Comput Optim Appl
DOI 10.1007/s10589-017-9949-y

MINQ8: general definite and bound constrained indefinite quadratic programming

Waltraud Huyer[1] · **Arnold Neumaier**[1]

Received: 11 August 2015
© The Author(s) 2017. This article is an open access publication

Abstract We propose new algorithms for (i) the local optimization of bound constrained quadratic programs, (ii) the solution of general definite quadratic programs, and (iii) finding either a point satisfying given linear equations and inequalities or a certificate of infeasibility. The algorithms are implemented in MATLAB and tested against state-of-the-art quadratic programming software.

Keywords Definite quadratic programming · Bound constrained indefinite quadratic programming · Dual program · Certificate of infeasibility

1 Introduction

We consider the quadratic programming problem with bound constraints

$$\begin{array}{l} \min\ c^T x + \frac{1}{2} x^T G x \\ \text{s.t.}\ \ x \in \mathbf{x}, \end{array} \tag{1}$$

where $c \in \mathbb{R}^n$, G is a symmetric $n \times n$ matrix, not necessarily semidefinite,

$$\mathbf{x} := [\underline{x}, \overline{x}] = \{x \in \mathbb{R}^n \mid \underline{x}_i \le x_i \le \overline{x}_i,\ i = 1, \ldots, n\} \tag{2}$$

✉ Waltraud Huyer
Waltraud.Huyer@univie.ac.at

Arnold Neumaier
Arnold.Neumaier@univie.ac.at

[1] Fakultät für Mathematik, Universität Wien, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

Published online: 03 October 2017

is a box in $\mathbb{R}^n$, and infinite bounds are permitted. Quadratic programming problems with simple bounds form an important class of nonlinear optimization problems, including for positive definite G linear least squares problems with nonnegativity constraints and the dual problems of linearly constrained least distance problems, and for indefinite G a number of combinatorial problems such as the maximum clique problem or the maximum cut problem (see, e.g., Floudas and Pardalos [4]). If G is positive semidefinite then every local minimizer is a global minimizer. On the other hand, the global optimization problem is quite difficult in the indefinite case, where the reliable solution needs quite different techniques (relaxation and branch and bound); see, e.g., several articles in the collection by Lee and Leyffer [17]. Here we only discuss the local solution of indefinite quadratic programs, needed repeatedly in branch and bound frameworks for finding good starting points for a global solver.

There are many optimization algorithms designed for finding local optima of quadratic programs under various conditions. A fairly comprehensive list of algorithms is maintained by Gould and Toint [9]. For strictly convex bound constrained quadratic programs, many algorithms are available. For a comparison of solvers for strictly convex bound constrained quadratic programs see Voglis and Lagaris [24]. We are interested in two more general situations: bound constrained quadratic programs without restriction (they may be strictly convex, rank-deficient or indefinite), and strictly convex quadratic programs with general linear constraints. Because of duality, the second problem class may be viewed as a special case of the first, hence is still significantly simpler than a general quadratic program.

MINQ5 [20] is a publicly available MATLAB program for bound constrained quadratic programming and strictly convex general quadratic programming, based on rank 1 modifications. It finds a local minimizer of the problem (1). If G is positive semidefinite, any local optimizer is global, so it finds (in theory, assuming exact arithmetic) the global optimum. The method combines coordinate searches and subspace minimization steps. The latter solve safeguarded equality constrained QPs whenever the coordinate searches no longer change the active set. Rank 1 updates, in a format suited for both the dense and sparse case, are used to keep the linear algebra reasonably cheap.

Applications of MINQ5 to bound constrained quadratic programs include the calculation of the formation constants for copper(I)-chloride complexes [19], subpixel land cover mapping [3], an optimal control model of redundant muscles in step-tracking wrist movements [10], entropy estimation for high-dimensional finite-accuracy data [16], finding the best code to represent a speech signal [23], the calculation of a nonnegative sparsity induced similarity measure for cluster analysis of spam images [7], a method for calibrating the scores of biased reviewers [22], a model for the braking behavior of industrial robots [2], and finding the filter coefficients for a filtering technique to denoise the far-field pattern in the presence of noise [21]. Moreover, many nonlinear optimization techniques are based on solving auxiliary quadratic problems [6,14,18,24,25]. Kanzow and Petra [15] used MINQ5 to solve a trust-region subproblem in a scaled filter trust region method. Finally, MINQ5 is an integral part of the global optimization algorithm MCS [12] and the noisy optimization algorithm SNOBFIT [13].

In some of our applications we encountered bound constrained quadratic programs where MINQ5 turned out to be very slow. This provided the impetus for developing a new quadratic programming algorithm. Like MINQ5 it employs coordinate searches and subspace steps, but the latter use smaller subspaces than in MINQ5. In contrast to MINQ5, we do not use rank 1 updates but instead make direct factorizations. In analogy to the MINQ5 package [20] (named after MATLAB 5), we call our new algorithm MINQ8. As MINQ5, our MATLAB 8 implementation of MINQ8 solves both the general bound constrained quadratic program and the related problems discussed in Sect. 4. The software is freely available at http://www.mat.univie.ac.at/~neum/software/minq/, together with the drivers for the problem classes from Sect. 5.3.

In Sect. 2 we start with a motivation of the ingredients of MINQ8, introduce some notation, describe the main subprograms of the algorithm in detail in the order in which they are called in MINQ8, discuss the modifications needed if some bound is infinite and finally present MINQ8 as a whole. We examine properties of the points obtained by the algorithm and prove convergence under the assumption of strict convexity and special settings of the parameters in Sect. 3. In Sect. 4 we describe how MINQ8 can be applied to general positive definite quadratic programs. Finally, in Sect. 5 we make an extensive comparison of MINQ8, MINQ5, the algorithms contained in the `quadprog` function of MATLAB, and `NewtonKKTqp` [1] on different test problems.

2 The MINQ8 algorithm

MINQ8 is designed to solve the problem of finding a *local* minimizer of a bound constrained (definite or indefinite) quadratic problem of the form

$$\begin{aligned} &\min\ f(x) := \gamma + c^T x + \rho(Ax - b) \\ &\text{s.t.}\quad x \in \mathbf{x}, \end{aligned} \tag{3}$$

where $\gamma \in \mathbb{R}$, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m\times n}$, $\mathbf{x}$ is as in (2), and $\rho : \mathbb{R}^m \to \mathbb{R}$ is given by

$$\rho(z) := \frac{1}{2} z^T D z,$$

where $D \in \mathbb{R}^{m\times m}$ is a diagonal matrix. Thus there are n variables and each residual $Ax-b$ has m components. Note that this form of writing a quadratic function $\gamma + c^T x + \frac{1}{2} x^T G x$ is not a restriction since each symmetric matrix $G \in \mathbb{R}^{n\times n}$ can be written as $G = A^T D A$, $A, D \in \mathbb{R}^{n\times n}$, D a diagonal matrix, using a spectral factorization or an LDL^T factorization of G.

By a well-known theorem by Frank and Wolfe [5], (3) has a finite solution if and only if $\{f(x) \mid x \in \mathbf{x}\}$ is bounded below. In this case, the necessary conditions for local optimality are given by the Kuhn–Tucker conditions

$$\begin{aligned} g_i &= 0 \quad \text{if } \underline{x}_i < x_i < \overline{x}_i, \\ g_i &\geq 0 \quad \text{if } \underline{x}_i = x_i < \overline{x}_i, \end{aligned}$$

$$g_i \le 0 \quad \text{if } \underline{x}_i < x_i = \overline{x}_i,$$

where

$$g := c + A^T D(Ax - b) \tag{4}$$

denotes the gradient of f at x. Therefore the **active set** at a point x, defined as

$$K(x) := \{i \mid x_i = \underline{x}_i \text{ or } x_i = \overline{x}_i\},$$

plays an important role. The variables x_i with $i \in K(x)$ are called **active** (at the respective bound); the variables that are not active are called **inactive** (or **free**) variables. We say that two points $x, y \in \mathbf{x}$ have the same **activities** iff $K(x) = K(y)$ and $x_i = y_i \ \forall i \in K(x) = K(y)$.

The easiest case is that of an unconstrained convex quadratic optimization problem, where the global optimizer can be computed easily since the Kuhn–Tucker conditions reduce to a linear system. If $x \in \mathbb{R}^n$ with the corresponding gradient g and the Hessian matrix G, the global minimum is attained at $x + p$ with $Gp = -g$. On the other hand, in the non-convex case even the problem of checking the existence of a Kuhn–Tucker point is NP-hard; cf. Horst et al. [11]. If the unconstrained minimizer of a convex quadratic optimization problem is not contained in $\mathbf{x}$ and in all other cases where the objective function is bounded on the box $\mathbf{x}$, minimizers occur at the boundary. Therefore it is important to find the correct activities.

For any putative active set, the corresponding variables can be fixed, and the remaining part of the gradient can be set to zero by solving a linear system. If the Hessian is positive definite on the subspace, this gives the optimal point in the corresponding subspace and defines part of the `subspacestep` (with modifications described below). The best feasible point on the entire line through the current best point and this point (if these are distinct) defines the result of this step. Thus a two-sided line search is performed; a directional line search would not always be sufficient, since the Hessian on the subspace might be indefinite or negative definite.

The solution of these linear systems is the most time-consuming part of the algorithm; hence it pays to try to make the subspace dimension small before performing the subspace step. This is done using three different techniques.

To increase the active set we use (as in MINQ5) a greedy method `fixbounds` that fixes single variables as long as the objective function decreases.

A further dimension reduction is possible by noting that some coordinates are unlikely to contribute much to the subspace step. These coordinates are determined by a simple heuristic and form together with the active coordinates an extended active set. The subspace direction is then computed only for the lower-dimensional subspace spanned by the complementary reduced inactive set, and is augmented to a direction in the full inactive set by components pointing towards the more favorable bound of this component.

A more expensive special `reductionstep` is done in cases where this still leaves more reduced inactive variables than the number of rows of A. Indeed, in this case we can fix more bounds by an approximate minimization on the subspace where Ax

is fixed. Compared to MINQ5, this is another theoretical innovation of the present algorithm.

These steps are alternated until `subspacestep` does not change the active set any more. Then we try to free some of the bounds, which is done in `freebounds`, where a two-sided line search is performed along a direction p that would be optimal in the case of a separable problem.

