REVISITING DESIGN ASPECTS OF A QP SOLVER FOR WORHP

Marcel Jacobse, Christof Büskens

University of Bremen
Center for Industrial Mathematics, WG Optimization and Optimal Control
Bibliothekstraße 5, 28359 Bremen, Germany
{mjacobse,bueskens}@math.uni-bremen.de

ABSTRACT

SQP methods for nonlinear programming rely on a quadratic programming solver for computing a search direction in each major iteration. From the start, the large scale NLP solver WORHP has been using the interior point method QPSOL within its SQP framework, which was developed specifically for WORHP. Experience from usage of WORHP in many areas and development of features like sensitivity analysis and feasibility refinement raised interest in a reworked, extended interface between WORHP and its QP solver. Furthermore, additional concepts like multiple centrality correctors seemed promising for improving the overall performance.

Hence, a revised QP solver was designed and implemented. Mehrotra's algorithm that was implemented in QPSOL was extended by Gondzio's multiple centrality correctors and weighting of corrector steps was added. Special care was taken to handle the very general NLP formulation of WORHP efficiently, yielding a very general problem formulation for standalone quadratic programming as well. A clear interface was implemented for retrieving sensitivity derivatives from the quadratic solver directly, allowing WORHP Zen and feasibility refinement procedures easy access to them.

The paper deals with these algorithmic and interface aspects for the development of the new solver within WORHP. Numerical results on the CUTEst test set for nonlinear programming are presented to show the performance improvements over the previous method.

***Index Terms*—** Quadratic Programming, Nonlinear Programming, Sequential Quadratic Programming, Solver Design

1. INTRODUCTION

The numerical solution of nonlinear programming problems is a common task arising in many applications, for example as parameter identification or optimal control problems. The nonlinear programming solver WORHP [1] solves large scale problems of this type using the underlying quadratic solver QPSOL [2]. Since its initial release, WORHP has been used widely in many fields and was extended by several additional features. As a result, there was a rise in opportunities and new requirements for the old Fortran code of QPSOL. Therefore, a reworked quadratic solver was designed and implemented in C++.

In this paper, we discuss several issues regarding the implementation of this new quadratic programming solver for WORHP, which we will refer to as WORHP QP. In Section 2, a brief summary of the role of quadratic programs within WORHP's solution algorithm is given. Solution methods for quadratic programs are described, leading into a more detailed description of interior point methods for quadratic programming in Section 3. The algorithmic techniques that are implemented in WORHP QP as an extension of QPSOL's algorithm are outlined. Section 4 deals with the improvements to the quadratic problem formulation of the new solver. The benefits of the implemented interface for sensitivity derivatives of WORHP QP over QPSOL are explained in Section 5. Finally, in Section 6 numerical results comparing WORHP using the old versus the new quadratic solver are presented.

2. FUNDAMENTALS

WORHP solves nonlinear programming (NLP) problems given in the form

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{subject to} \quad & \underline{x} \leq x \leq \overline{x} \\ & \underline{g} \leq g(x) \leq \overline{g}, \end{aligned} \tag{1}$$

with an objective function $f : \mathbb{R}^n \to \mathbb{R}$, a general constraint function $g : \mathbb{R}^n \to \mathbb{R}^m$ and bounds

$$\begin{aligned} \underline{x} &\in (\mathbb{R} \cup \{-\infty\})^n, & \overline{x} &\in (\mathbb{R} \cup \{+\infty\})^n \\ \underline{g} &\in (\mathbb{R} \cup \{-\infty\})^m, & \overline{g} &\in (\mathbb{R} \cup \{+\infty\})^m. \end{aligned}$$

By setting $\underline{x}_i = \overline{x}_i$ or $\underline{g}_j = \overline{g}_j$ for some i or j, equality constraints can be formulated. The distinction between simple bounds on the variables x (also called *box constraints*) and the *general constraints* given by g allows to handle the former more efficiently within the solver. Allowing lower, upper

and both bounded variables and constraints makes this formulation quite general. As such, it is more user friendly than equivalent formulations in which only certain types of bounds are allowed, e.g. only $\underline{g} = \{-\infty\}^m$ and $\overline{g} = 0$.

A sequential quadratic programming (SQP) method is employed to solve these problems[1]. In each iteration k, the nonlinear program (1) is approximated by the quadratic program (QP)

$$\begin{aligned} \min_{d \in \mathbb{R}^n} \quad & \frac{1}{2} d^\mathsf{T} H^{[k]} d + \nabla_x f(x^{[k]})^\mathsf{T} d \\ \text{subject to} \quad & \underline{x} \leq x^{[k]} + d \leq \overline{x} \\ & \underline{g} \leq g(x^{[k]}) + \nabla_x g(x^{[k]})^\mathsf{T} d \leq \overline{g} \end{aligned} \tag{2}$$

in which $H^{[k]}$ is either the Hessian of the Lagrangian of problem (1) at the current iterate, or an approximation following BFGS-type update methods. The solution of this quadratic subproblem yields a search direction $d^{[k]} := d$ for the current iterate $x^{[k]}$. A line search along this search direction is then employed to produce the next iterate

$$x^{[k+1]} := x^{[k]} + \alpha^{[k]} d^{[k]} \tag{3}$$

with a stepsize $\alpha^{[k]} > 0$ such that $x^{[k+1]}$ is in some sense superior to the previous iterate $x^{[k]}$, e.g. with a smaller violation of the constraints or a smaller objective function value. Merit functions or a filter are used to establish such improvement for the new iterate. Then, after incrementing the iteration counter k, the next quadratic subproblem (2) has to be solved.

