IMM
DEPARTMENT OF MATHEMATICAL MODELLING

Technical University of Denmark
DK-2800 Lyngby – Denmark

J. No. QP2
6.12.1996
HBN/ms

BOUND CONSTRAINED QUADRATIC PROGRAMMING SOLVED VIA PIECEWISE QUADRATIC FUNCTIONS

IMPLEMENTATION

Hans Bruun Nielsen

TECHNICAL REPORT

IMM-REP-1996-21

IMM

Abstract. A bound constrained quadratic program is solved via a dual problem, which is the minimization of an unbounded, piecewise quadratic function. The dual problem involves a lower bound of λ_1, the smallest eigenvalue of a symmetric, positive matrix, and is solved by Newton iteration with line search. The report describes the implementation of the algorithm, including estimation of λ_1, how to get a good starting point for the iteration, and up– and downdating of Choleky factorization. Results of extensive testing and comparison with other methods for constrained QP are given.

Key words. Bound constrained quadratic programming. Huber's M–estimator. Condition estimation. Newton iteration. Factorization update.

Contents

1. Introduction

Let $\boldsymbol{c} \in \mathbb{R}^n$ and $\boldsymbol{H} \in \mathbb{R}^{n\times n}$ be a given vector and a symmetric, positive definite matrix, respectively. We seek $\boldsymbol{y}^* \in \mathbb{R}^n$ as the solution to the constrained quadratic programming problem

$$\begin{aligned} &\min_{\boldsymbol{y}}\{q(\boldsymbol{y}) \equiv \tfrac{1}{2}\boldsymbol{y}^T\boldsymbol{H}\boldsymbol{y} - \boldsymbol{c}^T\boldsymbol{y}\} \\ &\text{subject to} \quad -\boldsymbol{e} \le \boldsymbol{y} \le \boldsymbol{e} \ . \end{aligned} \tag{1.1}$$

Here, $\boldsymbol{e} \in \mathbb{R}^n$ is the vector of all ones.

In [8] we discuss the solution of this problem via a dual ℓ_1-problem, which in its turn is solved via a series of Huber problems, cf. [7]. In [10] we give a simplified version of the method, and this report give some computational details.

Let $\lambda_1 > 0$ denote the smallest eigenvalue of $\boldsymbol{H}$, and let γ be a number such that $0 < \gamma < \lambda_1$. Further, let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a matrix that satisfies

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{H} - \gamma\boldsymbol{I} \ . \tag{1.2}$$

Now, define the Huber function

$$F(\mathbf{x}) = \frac{1}{2\gamma}\mathbf{r}^T\boldsymbol{W}\mathbf{r} + \boldsymbol{s}^T(\mathbf{r} - \frac{\gamma}{2}\boldsymbol{s}) + \frac{1}{2}\mathbf{x}^T\mathbf{x} \ , \tag{1.3a}$$

where

$$\mathbf{r} = \mathbf{r}(\mathbf{x}) = \boldsymbol{A}^T\mathbf{x} - \boldsymbol{c} \ , \tag{1.3b}$$

$$\boldsymbol{s} = \boldsymbol{s}(\mathbf{x}) = \begin{bmatrix} s_1(\mathbf{x}) \\ \vdots \\ s_n(\mathbf{x}) \end{bmatrix} \quad \text{with} \quad s_i(\mathbf{x}) = \begin{cases} -1 \text{ if } & r_i(\mathbf{x}) < -\gamma \\ \ \ 0 \text{ if } & |r_i(\mathbf{x})| \le \gamma \\ \ \ 1 \text{ if } & r_i(\mathbf{x}) > \gamma \end{cases} \tag{1.3c}$$

$$\boldsymbol{W} = \boldsymbol{W}(\mathbf{x}) = \mathrm{diag}(w_1, \ldots, w_n) \quad \text{with} \quad w_i = 1 - s_i^2 \ . \tag{1.3d}$$

