UNIVERSITÀ DI PISA

Master Degree in Data Science and Business Informatics

Optimization for Data Science Project

A Python Implementation of the Active-Set Method

Steffania Sierra (s.sierragalvis@studenti.unipi.it)
Alla Usova (a.usova@studenti.unipi.it)

Academic year 2023-2024

Contents

1 Introduction **2**

2 Problem Description **2**

3 Active-Set Method Description **3**

3.1 Finding the Direction p and the Lagrange Multipliers 5

3.1.1 Positive definite case 5

3.1.2 Positive semidefinite case 7

3.2 Convergence and Complexity 7

3.3 Implementation of the algorithm 8

4 Experiments **9**

4.1 Data generation and solvers selection 9

4.2 Results 11

1 Introduction

This work presents the application of the primal active-set method, as described in [3], to a specific quadratic problem. The paper is structured in three sections. In the first section, we define the problem to be solved. The second section describes the active-set method tailored to exploit the problem's structure. We explain how to solve the Karush–Kuhn–Tucker (KKT) system required to compute the search direction and Lagrange multipliers, and we analyze the algorithm's convergence and computational complexity. Additionally, we include a pseudocode representation of the algorithm and an example of its implementation. Finally, in the last section, we present experimental results obtained by applying the algorithm to various randomly generated quadratic problems.

2 Problem Description

Consider $Q \in \mathcal{M}_{n\times n}(\mathbb{R})$ a positive semidefinite matrix and $c \in \mathbb{R}^n$. Consider a set K of disjoint simplices in $\mathbb{R}^n$, and for $h \in K$ the index set I^h forming a partition of $\{1, \cdots, n\}$. Consider the convex quadratic program

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) = x^TQx + x^Tc \\ \text{subject to} \quad & \sum_{i\in I^h} x_i = 1, \\ & x \geq 0. \end{aligned} \tag{1}$$

Since $x^TQ^Tx = x^TQx \in \mathbb{R}$, we have that

$$x^T\left(\frac{Q+Q^T}{2}\right)x = x^TQx + x^T\left(\frac{Q^T-Q}{2}\right)x = x^TQx.$$

This is useful because $\frac{Q+Q^T}{2}$ is symmetric, so we may assume without loss of generality that the objective function can be written as $f(x) = x^TQx + x^Tc$ with Q symmetric. Multiplying $f(x)$ by $\frac{1}{2}$, we can rewrite our quadratic problem as

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) = \frac{1}{2}x^TQx + x^Tc \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{2}$$

with Q symmetric positive semidefinite, $q = \frac{1}{2}c \in \mathbb{R}^n$, $A \in \mathcal{M}_{k\times n}(\mathbb{R})$ is such that each column is a standard-basis vector of $\mathbb{R}^k$, with k the cardinality of the partition K, and $b = [1, \ldots, 1]^T \in \mathbb{R}^k$.

Consider the Lagrange function for this problem:

$$\mathcal{L}(x, \lambda, \mu) = \frac{1}{2}x^TQx + x^Tq - \lambda^T(Ax - b) - \mu^Tx.$$

The first-order necessary conditions can be stated as

$$
\begin{aligned}
Qx + q - A^T\lambda - \mu &= 0 && (3)\\
Ax - b &= 0 && (4)\\
x &\geq 0 && (5)\\
\mu &\geq 0 && (6)\\
x^T\mu &= 0. && (7)
\end{aligned}
$$

Notice that any feasible point x of the problem (2) can be written as $x = (x_L, x_U)$ where $L = \{i \in \{1, \dots, n\} : x_i = 0\}$ and $U = L^C$. We denote the cardinality of L and U by l and u respectively. Similarly, Q, A and q can be partitioned by blocks as follows:

$$
Q = \begin{bmatrix} Q_{LL} & Q_{LU} \\ Q_{UL} & Q_{UU} \end{bmatrix}, \qquad A = \begin{bmatrix} A_L & A_U \end{bmatrix}, \qquad q = \begin{bmatrix} q_L \\ q_U \end{bmatrix}.
$$

Using this block partition, we rewrite the problem (2) as

$$
\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & f(x) = \frac{1}{2} x_U^T Q_{UU} x_U + x_U^T q_U \\
\text{subject to} \quad & A_U x_U = b, \\
& x_U \geq 0.
\end{aligned} \tag{8}
$$

The vector x_U is a solution of problem (8) if there is a vector λ of Lagrange multipliers such that

$$
\begin{bmatrix} Q_{UU} & A_U^T \\ A_U & 0 \end{bmatrix} \begin{bmatrix} x_U \\ \lambda \end{bmatrix} = \begin{bmatrix} -q_U \\ b \end{bmatrix}. \tag{9}
$$

This system of equations for the equality constraint program is known as the Karush – Kuhn – Tucker (KKT) system.

