OQLA/QPALM – Convex quadratic optimization solvers using the augmented Lagrangian approach, with an appropriate behavior on infeasible or unbounded problems

Jean Charles Gilbert, Émilie Joannopoulos

OQLA/QPALM – Convex quadratic optimization solvers using the augmented Lagrangian approach, with an appropriate behavior on infeasible or unbounded problems

J. Charles Gilbert * and Émilie Joannopoulos *

January 28, 2015

When a solver of convex quadratic optimization problem (QP) is used within a nonlinear optimization code, implementing the SQP algorithm, it is important that it deals appropriately with the special QPs that can be generated by the nonlinear solver, those that are *infeasible* or *unbounded*. The goal of this paper is to highlight the potential of the augmented Lagrangian (AL) algorithm in that respect and to give an account on the efficiency of the implementation of this algorithm in the `C++`/`Matlab` codes `Oqla`/`Qpalm`. We show how these pieces of software compare with some frequently used QP solvers, which use active-set or interior-point methods, and demonstrate that they provide an appropriate response when they deal with the special QPs quoted above.

Keywords: augmented Lagrangian algorithm, augmentation parameter update, closest feasible problem, convex quadratic optimization, feasible shift, global linear convergence, infeasible problem, proximal point algorithm, shifted constraint.

Table of contents

1 Introduction

Convex quadratic optimization is often used to model practical problems. It also appears in the solution technique of numerical algorithms dealing with more complex nonlinear optimization problems, like the SQP algorithm (see [7, 33] and the references therein), which we had in mind while developing the solvers described in this paper. Over time, the numerical methods to solve a convex quadratic optimization problem (QP) have formed a significant corpus, representing the basis of a clearly identified discipline. To put it another way, this corpus owes in part its existence to the large body of researches that numerical analysts have dedicated to its central problem. A wide variety of techniques

*INRIA Paris-Rocquencourt, BP 105, F-78153 Le Chesnay, France. E-mails: `Jean-Charles.Gilbert@inria.fr`, `Emilie.Joannopoulos@inria.fr`.

has been proposed, most of them being more or less linked to active-set [25, 40], interior-point [51, 9], or nonsmooth methods [1, 32, 49, 4, 5, 12]. One also encounters other approaches that possibly make use of the just mentioned techniques at some points of their design. The augmented Lagrangian (AL) approach considered below belongs to this last group of heterogeneous methods, since the way of solving the bound constrained QPs generated by its outer loop is left unspecified.

This paper presents two implementations of the AL algorithm, which solves a convex QP by decomposing it in a sequence of bound constrained QPs, or *AL subproblems*, thanks to the AL relaxation. Therefore, this approach implicitly assumes that a convex *bound* constrained QP is simpler to solve than the original QP, which is indeed the case when active-set and gradient-projection [37] techniques are used to solve the AL subproblems, like in the developed solvers. The two implementations are similar, except that they use a different computer language: `C++` for the solver `Oqla` and `Matlab` for the solver `Qpalm`. `Oqla` is of course more computing time efficient than `Qpalm`, which is its main motivation, all the more so as the dimensions increase, but the latter is very useful to experiment rapidly algorithmic ideas and to analyze and understand the behavior of the algorithm. The two pieces of software have similar behavior and iterative performance. The object oriented language used by `Oqla` also makes it flexible enough to take into account problems with various data structures, such as dense and sparse data, but also the L-BFGS structure [39, 34, 24] and others, in a spirit similar to the one presented in [22].

The AL approach is not very often implemented for solving a QP [16, 12], but it has several nice features that we would like to highlight in this paper, in particular in the numerical experiments: (*i*) the AL outer loop converges *globally linearly*, so that the tuning of the penalty parameter can be done after completion of the second AL iteration [14], (*ii*) the algorithm always solves the *closest feasible problem* [10], which makes it insensitive to the possible infeasibility of the problem to solve, and (*iii*) when the closest feasible problem is unbounded, the algorithm can provide a *direction of unboundedness* of that problem during the first AL iteration [10]. These concepts and properties will be clarified below. Both `Oqla` and `Qpalm` have been designed to ensure the three features quoted above. Since in `Oqla`/`Qpalm`, the AL subproblems are solved by an active-set/gradient-projection algorithm, the method can take advantage of a good estimation of the primal-dual solution (warm start), as this is the case when the QPs are generated by the SQP algorithm; the experiments will confirm this intuition.

The paper is organized as follows. We state the problem to solve in section 2, as well as its closest feasible version (section 2.1). We also describe the outer loop of the AL algorithm used in `Oqla`/`Qpalm`, as well as its global linear convergence property and its capacity to detect a direction of unboundedness (section 2.2). The dual interpretation of the algorithm is finally recalled (section 2.3). Section 3 gives more details on the implementation of the AL algorithm, which are common to `Oqla` and `Qpalm`. We restrict the presentation to the features that help understanding the behavior of the solvers. Section 4 reports numerical experiments on the convex QPs of the `CUTEst` collection [27]. A comparison is made with some well known solvers, implementing active-set methods (`Quadprog` [47] and `Qpa` [26]) and interior point methods (`Ooqp` [22] and `Qpb` [26]). Our aim is not to make a definite comparison, since `Oqla`/`Qpalm` can probably still be improved, but to see whether the approach deserves continuing efforts beyond the present stage of the solver development. Section 5 concludes the paper with some discussions and a perspective on the possible

evolution of the solvers.

Notation

We denote by $\mathbb{R}$ the set of real numbers and sets $\mathbb{R}_+ := \{t \in \mathbb{R} : t \geqslant 0\}$, $\mathbb{R}_{++} := \{t \in \mathbb{R} : t > 0\}$, and $\overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, +\infty\}$.

Let $\mathbb{E}$ be a finite dimensional vector space (below, $\mathbb{E}$ is some $\mathbb{R}^p$) and $C \subseteq \mathbb{E}$ be a nonempty closed convex set. The *tangent cone* to C at $x \in C$ is denoted by $\mathrm{T}_x C$; it is the closure of $\mathbb{R}_+(C-x)$. The *asymptotic cone* of C is denoted by $C^\infty := \{d \in \mathbb{E} : C+d \subseteq C\}$. We denote by $\mathcal{I}_S$ the *indicator function* of a set $S \subseteq \mathbb{E}$: $\mathcal{I}_S(x) = 0$ if $x \in S$, $\mathcal{I}_S(x) = +\infty$ if $x \notin S$. The *domain* of a function $f : \mathbb{E} \to \overline{\mathbb{R}}$ is defined and denoted by $\operatorname{dom} f := \{x \in \mathbb{E} : f(x) < +\infty\}$ and its *epigraph* by $\operatorname{epi} f := \{(x, \alpha) \in \mathbb{E} \times \mathbb{R} : f(x) \leqslant \alpha\}$. As in [31], $\mathrm{Conv}(\mathbb{E})$ is the set of functions $f : \mathbb{E} \to \mathbb{R} \cup \{+\infty\}$ that are convex (i.e., $\operatorname{epi} f$ is convex) and proper (i.e., $\operatorname{epi} f \neq \varnothing$); while $\mathrm{C}\overline{\mathrm{onv}}(\mathbb{E})$ is the subset of $\mathrm{Conv}(\mathbb{E})$ of those functions f that are also closed (i.e., $\operatorname{epi} f$ is closed).

Suppose now that $\mathbb{E}$ is endowed with a scalar product denoted by $\langle \cdot, \cdot \rangle$ (below, this one is the standard Euclidean scalar product of some $\mathbb{E} = \mathbb{R}^p$) and let $C \subseteq \mathbb{E}$ be again a nonempty closed convex set. The *normal cone* is denoted and defined by $\mathrm{N}_x C := \{\nu \in \mathbb{E} : \langle \nu, d \rangle \leqslant 0 \text{ for all } d \in C - x\}$; it is the negative dual of the tangent cone: $\mathrm{N}_x C = (\mathrm{T}_x C)^- := \{\nu \in \mathbb{E} : \langle \nu, d \rangle \leqslant 0 \text{ for all } d \in \mathrm{T}_x C\}$. The *orthogonal projector* on C is denoted by P_C. The *subdifferential* at $x \in \mathbb{E}$ of a function $f \in \mathrm{Conv}(\mathbb{E})$ is the set defined and denoted by $\partial f(x) := \{s \in \mathbb{E} : f(y) \geqslant f(x) + \langle s, y - x \rangle, \text{ for all } y \in \mathbb{E}\}$. The *projected gradient* of a differential function $f : \mathbb{E} \to \mathbb{R}$ with respect to C at $x \in C$ is the vector $\nabla^{\mathrm{P}} f(x) := \mathrm{P}_{-\mathrm{T}_x C} \nabla f(x)$; it is known that $x \in C$ minimizes a convex differentiable function f on C if and only if $\nabla^{\mathrm{P}} f(x) = 0$.