2.1. Solving the quadratic subproblem

Solving the quadratic program (2) is therefore a crucial part of the SQP method. As WORHP follows the inequality-constrained (IQP) instead of the equality-constrained (EQP) approach [4], this quadratic program is in general inequality-constrained. This requires much more sophisticated solution methods compared to equality-constrained QPs for which only a single linear equation system has to be solved. The two methods that are mainly used are active set and interior point methods.

Active set methods follow the idea that an inequality-constrained QP could be treated like an equality-constrained one if it was known, which inequality constraints are *active*, i.e. meet their bounds exactly, at the optimal solution. As described above, the solution would then come down to solving a single linear equation system. Usually that active set is of course not known in advance. Active set methods therefore initially try to *guess* the optimal active set and treat active inequality constraints as equality constraints by forcing them to their bound while temporarily ignoring inactive inequality constraints. In each iteration, the guess for the active set is then updated by removing/adding inequality constraints from/to the current guess in a clever way to eventually arrive at the optimal active set and therefore the optimal solution.

Interior point methods on the other hand do not force a partition of the inequality constraints into active and inactive ones in every iteration. Instead, they approach the optimal solution from the interior by perturbing the complementarity condition which is closely related to the notion of activeness. The active set is then gradually revealed throughout the iterations by progressively decreasing the perturbation [5].

Most SQP methods use an active set solver for the quadratic subproblems, as they can easily be warmstarted, i.e. use the solution of a previous problem to solve a different, but similar problem more efficiently. As long as successive quadratic subproblems do not differ too much, this is very beneficial within the SQP framework. Warmstarting an interior point method on the other hand is somewhat difficult, as the initial iterates have to be strictly positive. An optimal iterate from a previous subproblem would have to be moved back into the interior [6]. In turn, interior point methods have a much better worst case iteration complexity and can make use of efficient general purpose sparse Cholesky-like linear equation solvers, whereas active set methods usually require more sophisticated, specialized linear algebra routines in order to be efficient [4].

WORHP uses the solver QPSOL which is an implementation of the well known primal-dual interior point method by Mehrotra [7] to solve the quadratic subproblems.

3. INTERIOR POINT METHOD

QPSOL implements Mehrotra's predictor-corrector method, closely following OOQP [8]. For sake of simplicity, the approach is briefly described for the simplified quadratic problem formulation

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{2} x^\mathsf{T} Q x + c^\mathsf{T} x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{4}$$

with matrices $Q \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m \times n}$ and vectors $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. By perturbing the complementarity condition of this problem with the barrier parameter $\mu > 0$, we obtain the optimality conditions for the barrier subproblem

$$\begin{aligned} Qx + c + A^\mathsf{T} y - z &= 0 \\ Ax - b &= 0 \\ ZXe - \mu e &= 0 \end{aligned}$$

as common for primal-dual interior point methods. The introduced vectors $y \in \mathbb{R}^m$ and $z \in \mathbb{R}^n$ are the Lagrange multipliers for the equality and inequality constraints respectively. Here and in the following, e denotes a vector of ones with

[1] Recently, a new penalty interior point algorithm [3] was added to WORHP, but in this paper we focus on the SQP algorithm.

appropriate size and for every vector v the uppercase letter V means the diagonal matrix with the entries of v on its diagonal. Applying Newton's method yields the equation system

$$\begin{pmatrix} Q & A^\mathsf{T} & 0 \\ A & 0 & 0 \\ Z & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = - \begin{pmatrix} Qx + c + A^\mathsf{T} y - z \\ Ax - b \\ ZXe - \mu e \end{pmatrix} \quad (5)$$

for the Newton direction. Interior point methods calculate such a step $\left(\Delta x^{[k]}, \Delta y^{[k]}, \Delta z^{[k]}\right)$ in each iteration k while progressively decreasing the barrier parameter μ and therefore the perturbation in the complementarity condition. The current iterates $\left(x^{[k]}, y^{[k]}, z^{[k]}\right)$ are then updated after determining a steplength multiplier $\alpha^{[k]} \in (0, 1]$, such that the new iterates

$$\begin{aligned} x^{[k+1]} &:= x^{[k]} + \alpha^{[k]} \Delta x^{[k]} \\ y^{[k+1]} &:= y^{[k]} + \alpha^{[k]} \Delta y^{[k]} \\ z^{[k+1]} &:= z^{[k]} + \alpha^{[k]} \Delta z^{[k]} \end{aligned}$$

still fulfill the required positivity conditions, i.e. $x^{[k+1]} > 0$ and $z^{[k+1]} > 0$.