3 Active-Set Method Description

Various methods have been proposed to solve quadratic problems, including the active-set methods. The *active set* $\mathcal{A}(x)$ of a convex quadratic problem at the point x is defined as the set of indices of constraints that are satisfied as equalities at x. In our problem, the active variables are the ones with an index in L.

Active-set methods identify the active set at each iteration and operate in two phases. In the first phase, a feasible point is found without considering the objective function. In the second phase, the objective function is minimized within the feasible region by solving quadratic subproblems, ensuring all equality constraints are met, and treating some inequality constraints as equalities.

For convex quadratic problems, there are three primary active-set methods: *primal*, *dual*, and *coupled primal-dual* methods. In this work, we utilize the primal method as described in [3].

At iteration k, assume that x_U^k does not minimize the objective function f in the subspace defined by U_k. To find the next feasible point, we have to find the direction p_U^k in which we will move and the length α_k of the step. Replacing $x_U^k + p_U^k$ in the objective function we get,

$$\begin{aligned} f(x_U^k + p_U^k) &= \frac{1}{2}(x_U^k + p_U^k)^T Q_{U_k U_k}(x_U^k + p_U^k) + (x_U^k + p_U^k)^T q_U \\ &= \frac{1}{2}{p_U^k}^T Q_{U_k U_k} p_U^k + q_U^T p_U^k + {x_U^k}^T Q_{U_k U_k} p_U^k + \frac{1}{2}{x_U^k}^T Q_{U_k U_k} x_U^k + {x_U^k}^T q_U \\ &= \frac{1}{2}{p_U^k}^T Q_{U_k U_k} p_U^k + {g_U^k}^T p_U^k + f(x_U^k), \end{aligned}$$

with $g_U^k = Q_{U_k U_k} x_U^k + q_U$. Putting $h_U^k = A_{U_k} x_U^k - b$, the vector p_U^k can be found by solving the equality-constrained quadratic problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \frac{1}{2}{p_U^k}^T Q_{U_k U_k} p_U^k + {g_U^k}^T p_U^k \\ \text{subject to} \quad & -A_{U_k} p_U^k = h_U^k, \\ & x_{U_k} \geq -p_U^k \end{aligned}$$

and the corresponding KKT system

$$\begin{bmatrix} Q_{UU} & A_U^T \\ A_U & 0 \end{bmatrix} \begin{bmatrix} -p_U^k \\ \lambda \end{bmatrix} = \begin{bmatrix} g_U^k \\ h_U^k \end{bmatrix}. \tag{10}$$

Since our active variables already satisfy the inequality constraints, by setting $p_L^k = 0$ we ensure that its value does not change when we move along p^k. Assume that $p_U^k \neq 0$. We need to decide how far to move along this direction. If $x_U^k + p_U^k$ is feasible with respect to all the constraints, then we set

$$x^{k+1} = x^k + p^k.$$

Otherwise, we need to find the largest step-length parameter $\alpha_k \in (0, 1]$ for which all the constraints are satisfied, and set

$$x^{k+1} = x^k + \alpha_k p^k.$$

If $p_i^k \geq 0$ for some $i \in U$, then for all nonnegative choices of the step-length parameter, the constraint $x_i^k + \alpha_k p_i^k \geq 0$ is satisfied. Whenever $p_i^k < 0$ for some $i \in U$, we have that $x_i^k + \alpha_k p_i^k \geq 0$ only if

$$\alpha_k \leq \frac{-x_i^k}{p_i^k}.$$

Therefore, α_k is defined as

$$\alpha_k := \min_{i\in U,\, p_i^k<0} \frac{-x_i^k}{p_i^k}.$$

The constraints for which the minimum in the definition of α_k is reached are called *blocking constraints* when $\alpha_k < 1$. The new active set L_{k+1} is formed by adding one of the blocking constraints to L_k until the point that minimizes the quadratic objective function over its current active set is reached.

Regarding the sign of the multipliers corresponding to the inequality constraints in the working set, if these multipliers are all nonnegative, the current point is a KKT point for the original problem. Since Q is positive semidefinite, the current point is a global solution of the problem (2). Otherwise, if at least one of the Lagrange multipliers μ is negative, the KKT conditions are not satisfied, and the objective function may be decreased by dropping this constraint.

3.1 Finding the Direction p and the Lagrange Multipliers

To simplify the notation in this section, the subscripts U and UU will be replaced with a hat symbol, for example, $Q_{UU} = \hat{Q}$, and $A_U = \hat{A}$.

3.1.1 Positive definite case

To compute the Lagrange multipliers in the KKT system (10) we will use the *null-space method* described in [3] which does not require $\hat{Q}$ to be invertible.