2 The AL algorithm

2.1 The problem to solve

We write the *convex quadratic optimization problem* considered by the solvers as follows

$$(P) \quad \begin{cases} \inf_x \, q(x) \\ l_B \leqslant x \leqslant u_B \\ l_I \leqslant Ax \leqslant u_I. \end{cases} \tag{2.1}$$

`Oqla`/`Qpalm` can also deal with linear equality constraints, but the incorporation of these in (2.1) would make the presentation too cumbersome without adding complexity (the unpublished companion paper [23] takes these constraints into account and gives more details on the solvers). In that problem, the objective function

$$q : x \in \mathbb{R}^n \mapsto q(x) = g^{\mathsf{T}} x + \frac{1}{2} x^{\mathsf{T}} H x$$

is convex quadratic ($g \in \mathbb{R}^n$ and $H \in \mathbb{R}^{n \times n}$ is positive semidefinite), while the constraints are defined by a matrix $A \in \mathbb{R}^{m \times n}$ and bounds $l = (l_B, l_I)$ and $u = (u_B, u_I) \in \overline{\mathbb{R}}^{n+m}$ that must satisfy $l < u$. The sign "$^{\mathsf{T}}$" denotes transposition. Since H may vanish, the problem encompasses linear optimization.

One can classify the types of convex optimization problem (P), according to their optimal value $\mathrm{val}(P)$. When its *feasible set* is empty (meaning that there is no $x \in \mathbb{R}^n$ satisfying its constraints), we write $\mathrm{val}(P) = +\infty$ (by definition) and we say that the problem is *infeasible*. When $\mathrm{val}(P) = -\infty$, the feasible set contains a sequence $\{x_k\}$ such that $q(x_k) \to -\infty$; we say in that case that the problem is *unbounded*. Otherwise, $\mathrm{val}(P) \in \mathbb{R}$ and by a result of Frank and Wolfe [20; 1956, appendix (i)], the problem has a solution (even when H is indefinite actually).

When problem (2.1) is infeasible, the AL algorithm described in the next section has the nice feature of finding a solution to the closest feasible problem (provided this one is bounded) [10]. To define that problem, let us first introduce the notion of *feasible shift*, which a vector $s \in \mathbb{R}^m$ such that $l_I \leqslant Ax + s \leqslant u_I$ is feasible for some $x \in [l_B, u_B]$. Note that the linear inequality constraints of (2.1) are shifted, but not its bound constraints. This choice arises out of the fact that the latter constraints will be maintained unchanged in the AL subproblems, while the former constraints will be relaxed. The set of feasible shifts is clearly nonempty and is denoted by $\mathcal{S}$. It is actually easy to see that this set can be written

$$\mathcal{S} = [l_I, u_I] - A([l_B, u_B]). \tag{2.2}$$

As the sum of two convex polyhedra, $\mathcal{S}$ is a convex polyhedron, hence a nonempty closed convex set. It has, therefore, a smallest element $\tilde{s}$, called the *smallest feasible shift*. This one is defined by

$$\tilde{s} := \operatorname*{arg\,min}_{s \in \mathcal{S}} \|s\|, \tag{2.3}$$

where "arg min" denotes the set of minimizers of the problem on which it applies and $\|\cdot\|$ denotes the Euclidean norm. The *closest feasible problem* can now be defined:

$$(P_{\tilde{s}}) \quad \begin{cases} \inf_x \ q(x) \\ l_B \leqslant x \leqslant u_B \\ l_I \leqslant Ax + \tilde{s} \leqslant u_I. \end{cases} \tag{2.4}$$

Since $\tilde{s} \in \mathcal{S}$, that problem is feasible, but may be unbounded.

It will be useful to have a characterization of the smallest feasible shift that does not make use of it (because that vector is unknown when the AL algorithm is running). The first step to get that characterization is to remove the feasible shift from problem (2.3) by substituting it by its expression in (2.2). Writing $s = y - Ax$ with $(x, y) \in [l, u]$, one gets as a new formulation of (2.3):

$$\min_{(x,y) \in [l,u]} \|Ax - y\|. \tag{2.5}$$

Proposition 2.1 (smallest feasible shift caracterizations) *The following properties of $(\tilde{x}, \tilde{y}) \in [l, u]$ are equivalent:*

(i) $(\tilde{x}, \tilde{y})$ is a solution to problem (2.5),

(ii) $A\tilde{x} + \tilde{s} = \tilde{y}$,

(iii) $\mathrm{P}_{-\mathrm{T}_{\tilde{x}}[l_B,u_B]}\left(A^{\mathsf{T}}(A\tilde{x} - \tilde{y})\right) = 0$ and $\mathrm{P}_{[l_I,u_I]}(A\tilde{x}) = \tilde{y}$.

Proof. $[(i) \Leftrightarrow (ii)]$ This is a direct consequence of the fact that (2.3) and (2.5) are two expressions of the same problem.

[$(i) \Leftrightarrow (iii)$] Since problem (2.5) consists in minimizing a convex function on a box, the pair $(\tilde{x}, \tilde{y}) \in [l, u]$ solves that problem if and only if the projected gradient of the half square of the objective of (2.5) vanishes at that point:

$$\mathrm{P}_{-\,\mathrm{T}_{(\tilde{x},\tilde{y})}[l,u]} \begin{pmatrix} A^{\mathsf{T}}(A\tilde{x} - \tilde{y}) \\ -(A\tilde{x} - \tilde{y}) \end{pmatrix} = 0.$$

This can be decomposed in

$$\mathrm{P}_{-\,\mathrm{T}_{\tilde{x}}[l_B,u_B]} \left(A^{\mathsf{T}}(A\tilde{x} - \tilde{y}) \right) = 0 \qquad \text{and} \qquad \mathrm{P}_{-\,\mathrm{T}_{\tilde{y}}[l_I,u_I]}(\tilde{y} - A\tilde{x}) = 0.$$

The first identity is the first part of (iii). The second identity can also be written $\mathrm{P}_{\mathrm{T}_{\tilde{y}}[l_I,u_I]}(A\tilde{x} - \tilde{y}) = 0$ or $A\tilde{x} - \tilde{y} \in (\mathrm{T}_{\tilde{y}}[l_I, u_I])^-$ (by Moreau's decomposition [38]) or $A\tilde{x} - \tilde{y} \in \mathrm{N}_{\tilde{y}}[l_I, u_I]$. This last statement can equivalently be written $\mathrm{P}_{[l_I,u_I]}(A\tilde{x}) = \tilde{y}$, which is the second part of (iii). □

As announced, the iterest of point (iii) is that it gives condtions ensuring that the constraints of the closest feasible problem (2.4) are satisfied, without using the unknown smallest feasible shift $\tilde{s}$, which appears in one constraint of that problem.

2.2 The AL algorithm

The AL algorithm can be defined by first introducing an auxiliary vector of variables $y \in \mathbb{R}^m$ and by rewriting (2.1) as follows

$$\begin{cases} \inf_{(x,y)} \; q(x) \\ l \leqslant (x, y) \leqslant u \\ Ax = y. \end{cases} \tag{2.6}$$

Given a *penalty* or *augmentation parameter* $r \geqslant 0$, the *augmented Lagrangian* $\ell_r : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is then defined at $(x, y, \lambda) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m$ by

$$\ell_r(x, y, \lambda) = q(x) + \lambda^{\mathsf{T}}(Ax - y) + \frac{r}{2}\|Ax - y\|^2.$$

For $r = 0$, one gets the ordinary Lagrangian function $\ell = \ell_0$, relaxing the equality constraints of (2.6) thanks to the *multiplier* or *dual variable* λ. This one will be used below for introducing the associated dual function, see (2.13).

The *AL algorithm* (sometimes called *method of multipliers*) generates a sequence of dual variables $\{\lambda_k\}_{k\in\mathbb{N}} \subseteq \mathbb{R}^m$, which are aimed at converging to a dual solution associated with the equality constraints of (2.6), and a sequence of augmentation parameters $\{r_k\}_{k\in\mathbb{N}} \subseteq \mathbb{R}_{++}$, as follows. Knowing $\lambda_k \in \mathbb{R}^m$ and $r_k > 0$, the next dual iterate λ_{k+1} and penalty parameter r_{k+1} are computed by

$$(x_{k+1}, y_{k+1}) \in \arg\min \left\{ \ell_{r_k}(x, y, \lambda_k) : (x, y) \in [l, u] \right\}, \tag{2.7}$$

$$\lambda_{k+1} := \lambda_k + r_k(Ax_{k+1} - y_{k+1}), \tag{2.8}$$

$$\text{update } r_k \curvearrowright r_{k+1}. \tag{2.9}$$

We have already said that the QP in (2.7) is called the *AL subproblem* and a cycle (2.7)-(2.9) is called an (*outer*) *AL iteration*. The rule to update r_k will be discussed in section 3.2.

For the definition of the AL algorithm and its use in nonlinear optimization, see [30, 42, 43, 8, 45, 2, 46, 11, 6]. The AL algorithm has also been used to solve problems in linear optimization [41, 28], in quadratic optimization [17, 19, 18, 14, 13, 21, 10], in SDP optimization [35, 36, 29, 50], and in conic optimization [48].

Instead of using the optimal values $\mathrm{val}(P)$ to classify the types of convex optimization problem (P) in (2.1), as we did in the second paragraph of section 2.1, one can also classify the types of problem, according to the status of its closest feasible problem. As we shall see, this is instructive when the AL algorithm is used to solve (P). There is now only two possibilities, since the closest feasible problem is always feasible.

