

## RESEARCH ARTICLE

### Benchmarking Large Scale Distributed Convex Quadratic Programming Algorithms

Attila Kozma<sup>a\*</sup>, Christian Conte<sup>b</sup> and Moritz Diehl<sup>a</sup>

<sup>a</sup>Department of Electrical Engineering, KU Leuven, Leuven, Belgium;

<sup>b</sup>Automatic Control Laboratory, ETH Zürich, Zürich, Switzerland

(Received 00 Month 200x; in final form 00 Month 200x)

This paper aims for collecting, benchmarking and implementing state-of-the-art decomposable convex quadratic programming methods employing duality. In order to decouple the original problem, these methods relax some constraints by introducing dual variables and apply a hierarchical optimization scheme. In the lower level of this scheme, a sequence of parametric quadratic programs is solved in parallel, while in the high-level problem a gradient based method is applied to achieve an optimal dual solution. Finding the optimal dual variables is a hard problem since the dual function is not twice continuously differentiable and not strongly convex. We investigate and compare several gradient based methods using a benchmark set of convex quadratic programming problems. We discuss the theoretical worst case convergence properties of the investigated methods, but we also evaluate their practical convergence behaviour. The benchmark set as well as the suite of implemented algorithms are released as open-source software. From our experiments it turns out that the alternating direction method of multipliers (ADMM) and the restarted version of the fast gradient method are the best methods for solving decomposable QPs in terms of the number of necessary lower level QP solutions.

**Keywords:** large scale optimization, quadratic programming, distributed optimization, dual decomposition

**AMS Subject Classification:** 49M27, 90C06, 90C20

## 1. Introduction

Large scale convex quadratic programming (QP) problems arise in many different areas such as optimal control, machine learning or telecommunication. Often they occur as a subproblem of solving even harder problems, e.g. nonlinear or mixed-integer programs. Thus, it is of vivid interest to develop efficient methods for this problem class. Large scale convex QPs possess a well-defined sparsity structure, thus the well established dense linear algebra decomposition methods cannot be employed to solve such problems. Instead, their sparse variant should be used. There are several methods proposed in the literature to solve large scale convex QPs centrally. For instance, in [25] and [31] active set methods are proposed with block LU-updates on the KKT matrix and with hot-starting capabilities. In [20], an interior-point method is presented for the same problem class. CPLEX from [26] implements both popular strategies with thread-based parallel linear algebra. These methods are unarguably efficient, but they all assume that the problem data fits into the shared memory of a computer. This assumption does not necessarily hold: it may be the case that the problem data is not available in a centralized form, but in portions, locally i.e. as a decoupled QP. For instance, one may consider control problems of complex networks such as electricity grids or search engine problems. For such applications, the centralized solution is not a possible option. In

---

\*Corresponding author. E-mail: attila.kozma@esat.kuleuven.be

distributed optimization, local decisions with minimal coordination and communication effort are preferred together with guaranteed convergence properties. For instance, in [22], the author redesigns an interior-point method such that no forming of Hessians or Jacobians is needed. Instead, it necessitates only matrix-vector products, which can be calculated locally.

In most classical distributed optimization methods (for a good introduction see [5]) convexity and duality are heavily exploited. In this paper, we restrict our attention to this class of methods. Most existing distributed approaches decouple the original problem by introducing some dual variables to relax coupled constraints. This scheme, which is applicable to general convex decomposable problems, is often regarded as *Lagrangian relaxation* [30] or *dual decomposition*. The price of the decoupling is that one has to find the optimal dual variables, which turns out to be a hard problem due to the non-smoothness of the dual function.

We investigate methods that make use of only gradient information, since these methods by construction are parallelizable or may be even implemented in a distributed fashion. We are concerned with several aspects of the discussed methods. It is of our interest how these approaches behave in practice and to what extent this differs from the theoretical worst-case complexity results. We discuss the communication requirements of different methods along with the level of possible parallelization and distribution. Moreover, we establish a benchmark suite of large scale convex QP problems to provide test problems for researchers of future distributed algorithms.

The rest of this paper proceeds as follows. In Section 2, the optimization problem of our interest is stated and the concept of dual decomposition is given. In Section 3, the special structure of the considered optimization problem and the communication requirements are discussed. In Sections 4 and 5 several commonly used and state-of-the-art distributed optimization methods are discussed, which use only gradients of the dual function. Section 4 is devoted to methods for strongly convex problems, while Section 5 presents methods for non-strongly convex problems. We present the benchmark suite of large decoupled quadratic programming problems in Section 6. The practical comparisons of the different methods are presented in Section 7. Some details regarding the software implementation are given in Section 8. The paper concludes in Section 9.

Throughout this paper,  $\mathbb{R}^n$  denotes the  $n$ -dimensional vector space of real numbers,  $H \succeq 0$  ( $H \succ 0$ ) means that matrix  $H$  is positive semidefinite (positive definite). The  $\text{col}(\cdot)$  ( $\text{row}(\cdot)$ ) operator concatenates its arguments in a column (row), the set  $\text{span}(x_1, \dots, x_n)$  denotes the vector space spanned by  $x_1, \dots, x_n$ . A variable or matrix with subscript, e.g.  $A_i$ ,  $x_i$  indicates that it belongs to a subproblem  $i$ , while a superscript in brackets, e.g.  $x^{(k)}$  indicates that it belongs to the  $k$ -th iterate of an optimization method.

## 2. Problem statement and dual decomposition

In this paper, we regard a decoupled convex quadratic program (QP) of the form

$$\min_x \sum_{i=1}^n \frac{1}{2} x_i^T H_i x_i + c_i^T x_i \quad (1a)$$

$$\text{s.t.} \quad \sum_{i=1}^n A_i x_i = b \quad (1b)$$

$$x_i \in \mathcal{X}_i, \quad i = 1, \dots, n, \quad (1c)$$

where  $x_i \in \mathbb{R}^{m_i}$ ,  $x = \text{col}(x_1, \dots, x_n) \in \mathbb{R}^m$  and for all  $i \in \{1, \dots, n\}$ ,  $A_i \in \mathbb{R}^{p \times m_i}$  holds. Furthermore, for all  $i \in \{1, \dots, n\}$ ,  $H_i \succeq 0$ , i.e.  $H_i$  are positive semi-definite, the sets  $\mathcal{X}_i = \{x_i \in \mathbb{R}^{m_i} | C_i x_i \leq d_i\}$  are polytopic constraints. We define  $\mathcal{X} = \mathcal{X}_1 \times \dots \times \mathcal{X}_n \subseteq \mathbb{R}^m$ ,  $A = (A_1 | \dots | A_n)$  and  $m = \sum_{i=1}^n m_i$ . Note that the cost function (1a) as well as the inequality constraint (1c) is decoupled and the different variables  $x_i$  are coupled via (1b). A non fully decoupled quadratic program might be transformed to this form by introducing new variables that are made equal by (1b).

### 2.1. Dual decomposition methods using only gradients

Dual decomposition has already been applied successfully in several areas, such as in optimal control [3, 14, 33, 40], in estimation problems [41] or for problems from telecommunications [8, 45]. It introduces a two-level optimization problem in order to decouple problem (1). The low-level optimization problems, which provide gradients to the high-level, are solved in parallel, while in the high-level problem the updates of the dual variables are carried out and propagated to the low-level. We consider several optimization methods for the high-level problem that make use of only gradients and give their convergence properties. We describe how the methods relate to each other, explain their parallel or distributed nature and clarify the communication requirements.

In the sequel, we restrict our attention to a subset of problem (1), namely where for all  $i \in \{1, \dots, n\}$  the matrix  $H_i \succ 0$ , i.e.  $H_i$  is positive definite. The methods considered here rely on the dualization of constraint (1b), which results in the (partial) *Lagrangian function* defined by

$$L(x, \lambda) := \sum_{i=1}^n L_i(x_i, \lambda) := \sum_{i=1}^n \left( \frac{1}{2} x_i^T H_i x_i + c_i^T x_i \right) + \lambda^T \left( \sum_{i=1}^n A_i x_i - b \right), \quad (2)$$

where  $\lambda \in \mathbb{R}^p$  are the dual variables corresponding to (1b). Using strong duality one can introduce the *Lagrange dual function*  $d(\lambda)$  as

$$d(\lambda) = \min_{x \in \mathcal{X}} L(x, \lambda) = \sum_{i=1}^n \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda) \quad (3)$$

and the unconstrained dual concave optimization problem as

$$\max_{\lambda} d(\lambda). \quad (4)$$

Note that the evaluation of  $d(\lambda)$  may be divided into solving  $n$  independent parametric QPs of form

$$\min_{x_i} \frac{1}{2} x_i^T H_i x_i + (c_i^T + \lambda^T A_i) x_i - \lambda^T \frac{b}{n}, \quad (5)$$

which provides space for parallelized or even distributed methods. For a detailed description for general convex functions see [5, page 229]. It can be shown that  $d(\lambda)$  is concave once  $H_i \succ 0$  holds and it is once continuously differentiable with  $\nabla d(\lambda) = Ax(\lambda)^* - b$ , where  $x^*(\lambda) = \arg \min_x L(x, \lambda)$  [5, pp. 669]. Moreover, it can be shown that  $\nabla d(\lambda)$  is Lipschitz continuous with constant  $L$ , i.e. there exists  $L \in \mathbb{R}$  such that for all  $\lambda_1, \lambda_2 \in \mathbb{R}^p$ ,

$$\|\nabla d(\lambda_1) - \nabla d(\lambda_2)\| \leq L \|\lambda_1 - \lambda_2\|. \quad (6)$$

This implies that the (concave) dual function may be bounded below by a quadratic function having curvature  $L$ . However, in certain points the dual function has zero curvature. A function  $f$  is strongly convex with convexity parameter  $\mu > 0$  if for all  $x, y$

$$(\nabla f(x) - \nabla f(y))^T (x - y) \geq \mu \|x - y\|_2^2 \quad (7)$$

holds. In other words, strongly convex functions can be bounded from below by a quadratic function. Since the dual function  $d(\lambda)$  has zero curvature in certain directions,  $-d(\lambda)$  is not strongly convex.

Since the second derivative of  $d(\lambda)$  is only defined piece-wise, only methods that employ first order derivatives are considered here. However, we note that also second-order derivatives may be used in a dual decomposition framework as in [15] requiring sparse linear algebra operations. Our work is related to [34], in which the authors present a decomposition method with a smoothing technique for general convex problems in order to improve the properties of  $d(\lambda)$ . The approach published in [36] uses a smoothing technique as well and thus is a possible candidate for finding an optimal dual solution for general convex decomposable problems. Furthermore, following the same idea in [43] the authors make use of smoothing in both the primal and the dual problems.