We start by assuming that $\hat{A}$ has full row rank and that $Z^T\hat{Q}Z$ is positive definite with Z the null-space basis matrix of $\hat{Q}$.

Suppose that we partition the vector $\hat{p}$ into two components as follows:

$$\hat{p} = Yp_Y + Zp_Z$$

where Y is any matrix such that $[Y|Z]$ is invertible. We can find p_Y using the second equation of the system (10) and recalling that $\hat{A}Z = 0$:

$$-\hat{h} = \hat{A}(Yp_Y + Zp_Z) = (\hat{A}Y)p_Y + (\hat{A}Z)p_Z = (\hat{A}Y)p_Y. \tag{11}$$

Since $\hat{A}$ has full rank and $[Y|Z]$ is invertible, the product $\hat{A}[Y|Z] = [\hat{A}Y|0]$ has full rank. Therefore, $\hat{A}Y$ is invertible and p_Y can be solved by the previous equation.

To determine p_Z we start by substituting $\hat{p}$ into the first equation of the system (10) to obtain

$$\hat{g} = -\hat{Q}\hat{p} + \hat{A}^T\lambda = -\hat{Q}Yp_Y - \hat{Q}Zp_Z + \hat{A}^T\lambda$$

and multiplying by Z^T to obtain

$$(Z^T\hat{Q}Z)p_Z = -Z^T\hat{Q}Yp_Y - Z^T\hat{g}. \tag{12}$$

Since we assumed $Z^T\hat{Q}Z$ positive definite the previous system can be solved by using Cholesky factorization.

Finally, the Lagrange multiplier can be obtained by solving the linear system obtained after multiplying the first equation of (10) by Y^T,

$$(\hat{A}Y)^T\lambda = Y^T(\hat{g} + \hat{Q}\hat{p}). \tag{13}$$

To find the decomposition of $\hat{p}$ we need to find the matrices Y and Z. We start by rearranging the columns of $\hat{A} \in \mathcal{M}_{k\times u}(\mathbb{R})$ such that we find a linearly independent subset of k columns. If we define an $u \times u$ permutation matrix P that swaps the independent columns to the first k columns of $\hat{A}$, we can write

$$\hat{A}P = \begin{bmatrix} \hat{A}_Y & \hat{A}_Z \end{bmatrix}, \qquad P^T\hat{p} = \begin{bmatrix} \hat{p}_Y \\ \hat{p}_Z \end{bmatrix},$$

where the subvectors $\hat{p}_Y \in \mathbb{R}^k$ and $\hat{p}_Z \in \mathbb{R}^{u-k}$. Since $PP^T = I$, we can rewrite the first equation in system (10) as

$$-\hat{h} = \hat{A}\hat{p} = \hat{A}P(P^T\hat{p}) = \begin{bmatrix} \hat{A}_Y & \hat{A}_Z \end{bmatrix} \begin{bmatrix} \hat{p}_Y \\ \hat{p}_Z \end{bmatrix} = \hat{A}_Y\hat{p}_Y + \hat{A}_Z\hat{p}_Z.$$

From this, $\hat{p}_Y$ can be expressed as follows:

$$\hat{p}_Y = -\hat{A}_Y^{-1}\hat{h} - \hat{A}_Y^{-1}\hat{A}_Z\hat{p}_Z.$$

This allow us to write any feasible point p for the linear constraints as $\begin{bmatrix} \hat{p}_Y \\ \hat{p}_Z \end{bmatrix} = -Y\hat{h} + Z\hat{p}_Z$, with $Y = \begin{bmatrix} \hat{A}_Y^{-1} \\ 0 \end{bmatrix} \in \mathcal{M}_{u\times k}(\mathbb{R})$ and $Z = \begin{bmatrix} -\hat{A}_Y^{-1}\hat{A}_Z \\ I \end{bmatrix} \in \mathcal{M}_{u\times(u-k)}(\mathbb{R})$.

The matrix Z has $u-k$ linearly independent columns and satisfies the equation

$$\hat{A}PZ = -\hat{A}_Y\hat{A}_Y^{-1}\hat{A}_Z + \hat{A}_Z = 0.$$

Therefore, Z is a basis for the null space of $\hat{A}P$. In addition, $\begin{bmatrix} PY & PZ \end{bmatrix}$ is non-singular, and $\hat{A}P$ is a full rank matrix, which implies that $\hat{A}PY$ is non-singular.