- If the closest feasible problem is bounded, it has a solution [20] and the AL algorithm finds one of its solutions (see [10; proposition 3.1] and the references therein). Of course if problem (P) is feasible, the smallest feasible shift $\tilde{s}$ vanishes and the AL algorithm finds a solution to the original problem. This is done without computing $\tilde{s}$ before running the algorithm (this would be as difficult as solving a general feasible convex quadratic optimization problem). Actually, it can be shown [10; theorem 3.4] that the constraint values
$$s_k := y_k - Ax_k \tag{2.10}$$
converge to $\tilde{s}$ at a global linear speed, in the sense that
$$\begin{array}{c} \forall\, \beta > 0, \quad \exists\, L > 0, \quad \mathrm{dist}(\lambda_0, \tilde{\mathcal{S}}_{\mathrm{D}}) \leqslant \beta \quad \text{implies that} \\ \forall\, k \geqslant 1, \quad \|s_{k+1} - \tilde{s}\| \leqslant \frac{L}{r_k} \|s_k - \tilde{s}\|. \end{array} \tag{2.11}$$
In this claim, "dist" denotes the Euclidean distance and $\tilde{\mathcal{S}}_{\mathrm{D}}$ is the set of dual solutions associated with the inequality constraints $l_I \leqslant A_I x + \tilde{s} \leqslant u_I$ of the closest feasible problem (2.4). The first line in (2.11) is used to determine the constant L, which depends on the distance between the initial dual iterate λ_0 and $\tilde{\mathcal{S}}_{\mathrm{D}}$ (more exactly, so that the inequality is useful, knowing λ_0, one must take β larger than $\mathrm{dist}(\lambda_0, \tilde{\mathcal{S}}_{\mathrm{D}})$; then L is set according to the value of β). The second line in (2.11) highlights the *global* linear speed of convergence of s_k to $\tilde{s}$, which occurs as soon as s_k and s_{k+1} are available (that is, at the end of iteration $k \geqslant 1$) and r_k is large enough (larger than L). The way this estimate is used to update r_k will be discussed in section 3.2.

- If the closest feasible problem is unbounded, it has a *direction of unboundedness* [10; lemma 2.2], which is a direction $d \in \mathbb{R}^n$ such that
$$g^{\mathsf{T}} d < 0, \qquad Hd = 0, \qquad d \in [l_B, u_B]^{\infty}, \qquad \text{and} \qquad Ad \in [l_I, u_I]^{\infty}. \tag{2.12}$$
The first two conditions tell us that the quadratic objective q tends to $-\infty$ along the direction d (starting from any point), while the last two conditions indicate that d is in the asymptotic cone of the feasible set $\mathcal{F}_{\tilde{s}}$ of the closest feasible problem. Hence, if $x \in \mathcal{F}_{\tilde{s}}$, then $x + td \in \mathcal{F}_{\tilde{s}}$ for all $t \geqslant 0$ and $q(x+td) \to -\infty$ when $t \to \infty$. Such a direction of unboundedness is useful when the QP solver is used within the SQP algorithm.

Let us now show that the AL algorithm can detect efficiently the unboundedness of the closest feasible problem, provided the solver of AL subproblem is equipped with a device allowing the detection of the unboundedness of the latter problem. Indeed, the map $d \in \mathbb{R}^n \mapsto (d, Ad) \in \mathbb{R}^n \times \mathbb{R}^m$ is a bijection between the directions of unboundedness of

the closest feasible problem and those of the AL subproblem. To see this, observe using (2.12), that a direction of unboundedness (d_x, d_y) of the AL subproblem in step (2.7) is characterized by

$$\Big(g + Hx + A^T\big(\lambda + r[Ax - y]\big)\Big)^T d_x - \Big(\lambda + r[Ax - y]\Big)^T d_y < 0,$$
$$\begin{pmatrix} H + rA^TA & -rA^T \\ -rA & rI \end{pmatrix} \begin{pmatrix} d_x \\ d_y \end{pmatrix} = 0, \quad \text{and} \quad (d_x, d_y) \in [l, u]^\infty = [l_B, u_B]^\infty \times [l_I, u_I]^\infty.$$

Then, easy mathematical manipulations show that these conditions are equivalent to (2.12), with the connection $(d_x, d_y) = (d, Ad)$.

2.3 Dual interpretation of the algorithm

The AL algorithm is essentially a dual method in the sense that it generates multipliers λ_k in the range space (actually, its dual identified to $\mathbb{R}^m$) of the equality constraints of (2.6), while the primal pair (x_{k+1}, y_{k+1}) can be considered as made of auxiliary variables generated by (2.7) in order to make the update of λ_k in (2.8). Even though this paper focuses on an implementation of the AL algorithm, its proximal interpretation cannot be ignored at some point of the discussion (e.g., in example 3.1) and this goes through the introduction of the dual function.

The *Lagrangian* of problem (2.6), relaxing its equality constraints, is the function $\ell : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m \mapsto \mathbb{R}$ defined at (x, y, λ) by

$$\ell(x, y, \lambda) = q(x) + \lambda^T(Ax - y). \tag{2.13}$$

The associated *dual function* $\delta : \mathbb{R}^m \to \overline{\mathbb{R}}$ is then defined at λ by

$$\delta(\lambda) := -\inf_{(x,y)\in\mathbb{R}^n\times[l,u]} \ell(x, y, \lambda). \tag{2.14}$$

With the minus sign in front of the infimum, this function is convex, closed, and does not take the value $-\infty$. Therefore,

$$\delta \in \mathrm{C}\overline{\mathrm{onv}}(\mathbb{R}^m) \qquad \Longleftrightarrow \qquad \operatorname{dom}\delta \neq \varnothing. \tag{2.15}$$

It can be shown [10; proposition 2.5] that the following properties are equivalent:

- (i) $\operatorname{dom}\delta \neq \varnothing$,
- (ii) the closest feasible problem (2.4) has a solution,
- (iii) the AL subproblem in (2.7) has a solution, whatever is $r_k > 0$ and $\lambda_k \in \mathbb{R}^m$.

By (2.15), condition (i) is equivalent to $\delta \in \mathrm{C}\overline{\mathrm{onv}}(\mathbb{R}^m)$. When the equivalent conditions (i)-(iii) hold, the AL algorithm is well defined (i.e., it does not fail to find a solution to the AL subproblem, thanks to (iii)) and eventually computes a solution to the closest feasible problem (2.4) [10; section 3.1]. On the other hand, when the equivalent conditions (i)-(iii) fail, the first AL subproblem is unbounded and any of its directions of unboundedness can be used to get a direction of unboundedness of the closest feasible problem, as described at the end of section 2.2. It is therefore important that the solver of the AL subproblem is able to find a direction of unboundedness if such a direction exists. The design of the algorithm to solve the AL subproblem has been done with that concern in mind (see section 3.3).

It is also instructive to have in mind the proximal interpretation of the AL algorithm [44]: it can indeed be viewed as the *proximal (point) algorithm* on the dual function. This means that, when (i)-(iii) hold, the new multiplier λ_{k+1} computed by (2.8) is also the unique solution to the problem

$$\inf_{\mu \in \mathbb{R}^m} \left(\delta(\mu) + \frac{1}{2r_k} \|\mu - \lambda_k\|^2 \right).$$

Furthermore, the constraint value $s_{k+1} := y_{k+1} - Ax_{k+1}$ is a subgradient of the dual function at λ_{k+1}:

$$s_{k+1} \in \partial\delta(\lambda_{k+1}). \tag{2.16}$$

Hence, the AL algorithm tries to find a minimizer of the dual function, if this one exists. If problem (2.1) is infeasible, then δ is unbounded below and $\delta(\lambda_k) \to -\infty$ [10; proposition 2.7].

3 Implementation details

There are two aspects of the outer loop of the AL algorithm (2.7)-(2.9) that have not been made explicit so far: the stopping criterion of the outer iterations and the way of updating the augmentation parameter r_k at each outer iteration in (2.9); these topics are addressed in sections 3.1 and 3.2 respectively. The difficult part of the AL algorithm is, of course, the minimization problem in (2.7); the techniques implemented in `Oqla`/`Qpalm` to solve it are described in section 3.3. By lack of space, we restrict the description to the most salient and stable aspects of the implementation, letting the evolving companion paper [23] give more details.

3.1 Stopping criterion

In exact arithmetic, a solution to problem (2.1) is found at the end of the kth outer iteration (2.7)-(2.9) when the constraint value $s_{k+1} = y_{k+1} - Ax_{k+1}$ vanishes (then λ_{k+1} is a minimizer of the dual function by (2.16)). When the QP is infeasible, however, this situation will never occur and it is desirable to detect instead whether a solution to the closest feasible problem (2.4) has been found (we have already said that the AL algorithm solves that problem if this one is bounded). This is less straightforward, since the smallest feasible shift $\tilde{s}$ is not known before problem (2.4) is solved, and its approximation s_k claimed by (2.11) is of no help at this point. Luckily, it can be shown [10; proposition 2.18] that a pair $(\overline{x}, \overline{y})$ solves the closest feasible problem if and only if there is some $\overline{\lambda} \in \mathbb{R}^m$ such that

$$(\overline{x}, \overline{y}) \in \operatorname*{arg\,min}_{(x,y) \in \mathbb{R}^n \times [l,u]} \ell_r(x, y, \overline{\lambda}), \tag{3.1}$$

$$A^T(A\overline{x} - \overline{y}) = 0, \tag{3.2}$$

$$\mathrm{P}_{[l,u]}(A\overline{x}) = \overline{y}. \tag{3.3}$$

where $\mathrm{P}_{[l,u]}$ denotes the *orthogonal projector* on $[l, u]$. The interest of these conditions is that they do not use $\tilde{s}$. Now, condition (3.1) is realized by the minimization phase in (2.7), so that only (3.2) and (3.3) must be verified to guarantee convergence to a solution to

the closest feasible problem. The stopping criterion used in `Oqla`/`Qpalm` is based on this observation and reads

$$\|A^{\mathsf{T}}(Ax_{k+1} - y_{k+1})\| \leqslant \varepsilon^{\mathrm{s}}_{\mathrm{feas}} \qquad \text{and} \qquad \|\,\mathrm{P}_{[l,u]}(Ax_{k+1}) - y_{k+1}\| \leqslant \varepsilon^{\mathrm{s}}_{\mathrm{feas}},$$

where $\varepsilon^{\mathrm{s}}_{\mathrm{feas}}$ is a positive shifted feasibility tolerance. These conditions are checked at the end of the kth outer iteration (2.7)-(2.9).