Notation. We first introduce some notation, which will be used in the remainder of this section. e^i denotes the ith standard unit vector of $\mathbb{R}^n$. We write $A_{i,j}$ for the (i, j)-entry of A. If I is a subset of $\{1, \ldots, m\}$ and J a subset of $\{1, \ldots, n\}$, $A_{I,J}$ denotes the submatrix of A with the row indices taken from I and the column indices from J, i.e., $A_{I,J} := (A_{i,j})_{i\in I, j\in J}$, $A_{I,:}$ the submatrix $A_{I,:} := (A_{i,j})_{i\in I, 1\le j\le n}$, and similarly $A_{:,J} := (A_{i,j})_{1\le i\le m, j\in J}$. If $J = \{l\}$, we also write $A_{:,l}$ instead of $A_{:,J}$, analogously for the other cases of one-element index sets. x_J is the subvector of x with the indices taken from the set J, and the vector $|x| := (|x_1|, \ldots, |x_n|)^T$ is obtained by taking the absolute values of all components of x.

We also introduce the quantities

$$d_i := \frac{1}{2} A_{:,i}^T D A_{:,i}, \quad i = 1, \ldots, n. \tag{5}$$

They are independent of the point x and stay constant during the algorithm. On the other hand, the gradient (4) must be computed at each point x. The minimum of the difference

$$\Delta f := f(x + \alpha e^i) - f(x) = \alpha g_i + \alpha^2 d_i$$

in function value when changing the ith coordinate within the box, so that $x + \alpha e^i \in \mathbf{x}$, is attained at one of

$$\alpha = \underline{\alpha}_i := \underline{x}_i - x_i, \quad \Delta f = \underline{\Delta} f_i := \underline{\alpha}_i (g_i + \underline{\alpha}_i d_i), \tag{6}$$

$$\alpha = \overline{\alpha}_i := \overline{x}_i - x_i, \quad \Delta f = \overline{\Delta} f_i := \overline{\alpha}_i (g_i + \overline{\alpha}_i d_i), \tag{7}$$

$$\alpha = \hat{\alpha}_i := -\frac{g_i}{2d_i}, \quad \Delta f = \hat{\Delta} f_i := \frac{g_i}{2}\hat{\alpha}_i = -\frac{g_i^2}{4d_i}, \tag{8}$$

where the minimizer can be different from $\underline{\alpha}_i$ and $\overline{\alpha}_i$ only if $d_i > 0$ and $\underline{\alpha}_i < \hat{\alpha}_i < \overline{\alpha}_i$.

In the first five subsections of this section, we describe the ingredients of the algorithm in detail under the condition that all bounds are finite. Subsequently we deal with the possibility of infinite bounds, and in the final subsection the MINQ8 algorithm is presented as a combination of its ingredients.

2.1 `fixbounds`

The subprogram `fixbounds` cyclically tries to fix as many variables at bounds as possible in order to be left with a not too large set of inactive variables to be optimized in the subsequent subspace step.

Let `ind` be an index vector that sorts the components of d in ascending order. Then the application of `fixbounds` to a point x is described in Algorithm 1 for the case that the function is bounded below, where $\underline{\Delta} f_i$ and $\overline{\Delta} f_i$ are defined by (6) and (7). Note that we use the equality sign for assignment as well as comparison in our pseudocodes.

Input: x, d, `ind`, problem characteristics
while *the loop changes the point* **do**
  for $j = 1$ **to** n **do**
    $i = \texttt{ind}(j)$
    Compute $\underline{\Delta} f_i$ and $\overline{\Delta} f_i$
    if $\min(\underline{\Delta} f_i, \overline{\Delta} f_i) < 0$ **then**
      Set x_i to the boundary value yielding the lower function value
    end
  end
end
Output: x, $f(x)$

Algorithm 1: `fixbounds`

After `fixbounds` has been completed it is not possible to improve the function value by fixing an additional bound. Let $\overline{I}$ be the inactive set of the point x obtained by `fixbounds`. Then the minimum of $f(x + \alpha e^i)$, $x + \alpha e^i \in \mathbf{x}$, is attained for $\alpha = \hat{\alpha}_i \in (\underline{\alpha}_i, \overline{\alpha}_i)$ for $i \in \overline{I}$ and thus $d_i > 0$, $\underline{\Delta} f_i \geq 0$, $\overline{\Delta} f_i \geq 0$, and $\hat{\Delta} f_i \leq 0$, where these quantities are defined by (5)–(8).

2.2 The reduced inactive set

We now describe the heuristic screening step used for choosing a sensible **reduced inactive set** (a subset of the inactive set $\overline{I}$ of x) and the complementary **extended active set** (a superset of the active set of x).

The idea is to incorporate some information from the bounds (which, in a full subspace step, would not influence the search direction). We compare for each inactive variable i of the point x obtained by `fixbounds` the (nonnegative) gains $\underline{\Delta} f_i$ and $\overline{\Delta} f_i$ of moving this variable to a bound with the (nonpositive) gain $\hat{\Delta} f_i$ when moving to the optimal interior value. The absolute value of the ratio is taken to be a heuristic indicator of how important the coordinate is for the subspace. A tunable threshold κ for this ratio is used to determine which inactive variables are assigned to the reduced inactive set.

The reduced inactive set is thus defined to be

$$I := \{i \in \overline{I} \mid \kappa |\hat{\Delta} f_i| < \min(\underline{\Delta} f_i, \overline{\Delta} f_i)\}$$

for some fixed $\kappa \geq 0$, and the extended active set is its complement $K := \{1, \ldots, n\} \backslash I$. In the special case that $\kappa = 0$, the extended active and restricted inactive sets coincide with the active and inactive sets, respectively. Another special case is $\kappa = 1$, where $\underline{\Delta} f_i$, $\overline{\Delta} f_i$, and $|\hat{\Delta} f_i| = -\hat{\Delta} f_i$ are compared. For very large κ, only the indices i with

$\hat{\Delta} f_i = 0$ (i.e., x_i has its optimal value along coordinate i in the interior of $[\underline{x}_i, \overline{x}_i]$) belong to the reduced inactive set and thus very large values of κ do not make sense.

A non-vectorized version of the algorithm of computing the reduced inactive and extended active sets is given in Algorithm 2.

Input: $x, g, d, \underline{x}, \overline{x}$
$I = \{i \in \{1, \ldots, n\} \mid \underline{x}_i < x_i < \overline{x}_i\}$
$K_l = \{i \in \{1, \ldots, n\} \mid x_i = \underline{x}_i\}$
$K_u = \{i \in \{1, \ldots, n\} \mid x_i = \overline{x}_i\}$
for $i \in I$ **do**
  Compute $\underline{\Delta} f_i, \overline{\Delta} f_i, \hat{\Delta} f_i$
  if $\underline{\Delta} f_i \le \min(\overline{\Delta} f_i, -\kappa \hat{\Delta} f_i)$ **then**
    $K_l = K_l \cup \{i\},\ I = I \setminus \{i\}$
  else if $\overline{\Delta} f_i \le -\kappa \hat{\Delta} f_i$ **then**
    $K_u = K_u \cup \{i\},\ I = I \setminus \{i\}$
  end
end
Output: K_l, K_u, I

Algorithm 2: Computation of the reduced inactive and extended active sets

2.3 `reductionstep`

Let I be the reduced inactive set of the current point x. If $|I| > m$, $G_{I,I}$ is singular and we reduce the number of elements of I by the following procedure to obtain $|I| \le m$.

The idea is to perform an approximate minimization on the subspace $Ax = Ax^0$, where x^0 is the current best point. On this subspace, the objective function simplifies to $c^T x$ plus a constant, and all currently active variables are fixed. Thus proceeding along a descent direction (if there is one) of the linear program

$$\begin{aligned} &\min\ c^T x \\ &\text{s.t.}\quad Ax = Ax^0, \quad x \in \mathbf{x} \end{aligned}$$

we are guaranteed to fix a bound, and this can be iterated in strongly polynomial time until $|I| \le m$.

Noting that the case $|I| > m$ is only possible for $n > m$, we use LU factorizations of $A^T_{:,I}$ and submatrices of $A^T_{:,I}$ to determine maximal subsets I' of I and J' of $\{1, \ldots, m\}$, $m \ge r := |I'| = |J'|$, such that $A_{J',I'}$ is invertible, and compute its inverse $C := A^{-1}_{J',I'}$. The set $J'' := \{1, \ldots, m\} \setminus J'$ might be empty, but $I'' := I \setminus I' \ne \emptyset$ since $|I| > m$.

We now define a vector $u \in \mathbb{R}^n$ with $Au = 0$ by assigning the subvectors $u_{I'}$, $u_{I''}$ and u_K (pertaining to the index sets I', I'' and K, which form a disjoint partition of $\{1, \ldots, n\}$) as follows. Let $u_{I''} = e^k$ for some $k \in \{1, \ldots, |I''|\}$, and let $u_{I'} := -CA_{J',I''}u_{I''}$ and $u_K = 0$. Then we have $A_{:,I''}u_{I''} = A_{:,l}$ for some $l \in I''$ and

$$Au = A_{:,I'}u_{I'} + A_{:,I''}u_{I''} = -A_{:,I'}CA_{J',I''}u_{I''} + A_{:,l} = -A_{:,I'}CA_{J',l} + A_{:,l}.$$

Since r is the (numerical) rank of $A_{:,I}$ and $A_{J',I'}$ is nonsingular, the columns of $A_{:,I'}$ form a basis of the column space of $A_{:,I}$. Hence $A_{:,l}$ can be written as a linear combination

$$A_{:,l} = \sum_{j=1}^{r} \lambda_j A_{:,i_j}$$

for appropriate λ_j, where $I' = \{i_1, \ldots, i_r\}$. Then

$$\begin{aligned} Au &= -A_{:,I'} C A_{J',l} + A_{:,l} = -\sum\nolimits_{j=1}^{r} \lambda_j A_{:,I'} C A_{J',i_j} + A_{:,l} \\ &= -\sum\nolimits_{j=1}^{r} \lambda_j A_{:,I'} e^j + A_{:,l} = -\sum\nolimits_{j=1}^{r} \lambda_j A_{:,i_j} + A_{:,l} = 0, \end{aligned}$$

where we have used the definition of C.

Since $A(x + \alpha u) = Ax$, minimizing $c^T(x + \alpha u) + \rho(A(x + \alpha u))$ over the box $\mathbf{x}$ amounts to minimizing $\alpha c^T u$ subject to $x + \alpha u \in \mathbf{x}$, which is easy to do.

