New Algorithms for Singly Linearly Constrained Quadratic Programs Subject to Lower and Upper Bounds *

Yu-Hong Dai † Roger Fletcher ‡

November 19, 2003

Numerical Analysis Report NA/216, November 2003

Abstract

There are many applications related to singly linearly constrained quadratic programs subjected to upper and lower bounds. In this paper, a new algorithm based on secant approximation is provided for the case in which the Hessian matrix is diagonal and positive definite. To deal with the general case where the Hessian is not diagonal, a new efficient projected gradient algorithm is proposed. The basic features of the projected gradient algorithm are: 1) a new formula is used for the stepsize; 2) a recently-established adaptive nonmonotone line search is incorporated; and 3) the optimal stepsize is determined by quadratic interpolation if the nonmonotone line search criterion fails to be satisfied. Numerical experiments on large-scale random test problems and some medium-scale quadratic programs arising in the training support vector machines demonstrate the usefulness of these algorithms.

1 Introduction

In this paper we consider the following quadratic programming problem

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{l} \le \mathbf{x} \le \mathbf{u} \\ & \mathbf{a}^T\mathbf{x} = b, \end{array} \tag{1.1}$$

*This work was supported by the EPRSC in UK (no. GR/R87208/01) and the Chinese NSF grant (no. 10171104).

†State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering computing, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, PO Box 2719, Beijing 100080, PR China. Email: dyh@lsec.cc.ac.cn

‡Department of Mathematics, University of Dundee, Dundee DD1 4HN, Scotland, UK. Email: fletcher@maths.dundee.ac.uk

where $A \in \mathbb{R}^{n \times n}$ is symmetric but may be indefinite, $\mathbf{a}$, $\mathbf{c}$, $\mathbf{l}$ and $\mathbf{u}$ (with $\mathbf{l} < \mathbf{u}$) are vectors in $\mathbb{R}^n$, and b is a scalar. Thus there is a single linear equality constraint in the problem, in addition to simple bounds (box constraints) on the variables. We refer to this as the SLBQP problem.

There are many real-world instances of SLBQP problems. For example, some problems in multicommodity network flow and logistics have the form (1.2) in which the matrix A is diagonal (see Held *et al* [11], Meyer [14], Pardalos and Rosen [17]). The general SLBQP is also solved in training the learning methodology known as Support Vector Machine (SVM) (see Vapnik [23]). This SVM learning methodology has empirically been shown to give good performance on a wide variety of problems such as handwritten character recognition, face detection, pedestrian detection, and text categorization (see Platt [18]).

In this paper, we consider new algorithms for solving SLBQP problems. In the special case that A is diagonal and positive definite, it is possible to have algorithms that only require $O(n)$ operations per iteration. Helgason *et al* [12] propose an $O(n \log n)$ algorithm that is based on appropriate manipulation of the corresponding Kuhn-Tucker conditions. Several linear time algorithms have been proposed by Brucker [3], Calamai and Moré [4], and Pardalos and Kovoor [16] that are based on a similar characterization of the solution. The algorithms in [3] and [4] are based on binary search. For large practical problems, [16] proposes a randomized algorithm that runs in expected linear time and has a very small time constant. A new simple and very efficient algorithm based on secant approximation is proposed in Section 2 of this paper for this special case.

Next we consider general SLBQP problems in which A is non-diagonal. One possibility is to solve these using a standard solver for the general QP problem. This approach is adequate for small problems, but may not work well if the dimension n is large. For example, active set methods require many iterations if the initial active set and the optimal active set are significantly different since only one constraint is dropped or added at each iteration. Also the methods become inefficient if the reduced Hessian matrix becomes large. A successful way of avoiding these difficulties for box constrained QP (BQP) problems has been to use gradient projection methods. This idea dates back a long way, and various references can be found in Dai and Fletcher [6], where we use this idea in conjunction with the Barzilai-Borwein [1] (BB) stepsize formulae. For BQP problems, gradient projection methods take advantage of the fact that projection on to the box is a trivial calculation. However, a line search must be used to ensure global convergence. In fact, gradient projection methods are attractive for any type of optimization problem in which projection on to a (convex) feasible set can be done efficiently. Birgin, Martínez, and Raydan [2] propose a general framework based on gradient projection, the BB formula, and a non-monotonic line search. For the general SLBQP problem, projection on to the feasible set of (1.1) can also be done very efficiently using one of the algorithms described in the previous paragraph. Thus in Section 3 of this paper we explore the practical implementation of this type of method. A new choice (3.4) for the stepsize is proposed which seems to be a promising alternative to the BB formula. Following our experience in [6], we use a different adaptive non-monotonic line search. In addition, the optimal stepsize is determined by quadratic interpolation if the nonmonotone line search criterion fails to be satisfied.

In Section 4 we test our approach on randomly generated test problems of moderately large dimension, both for positive definite and for indefinite Hessian matrices. We find that the new stepsize (3.4) with $m = 2$ is superior to the BB formula. As in [6], we also observe that the adaptive non-monotonic line search works better than that used in [2]. We also explore the real-world applications of SVM learning methodology. Such problems can be very large indeed ($n \approx 10^6$, say) and the matrix A is dense. In this case, standard QP solvers based on explicit storage of A can be impractical due to the difficulty of storing the Hessian. Even in the case that n is not all that large, which we consider here, standard QP solvers can be inefficient. Serafini, Zanghirati and Zanni [21] have implemented two projection gradient algorithms for medium-scale SVM problems and have reported numerical results that are much better than are obtained with the QP solvers pr_LOQO and MINOS, as suggested in the SVM^{light} package by Joachims [13]. Moreover, in combination with a decomposition technique, it is shown in [21] that projection algorithms are superior to the SVM^{light} software (version 3.5) equipped with pr_LOQO for SVM problems of up to $60,000$ variables. In this paper, we will report numerical results for some medium-scale SVM problems, that are better than GVPM proposed in [21] and SPGM proposed in [2]. Further discussion of all these issues is given in Section 4.

2 An algorithm for the diagonal and strictly convex case

In this section we develop an algorithm for the special case of (1.1) in which $A = \mathrm{diag}(d_1, d_2, \ldots, d_n)$ is a positive definite diagonal matrix. The positivity of the d_i is important for the development of the algorithm, and we shall show at the end of the section that there are difficulties to be surmounted if the positive definite condition is to be relaxed. We shall refer to the resulting algorithm as `Algorithm 1`. The algorithm has some features in common with previous work, notably that it runs in expected linear time, but also includes some new features. The description of the algorithm aims to provide a more simple exposition of the underlying principles than appears elsewhere.