Mehrotra's predictor-corrector method constructs the step $\Delta^{[k]} = \left(\Delta x^{[k]}, \Delta y^{[k]}, \Delta z^{[k]}\right)$ as a combination of a predictor and a corrector step and employs a heuristic for generating a sequence of decreasing barrier parameters. Equation system (5) is first solved for $\mu = 0$ to get a predictor step

$$\Delta_{\text{pred}} = \left(\Delta x_{\text{pred}}, \Delta y_{\text{pred}}, \Delta z_{\text{pred}}\right).$$

The maximum steplength α_{pred} is determined such that the current iterates updated by $\alpha_{\text{pred}} \Delta_{\text{pred}}$ remain positive. A target complementarity μ_{tar} is then chosen by the heuristic

$$\mu_{\text{tar}} := \left(\frac{\mu_{\text{pred}}}{\mu^{[k]}}\right)^{\eta} \mu^{[k]},$$

in which $\mu^{[k]} := {x^{[k]}}^\mathsf{T} z^{[k]}/n$ is the current average complementarity product and

$$\mu_{\text{pred}} := \frac{\left(x^{[k]} + \alpha_{\text{pred}} \Delta x_{\text{pred}}\right)^\mathsf{T} \left(z^{[k]} + \alpha_{\text{pred}} \Delta z_{\text{pred}}\right)}{n}$$

the predicted one. The exponent η is a parameter, commonly chosen from the interval $[2, 4]$. System (5) is then solved for the right hand side

$$- \begin{pmatrix} 0 \\ 0 \\ -\mu_{\text{tar}} e + \Delta X_{\text{pred}} \Delta Z_{\text{pred}} e \end{pmatrix} \quad (6)$$

to obtain the corrector step Δ_{corr}. Predictor and corrector step are then combined for the final step $\Delta^{[k]} := \Delta_{\text{pred}} + \Delta_{\text{corr}}$. Many algorithmic summaries can be found in the literature, for instance [8].

For the reworked solver, several extensions of Mehrotra's method were implemented.

3.1. Added extensions

Gondzio [9] proposes the use of multiple centrality correctors that use an abritary amount of additional solves of system (5) for different right hand sides to generate further corrector directions.

As starting point, this technique takes the combined Mehrotra predictor-corrector step as its initial predictor step Δ_{pred}, with the determined maximum stepsize α_{pred} that can be taken in this direction. With the goal to increase the maximum possible stepsize with another correction step, an increased, aspired stepsize $\tilde{\alpha} > \alpha_{\text{pred}}$ is chosen. For this increased stepsize, intermediary iterates

$$\tilde{x} := x^{[k]} + \tilde{\alpha} \Delta x_{\text{pred}} \text{ and } \tilde{z} := z^{[k]} + \tilde{\alpha} \Delta z_{\text{pred}}$$

are computed. Of course, these intermediary variables will in general violate the non-negativity constraints $\tilde{x} \geq 0$ and $\tilde{z} \geq 0$. To compensate, a corrector direction that aims to reduce this violation needs to be computed. For this, a more conservative target $t \in \mathbb{R}^n$ with

$$t_i := \begin{cases} \gamma \mu_{\text{tar}} & \text{if } \tilde{x}_i \tilde{z}_i \leq \gamma \mu_{\text{tar}} \\ \frac{1}{\gamma} \mu_{\text{tar}} & \text{if } \tilde{x}_i \tilde{z}_i \geq \frac{1}{\gamma} \mu_{\text{tar}} \\ \tilde{x}_i \tilde{z}_i & \text{otherwise} \end{cases} \quad \text{for all } i \in \{1, \ldots, n\}$$

is chosen, compared to $\mu_{\text{tar}} e$ in Mehrotra's correction (6). The parameter $\gamma \in (0, 1)$ controls the size of the interval around the target complementarity μ_{tar} that is projected onto. System (5) is then solved for the right hand side

$$- \begin{pmatrix} 0 \\ 0 \\ 0 \\ \tilde{X} \tilde{Z} e - t \end{pmatrix},$$

yielding the corrector direction Δ_{corr} which is then combined with the predictor step to the new, corrected step $\Delta := \Delta_{\text{pred}} + \Delta_{\text{corr}}$. Unlike Mehrotra's correction, this correction can be applied as often as desired, by treating Δ as a new predictor direction Δ_{pred} again. A more detailed description is given in [9].

Numerical results show that significant iteration savings can be achieved and benefits for warmstarting can also be observed [6]. Therefore, Gondzio's correctors are already implemented in many interior point solvers like HOPDM [9], OOPS [6], OOQP [8] and `sparse_quadratic_prog` [10] and now also in WORHP QP.