We shortly summarize the most important convergence rates. Assume that  $\lambda^{(k)} \rightarrow \lambda^*$  as  $k \rightarrow \infty$ . The series  $\lambda^{(k)}$  converges

- sublinearly if

$$\|\lambda^{(k)} - \lambda^*\| \leq c \frac{1}{k^\alpha}, \text{ with } \alpha \in (0, \infty], c \in \mathbb{R} \quad (8)$$

- linearly if

$$\|\lambda^{(k)} - \lambda^*\| \leq c\alpha^k, \text{ with } \alpha \in [0, 1), c \in \mathbb{R} \quad (9)$$

- superlinearly if

$$\|\lambda^{(k)} - \lambda^*\| \leq c\alpha_{(k)}^k, \text{ with } \alpha_{(k)} \rightarrow 0, c \in \mathbb{R} \quad (10)$$

- quadratically if

$$\|\lambda^{(k)} - \lambda^*\| \leq c\alpha^{2^k}, \text{ with } \alpha \in [0, 1), c \in \mathbb{R}. \quad (11)$$

From an optimization point of view, the sublinear convergence rate means very poor performance, the linear convergence rate has sometimes acceptable performance depending on the value of  $\alpha$ . The superlinear and quadratic convergence rates are desirable.

The methods discussed in our paper are based upon the general scheme

$$x^{(k)} = \arg \min_{x \in \mathcal{X}} L(x, \lambda^{(k)}) \quad (12a)$$

$$\lambda^{(k+1)} = \lambda^{(k)} + \mathcal{G} \left( \nabla d(\lambda^{(0)}), \dots, \nabla d(\lambda^{(k)}) \right), \quad (12b)$$

where  $\mathcal{G} : \mathbb{R}^{p \times (k+1)} \rightarrow \mathbb{R}^p$ . In (12a), we have to solve  $n$  parametric quadratic programs in parallel, which provide gradient information to the dual space. In (12b), we have to calculate the next iterate in the dual variable  $\lambda$  by mapping the previous gradients to a correction term.

There exists an important theoretical result from Nesterov [35, page 61], who showed that if a method implements  $\mathcal{G}$  as a linear combination operator, i.e.

$\mathcal{G}(\nabla d(\lambda^{(0)}), \dots, \nabla d(\lambda^{(k)})) \in \text{span}(\nabla d(\lambda^{(0)}), \dots, \nabla d(\lambda^{(k)}))$  then the convergence cannot be fast in the beginning. More precisely, for any method following a scheme

$$\lambda^{(k+1)} \in \lambda^{(k)} + \text{span}(\nabla f(\lambda^{(0)}), \dots, \nabla f(\lambda^{(k)})), \quad (13)$$

and for any  $k : 1 \leq k \leq \frac{1}{2}(p-1)$  and  $\lambda^{(0)}$  there exists a concave, differentiable function  $f(\lambda) : \mathbb{R}^p \rightarrow \mathbb{R}$  with  $L$ -Lipschitz continuous gradient such that

$$f^* - f(\lambda^{(k)}) \geq \frac{3L\|\lambda^{(0)} - \lambda^*\|}{32(k+1)^2}, \quad (14a)$$

$$\|\lambda^{(k)} - \lambda^*\|^2 \geq \frac{1}{8}\|\lambda^{(0)} - \lambda^*\|^2. \quad (14b)$$

In other words, the existence of function  $f(\cdot)$  renders the worst-case convergence rate of any first-order method sublinear in the beginning. However, this does not mean that with practical applications one might not observe faster convergence. We will see that different realizations of (12b) lead to different convergence behaviour. Our aim is to summarize the theoretical properties of several methods and compare their practical performance on different problems.

### 3. Structure of constraint matrix $A$ and communication requirements

In this section, we discuss the particular structure of the constraint matrix  $A = [A_1 | \dots | A_n]$  of (1) in order to establish the basics of the required communication between subproblems. Throughout this paper we make the following assumption on the sparsity structure of  $A_i$ . For all  $i = 1, \dots, n$ ,

- most of the rows of  $A_i$  are zeros,
- if row  $j$  of  $A_i$  has non-zero elements then there exists  $k$  such that row  $j$  of  $A_k$  has non-zero elements.

The coupling between subproblems are very typical and determine the sparsity pattern of  $A$ . Subproblems can share one or more variables. Or even a weighted sum of some variables of each subsystem can be represented as a constraint. For example, assuming that we have  $n = 3$  subproblems, and  $A = [A_1 | A_2 | A_3]$  is defined by

$$A_1 = \begin{bmatrix} \dots & 1 & 0 & 0 & \dots \\ \dots & 0 & 1 & 0 & \dots \\ \dots & 0 & 0 & 1 & \dots \end{bmatrix}, \quad A_2 = \begin{bmatrix} \dots & -1 & 0 & 0 & \dots \\ \dots & 0 & 0 & 0 & \dots \\ \dots & 0 & 0 & 1 & \dots \end{bmatrix}, \quad A_3 = \begin{bmatrix} \dots & 0 & 0 & 0 & \dots \\ \dots & 0 & 0 & -1 & \dots \\ \dots & 0 & 0 & 1 & \dots \end{bmatrix}, \quad (15)$$

then the dimension of the dual space is 3. The first row of  $A$  ensures that subproblem 1 and 2 share a variable. The second row ensures that subproblem 1 and 3 have a common value. While the third row sums up some variables of each subsystem.

It is important to understand that in general each dual variable correspond to a coupling equation in (1b). Thus, the optimal solution of the  $i$ -th subproblem in (5) depends only on those elements of  $\lambda$  which correspond to couplings that involve subproblem  $i$ , i.e.  $A_i$  has non-zero elements in the corresponding row. In other words, in subproblem  $i$  the knowledge of only those dual variables are necessary, which correspond to constraints that subproblem  $i$  contributes to. The same reasoning holds for the gradient of the dual function  $\nabla d(\lambda)$ . If  $\mathcal{G}(\cdot)$  in (12b) is a simple enough operation the update of the dual variables can be carried out by using only local communication of the corresponding gradient vector slices, e.g. see the gradient method later on. Otherwise, if  $\mathcal{G}(\cdot)$  is more

involved, global communication is required. This feature is always algorithm dependent and thus has to be discussed in a more specific context.

#### 4. Methods for strongly convex separable QPs

In this section, we present methods that employ the Lagrangian function directly. These schemes can cope only with strongly convex QPs, i.e. the original problem has to be positive definite. Non-strongly convex problems can be transformed into strongly convex ones by adding a small diagonal regularization to the Hessian matrix.

##### 4.1. Gradient method with fixed stepsize

The *gradient method*, also referred to as steepest descent method, is one of the oldest first order methods proposed first by Cauchy in 1847 [7]. It uses only the latest gradient to obtain a search direction.

$$\lambda^{(k+1)} := \lambda^{(k)} + t^{(k)} \nabla d(\lambda^{(k)}) \quad (16)$$

If we apply this method in (12b) it can be shown (see [35, pp. 69-70] for details) that the optimal stepsize is  $t^{(k)} = \frac{1}{L}$ .

---

**Algorithm 1:** Gradient method with fixed stepsize. (GM)

---

**Input** :  $\lambda^{(0)}, L$   
1  $k := 0$   
2 **while** no convergence **do**  
3   **foreach**  $i = 1, \dots, n$  **do**  
4      $x_i^{(k)} := \arg \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda^{(k)})$   
5   **end**  
6    $\nabla d(\lambda^{(k)}) := \left( \sum_{i=1}^n A_i x_i^{(k)} \right) - b$   
7    $\lambda^{(k+1)} := \lambda^{(k)} + \frac{1}{L} \nabla d(\lambda^{(k)})$   
8    $k := k + 1$   
9 **end**  
**Output:**  $\lambda^{(k)}, x^{(k)}$

---

**Convergence:** We can see that if the function  $d(\lambda)$  has high curvature, i.e. the Lipschitz constant  $L$  gets large, the method takes very short steps in Step 7, while if it has moderately flat curvature it takes longer steps. The convergence rate of the algorithm is sublinear [35, pp. 70] that is

$$d^* - d(\lambda^{(k)}) \leq \frac{2L \|\lambda^{(0)} - \lambda^*\|^2}{k + 4}. \quad (17)$$

Since this is not proportional to the lower bound given in (14a), this method is not optimal in the sense of Nesterov [35, p. 71].

**Communication:** Algorithm 1 can be implemented in a fully distributed fashion using only local communication. Indeed, assume that we want to update the dual variables

$\lambda^{(k)}$  in subproblem  $i$ . First, we notice that only a slice of  $\lambda^{(k)}$  is necessary in subproblem  $i$ , i.e. the elements that correspond to constraints to which subproblem  $i$  contributes. Other parts of  $\lambda^{(k)}$  can be excluded from the update, since they don't have an influence on the optimal solution of subproblem  $i$ . Second, in order to update  $\lambda^{(k)}$ , the necessary directions of the dual gradient function  $\nabla(d(\lambda))$  needs to be available and for this reason all contributions have to be collected from other subsystems that are coupled with subsystem  $i$ . In essence, in every iteration each subproblem sends its contribution to the dual gradient to other subsystems that are coupled to it and make the step locally. In this way, only local communication of vectors between coupled subproblems is necessary and no central coordination is needed. However, the Lipschitz constant  $L$  must be globally available a priori.

Note that calculating the Lipschitz constant  $L$  might be hard or even impossible in practice. This motivates us to use adaptive stepsizes or equivalently to approximate the Lipschitz constant based on local information. In the literature one can find many different approaches on how to do this (for a good summary see [48] and the references therein).

#### 4.2. Global Barzilai-Borwein method

We consider the adaptive gradient method from [2] that is often regarded as *Barzilai-Borwein update rule* and reads as

$$y^{(k)} := \nabla d(\lambda^{(k+1)}) - \nabla d(\lambda^{(k)}), \quad (18a)$$

$$s^{(k)} := \lambda^{(k+1)} - \lambda^{(k)}, \quad (18b)$$

$$t^{(k)} := \frac{(s^{(k)})^T s^{(k)}}{(s^{(k)})^T y^{(k)}}. \quad (18c)$$

A variant of the method for nonlinear convex functions was published in [39] and can be summarized as in Algorithm 2.

Note that in Algorithm 2, there is no explicit need for a Lipschitz constant and the stepsize is determined by a non-monotone line-search subprocedure, see the while loop in Line 14.

**Convergence:** In [2], the authors prove superlinear convergence for two-dimensional strongly convex quadratic functions. Moreover, R-linear convergence was shown for general strongly convex quadratics in [10]. The general convex quadratic case was investigated in [17].

**Communication:** As a consequence of the line-search procedure, the update of the dual variables has to be coordinated from a dedicated process, while the primal variables  $x_i$  are updated in parallel in the coordinated processes, see Lines 9 and 17. These processes compute their contribution to the dual objective and dual gradient functions, which have to be transmitted to the coordinator process, which updates the dual variables and propagates them to the coordinated processes. In our case, the price of calculating  $d(\lambda)$  and  $\nabla d(\lambda)$  is the same, namely solving  $n$  QPs in parallel.