3.2 Augmentation parameter update

Finding an appropriate value for the augmentation parameter r_k is a difficult task. In theory, the convergence of the AL algorithm, in the sense that $s_k \to \tilde{s}$, is guaranteed if r_k is bounded away from zero [10; proposition 3.1], but the convergence may be slow. In order to have a sufficiently fast convergence, r_k must be chosen large enough as the estimate (2.11) suggests it. The solvers `Oqla`/`Qpalm` use this estimate to set up the penalty parameters r_k at each iteration, but, in order to use it two difficulties must be overcome: getting rid of the unknown smallest feasible shift $\tilde{s}$ and estimating the unknown constant L.

3.2.1 Global linear convergence of the constraint change

If the constraint value $s_k := y_k - Ax_k$ converges globally linearly to $\tilde{s}$, with a rate of convergence $\rho_k < \sqrt{2} - 1$, meaning that $\|s_{k+1} - \tilde{s}\| \leqslant \rho_k \|s_k - \tilde{s}\|$ for all $k \geqslant 1$, the *constraint change*

$$s'_k := s_k - s_{k-1}$$

converges globally linearly to 0 with the rate $\rho'_k := (1 + \rho_k)\rho_k/(1 - \rho_k) < 1$, meaning that $\|s'_{k+1}\| \leqslant \rho'_k \|s'_k\|$ for all $k \geqslant 2$ (see for instance [10; proposition 4.1]). The interest of s'_k here is that its limit is known, as opposed to s_k. Now, if $\rho_k = L/r_k$ like in (2.11), and if $L/r_k < \sqrt{2} - 1$, there holds

$$\forall\, k \geqslant 2 : \quad \|s'_{k+1}\| \leqslant \frac{L'}{r_k} \|s'_k\|, \tag{3.4}$$

where $L' = L/(2/\sqrt{2} - 1)$.

3.2.2 Estimating the rate of convergence of the constraint change

The constant L appearing explicitly in (2.11), and implicitly in (3.4), is generally unknown. Sometimes it depends on the solution to the problem in a complex manner. To highlight this fact we consider example 4.3 from [14; 2005].

Example 3.1 Consider the trivial problem

$$\begin{cases} \inf 0 \\ l \leqslant 0x \leqslant u, \end{cases}$$

where l and u are both in $\mathbb{R}$ and verify $l < 0 < u$ (it has the form (2.1) with $q \equiv 0$, $m = 1$, and $A = 0$). The problem has primal solutions (all the points in $\mathbb{R}^n$ are solutions) and a single dual solution $\overline{\lambda} = 0 \in \mathbb{R}$. The dual function (2.14) has for value at $\lambda \in \mathbb{R}$:

$$\delta(\lambda) = \begin{cases} l\,\lambda & \text{if } \lambda \leqslant 0 \\ u\,\lambda & \text{if } \lambda > 0. \end{cases}$$

Suppose now that the initial multiplier λ_0 of the AL algorithm is positive. We claim that, if $r \leqslant \lambda_0/(2u)$, there is no strict reduction in the constraint value during the first two iterations: $y_2/y_1 = 1$ (see figure 3.1). Indeed when $\lambda_k > 0$, one gets by (2.7):

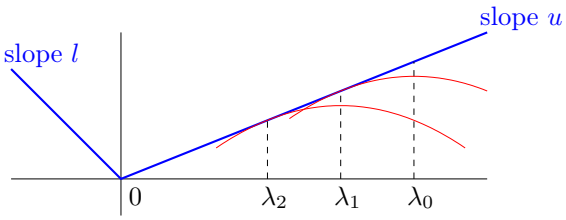


Figure 3.1: Illustration of the first two AL iterations of example 3.1, viewed as proximal iterations on the dual function, when $r \leqslant \lambda_0/(2u)$

$$y_{k+1} := \operatorname*{arg\,min}_{y \in [l,u]} \left(-\lambda_k y + \frac{r}{2} y^2 \right) = \min \left(u, \frac{\lambda_k}{r} \right).$$

Since $\lambda_0 > 0$ and $r \leqslant \lambda_0/(2u)$, it follows that

$$\begin{aligned} y_1 &= \min(u, \lambda_0/r) = u, \\ \lambda_1 &= \lambda_0 - r y_1 = \lambda_0 - ru \geqslant \lambda_0/2, \\ y_2 &= \min(u, \lambda_1/r) = u, \end{aligned}$$

so that $y_2/y_1 = 1$.

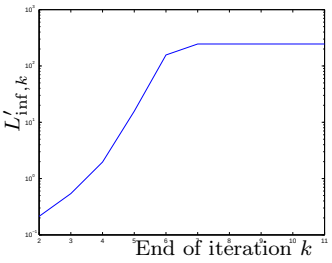
A consequence of this claim is that a strict reduction in the constraint norm can be obtained only if $r > \lambda_0/(2u)$. For fixed $\lambda_0 > 0$ and $u \downarrow 0$ (i.e., when the constraint value 0 is closer and closer to its bound u), r must tend to ∞ to get a strict reduction in the constraint norm, hence L also tends to infinity. □

A consequence of example 3.1 is that hoping to find an appropriate value for r_k from the problem data by a magic formula (in particular for r_0 like in [6]) is probably highly unrealistic. In `Oqla`/`Qpalm`, the initial augmentation parameter r_0 is set therefore to 1 (unless the user proposes another initial value) and the next values of r_k, for $k \geqslant 3$, are determined by using the estimate (3.4) and a desired rate of convergence provided by the user, as follows.

At the end of iteration $k \geqslant 2$, one can compute the following lower bound of the constant L' appearing in (3.4):

$$L'_{\mathrm{inf},k} := \max_{2 \leqslant i \leqslant k} \frac{r_i \, \|s'_{i+1}\|}{\|s'_i\|}.$$

The figure on the right shows a typical evolution of $L'_{\mathrm{inf},k}$ during the iterations and its fast stabilization (the plot has been obtained from the output generated by `Qpalm` on a random convex QP).



3.2.3 Updating the augmentation parameter

If the desired linear convergence rate of s'_k to zero is $\rho'_{\text{des}} \in]0,1[$ and if $L'_{\text{inf},k}$ is a good approximation to L', it makes sense to set

$$r_{k+1} = \frac{L'_{\text{inf},k}}{\rho'_{\text{des}}}, \tag{3.5}$$

which is the setting used by the solvers `Oqla`/`Qpalm`.

Now the rate of convergence of the constraint changes $\{s'_k\}$ to zero is less striking than the one of the constraint values $\{s_k\}$ to $\tilde{s}$, so that the user of `Oqla`/`Qpalm` is invited to provide a desired rate of convergence $\rho_{\text{des}} \in]0,1[$ of the latter sequence, while ρ'_{des} used in (3.5) is set to

$$\rho'_{\text{des}} := \frac{\rho_{\text{des}}}{1 + 2\rho_{\text{des}}}. \tag{3.6}$$

This is justified by the fact that, if the sequence $\{s'_k\}$ satisfies $\|s'_{k+1}\| \leqslant \rho' \|s'_k\|$ for all $k \geqslant k_1$ and some $\rho' \in]0,1[$, then the sequence $\{s_k\}$ converges to some $\tilde{s}$ with $\|s_{k+1} - \tilde{s}\| \leqslant \rho'/(1-2\rho')\|s_k - \tilde{s}\|$ for all $k \geqslant k_1 - 1$ (see for instance [10; proposition 4.1]). Now the rate of convergence $\rho'/(1-2\rho')$ of the sequence $\{s_k\}$ is less than the desired rate ρ_{des} when $\rho' \leqslant \rho_{\text{des}}/(1+2\rho_{\text{des}}) = \rho'_{\text{des}}$.

The `Oqla`/`Qpalm` solvers set $r_1 := r_0$ and set r_2 by using a technique similar to the one described above, but assuming that (2.11) holds with $\tilde{s} = 0$.