The constraints of our problem are disjoint simplices, allowing us to set $\hat{A}_Y = I$ and thus simplifying the computation of Y and Z:

$$Y = \begin{bmatrix} I \\ 0 \end{bmatrix}, \qquad Z = \begin{bmatrix} -A_Z \\ I \end{bmatrix}. \tag{14}$$

Equation (11) simplifies as

$$\hat{A}PY\hat{p}_Y = \begin{bmatrix} I & \hat{A}_Z \end{bmatrix} \begin{bmatrix} I \\ 0 \end{bmatrix} \hat{p}_Y = \hat{p}_Y = -\hat{h}.$$

From equation (13) and $\hat{A}PY = I$, we have that the Lagrange multipliers λ of the equalities can be found as

$$(\hat{A}PY)^T\lambda = \lambda = (PY)^T(\hat{g} + \hat{Q}\hat{p}). \tag{15}$$

Note that substituting the new Y and Z matrices does not sufficiently simplify the system (12) to find $\hat{p}_Z$. Therefore, we will solve the linear system using the Cholesky factorization of the matrix $(PZ)^T\hat{Q}(PZ)$. Finally, we obtain $\hat{p}$ by multiplying the vector $\begin{bmatrix} \hat{p}_Y & \hat{p}_Z \end{bmatrix}^T$ by P.

3.1.2 Positive semidefinite case

When $Z^T\hat{Q}Z$ is not positive definite, the Cholesky factorization of $Z^T\hat{Q}Z$ cannot be computed. Instead, a step is taken in an unbounded direction of the quadratic problem (8). This direction $\hat{p}$ satisfies

$$\hat{Q}\hat{p} = 0, \qquad \hat{A}\hat{p} = 0, \qquad q^T\hat{p} \neq 0, \qquad (\hat{Q}\hat{x} + \hat{q})\hat{p} < 0. \tag{16}$$

We will use Singular Value Decomposition (*scipy.linalg.null_ space* method) to return an orthonormal basis Z for the null space of the matrix of size $(u + k) \times u$ obtained from concatenating vertically $\hat{A}$ and $\hat{Q}$. We will then project q onto the null space Z, take $\hat{p}$ as the linear combination of the vectors of the basis obtained after the projection, and scale and align it opposite to the gradient to ensure that $\hat{p}$ is a descent direction.

3.2 Convergence and Complexity

Since the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is a quadratic polynomial, it is a continuous function. Moreover, the feasible region is compact in $\mathbb{R}^n$. Indeed, the intersection of the hyperplanes defined by the equality constraints is a closed set and because of the inequality constraints, it is also bounded. Therefore, the image of the feasible region by f is also compact, and so the problem (8) has a global solution.

If we suppose that for all subsets U the KKT system (9) has a solution such that for all $i \in U$, $\lambda_i \neq 0$, then the sequence of points generated by the active set strategy converges to the solution of the problem. This is because after finding the solution corresponding to the active set, the objective function is reduced and it is impossible to return to the working set. So, the finiteness of the quantity of active sets implies that the algorithm can iterate only a finite number of times.

Let n be the number of decision variables, $k + n$ be the number of constraints, and u the number of non-active variables. Each iteration involves solving a system of linear equations restricted to the non-active variables. This will be done using the *null space method.* The computation of Y and Z can be done in $2n$ operations. The matrices P, Y and Z are sparse matrices with less than or equal to n, k, and n non-zero entries at each iteration. Then, the computation of PY and PZ has a complexity of $n + k$ and $2n$ respectively. After finding Y and Z, we have to do multiplication of matrices to find $\hat{p}_Y$ and λ, and to solve a linear system to find $\hat{p}_z$. Therefore, the total time complexity of the precomputations is in $O(n^3)$.

When Q is positive semidefinite and of low rank, it may happen that $ZQ_{UU}Z^T$ is singular. To find the unbounded direction $\hat{p}$ we compute a basis for the row space of Q_{UU} and A_U by using the *scipy.linalg.null_ space* method that has a computational cost of order $O((u + k)u^2)$. Since u and k are less or equal than n, the computational complexity is in $O(n^3)$.

Finally, the number of iterations in the active-set method can be exponential in the number of constraints ([1], Section 9.2, page 249). Therefore, in the worst-case scenario, the total computational complexity of the active set method applied to our problem is in $O(n^3 \times 2^{k+n})$.

3.3 Implementation of the algorithm

After describing the active-set algorithm for our convex quadratic problem, we will demonstrate its implementation starting with an example followed by the pseudocode tailored to our specific problem.