The algorithm is based on constructing a Lagrangian penalty function

$$\phi(\mathbf{x};\, \lambda) = \tfrac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{c}^T\mathbf{x} - \lambda(\mathbf{a}^T\mathbf{x} - b), \tag{2.1}$$

in which λ is a scalar parameter. Because A is positive definite, no augmentation term is needed. For any fixed value of λ, we may solve the box constrained QP problem

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbf{R}^n}{\text{minimize}} & \phi(\mathbf{x}) \\ \text{subject to} & \mathbf{l} \le \mathbf{x} \le \mathbf{u} \end{array} \tag{2.2}$$

giving a minimizer which we denote by $\mathbf{x}(\lambda)$. Then λ is adjusted in an outer secant-like method to solve the single nonlinear equation

$$r(\lambda) := \mathbf{a}^T\mathbf{x}(\lambda) - b = 0 \tag{2.3}$$

in one variable λ. The minimizer of $\phi(\mathbf{x})$ is readily obtained because (2.2) separates into n problems, each in one variable x_i. The unconstrained minimizer of each problem is the solution of the equation $d_i x_i = c_i + \lambda a_i$, so the solution of (2.2) is given by

$$\mathbf{x}(\lambda) = \text{mid}(\mathbf{l}, \mathbf{h}, \mathbf{u}), \tag{2.4}$$

where $\mathbf{h} = \mathbf{h}(\lambda)$ has components $h_i = (c_i + \lambda a_i)/d_i$, and mid$(\mathbf{l}, \mathbf{h}, \mathbf{u})$ is the componentwise operation that supplies the median of its three arguments.

If a value λ^* is located such that $\mathbf{a}^T\mathbf{x}(\lambda^*) = b$, it follows that KT conditions for (1.1) are satisfied. This is because the KT conditions for (2.2) are also necessary for (1.1), and together with the feasibility condition $\mathbf{a}^T\mathbf{x} = b$, they make up the KT conditions for (1.1). It follows when (2.3) is solved that $\mathbf{x}(\lambda^*)$ is a KT point for (1.1) and hence a global minimizer by convexity.

We now consider how the equation $r(\lambda) = 0$ might be solved. It is possible that the constraints of (1.1) are inconsistent, in which case $r(\lambda) = 0$ has no solution, a possibility which we ignore at present and return to towards the end of this section. We note that $a_i x_i$ is a piecewise linear continuous function of λ by virtue of (2.4), which takes either the constant values $a_i l_i$ or $a_i u_i$, or the value $a_i(c_i + \lambda a_i)/d_i$, and hence is either constant or a monotonically increasing function of λ. It follows that the same is true of $r(\lambda)$. We note that $r(\lambda)$ might be constant over part of its domain. In fact we may interpret $r(\lambda)$ as the gradient of a dual function of λ after eliminating the primal variables.

Although it would be a simple matter to evaluate the slope of the function $r(\lambda)$ (except at a breakpoint), to be used in an iteration of the Newton-Raphson method, the piecewise linear nature of $r(\lambda)$ makes this unappealing. For example the correction would be infinite on a constant piece of the graph. Yet we would wish to use linear approximation when the linear piece that crosses the λ axis is located, so as to locate λ^* exactly. Thus we have chosen a secant-type approach, with modifications to promote rapid global convergence. Our algorithm divides into two parts, a *Bracketing phase* in which a bracket on a solution is sought, followed by a *Secant phase* in which the bracket is uniformly reduced until the piece which crosses the λ axis is located, and the process terminates.

In the bracketing phase we ask the user to supply an initial value of λ and an initial estimate $\Delta\lambda > 0$ of the likely change to λ. The bracketing phase may be described by the following matlab statements, in which `a, b, c, d, l, u` refer to the data that defines (1.1). If a value of $r(\lambda)$ is located which is sufficiently close to zero, then the process terminates. If $r(\lambda) < 0$ then the search for a bracket takes place in the positive λ direction, else if $r(\lambda) > 0$ then in the negative λ direction. If the problem (1.1) is feasible, then the bracketing phase is guaranteed to terminate with a bracket $[\lambda_l,\ \lambda_u]$ which contains a solution of the equation $r(\lambda) = 0$. The calculation of `s` in the above is such that `dlambda/s` is the correction to λ that gives either a secant step based on the two most recent iterates, or a step of 10 times the previous step, whichever is the smaller. In practice, examples were found in which `dlambda/s` could be much smaller than `dlambda`, giving rise to slow convergence. Hence the updated value `dlambda=dlambda+dlambda/s` has been used, so as to ensure that the new value of `dlambda` is greater than the old value. Perhaps a more appealing way of achieving this would have been to define `s=min(1,max(rl/r-1,0.1))`

```
% Bracketing Phase of Algorithm 1
x=max(l,min(u,(c+lambda*a)./d)); r=a'*x-b;
if r < 0
  lambda_l=lambda; rl=r; lambda=lambda+dlambda;
  x=max(l,min(u,(c+lambda*a)./d)); r=a'*x-b;
  while r < 0
    lambda_l=lambda; s=max(rl/r-1,0.1);
    dlambda=dlambda+dlambda/s; lambda=lambda+dlambda;
    x=max(l,min(u,(c+lambda*a)./d)); r=a'*x-b;
  end
  lambda_u=lambda; ru=r;
else
  lambda_u=lambda; ru=r; lambda=lambda-dlambda;
  x=max(l,min(u,(c+lambda*a)./d)); r=a'*x-b;
  while r > 0
    lambda_u=lambda; s=max(ru/r-1,0.1);
    dlambda=dlambda+dlambda/s; lambda=lambda-dlambda;
    x=max(l,min(u,(c+lambda*a)./d)); r=a'*x-b;
  end
  lambda_l=lambda; rl=r;
end
```

and to use `dlambda=dlambda/s`. This would enable the secant step to be accepted in some cases, and hence give termination if the current piece intersects the λ axis. However, in practice the difference between these strategies is likely to be small.