**Algorithm 2:** Global Barzilai-Borwein method. (GBB)

---

**Input** :  $\lambda^{(0)}$ ,  $M \geq 0$ ,  $\gamma \in (0, 1)$ ,  $\delta > 0$ ,  $\sigma \in (0, 1)$ ,  $0 < \epsilon < 1$

```

1  $k := 0$ 
2  $\alpha_0 = 1$ 
3 while no convergence do
4   if  $\alpha_k \leq \epsilon$  or  $\alpha_k \geq \frac{1}{\epsilon}$  then
5      $\alpha_k := \delta$ 
6   end
7   if  $k == 0$  then
8     foreach  $i = 1, \dots, n$  do
9        $x_i^{(k)} := \arg \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda^{(k)})$ 
10    end
11     $\nabla d(\lambda^{(k)}) := \left( \sum_{i=1}^n A_i x_i^{(k)} \right) - b$ 
12  end
13   $t := \frac{1}{\alpha_k}$ 
14  while True do
15     $\lambda^{(k+1)} := \lambda^{(k)} + t \nabla d(\lambda^{(k)})$ 
16    foreach  $i = 1, \dots, n$  do
17       $x_i^{(k+1)} := \arg \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda^{(k+1)})$ 
18    end
19     $d(\lambda^{(k+1)}) := L(x^{(k+1)}, \lambda^{(k+1)})$ 
20    if  $-d(\lambda^{(k+1)}) \leq \max_{0 \leq j \leq \min(k, M)} (-d(\lambda^{(k-j)})) - \gamma t \|\nabla d(\lambda^{(k)})\|_2^2$  then
21      break
22    else
23       $t := \sigma t$ 
24    end
25  end
26   $\nabla d(\lambda^{(k+1)}) := \left( \sum_{i=1}^n A_i x_i^{(k+1)} \right) - b$ 
27   $y^{(k)} := \nabla d(\lambda^{(k+1)}) - \nabla d(\lambda^{(k)})$ 
28   $s^{(k)} := \lambda^{(k+1)} - \lambda^{(k)}$ ,
29   $\alpha_{k+1} := \frac{(s^{(k)})^T y_k}{\|s^{(k)}\|_2^2}$ 
30   $k := k + 1$ 
31 end
Output:  $\lambda^{(k)}$ ,  $x^{(k)}$ 

```

---

**4.3. Fast gradient method with fixed stepsize**

One can introduce another type of acceleration without a line-search procedure but fixed stepsize. The family of *fast gradient methods* was introduced in [36]. We describe a simplified optimal method for convex differentiable functions with  $L$ -Lipschitz gradient from [35, pp. 80] in Algorithm 3.

The fast gradient methods maintain a sequence  $\lambda_B^{(k)}$ , which is often referred to as the momentum term. Indeed, the momentum term tries to give an approximation of the next iterate by extrapolation.

**Convergence:** The convergence rate of Algorithm 3 is optimal, yet sublinear, i.e.

$$d^* - d(\lambda) \leq \frac{4L \|\lambda_0 - \lambda^*\|^2}{(k+2)^2}. \quad (19)$$

**Algorithm 3:** Fast gradient method with constant stepsize (FG)

---

**Input** :  $\lambda^{(0)}$ ,  $L$ ,  $\alpha_0 \in (0, 1)$

```

1  $\lambda_A^{(k)} := \lambda_B^{(k)} := \lambda^{(0)}$ 
2  $k := 0$ 
3 while no convergence do
4   foreach  $i = 1, \dots, n$  do
5      $x_i^{(k)} := \arg \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_B^{(k)})$ 
6   end
7    $\nabla d(\lambda_B^{(k)}) := \left( \sum_{i=1}^n A_i x_i^{(k)} \right) - b$ 
8    $\lambda_A^{(k+1)} := \lambda_B^{(k)} + \frac{1}{L} \nabla d(\lambda_B^{(k)})$ 
9   Compute  $\alpha_{k+1} \in (0, 1)$  from  $\alpha_{k+1}^2 = (1 - \alpha_{k+1})\alpha_k^2$ 
10   $\beta_k := \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2 + \alpha_{k+1}}$ 
11   $\lambda_B^{(k+1)} := \lambda_A^{(k+1)} + \beta_k \left( \lambda_A^{(k+1)} - \lambda_A^{(k)} \right)$ 
12   $k := k + 1$ 
13 end
Output:  $\lambda_A^{(k)}$ ,  $x^{(k)}$ 

```

---

The optimality of the method refers to the fact that one can give lower and upper estimates on  $d^* - d(\lambda^{(k)})$  that are proportional. In this case, they are both in the order of  $\frac{1}{k^2}$ .

**Communication:** One can implement Algorithm 3 in a fully distributed fashion, since for the update of dual variables only gradient vectors of the coupled subproblems are necessary. The same reasoning as for the gradient method applies here as well. Thus, local communication of vectors and knowledge of the Lipschitz constant system-wide is necessary.

#### 4.4. Fast gradient method with adaptive stepsize

Finding the tightest Lipschitz constant, or even an approximation might be again a difficult problem, thus we discuss an adaptive variant of Algorithm 3 from [27], with a line-search strategy from [4]. In Algorithm 4, the Lipschitz constant is adjusted in each iteration by a line-search procedure. A similar procedure is proposed in [37] in a different context. A Lipschitz constant that is too large may lead to very conservative steps, while a Lipschitz constant that is too small may lead to divergence of the algorithm.

This method does not require the knowledge of the Lipschitz constant  $L$ . Instead, an approximation  $L^{(k)} \leq L$  is maintained based upon local information. In the inner while loop from Line 8 on, we increase  $L^{(k)}$  repeatedly until it fulfills the Lipschitz property in points  $\lambda_B^{(k)}$ ,  $\lambda_A$ . In this approach, the decrease of  $L^{(k)}$  is not allowed and the convexity parameter is not approximated as it is known to be zero.

---

**Algorithm 4:** Fast gradient method with adaptive Lipschitz approximation (FG-AL)

---

**Input :**  $\lambda^{(0)}, L^{(k)}, \rho > 1, \alpha_0 \in (0, 1)$

```

1  $\lambda_A^{(0)} := \lambda_B^{(0)} := \lambda^{(0)}$ 
2  $k := 0$ 
3 while no convergence do
4   foreach  $i = 1, \dots, n$  do
5      $x_i^{(k)} := \arg \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_B^{(k)})$ 
6   end
7    $\nabla d(\lambda_B^{(k)}) := \left( \sum_{i=1}^n A_i x_i^{(k)} \right) - b$ 
8   while True do
9      $\lambda_A := \lambda_B^{(k)} + \frac{1}{L^{(k)}} \nabla d(\lambda_B^{(k)})$ 
10    foreach  $i = 1, \dots, n$  do
11       $x_i^{(k)} := \arg \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_A)$ 
12    end
13     $\nabla d(\lambda_A) := \left( \sum_{i=1}^n A_i x_i^{(k)} \right) - b$ 
14    if  $|(\lambda_B^{(k)} - \lambda_A)^T (\nabla d(\lambda_B^{(k)}) - \nabla d(\lambda_A))| \leq \frac{1}{2} L^{(k)} \|\lambda_A - \lambda_B^{(k)}\|_2^2$  then
15      break
16    else
17       $L^{(k)} := \rho L^{(k)}$ 
18    end
19  end
20   $\lambda_A^{(k+1)} := \lambda_A$ 
21  Compute  $\alpha_{k+1} \in (0, 1)$  from  $\alpha_{k+1}^2 = (1 - \alpha_{k+1}) \alpha_k^2$ 
22   $\beta_k := \frac{\alpha_k(1 - \alpha_k)}{\alpha_k^2 + \alpha_{k+1}}$ 
23   $\lambda_B^{(k+1)} := \lambda_A^{(k+1)} + \beta_k (\lambda_A^{(k+1)} - \lambda_A^{(k)})$ 
24   $k := k + 1$ 
25 end
Output:  $\lambda_A^{(k)}$ 

```

---

**Convergence:** If we choose  $L^{(k)} := L$  the convergence speed of Algorithm 4 is optimal sublinear, i.e.

$$d(\lambda^*) - d(\lambda^{(k)}) \leq \left( \frac{4L}{(2\sqrt{L} + k\sqrt{\gamma_0})^2} \right) \left( d(\lambda^*) - d(\lambda^{(0)}) + \frac{\gamma_0}{2} \|\lambda^{(0)} - \lambda^*\|_2^2 \right), \quad (20)$$

with some  $\gamma_0 \in \mathbb{R}$ .  $L$  is the Lipschitz constant of  $\nabla d(\lambda)$ . The addition of the line-search based Lipschitz approximation does not affect the asymptotic convergence, thus FG-AL is an optimal method.

**Communication:** Since  $L^{(k)}$  is adjusted with a line-search procedure, central coordination is necessary. The coordinated processes calculate their contribution to the dual gradient in parallel, see Lines 5 and 11. Then the contributions are sent to the coordinator, which governs the stepsize procedure accordingly.

#### 4.5. Fast gradient method with adaptive restart

A recent approach from [38] tries to avoid the increase of the objective in fast gradient methods by a simple heuristic. Whenever

$$-\nabla d(\lambda_B^{(k)})^T (\lambda_A^{(k+1)} - \lambda_A^{(k)}) > 0 \quad (21)$$

holds,  $\lambda_B^{(k+1)}$  is reset to  $\lambda_A^{(k+1)}$ . This method is given in Algorithm 5.

---

**Algorithm 5:** Fast gradient method with adaptive restart (FG-AR)

---

**Input** :  $\lambda^{(0)}$ ,  $L$ ,  $\alpha_0 \in (0, 1)$

```

1  $\lambda_A^{(k)} := \lambda_B^{(k)} := \lambda^{(0)}$ 
2  $k := 0$ 
3 while no convergence do
4   foreach  $i = 1, \dots, n$  do
5      $x_i^{(k)} := \arg \min_{x_i \in \mathcal{X}_i} L_i(x_i, \lambda_B^{(k)})$ 
6   end
7    $\nabla d(\lambda_B^{(k)}) := \left( \sum_{i=1}^n A_i x_i^{(k)} \right) - b$ 
8    $\lambda_A^{(k+1)} := \lambda_B^{(k)} + \frac{1}{L} \nabla d(\lambda_B^{(k)})$ 
9   Compute  $\alpha_{k+1} \in (0, 1)$  from  $\alpha_{k+1}^2 = (1 - \alpha_{k+1})\alpha_k^2$ 
10   $\beta_k := \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2 + \alpha_{k+1}}$ 
11   $\lambda_B^{(k+1)} := \lambda_A^{(k+1)} + \beta_k (\lambda_A^{(k+1)} - \lambda_A^{(k)})$ 
12  if  $-\nabla d(\lambda_B^{(k)})^T (\lambda_A^{(k+1)} - \lambda_A^{(k)}) > 0$  then
13     $\lambda_B^{(k+1)} := \lambda_A^{(k+1)}$ 
14     $\alpha_{k+1} := 1$ 
15  end
16   $k := k + 1$ 
17 end
Output:  $\lambda_A^{(k)}$ 
```

---

**Convergence:** The authors show that for strongly convex quadratic functions (optimal) linear convergence can be attained. In other words, the optimal linear convergence is not affected by the restart mechanism. For our problem, we expect sublinear convergence rate.

**Communication:** The heuristic restart procedure has to be implemented with centralized coordination, since we have to check the sign of an inner product of two vectors in Line 12. However, the gradient computation may take place in parallel in Line 5. Again, the knowledge of the Lipschitz constant  $L$  is necessary in all subproblems.