Colombo and Gondzio [11] suggest a further improvement of the corrector scheme by introducing weighted correctors. They motivate this based on observations that Mehrotra's corrector does not always improve the possible stepsize compared to the predictor step, which is not unexpected as the second order approximation is not necessarily sufficient to follow the central path properly. Therefore, instead of adding

the full corrector step Δ_{corr} to the previously determined predictor step Δ_{pred}, a weight $\omega \in (0, 1]$ is used such that the allowed stepsize α that can be taken with the combined direction

$$\Delta := \Delta_{\text{pred}} + \omega \Delta_{\text{corr}} \tag{7}$$

is maximized. As this would require the solution of a nontrivial bi-level optimization problem, a simple line search strategy is used to find an approximate ω. Weighting can be applied for Mehrotra's corrector step, as well as for each iteration of Gondzio's centrality correctors by considering the current combined step as the predictor step Δ_{pred} in (7). Numerical results in [11] show the benefits of this approach, so that it was implemented in WORHP QP as well.

4. QUADRATIC PROBLEM FORMULATION

QPSOL takes problems of the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{2} x^\mathsf{T} Q x + c^\mathsf{T} x \\ \text{subject to} \quad & \underline{x} \leq x \leq \overline{x} \\ & Ax = b \\ & Cx \leq d. \end{aligned} \tag{8}$$

Hence, general constraints with lower and upper bounds cannot be handled directly and equality and inequality constraints are separated from each other. Furthermore, fixed variables are not dealt with explicity, so if $\underline{x}_i = \overline{x}_i$ for an $i \in \{1, \ldots, n\}$, the solver will employ two directly conflicting logarithmic barrier terms which is quite problematic. As described in Section 2, WORHP needs the solution of (2) in every iteration, which has to be transformed into formulation (8) so that it can be understood by QPSOL. Therefore, in every iteration WORHP has to

1. split the Jacobian $\nabla_x g(x^{[k]})^\mathsf{T}$ into the equality and inequality constraint matrices A and C
2. duplicate all general inequality constraints with bounds on both sides in C, i.e for a constraint

   $$-\infty < \underline{g}_i \leq g_i(x) \leq \overline{g}_i < \infty$$

   we get the *two* constraints

   $$\begin{aligned} g_i(x^{[k]}) + \nabla_x g_i(x^{[k]})^\mathsf{T} d &\leq \overline{g}_i \\ -g_i(x^{[k]}) - \nabla_x g_i(x^{[k]})^\mathsf{T} d &\leq -\underline{g}_i \end{aligned}$$

   in the QP
3. create an extra equality constraint in A for all fixed variables $\underline{x}_i = \overline{x}_i$.

While 1. is mostly an inconvenience, the other two items also increase the dimension and number of nonzeros in the system matrix of the linear equation system which ultimately has to be solved. That system is essentially (5), except for transformations due to the different problem formulation (see [8]). Performance can therefore directly suffer from the problem formulation restrictions imposed by QPSOL.

The newly designed interface on the other hand allows to formulate quadratic problems in the most general form

$$\begin{aligned} \min_{d \in \mathbb{R}^n} \quad & \frac{1}{2} x^\mathsf{T} Q x + c^\mathsf{T} x \\ \text{subject to} \quad & \underline{x} \leq x \leq \overline{x} \\ & \underline{b} \leq Ax \leq \overline{b} \end{aligned} \tag{9}$$

which directly resembles WORHP's formulation and allows to input (2) directly, without any transformations. This makes the interface code between the NLP and the QP layer a lot cleaner. However, the transformations that were done for QPSOL in this code before have to be handled internally within the new solver WORHP QP now. This interal handling is described in the following.

4.1. Ordering

The ordering of the general constraints on the NLP layer is kept for the QP. That way, sensitivities and multipliers stay in the same order as well (see also Section 5). As a result, the optimal Lagrange multipliers of the quadratic subproblem directly correspond to the current Lagrange multipliers of the NLP, without any necessary reordering. Grouping of constraints, for example separating equality and inequality constraints can then be done interally in the QP solver. That way, for instance calculations of complementarity related values can iterate easily on only the appropriate constraint type, inequality constraints in this example. However, this index mapping is hidden from the user, who therefore does not have to worry about it, unlike when using QPSOL.

4.2. Fixed variables

Fixed variables are completely removed from the problem, meaning that not (9), but a reduced problem is formed internally. To remove a fixed variable x_i from the problem, its constant value has to be put into every appearance of the variable in the unmodified problem. If x_i appears in the j-th constraint, i.e. if the entry A_{ij} in the i-th row and j-th column of A is nonzero, then the values of $\underline{b}_j$ and $\overline{b}_j$ have to be updated by

$$\begin{aligned} \underline{b}_j &\leftarrow \underline{b}_j - A_{ij} x_i \\ \overline{b}_j &\leftarrow \overline{b}_j - A_{ij} x_i. \end{aligned}$$

Similarly, for each entry Q_{ik}, $k \in \{1, \ldots, n\}$ in the i-th row of Q, the gradient c has to be updated to

$$c_k \leftarrow c_k + Q_{ik} x_i.$$

Technically, another constant term in the objective function would appear, but it can be neglected. Then, the i-th column of Q and A and the i-th row of Q and c can be dropped, as the variable x_i does not appear in the problem as an optimization variable anymore. This and several other preprocessing techniques for quadratic programs are discussed in [12].