Input: x, I, I', J', problem characteristics
$K = \{1, \ldots, n\} \setminus I$
$I'' = I \setminus I'$
while $I'' \neq \emptyset$ **do**
  for *all possible choices for* u **do**
    Set α to the boundary value that minimizes $c^T(x + \alpha u)$
    $L = \{i \in I \mid x_i + \alpha u_i = \underline{x}_i \text{ or } \overline{x}_i\}$
    if $L \neq \emptyset$ **then**
      break
    end
  end
  $x = x + \alpha u$
  for $i \in L$ **do**
    if $i \in I'$ **then**
      Determine $j \in I''$ such that $|1 + (Ch)_{i'}|$, $h := A_{J',j} - A_{J',i}$, is maximal and i' is chosen such that i is the i'th element of I'
      $I' = (I' \setminus \{i\}) \cup \{j\}$, $I'' = (I'' \setminus \{j\}) \cup \{i\}$
      $C = (A_{J',I'} + h(e^{i'})^T)^{-1} = C - \frac{(Ch)C_{i',:}}{1+(Ch)_{i'}}$
    end
    $I'' = I'' \setminus \{i\}$, $I = I \setminus \{i\}$, $K = K \cup \{i\}$
  end
end
Output: x, $f(x)$, I, K

Algorithm 3: `reductionstep`

Putting things together results in Algorithm 3, where it is assumed that subsets J' of $\{1, \ldots, m\}$ and I' of I such that $A_{J',I'}$ is invertible have already been determined. $|I''|$ choices for $u_{I''}$ and thus for u are possible, but in the first `for` loop the algorithm just looks for one u that manages to fix at least one bound, and that will usually be the one with $u_{I''} = e^1 \in \mathbb{R}^{|I''|}$ since the standard basis vectors are tried out in the

natural order. Let $L \subset I$ be the set of bounds that are fixed. In the second `for` loop the elements $i \in L$ are transferred from I to K. If $i \in I''$, this can be done directly, thus reducing I''. In the case where $i \in I'$, $|I'|$ should stay the same. We therefore replace i by a $j \in I''$ for which $A_{J',\bar{I}'}$ is still invertible, where $\bar{I}' := (I' \cup \{j\}) \setminus \{i\}$, and $\bar{C} := A^{-1}_{J',\bar{I}'}$ is computed from C with the aid of the Sherman–Morrison formula. To improve numerical stability, the index j is chosen such that the denominator in the Sherman–Morrison formula is maximal. This procedure is repeated until $I = I'$, $I'' = \emptyset$. The indices that are transferred from I to K result in active indices of the new current point. The matrix $A_{:,I}$ obtained at the end of the algorithm has full rank by construction, which is necessary (but not sufficient) for $G_{I,I} = A^T_{:,I} D A_{:,I}$ to be nonsingular.

2.4 `subspacestep`

Now we have a reduced inactive set I with $|I| \le m$ and an extended active set K, which is a superset of the active set K' of the current point x (the output of `reductionstep` if it has been called, otherwise the output of `fixbounds`) with the gradient g and the Hessian matrix G. I and K (or rather K_l and K_u) have been computed for x^{fb} (the point obtained by the last call to `fixbounds`) according to Algorithm 2 and reduced and augmented, respectively, by Algorithm 2.3 (if it is called). We define a search direction p as follows. For $k \in K'$, we set $p_k = 0$, and for $k \in K \setminus K'$ we set $p_k = \underline{x}_k - x_k$ or $p_k = \overline{x}_k - x_k$ (the value that yields the smaller function value, cf. Sect. 2.2). The optimal step p_I for the reduced inactive variables is determined by $G_{I,:}p = -g_I$, which results in solving

$$G_{I,I}\, p_I = -g_I - G_{I,K}\, p_K, \tag{9}$$

where p_K denotes the already determined part of p pertaining to the extended active set. To improve numerical stability we regularize the matrix G by slightly increasing its diagonal entries. We set $G_{i,i} = (1+\delta_1)G_{i,i}$ if $G_{i,i} \neq 0$ and $G_{i,i} = \delta_1$ if $G_{i,i} = 0$, where δ_1 is a fixed small nonnegative number. In the case $\delta_1 = 0$ G remains unchanged, but choosing $\delta_1 > 0$ might be advantageous for rank-deficient problems; cf. Sect. 5.1. The vector p_I is computed from (9) using the backslash operator in MATLAB.

If the linear system does not have a well-defined solution, the remainder of the subspace step is skipped. Otherwise an exact two-sided line search is made to find the best α with $x + \alpha p \in \mathbf{x}$, and a new current point $x + \alpha p$ is obtained. Note that α can even be negative, as in case of an indefinite Hessian nothing guarantees that p is a descent direction. It may even point to a local maximum, from which we must move away!

Since $p_i = 0$ for active indices i, we always have $x + \alpha p \in \mathbf{x}$ at least for small $|\alpha|$. $p = 0$ is only possible if $g_I = 0$ and all indices in K are active. The pseudocode of the `subspacestep` algorithm is given in Algorithm 4, where the sets K_l and K_u are the extended lower and upper active sets, respectively, $K = K_l \cup K_u$, and I is the reduced inactive set.

Input: x, g, K_l, K_u, I, problem characteristics
$p_{K_l} = \underline{x}_{K_l} - x_{K_l}$
$p_{K_u} = \overline{x}_{K_u} - x_{K_u}$
$h = \delta_1 \operatorname{diag}(G_{I,I})$
Set the coordinates of h that are 0 to δ_1
$p_I = -(G_{I,I} + \operatorname{diag}(h))\backslash(g_I + G_{I,K}\, p_K)$
if *p is finite* **then**
  $x = \operatorname{argmin}\{f(y) \mid y = x + \alpha p \in \mathbf{x}\}$
end
Output: x, $f(x)$

Algorithm 4: `subspacestep`

2.5 `freebounds`

In this subprogram $f(x + \alpha p)$ is minimized for $x + \alpha p \in \mathbf{x}$ with $p_i = \alpha_i$, where $\alpha_i \in \{\underline{\alpha}_i, \overline{\alpha}_i, \hat{\alpha}_i\}$ with smallest Δf as defined by (6)–(8). This choice of p would be optimal for separable quadratic problems. A non-vectorized version of this subprogram is given in Algorithm 5.

Input: x, g, d, problem characteristics
for $i = 1$ **to** n **do**
  if $d_i \le 0$ **then**
    Compute $\underline{\Delta} f_i$ and $\overline{\Delta} f_i$
    if $\underline{\Delta} f_i \le \overline{\Delta} f_i$ **then**
      $p_i = \underline{x}_i - x_i$
    else
      $p_i = \overline{x}_i - x_i$
    end
  else
    $\hat{\alpha}_i = -\frac{g_i}{2d_i}$
    $p_i = \max(\min(\hat{\alpha}_i, \overline{x}_i - x_i), \underline{x}_i - x_i)$
  end
end
$x = \operatorname{argmin}\{f(y) \mid y = x + \alpha p \in \mathbf{x}\}$
Output: x, $f(x)$

Algorithm 5: `freebounds`

2.6 Infinite bounds

Infinite bounds need a special treatment. Let I_l be the set of i such that $\underline{x}_i = -\infty$, let I_u be the set of i such that $\overline{x}_i = \infty$, and assume that $I_{\inf} := I_l \cup I_u \neq \emptyset$. If there is an $i \in I_{\inf}$ such that $d_i < 0$, the function is unbounded below on $\mathbf{x}$ and the program returns an error flag and a finite vector x with huge norm and $f(x) \ll 0$. Such a case can already be detected at the beginning of the algorithm.

For the case that $d_i = 0$ for some $i \in I_{\inf}$, the boundedness of the quadratic function on $\mathbf{x}$ depends on the ith component g_i of the gradient, a quantity that varies with the point x. The function is unbounded below on $\mathbf{x}$ if $i \in I_l$ and $g_i > 0$, or $i \in I_u$ and

$g_i < 0$. Such a case can be detected in `freebounds` and is handled similarly as the unbounded case of $d_i < 0$. The same is done in the case that an unbounded direction is found in an exact line search or in `reductionstep`, where setting α to the boundary that minimizes $c^T(x + \alpha u)$ in Algorithm 3 might identify an unbounded problem.

Since $|I| > m$ in `reductionstep` can only occur in the case $n > m$, where the Hessian matrix is singular, Algorithm 3 either finds a direction of infinite descent or manages to reduce $|I|$. The only exception is the case where $|c^T u|$ is so small that the search direction leaves $c^T x$ almost constant. In the case that $|c^T u| \leq \delta_2 |c|^T |u|$ (where δ_2 is a fixed small positive number) and the minimization would yield a direction of infinite descent, we do not minimize $\alpha c^T u$ but set $\alpha = 0$; if $c^T u = 0$ (the function is constant along the direction u) we also set $\alpha = 0$. If that is the case for all possible choices of u, it is not possible to reduce I and `subspacestep` is carried out with this I.

For the changes needed in Algorithms 1–5 due to possible infinite bounds see the actual MATLAB code at http://www.mat.univie.ac.at/~neum/software/minq/.

2.7 The MINQ8 algorithm

Now we are ready to describe the MINQ8 algorithm as a whole. MINQ8 performs four different subtasks, namely `fixbounds`, `reductionstep`, `subspacestep`, and `freebounds`. Each of these subprograms yields, starting from the point x produced by the previously called subprogram (or the initial point) with function value f, a point x^{new} with function value $f_{\text{new}} < f$ or stays at $x^{\text{new}} = x$, and x^{new} is used as input for the subsequent subtask. This is indicated by writing $x = \texttt{subprogram}(x)$ in the pseudocode for the subprograms of MINQ8 (we do not care for the other input and output parameters of the subprograms in the algorithm as a whole). The subtasks are carried out in an appropriate sequence according to Algorithm 6. Here δ_3 is a fixed small positive number to handle the stopping criterion for function values close to zero. This number and all other parameters in the algorithm will be specified in Sect. 5.

To avoid inefficient zigzagging, i.e., multiple freeing and fixing the same set of active variables in consecutive iterations, the subprogram `freebounds` is only called if the activities of the current points after the application of `fixbounds` and after the application of `subspacestep` are the same. This is equivalent to `reductionstep` (if applied) and `subspacestep` not generating any additional active variables (since it leaves active variables unchanged).

In addition to the possibility to identify the problem as unbounded at the beginning of the algorithm (before entering the loop in Algorithm 6) as described in Sect. 2.6, the algorithm checks at the beginning whether the function is constant along some coordinate axis. If $c_i = 0$ and $A_{:,i} = 0$ for some i, f does not depend on x_i. In that case, x_i is fixed at its initial value and only the remaining variables are optimized.