## 5. Methods for non-strongly convex QPs

In this section, we consider methods for non-strongly convex problems of form (1). Hence for each  $i \in \{1, \dots, n\}$ , the matrix  $H_i$  in (1) is positive semidefinite. The two methods

presented here have their origin in the method of multipliers that was initially suggested by Hestenes [24] and make use of the (partial) *augmented Lagrangian function* defined as

$$L_\rho(x, \lambda) := \left( \sum_{i=1}^n \frac{1}{2} x_i^T H_i x_i + c_i^T x_i \right) + \lambda^T \left( \sum_{i=1}^n A_i x_i - b \right) + \frac{\rho}{2} \left\| \sum_{i=1}^n A_i x_i - b \right\|_2^2, \quad (22)$$

where  $\rho$  is often called *penalty parameter*. This function (compared to the partial Lagrangian function defined in (2)) has an extra quadratic term that introduces curvature to the quadratic subproblems. The corresponding dual function  $d_\rho(\lambda)$  is concave and defined by

$$d_\rho(\lambda) := \min_{x \in \mathcal{X}} L_\rho(x, \lambda). \quad (23)$$

The main drawback of this construction is that the evaluation of  $d_\rho(\lambda)$  cannot be parallelized anymore due to the added quadratic term. However, as it will be presented later on, by adding some auxiliary variables one can introduce separability.

### 5.1. Alternating direction method of multipliers

The *alternating direction method of multipliers* (ADMM) is an increasingly popular method originating from [18, 21] and employing the augmented Lagrangian. A recent summary was published in [6]. In order to make the partial augmented Lagrangian function separable, we introduce slack variables  $z_i \in \mathbb{R}^p$ ,  $i = 1, \dots, n$  and solve an equivalent problem of the form

$$\min_{x, z} \sum_{i=1}^n \frac{1}{2} x_i^T H_i x_i + c_i^T x_i \quad (24a)$$

$$\text{s.t. } A_i x_i - z_i = 0, \quad i = 1, \dots, n \quad (24b)$$

$$x_i \in \mathcal{X}_i, \quad i = 1, \dots, n, \quad (24c)$$

$$\sum_{i=1}^n z_i = b. \quad (24d)$$

The corresponding augmented Lagrangian function reads as

$$L_\rho(x, z, \lambda) := \sum_{i=1}^n L_{\rho,i}(x_i, z_i, \lambda) := \sum_{i=1}^n \left( \frac{1}{2} x_i^T H_i x_i + c_i^T x_i + \lambda_i^T (A_i x_i - z_i) + \frac{\rho}{2} \|A_i x_i - z_i\|_2^2 \right), \quad (25)$$

where  $z = \text{col}(z_1, \dots, z_n)$ ,  $\lambda_i$  is the dual variable corresponding to the  $i$ -th constraint in (24b) and  $\lambda = \text{col}(\lambda_1, \dots, \lambda_n)$ . This method is closely related to the method of multipliers [24], although ADMM divides the primal variables to two subsets,  $x$  and  $z$ , respectively and minimizes the augmented Lagrangian along these variables in an alternating fashion. More precisely, in each iteration, an update of primal variables  $x$ ,

$z$  and dual variables  $\lambda$  is carried out as

$$x^{(k+1)} := \arg \min_{x \in \mathcal{X}} L_\rho \left( x, z^{(k)}, \lambda^{(k)} \right) \quad (26a)$$

$$z^{(k+1)} := \left( \arg \min_z L_\rho \left( x^{(k+1)}, z, \lambda^{(k)} \right) \right. \\ \left. \text{s.t. } \sum_{i=1}^n z_i = b \right) \quad (26b)$$

$$\lambda_i^{(k+1)} := \lambda_i^{(k)} + \rho \left( A_i x_i^{(k+1)} - z_i^{(k+1)} \right). \quad (26c)$$

Note that (26a) can be divided into  $n$  independent quadratic programming problems, i.e.

$$\min_{x \in \mathcal{X}} L_\rho \left( x, z^{(k)}, \lambda^{(k)} \right) = \sum_{i=1}^n \min_{x_i \in \mathcal{X}_i} \frac{1}{2} x_i^T H_i x_i + c_i^T x_i + \\ + \left( \lambda_i^{(k)} \right)^T \left( A_i x_i - z_i^{(k)} \right) + \frac{\rho}{2} \|A_i x_i - z_i^{(k)}\|_2^2. \quad (27)$$

In (26b), we have to solve a large, convex, sparse, equality constrained QP. This step can be done explicitly by first calculating the dual solution  $\mu^* \in \mathbb{R}^p$  of (26b) as

$$\mu^* = \frac{\left( \sum_{i=1}^n \rho A_i x_i^{(k+1)} + \lambda_i^{(k)} \right) - \rho b}{n}, \quad (28)$$

and computing the primal solution  $z_i^*$  following

$$z_i^* = \frac{\rho A_i x_i^{(k+1)} + \lambda_i^{(k)} - \mu^*}{\rho}. \quad (29)$$

---

**Algorithm 6:** Alternating Direction Method of Multipliers (ADMM)

---

**Input** :  $x^{(0)}, z^{(0)}, \lambda^{(0)}, \rho$   
1  $k := 0$   
2 **while** no convergence **do**  
3     **foreach**  $i = 1, \dots, n$  **do**  
4          $x_i^{(k+1)} := \arg \min_{x_i \in \mathcal{X}_i} L_{\rho,i}(x_i, z^{(k)}, \lambda^{(k)})$   
5     **end**  
6      $\mu^* := \frac{\left( \sum_{i=1}^n \rho A_i x_i^{(k+1)} + \lambda_i^{(k)} \right) - \rho b}{n}$   
7     **for**  $i = 1, \dots, n$  **do**  
8          $z_i^{(k+1)} := \frac{\rho A_i x_i^{(k+1)} + \lambda_i^{(k)} - \mu^*}{\rho}$   
9          $\lambda_i^{(k+1)} := \lambda_i^{(k)} + \rho (A_i x_i^{(k+1)} - z_i^{(k+1)})$   
10     **end**  
11      $k := k + 1$   
12 **end**  
**Output:**  $x^{(k)}, z^{(k)}, \lambda^{(k)}$

---

**Convergence:** The convergence rate of ADMM for convex problems in the worst case was shown to be  $O\left(\frac{1}{k}\right)$  in [23]. More precisely, assuming that at least one of the subproblems is solved exactly we have

$$\|w^{(k)} - w^{(k+1)}\|_H^2 \leq \frac{1}{k+1} \|w^{(0)} - w^*\|_H^2, \quad (30)$$

with  $w = \text{col}(x, z, \lambda)$  and some matrix  $H \succeq 0$ . Moreover, if (24a) is strongly convex global linear convergence is guaranteed [12], i.e.

$$\|w^{(k+1)} - w^*\|_G^2 \leq \frac{1}{1+\delta} \|w^{(k)} - w^*\|_G^2, \quad (31)$$

where  $\delta > 0$  and for some  $G \succeq 0$ . In practise, ADMM is often sensitive to the choice of the penalty parameter  $\rho$ . A too low penalty parameter might lead to divergence, while a too high one may cause numerical difficulties.

**Communication:** ADMM may be implemented in a fully distributed fashion. Indeed, the update of variable  $x_i$  in Line 4 uses only locally available variables. Assume that we want to update  $z_i$  of subproblem  $i$ . Here, the knowledge of the whole  $\mu^*$  is not necessary, only those coordinates are needed which correspond to constraints to which subproblem  $i$  contributes. In order to calculate these, we need to transmit the variables  $x_i$  and  $\lambda_i$  between connected subproblems. Similarly, only certain directions of variables  $z_i$  and  $\lambda_i$  need to be updated, for which the corresponding  $\mu^*$  values are already available.

## 5.2. Inexact Uzawa method

The *inexact Uzawa method* [47] employs the augmented Lagrangian function (25) also and is closely related to ADMM. However, the augmented Lagrangian is extended with an extra term, which penalizes big changes in the variable  $x_i$ . The method proceeds as

$$x^{(k+1)} := \arg \min_{x \in \mathcal{X}} L_\rho(x, z^{(k)}, \lambda^{(k)}) + \frac{1}{2} \|x - x^{(k)}\|_D^2 \quad (32a)$$

$$z^{(k+1)} := \left( \arg \min_z L_\rho \left( x^{(k+1)}, z, \lambda^{(k)} \right) \right) \quad (32b)$$

s.t.  $\sum_{i=1}^n z_i = b$

$$\lambda_i^{(k+1)} := \lambda_i^{(k)} + \rho \left( A_i x_i^{(k+1)} - z_i^{(k+1)} \right), \quad (32c)$$

where  $D = \text{diag}(D_1, \dots, D_n) \succeq 0$ . Note that once  $D = 0$ , the resulting algorithm is ADMM. Similarly to ADMM, in (32a) we have to solve  $n$  QPs in parallel if  $D$  is chosen to be block-diagonal, which we assume. However, if  $D \succ 0$  holds, the QP subproblems become strictly convex, which is not necessarily the case with ADMM. The QP in (32b) can be solved again explicitly with local communications.

**Convergence:** The convergence properties of the Inexact Uzawa method are the same as the ones of ADMM, we refer to [23] for more details.

**Algorithm 7:** Inexact Uzawa Method (IUM)

---

**Input** :  $x^{(0)}, z^{(0)}, \lambda^{(0)}, \rho, D$

1 **while** *no convergence* **do**

2   **foreach**  $i = 1, \dots, n$  **do**

3      $x_i^{(k+1)} := \arg \min_{x_i \in \mathcal{X}_i} L_{\rho,i}(x_i, z^{(k)}, \lambda^{(k)}) + \frac{1}{2} \|x_i - x_i^{(k)}\|_D^2$

4   **end**

5    $\mu^* := \frac{\left( \sum_{i=1}^n \rho A_i x_i^{(k+1)} + \lambda_i^{(k)} \right) - \rho b}{n}$

6   **for**  $i = 1, \dots, n$  **do**

7      $z_i^{(k+1)} := \frac{\rho A_i x_i^{(k+1)} + \lambda_i^{(k)} - \mu^*}{\rho}$

8      $\lambda_i^{(k+1)} := \lambda_i^{(k)} + \rho (A_i x_i^{(k+1)} - z_i^{(k+1)})$

9   **end**

10    $k := k + 1$

11 **end**

**Output:**  $\lambda^{(k+1)}$

---

**Communication:** The communication scheme can be again distributed. The reasoning presented at ADMM holds for IUM as well.

We summarize the discussed methods in Table 1. In the next chapter, we present experimental results on the practical convergence behaviour of the discussed methods.

Table 1. List of implemented methods. The meaning of the columns are "Abbr": short abbreviation, "Name": a longer name, "Hessian": definiteness of Hessian, "Comm": nature of communication, "Lip.": if the knowledge of the Lipschitz constant of the dual gradient function is necessary.

| Abbr. | Name                                         | Hessian         | Comm.  | Lip. |
|-------|----------------------------------------------|-----------------|--------|------|
| GM    | Gradient method                              | $H_i \succ 0$   | local  | Yes  |
| GBB   | Global Barzilai-Borwein method               | $H_i \succ 0$   | global | No   |
| FG    | Fast gradient method                         | $H_i \succ 0$   | local  | Yes  |
| FG-AL | Fast gradient method with adaptive Lipschitz | $H_i \succ 0$   | global | No   |
| FG-AR | Fast gradient method with adaptive restart   | $H_i \succ 0$   | local  | Yes  |
| ADMM  | Alternating direction of multipliers method  | $H_i \succeq 0$ | local  | No   |
| IUM   | Inexact Uzawa method                         | $H_i \succeq 0$ | local  | No   |

## 6. Benchmark Suite

In this section, we introduce a set of benchmark quadratic programming problems. In Table 2, we have listed all the test problems with their dimensions. The non strongly convex problems were all made strongly convex for all the methods to be able to carry out a fair comparison.