4.3. Both sided general constraints

To handle general inequality constraints with lower and upper bounds, the following transformation is done to avoid duplication of rows in matrix A. For sake of simple presentation assume that there are no simple box constraints, i.e. $\underline{x} = -\infty$ and $\overline{x} = \infty$ and that all general constraints have lower and upper bounds, i.e. $\underline{b} > -\infty$ and $\overline{b} < \infty$. Then for the Lagrangian we have

$$L(x, \underline{z}, \overline{z}) = \frac{1}{2} x^\mathsf{T} Q x + c^\mathsf{T} x + \underline{z}^\mathsf{T} (\underline{b} - Ax) + \overline{z}^\mathsf{T} (Ax - \overline{b})$$

which yields the optimality conditions

$$\begin{aligned} Qx + c - A^\mathsf{T} \underline{z} + A^\mathsf{T} \overline{z} &= 0 \\ \underline{b} - Ax &\leq 0 \\ Ax - \overline{b} &\leq 0 \\ \underline{Z} \left(Ax - \underline{b}\right) &= 0 \\ \overline{Z} \left(\overline{b} - Ax\right) &= 0 \end{aligned} \tag{10}$$

with the Lagrange multipliers

$$\underline{z} \geq 0 \text{ and } \overline{z} \geq 0.$$

To avoid duplication of A in the Jacobian of Newton's method due to (10), we introduce the difference between lower and upper multipliers as $z_d := \overline{z} - \underline{z}$ and the average as $z_a := \frac{\overline{z} + \underline{z}}{2}$ and further simplify the complementarity conditions with the slack variables s, leading to

$$\begin{aligned} Qx + c + A^\mathsf{T} z_d &= 0 \\ Ax - s &= 0 \\ \left(z_a - \frac{1}{2} z_d\right) (s - \underline{b}) &= 0 \\ \left(z_a + \frac{1}{2} z_d\right) \left(\overline{b} - s\right) &= 0. \end{aligned}$$

with

$$\underline{b} \leq s \leq \overline{b}, z_a - \frac{1}{2} z_d \geq 0 \text{ and } z_a + \frac{1}{2} z_d \geq 0.$$

The Jacobian for Newton's method is then

$$\begin{pmatrix} Q & 0 & 0 & A^\mathsf{T} \\ A & -I & 0 & 0 \\ 0 & Z_a - \frac{1}{2} Z_d & S - \underline{B} & -\frac{1}{2} (S - \underline{B}) \\ 0 & -\left(Z_a + \frac{1}{2} Z_d\right) & \overline{B} - S & \frac{1}{2} \left(\overline{B} - S\right) \end{pmatrix}$$

so that we can write the step equation as

$$\begin{pmatrix} Q & 0 & 0 & A^\mathsf{T} \\ A & -I & 0 & 0 \\ 0 & \underline{Z} & \underline{S} & -\frac{1}{2} \underline{S} \\ 0 & -\overline{Z} & \overline{S} & \frac{1}{2} \overline{S} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta z_a \\ \Delta z_d \end{pmatrix} = \begin{pmatrix} -r_Q \\ -r_A \\ -\underline{r} \\ -\overline{r} \end{pmatrix}$$

by using the now auxiliary variables $\underline{z}, \overline{z}$ and additionally $\underline{s} := s - \underline{b}$ and $\overline{s} := \overline{b} - s$. This reduces to

$$\begin{pmatrix} Q & A^\mathsf{T} \\ A & -\left(\overline{S}^{-1} \overline{Z} + \underline{S}^{-1} \underline{Z}\right)^{-1} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta z_d \end{pmatrix} = \begin{pmatrix} -r_Q \\ -r_A + \left(\overline{S}^{-1} \overline{Z} + \underline{S}^{-1} \underline{Z}\right)^{-1} \left(-\underline{S}^{-1} \underline{r} + \overline{S}^{-1} \overline{r}\right) \end{pmatrix} \tag{11}$$

with

$$\begin{aligned} \Delta z_a &= \underline{S}^{-1} \left(-\underline{r} - \underline{Z} \Delta s\right) + \frac{1}{2} \Delta z_d \\ &= \overline{S}^{-1} \left(-\overline{r} + \overline{Z} \Delta s\right) - \frac{1}{2} \Delta z_d, \end{aligned}$$

$$\Delta s = \left(\overline{S}^{-1} \overline{Z} + \underline{S}^{-1} \underline{Z}\right)^{-1} \left(-\underline{S}^{-1} \underline{r} + \overline{S}^{-1} \overline{r} + \Delta z_d\right) \tag{12}$$

and consequently

$$\begin{aligned} \Delta \underline{z} &= \underline{S}^{-1} \left(-\underline{r} - \underline{Z} \Delta s\right) \\ \Delta \overline{z} &= \overline{S}^{-1} \left(-\overline{r} + \overline{Z} \Delta s\right). \end{aligned} \tag{13}$$

The interior point algorithm can therefore work with the primal iterates $x^{[k]}$ and $s^{[k]}$ and the dual iterates $\underline{z}^{[k]}$ and $\overline{z}^{[k]}$. System (11) then yields the auxiliary values Δz_d from which the actual directions of interest can be determined directly using (12) and (13). To achieve this, the constraint matrix A did not have to be doubled in (11), in contrast to what was described for QPSOL earlier on.