3.2.4 Effect on the number of AL iterations

Of course, the choice of ρ_{des} has, by its role in the determination of the augmentation parameter r_k explained above in this section, in particular by its role in (3.5)-(3.6), a direct impact on the number of AL iterations required to reach a solution with the desired accuracy. Figure 3.2 shows in the left plot the impact of the choice of ρ_{des} on the number of

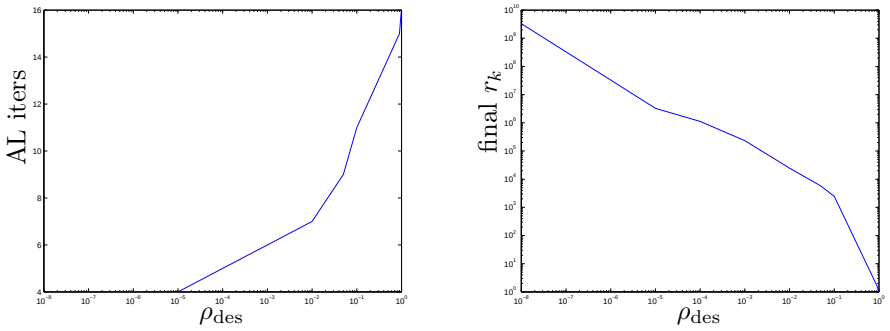


Figure 3.2: Impact of ρ_{des} on the number of iterations (left) and the final r_k

AL iterations, which ranges from 4 to 16, when ρ_{des} ranges from 10^{-8} to 0.99 (logarithmic scale), on a particular convex QP, which seems to us representative. It shows in the right plot the impact of the choice of ρ_{des} on the final value of r_k, which ranges from $3\,10^9$ to 1 (logarithmic scale), when ρ_{des} ranges again from 10^{-8} to 0.99 (logarithmic scale), for the same runs. One can remember from these experiments that the number of AL iterations can be kept rather small (around 10) with a desired rate of convergence $\rho_{\text{des}} = 0.1$.

3.3 Solving the AL subproblem

The cornerstone of the AL algorithm (2.7)-(2.9) is of course the technique used to minimize the AL at each iteration, its step (2.7). Many algorithms could be used to solve that AL subproblem. A selection criterion should certainly be that such an algorithm must be significantly more efficient for solving (2.7) than for solving the original problem (2.1), since otherwise nothing would be gained by decomposing a convex QP in a series of AL subproblems, like the AL algorithm does. This is not hopeless since the latter has only bound constraints while the former has general linear constraints. Any choice has an impact on the efficiency of the overall algorithm and on its features. The one that has been implemented in the current version of `Oqla`/`Qpalm` combines active-set (AS) and gradient-projection (GP) techniques, hence inspired by the work of Moré and Toraldo [37]. We shall briefly discuss several possible implementations and show that each of them can detect a direction of unboundedness, i.e., verify (2.12).

The method used to solve the AL subproblem may be viewed as an infinite loop, made of two phases: the gradient projection (GP) phase and the minimization (MIN) phase. Each phase decides which phase to do next, so that they may be several GP or MIN phases in sequence, but it is the GP phase that decides when to break the infinite loop. In pseudo-language, the code looks like this:

```
phase = gp_phase_flag;
while true
  if phase == gp_phase_flag
    gp_phase;
  elseif phase == cg_phase_flag
    cg_phase;
  else
    break;
  end
end
```

By the first statement, the loop starts by a GP phase, which is appropriate since this one can detect optimality that could occur at the starting point. Next, it is assumed that the variable `phase` is set by the procedures `gp_phase` and `cg_phase` to one of the flags `gp_phase_flag` and `gp_phase_flag`, or to a flag indicating a desire of loop break.

We now give more details on the GP and MIN phases. In both cases, we will see that the variable $y \in \mathbb{R}^m$ can be eliminated, which decreases the number of variables by working only with the variables $x \in \mathbb{R}^n$. To lighten the notation we set $\lambda \equiv \lambda_k$ and $r \equiv r_k$ and assume that the problem to solve at the kth outer iteration of the AL algorithm reads

$$\inf_{(x,y)\in\mathbb{R}^n\times[l,u]} \ell_r(x, y, \lambda). \tag{3.7}$$

One iteration of the algorithm to solve that problem corresponds to one loop in the schematic algorithm above: it starts with the iterate denoted by (x_i, y_i) and ends up with the new iterate (x_{i+1}, y_{i+1}), hence we drop the index k of the outer loop (2.7)-(2.9) of the AL algorithm. The initial iterate (x_0, y_0) is the solution found to the previous AL subproblem if any.

3.3.1 The gradient-projection phase

The goal of a gradient-projection (GP) phase is to ensure the theoretical convergence of the algorithm that solves the AL subproblem and to make it more efficient by activating/inactivating many bounds at the same time in order to find rapidly those that are active at the search solution.

The function φ_λ

The special structure of the AL, with a diagonal Hessian in y (it is a multiple of the identity) makes it possible to have a closed-form formula for its minimizer in $y \in [l, u]$. Problem (3.7) can indeed be written

$$\inf_{x \in \mathbb{R}^n} \varphi_\lambda(x),$$

where the map $\varphi_\lambda : \mathbb{R}^n \to \mathbb{R}$ is defined at $x \in \mathbb{R}^n$ by

$$\varphi_\lambda(x) = \inf_{y \in [l,u]} \ell_r(x, y, \lambda) \tag{3.8}$$

$$= \inf_{y \in [l,u]} \left(\frac{r}{2} \left\| Ax + \frac{\lambda}{r} - y \right\|^2 + \text{terms independent of } y \right) \tag{3.9}$$

$$= \ell_r(x, \breve{y}(x), \lambda), \tag{3.10}$$

in which the map $\breve{y} : \mathbb{R}^n \to \mathbb{R}^m$, defined at $x \in \mathbb{R}^n$ by

$$\breve{y}(x) := \mathrm{P}_{[l,u]} \left(Ax + \frac{\lambda}{r} \right), \tag{3.11}$$

provides the unique solution to the problem in (3.9). The map $\breve{y}$ is clearly continuous and piecewise linear. Hence φ_λ is continuous and piecewise quadratic. Actually, the function φ_λ is also convex and differentiable.

Proposition 3.2 (convexity and differentiability of φ_λ) *The function φ_λ defined by (3.8) is convex and differentiable on $\mathbb{R}^n$. Its gradient at $x \in \mathbb{R}^n$ is given by*

$$\nabla \varphi_\lambda(x) = \nabla_x \ell_r(x, \breve{y}(x), \lambda) = g + Hx + A^\mathsf{T}\big(\lambda + r[Ax - \breve{y}(x)]\big), \tag{3.12}$$

where $\breve{y}(x)$ is defined by (3.11).

Proof. Observe first that φ_λ is convex since it is the marginal function [31; section IV.2.4] of the convex function $(x, y) \mapsto \ell_r(x, y, \lambda) + \mathcal{I}_{\mathbb{R}^n \times [l_I, u_I]}(x, y)$. This property also implies that its subdifferential at x is given by [31; corollary VI.4.5.3]

$$\partial \varphi_\lambda(x) = \{s : (s, 0) \in \partial_{(x,y)} \ell_r(x, \breve{y}(x), \lambda)\}.$$

Now, since ℓ_r is differentiable, $\partial_{(x,y)} \ell_r(x, \breve{y}(x), \lambda)$ is a singleton. Therefore, $\partial \varphi_\lambda(x)$ is also a singleton. This implies [31; corollary VI.2.1.4] that φ_λ is actually a convex differentiable function with a gradient given by

$$\nabla \varphi_\lambda(x) = \nabla_x \ell_r(x, \breve{y}(x), \lambda),$$

which yields (3.12). □

Detecting optimality

Optimality of the AL subproblem is detected at the beginning of the GP phase by checking whether the gradient of φ_λ almost vanishes at the iterate x_i:

$$\|\nabla\varphi_\lambda(x_i)\| \leqslant \varepsilon_{\rm opt}, \tag{3.13}$$

where $\varepsilon_{\rm opt}$ is a small positive tolerance given by the user (when there are also bounds on the variable x, one uses the *projected gradient* of φ_λ instead, see [23]). This makes sense, since it follows from the computation (3.8)-(3.10) that $\nabla\varphi_\lambda(x_i) = 0$ implies that $(x_i, \breve{y}(x_i))$ minimizes the AL on $\mathbb{R}^n \times [l, u]$. Therefore, if (3.13) holds, the AL subproblem is declared to be solved at $(x_i, y_i) := (x_i, \breve{y}(x_i))$.

Detecting unboundedness

The GP phase

When optimality is not reached at the current iterate x_i, then $\nabla\varphi_\lambda(x_i) \neq 0$ and the GP phase consists in forcing the decrease of φ_λ along the gradient path

$$p : \alpha \in \mathbb{R}_+ \mapsto p(\alpha) := x_i - \alpha\nabla\varphi_\lambda(x_i) \tag{3.14}$$

(or along the projected gradient path when there are bounds on x, see [23]). Since $\alpha \in \mathbb{R}_+ \mapsto \varphi_\lambda(p(\alpha))$ is piecewise quadratic, conputing the exact minimizer is possible by comparing the values of φ_λ at its local minimizers on all the pieces. Some authors suggest that technique (see for example [40; section 16.7]), but for problems with many inequality constraints (or bound constraints), this option may be time consuming. For this reason, we have preferred following Moré and Toraldo [37] who only require a sufficient decrease of the function to minimize, here φ_λ.

The `Oqla`/`Qpalm` solvers offer the possibility to use Armijo's or Goldstein's rule [7; section 3.5] for realizing the approximate minimization of φ_λ along p. The former rule has the minor inconvenient of requiring a "good" initial trial stepsize (bounded away from zero along the iterations is sufficient in theory), but it is usually more robust in the presence of rounding errors in the final iterations minimizing the AL, so that we only present that rule here. This one consist in determining a stepsize $\alpha_i > 0$ such that

$$\varphi_\lambda(p(\alpha_i)) \leqslant \varphi_\lambda(x_i) - \omega\,\alpha_i\,\|\nabla\varphi_\lambda(x_i)\|^2,$$

where ω is a constant taken in $]0, 1/2[$. Such a stepsize always exists. The next iterate is then set to

$$x_{i+1} := p(\alpha_i) \qquad \text{and} \qquad y_{i+1} := \breve{y}(x_{i+1}).$$

3.3.2 The MIN phase

The goal of the minimization (MIN) phase is to decrease the AL significantly on the face activated by the GP phase (hit-and-fix strategy) or the affine hull of this one (explore-outside strategy). In `Oqla`/`Qpalm`, this is done using conjugate-gradient (CG) iterations. The interest of using CG iterations is that the solvers are then adapted to large scale problems.