Consider the 3-dimensional quadratic programming problem

$$\begin{aligned} \min_{x\in\mathbb{R}^3} \quad & f(x) = 3x_1^2 + 2x_1x_2 + x_1x_3 + 2.5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3, \\ \text{subject to} \quad & x_1 + x_2 + x_3 = 1, \\ & x \geq 0. \end{aligned}$$

We can write this problem in the form (2) by defining

$$Q = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix}, \qquad q = \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix}, \qquad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \qquad b = 1.$$

Consider the feasible initial point $x^0 = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}^T$. All variables are active at this point, so we set $L_0 = \emptyset$, $U_0 = \{1, 2, 3\}$. We need to find first the direction in which to move, p_{U_0}, by decomposing it as the sum of the vectors $Y_0 p_{Y_0}$ and $Z_0 p_{Z_0}$. Notice that we can take the identity matrix as the permutation matrix P_0. From Equation (14), we have

$$A_{U_0} = [\underbrace{1}_{A_{Y_0}} \ \underbrace{1 \ \ 1}_{A_{Z_0}}], \qquad Y_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad Z_0 = \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

By using A_{U_0} and Y_0 in Equation (11) and Y_0, Z_0, and p_{Y_0} in Equation (12), we find that

$$p_{Y_0} = 0, \qquad p_{Z_0} = \begin{bmatrix} -0.76 & -0.09 \end{bmatrix}^T.$$

Then

$$p_{U_0} = Y_0 p_{Y_0} + Z_0 p_{Z_0} = \begin{bmatrix} 0.85 & -0.76 & -0.09 \end{bmatrix}^T.$$

Using the initial point and the direction found, $x_U = x_{U_0} + p_{U_0} = \begin{bmatrix} 1.1 & -0.26 & 0.16 \end{bmatrix}^T$. Since one of the components is negative, the next iteration is not feasible, so we have to find the length of the step to move along p^0. Following the steps 21 - 26 in the pseudocode, we find that

$$p^0 = p_{U_0}, \quad \alpha_0 = \min_{\{i\in U_0,\, p_i^0<0\}} \frac{-x_i^0}{p_i^0} = 0.66, \quad r = \arg\min_{i\in U_0} x_{U_i} = 2, \quad L_1 = \{2\}, \quad U_1 = \{1, 3\}.$$

with $x^1 = \begin{bmatrix} 0.81 & 0 & 0.19 \end{bmatrix}^T$ as the new approximate to the optimal solution. In iteration $k = 1$, we will work with the submatrices

$$Q_{U_1U_1} = \begin{bmatrix} 6 & 1 \\ 1 & 4 \end{bmatrix}, \qquad A_{U_1} = \begin{bmatrix} 1 & 1 \end{bmatrix}, \qquad Y_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad Z_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

From these submatrices we obtain p_{U_1} as

$$p_{U_1} = Y_1 p_{Y_1} + Z_1 p_{Z_1} = 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} - 0.19 \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.19 \\ -0.19 \end{bmatrix},$$

and

$$x_U = x_{U_1} + p_{U_1} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T.$$

The new iteration is feasible. The next steps are to find the Lagrange multipliers of the equality and inequality constraints and to check the sign of the last ones. By using equations (15) and (3), we get

$$\lambda = -2, \qquad x^2 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T, \qquad \mu = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T.$$

Since $\mu \geq 0$, the KKT conditions are satisfied, so we have found the solution. We terminate by setting

$$x^* = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T.$$

4 Experiments

To evaluate the performance of the active-set algorithm for our specific problem, we generate random positive semidefinite matrices Q, random vectors c, and random partitions across various sizes. During the generation of Q, we manipulate three key properties: sparsity level, definiteness (positive definite or semidefinite), and rank (full or low rank). Additionally, we aim to analyze the algorithm's behavior without constraints and compare its performance against a known solution when constraints are introduced. For this purpose, we test the algorithm using $q = -Qx_0$. The metrics selected for comparison are the CPU time in seconds and the relative gap between the computed solutions. Since the algorithm was coded in Python, we benchmark our algorithm against two Python quadratic problem solvers: *clarabel* and *quadprog* [2].

4.1 Data generation and solvers selection

Full-rank and low-rank dense matrices Q are generated in Python. To create random positive semidefinite and definite matrices, we leverage the property that for any matrix $Q \in \mathcal{M}_{m\times n}(\mathbb{R})$, the matrices Q^TQ and QQ^T are symmetric and positive semidefinite. If Q is full rank both matrices will also be positive definite. To construct low-rank matrices we exploit the property that $\text{Rank}(AB) \leq \min\{\text{Rank}(A), \text{Rank}(B)\}$, generating Q as the product of two smaller matrices with the desired ranks. Sparse matrices, on the other hand, are generated in MATLAB using the *sprandsym* function. This function allows us to specify the dimensions, density level, and vector of eigenvalues of the matrix generated randomly using the gamma distribution. The partition that defines the constraints for each problem is generated randomly.