In [10] we show that (1.1) has the dual problem

$$\max_{\mathbf{x}}\{-F(\mathbf{x})\} = -\min_{\mathbf{x}}\{F(\mathbf{x})\} \ . \tag{1.4}$$

The hyperplanes given by $r_j(\mathbf{x}) = \pm\gamma$ divide $\mathbb{R}^n$ into subregions, in each of which $F(\mathbf{x})$ is a quadratic. So the dual problem is to minimize a piecewise quadratic. It is easy to show that F is continuous, and also the gradient

$$\boldsymbol{g}(\mathbf{x}) = \nabla F(\mathbf{x}) = \mathbf{A}(\frac{1}{\gamma}\boldsymbol{W}\mathbf{r} + \boldsymbol{s}) + \mathbf{x} \ , \qquad (1.5)$$

varies continuously across the hyperplanes.

The minimizer $\mathbf{x}_\gamma$ of F is found by Newton iteration with line search. It satisfies $\boldsymbol{g}(\mathbf{x}_\gamma) = \boldsymbol{0}$, or

$$\mathbf{x}_\gamma = \boldsymbol{A}\boldsymbol{y}_\gamma \ , \qquad (1.6a)$$

where we have defined

$$\boldsymbol{y}_\gamma = -(\frac{1}{\gamma}\boldsymbol{W}(\mathbf{x}_\gamma)\boldsymbol{r}(\mathbf{x}_\gamma) + \boldsymbol{s}(\mathbf{x}_\gamma)) \ . \qquad (1.6b)$$

In [10] we show that $\boldsymbol{y}_\gamma = \boldsymbol{y}^*$ and $q(\boldsymbol{y}_\gamma) = -F(\mathbf{x}_\gamma)$. Thus,

$$\begin{aligned}\Gamma(\mathbf{x}) &\equiv F(\mathbf{x}) + q\big(-\tfrac{1}{\gamma}\boldsymbol{W}(\mathbf{x})\boldsymbol{r}(\mathbf{x}) - \boldsymbol{s}(\mathbf{x})\big) \\ &\geq F(\mathbf{x}_\gamma) + q(\boldsymbol{y}_\gamma) = 0\end{aligned} \qquad (1.7)$$

can serve as a gap function.

In the discussion of the algorithm we say that r_i (and the ith column of $\mathbf{A}$) is *active* at $\mathbf{x}$, if $s_i(\mathbf{x}) = 0$ (and therefore $w_i(\mathbf{x}) = 1$). The *active set* is

$$\mathcal{A} = \mathcal{A}(\mathbf{x}) = \{\ i \mid 1 \leq i \leq n \ \wedge s_i(\mathbf{x}) = 0\} \ . \qquad (1.8)$$

From (1.6b) it follows that an active $r_i(\mathbf{x}_\gamma)$ is equivalent with $|y_i^*| < 1$, so the ith constraint in (1.1) is *non active*. This can lead to confusion when we compare with other methods for solving the problem. Note, that if we knew the solution $\boldsymbol{y}^*$, then it follows from (1.6) that the corresponding $\boldsymbol{s}(\mathbf{x}_\gamma)$ is given by

$$s_i(\mathbf{x}_\gamma) = \begin{cases} -y_i^* & \text{if } |y_i^*| = 1 \\ 0 & \text{otherwise} \end{cases} \ , \qquad (1.9a)$$

and $\mathbf{x}_\gamma$ is the solution to the linear problem derived from the condition $\boldsymbol{g}(\mathbf{x}_\gamma) = \boldsymbol{0}$,

$$(\boldsymbol{A}\boldsymbol{W}(\mathbf{x}_\gamma)\boldsymbol{A}^T + \gamma\boldsymbol{I})\mathbf{x}_\gamma = \boldsymbol{A}\big(\boldsymbol{W}(\mathbf{x}_\gamma)\boldsymbol{c} - \gamma\boldsymbol{s}(\mathbf{x}_\gamma)\big) \ . \qquad (1.9b)$$