The hit-and-fix strategy

The explore-outside strategy

4 Numerical experiments

The goal of this section is not to convince the reader that `Oqla`/`Qpalm` are the best convex QP solvers written so far. They are not, but they can still be improved significantly if some theoretical obstacles can be overcome. Rather, we would like to see whether it is worth continuing exploring the properties of the AL algorithm with the aim at improving the efficiency of solvers based on this algorithm. For this reason, we have chosen to compare `Oqla`/`Qpalm` to a limited number of some well established solvers, some use an active-set (AS) method, others use an interior-point (IP) method. We will show that the AS-GP technique used to solve the AL subproblem makes the solvers appropriate to solve convex QPs generated by the SQP algorithm.

4.1 Selection of the test-problems

4.2 Default options of the solver

Determining the default options of an optimization solver can be considered as an optimization in itself [3].

Figure 4.1 compares the performances of `Qpalm` without preconditioner (dashed curve)

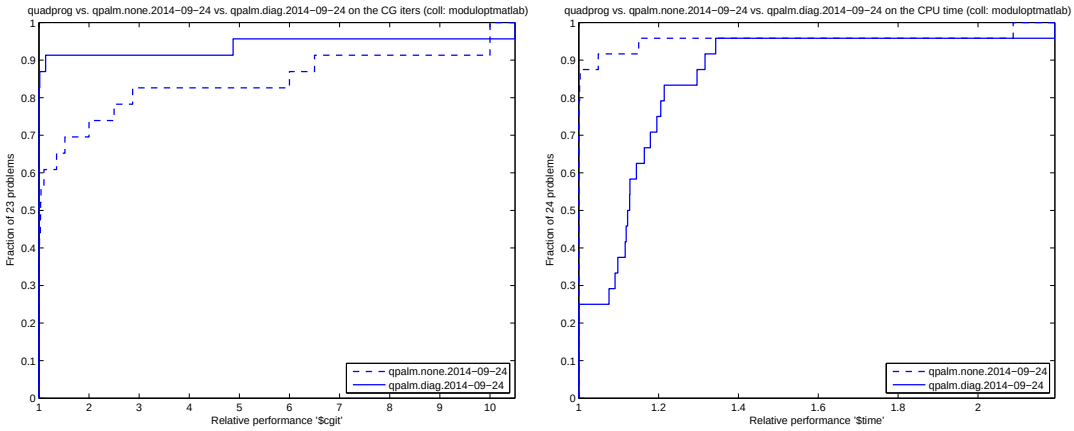


Figure 4.1: Performance profiles comparing `Qpalm` without (dashed curve) and with diagonal (plain curve) preconditioner on the number of CG iterations (left) and CPU time (right)

and with diagonal preconditioner (plain curve) on the number of CG iterations (left) and CPU time (right). If the preconditioner decreases significantly the number of CG iterations, it also deteriorates a little the CPU time because of the additional computation it requires.

Compare

- the active set strategy, which activates any hit constraint and restart CG iterations on the new activated face, and

- the explore-outside strategy, in which the cost function is minimized on the activated face, even outside the constraint box, until a stopping criterion is satisfied, and then a piecewise linesearch is done on the projected path.

Plot the number of iteration and the final r with respect to `options.dcr`. Show that the solver enters into trouble when r becomes large (stop on `dxmin`?).

Plot the speed of convergence as a function of r.

4.3 Comparison with other solvers

The QP solvers used in the experiments are the following:

- `Ooqp` (`C++`, interior point),
- `Oqla` (`C++`, augmented Lagrangian, active set, gradient projection),
- `Qpa` (`Fortran`, active set),
- `Qpb` (`Fortran`, interior point),
- `Qpalm` (`Matlab`, same approach as `Oqla`),
- Quadprog (`Matlab`, with the options `'Algorithm'` set to `'active-set'`, `'LargeScale'` set to `'off'`, `'TolFun'` set to `1.e-8`, `'TolX'` set to `1.e-8`, `'TolCon'` set to `1.e-8`, `'MaxIter'` set to `10000`); the option `'LargeScale'` must be set ot `'off'`, otherwise Quadprog switches to the `'trust-region-reflective'` algorithm (note that `Qpalm` can use sparse data structure).

Specify the stopping criterion of all the solvers.

In order to compare solvers with different convergence tests, we follow the approach described by Dolan, Moré, and Munson [15] and choose as common stopping criteria, the following conditions:

$$\operatorname{dist}_{[l_B,u_B]}(x) \leqslant \varepsilon_B, \tag{4.1a}$$

$$\operatorname{dist}_{[l_I,u_I]}(A_I x) \leqslant \varepsilon_I, \tag{4.1b}$$

$$\|A_E x - b_E\| \leqslant \varepsilon_E, \tag{4.1c}$$

$$G = \nabla_{(x,y)} l_0(x,y,\lambda) = \begin{pmatrix} g + Hx + A_I^{\mathsf{T}}\lambda_I + A_E^{\mathsf{T}}\lambda_E \\ -\lambda_I \end{pmatrix} \tag{4.1d}$$

$$\left(\mathrm{P}_{\mathrm{T}_{(x,y)}[l,u]}\ G\right)_i = \begin{cases} \min(0, G_i) & \text{if} \quad l_i = (x,y)_i \\ G_i & \text{if} \quad l_i < (x,y)_i < u_i \\ \max(0, G_i) & \text{if} \quad (x,y)_i = u_i \end{cases} \tag{4.1e}$$

In these conditions, $d_P(z)$ denotes the Euclidean distance from a point z to the set P, while $\varepsilon_B > 0$, $\varepsilon_I > 0$, and $\varepsilon_E > 0$ are tolerances that will be specified below.

Compare the solvers at low and high accuracy.

Compare `Oqla` with active set methods and interior point methods. For the comparison with IP methods, consider the case when the initial working set is correct at 50, 70, and 90 %.

4.3.1 Comparison between `Oqla` and `Qpalm`

To offer a comparison between Quadprog and `Qpalm`, that focuses on the algorithm rather the implementation, we have launched `Qpalm` on dense data (recall that Quadprog-activeset does not deal with sparse data).

4.3.2 Comparison with active set methods

Let us start by comparing the `Matlab` version of `Oqla`, called `Qpalm`, with the standard `Matlab` solver `Quadprog`. The comparison with `Quadprog` is made with `Qpalm` for two reasons. First the comparison can only be made on the computing time, since the methods used in the two solvers are quite different. Second, a comparison between `Oqla` and `Quadprog` favors to much the former, because it is written in `C++`, not in `Matlab` like `Quadprog` and `Qpalm`. The performance profiles for the CPU time are given in figure 4.2. Only the problems with less than 500 variables have been selected, which explains their

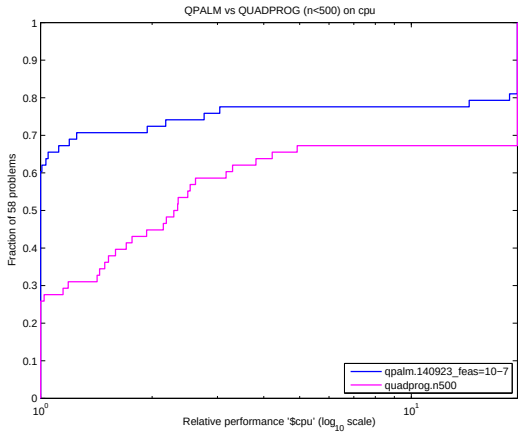


Figure 4.2: Performance profiles comparing `Qpalm` (blue) and `Quadprog`-active-set on the CPU time (coll: Cutest)

numbers (58 among ...). `Qpalm` appears to be more robust and faster. With the given options, `Quadprog` cannot deal with sparse data, which explains sometimes the huge computing time.

Figure 4.3 compares the computing time required by `Qpalm` (blue) and `Quadprog` (magenta) when they work on sparse (plain curves) and full (dashed curves) data on the solvable problems in the collection ModultoptMatlab. It can be seen that `Qpalm` can take advantage on the sparsity of the data, while this is not the case for `Quadprog` (actually, this cannot be deduced from these performance profiles but a comparison of the two versions of Quadprog confirms that one version is never worsens the other by more than 13 %, which is within the observed variations between runs). One also observe that `Qpalm` is 35 % more robust than `Quadprog` (`Qpalm` solves all solvable problems but one, while `Quadprog` solves only 60 % of the solvable problems) and usually faster (some times 80 times).

4.3.3 Comparison with interior point methods

Comparison 1. Compare the IP solvers with OQLA/QPALM, all starting from the default unknown starting points.