The program stops either due to finding an unbounded direction (before entering the loop or in the loop), or by reaching the maximum number `maxit` of iterations, or due to an insufficient change in the function value given by the input parameter `tol`. Since a very small step is rarely followed by a large step, it is typical of optimization algorithms

Input: x, `maxit`, `tol`, problem characteristics
Check whether the problem is unbounded along a coordinate and return in that case
Check whether the function does not depend on some variables
$f_{\text{old}} = \infty$
`nit` $= 0$
for `nit` $= 1$ **to** `maxit` **do**
  $x =$ `fixbounds`(x)
  $x^{\text{fb}} = x$
  Compute the reduced inactive set I of x
  if $|I| > m$ **then**
    $x =$ `reductionstep`(x)
  end
  $x =$ `subspacestep`(x)
  if x *has the same activities as* x^{fb} **then**
    $f = f(x)$
    if $f = f_{old}$ **or** $(f_{old} - f)/\max(|f_{old}|, |f|, \delta_3) <$ `tol` **then**
      break
    end
    $f_{\text{old}} = f$
    $x =$ `freebounds`(x)
  end
end
Output: x, $f(x)$, `nit`

Algorithm 6: The MINQ8 algorithm

to terminate them when the gain in function value is too small. The algorithm returns the best point (i.e., the current point at termination), its function value, the number of times the loop in Algorithm 6 has been carried out, and an error flag indicating whether the problem was detected to be unbounded.

Due to the fact that the subtasks of the algorithm only accept a new point if it has a strictly smaller function value, a local minimizer returned might belong to a locally optimal ray. Moreover, if the function value has not been changed by MINQ8, the current point has not changed. Therefore, in the case that `tol` is set to zero and `maxit` to ∞, the algorithm stops at a Kuhn–Tucker point; cf. Theorem 1.

3 Termination and convergence

In this section we prove several results concerning the termination of our algorithm. Theorem 1 deals with the case where a whole iteration of the main loop of MINQ8 does not change the point any more and the algorithm stops due to not making any progress in function value. To this end, in Lemma 1 we consider the case where `reductionstep` (if applied) and `subspacestep` have not changed the point, i.e., where some of the ingredients do not make any progress. In that case `freebounds` is called (since no change in the point implies no change in the activities) and is applied to the output of `fixpoints`.

Lemma 1 *If* `freebounds` *is applied to the output of* `fixbounds`*, then the search direction* p *in* `freebounds` *is a descent direction or* $p = 0$.

Proof Let x be the output of `fixbounds`. If $\underline{x}_i < x_i < \overline{x}_i$ for some i, then for $\underline{\alpha}_i, \overline{\alpha}_i$, $\underline{\Delta} f_i$, and $\overline{\Delta} f_i$ defined by (6) and (7) we have $\underline{\alpha}_i < 0, \overline{\alpha}_i > 0$, $\underline{\Delta} f_i \geq 0$ and $\overline{\Delta} f_i \geq 0$. This implies $-\overline{\alpha}_i d_i \leq g_i \leq -\underline{\alpha}_i d_i$ and thus $d_i \geq 0$. If $d_i = 0$, we also have $g_i = 0$ and $p_i = 0$. For $d_i > 0$ we obtain $\frac{1}{2}\underline{\alpha}_i \leq -\frac{g_i}{2d_i} \leq \frac{1}{2}\overline{\alpha}_i$. Then for $\hat{\alpha}_i$ and $\hat{\Delta} f_i$ defined by (8) we have $\underline{\alpha}_i < \hat{\alpha}_i < \overline{\alpha}_i$ and $\hat{\Delta} f_i < 0$, which implies $p_i = \hat{\alpha}_i = -\frac{g_i}{2d_i}$.

If $x_i = \underline{x}_i$, then $\underline{\alpha}_i = 0$ and $\overline{\alpha}_i > 0$. Then $d_i \leq 0$ implies $p_i = \underline{\alpha}_i = 0$. $p_i = -\frac{g_i}{2d_i}$ is only possible in the case that $d_i > 0$ and $0 < -\frac{g_i}{2d_i} < \overline{\alpha}_i$, which yields $g_i < 0$. Similarly, if $x_i = \overline{x}_i$, we either have $p_i = 0$ or $p_i = -\frac{g_i}{2d_i}$, $d_i > 0$, and $g_i > 0$.

This shows that $x + \alpha p \in [\underline{x}, \overline{x}]$ for sufficiently small positive α. If $p \neq 0$ then $M := \{i \mid p_i \neq 0\}$ is nonempty, hence $p^T g = -\sum_{i \in M} \frac{g_i^2}{2d_i} < 0$. □

If the algorithm stops with no change in the function value and thus the point, the following holds.

Theorem 1 *If a point x is not changed by* `fixbounds`, `reductionstep`, `subspacestep`, *and* `freebounds`, *then x is a Kuhn–Tucker point, i.e., the reduced gradient g^{red} at x, defined by*

$$g_i^{red} := \begin{cases} g_i & \text{if } \underline{x}_i < x_i < \overline{x}_i, \\ \max(0, -g_i) & \text{if } x_i = \underline{x}_i, \\ \max(0, g_i) & \text{if } x_i = \overline{x}_i, \end{cases} \tag{10}$$

is zero. Here g is the gradient of f at x.

Proof Let x be point that `fixbounds`, `reductionstep`, `subspacestep`, and `freebounds` leave the same. Then, by Lemma 1, the vector p defined by `freebounds` should be 0. Let $i \in \{1, \ldots, n\}$. If $\underline{x}_i < x_i < \overline{x}_i$, $p_i = 0$ implies $g_i = 0$. Let now $x_i = \underline{x}_i$, i.e., $0 = \underline{\alpha}_i < \overline{\alpha}_i$ with the definitions from (6) and (7). In the case $d_i \leq 0$, $\overline{\Delta} f_i = \overline{\alpha}_i (g_i + \overline{\alpha}_i d_i) \geq 0$ yields $g_i \geq -\overline{\alpha}_i d_i \geq 0$. If $d_i > 0$, we have $-\frac{g_i^2}{4d_i} \leq 0$, and $p_i = 0$ implies $\hat{\alpha}_i = -\frac{g_i}{2d_i} \leq \overline{\alpha}_i = 0$, again yielding $g_i \geq 0$. Similarly $x_i = \overline{x}_i$ implies $g_i \leq 0$. □

Under certain conditions, the result of `subspacestep` only depends on the activities.

Lemma 2 *Assume that the parameters κ in Sect. 2.2 and δ_1 in Sect. 2.4 are set to 0 and $x \in \mathbf{x} := [\underline{x}, \overline{x}]$ is the input of* `subspacestep`. *Let I be the inactive set of x, suppose that $G_{I,I}$ is positive definite and $K = \{1, \ldots, n\} \setminus I = K_l \cup K_u$ is the active set of x, where*

$$K_l := \{i \in \{1, \ldots, n\} \mid x_i = \underline{x}_i\} \text{ and } K_u := \{i \in \{1, \ldots, n\} \mid x_i = \overline{x}_i\}.$$

Moreover, assume that

$$G_{I,I}^{-1} \left(A_{:,I}^T Db - c_I - G_{I,K} x_K \right) \in [\underline{x}_I, \overline{x}_I].$$

Then the point obtained by applying `subspacestep` *to x only depends on K_l and K_u.*

Proof Since K_l and K_u are given, x_K is uniquely determined. $\kappa = 0$ implies that the reduced inactive set is equal to the inactive set I of x and thus the search direction p defined by `subspacestep` fulfils $p_K = 0$ for the active set $K = K_l \cup K_u$ and $G_{I,I} p_I = -g_I$, which by assumption yields the unique solution $p_I = -G_{I,I}^{-1} g_I$. Then $x_I + p_I = x_I - G_{I,I}^{-1}(c_I + G_{I,:}x - A_{:,I}^T Db) = -G_{I,I}^{-1}(c_I + G_{I,K}x_K - A_{:,I}^T Db)$, which is independent of x_I and in $[\underline{x}_I, \overline{x}_I]$ by assumption. The line search along $x + \alpha p$ yields $\alpha^* = 1$ since $x + p$ is the minimizer of $f(y)$ s.t. $y_K = x_K$, $y_I \in [\underline{x}_I, \overline{x}_I]$. □

Corollary 1 *Assume that G is positive definite and the parameters κ in Sect. 2.2 and δ_1 in Sect. 2.4 are set to 0. If a point x with the correct activities has been found by* `fixbounds`*, then* `subspacestep` *yields the unique minimizer.*

Since `fixbounds`, `reductionstep` and `subspacestep` cannot free any bounds and the number of active variables cannot increase indefinitely, it has to happen from time to time that `reductionstep` and `subspacestep` do not change the activities and `freebounds` is called.

Theorem 2 *If G is positive definite and the parameters κ in Sect. 2.2 and δ_1 in Sect. 2.4 are set to 0, the algorithm stops after finitely many iterations even if* `tol` $= 0$ *and* `maxit` $= \infty$ *and the global minimizer of the problem is obtained.*

Proof Let x be the current point before the application of `freebounds`, I its inactive set, K its active set and g the corresponding gradient. If `subspacestep` has not changed the active set, two cases are possible. In the first case we have $p = 0$, i.e., the search direction of `subspacestep` is zero. Then $p_I = 0$, $p_K = 0$ and $G_{I,I}$ nonsingular by assumption imply $g_I = c_I + G_{I,:}x - A_{:,I}^T Db = 0$, and we obtain $x_I = G_{I,I}^{-1}(A_{:,I}^T Db - c_I - G_{I,K}x_K)$, i.e., x_I is uniquely determined by the activities x_K.

In the second case, the line search into the direction of the minimizer of $f(y)$ s.t. $y_K = x_K$ yields a point x with x_I in the interior of $[\underline{x}_I, \overline{x}_I]$, which implies that x_I is the optimal point in the subspace, i.e., again $x_I = G_{I,I}^{-1}(A_{:,I}^T Db - c_I - G_{I,K}x_K)$.

Therefore, in both cases, only a single choice of x_I is possible for every choice of x_K. The algorithm cannot cycle because the objective function is strictly decreasing from one iteration of Algorithm 6 to the next. Since there are only finitely many possible active sets, `freebounds` is called only finitely often. Therefore the algorithm stops, the assumptions of Theorem 1 are satisfied and we obtain a Kuhn–Tucker point, which is the global minimizer of the problem by assumption. □

If $\kappa > 0$, the reduced inactive set may be smaller than the full inactive set, while if $\delta_1 > 0$, the Hessian is modified. In these two cases, as well as in the case that G is not positive definite, it is conceivable in exact arithmetic that we may perform infinitely many iterations involving the same sets K_u and K_l. However, in floating-point arithmetic this is not possible since the objective function is strictly decreasing in each iteration. In practice, a properly chosen tolerance κ (and in some cases also a

properly chosen $\delta_1 > 0$, cf. the end of Sect. 5.1) improves the numerical stability of the algorithm and leads to a significant improvement in speed, without impairing the convergence properties to a local minimizer.

4 Solving strictly convex quadratic programs

We consider the strictly convex quadratic program subject to linear constraints

$$\begin{array}{l} \min\ f(x) := c^T x + \frac{1}{2} x^T G x \\ \text{s.t.}\ \ (Ax)_I \geq b_I, \quad (Ax)_E = b_E, \end{array} \tag{11}$$

where $c \in \mathbb{R}^n$, G is a symmetric, positive definite $n \times n$ matrix, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and (I, E) is a partition of $\{1, \ldots, m\}$ with $|E| \leq n$.