5. SENSITIVITIES

WORHP offers parametric sensitivity analysis for parameter dependent nonlinear programs with the module WORHP Zen [13]. The implicit function theorem implies that the necessary sensitivity derivatives can be calculated using the KKT matrix as it appears during the solution of the quadratic subproblems in the QP solver [14].

As QPSOL does not handle parameter perturbations or sensitivities at all on its own, WORHP Zen has to go deep into the structures of QPSOL to compute the sensitivities with the existing factorized matrix. To make matters worse WORHP Zen has to take the altered general constraint layout within the QP solver into account to refer perturbation and sensitivity indices on the NLP layer to those on the QP layer. Furthermore, WORHP Zen is intended for *post-optimal* sensitivity analysis of the NLP, i.e. after termination of WORHP.

Listing 1. Illustrative C++ pseudo code of the declarations of sensitivity related functions.

```
enum class SimplePerturbationKind {
  OBJECTIVE_LINEAR,
  GENERAL_CONSTRAINTS_CONSTANT,
  BOX_CONSTRAINTS_CONSTANT
};

class WorhpSubproblem : public QP {
  ...

  void SetSimplePerturbation(
      SimplePerturbationKind kind, int index);
  void SetNonlinearPerturbation(Vector dL_dp,
      Vector dg_dp);
  Vector GetSensitivitiesX();
  Vector GetSensitivitiesMultsBox();
  Vector GetSensitivitiesMultsGeneral();
};
```

Hence, features that rely on post-optimal sensitivity analysis of QP subproblems like feasibility refinement [15, 16] have to access the system matrix directly as well.

The newly designed solver WORHP QP simplifies these matters greatly. Two functions `SetSimplePerturbation` and `SetNonlinearPerturbation` allow setting perturbations of any kind through the new interface. The sensitivity derivatives with respect to the given perturbation can then be retrieved for the primal variables x, the Lagrange multipliers for the box constraints and those for the general constraints with the functions `GetSensitivitiesX`, `GetSensitivitiesMultsBox` and `GetSensitivitiesMultsGeneral` respectively. Listing 1 roughly illustrates how these functions are declared in the C++ code of WORHP QP. All of WORHP's post-optimal QP sensitivity analysis features can therefore retrieve the necessary sensitivity derivatives using this clean interface directly from the QP solver. Similarly, WORHP Zen can simply forward and/or use the sensitivity derivatives of the QP for the post-optimal NLP sensitivity analysis. And, as a valuable side-effect, sensitivity derivatives are also available to the user if WORHP QP is used as a standalone.

6. NUMERICAL RESULTS

To evaluate the performance of the new QP solver, we compare WORHP using first QPSOL and second WORHP QP as its QP solver. We use the test set CUTEst [17] for our evaluation. In the version from 2018-05-08 that we are using, the test set contains 1305 continuous optimization problems.

To allow for a better comparison, WORHP's default settings were altered to give stricter termination criteria and more consistent iteration behaviour. Several termination heuristics were disabled, as well as most recovery strategies. Table 1 shows the changed parameters in detail. All tests were run on a system with a dual CPU, the Intel® Xeon® CPU E5-2637 v3 @ 3.50GHz. To avoid inconsistent performance caused by varying clock speeds as much as possible, at most two problem instances were run simultaneously at all times. No parallelism for linear algebra or other operations was enabled, i.e. all problem instances ran with only one thread. All instances were compiled using `gcc` version 5.4.0 on Ubuntu 16.04.

Table 1. Changed WORHP parameters (compared to default) for the numerical experiments.

Parameter	default value	used value
KeepAcceptableSol	`True`	`False`
LowPassFilter	`True`	`False`
ScaledKKT	`True`	`False`
Ares	[42, 41, 42, 43, 44, 41, 50]	[50]

Table 2. Number of problems of the CUTEst test set for which WORHP terminated with an optimal solution, an acceptable solution or unsuccessfully for the two QP solvers.

Used QP solver	optimal	acceptable	unsuccessful
WORHP QP	1039	61	205
QPSOL	971	96	238

Table 2 shows the number of *optimal*, *acceptable* and *unsuccessful* terminations of WORHP using the old versus the new QP solver. This categorization is that of WORHP for the default tolerance parameters. The results indicate that the newly designed solver improves stability by a significant margin.