Comparison 2. To show that active set methods can be interesting in an SQP framework, we compare the IP solvers with OQLA/QPALM, all starting from various perturbations

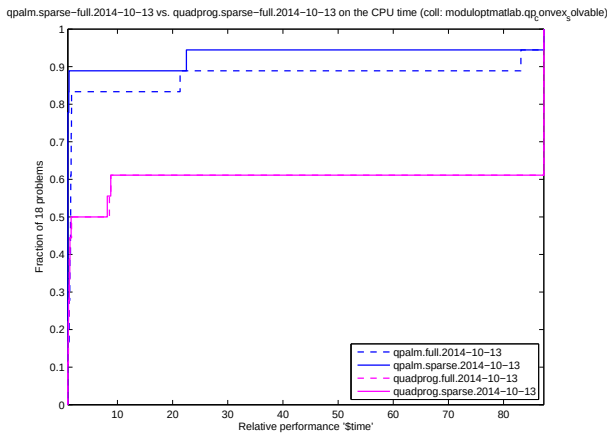


Figure 4.3: `Qpalm` (blue) and `Quadprog` (magenta) when they work on sparse (plain curves) and full (dashed curves) data (coll: ModultoptMatlab, solvable problems

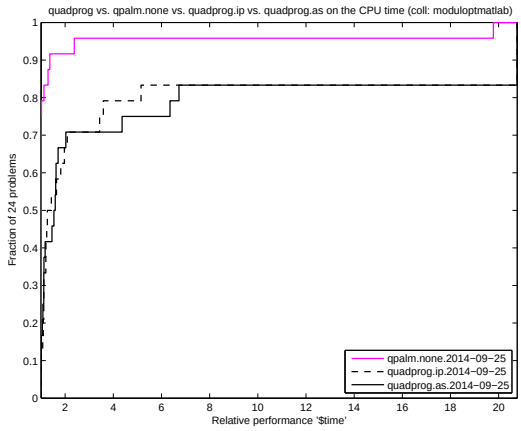


Figure 4.4: Performance profiles comparing `Qpalm` (magenta), `Quadprog`-active-set (plain black), and `Quadprog`-interior-point-convex (dashed black) on the CPU time (coll: ModuloptMatlab)

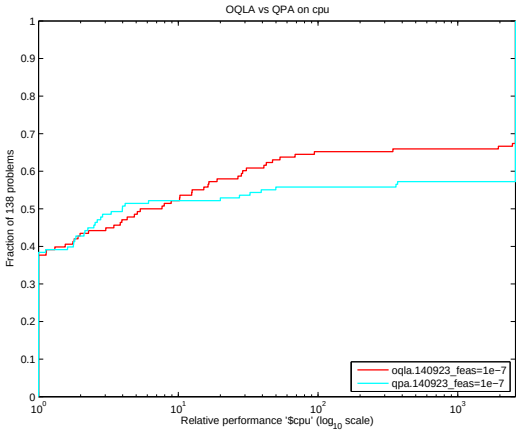


Figure 4.5: Performance profiles comparing `Oqla` (red) and `Qpa` on the CPU time (coll: Cutest)

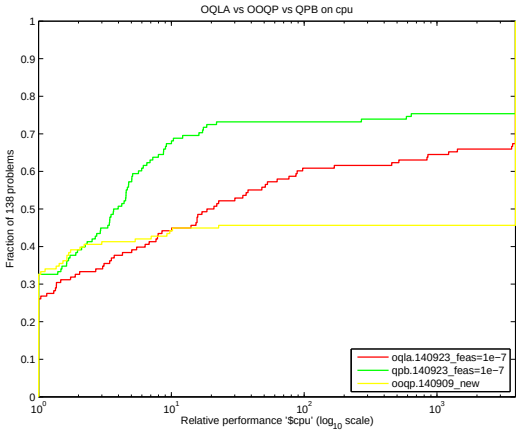


Figure 4.6: Performance profiles comparing `Ooqp`, `Oqla` (red), and `Qpb` on the CPU time

of a primal-dual solution to the QP. The logic of the setting of these experiments is the following. In an SQP framework, the primal-dual variables (x_k, λ_k) of the SQP algorithm are updated according to

$$x_{k+1} := x_k + d_k \qquad \text{and} \qquad \lambda_{k+1} := \lambda_k^{\text{QP}},$$

where $(d_k, \lambda_k^{\text{QP}})$ is the primal-dual solution to the quadratic problem solved at iteration k. In case of convergence of (x_k, λ_k) to some (x_*, λ_*),

$$(d_k, \lambda_k^{\text{QP}}) = (x_{k+1} - x_k, \lambda_{k+1}) \simeq (0, \lambda_k),$$

so that it makes sense to start the QP solver at iteration k with the primal-dual variables set to $(0, \lambda_k)$. To assess the performance of OQLA/QPALM and IP solvers in such circumstance, we have launched them from a primal point that is close to the solution (this is equivalent to launching them from zero for problems having a primal solution close to zero) and a dual point close to the dual solution.

5 Discussion

The salient features of the AL algorithm just described, which specify the contour of its application niche, are

- it does not require any matrix factorization, so that it can be used for large scale problems,
- it can provide precious information when the considered problem is infeasible or unbounded; this information can be useful when the QP solver is viewed as a tool in nonlinear optimization:
  - in case the QP is unbounded, it provides a direction of unboundedness, which is interesting for the SQP method,
  - in case the QP is infeasible, it returns a solution to the closest feasible problem, that is the problem with the same objective, but with constraints that are shifted by the smallest possible shift (in the Euclidean norm) that make them feasible (the theory is given in the join paper - I think that Manlio was interested by that paper a few years ago, but the paper has only be completed recently, so that I did not sent it to him at the time I presented it in Erice); this is also interesting for the SQP algorithm,
- the AL subproblems are currently solved by an active-set-gradient-projection algorithm, so that it can take advantage of an initial primal-dual pair that is not too far from a primal-dual solution; this is also interesting on the SQP algorithm.

Acknowledgments

References

[1] M. Aganagić (1984). Newton's method for linear complementarity problems. *Mathematical Programming*, 28, 349–362. [doi]. 2

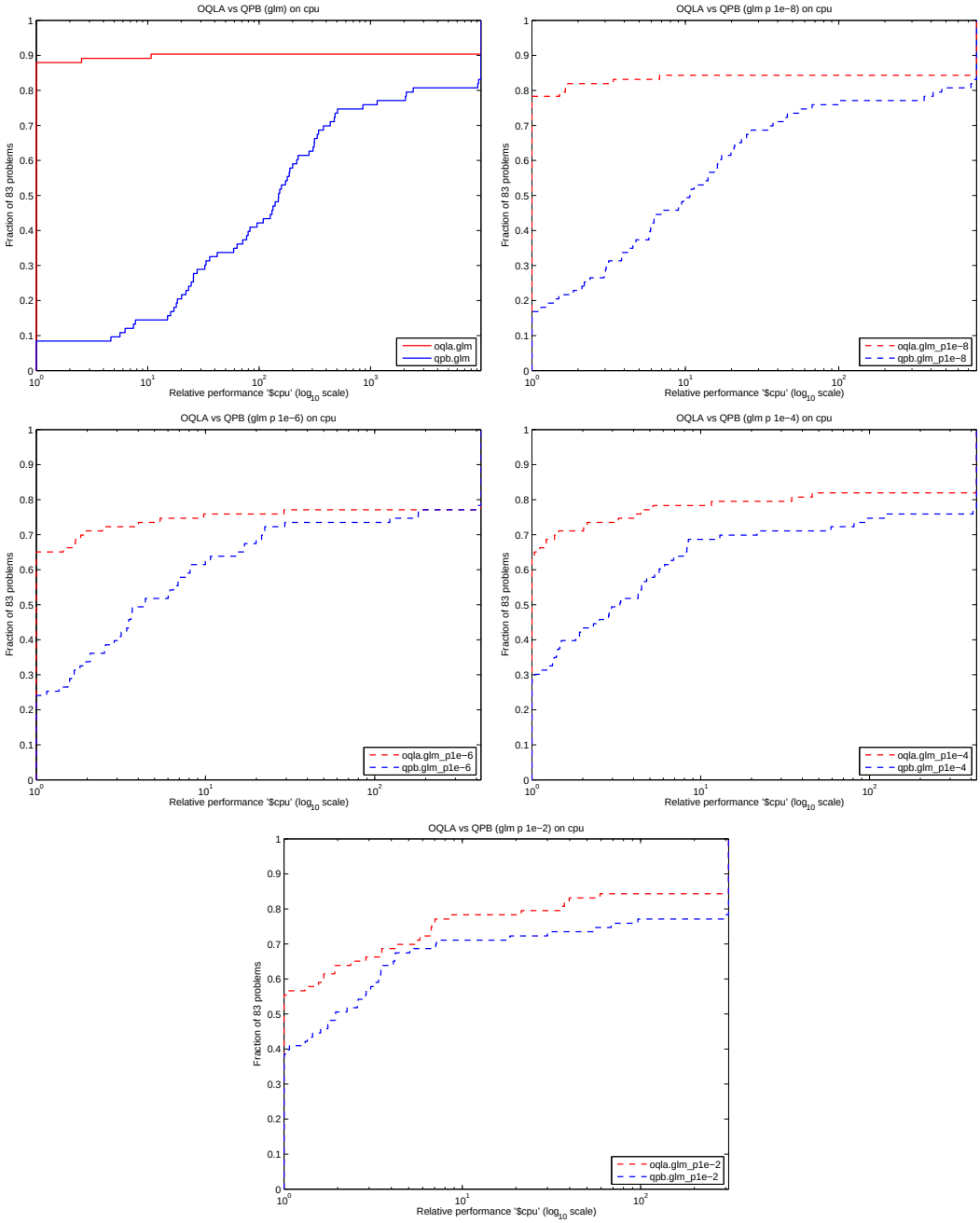


Figure 4.7: Performance profiles comparing `Oqla` (red), and `Qpb` (blue) on the CPU time, when the solvers start from a primal-dual solution modified by a random perturbation whose relative size is 0 (no perturbation), 10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}