The extra curvature was introduced by a diagonal regularization of form

$$\hat{H}_i := H_i + \sigma \mathbf{I} \quad (33)$$

with some  $\sigma > 0$ . A Lipschitz constant of the dual gradient function  $\nabla d(\lambda)$  of each

Table 2. List of benchmark problems.  $m$ : number of variables,  $n$ : number of subproblems, #eq: number of equalities, #ineq: number of inequalities,  $\sigma$ : regularization parameter,  $L$ : Lipschitz constant of dual gradient function.

| Name         | $m$    | $n$ | #eq   | #ineq  | $\sigma$ | $L$      | Citation   |
|--------------|--------|-----|-------|--------|----------|----------|------------|
| AUG2DC-5     | 20200  | 5   | 10000 | 0      | 0.       | 8.00E+00 | [32]       |
| AUG2DCQP-5   | 20200  | 5   | 10000 | 40400  | 0.       | 8.00E+00 | [32]       |
| CONT-100-2   | 10197  | 2   | 9801  | 20394  | 0.       | 6.40E+05 | [32]       |
| CONT-200-8   | 40397  | 8   | 39601 | 80794  | 0.       | 2.56E+06 | [32]       |
| AUG2D-5      | 20200  | 5   | 10000 | 0      | 1E-1     | 2.20E+01 | [32]       |
| AUG2DQP-5    | 20200  | 5   | 10000 | 20200  | 1E-0     | 4.00E+00 | [32]       |
| CONT-101-2   | 10197  | 2   | 10098 | 20394  | 1E-2     | 6.38E+03 | [32]       |
| CONT-300-9   | 90597  | 9   | 90298 | 181194 | 1E-3     | 6.40E+04 | [32]       |
| DTOC3-5      | 14999  | 5   | 9998  | 2      | 1E-5     | 1.05E+05 | [32]       |
| UBH1-18      | 18009  | 18  | 12000 | 12030  | 1E-4     | 4.06E+04 | [32]       |
| JG1-8        | 1288   | 8   | 1205  | 87     | 1E-9     | 3.31E+14 | this paper |
| JG2-4        | 5788   | 4   | 5701  | 95     | 1E-9     | 3.31E+14 | this paper |
| JG3-11       | 5789   | 11  | 5701  | 97     | 1E-9     | 7.11E+15 | this paper |
| DSL1-PT-8    | 10976  | 8   | 1569  | 21952  | 5E-1     | 2.85E+05 | this paper |
| DSL2-PT-12   | 41292  | 12  | 6883  | 82584  | 5E-1     | 7.57E+05 | this paper |
| 2nd-ord-ch   | 117760 | 128 | 79360 | 235520 | 0.       | 1.40E+01 | [9]        |
| HPV-sys-8    | 27812  | 8   | 26976 | 2518   | 1E-3     | 1.81E+08 | [42]       |
| HPV-full-384 | 27812  | 384 | 26976 | 2518   | 1E-3     | 1.81E+08 | [42]       |
| SMOKEc-AG-16 | 41850  | 16  | 20250 | 43200  | 1E-4     | 4.90E+04 | [29]       |
| SMOKEd-AG-24 | 5730   | 24  | 2826  | 5808   | 1E-4     | 4.44E+04 | [29]       |
| SMOKEd-AG-66 | 17406  | 66  | 9420  | 15972  | 1E-4     | 4.75E+04 | [29]       |

problem was calculated by neglecting local constraints  $x_i \in \mathcal{X}_i$ , more precisely

$$L := \max \text{eig} \sum_{i=1}^n A_i Q_i^{-1} A_i^T. \quad (34)$$

#### *Benchmarks from Maros and Mészáros*

The first group of benchmarks include problems from the convex quadratic programming test suite from Maros and Mészáros [32]. This group includes problems AUG2DC-5, AUG2DCQP-5, CONT-100-2, CONT-200-8, AUG2D-5, AUG2DQP-5, CONT-101-2, CONT-300-9, DTOC3-5 and UBH1-18, which are all academic and have block-diagonal Hessian matrix. The separation of variables took place artificially and has no underlying meaning.

#### *Benchmarks from practical applications*

The problems in this group are JG1-8, JG2-4, JG3-11, DSL1-PT-8 and DSL2-PT-12. The first three benchmarks originate from a nonlinear robust optimal control problem for obstacle avoidance. The last two benchmarks are two instances of a problem related to sparse vectoring resource allocation for improving the performance of digital subscriber line (DSL) broadband access networks [44]. Here, the subproblems summarize the optimization problem of a number of DSL agents.

#### *Benchmarks with natural distributed structure*

In this group, problems 2nd-ord-ch, SMOKEc-AG-16, SMOKEd-AG-24 and SMOKEd-AG-66, HPV-sys-8 and HPV-full-384 can be found. 2nd-ord-ch is an academic example of a chain of double integrators. Each subproblem correspond to a certain time interval of an optimal control problem of the whole system. Thus, the coupling constraints couple the whole system behaviour between different time intervals.

Problem HPV-sys-8 originates from [42], which is a subproblem of a nonlinear optimal control problem of a hydro power plant. The goal is on the one hand to track the power reference, on the other hand to track the steady state of the system, while respecting operational constraints. Here, the subproblems describe the whole system behaviour in different time intervals, which are coupled together by the coupling constraints. The HPV-sys-8 problem has the form of

$$\min_{x,u} \sum_{i=0}^{n-1} \frac{1}{2} x_i^T Q_i x_i + q_i^T x_i + \frac{1}{2} u_i^T R_i u_i + r_i^T u_i \quad (35a)$$

$$\text{s.t. } x_{j+1} = Ax_j + Bu_j \quad (35b)$$

$$x_0 = \tilde{x}_0 \quad (35c)$$

$$\underline{x}_j \leq x_j \leq \bar{x}_j, \quad \underline{u}_j \leq u_j \leq \bar{u}_j, \quad j = 0, \dots, n-1 \quad (35d)$$

where  $x_i$  and  $u_i$  are the state and control variables of the  $i$ -th time interval,  $Q_i \succ 0, R_i \succ 0$  are weighting matrices,  $A$  is the system matrix,  $B$  is the input matrix.

Another instance of the HPV problem is the HPV-full-384 benchmark. Since this plant is intrinsically distributed, several river reaches terminated with dams and reservoirs are connected together, each subproblem correspond to a certain subsystem of the plant on a certain time interval. More precisely, we have

$$\min_{x,u,y,z} \sum_{i=0}^{n-1} \sum_{k=1}^m \frac{1}{2} (x_i^k)^T Q_i^k x_i^k + (q_i^k)^T x_i^k + \frac{1}{2} (u_i^k)^T R_i^k u_i^k + (r_i^k)^T u_i^k \quad (36a)$$

$$\text{s.t. } x_{j+1}^l = A_j^l x_j^l + B_j^l u_j^l + C_j^l z_j^l \quad (36b)$$

$$y_j^l = D_j^l x_j^l + E_j^l u_j^l + F_j^l z_j^l \quad (36c)$$

$$z_j^l = \sum_{k=1}^m G_j^k y_j^k \quad (36d)$$

$$\underline{x}_i^l \leq x_i^l \leq \bar{x}_i^l, \quad \underline{u}_i^l \leq u_i^l \leq \bar{u}_i^l, \quad j = 0, \dots, n-1, \quad l = 1, \dots, m, \quad (36e)$$

where  $x_j^l$ ,  $u_j^l$ ,  $y_j^l$  and  $z_j^l$  are the state variable, the control variable, the coupling output variable and the coupling input variable of subsystem  $l$  on the  $j$ -th time interval, respectively. The constraint (36b) represents the dynamic behaviour of subsystem  $l$  on the  $j$ -th time interval. In (36c), the finite approximation of some state variables, typically water flow and level, are calculated. Via (36d), the different subsystems are coupled together, e.g. water levels are made equal on the boundary of two reaches.

Problems SMOKEc-AG-16, SMOKEd-AG-24 and SMOKEd-AG-66 represent different instances of a dynamic estimation problem. Given a set of smoke sensors in a building consisting of connected rooms, based upon the measurements the origin of the smoke is sought for. The subproblems correspond to the connected rooms in the building. The problem structure is very similar as the one of (36).

## 7. Numerical results

In this section, we present the practical convergence properties of methods considered in the previous sections. We report our comparisons based on the number of QP solutions necessary to fulfill a well-defined convergence criterion. Let  $Ax^{(k)} := \sum_{i=1} A_i x_i^{(k)}$ ,  $\epsilon > 0$ .

We consider the primal infeasibility of the coupling constraints

$$\frac{\|Ax^{(k)} - b\|_\infty}{\max(1, \|Ax^{(0)} - b\|_\infty)} \leq \epsilon, \quad (37)$$

as a measure to terminate the benchmarked algorithms.

In our performance evaluation, we follow the methodology of *performance profiles* proposed in [13]. Given a set of problems  $\mathcal{P}$  with  $|\mathcal{P}| = n_p$  elements and a set of methods  $\mathcal{M}$  with  $|\mathcal{M}| = n_m$  elements, we define the quantity  $K_{p,m}$  as the number of QP solutions to reach convergence in terms of (37) with problem  $p \in \mathcal{P}$  and method  $m \in \mathcal{M}$ . The *performance ratio*  $r_{p,m}$  of a fixed problem  $p$  and a fixed method  $m$  shows how  $m$  performs on problem  $p$  when compared to the best performance on the same problem, i.e.

$$r_{p,m} := \frac{K_{p,m}}{\min \{K_{p,\hat{m}} | \hat{m} \in \mathcal{M}\}}. \quad (38)$$

We assume that for all  $p \in \mathcal{P}$  and  $m \in \mathcal{M}$  there is  $K_{\max} \geq K_{p,m}$  and  $K_{\max} = K_{p,m}$  holds if and only if method  $m$  cannot solve problem  $p$ . The value of  $K_{\max}$  has no effect on the performance evaluation. We define the function  $\tilde{\rho}_m(\tau) : \mathbb{R} \rightarrow [0, 1]$  as

$$\tilde{\rho}_m(\tau) := \frac{1}{n_p} |\{p \in \mathcal{P} | r_{p,m} \leq \tau\}| \quad (39)$$

The function  $\tilde{\rho}_m(\tau)$  shows the empirical probability of the performance ratio being smaller than  $\tau$  for method  $m$  on the problem set  $\mathcal{P}$ . In other words, the ratio of problems solved by method  $m$  in not more iterations than  $\tau$  times the iterations of the best method on the same problem.

### ***Comparison based upon inaccurate solutions***

First-order methods often have difficulties with converging to a highly accurate solution due to the lack of quadratic convergence. For this reason, we evaluate our comparison in two different scenarios. In the first scenario, we request for a solution of accuracy  $\epsilon = 10^{-3}$  only.