Theorem 3 *Let x be a feasible point of (11) and let y be a feasible point of the bound constrained quadratic program*

$$\begin{array}{l} \min\ d(y) := \frac{1}{2}(A^T y - c)^T G^{-1}(A^T y - c) - b^T y \\ s.t.\ \ y_I \geq 0. \end{array} \tag{12}$$

Then

$$f(x) + d(y) \geq 0,$$

with equality iff x and y are optimal solutions of (11) and (12), respectively. In this case,

$$Gx + c = A^T y, \tag{13}$$

and the complementarity conditions

$$(Ax - b)_i y_i = 0 \quad \textit{for all } i \in I \tag{14}$$

hold.

Proof Write $r := A^T y - c - Gx$. Then

$$\begin{aligned} f(x) + d(y) &= c^T x + \frac{1}{2} x^T G x + \frac{1}{2}(r + Gx)^T G^{-1}(r + Gx) - b^T y \\ &= (c + Gx)^T x + \frac{1}{2} r^T G^{-1} r + r^T x - b^T y = \frac{1}{2} r^T G^{-1} r + y^T (Ax - b) \\ &\geq y^T (Ax - b) = \sum_{i \in I} (Ax - b)_i y_i \geq 0. \end{aligned}$$

Equality is possible only if $r = 0$ and $(Ax - b)_i y_i = 0$ for all $i \in I$. □

Together with the feasibility constraints, (13) and (14) are the common primal-dual optimality conditions for (11) and (12). Thus these problems are dual to each other,

and we may find the solution of (11) by first solving (12) with a bound constrained solver and then computing

$$x = G^{-1}(A^T y - c).$$

This is how MINQ8 solves strictly convex problems of the form (11).

Since G is positive definite, (11) has a unique solution if it is feasible. However, (12) is not necessarily strictly convex, and may have a direction p of infinite descent. Since

$$d(y + \alpha p) = d(y) + \alpha\big((A^T y - c)^T G^{-1} A^T - b^T\big)p + \frac{1}{2}\alpha^2 (A^T p)^T G^{-1} A^T p,$$

this is possible only if

$$A^T p = 0, \quad b^T p > 0, \quad p_i \geq 0 \quad \text{for all } i \in I. \tag{15}$$

But in this case, any feasible x satisfies

$$0 > (Ax - b)^T p = \sum_{i \in I} (Ax - b)_i p_i \geq 0,$$

contradiction. Thus p defines a certificate of infeasibility for (11). (Numerically, due to rounding errors, one may find in place of a direction of infinite descent just a huge dual solution y with a very large negative objective function value. In this case, y is itself an approximate certificate of infeasibility, and one may check whether $p = y$ satisfies (15) to sufficient accuracy to report infeasibility.)

In general, to obtain the form (3) required by MINQ8 one has to make a spectral factorization or an LDL^T factorization of G (cf. Sect. 2). If the sparsity pattern of G is not unfavorable, this can be done in very large dimensions.

However there are two special cases that deserve a separate handling.

(i) In the special case where G is a diagonal matrix, the problem (12) has already the form required for MINQ8.
(ii) Considering the even more specialized case where $c = 0$ and $G = I_n$ in (11), we see that bound constrained optimization can also be used to find either the point with minimal Euclidean norm satisfying a system of linear equations and inequalities, or a certificate of infeasibility of this system.

5 Numerical examples

This section discusses extensive computer experiments evaluating the performance of the new algorithm. The numerical examples were carried out with MATLAB 8.3.0.532 (R2014a) for Linux on an Intel Core i5-4670S processor with 3.1 GHz. The default value for `tol` is `tol` $= 10^{-8}$. The small parameters δ_1, δ_2, and δ_3 from Sects. 2.4, 2.6, and 2.7 were set to $\delta_1 = 0$, $\delta_2 = 10^{-12}$, and $\delta_3 = 10^{-4}$, respectively. Moreover, κ in

Sect. 2.2 was set to $\kappa = 0.5$, which turned out to perform much better in practice than $\kappa = 0$ (the extended active set coincides with the active set) and slightly better than $\kappa = 1$, although the convergence proofs in Sect. 3 assume $\kappa = 0$.

In Sects. 5.1 and 5.2 we consider problems of the form (3), and Sect. 5.3 is devoted to problems of the form (11). On the first two test sets we compare MINQ8 with the default value for `tol` with MINQ5, `NewtonKKTqp`, and the MATLAB function `quadprog`. For all problems and algorithms (except for the interior-point-convex `quadprog` algorithm, which does not take a starting point), the zero vector is used as the starting point. If MINQ5 returns a nonzero error flag `ier` (if the maximal number iterations `maxit` $= 10n$ is exceeded or the algorithm incorrectly assumes that the problem is unbounded below), another call to MINQ5 is made with the result from the previous call as the starting point.

The MATLAB version `NewtonKKTqp` of the Newton-KKT interior method for indefinite quadratic programming by Absil and Tits [1] is designed to return a local minimizer of the indefinite quadratic function $f(x) = \frac{1}{2}x^T Hx + c^T x$ subject to the linear inequality constraints $Ax \leq b$, starting with a feasible initial point. We implemented the bound constraints as $Ax \leq b$, and in the case of an unconstrained problem, we set A to the $2 \times n$ zero matrix and $b := (1, 1)^T$ since the program exited with an error message if A was chosen to be empty or a one-row matrix.

The MATLAB function `quadprog` contains three algorithms for quadratic programming: interior-point-convex, trust-region-reflective, and active-set. The interior-point-convex algorithm is the default algorithm in MATLAB 8, but it handles only convex problems and does not allow the choice of an initial point. The trust-region-reflective algorithm is not applicable to unconstrained problems. We used an increased limit of 10,000 (instead of the default value 200) on the number of iterations for all examples.

5.1 Test Set 1: random sparse problems

We consider $A \in \mathbb{R}^{m\times n}$ sparse with 6 nonzero integer entries in $[-5, 5]$ per row, $c_i, b_j \in \{0, 1, 2, 3, 4, 5\}$, $i = 1, \ldots, n$, $j = 1, \ldots, m$, $\gamma = 0$, and $|D_{i,i}| \in \{1, 2, 3, 4, 5\}$, $i = 1, \ldots, n$, where all entries are chosen randomly from uniform distributions. A was always chosen to have full rank. The same problem is solved without bounds (only in the positive definite case) or on the box $[-10, 10]^n$, and the problems are modified by taking $D = D_1$, D_1 a positive definite diagonal matrix (i.e., $D_{i,i} \in \{1, 2, 3, 4, 5\}$), $D = -D_1$ and $D = ED_1$, where E is a diagonal matrix with diagonal elements randomly chosen from $\{-1, 1\}$. The problem characteristics n, m, the signs of $D_{i,i}$ and the bounds are displayed in Table 1. An empty entry means that the quantity is the same as in the previous line. This table also contains the number of activities `nact` of the best solution found among the algorithms. If the lowest f_{best} (up to 5 significant digits) was also achieved by MINQ8, `nact` is the number of exact activities (since MINQ8 easily finds exact inactivities due to `fixbounds`), and MINQ5 also finds exact inactivities for Problem 1c. In contrast, `NewtonKKT` very rarely yields exact inactivities. If the lowest function value was achieved by `NewtonKKT`, we applied `MINQ8` to the point obtained by `NewtonKKT`,

Table 1 Problem characteristics of Test Set 1 (random sparse problems)

No.	m	n	D	Bounds	nact
1a	1000	1000	+	No bounds	
1b			+	$[-10, 10]^n$	35
1c			−	$[-10, 10]^n$	1000
1d			±	$[-10, 10]^n$	888
2a	3000	3000	+	No bounds	
2b			+	$[-10, 10]^n$	112
2c			−	$[-10, 10]^n$	3000
2d			±	$[-10, 10]^n$	2684
3a	1500	1000	+	No bounds	
3b			+	$[-10, 10]^n$	0
3c			−	$[-10, 10]^n$	1000
3d			±	$[-10, 10]^n$	898
4a	4500	3000	+	No bounds	
4b			+	$[-10, 10]^n$	1
4c			−	$[-10, 10]^n$	3000
4d			±	$[-10, 10]^n$	2714
5a	2000	1000	+	No bounds	
5b			+	$[-10, 10]^n$	0
5c			−	$[-10, 10]^n$	1000
5d			±	$[-10, 10]^n$	921
6a	6000	3000	+	No bounds	
6b			+	$[-10, 10]^n$	0
6c			−	$[-10, 10]^n$	3000
6d			±	$[-10, 10]^n$	2735
7a	1000	1500	+	$[-10, 10]^n$	507
7b			−	$[-10, 10]^n$	1498
7c			±	$[-10, 10]^n$	1329
8a	3000	4500	+	$[-10, 10]^n$	1525
8b			−	$[-10, 10]^n$	4494
8c			±	$[-10, 10]^n$	3974
9a	1000	2000	+	$[-10, 10]^n$	993
9b			−	$[-10, 10]^n$	1992
9c			±	$[-10, 10]^n$	1629
10a	2500	5000	+	$[-10, 10]^n$	2498
10b			−	$[-10, 10]^n$	4981
10c			±	$[-10, 10]^n$	4402

which resulted in a point with a slightly lower function value, and we display the number of activities of that point. Note that in the case of concave problems with a singular Hessian (7–10b) it is possible for minimizers to occur also at points that are not vertices.

The results are presented in Table 2. The best function value f_{best} and the CPU time needed for executing the algorithm (excluding the time for preparing the input data) are given for all algorithms. In addition, we present the number `nit` of iterations for MINQ8 (the number of times the loop in Algorithm 6 has been executed), for `NewtonKKTqp` (where it is called `nb_iter`) and for `quadprog`, and the number `nsub` of subspace steps for MINQ5. An asterisk after the time indicates that `NewtonKKTqp` terminated with the warning "Computed a non-KKT stationary point". For `quadprog`, we report the results obtained by the interior-point-convex algorithm for the convex (i.e., positive semidefinite) problems and the ones for the trust-region-reflective algorithm for the non-convex problems. The trust-region-reflective algorithm does not accept unconstrained problems and also does not find the global minimum for six of the ten bound constrained convex problems. The active-set `quadprog` algorithm yields lower f_{best} values than the trust-region-reflective algorithm for some of the non-convex problems, but it is very slow and sometimes does not even finish within a limit of 10,000 iterations.