An initial solution is required to start the algorithm. Observe that in our case, it is easy to find an initial feasible point since the equality constraints are defined by disjoint

Algorithm 1 Active set algorithm

```
Require: ASQP($Q$, $q$, $A$, $x_0$, $L_0$, $U_0$)
 1: for $k = 0, 1, 2, \dots$ do
 2:     Compute the permutation matrix $P$                          ▷ $A_{U_k} P_k = [I|A_{Z_k}]$
 3:     $Y_k \leftarrow \begin{bmatrix} I \\ 0 \end{bmatrix}, \quad Z_k \leftarrow \begin{bmatrix} -A_{Z_k} \\ I \end{bmatrix}$
 4:     $p_{Y_k} \leftarrow -h_{U_k}$
 5:     if $(PZ)^T Q_{U_k U_k} (PZ)$ is non-singular then
 6:         Compute $p_{Z_k}$ by solving the system
                $$(PZ)^T Q_{U_k U_k} (PZ) p_{Z_k} = -(PZ)^T (Q_{U_k U_k} P Y_k p_{Y_k} + g_k)$$
 7:         $p_U^k \leftarrow P(Y_k p_{Y_k} + Z_k p_{Z_k})$
 8:         $x_U \leftarrow x_U^k + p_U^k$
 9:         if $x_U \geq 0$ then                                    ▷ *New iteration is a feasible point*
10:             $x^k \leftarrow [x_{L_k}^k, x_U]$                          ▷ $x_{L_k}^k = 0$ *active variables*
11:             $\lambda \leftarrow (PY_k)^T (g_{U_k} + Q_{U_k U_k} p_{U_k}^k)$    ▷ *Multipliers of the equality constraints*
12:             $\mu \leftarrow Qx^k + q - A^T \lambda$   ▷ *Lagrange multipliers of the inequality constraints*
13:             if $\mu \geq 0$ then
14:                 Stop. $x^k$ is the optimal solution
15:             else
16:                 $s \leftarrow \arg\min_{i \in L} \mu_i$
17:                 $L_{k+1} \leftarrow L_k \setminus \{s\}$       ▷ *Decrease the objective function by dropping s*
18:                 $U_{k+1} \leftarrow U_k \cup \{s\}$
19:                 $x^{k+1} \leftarrow x^k$
20:             end if
21:         else                                                  ▷ *New iteration is not feasible*
22:             $p^k \leftarrow [p_U^k, 0]$
23:             $\alpha_k \leftarrow \min_{i \in U_k,\, p_i^k < 0} \frac{-x_i^k}{p_i^k}$         ▷ *Compute the step-length parameter*
24:             $x^{k+1} \leftarrow x^k + \alpha_k p^k$                            ▷ *Compute new iterate*
25:             $r \leftarrow \arg\min_{i \in U_k} x_{U_i}$
26:             $L_{k+1} \leftarrow L_k \cup \{r\}$                               ▷ *Update the active set*
27:             $U_{k+1} \leftarrow U_k \setminus \{r\}$
28:         end if
29:     else
30:         Find $p_U^k$ satisfying (16)
31:         Follow steps (23)-(27).
32:     end if
33:     $k \leftarrow k + 1$
34: end for
```

simplices and $b = 1$. Therefore, the initial point will be constructed as a vector of zeros and ones, where the ones are in the positions associated with the minimum value in each subset of the partition.

Initially, we considered comparing our results with the *quadprog* solver, which implements the Goldfarb / Idnani dual algorithm, which is a numerically stable dual method to solve strictly convex quadratic programs. However, *quadprog* performs best in well-conditioned, dense problems and requires dense positive definite matrices Q. To extend our test to positive semidefinite matrices, both dense and sparse, we decided to include the solver *clarabel* in our benchmark which is an interior point solver for general convex optimization problems.

4.2 Results

Result based on Q random positive semidefinite matrix with full rank (random q)

Dimension			**asqp**	**clarabel**	
k	**n**	**sp**	**CPU time**	**CPU time**	**relative gap**
10	100	0%	0.5312	0.0156	2.61643$e-10$
	200	0%	1.1875	0.2031	1.14122$e-10$
	500	0%	5.625	1.1093	1.34846$e-09$
	1000	0%	31.5312	6.6093	1.026$e-10$
50	100	0%	0.2344	0.0312	1.6276$e-09$
	200	0%	0.9844	0.1875	4.14072$e-11$
	500	0%	5.5625	0.6718	1.43067$e-09$
	1000	0%	29.5	5.0312	1.01139$e-09$
90	100	0%	0.0625	0.0312	1.29174$e-11$
	200	0%	1	0.0781	2.01968$e-10$
	500	0%	5.0781	0.6875	4.03154$e-10$
	1000	0%	27.625	4.4218	2.1708$e-10$

Table 1: Comparison of *asqp* with the Python solver *clarabel*.