We have plotted  $\tilde{\rho}_m(\tau)$  on  $\tau \in [0, 10]$  for all discussed methods in Figure 1 to discuss their the overall quality. From this plot, we can conclude that IUM has the most wins. It has a probability of 61.56 % to be the best solver for a fixed problem. The second best is ADMM with 44.44 %. The third best is FG-AR, which is the winner in 38.89 % of the cases. With FG this number is 11.11 %. With GM, FG-AL and GBB, the probability of being the best solver is 0, meaning that they have never performed the best on any problems. If we chose being within a factor of  $\tau = 3$  of the best solver then FG-AR becomes competitive and solves the 55.56 % of the problems. It can also be seen that methods FG-AR, GBB, FG and FG-AR solve above 50 % of the problems at most with a factor 4 of the best solver. In other words, allowing 4 times more QP solutions than the best method results in solving the 50 % of the problems with these methods. In the interval  $\tau \in [0, 10]$  GM cannot be even seen. This demonstrates the poor performance of GM. If we consider  $\tau = 21$ , GM can solve only 5.56 % of the problems.

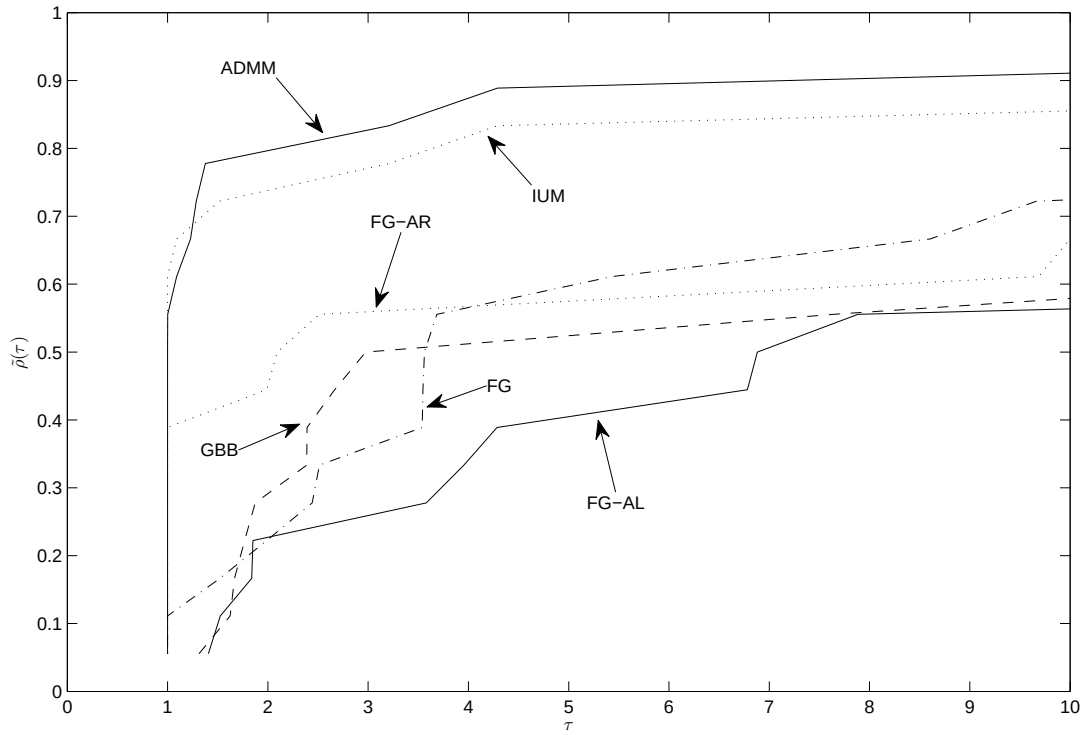


Figure 1. The performance measures of different methods on  $\tau \in [0, 10]$  to reach a tolerance of  $\epsilon = 10^{-3}$ .

### *Comparison based upon highly accurate solutions*

We expect that due to the sublinear convergence behaviour of the discussed approaches, a highly accurate solution requires many more iterations. Figure 2 shows the performance comparison where we have set a high accuracy of  $\epsilon = 10^{-5}$  in the stopping criterion. In this scenario, ADMM is the best solver with 50 % probability. IUM and FG-AR have solved 27.78 % of the problems as best methods. This is a remarkable decrease compared to the performance measure with low accurate solutions. FG-AL is the best method with 5.56 % probability. FG-AL, GM and GBB have never been the best methods on our test suite. If we regard a factor  $\tau > 1$  of the best solver, IUM becomes competitive very quickly and also FG-AR performs fairly well.

In order to obtain better insight into how these methods behave, we have plotted the primal infeasibility defined in (37) versus the iteration counter on a log-log scale for some problems. In Figure 3 the evolution of the primal infeasibility is depicted on problem AUG2DC-5. It is clear that for this problem FG-AR is the winner, while GM needs almost two order of magnitude more iterations to reach the same accuracy. ADMM and IUM are also competitive.

In Figure 4, the same converge measure is shown on problem DSL1-PT-8. Here, ADMM and IUM coincide and the fast gradient methods are competitive and GM performs the worst. It is interesting to notice that all the methods seem to have difficulties at a certain tolerance value and require many iterations to obtain a highly accurate solution.

Figure 5 depicts the primal infeasibility on problem SMOKE-AG-66. This time ADMM and IUM are the winners, FG and FG-AR are competitive. FG-AL had some difficulties in the beginning due to its adaptivity. GM seems to be stable, but very slow.

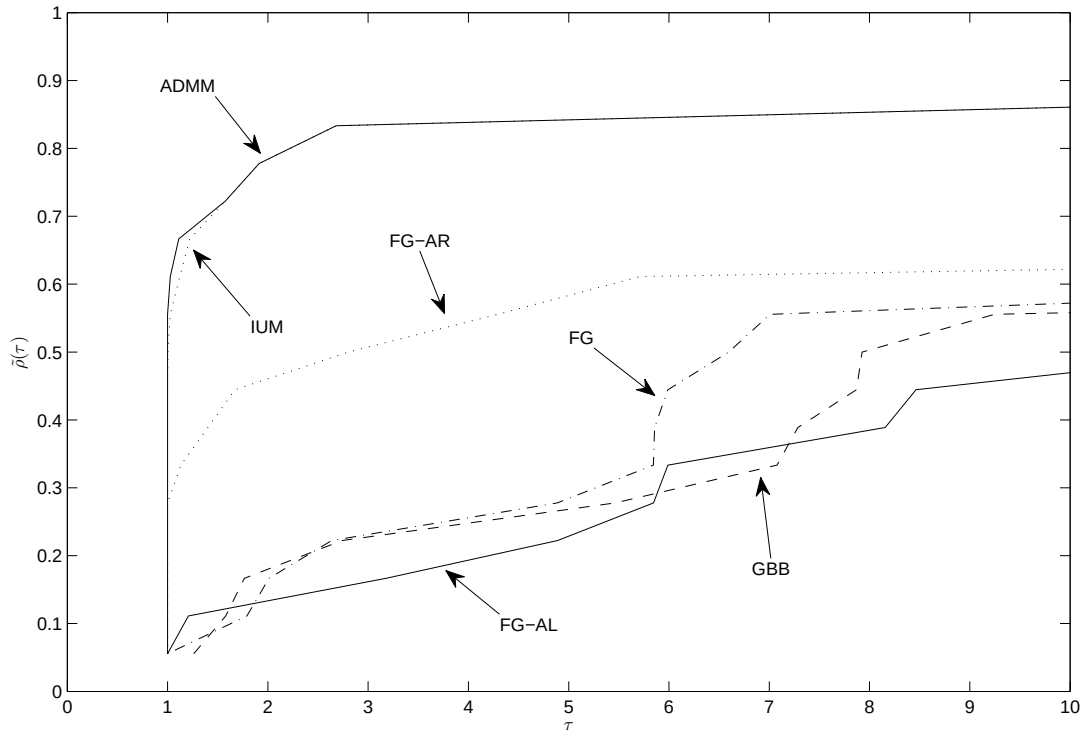


Figure 2. The performance measures of different methods on  $\tau \in [0, 10]$  to reach a tolerance of  $\epsilon = 10^{-5}$ .

### *Comparison of runtime with centralized solution*

As we have seen before, the number of necessary QP solutions are often in the order of  $10^3$  or even more. One consequence is that the runtime to solve these problems in a distributed fashion is very large. We have solved all problems also centrally with CPLEX, a sparsity exploiting interior-point algorithm, to get an idea how competitive the distributed methods can be. This approach turns out to be extremely efficient. We have managed to solve each problem in maximum 12 seconds. CPLEX uses thread-based parallel linear algebra within the interior-point algorithm.

We have carried out our experiments on an x86-64 computer with 24 Intel Xeon 2.5 GHz cores equipped with 64377 MB shared memory. In Table 3, the ratio between the runtime of 1 iteration of each distributed method and the solution time of the centralized solver can be found. The smaller ratio means less runtime per iteration. For instance, with GM on problem SMOKE-AGd-24 this ratio is 0.01 meaning that 1 iteration of GM takes 1 % of the time that CPLEX needs to solve the complete problem. Another example is ADMM on DSL2-PT-12, where this ratio is 2747.01. Essentially, each iteration of ADMM costs about 2700 times more than solving the whole problem centrally. We conclude that all distributed approaches considered in this paper are not competitive with centralized structure-exploiting solvers.

## **8. Software implementation - PyDQP**

In this section, we give a brief introduction to the software implementation. The discussed methods were implemented in Python [46] and released as open-source software package called PyDQP. At the moment, 11 different dual decomposition based algorithms are available to solve decomposable QPs on a distributed memory system having the form

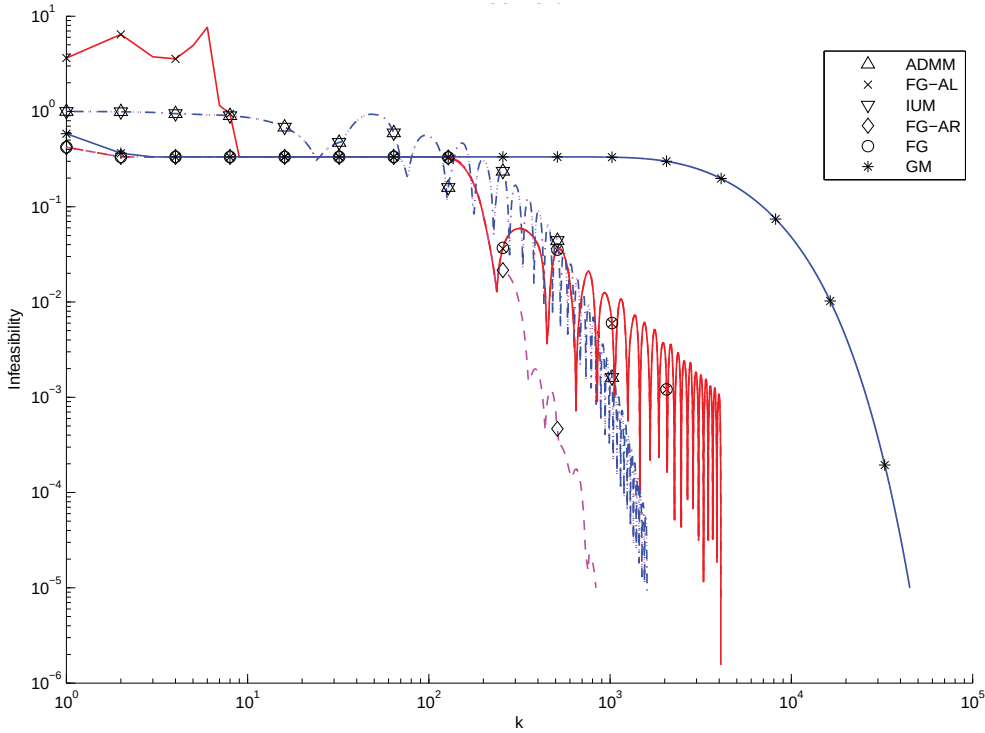


Figure 3. The primal infeasibility versus the iteration number of different methods on problem AUG2DC-5.