Once a bracket is located, the second stage is to find the solution of the equation $r(\lambda) = 0$ by repeated use of the secant method, with certain modifications designed to speed up convergence remote from the solution. Again the process is defined by some matlab code. Initially values of λ_l and λ_u are available with $r(\lambda_l) < 0$ and $r(\lambda_u) > 0$, and a secant step to a new point λ is taken. If $r(\lambda) > 0$ then the iteration proceeds as follows. If λ lies in the left half of the interval $[\lambda_l, \lambda_u]$ (that is `s<=2`), then a secant step based on λ_l and λ is taken on the next iteration. If λ lies in the right half of the interval, then either a secant step based on λ and λ_u, or a step to the point $\frac{3}{4}\lambda_l + \frac{1}{4}\lambda$ is taken, whichever is the smaller step. This ensures that the interval length is reduced by a factor of $\frac{3}{4}$ or less. In both cases λ_u is replaced by λ to give a new bracket. Similar decisions are taken if $r(\lambda) < 0$ at the start of the iteration. We terminate the secant phase if preset tolerances on either $r(\lambda)$ or $\Delta\lambda$ are met. Of course the parameters in the above pieces of code are somewhat arbitrary, but are of an appropriate size. We feel that small changes in these values would be unlikely to change the effectiveness of the method to any great extent.

We now return to the issue of how to decide if the constraints of (1.1) are consistent. If a bracket is found during the bracketing phase, then it follows that the constraints are consistent, and no further calculation is required. Alternatively, we may define the

```
% Secant phase of Algorithm 1
s=1-rl/ru; dlambda=dlambda/s; lambda=lambda_u-dlambda;
x=max(l,min(u,(c+lambda*a)./d)); r=a'*x-b;
while 1     % repeat until converged
  if r > 0
    if s <= 2
      lambda_u=lambda; ru=r; s=1-rl/ru;
      dlambda=(lambda_u-lambda_l)/s; lambda=lambda_u-dlambda;
    else
      s=max(ru/r-1,0.1); dlambda=(lambda_u-lambda)/s;
      lambda_new=max(lambda-dlambda,0.75*lambda_l+0.25*lambda);
      lambda_u=lambda; ru=r; lambda=lambda_new;
      s=(lambda_u-lambda_l)/(lambda_u-lambda);
    end
  else
    if s >= 2
      lambda_l=lambda; rl=r; s=1-rl/ru;
      dlambda=(lambda_u-lambda_l)/s; lambda=lambda_u-dlambda;
    else
      s=max(rl/r-1,0.1); dlambda=(lambda-lambda_l)/s;
      lambda_new=min(lambda+dlambda,0.75*lambda_u+0.25*lambda);
      lambda_l=lambda; rl=r; lambda=lambda_new;
      s=(lambda_u-lambda_l)/(lambda_u-lambda);
    end
  end
  x=max(l,min(u,(c+lambda*a)./d)); r=a'*x-b;
end
```

vectors

$$\mathbf{a}_+ = \max(\mathbf{a}, \mathbf{0}), \qquad \mathbf{a}_- = \min(\mathbf{a}, \mathbf{0}) \tag{2.5}$$

to be the positive and negative parts of $\mathbf{a}$, respectively. Since the linear function has a maximum value of $\mathbf{a}_+^T\mathbf{u}+\mathbf{a}_-^T\mathbf{l}$ and a minimum value of $\mathbf{a}_+^T\mathbf{l}+\mathbf{a}_-^T\mathbf{u}$ on the set $\{\mathbf{x} \mid \mathbf{l} \le \mathbf{x} \le \mathbf{u}\}$, we can conclude inconsistency if either of the conditions

$$\mathbf{a}_+^T\mathbf{u} + \mathbf{a}_-^T\mathbf{l} < b, \qquad \mathbf{a}_+^T\mathbf{l} + \mathbf{a}_-^T\mathbf{u} > b \tag{2.6}$$

hold, and consistency otherwise. However, this requires a certain amount of calculation, which might be avoided if the problem is known to be consistent, or if a bracket is found quickly. In our codes we have resolved this matter in the following way. The user is asked to supply a parameter `ktest`. If a bracket is not found in `ktest` iterations then the calculation in (2.6) is carried out to decide on consistency. If the user knows that the problem is consistent, then `ktest`$= \infty$ can be set. Otherwise we suggest using the value `ktest=4`.

Finally we examine whether it is possible to relax the condition that the A is positive definite. Consider the case that $d_i = 0$ occurs for some value of i. The minimization of

$\phi(\mathbf{x}, \lambda)$ still separates into n one-variable problems, and hence is readily solved. In the case of x_i, we simply set

$$x_i = \begin{cases} l_i & \text{if } c_i + \lambda a_i < 0 \\ u_i & \text{if } c_i + \lambda a_i > 0 \end{cases} .$$

The difficulty is that x_i, and hence $\mathbf{x}(\lambda)$, is no longer continuous at a value of λ at which the solution x_i switches from one bound to the opposite bound. The difficulties that arise are illustrated by a simple example. Take $n = 2$, $d_1 = 1$, $d_2 = 0$, $\mathbf{c} = (1, 1)^T$, $\mathbf{a} = (2, 1)^T$, $b = 1$, $\mathbf{l} = \mathbf{0}$ and $\mathbf{u} = (2, 2)^T$. The solution is at $\mathbf{x}^* = (0, 1)^T$ with multiplier $\lambda^* = -1$. The problem is convex and $\mathbf{x}^*$ is the unique global solution, so the problem is in no sense badly behaved. However, consider the graph of the function $r(\lambda)$. For $\lambda < -1$ the minimizer of $\phi(\mathbf{x}, \lambda)$ is $\mathbf{x} = (0, 2)^T$, whereas for $-1 < \lambda \leq \frac{1}{2}$ it is $\mathbf{x} = (0, 0)^T$. Thus $r(\lambda)$ is piecewise constant in the neighbourhood of $\lambda < -1$, and jumps from -1 to $+1$ as λ passes through -1. The solution is determined by choosing $\mathbf{x}$ to solve $\mathbf{a}^T\mathbf{x} = b$ at the point of discontinuity in λ. The discontinuity in $r(\lambda)$ might adversely affect the behaviour of the secant algorithm described above. In particular we cannot expect the algorithm to terminate by locating a linear piece that crosses the λ axis. In the case that A is diagonal and positive semi-definite, one way to circumvent these difficulties is to calculate the values of λ such that $c_i + \lambda a_i = 0$ for $i \in \{i : d_i = 0\}$. The corresponding values of $r(\lambda)$ can then be used to determine an interval that includes the optimal multiplier. If $d_i \neq 0$ for all i but $d_i < 0$ for some values of i, then the problem may have many KT points. In this case, we can still define the one-variable function $r(\lambda)$ as before. However, this function is not in general monotonic, but is still continuous. Methods based on its continuity may be developed to calculate one KT point of the problem.