Result based on Q random positive semidefinite matrix with full rank (with $q = -Qx_0$)

Dimension			**asqp**	**clarabel**	
k	**n**	**sp**	**CPU time**	**CPU time**	**relative gap**
10	100	0%	0	0.0156	$1.35281e{-}09$
	200	0%	0.0156	0.1875	$2.78056e{-}09$
	500	0%	0	0.9531	$3.20803e{-}09$
	1000	0%	0	6.4687	$1.62224e{-}09$
50	100	0%	0	0.0312	$8.62897e{-}10$
	200	0%	0	0.1406	$1.60546e{-}09$
	500	0%	0	0.8125	$2.15177e{-}09$
	1000	0%	0	4.8593	$9.04293e{-}10$
90	100	0%	0	0.0312	$1.75287e{-}09$
	200	0%	0	0.1093	$1.12391e{-}09$
	500	0%	0	0.75	$8.08526e{-}10$
	1000	0%	0	4.6718	$1.57197e{-}09$

Table 2: Comparison of our algorithm *asqp* with the Python solver *clarabel* across different dimensions of Q and partition size with special choice of q

Results based on random matrix Q positive semidefinite and low rank, with rank 50 (with random q)

Dimension			**asqp**	**clarabel**	
k	**n**	**sp**	**CPU time**	**CPU time**	**relative gap**
10	100	0%	0.3125	0.0156	$3.80402e{-}09$
	200	0%	0.5781	0.1093	$2.63117e{-}10$
	500	0%	5.3594	0.8906	$4.75016e{-}10$
	1000	0%	15.0781	6.875	$1.55995e{-}09$
50	100	0%	0.2344	0.0312	$8.9139e{-}11$
	200	0%	0.7344	0.125	$1.05165e{-}09$
	500	0%	10.8906	0.9062	$2.0972e{-}09$
	1000	0%	27.0312	7.5	$6.30032e{-}09$
90	100	0%	0.0625	0.0312	$1.10043e{-}09$
	200	0%	0.75	0.0468	$1.72776e{-}09$
	500	0%	9.6562	0.8125	$1.18405e{-}09$
	1000	0%	28.9688	6.3906	$4.99909e{-}11$

Table 3: Comparison of our algorithm *asqp* with the Python solver *clarabel*.

Results based on random matrix Q positive semidefinite and low rank, with rank (with $q = -Qx_0$)

Dimension			**asqp**	**clarabel**	
k	**n**	**sp**	**CPU time**	**CPU time**	**relative gap**
10	100	0%	0	0.0312	$1.54038e-09$
	200	0%	0	0.125	$1.0532e-09$
	500	0%	0.0312	0.7656	$2.5541e-09$
	1000	0%	0.0312	5.6406	$2.3897e-09$
50	100	0%	0	0.0156	$9.73945e-10$
	200	0%	0	0.125	$1.39714e-09$
90	100	0%	0.0312	0.0312	$3.0153e-09$
	200	0%	0	0.0781	$2.30374e-09$

Table 4: Comparison of our algorithm *asqp* with the Python solver *clarabel*.

Result based on random matrix Q positive semidefinite, sparse and full rank (with random q)

Dimension			**asqp**	**clarabel**	
k	**n**	**sp**	**CPU time**	**CPU time**	**relative gap**
10	100	80%	0.2188	0.0156	$3.24306e-10$
	200	80%	0.4219	0.0468	$4.65353e-11$
	500	80%	0.3906	0.1875	$2.82609e-09$
	1000	80%	2.9062	2.125	$9.4187e-10$
50	100	80%	0.2812	0.0156	$2.46114e-09$
	200	80%	0.7188	0.0156	$1.68971e-09$
	500	80%	1.5469	0.1875	$3.75594e-10$
	1000	80%	10.7031	2.5312	$7.89196e-10$
90	100	80%	0.1406	0.0156	$4.69994e-10$
	200	80%	1.2656	0.0625	$1.27343e-09$
	500	80%	3.5469	0.375	$1.20888e-10$
	1000	80%	20.3125	1.4062	$8.24265e-10$

Table 5: Comparison of our algorithm *asqp* with the Python solver *clarabel*.

Result based on random matrix Q positive semidefinite, sparse and full rank (with $q = -Qx_0$)

Dimension			asqp	clarabel	
k	**n**	**sp**	**CPU time**	**CPU time**	**relative gap**
10	100	80%	0	0.0156	$9.20617e-10$
	200	80%	0	0.0468	$1.40062e-09$
	500	80%	0	0.2656	$8.64422e-10$
	1000	80%	0	1.6718	$1.99291e-09$
50	100	80%	0.0153	0	$2.90635e-09$
	200	80%	0.0312	0.0312	$2.59167e-09$
	500	80%	0	0.3437	$7.28185e-10$
	1000	80%	0.0312	1.6718	$2.74467e-09$
90	100	80%	0	0	$7.70703e-10$
	200	80%	0	0.0625	$1.91252e-09$
	500	80%	0	0.2968	$4.59655e-10$
	1000	80%	0	2.125	$2.50951e-09$

Table 6: Comparison of our algorithm *asqp* with the Python solver *clarabel*.