| Problem         | GM   | FG   | FG-AL | FG-AR | GBB  | ADMM    | IUM     |
|-----------------|------|------|-------|-------|------|---------|---------|
| AUG2DC-5        | 1.30 | 1.31 | 1.32  | 1.38  | 1.38 | 5.70    | 5.57    |
| AUG2DCQP-5      | 0.20 | 0.23 | 0.20  | 0.21  | 0.21 | 1.69    | 1.71    |
| CONT-100-2      | 0.13 | 0.14 | 0.13  | 0.14  | 0.14 | 5.45    | 5.52    |
| CONT-200-8      | 0.08 | 0.09 | 0.10  | 0.08  | 0.08 | 1.21    | 1.48    |
| AUG2D-reg-5     | 3.07 | 2.96 | 3.02  | 3.05  | 3.22 | 11.41   | 12.05   |
| AUG2DQP-reg-5   | 0.42 | 0.46 | 0.42  | 0.44  | 0.42 | 2.82    | 2.80    |
| CONT-101-reg-2  | 0.35 | 0.33 | 0.36  | 0.35  | 0.33 | 7.30    | 7.40    |
| DTOC3-reg-5     | 3.53 | 3.30 | 3.43  | 3.25  | 3.48 | 6.24    | 5.60    |
| HPV-sys-reg-8   | 0.05 | 0.05 | 0.04  | 0.04  | 0.05 | 1.05    | 1.06    |
| JG1-8           | 0.52 | 0.27 | 0.15  | 0.28  | 0.39 | 1.11    | 1.66    |
| JG2-4           | 0.12 | 0.14 | 0.05  | 0.12  | 0.14 | 1.55    | 1.45    |
| JG3-11          | 0.09 | 0.11 | 0.10  | 0.11  | 0.12 | 0.98    | 1.02    |
| UBH1-reg-18     | 0.86 | 0.69 | 1.07  | 1.26  | 1.30 | 2.42    | 2.43    |
| DSL1-PT-8       | 5.68 | 5.52 | 5.27  | 5.59  | 5.51 | 1385.16 | 1246.13 |
| DSL2-PT-12      | 3.14 | 3.19 | 3.24  | 3.19  | 3.12 | 2747.01 | 2602.32 |
| SMOKE-AGd-24    | 0.01 | 0.01 | 0.01  | 0.01  | 0.01 | 0.46    | 0.52    |
| SMOKE-AGd-66    | 0.01 | 0.01 | 0.01  | 0.01  | 0.01 | 0.26    | 0.26    |
| 2nd-order-chain | 0.07 | 0.07 | 0.07  | 0.07  | 0.08 | 0.24    | 0.25    |

Table 3. Comparison of the computation time of distributed and centralized solvers. Each number is a ratio of one iteration of a distributed QP method and the solution time with a centralized interior point solver.

of

$$\min_x \sum_{i=1}^n \frac{1}{2} x_i^T H_i x_i + c_i^T x_i \quad (40a)$$

$$\text{s.t.} \quad \sum_{i=1}^n A_i x_i = b \quad (40b)$$

$$\underline{x}_i \leq x_i \leq \bar{x}_i, \quad i = 1, \dots, n. \quad (40c)$$

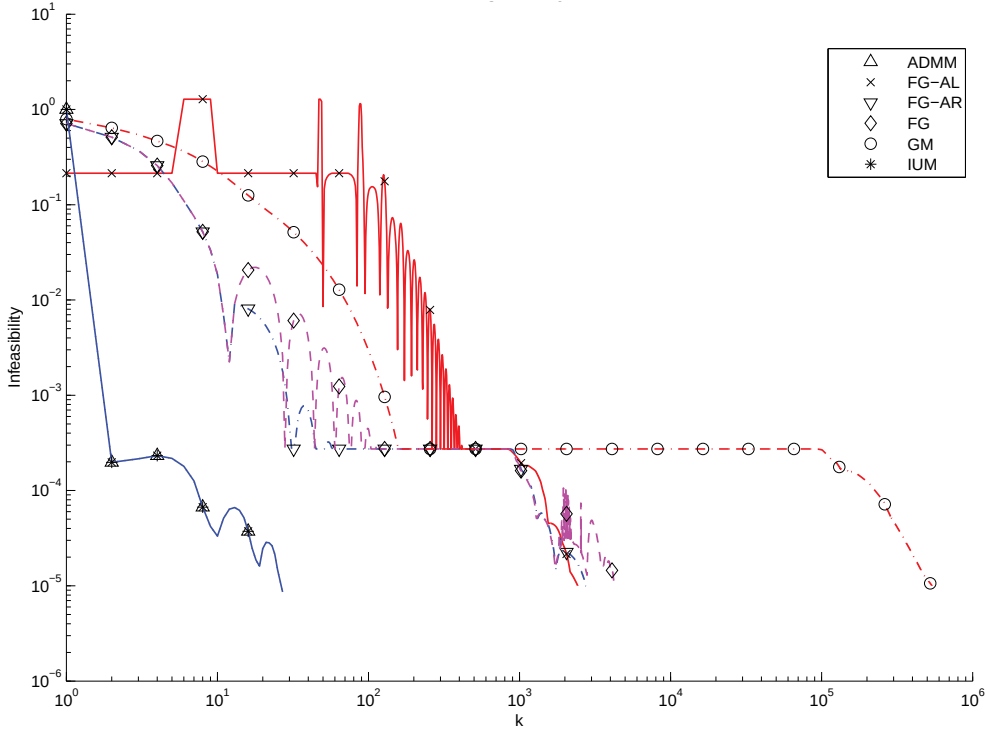


Figure 4. The primal infeasibility versus the iteration number of different methods on problem DSL1-PT-8.

In each iteration of all algorithms, the solution of several independent small scale QPs are necessary, for this purpose we have used CPLEX [26], which is invoked via a Python interface. One could use open-source QP methods to solve the local problems such as [16, 20]. In an efficient implementation, the fact that only the linear term of the subproblems change in between dual iterations should be exploited with an appropriate QP solver such as [16]. The sparse matrix algebra is provided by the scientific library of Python called SciPy [28] relying on the LAPACK library [1]. The `mpi4py` library [11], built on the top of the OpenMPI library [19] was used to provide high-level but yet efficient message passing between the parallel processes of the distributed memory system.

The input of PyDQP has to be provided via a `".mat"` file on the command-line. This file has to contain a cell array of matrices  $H_i$  and  $A_i$ , a cell array of vectors  $c_i$ ,  $\underline{x}_i$  and  $\bar{x}_i$ , and vector  $b$ . The input file can be set up easily in Matlab and in Python itself.

### *Parameters of different methods*

Many different methods depend on a number of parameters. Here, we provide a list of parameters that we used to carry out our experiments.

- GM:  $L$  is calculated
- GBB:  $M = 1$ ,  $\gamma = 10^{-4}$ ,  $\epsilon = 10^{-18}$ ,  $\delta = 10^5$ ,  $\sigma = 0.3$
- FG:  $L$  is calculated,  $\alpha_0 = 0.5$
- FG-AL:  $\rho = 2$ ,  $\alpha_0 = 0.5$ ,  $L^{(0)} = 1$
- FG-AR:  $L$  is calculated,  $\alpha_0 = 0.5$
- ADMM:  $\rho = 10$
- IUM:  $\rho = 10$ ,  $D = \mathbf{I} \cdot 10^{-3}$

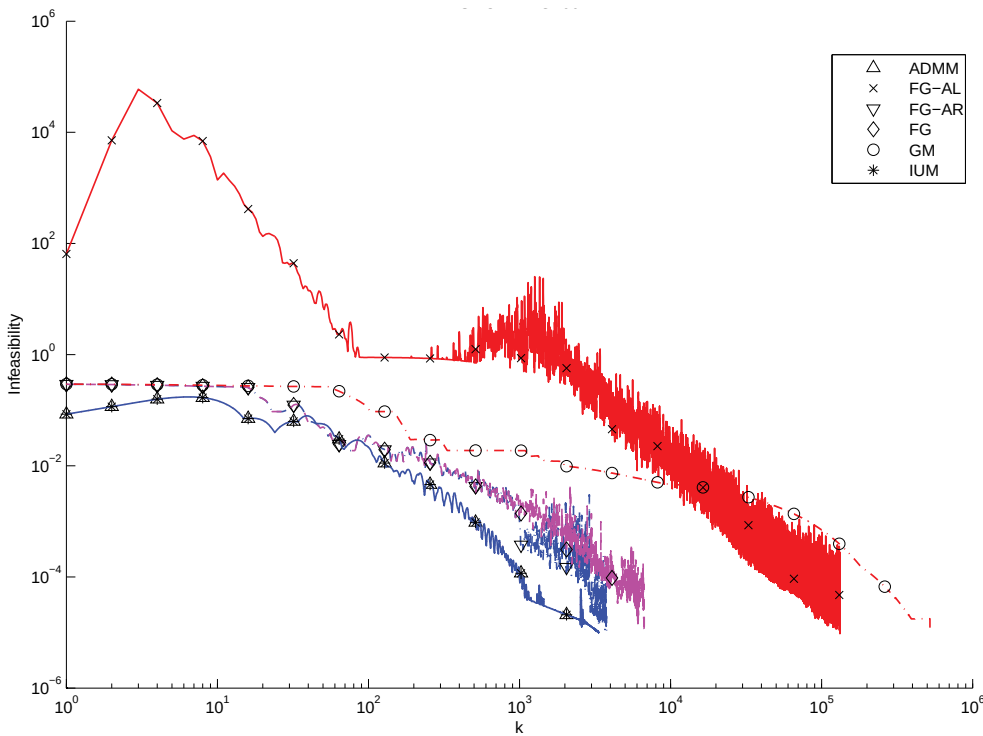


Figure 5. The primal infeasibility versus the iteration number of different methods on problem SMOKE-AG-66.

## 9. Conclusions

In this study, we have investigated parallel and distributed algorithms exploiting duality for decomposable convex quadratic programming problems. We have studied several classical and state-of-the-art optimization methods using only gradients, such as gradient and fast gradient methods, alternating direction method of multipliers and their variants. We have assessed their theoretical properties and communication requirements. The three main contributions of our paper are the following. First, a benchmark suite of large scale convex quadratic programs with decoupled structure was set up. This test suite has been made freely available. Second, an open-source software has been released, which implements 11 different distributed QP methods including the discussed ones. Third, from our computational experiments, we concluded that the alternating direction method of multipliers (ADMM) and its close relative the inexact Uzawa method (IUM) are the best candidates for distributed QP solvers that use dual decomposition. The restarted version of the fast gradient method is also often well performing, whereas GM and GBB gives very poor performance. According to our experiments, none of the distributed methods are competitive with a centralized structure-exploiting interior-point method.