3 A projected gradient algorithm for the general case

In this section we consider an algorithm for solving (1.1) when A is not a diagonal matrix. The algorithm has the same framework as that of the SPG2 algorithm of Birgin, Martínez and Raydan [2]. However a new choice for the steplength is used here, and the adaptive non-monotone line search from [6] is incorporated. Our numerical experiments in the next section show that these new techniques are very useful in practice. An overview of the algorithm, referred to as Algorithm 2, is

```
% Overview of Algorithm 2
Initialization
for k=1,2,...until converged
   Calculate the projection step using Algorithm 1,
   Possibly carry out a line search,
   Calculate the BB step length,
   Update the line search control parameters
end
```

Let us denote the feasible region of (1.1) by Ω. The projection of any vector $\mathbf{z} \in \mathbb{R}^n$ on to Ω is the minimizer of the problem

$$\min\{\tfrac{1}{2}\|\mathbf{x}-\mathbf{z}\|_2^2 : \ \mathbf{x} \in \Omega\}. \tag{3.1}$$

This is a special case of an SLBQP problem in which $A = I$ is the identity matrix, and hence it can be solved efficiently by for example Algorithm 1. On average, we find that only about five iterations are required by Algorithm 1. The first stage in the loop of Algorithm 2 is to take a steepest descent step from a current iterate $\mathbf{x}_k$ with fixed steplength α_k, and then project the resulting point on to Ω. If $\mathbf{g} = A\mathbf{x} - \mathbf{c}$ denotes the gradient vector, we may express this as

$$\mathbf{d}_k = P_\Omega(\mathbf{x}_k - \alpha_k \mathbf{g}_k) - \mathbf{x}_k \tag{3.2}$$

where $P_\Omega(\mathbf{z})$ represents the projection of $\mathbf{z}$ on to Ω. We note that the projection operation has the property that $(\mathbf{x} - P_\Omega(\mathbf{z}))^T(\mathbf{z} - P_\Omega(\mathbf{z})) \leq 0$ for all $\mathbf{x} \in \Omega$, and it readily follows if $\mathbf{d}_k \neq \mathbf{0}$ that $\mathbf{d}_k$ is a feasible descent direction. The solution of (1.1) is characterised by $P_\Omega(\mathbf{x}^* - \mathbf{g}^*) = \mathbf{x}^*$, so the algorithm is deemed to have converged when $\|P_\Omega(\mathbf{x}_k - \mathbf{g}_k) - \mathbf{x}_k\|$ is within a prescribed tolerance. Although this is how convergence was recognised in our codes, it does require an extra projection calculation, and a less expensive alternative is to test $\|P_\Omega(\mathbf{x}_k - \alpha_k \mathbf{g}_k) - \mathbf{x}_k\|/\alpha_k$ (that is $\|\mathbf{d}_k\|/\alpha_k$).

The second stage of Algorithm 2 is to decide if a line search is necessary. A feature of BB type algorithms is that they are inherently non-monotonic, that is to say the objective function value $f(\mathbf{x}_k)$ must be allowed to increase on some iterations in order for the methods to work well (see Fletcher [8]). When minimizing quadratic functions without any constraints, convergence can be established without introducing a line search (see Raydan [19]). However we show in [6] for box constrained QP that it is possible (albeit unlikely) for the algorithm to fail, unless some provision for a line search is made. The same will certainly be true for the SLBQP problem. However it is important that a non-monotonic line search is used, a fact first recognised by Raydan [20] for general unconstrained minimization. In [20] and [2], a non-monotonic line search of the type suggested by Grippo, Lampariello and Lucidi [10] is used. In [6] we show that an alternative adaptive non-monotonic line search is preferable, as it cuts down the number of times the line search is brought into play, and hence enables the non-monotonic aspects of the BB method to operate more freely. All these non-monotonic algorithms make use of a control parameter $f_{ref} \geq f(\mathbf{x}_k)$, and no line search is carried out if $f(\mathbf{x}_k + \mathbf{d}_k) < f_{ref}$. (In some methods a sufficient reduction on f_{ref} is required.) However, the way in which f_{ref} is defined is different.

For $k \geq 2$ we carry out a line search if $f(\mathbf{x}_k + \mathbf{d}_k) \geq f_{ref}$. We do this simply by a quadratic interpolation along $\mathbf{x}_k + \lambda \mathbf{d}_k$, using the function values $f(\mathbf{x}_k)$ and $f(\mathbf{x}_k + \mathbf{d}_k)$, and the slope $\mathbf{g}_k^T \mathbf{d}_k$. An alternative possibility would be to search along the path obtained by projecting $\mathbf{x}_k - \lambda \mathbf{g}_k$ (as in SPG1 of [2]). However this requires extra projection operations, and in any event the distinction is of little importance because we use very few line searches. Because $f(\mathbf{x})$ is quadratic, $f(\mathbf{x}_k) \leq f_{ref}$ and $f(\mathbf{x}_k + \mathbf{d}_k) \geq f_{ref}$, and $\mathbf{d}_k$ is a descent direction, it follows that we obtain an exact line search along $\mathbf{x}_k + \lambda \mathbf{d}_k$. This is different to methods based on [10] which use an Armijo line search. On iteration

$k = 1$, we carry out a line search whenever $f(\mathbf{x}_k + \mathbf{d}_k) \geq f(\mathbf{x}_k)$, because the initial value of α_1 might possibly be unreliable.