For a more detailed comparison, performance profiles as proposed by Dolan and Moré [18] are considered. These plot the percentage of problems that is solved in at most "τ-times the cost" as the best of the considered solvers for all considered solvers over τ. Different performance measures can be used to quantify the meaning of "τ-times the cost". Here, we consider the number of major iterations, the CPU time (user time of the Linux command `time`) and the number of objective function evaluations as performance measures. The CPU time performance profile for instance shows the percentage of problems that are solved within at most τ-times the fastest observed solution time over the factor τ. To overcome runtime fluctuations, the problem instances were restarted until the total runtime across all restarts reached 10 minutes, or at most 100 times and the average time was taken.

Figure 1 shows these performance profiles for all problems that terminated successfully for at least one of the two

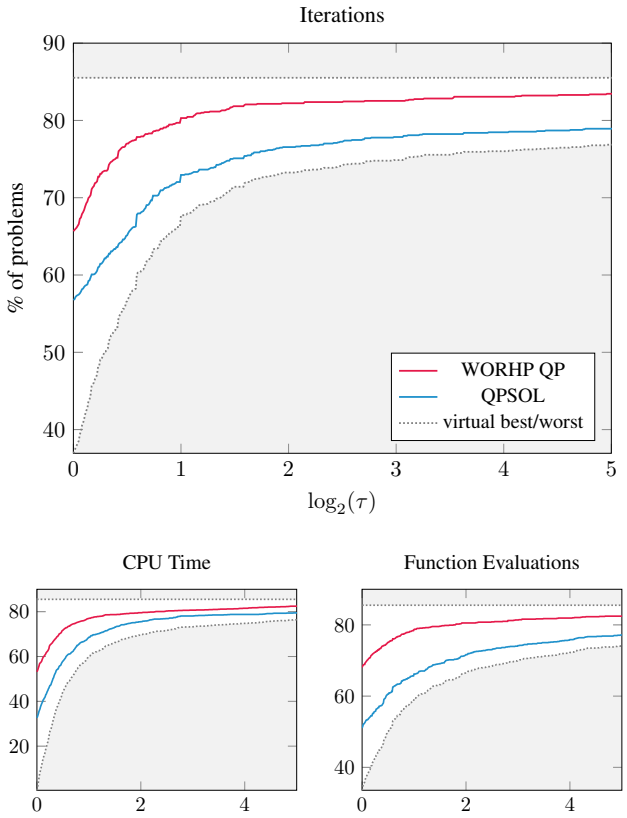

Fig. 1. Performance profiles for WORHP on the CUTEst test set using different QP solvers. All problems for which at least one variant terminated successfully are considered.

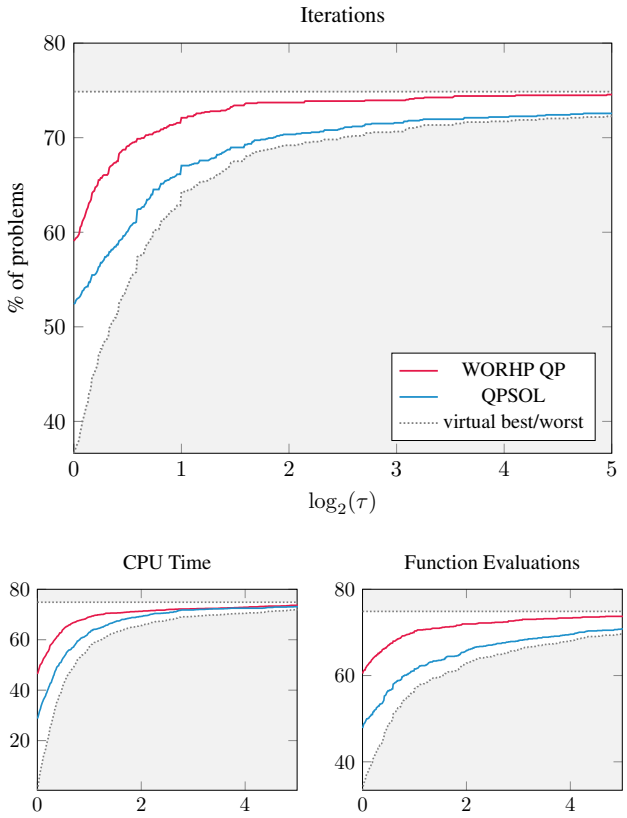

Fig. 2. Performance profiles for WORHP on the CUTEst test set using different QP solvers. Only problems for which both variants terminated successfully with close objective values are taken into account.

variants. The graphs show a clear advantage of using the new solver over the old one. Roughly 66% of problems are solved in less or the same amount of iterations with WORHP QP than with QPSOL, only 57% of problems the other way round. For CPU time and number of function evaluations these percentages are 53% to 32% and 62% to 53% respectively. Furthermore, for increasing τ the new solver stays in the lead.