Result based on Q random positive definite matrix and full rank (random q)

Dimension			asqp	quadprog	
k	**n**	**sp**	**CPU time**	**CPU time**	**relative gap**
10	100	0%	0.1255	0.0156	$5.74464e-15$
	200	0%	0.2188	0.0468	$2.19445e-13$
	500	0%	0.3906	0.7343	$6.3479e-15$
	1000	0%	1.2031	5.75	$1.18691e-09$
50	100	0%	0.1094	0.0312	$7.95087e-15$
	200	0%	0.3906	0.0625	$1.66068e-14$
	500	0%	0.8906	0.7031	$1.3802e-13$
	1000	0%	0.8906	5.9062	$3.14331e-14$
90	100	0%	0.0781	0	$5.77498e-14$
	200	0%	0.5312	0.0625	$8.55327e-14$
	500	0%	1.4219	0.7656	$1.69554e-14$
	1000	0%	3.1094	5.4687	$1.4199e-13$

Table 7: Comparison of our algorithm *asqp* with the Python solver *quadprog* across different dimensions of Q positive definite.

Result based on Q random positive definite matrix and full rank (with $q = -Qx_0$)

Dimension			asqp	quadprog	
k	n	sp	CPU time	CPU time	relative gap
10	100	0%	0	0	$3.52559e-16$
	200	0%	0	0.0625	0
	500	0%	0	0.7031	$1.98553e-15$
	1000	0%	0	4.9218	$4.38434e-15$
50	100	0%	0	0	$1.14492e-16$
	200	0%	0	0.0625	0
	500	0%	0	0.625	0
	1000	0%	0	3.9375	$1.66249e-15$
90	100	0%	0	03125	0
	200	0%	0	0.0625	$1.44387e-16$
	500	0%	0	0.6875	$2.29408e-16$
	1000	0%	0.0156	4.125	$6.87618e-16$

Table 8: Comparison of our algorithm *asqp* with the Python solver *quadprog* across different dimensions of Q positive definite and special choice of q.

In Table (1) we can see that the relative gap between the objective values is really small, with *asqp* given better result than *clarabel*, however, taking longer to find the optimal. When keeping Q random positive semidefinite and using the special q, we observe in Table (2) that *asqp* is faster than *Clarabel*. Furthermore, it provides a better solution in all cases.

In the case when Q is random positive semidefinite and low rank, Table (3), we can observe that our algorithm is slow compared to *Clarabel*, but returning a better solution. In the fourth experiment, Table (4), *asqp* shows a better performance than *clarabel* not only in the time used, but also, in providing the optimal solution. *Clarabel* can find the optimal for all the values of n when $k = 10$, and just in two cases when $k = 50, 90$ returning in the last cases a numerical error message. The algorithm *asqp* instead can return the optimal in all cases, these values being smaller than the ones given by *Clarabel*.

Table (5) and Table (6) show the results for Q is positive semidefinite and sparse. When q is random, both algorithms found the optimal values, again *Clarabel* performing in less time but returning a less accurate solution. On the other hand, when q is the special vector $q = -Qx_0$ we observe in Table (6) a similar performance of *clarabel* than in the previous experiment, this time taking longer to reach the optimal than *asqp*.

Finally, we consider the cases where Q is positive definite and we perform a benchmark using the Python solver *quadprog*. When taking q random we see that *asqp* is faster than *quadprog* when the dimension of Q is $n = 1000$, see Table (7). In most cases, *quadprog* provided a more accurate solution than *asqp*, however, with a small gap between them.

The algorithm *asqp* reached a smaller objective value for the following pairs of (k, n): $(10, 100), (50, 500), (50, 1000), (90, 1000)$. The opposite behavior can be appreciated when q is taken $q = -Qx_0$, see Table (8), in which *asqp* is faster than *quadprog* in all cases, providing a more accurate solution or the same solution than *quadprog* in almost all cases when $k = 50, 90$.

References

[1] Aharon Ben-Tal and Arkadi Nemirovski. Lecture notes optimization iii, 2013.

[2] Stéphane Caron, Akram Zaki, Pavel Otta, Daniel Arnström, Justin Carpentier, Fengyu Yang, and Pierre-Alexandre Leziart. qpbenchmark: Benchmark for quadratic programming solvers available in python, 2024.

[3] Jorge Nocedal and Stephen J Wright. *Numerical optimization.* Springer, 1999.