The third stage of the algorithm is to compute the stepsize α_{k+1} for the next iteration. Denote the difference vectors $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$. One of the two BB formulae in [1] is defined by

$$\alpha_{k+1} = \mathbf{s}_k^T \mathbf{s}_k / \mathbf{s}_k^T \mathbf{y}_k. \tag{3.3}$$

There are many references to the fact that this choice is far more efficient than the classical steepest descent stepsize, and in Section 4 we give some results for this choice in the context of an SLBQP problem. Recently many other BB-like formulae for the stepsize have been proposed. We have therefore tried alternating the two BB formulae on successive iterations, as recommended in [6], but find this to be ineffective for random indefinite test problems and SVM test problems. Other alternative stepsize choices, such as the AS and CSDS methods described in [5], did not perform well in the box constrained QP context [6], and we have not used them here. Here we also present a new choice for the stepsize. To this end, we recall that the BB formula (3.3) can be obtained by solving the one-dimensional subproblem: minimize $\|\alpha_{k+1}^{-1}\mathbf{s}_k - \mathbf{y}_k\|_2$, in which $\alpha_{k+1}^{-1} I$ can be regarded as some approximation to the Hessian at $\mathbf{x}_{k+1}$. For each integer $i \geq 0$, we can have a similar formula if we replace the pair $(\mathbf{s}_k, \mathbf{y}_k)$ with $(\mathbf{s}_{k-i}, \mathbf{y}_{k-i})$, yielding $\alpha_{k+1} = \mathbf{s}_{k-i}^T \mathbf{s}_{k-i} / \mathbf{s}_{k-i}^T \mathbf{y}_{k-i}$ (this formula can be seen in [9]). It might be interesting to study how to pick the *best* integer i according to some rule. Here we propose a new formula based on the idea of taking an average of the most recent $m \geq 1$ difference pairs, where m is a preset integer. Hence we define the vectors $S^{(k)} = (\mathbf{s}_k^T, \ldots, \mathbf{s}_{k-m+1}^T)^T$ and $Y^{(k)} = (\mathbf{y}_k^T, \ldots, \mathbf{y}_{k-m+1}^T)^T$, and determine α_{k+1} by minimizing $\min \|\alpha_{k+1}^{-1} S^{(k)} - Y^{(k)}\|_2$. Therefore we obtain

$$\alpha_{k+1} = \frac{S^{(k)T} S^{(k)}}{S^{(k)T} Y^{(k)}} = \frac{\sum_{i=0}^{m-1} \mathbf{s}_{k-i}^T \mathbf{s}_{k-i}}{\sum_{i=0}^{m-1} \mathbf{s}_{k-i}^T \mathbf{y}_{k-i}}. \tag{3.4}$$

In fact, we let $\bar{m}$ be the maximal integer such that $\mathbf{s}_{k-i}^T \mathbf{y}_{k-i} > 0$ for all $0 \leq i \leq \bar{m}$, and we assume conventionally that $\mathbf{s}_k^T \mathbf{y}_k \leq 0$ if $k \leq 0$. Our practical algorithm then calculates the stepsize by (3.4) with m replaced by $\min(m, \bar{m})$. We also chop any extreme values of α_k by truncating them to lie in preassigned interval $[\alpha_{min}, \alpha_{max}]$. If it happens that $\bar{m} = 0$, that is $\mathbf{s}_k^T \mathbf{y}_k \leq 0$, we simply set $\alpha_{k+1} = \alpha_{max}$. It is obvious that the formula (3.4) with $m = 1$ reduces to the BB formula (3.3). However, our numerical experiments suggest that $m = 2$ is a better choice in the BQP or SLBQP context. Results obtained using this choice of stepsize are described in Section 4.

The fourth stage of Algorithm 2 is to update the line search parameters. The line search we use is that described in [6], which is related to methods described by Toint [22] and by Dai and Zhang [7]. The parameters involved are the reference value f_{ref} referred to above, the current best function value f_{best}, a candidate value f_c for possible reduction of f_{ref}, and a preassigned number L denoting the number of iterations with $f(\mathbf{x}_k) \geq f_{best}$ that are allowed before reducing f_{ref} to f_c. The updating process may be described by the matlab code

```
if f_k < f_best,
   f_best=f_k, f_c=f_k, l=0,
else
   f_c=max{f_c,f_k}, l=l+1,
   if l=L, f_ref=f_c, f_c=f_k, l=0, end
end.
```

A simple argument in [6] proves global convergence in real arithmetic. An important property of this method is that if a better value of $f(\mathbf{x}_k)$ is always found in at most L iterations, then f_{ref} remains at its initial value of infinity, and no line searches are needed after the first iteration.

Algorithm 2 requires parameter settings for L, α_{min} and α_{max}, and initial values for $\mathbf{x}_1$, $f_{ref} = \infty$, $\mathtt{l} = 0$ and $f_{best} = f_c = f(\mathbf{x}_1)$. The major computational expense is that required to update the gradient vector, which can be done in $O(np)$ operations, where p is the number of non-zero components of $\mathbf{d}_k$. The number of projection calculations can also be significant, since although each iteration of Algorithm 1 only requires $O(n)$ operations, quite a large number of projection calculations may be required.

4 Numerical experiments and discussion

We tested Algorithm 2 with MATLAB 6.5.0 on a Dell^{TM} OptiPlex^{TM} computer. The preset parameters for Algorithm 2 were chosen to be $\alpha_{\min} = 10^{-5}$, $\alpha_{\max} = 10^5$ and $L = 10$. For Algorithm 1, which is used as the projection subroutine of Algorithm 2, the initial values for λ and $\Delta\lambda$ on the first call were set to 0 and 2, respectively. They were then set to λ' and $1 + |\lambda'|$ on the second call and λ' and $1 + |\lambda' - \lambda''|$ subsequently. Here λ' and λ'' are the values of λ provided by the previous two projections. The accuracy required in the projection algorithm is either $|r(\lambda)| \leq 10^{-8}$ or $\Delta\lambda \leq 10^{-8}$. The choice of the initial stepsize α_1 is described later. The parameter m in the formula (3.4) is chosen to be $m = 2$.

Three kinds of problem were used in our tests: (1) random symmetric positive definite (SPD) test problems, (2) random indefinite test problems, and (3) medium-scale problems generated by training Gaussian SVMs on two real-world data sets. The descriptions of the test problems and the corresponding numerical results are presented as follows. We let $\mathbf{e}$ denote the column vector of ones, and the vectors $\mathbf{p}_i$ $i = 1, 2, \ldots, 7$ denote random vectors whose elements are sampled from a uniform distribution in [0, 1]. Also we denote $v_{i,j}$ to be the j-th component of the vector $\mathbf{v}_i$.