Problem 1a is the only problem where the first call to MINQ5 returns the not yet optimal function value -1.5510×10^5 and the error flag for a problem that is unbounded below, but the optimal function value and a zero error flag are obtained after the second call to MINQ5. If two calls to MINQ5 were made, `nsub` column of Table 2 contains two numbers separated by a plus sign.

All algorithms yield the unique optimal function value f_{best} (up to the number of digits presented here) for the six convex unconstrained problems (1–6a). Even though the bound constrained convex problems (1–6b and 7–10a) also possess a unique local and therefore also global minimum f_{best}, MINQ8 and interior-point-convex `quadprog` are the only algorithms that always manage to find the optimal f_{best}. MINQ5 finds the correct f_{best} only in the cases where the minima coincide with the unconstrained minima (3b, 5b, 6b), i.e., no bounds are active at the best point, and in a case where one bound is active (4b). `NewtonKKTqp` never achieves the correct f_{best} and sometimes, as explained at the beginning of this section, claims that it has found a non-KKT stationary point (which is actually a non-stationary point). So, in spite of using different parameter values than the ones in the convergence proof, MINQ8 does generally well on convex problems. For all unconstrained convex problems and for the convex bound constrained problems where the minima coincide with the unconstrained minima, MINQ8 only needs two iterations of the main loop, i.e., it already finds the minimizer after the first iteration (as supported by theory for $\kappa = 0$) and then makes a second one in order to be able to fulfil the stopping criterion that the function value does not change any more. However, for the rank-deficient positive semidefinite problems 7–10a, MINQ8 needs a high number of iterations compared to all other problems (and longer time than the interior-point-convex `quadprog` algorithm) since it seems to be hard to find the correct activities in such a case (a larger number of activities than for Problems 1–6b).

Non-convex problems usually have several local minimizers, and MINQ8 also finds good f_{best} values for many non-convex problems. Even though `NewtonKKTqp` frequently achieves the lowest f_{best} for non-convex problems, the points returned are not even local minimizers and the algorithm somehow gets stuck.

Table 2 Results for Test Set 1 (random sparse problems)

No.	MINQ8			MINQ5			NewtonKKTqp			quadprog (ipc/trr)		
	nit	f_{best}	t (s)	nsub	f_{best}	t (s)	nit	f_{best}	t (s)	nit	f_{best}	t (s)
1a	2	-1.5512×10^5	0.2	2 + 1	-1.5512×10^5	5.8	2	-1.5512×10^5	0.5	1	-1.5512×10^5	0.3
1b	33	-1.7493×10^3	1.8	100	-1.4030×10^3	13.1	2	2.7644×10^3	3.1*	9	-1.7493×10^3	0.8
1c	2	-2.5765×10^7	0.1	0	-2.6045×10^7	0.2	131	-2.5610×10^7	186	23	-2.2121×10^7	0.2
1d	24	-1.3744×10^7	0.5	8	-1.3099×10^7	0.4	165	-1.3580×10^7	231*	27	-1.3307×10^7	0.2
2a	2	-3.4473×10^6	1.7	1	-3.4473×10^6	72	11	-3.4473×10^6	339	1	-3.4473×10^6	2.2
2b	76	-5.3100×10^3	60	133	-5.3061×10^3	143	2	6.8150×10^3	151*	12	-5.3100×10^3	16
2c	2	-7.9075×10^7	0.5	0	-7.8353×10^7	0.7	178	-7.8482×10^7	8719	24	-6.5940×10^7	0.5
2d	40	-3.9264×10^7	2.1	4	-4.0065×10^7	1.8	204	-4.1036×10^7	1010	35	-3.8998×10^7	0.8
3a	2	6.2145×10^3	0.2	1	6.2145×10^3	3.2	2	6.2145×10^3	0.5	1	6.2145×10^3	0.4
3b	2	6.2145×10^3	0.2	1	6.2145×10^3	3.2	2	6.2891×10^3	3.0*	4	6.2145×10^3	0.9
3c	2	-3.4313×10^7	0.2	0	-3.4269×10^7	0.2	119	-3.4108×10^7	165	1011	-3.1461×10^7	7.6
3d	6	-1.6170×10^7	0.3	6	-1.5777×10^7	0.5	187	-1.6441×10^7	261*	28	-1.5474×10^7	0.3
4a	2	1.8621×10^4	2.4	2	1.8621×10^4	8	2	1.8621×10^4	12	1	1.8621×10^4	3.1
4b	2	1.8624×10^4	2.4	2	1.8624×10^4	82	2	1.8911×10^4	85*	6	1.8624×10^4	9.4
4c	2	-1.0289×10^8	0.5	0	-1.0287×10^8	1.1	159	-1.0208×10^8	7534	16	-8.5424×10^7	0.6
4d	14	-5.1231×10^7	1.2	11	-5.0978×10^7	2.4	220	-5.2754×10^7	10,530	35	-4.9071×10^7	0.9
5a	2	1.2369×10^4	0.2	1	1.2369×10^4	3.3	2	1.2369×10^4	0.5	1	1.2369×10^4	0.3
5b	2	1.2369×10^4	0.2	1	1.2369×10^4	3.2	2	1.2405×10^4	3.1*	4	1.2369×10^4	0.6
5c	2	-4.2284×10^7	0.1	0	-4.1971×10^7	0.3	107	-4.1134×10^7	151	18	-3.3676×10^7	0.2
5d	40	-2.0292×10^7	0.9	1	-1.9308×10^7	0.4	193	-1.9701×10^7	275	27	-1.8812×10^7	0.2

Table 2 continued

No.	MINQ8			MINQ5			NewtonKKTqp			quadprog (ipc/trr)		
	nit	f_{best}	t (s)	nsub	f_{best}	t (s)	nit	f_{best}	t (s)	nit	f_{best}	t (s)
6a	2	3.8614×10^4	2.8	1	3.8614×10^4	83	2	3.8614×10^4	12	1	3.8614×10^4	3.0
6b	2	3.8614×10^4	2.8	1	3.8614×10^4	83	2	3.8722×10^4	84*	1	3.8614×10^4	12
6c	2	-1.2808×10^8	0.6	0	-1.2721×10^8	1.4	157	-1.2316×10^8	7550	20	-1.0369×10^8	0.8
6d	39	-6.0433×10^7	2.4	7	-6.0547×10^7	2.4	206	-6.0894×10^7	10,057	44	-5.7184×10^7	1.5
7a	850	-1.5373×10^4	63	570	-1.3517×10^4	57	2	3.2184×10^3	18*	14	-1.5373×10^4	2.4
7b	2	-2.8071×10^7	0.1	0	-2.8487×10^7	0.3	169	-2.8533×10^7	931	26	-2.3291×10^7	0.3
7c	44	-1.5548×10^7	1.4	20	-1.5516×10^7	1.3	194	-1.6388×10^7	1073*	31	-1.5226×10^7	0.3
8a	2552	-5.0900×10^4	1936	1012	-4.9023×10^4	632	2	3.6107×10^4	542*	11	-5.0900×10^4	21
8b	2	-9.1112×10^7	0.5	0	-9.1050×10^7	1.2	281	-9.0852×10^7	45,062	21	-7.4709×10^7	1.0
8c	108	-4.9305×10^7	8.0	21	-4.9939×10^7	3.6	237	-5.1612×10^7	35,895	41	-4.7165×10^7	1.2
9a	1297	-2.9951×10^4	101	238	-2.8128×10^4	58	2	8.8475×10^3	44*	11	-2.9951×10^4	2.5
9b	2	-3.1822×10^7	0.2	0	-3.1680×10^7	0.5	200	-3.2713×19^7	2620	18	-2.7303×10^7	0.3
9c	85	-1.7780×10^7	2.7	155	-1.8182×10^7	5.3	230	-1.8934×10^7	2958	38	-1.7580×10^7	0.5
10a	3428	-7.6119×10^4	1777	857	-7.3865×10^4	411	2	3.0464×10^4	690*	12	-7.6119×10^4	19
10b	2	-8.2895×10^7	0.5	0	-8.3253×10^7	1.1	232	-8.4774×10^7	48,471*	18	-7.0562×10^7	0.7
10c	258	-4.8432×10^7	19	134	-4.4983×10^7	11	249	-4.7643×10^7	51,048	34	-4.2740×10^7	1.4

Table 3 Results for MINQ8 with $\delta_1 = 10^{-10}$ on some problems of Test Set 1

No.	MINQ8		
	nit	f_{best}	t (s)
7a	850	-1.5373×10^4	67
7b	2	-2.8071×10^7	0.2
7c	59	-1.5816×10^7	1.3
8a	2503	-5.0900×10^4	1738
8b	2	-9.1112×10^7	0.5
8c	145	-5.0751×10^7	9.4
9a	1350	-2.9951×10^4	115
9b	2	-3.1822×10^7	0.2
9c	52	-1.8673×10^7	1.6
10a	3489	-7.6119×10^4	1674
10b	2	-8.2895×10^7	0.5
10c	126	-4.6840×10^7	9.3

Since the performance of each algorithm is basically the same on each problem class of the problems reported here (and several more that are not reported), the random element in the problems does not seem to be crucial and we did not generate several problems with the same n and m.

We also want to show the behavior of MINQ8 for unbounded problems. Applying MINQ8 to the Problem 7a modified by minimizing the function on $\mathbb{R}^n$ instead of $[-10, 10]^n$ yielded a warning "The problem is unbounded below", a corresponding error flag and a finite point with function value $f \approx -5 \times 10^{307}$ before entering the loop in Algorithm 6 (nit $= 0$).

For some of the rank-deficient problems 7–10, in particular for the indefinite problems 7–10c, MATLAB produced several warnings "Matrix is singular to working precision" when solving (9) with the backslash operator, and the same occurred twice for Problem 9a. This is a situation where setting δ_1 to a nonzero value makes sense since there are no MATLAB warnings any more and thus the stability is improved. However, if δ_1 is set to a nonzero value, some of the unconstrained convex problems in this test set and in the following subsection might take three iterations to finish instead of two. In Table 3 we present the results for MINQ8 with $\delta_1 = 10^{-10}$ for Problems 7–10; for all other problems the results do not change significantly. Different local minima are obtained for Problems 7–10c, but for the other problems f_{best} stays the same up to the number of digits displayed here.