## Acknowledgements

The authors thank Joris Gillis, Maarten Breckpot, Arsin Grünig and Paschalis Tsiaflakis for their contribution to the QP test suite.

This research was supported by Research Council KUL: PFV/10/002 Optimization in Engineering Center OPTEC, GOA/10/09 MaNet and GOA/10/11 Global real-time optimal control of autonomous robots and mechatronic systems. Flemish Government: IOF/KP/SCORES4CHEM, FWO: PhD/postdoc grants and projects: G.0320.08 (con-

vex MPC), G.0377.09 (Mechatronics MPC); IWT: PhD Grants, projects: SBO LeCoPro; Belgian Federal Science Policy Office: IUAP P7 (DYSCO, Dynamical systems, control and optimization, 2012-2017); EU: FP7-EMBOCON (ICT-248940), FP7-SADCO ( MC ITN-264735), ERC ST HIGHWIND (259 166), Eurostars SMART, ACCM.

## References

- [1] E. Anderson, Z.B. C., Bischof, S. Blackford, J. Demmel, J. Dongarra, J.D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, PA, 1999.
- [2] J. Barzilai and J.M. Borwein, *Two-Point Step Size Gradient Methods*, IMA J Numer Anal 8 (1988), pp. 141–148.
- [3] A. Beccuti and M. Morari, *A distributed solution approach to centralized emergency voltage control*, in *American Control Conference, 2006*, June, 2006, p. 6.
- [4] S. Becker, E. Candès, and M. Grant, *Templates for convex cone problems with applications to sparse signal recovery*, Mathematical Programming Computation 3 (2011), pp. 165–218, Available at <http://dx.doi.org/10.1007/s12532-011-0029-5>.
- [5] D. Bertsekas and J.N. Tsitsiklis, *Parallel and distributed computation: Numerical methods*, Prentice Hall, 1989.
- [6] S. Boyd, N. Parikh, E. Chu, and B. Peleato, *Distributed optimization and statistics via alternating direction method of multipliers*, Foundations and Trends in Machine Learning 3 (2011), pp. 1–122.
- [7] A. Cauchy, *Méthode générale pour la résolution des systèmes d'équations simultanées*, Comp. Rend. Sci. Paris 25 (1847), pp. 536–538.
- [8] R. Cendrillon, W. Yu, M. Moonen, J. Verlinden, and T. Bostoen, *Optimal multiuser spectrum balancing for digital subscriber lines*, Communications, IEEE Transactions on 54 (2006), pp. 922 – 933.
- [9] C. Conte, T. Summers, M. Zeilinger, M. Morari, and C. Jones, *Computational Aspects of Distributed Optimization in Model Predictive Control*, in *Conference on Decision and Control (CDC)*, Dec., Available at <http://control.ee.ethz.ch/index.cgi?page=publications;action=details;id=4192>, Maui, HI, USA, 2012.
- [10] Y. Dai and L. Liao, *R-linear convergence of the barzilai and borwein gradient method*, IMA Journal of Numerical Analysis 22 (2002), pp. 1–10, Available at <http://imajna.oxfordjournals.org/content/22/1/1.abstract>.
- [11] L. Dalcín, R. Paz, M. Storti, and J. D'Elía, *MPI for Python: Performance improvements and MPI-2 extensions*, J. Parallel Distrib. Comput. 68 (2008), pp. 655–662, Available at <http://dx.doi.org/10.1016/j.jpdc.2007.09.005>.
- [12] W. Deng and W. Yin, *On the global and linear convergence of the generalized alternating direction method of multipliers*, Tech. Rep., Rice University CAAM, 2012, tR12-14.
- [13] E. Dolan and J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program. 91 (2002), pp. 201–213.
- [14] M. Fardad, F. Lin, and M. Jovanovic, *On the dual decomposition of linear quadratic optimal control problems for vehicular formations*, in *Decision and Control (CDC), 2010 49th IEEE Conference on*, Dec., 2010, pp. 6287 –6292.
- [15] H. Ferreau, A. Kozma, and M. Diehl, *A Parallel Active-Set Strategy to Solve Sparse Parametric Quadratic Programs arising in MPC*, in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands*, 2012.
- [16] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, *qpOASES: A parametric active-set algorithm for quadratic programming*, Mathematical Programming Computation (2013), (under review).
- [17] A. Friedlander, J. Martinez, and M. Raydan, *A new method for large-scale box constrained convex quadratic minimization problems*, Optimization Methods and Software 5 (1995), pp. 57–74.
- [18] D. Gabay and B. Mercier, *A dual algorithm for the solution of nonlinear variational problems via finite element approximation*, Computers & Mathematics with Applications 2 (1976), pp. 17 – 40, Available at <http://www.sciencedirect.com/science/article/pii/0898122176900031>.
- [19] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kam-badur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, and T.S. Woodall, *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation*, in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, 2004, pp. 97–104.
- [20] E. Gertz and S. Wright, *Object-Oriented Software for Quadratic Programming*, ACM Transactions on Mathematical Software 29 (2003), pp. 58–81.
- [21] R. Glowinski and A. Marrocco, *Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité, d'une classe de problèmes de Dirichlet non linéaires*, Rev. Française Automat. Informat. Recherche Opérationnelle, RAIRO Analyse Numérique 9 (1975), pp. 41–76.
- [22] J. Gondzio, *Matrix-free interior point method*, Computational Optimization and Applications (2010), pp. 1–24, Available at <http://dx.doi.org/10.1007/s10589-010-9361-3>, 10.1007/s10589-010-9361-3.
- [23] B. He and X. Yuan, *On non-ergodic convergence rate of Douglas-Rachford alternating direction*

- method of multipliers*, 2012. (submitted for publication).
- [24] M.R. Hestenes, *Multiplier and gradient methods*, Journal of Optimization Theory and Applications 4 (1969), pp. 303–320, Available at <http://dx.doi.org/10.1007/BF00927673>, 10.1007/BF00927673.
  - [25] H. Huynh, *A large-scale quadratic programming solver based on block-LU updates of the KKT system*, Ph.D. thesis, Stanford University, 2008.
  - [26] IBM Corp., *IBM ILOG CPLEX V12.1, User's Manual for CPLEX* (2009).
  - [27] T. Jensen, J. Jørgensen, P. Hansen, and S. Jensen, *Implementation of an optimal first-order method for strongly convex total variation regularization*, BIT Numerical Mathematics 52 (2012), pp. 329–356, Available at <http://dx.doi.org/10.1007/s10543-011-0359-8>.
  - [28] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python* (2001–), Available at <http://www.scipy.org/>.
  - [29] A. Kozma, J. Andersson, C. Savorgnan, and M. Diehl, *Distributed Multiple Shooting for Optimal Control of Large Interconnected Systems*, in *Proceedings of the International Symposium on Advanced Control of Chemical Processes*, 2012.
  - [30] C. Lemaréchal, *Lagrangian relaxation*, in *Computational Combinatorial Optimization*, M. Jünger and D. Naddef, eds., Lecture Notes in Computer Science, Vol. 2241, Springer Berlin / Heidelberg, 2001, pp. 112–156, Available at [http://dx.doi.org/10.1007/3-540-45586-8\\_4](http://dx.doi.org/10.1007/3-540-45586-8_4), 10.1007/3-540-45586-8\_4.
  - [31] C. Maes, *A regularized active-set method for sparse convex quadratic programming*, Ph.D. thesis, Stanford University, 2010.
  - [32] I. Maros and C. Mészáros, *A repository of convex quadratic programming problems*, Optimization Methods and Software 11 (1999), pp. 431–449.
  - [33] I. Necoara, D. Doan, and J. Suykens, *Application of the proximal center decomposition method to distributed model predictive control*, in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, dec., 2008, pp. 2900–2905.
  - [34] I. Necoara and J. Suykens, *Applications of a smoothing technique to decomposition in convex optimization.*, IEEE Trans. Automatic control 53 (2008), pp. 2674–2679.
  - [35] Y. Nesterov, *Introductory lectures on convex optimization: a basic course*, Applied Optimization, Vol. 87, Kluwer Academic Publishers, 2004.
  - [36] Y. Nesterov, *Smooth minimization of non-smooth functions*, Math. Program. 103 (2005), pp. 127–152, Available at <http://www.springerlink.com/content/c7wheayx86u1vfwx/>.
  - [37] Y. Nesterov, *Gradient methods for minimizing composite objective function*, CORE Discussion paper 76 (2007).
  - [38] B. O'Donoghue and E. Candes, *Adaptive Restart for Accelerated Gradient Schemes*, 2012, Available at <http://www-stat.stanford.edu/~candes/publications.html>. (submitted for publication).
  - [39] M. Raydan, *The barzilai and borwein gradient method for the large scale unconstrained minimization problem*, SIAM Journal on Optimization 7 (1997), pp. 26–33, Available at <http://epubs.siam.org/doi/abs/10.1137/S1052623494266365>.
  - [40] S. Richter, M. Morari, and C. Jones, *Towards computational complexity certification for constrained MPC based on Lagrange Relaxation and the fast gradient method*, in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, dec., 2011, pp. 5223–5229.
  - [41] S. Samar, S. Boyd, and D. Gorinevsky, *Distributed Estimation via Dual Decomposition*, in *Proceedings European Control Conference (ECC)*, Kos, Greece, 2007, pp. 1511–1516.
  - [42] C. Savorgnan and M. Diehl, *Control benchmark of a hydro power plant*, Tech. report, Optimization in Engineering Center, KU Leuven, <http://homes.esat.kuleuven.be/~mdiehl>, 2010, Available at <http://homes.esat.kuleuven.be/~mdiehl>.
  - [43] Q. Tran-Dinh, C. Savorgnan, and M. Diehl, *Combining lagrangian decomposition and excessive gap smoothing technique for solving large-scale separable convex optimization problems*, Computational Optimization and Applications (2011), pp. 1–29, Available at <http://arxiv.org/abs/1105.5427>.
  - [44] P. Tsiaflakis, J. Vangorp, M. Moonen, J. Verlinden, and G. Ysebaert, *Partial crosstalk cancellation in a multi-user xDSL environment*, in *Communications, 2006. ICC '06. IEEE International Conference on*, Vol. 7, 2006, pp. 3264–3269.
  - [45] P. Tsiaflakis, I. Necoara, J. Suykens, and M. Moonen, *Improved Dual Decomposition Based Optimization for DSL Dynamic Spectrum Management*, IEEE Transactions on Signal Processing 58 (2010), pp. 2230–2245.
  - [46] G. van Rossum and F. Drake, *Python Reference Manual*, PythonLabs, Virginia, USA (2001).
  - [47] X. Zhang, M. Burger, and S. Osher, *A unified primal-dual algorithm framework based on bregman iteration*, J. Sci. Comput. 46 (2011), pp. 20–46, Available at <http://dx.doi.org/10.1007/s10915-010-9408-8>.
  - [48] B. Zhou, L. Gao, and Y. Dai, *Gradient methods with adaptive step-sizes*, Computational Op-

timization and Applications 35 (2006), pp. 69–86, Available at <http://dx.doi.org/10.1007/s10589-006-6446-0>, 10.1007/s10589-006-6446-0.