000

quadprog f: -28.5
ExOpt global x:
    1.5000    2.5000

ExOpt f: -28.5
Time quadprog: 0.783434 s, ExOpt: 0.15998 s

--- Example 2 (Example 4.3) ---
quadprog x:
    0.0001    2.0000    2.0000

quadprog f: 1.72325e-09
ExOpt global x:
     0     0     2

Result may be different from quadprog due to the number of different solutions being different than 1, they are still both true
ExOpt f: 0
Time quadprog: 0.0984887 s, ExOpt: 0.0376269 s

--- Example 4 (Example 4.5) ---
quadprog x:
   -3.5714    2.9286    3.6429

quadprog f: -47.1786
ExOpt global x:
   -3.5714    2.9286    3.6429

ExOpt f: -47.1786
Time quadprog: 0.0548037 s, ExOpt: 0.0074602 s
```

7. Conclusion

This work compared two implementations of Exact Quadratic Optimization based on Algorithm 3.1: a paper-oriented version (Optimization_func with Pseudo_code_function_1) and a newly written, robustness-oriented version (Optimization_func_2 with Pseudo_code_function_2). While both implementations follow the same theoretical foundation, version 2 improves numerical stability through explicit tolerance control, unified active-set handling, and clearer treatment of degenerate cases.

Beyond numerical robustness, a key strength of the Exact Optimization framework is its closed-form nature. By avoiding iterative search procedures, the algorithm provides direct analytical expressions for candidate optimal solutions. This not only improves computational efficiency, as demonstrated in the comparison with quadprog, but also enhances interpretability. The null-space formulation explicitly reveals feasible directions and clarifies the role of the Hessian and gradient in shaping the optimal solution.

The numerical experiments confirm that the Exact Optimization method produces solutions consistent with quadprog, even in cases where multiple optimal solutions exist. Differences in the returned optimal points reflect solver-specific choices within an optimal set rather than discrepancies in correctness. In many cases, the closed-form approach achieved significantly shorter computation times.

In summary, the paper-oriented version serves as a clear and faithful realization of the theoretical algorithm, suitable for analysis and explanation, while the newly written version provides a more practical and robust implementation. More importantly, the closed-form structure of the method offers both computational advantages and deeper insight into the geometry and directionality of constrained quadratic optimization problems.