5.2 Test Set 2: quadratic problems from CUTEr

To complement the test set with random functions we coded some quadratic problems from the CUTEr test set [8]. We use the default bounds and the default starting point for all problems. The names, dimensions, bounds, starting points, and numbers of

Table 4 Problem characteristics of Test Set 2

No.	Name	n	m	Bounds	Starting point	`nact`
C1	BIGGSB1	1000	$n+1$	$[0, 0.9]^n \times \mathbb{R}$	$(0, \ldots, 0)$	999
C2	CHENHARK	1000	$n+2$	$[0, \infty)^n$	$(0.5, \ldots, 0.5)$	499
C3	CVXBQP1	1000	n	$[0.1, 10]^n$	$(0.5, \ldots, 0.5)$	1000
C4		10,000				10,000
C5	DIXON3DQ	10,000	n	No bounds	$(-1, \ldots, -1)$	
C6	DQDRTIC	5000	n	No bounds	$(3, \ldots, 3)$	
C7	HARKERP2	1000	$2n$	$[0, \infty)^n$	$(1, 2, \ldots, n)$	999
C8	NCVXBQP1	1000	n	$[0.1, 10]^n$	$(0.5, \ldots, 0.5)$	1000
C9		10,000				10,000
C10	NCVXBQP2	1000	n	$[0.1, 10]^n$	$(0.5, \ldots, 0.5)$	993
C11		10,000				9935
C12	NCVXBQP3	1000	n	$[0.1, 10]^n$	$(0.5, \ldots, 0.5)$	984
C13		10,000				9839
C14	PENTDI	5000	$4n-6$	$[0, \infty)^n$	$(1, \ldots, 1)$	4998
C15	QUDLIN	1200	$n+1$	$[0, 10]^n$	$(1, \ldots, 1)$	1200
C16		5000				5000
C17	TOINTQOR	50	83	No bounds	$(0, \ldots, 0)$	
C18	TRIDIA	50	n	No bounds	$(1, \ldots, 1)$	

activities at the global minimizers of the problems are listed in Table 4. In order to save space in the presentation of the results, we number the problems from C1 to C18. For all problems we have $m \geq n$.

Table 5 contains the results. A failure of `NewtonKKTqp` is indicated by a dash: DQDRTIC terminated with an error message, the 10,000-dimensional problems could not be solved with `NewtonKKTqp` due to limitations in memory, and CHENHARK produced a warning (`large k!!!`) and never finished. NCVXBQP1–3 and QUDLIN are the only non-convex problems in this test set. Since on this test set interior-point-convex `quadprog` occasionally fails to find the global minimum and most of the problems are convex, we report the results of interior-point-convex `quadprog` for all convex problems as well as the results of trust-region-reflective `quadprog` for all constrained problems.

The active-set `quadprog` algorithm was also tested, but it is inferior to the `quadprog` results presented in Table 5. It does not solve the 10,000-dimensional problems due to memory limitations, and it takes hours to solve PENTDI and the 5000-dimensional QUDLIN problem.

5.3 Test Set 3: separable quadratic programs

To test the behavior of MINQ8 on problems of the form discussed in Sect. 4, we apply MINQ8 and MINQ5 to separable quadratic problems with linear constraints by

Table 5 Results of Test Set 2

No.	MINQ8			MINQ5			NewtonKKTqp		
	nit	f_{best}	t (s)	nsub	f_{best}	t (s)	nit	f_{best}	t (s)
C1	505	0.0150	5.5	502	0.0150	13	3	1.5	3.2*
C2	2	-2	4.6	1	-2.0000	0.3	–	–	–
C3	2	2.2523×10^{4}	0.1	0	2.2523×10^{4}	0.1	13	2.2523×10^{4}	19
C4	2	2.2502×10^{6}	0.1	0	2.2502×10^{6}	1.3	–	–	–
C5	2	0	0.8	1	4.4404×10^{-10}	15	–	–	–
C6	2	0	0.3	2	0	3.8	–	–	–
C7	103	-0.5000	2.5	2	-0.5000	0.8	44	-0.5000	37
C8	2	-1.9868×10^{8}	0.1	0	-1.9868×10^{8}	0.1	35	-1.9868×10^{8}	52
C9	2	-1.9855×10^{10}	1.1	0	-1.9855×10^{10}	1.2	–	–	–
C10	3	-1.3339×10^{8}	0.1	2	-1.3339×10^{8}	0.2	66	-1.3339×10^{8}	97
C11	5	-1.3340×10^{10}	1.6	2	-1.3340×10^{10}	2.4	–	–	–
C12	5	-6.5557×10^{7}	0.1	2	-6.5776×10^{7}	0.2	66	-6.5790×10^{7}	96
C13	9	-6.5361×10^{9}	2.2	2	-6.5584×10^{9}	2.8	–	–	–
C14	3	-0.7500	1.1	1	-0.7500	12	22	-0.7500	793
C15	2	-7.2×10^{7}	0.2	0	-7.2×10^{7}	0.2	2	-8.0902×10^{6}	7.9*
C16	2	-1.2500×10^{9}	6.3	0	-1.2500×10^{9}	5.9	2	-1.2882×10^{8}	668*
C17	2	1.1755×10^{3}	0.1	1	1.1755×10^{3}	0.1	2	1.1755×10^{3}	0.1
C18	2	3.6887×10^{-58}	0.1	1	1.6021×10^{-13}	0.3	2	0	0.1

Table 5 continued

No.	quadprog (ipc)			quadprog (trr)		
	nit	f_{best}	t (s)	nit	f_{best}	t (s)
C1	8	0.0152	0.2	221	0.0150	2.9
C2	7	-2.0000	0.2	18	-2.0000	0.6
C3	8	2.2523×10^{4}	0.3	13	2.2523×10^{4}	0.3
C4	11	2.2502×10^{6}	2.7	24	2.2502×10^{6}	1.9
C5	1	0	1.2	–	–	–
C6	1	1.2326×10^{-30}	0.4	–	–	–
C7	14	-0.4912	14	24	-0.5000	1.0
C8	–	–	–	14	-1.9868×10^{8}	0.2
C9	–	–	–	21	-1.9855×10^{10}	2.0
C10	–	–	–	18	-1.3339×10^{8}	0.1
C11	–	–	–	32	-1.3340×10^{10}	1.8
C12	–	–	–	15	-6.5791×10^{7}	0.2
C13	–	–	–	24	-6.5593×10^{9}	1.9
C14	8	-0.7362	0.6	25	-0.7500	0.7
C15	–	–	–	10	-7.2000×10^{7}	0.1
C16	–	–	–	9	-1.2500×10^{9}	0.4
C17	1	1.1755×10^{3}	0.2	–	–	–
C18	1	2.2204×10^{-16}	0.2	–	–	–

solving the dual problems. For comparison we only apply `quadprog` to the problems since `NewtonKKTqp` requires a feasible starting point.

Example 1 We consider the minimization of a separable quadratic form subject to linear constraints

$$\begin{aligned} &\min\ c^T x + \tfrac{1}{2} x^T D x \\ &\text{s.t.}\ \ (Ax)_I \geq b_I, \quad (Ax)_E = b_E, \end{aligned} \tag{16}$$

where $c \in \mathbb{R}^n$, D is a positive definite $n \times n$ diagonal matrix, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and (I, E) is a partition of $\{1, \ldots, m\}$ with $|E| \leq n$. This problem is a special case of (11) and we proceed as described in Sect. 4.

We construct the instances in the following way. We first choose the elements of $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, $y_{\text{des}}, s \in \mathbb{R}^m$ and the diagonal elements of D from a uniform distribution on $[0, 1]$. Let $I = N \cup N'$, where N is the set of nonactive and N' the set of active inequalities, and $|N'| + |E| \leq \min(m, n)$. Then we set $y_{\text{des},N} = 0$, $s_{E \cup N'} = 0$, $x_{\text{des}} := D^{-1}(A^T y_{\text{des}} - c)$, and $b := Ax_{\text{des}} - s$. In the case $N = \emptyset$ (which can only happen if $m \leq n$), all indices are active or equalities and $b = Ax_{\text{des}}$.

For the solution vector x we define the vector $r \in \mathbb{R}^n$ by $r_E := (Ax - b)_E$ and $r_I := \min((Ax - b)_I, 0)$, i.e., r measures the violation of the constraints. Moreover, we use the abbreviation $\Delta x := x - x_{\text{des}}$. For MINQ5, we use the program `minqsep.m` included in the MINQ5 package. It applies MINQ5 to the dual problem, and one step of iterative refinement is applied if $|r| \leq \text{nnz}(A)\varepsilon(|A||x| + |b|)$ (with componentwise absolute value and inequalities) is violated, where $\text{nnz}(A)$ denotes the number of nonzeros of the matrix A and ε the machine precision. For MINQ8, we solve the dual problem with `tol` $= \varepsilon = 2.2204 \times 10^{-16}$ (the machine precision) instead of the default value since the default value does not yet yield the optimal f_{best} for the original problems for the test cases with $I \neq \emptyset$. (Note that MINQ8 and MINQ5 are not applied to the original problems but to the dual ones.) We also apply one step of iterative refinement in the above-mentioned case.

Table 6 contains the problem characteristics of the problems of Example 1. For each value of (m, n), we consider one instance with empty N' and one with nonempty N'. In Table 7 the test results for MINQ8, MINQ5 and the interior-point-convex `quadprog` algorithm are presented; two problems cannot be solved with `quadprog` due to memory problems.

All three algorithms find approximately feasible points with the same function value up to at least 5 significant digits (not displayed here). Most of the points are very close to x_{des}, with the exception of a few solutions produced by MINQ8 and `quadprog` (usually the problems that converge most slowly). The active-set `quadprog` algorithm is faster than the interior-point-convex algorithm (but not faster than MINQ8) for 9 of the 12 problems that are solved by `quadprog`, but it takes a long time to solve S9 and also S3 and therefore we do not display the results. The trust-region-reflective `quadprog` algorithm handles problems with only bounds, or only linear equality constraints, but not both and is therefore not applicable.

Table 6 Problem characteristics of Example 1

No.	m	n	$\lvert E\rvert$	$\lvert N'\rvert$
S1	700	1000	500	10
S2	700	1000	700	0
S3	2000	3000	1000	300
S4	2000	3000	2000	0
S5	7000	10,000	5000	100
S6	7000	10,000	7000	0
S7	1000	1000	500	100
S8	1000	1000	1000	0
S9	3000	3000	1500	300
S10	3000	3000	3000	0
S11	1000	700	500	100
S12	1000	700	700	0
S13	3000	2000	1500	300
S14	3000	2000	2000	0

Example 2 We consider the problem of finding a point of minimal norm satisfying a given system of linear inequalities,

$$\begin{array}{ll} \min & \frac{1}{2}\|x\|_2^2 \\ \text{s.t.} & Ax \geq b. \end{array} \tag{17}$$

This problem is a special case of (16) with $c = 0$, $D = I_n$, and $E = \emptyset$, and we consider the three cases $m < n$, $m = n$ and $m > n$. The entries of A and b are chosen from a uniform distribution on [0, 1]. We applied MINQ8, MINQ5, and the active-set `quadprog` algorithm and present the results in Table 8. (The interior-point-convex algorithm yielded suboptimal feasible points for all problems.) The three algorithms produce essentially the same solutions. We display the maximal violation $\|r\|_\infty$ of the constraints and the CPU time needed for the computation for all three algorithms and the number of iterations for MINQ8. The last block of Table 8 contains the minimal function value $\frac{1}{2}\|x\|_2^2$ of the solution and the number of constraints that are numerically active ($|(Ax)_i - b_i| \leq 10^{-8}$); these quantities are the same (up to the number of digits presented here) for the successful algorithms. `quadprog` does not solve the three largest problems due to memory problems. All solutions satisfy the constraints very well.