[2] K.J. Arrow, F.J. Gould, S.M. Howe (1973). A generalized saddle point result for constrained optimization. *Mathematical Programming*, 5, 225–234. [doi]. 6

[3] Ch. Audet, D. Orban (2006). Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal on Optimization*, 17, 642–664. [doi]. 15

[4] I. Ben Gharbia, J.Ch. Gilbert (2012). Nonconvergence of the plain Newton-min algorithm for linear complementarity problems with a P-matrix. *Mathematical Programming*, 134, 349–364. [doi]. 2

[5] I. Ben Gharbia, J.Ch. Gilbert (2013). An algorithmic characterization of P-matricity. *SIAM Journal on Matrix Analysis and Applications*, 34, 904–916. [doi]. 2

[6] E.G. Birgin, J.M. Martínez (2014). *Practical Augmented Lagrangian Methods for Constrained Optimization*. SIAM Publication, Philadelphia. [doi]. 6, 10

[7] J.F. Bonnans, J.Ch. Gilbert, C. Lemaréchal, C. Sagastizábal (2006). *Numerical Optimization – Theoretical and Practical Aspects* (second edition). Universitext. Springer Verlag, Berlin. [authors] [editor] [google books]. 1, 14

[8] J.D. Buys (1972). *Dual algorithms for constrained optimization*. PhD Thesis, Rijksuniversiteit te Leiden, Leiden, The Netherlands. 6

[9] R.H. Byrd, J.Ch. Gilbert, J. Nocedal (2000). A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89, 149–185. [doi]. 2

[10] A. Chiche, J.Ch. Gilbert (2015). How the augmented Lagrangian algorithm can deal with an infeasible convex quadratic optimization problem. *Journal of Convex Analysis* (to appear), 22(4). [pdf]. 2, 4, 6, 7, 8, 9, 11

[11] A.R. Conn, N.I.M. Gould, Ph.L. Toint (1992). *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Computational Mathematics 17. Springer Verlag, Berlin. 6

[12] F.E. Curtis, Z. Han, D.P. Robinson (2014). A globally convergent primal-dual active-set framework for large-scale convex quadratic optimization. *Computational Optimization and Applications* (to appear). [doi]. 2

[13] E.J. Dean, R. Glowinski (2006). An augmented Lagrangian approach to the numerical solution of the Dirichlet problem for the elliptic Monge-Ampère equation in two dimensions. *Electronic Transactions on Numerical Analysis*, 22, 71–96. [pdf]. 6

[14] F. Delbos, J.Ch. Gilbert (2005). Global linear convergence of an augmented Lagrangian algorithm for solving convex quadratic optimization problems. *Journal of Convex Analysis*, 12, 45–69. [preprint] [editor]. 2, 6, 9

[15] E.D. Dolan, J.J. Moré, T.S. Munson (2006). Optimality measures for performance profiles. *SIAM Journal on Optimization*, 16, 891–909. [doi]. 16

[16] Z. Dostál (2009). *Optimal Quadratic Programming Algorithms*, volume 23 of *Springer Optimization and Its Applications*. Springer. 2

[17] Z. Dostál, A. Friedlander, S.A. Santos (1999). Augmented Lagrangians with adaptive precision control for quadratic programming with equality constraints. *Computational Optimization and Applications*, 14, 37–53. [doi]. 6

[18] Z. Dostál, A. Friedlander, S.A. Santos (2003). Augmented Lagrangians with adaptive precision control for quadratic programming with simple bounds and equality constraints. *SIAM Journal on Optimization*, 13, 1120–1140. [doi]. 6

[19] Z. Dostál, A. Friedlander, S.A. Santos, K. Alesawi (2000). Augmented Lagrangians with adaptive precision control for quadratic programming with equality constraints: corrigendum and addendum. *Computational Optimization and Applications*, 23, 127–133. [doi]. 6

[20] M. Frank, P. Wolfe (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3, 95–110. [doi]. 4, 6

[21] M. Friedlander, S. Leyffer (2008). Global and finite termination of a two-phase augmented Lagrangian filter method for general quadratic programs. *SIAM Journal on Scientific Computing*, 30, 1706–1726. [doi]. 6

[22] E.M. Gertz, S. Wright (2003). Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29, 58–81. [doi]. 2

[23] J.Ch. Gilbert, É. Joannopoulos (2014). Inside OQLA and QPALM. Technical report, INRIA, BP 105, 78153 Le Chesnay, France. (to appear). 3, 8, 14

[24] J.Ch. Gilbert, C. Lemaréchal (1989). Some numerical experiments with variable-storage quasi-Newton algorithms. *Mathematical Programming*, 45, 407–435. [doi]. 2

[25] P.E. Gill, W. Murray, M.H. Wright (1981). *Practical Optimization*. Academic Press, New York. 2

[26] N.I.M. Gould, D. Orban, Ph.L. Toint (2003). GALAHAD: a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. Technical report, Rutherford Appleton Laboratory, Oxfordshire OX11 0QX. [internet]. 2

[27] N.I.M. Gould, D. Orban, Ph.L. Toint (2013). Cutest: a constrained and unconstrained testing environment with safe threads. Technical report, Rutherford Appleton Laboratory, Didcot, Oxfordshire OX11 0QX. 2

[28] O. Güler (1992). Augmented Lagrangian algorithms for linear programming. *Journal of Optimization Theory and Applications*, 75, 445–470. [doi]. 6

[29] D. Henrion, J. Malick (2009). Projection methods for conic feasibility problems: applications to polynomial sum-of-squares decompositions. *Optimization Methods and Software*, 26, 23–46. [doi]. 6

[30] M.R. Hestenes (1969). Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4, 303–320. 6

[31] J.-B. Hiriart-Urruty, C. Lemaréchal (1993). *Convex Analysis and Minimization Algorithms*. Grundlehren der mathematischen Wissenschaften 305-306. Springer-Verlag. 3, 13

[32] K. Ito, K. Kunisch (2008). *Lagrange Multiplier Approach to Variational Problems and Applications*. SIAM. 2

[33] A.F. Izmailov, M.V. Solodov (2014). *Newton-Type Methods for Optimization and Variational Problems*. Springer Series in Operations Research and Financial Engineering. Springer. 1

[34] D.C. Liu, J. Nocedal (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45, 503–520. 2

[35] J. Malick (2004). A dual approach to semidefinite least-squares problems. *SIAM Journal on Matrix Analysis and Applications*, 26, 272–284. [doi]. 6

[36] J. Malick, J. Povh, F. Rendl, A. Wiegele (2009). Regularization methods for semidefinite programming. *SIAM Journal on Optimization*, 20, 336–356. [doi]. 6

[37] J.J. Moré, G. Toraldo (1991). On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, 1, 93–113. [doi]. 2, 12, 14

[38] J.-J. Moreau (1965). Proximité et dualité dans un espace hilbertien. *Bulletin de la Société Mathématique de France*, 93, 273–299. [url]. 5

[39] J. Nocedal (1980). Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35, 773–782. 2

[40] J. Nocedal, S.J. Wright (2006). *Numerical Optimization* (second edition). Springer Series in Operations Research. Springer, New York. 2, 14

[41] B.T. Poljak, N.V. Tret'jakov (1972). A certain iteration method of linear programming and its economic interpretationin russian. *Èkonomika i Matematicheskie Metody*, 8, 740–751. 6

[42] M.J.D. Powell (1969). A method for nonlinear constraints in minimization problems. In R. Fletcher (editor), *Optimization*, pages 283–298. Academic Press, London. 6

[43] R.T. Rockafellar (1971). New applications of duality in convex programming. In *Proceedings of the 4th Conference of Probability, Brasov, Romania*, pages 73–81. 6

[44] R.T. Rockafellar (1973). A dual approach to solving nonlinear programming problems by unconstrained optimization. *Mathematical Programming*, 5, 354–373. [doi]. 8

[45] R.T. Rockafellar (1973). The multiplier method of Hestenes and Powell applied to convex programming. *Journal of Optimization Theory and Applications*, 12, 555–562. [doi]. 6

[46] R.T. Rockafellar (1974). Augmented Lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control*, 12, 268–285. 6

[47] QUADPROG (2014). Quadratic programming. [internet]. 2

[48] A. Shapiro, J. Sun (2004). Some properties of the augmented Lagrangian in cone constrained optimization. *Mathematics of Operations Research*, 29, 479–491. [doi]. 6

[49] M. Ulbrich (2011). *Semismooth Newton Methods for Variational Inequalities and Constrained Optimization Problems in Function Spaces*. MPS-SIAM Series on Optimization 11. SIAM Publications, Philadelphia, PA, USA. [doi]. 2

[50] Z. Wen, D. Goldfarb, W. Yin (2010). Alternating direction augmented Lagrangian methods for semidefinite programming. *Mathematical Programming Computation*, 2, 203–230. [doi]. 6

[51] S.J. Wright (1997). *Primal-Dual Interior-Point Methods*. SIAM Publication, Philadelphia. 2

Index