Random SPD test problems. The generation of these problems is based on the generation of random SPD BQP test problems in Moré and Toraldo [15] and Dai and Fletcher [6]. At first, we use generate a BQP problem using the five parameters n, $ncond$, $ndeg$, $na(\bar{\mathbf{x}})$ and $na(\bar{\mathbf{x}}_1)$ such that $\bar{\mathbf{x}}$ is the solution of the BQP problem and $\bar{\mathbf{x}}_1$ is the starting point. Specifically, we denote $A = PDP^T$, where

$$P = (I - 2\mathbf{p}_3\mathbf{p}_3^T)(I - 2\mathbf{p}_2\mathbf{p}_2^T)(I - 2\mathbf{p}_1\mathbf{p}_1^T),$$

Table 4.1. Numerical results for random SPD test problems

P	$ncond$	$ndeg$	$na(\bar{\mathbf{x}})$	$na(\bar{\mathbf{x}}_1)$	$\#iter$	k_{aver}	k_{max}	BB
1	4	1	6788	6792	75	4.09	6	66
2	5	1	1345	8909	123	3.13	4	129
3	4	7	782	6117	79	3.00	4	85
4	6	7	2989	6526	238	4.05	5	287
5	4	9	1786	678	78	3.71	4	73
6	4	5	7298	3911	68	4.73	6	76
7	7	3	2868	8823	410	3.41	6	438
8	7	5	17	7547	399	3.36	9	390
9	7	5	609	8891	429	3.33	10	536
10	5	1	5697	8831	126	2.76	5	154
11	4	5	1873	8362	71	3.35	4	80
12	6	3	4218	6529	263	4.73	6	248
13	7	7	8542	6840	459	6.50	10	540
14	5	1	448	156	115	3.34	4	130
15	4	5	1851	2689	69	3.25	4	76
16	7	5	6287	1378	713	5.85	7	*
17	7	9	9670	6729	513	7.37	12	511
18	7	7	727	5997	426	3.62	6	468

where D is a diagonal matrix with whose i-th component is defined by

$$\log d_i = \frac{i-1}{n-1} ncond, \quad i = 1, \ldots, n.$$

The parameter $ncond$ in the above relation specifies the condition number of A. We set $\bar{\mathbf{x}} = 2\mathbf{p}_4 - 1$, namely, $\bar{\mathbf{x}}$ is chosen randomly in the interval $[-1, 1]$. The choice of active set $\mathcal{A}(\bar{\mathbf{x}})$ depends on the integer parameter $na(\bar{x})$. Random numbers μ_i in $[0, 1]$ are generated for $i = 1, \ldots, n$ and i is selected for $\mathcal{A}(\bar{x})$ if $\mu_i \leq na(\bar{\mathbf{x}})/n$. This algorithm is also used to select the active constraints $\mathcal{A}(\bar{\mathbf{x}}_1)$ at the starting point of the BQP problem on the basis of the parameter $na(\bar{\mathbf{x}}_1)$. Components of $\bar{\mathbf{x}}_1$ that are not in $\mathcal{A}(\mathbf{x}_1)$ are set to $(l_i + u_i)/2$.

The values of $\mathbf{l}$, $\mathbf{u}$, and $\mathbf{c}$ can now be determined using the parameter $ndeg$. This parameter specifies the extent to which the resulting problem will be close to being dual degenerate. We determine the value of $\nabla f(\bar{\mathbf{x}}) = \mathbf{r}$ by setting $|r_i| = 10^{-\mu_i ndeg}$ for $i \in \mathcal{A}(\bar{\mathbf{x}})$, where μ_i is randomly generated in $[0, 1]$. The right hand term $\mathbf{c}$ is set to $\mathbf{c} = A\bar{\mathbf{x}} - \mathbf{r}$ and we define

$$l_i = -1, \quad u_i = +1, \quad r_i = 0,$$

for $i \notin \mathcal{A}(\bar{x})$, and

$$l_i = \bar{x}_i, \quad u_i = +1, \quad r_i > 0,$$

or

$$l_i = -1, \quad u_i = \bar{x}_i, \quad r_i < 0,$$

for $i \in \mathcal{A}(\bar{x})$. Then we have constructed an SPD BQP problem whose solution is $\bar{\mathbf{x}}$. Further, we let $\mathbf{x}_{fs}$ be the vector with components $l_i + p_{5,i}(u_i - l_i)$, $\mathbf{a} = 2\mathbf{p}_6 - \mathbf{e}$ and

$b = \mathbf{a}^T \mathbf{x}_{fs}$. The starting point for the SLBQP problem is set to $\mathbf{x}_1 = P_\Omega(\bar{\mathbf{x}}_1)$. Hence we have described how to generate our random SPD SLBQP test problems by five parameters n, $ncond$, $ndeg$, $na(\bar{\mathbf{x}})$ and $na(\bar{\mathbf{x}}_1)$.

In our random SPD tests, we generated 18 problems in which the dimension is fixed at $n = 10^4$ and the other parameters are randomly generated. For these problems, the first stepsize used is $\alpha_1 = \|P_\Omega(\mathbf{x}_1 - \mathbf{g}_1) - \mathbf{x}_1\|_\infty^{-1}$ and the stopping condition is

$$\|P_\Omega(\mathbf{x}_k - \mathbf{g}_k) - \mathbf{x}_k\|_\infty \leq 10^{-5},$$

as used in [2]. The numerical results are shown in Table 4.1, where P refers to the problem number, $\#iter$ the number of required iterations and $\#ls$ the number of line search. Also, k_{aver} and $k_{\max}$ denote the average number of iterations and the maximum number of iterations, respectively, taken by Algorithm 1. In these tests, no line searches are carried out after the first iteration and hence the number of line searches is at most one. That is to say, in this case Algorithm 2 reduces to the unmodified algorithm. The number of iterations required by the BB formula (3.3) is also listed in Table 4.1. For Problem 16, the BB method fails to provide a solution in 2000 iterations. From this table, we can see that the formula (3.4) with $m = 2$ is significantly better overall than (3.3) for random SPD problems. Discounting problems in which the number of iterations required differs by less than 10, then formula (3.4) wins in 8 cases, as against 1 for (3.3). Further tests with 100 random SPD problems showed a ratio of $42 : 6$ wins in favour of (3.4).

Random indefinite test problems. The generation of these problems is based on four parameters n, $ncond$, $na(\bar{\mathbf{x}}_1)$ and $negeig$. The first three parameters are used to generate the matrix A, the vector $\mathbf{c}$, and the starting point $\bar{\mathbf{x}}_1$ as before. The parameter $negeig$ is used to specify the number of negative eigenvalues of A. Given an integer $negeig \in [1, n]$, we change the sign of the i-th diagonal entry of D if $p_{7,i} \leq negeig/n$. The lower and upper bounds are set to $\mathbf{l} = -\mathbf{e}$ and $\mathbf{u} = \mathbf{e}$. The vector $\mathbf{a}$ and the scalar b are generated as before.