5.4 Discussion summary

The numerical examples show that the MINQ8 performs well and is competitive with other quadratic programming algorithms implemented in MATLAB. MINQ8 is good at solving convex problems and usually finds good local minima for non-convex problems. The CPU times needed for the computations are competitive as well. MINQ8

Table 7 Results for Example 1

No.	MINQ8				MINQ5			quadprog (ipc)		
	nit	$\|r\|_\infty$	$\|\Delta x\|_\infty$	t (s)	$\|r\|_\infty$	$\|\Delta x\|_\infty$	t (s)	$\|r\|_\infty$	$\|\Delta x\|_\infty$	t (s)
S1	31	1.98×10^{-9}	3.95×10^{-10}	0.6	4.07×10^{-9}	9.90×10^{-10}	7.2	2.56×10^{-9}	7.22×10^{-4}	39
S2	2	4.42×10^{-9}	1.11×10^{-9}	0.1	6.29×10^{-9}	2.39×10^{-9}	2.4	6.52×10^{-9}	1.43×10^{-9}	3.5
S3	308	6.24×10^{-8}	1.27×10^{-3}	48	5.22×10^{-8}	1.21×10^{-8}	53	7.08×10^{-8}	3.50×10^{-3}	524
S4	3	3.35×10^{-8}	3.83×10^{-9}	1.8	4.10×10^{-8}	6.46×10^{-9}	47	5.40×10^{-8}	5.48×10^{-9}	49
S5	99	7.75×10^{-7}	5.26×10^{-8}	371	1.06×10^{-6}	7.82×10^{-8}	9938	–	–	–
S6	4	2.65×10^{-6}	2.50×10^{-7}	57	1.13×10^{-6}	1.16×10^{-7}	3921	–	–	–
S7	101	8.61×10^{-9}	2.33×10^{-9}	2.3	7.92×10^{-9}	8.73×10^{-10}	33	8.15×10^{-9}	9.82×10^{-4}	67
S8	3	6.98×10^{-9}	1.63×10^{-8}	0.3	5.36×10^{-9}	3.00×10^{-8}	6.9	4.89×10^{-9}	2.74×10^{-8}	4.1
S9	898	4.10×10^{-8}	4.41×10^{-2}	298	5.03×10^{-8}	2.24×10^{-8}	761	4.10×10^{-8}	2.30×10^{-3}	1165
S10	2	8.75×10^{-8}	1.82×10^{-7}	3.2	8.38×10^{-8}	1.83×10^{-7}	144	9.31×10^{-8}	3.28×10^{-7}	50
S11	142	1.34×10^{-9}	2.98×10^{-10}	3.3	1.51×10^{-9}	3.81×10^{-10}	35	1.46×10^{-9}	6.56×10^{-4}	33
S12	3	2.21×10^{-9}	5.47×10^{-9}	0.1	5.12×10^{-9}	1.75×10^{-8}	12	3.03×10^{-9}	4.60×10^{-9}	4.8
S13	86	7.08×10^{-8}	7.45×10^{-9}	21	5.49×10^{-8}	5.98×10^{-9}	59	4.75×10^{-8}	2.18×10^{-4}	529
S14	2	1.91×10^{-8}	2.65×10^{-8}	1.7	1.26×10^{-7}	2.12×10^{-7}	255	2.33×10^{-8}	1.21×10^{-7}	79

Table 8 Results for Example 2

m	n	MINQ8			MINQ5		quadprog (as)		Solution	
		nit	$\|r\|_\infty$	t (s)	$\|r\|_\infty$	t (s)	$\|r\|_\infty$	t (s)	$\frac{1}{2}\|x\|_2^2$	nact
700	1000	16	1.55×10^{-15}	0.3	4.11×10^{-15}	0.2	2.33×10^{-15}	4.6	1.9467×10^{-3}	17
2000	3000	58	2.11×10^{-15}	2.6	4.33×10^{-15}	1.3	2.55×10^{-15}	150	6.6330×10^{-4}	25
7000	10,000	45	7.77×10^{-15}	40	7.33×10^{-15}	30	–	–	1.9857×10^{-4}	55
1000	1000	30	2.55×10^{-15}	0.3	3.55×10^{-15}	0.3	2.33×10^{-15}	4.3	1.9656×10^{-3}	18
3000	3000	48	3.33×10^{-15}	3.9	4.00×10^{-15}	2.8	2.66×10^{-15}	188	6.5571×10^{-4}	38
10,000	10,000	66	5.77×10^{-15}	77	6.55×10^{-15}	62	–	–	1.9857×10^{-4}	62
1000	700	24	8.88×10^{-16}	0.2	3.77×10^{-15}	0.2	2.78×10^{-15}	2.0	2.8136×10^{-3}	16
3000	2000	31	3.66×10^{-15}	2.3	3.44×10^{-15}	2.5	2.55×10^{-15}	49	9.9814×10^{-4}	31
10,000	7000	52	5.88×10^{-15}	51	6.11×10^{-15}	46	–	–	2.8584×10^{-4}	62

and MINQ5 are the only of the tested algorithms that are applicable to all problems of our test sets; the other algorithms contain restrictions concerning the problem class and more severe memory restrictions in MATLAB. MINQ8 solves bound constrained quadratic programs well with the default value of `tol`, but for the separable quadratic programs from Sect. 5.3 a lower default value of `tol` is needed. The default value of n (the dimension of the problem) for `maxit` turned out to be sufficient for our test problems as well. The fact that MINQ8 easily produces points with exact activities seems to be an advantage for bound constrained problems.

Acknowledgements Open access funding provided by University of Vienna.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Absil, P.-A., Tits, A.L.: Newton-KKT interior-point methods for indefinite quadratic programming. Comput. Optim. Appl. **36**, 5–41. http://www.montefiore.ulg.ac.be/~absil/Publi/indefQP.htm (2007)
2. Dietz, T., Verl, A.: Simulation of the stopping behavior of industrial robots using a complementarity-based approach. In: IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 428–433. IEEE (2011)
3. Fernandes, R., Fraser, R., Latifovic, R., Cihlar, J., Beaubien, J., Du, Y.: Approaches to fractional land cover and continuous field mapping: a comparative assessment over the BOREAS study region. Remote Sens. Environ. **89**, 234–251 (2004)
4. Floudas, C.A., Pardalos, P.M.: A Collection of Test Problems for Constrained Global Optimization Algorithms. Springer, Berlin (1990)
5. Frank, M., Wolfe, P.: An algorithm for quadratic programming. Nav. Res. Logist. Q. **3**, 95–110 (1956)
6. Friedlander, M.P., Leyffer, S.: Global and finite termination of a two-phase augmented Lagrangian filter method for general quadratic programs. SIAM J. Sci. Comput. **30**, 1706–1729 (2008)
7. Gao, Y., Choudhary, A., Hua, G.: A nonnegative sparsity induced similarity measure with application to cluster analysis of spam images. In: 35th IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 5594–5597. IEEE (2010)
8. Gould, N.I.M., Orban, D., Toint, Ph.L.: CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited. CERFACS technical report no. TR/PA/01/04 (2004)
9. Gould, N.I.M., Toint, Ph.L.: A quadratic programming page. http://www.numerical.rl.ac.uk/people/nimg/qp/qp.html. Accessed 21 July 2015
10. Haruno, M., Wolpert, D.M.: Optimal control of redundant muscles in step-tracking wrist movements. J. Neorophysiol. **94**, 4244–4255 (2005)
11. Horst, R., Pardalos, P.M., Thoai, N.V.: Introduction to Global Optimization, 2nd edn. Kluwer, Dordrecht (2000)
12. Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search. J. Glob. Optim. **14**, 331–355 (1999)
13. Huyer, W., Neumaier, A.: SNOBFIT—stable noisy optimization by branch and fit. ACM Trans. Math. Softw. **35**(2), Article 9 (2008)
14. Kannan, A., Wild, S.M.: Obtaining quadratic models of noisy functions. Preprint ANL/MCS-P1975-1111, Argonne National Laboratory (2012)
15. Kanzow, C., Petra, S.: Projected filter trust region methods for a semismooth least squares formulation of mixed complementarity problems. Optim. Methods Softw. **22**(5), 713–735 (2007)
16. Kybic, J.: High-dimensional entropy estimation for finite accuracy data: R-NN entropy estimator. In: Karssemeijer, N., Lelieveldt, B. (eds.) Information Processing in Medical Imaging. Lecture Notes in Computer Science, vol. 4584, pp. 569–580. Springer, Berlin (2007)

17. Lee, J., Leyffer, S. (eds.): Mixed Integer Nonlinear Programming. IMA Volumes in Mathematics and its Applications, vol. 154. Springer, Berlin (2012)
18. Lin, Y.Y., Pang, J.-S.: Iterative methods for large convex quadratic programs: a survey. SIAM J. Control Optim. **25**, 383–411 (1987)
19. Liu, W., Brugger, J., McPhail, D.C., Spiccia, L.: A spectrophotometric study of aqueous copper(I)-chloride complexes in LiCl solutions between 100 °C and 250 °C. Geochim. Cosmochim. Acta **66**(20), 3615–3633 (2002)
20. Neumaier, A.: MINQ: general definite and bound constrained indefinite quadratic programming. Web document. http://www.mat.univie.ac.at/~neum/software/minq/ (1998)
21. Olshansky, Y., Turkel, E.: Simultaneous scatterer shape estimation and partial aperture far-field pattern denoising. Commun. Comput. Phys. **11**(2), 271–284 (2012)
22. Roos, M., Rothe, J., Scheuermann, B.: How to calibrate the scores of biased reviewers by quadratic programming. In: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, pp. 255–260. AAAI (2011)
23. Smit, W.J., Barnard, E.: Continuous speech recognition with sparse coding. Comput. Speech Lang. **23**, 200–219 (2009)
24. Voglis, C., Lagaris, I.E.: BOXCQP: an algorithm for bound constrained convex quadratic problems. In: 1st International Conference from Scientific Computing to Computational Engineering. IC-SCCE (2004)
25. Xu, S.: A non-interior path following method for convex quadratic programming problems with bound constraints. Comput. Optim. Appl. **27**, 285–303 (2004)