As nonlinear programs can have many local minima, comparing solver runs without any further checks can be misleading. For example, if a run happens to find its way to a local minimum that is numerically much easier to handle than a minimum which a different run finds, the former run might seem much better performance wise. However, the runs are simply not really comparable in such case.

Therefore, in Figure 2 only problems for which both variants terminate successfully with roughly the same objective value (with both absolute and relative tolerances of 0.001) are taken into account. These are in total 977 of the 1305 problems, i.e. about 75%. The plots show very similar results, confirming the observations from before. WORHP QP finds the same minimum in less time than QPSOL for 46% of all problems. On the other hand, QPSOL finds the same minimum in less time than WORHP QP for only 29% of all problems. Note that these percentages are in relation to all 1305 problems, i.e. the remaining 25% are problems for which different or no minima were found. Comparing only on the 75% of problems that are solved equally, we can say that WORHP QP is faster than QPSOL in 61% of these cases. Similar advantages of WORHP QP can be observed for the other two cost measures, number of major iterations and number of objective function evaluations. Again, these improvements of the new solver for $\tau = 1$ continue for larger τ.

7. CONCLUSION

We discussed several issues regarding a reworked QP solver for WORHP. Additional implemented algorithmic approaches were briefly described. Limitations imposed by the problem formulation of the old solver QPSOL were outlined and a reworked formulation that solves these problems was presented. We described a clean interface for using sensitivity derivatives for post-optimal sensitivity analysis for both the QP as well

as the NLP layer, circumventing many of the caveats of the old implementation. Numerical results of WORHP with the reworked solver on the CUTEst test set show significant robustness and performance improvements over WORHP with QPSOL.

The authors would like to thank Renke Kuhlmann for providing the scripts to generate the performance profiles.

8. REFERENCES

[1] C. Büskens and D. Wassel, "The ESA NLP solver WORHP," in *Modeling and Optimization in Space Engineering*, G. Fasano and J. D. Pintér, Eds., New York, NY, 2013, pp. 85–110, Springer New York.

[2] M. Gerdts, *User's Guide QP Solver*, Universität der Bundeswehr München, Feb. 2013.

[3] R. Kuhlmann and C. Büskens, "A primal–dual augmented lagrangian penalty-interior-point filter line search algorithm," *Mathematical Methods of Operations Research*, vol. 87, no. 3, pp. 451–483, Jun 2018.

[4] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, NY, USA, 1999.

[5] J. Gondzio, "Interior point methods 25 years later," *European Journal of Operational Research*, vol. 218, no. 3, pp. 587 – 601, 2012.

[6] J. Gondzio and A. Grothey, "A new unblocking technique to warmstart interior point methods based on sensitivity analysis," *SIAM Journal on Optimization*, vol. 19, no. 3, pp. 1184–1210, 2008.

[7] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.

[8] E. M. Gertz and S. J. Wright, "Object-oriented software for quadratic programming," *ACM Trans. Math. Softw.*, vol. 29, no. 1, pp. 58–81, Mar. 2003.

[9] J. Gondzio, "Multiple centrality corrections in a primal-dual method for linear programming," *Computational Optimization and Applications*, vol. 6, no. 2, pp. 137–156, Sep 1996.

[10] Rogue Wave Software, "Solving sparse convex quadratic programming problems with the C numerical library," Tech. Rep., Rogue Wave Software, Boulder, CO 80301, USA, 11 2012.

[11] M. Colombo and J. Gondzio, "Further development of multiple centrality correctors for interior point methods," *Computational Optimization and Applications*, vol. 41, no. 3, pp. 277–305, Dec 2008.

[12] N. I. M. Gould and Ph. L. Toint, "Preprocessing for quadratic programming," *Mathematical Programming*, vol. 100, no. 1, pp. 95–132, May 2004.

[13] R. Kuhlmann, S. Geffken, and C. Büskens, "WORHP Zen: Parametric sensitivity analysis for the nonlinear programming solver WORHP," in *Operations Research Proceedings 2017*, N. Kliewer, J. F. Ehmke, and R. Borndörfer, Eds., Cham, 2018, pp. 649–654, Springer International Publishing.

[14] A. V. Fiacco and Y. Ishizuka, "Sensitivity and stability analysis for nonlinear programming," *Annals of Operations Research*, vol. 27, no. 1, pp. 215–235, Dec 1990.

[15] S. Geffken and C. Büskens, "Feasibility refinement in sequential quadratic programming using parametric sensitivity analysis," *Optimization Methods and Software*, vol. 32, no. 4, pp. 754–769, 2017.

[16] S. Geffken, *Effizienzsteigerung numerischer Verfahren der nichtlinearen Optimierung*, Ph.D. thesis, Universität Bremen, 2017.

[17] N. I. M. Gould, D. Orban, and Ph. L. Toint, "CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization," *Computational Optimization and Applications*, vol. 60, no. 3, pp. 545–557, Apr 2015.

[18] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical Programming*, vol. 91, no. 2, pp. 201–213, Jan 2002.