We likewise generated 18 indefinite SLBQP test problems with $n = 10^4$. The choice of the first stepsize and the stopping condition are the same as before. The numerical results are shown in Table 4.2. Again, we found that no line searches are done after the first iteration. From Table 4.2, we see that the formula (3.4) is only slightly better than the BB formula (3.3) for random indefinite problems. Again, discounting problems in which the number of iterations required differs by less than 10, the ratio of wins is only $8 : 7$ in favour of (3.4). Further tests with 100 random indefinite problems showed a ratio in favour of (3.4) as $46 : 40$.

SVM test problems [21]. Given a training set of labelled examples

$$D = \{(\mathbf{z}_i, w_i), i = 1, \ldots, n, \quad \mathbf{z}_i \in \mathbb{R}^m, w_i \in \{-1, 1\}\},$$

the SVM algorithm classifies new examples $\mathbf{z} \in \mathbb{R}^m$ by a decision function $F : \mathbb{R}^m \to \{-1, 1\}$ of the form

$$F(\mathbf{z}) = \text{sign}\left(\sum_{i=1}^{n} x_i^* \mathbf{w}_i K(\mathbf{z}, \mathbf{z}_i) + b^*\right),$$

Table 4.2. Numerical results for random indefinite test problems

P	$ncond$	$negeig$	$na(\bar{\mathbf{x}}_1)$	$\#iter$	k_{aver}	k_{max}	BB
1	4	6788	470	136	3.42	19	149
2	6	3291	1786	249	3.85	26	417
3	4	199	5189	76	3.16	4	73
4	4	701	9535	117	3.47	18	133
5	4	3653	2332	194	3.89	26	116
6	7	4403	2525	480	6.26	22	637
7	4	1199	1379	95	2.82	14	124
8	4	2749	4854	228	3.17	23	125
9	6	30	4667	247	3.26	4	243
10	7	6067	4118	458	4.08	28	559
11	4	4942	9427	96	3.87	22	113
12	5	7736	8389	211	5.23	28	216
13	4	1672	1930	118	3.11	16	100
14	6	2015	2250	414	3.29	19	280
15	4	5871	930	113	3.56	21	138
16	7	9387	9552	375	6.45	44	329
17	6	179	3818	315	3.26	17	289
18	7	9712	715	387	7.33	37	306

where $K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is some kernel function and $\mathbf{x}^* = (x_i^*)$ solves

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\mathbf{x}^T G\mathbf{x} - \mathbf{e}^T\mathbf{x} \\ \text{subject to} & \mathbf{0} \le \mathbf{x} \le C\mathbf{e} \\ & \mathbf{w}^T\mathbf{x} = 0, \end{array} \tag{4.1}$$

where G has entries $G_{ij} = w_i w_j K(\mathbf{z}_i, \mathbf{z}_j)$, $i, j = 1, 2, \ldots, n$, and C is a parameter of the SVM algorithm. The quantity $b^* \in \mathbb{R}$ is easily derived after $\mathbf{x}^*$ is computed. Our problems are generated by training SVMs on two real-world data sets: the MNIST database of handwritten digits and the UCI Adult data set. Test problems of size $n = 800$, 1600 and 3200 are constructed from MNIST by considering the first $n/2$ inputs of digit "8"and the first $n/2$ inputs of the other digits. From the UCI Adult, the versions with $n = 1605$, 2265 and 3185 are considered. The Gaussian kernel

$$K(\mathbf{z}_i, \mathbf{z}_j) = \exp\left(-\|\mathbf{z}_i - \mathbf{z}_j\|_2^2/(2\sigma^2)\right) \tag{4.2}$$

is used in our tests, which ensures the symmetric positive definiteness of the matrix G. The parameters C in (4.2) and σ in (4.2) are set to $(C, \sigma) = (10, 2000)$ for the MNIST database and $(1, \sqrt{10})$ for the UCI Adult data sets.

For these SVM test problems, a different stopping condition is used, which is based on the fulfilment of the KKT conditions within 10^{-3} (see [13]). The initial point is $\mathbf{x}_1 = \mathbf{0}$. The first stepsize is simply set to $\alpha_1 = \|\mathbf{g}_1\|_\infty$ since the projection $P_\Omega(\mathbf{x}_k - \mathbf{g}_k)$ is not used by the stopping condition. The numerical results are shown in Table 4.3, where SV and

Table 4.3. Numerical results for SVM test problems

Problem	n	#iter	#ls	SV	BSV	k_{aver}	k_{max}	BB
MNIST	800	128	8	281	1	4.18	12	152
	1600	198	20	457	9	4.87	13	273
	3200	508	82	807	25	5.28	16	558
UCI Adult	1605	106	4	691	584	4.49	7	108
	2265	141	9	1011	847	4.22	11	130
	3185	186	17	1303	1108	4.43	11	167

BSV stand for the number of support vectors and bound support vectors, respectively (a training example $\mathbf{z}_i$ is called a support vector if the corresponding x_i^* is nonzero and a bound support vector if $x_i^* = C$). The iteration numbers required by the BB formula (3.3) is also listed in Table 4.3. These numerical experiments again demonstrate the usefulness of the formula (3.4).

From Tables 4.1-4.3, we see that both Algorithm 2 and the projection subalgorithm Algorithm 1 are very efficient. To achieve a very accurate solution, Algorithm 1 only requires around 4 or 5 iterations on average and in the worst case 44 iterations, although often a lot less. We have observed the worst case to occur when the linear piece crossing the λ axis is very short, and so is difficult to determine quickly. The average performance of Algorithm 1 is very similar over all the tables, suggesting that in most cases the heuristics for globalizing the secant iteration are working well. All these experiments suggest that the formula (3.4) is a promising alternative to the BB formula (3.3).

In order to test the efficacy of our adaptive line search procedure, we also tested Algorithm 2 with the nonmonotone line search in [10]. This line search determines the reference function value f_{ref} at the k-th iteration by

$$f_{ref} = \max\{f(\mathbf{x}_{k-i}) : i = 0, \ldots, \max\{k-1, M-1\}\}, \tag{4.3}$$

where M is some preset positive integer. Again, a quadratic interpolation will be done if the initial stepsize fails to meet the line search condition. In these tests we have chosen the commonly accepted value of $M = 10$. We found that this nonmonotone line search is much worse than the adaptive one for the random SPD and indefinite test problems (the results for these problems are not listed in this paper). However, it performs better than the latter for the SVM test problems. See the column of Alg. 2* in Table 4.4 for the numerical results corresponding to Alg. 2 with the choice (4.3) (inside the brackets are the number of line searches).

To see whether the line search or the stepsize formula is the more significant influence on numerical performance for the SVM test problems, we also tested the BB formula (3.3) with the choice (4.3) (see the column BB* in Table 4.4 for the iteration number). In Table 4.4 are also listed the iteration numbers required by Algorithm 2, the BB formula with the adaptive line search, and the SPGM and GVPM algorithms that can be found in [21] (here we note that the SPGM method is first proposed in [2]) . From the table, we can see that the results of (3.4) are better than those obtained by the BB formula no matter what line search is used. Thus this again confirms the usefulness of the formula

Table 4.4. Numerical comparisons for SVM test problems

Problem	n	SPGM	GVPM	Alg. 2	Alg. 2*	BB	BB*
MNIST	800	163	161	128	141(16)	152	161(22)
	1600	303	277	198	218 (30)	273	287(44)
	3200	691	513	508	428 (85)	558	558(107)
UCI Adult	1605	90	153	106	96 (10)	108	90(8)
	2265	135	196	141	142 (17)	130	150(23)
	3185	175	282	186	155 (23)	167	143(19)

(3.4). We also see from Table 4.4 that Alg. 2* presents the best results for the SVM test problems among the six gradient algorithms. Finally, we may have a comparison for SPGM and BB*. We know that both the algorithms define the reference function values by (4.3) with $M = 10$. The difference is in that, if the initial stepsize fails to meet the line search conditions, the former decides the second initial stepsize by some Armijo rule but the latter takes the optimal stepsize by a quadratic interpolation. Table 4.4 shows that BB* is better than SPGM. This might suggest the use of quadratic interpolation in the line search of gradient algorithms in the context of QP problems.

In our work on the box constrained QP problem [6], we found that the best strategy was to alternate the use of the two BB stepsize formulae given in [1]. We tried a similar approach for the SLBQP problems used in this paper, but found that the results were noticeably worse for the SVM problems. For the problems, the alternation method failed to provide a solution in a reasonable number of iterations. At present we have no explanation for the discrepancy in these results. However, we note that [21] have had considerable success in using the two stepsize formulae in a more sophisticated way in the solution of SVM problems (see Table 4.4 for the numerical results of the method, GVPM). It is worth to mention that GVPM is a monotone algorithm. We do not yet know whether there exists some efficient way to combine the alternation technique of the stepsize and the nonmonotone line search.

Acknowledgements The authors would very much like to thank Professor Gaetano Zanghirati at Università di Ferrara, and his colleagues, for the invaluable help that they provided during our numerical tests on the SVM problems.

References

[1] J. Barzilai and J. M. Borwein, *Two-point step size gradient methods*, IMA J. Numer. Anal., 8 (1988), pp. 141-148.

[2] E. G. Birgin, J. M. Martínez, and M. Raydan, *Nonmonotone spectral projected gradient methods on convex sets*, SIAM J. Optim. 10 (2000), pp. 1196-1211.

[3] P. Brucker, *An O(n) algorithm for quadratic knapsack problems*, Operations Research Letters, 3 (1984), pp. 163-166.

[4] P. H. Calamai and J. J. Moré, *Quasi-Newton updates with bounds*, SIAM Journal on Numerical Analysis, 24 (1987), pp. 1434-1441.

[5] Y. H. Dai and R. Fletcher, *On the asymptotic behaviour of some new gradient methods*, Research report NA/212, Department of Mathematics, University of Dundee, 2003.

[6] Y. H. Dai and R. Fletcher, *Projected Barzilai-Borwein methods for large-scale box-constrained quadratic programming*, Research report NA/215, Department of Mathematics, University of Dundee, 2003.

[7] Y. H. Dai and H. C. Zhang, *An adaptive two-point stepsize gradient algorithm*, Numerical Algorithms, 27 (2001) 377-385.

[8] R. Fletcher, *On the Barzilai-Borwein method*, Research report, Department of Mathematics, University of Dundee, 2001.

[9] A. Friedlander, J. M. Martínez, B. Molina, and M. Raydan, *Gradient method with retards and generalizations*, SIAM J. Numer. Anal., 36 (1999), 275-289.

[10] L. Grippo, F. Lampariello, and S. Lucidi, *A nonmonotone line search technique for Newton's method*, SIAM J. Numer. Anal., 23 (1986), pp. 707-716.

[11] M. Held, P. Wolfe and H. Crowder, *Validation of subgradient algorithms*, Mathematical Programming, 6 (1974), pp. 62-88.

[12] R. Helgason, J. Kennington and H. Lall, *A polynomially bound algorithms for a singly constrained quadratic program*, Mathematical Programming, 18 (1980), pp. 338-343.

[13] T. Joachims, *Making large-scale SVM learning practical*, In: Advances in Kernel Methods - Support Vector Learning (B. Schölkopf, C. J. C. Burges and A. Smola, eds.), MIT Press, Cambridge, Massachussets, 1998.

[14] R. R. Meyer, *Multipoint methods for separable nonlinear networks*, Mathematical Programming Study, 22 (1984), pp. 185-205.

[15] J. Moré and G. Toraldo, *Algorithms for bound constrained quadratic programming problems*, Numer. Math., 55 (1989), pp. 377-400.

[16] P. M. Pardalos and N. Kovoor, *An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds*, Math. Prog., 46 (1990), pp. 321-328.

[17] P. M. Pardalos and J. B. Rosen, *Constrained global optimization: Algorithms and applications*, in: Lecture Notes in Computer Science, Vol. 268 (Springer, Berlin, 1987).

[18] J. C. Platt, *Fast training of support vector machines using sequential minimal optimization*, In: Advances in Kernel Methods - Support Vector Learning, B. Schölkopf, C. Burges and A. Smola, eds., MIT Press, Cambridge, Massachusets, 1998.

[19] M. Raydan, *On the Barzilai and Borwein choice of steplength for the gradient method*, IMA J. Numer. Anal., 13 (1993), pp. 321-326.

[20] M. Raydan, *The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem*, SIAM J. Optim., 7 (1997), pp. 26-33.

[21] T. Serafini, G. Zanghirati, L. Zanni, *Gradient projection methods for quadratic programs and applications in training support vector machines*, Research report, 2003.

[22] Ph. L. Toint, *A non-monotone trust region algorithm for nonlinear optimization subject to convex constraints*, Math. Prog. 77 (1997), pp. 69-94.

[23] V. Vapnik, *Estimation of dependences based on empirical data*, Springer-Verlag, 1982.