Some of the competing methods are based on the Kuhn–Tucker condition

$$\boldsymbol{H}\boldsymbol{y}^* - \boldsymbol{c} + \boldsymbol{u} = \boldsymbol{0} \ , \qquad (1.10)$$

where $u_i \neq 0$ only if the ith constraint is active. This is also the background for the widely used test problem generator of Moré and Toraldo [12]. The vector $\boldsymbol{c}$ is found as

$$\boldsymbol{c} = \boldsymbol{H}\boldsymbol{y}^* + \boldsymbol{u} \ , \qquad (1.11a)$$

where $\boldsymbol{H}$, $\boldsymbol{y}^*$ and $\boldsymbol{u}$ are generated[1)] so that a prescribed number of constraints are active and

$$\boldsymbol{H} = 10^{-desc}\boldsymbol{H}_0, \quad u_i = \begin{cases} 0 & \text{if } |y_i^*| < 1 \\ -y_i^* \cdot 10^{-\nu \cdot deg} & \text{otherwise} \end{cases} \ . \qquad (1.11b)$$

Here, ν is uniform random in [0,1], $\boldsymbol{H}_0$ is a symmetric, positive definite matrix with $\|\boldsymbol{H}_0\|_2 = 10^{ncond}$, and *desc*, *ncond*, *deg* are chosen, nonnegative parameter values. It follows that

$$\begin{aligned}&\|\boldsymbol{y}^*\|_\infty = 1, \quad \|\boldsymbol{u}\|_\infty \leq 1 \ , \\ &\|\boldsymbol{H}\boldsymbol{y}^*\|_\infty \leq \|\boldsymbol{H}\|_\infty = 10^{-desc}\|\boldsymbol{H}_0\|_\infty \sim 10^{ncond-desc} \ .\end{aligned}$$

The original Moré–Toraldo generator corresponds to $desc = 0$, and in that case we are sure that $\boldsymbol{H}\boldsymbol{y}^*$ has a marked influence on $\boldsymbol{c}$. The choice of $desc > 0$ leads to more difficult problems – see Section 2.1.

2. The Algorithm

The algorithm for computing the solution to problem (1.1) can be stated as follows.

1° Compute $\boldsymbol{R}$ as the upper triangular Cholesky factor of $\boldsymbol{H}$,

$$\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{H} \ . \qquad (2.1)$$

2° Compute γ and $\boldsymbol{A}$; Sections 3.1 and 3.2.

3° Find initial active set; Section 2.1.

4° Compute $\mathbf{x}_\gamma$; Section 2.2.

5° Compute solution to original problem (Section 2.3) and return.

1) Details are given in Section 4.1.

The problem may be given in terms of $\boldsymbol{R}$ and $\boldsymbol{c}$, in which case step 1° is omitted. Further, the algorithm is suited for "warm starts": If a problem with the same $\boldsymbol{H}$ has been solved previously, then steps 1° and 2° can be skipped. Also, step 3° can be replaced by information from the previous solution and the new $\boldsymbol{c}$.

2.1. Initial Guess

Let $\boldsymbol{s}$ and the corresponding $\boldsymbol{W}$, (1.3d), be given, and define $\boldsymbol{x}_{\mathbf{s}}$ as the solution to

$$(\boldsymbol{A}\boldsymbol{W}\boldsymbol{A}^T + \gamma\boldsymbol{I})\boldsymbol{x}_{\mathbf{s}} = \boldsymbol{A}(\boldsymbol{W}\boldsymbol{c} - \gamma\boldsymbol{s}) \ . \tag{2.2}$$

If $\boldsymbol{s} = \boldsymbol{s}(\boldsymbol{x}_\gamma)$, then it follows from Section 1 that $\boldsymbol{x}_{\mathbf{s}} = \boldsymbol{x}_\gamma$.

We want a good strategy for choosing $\boldsymbol{s}$ so that the corresponding $\boldsymbol{x}_{\mathbf{s}}$ is a good starting point for the Newton iteration described in Section 2.2.

A first guess on $\boldsymbol{s}$ is motivated by the discussion in Section 1: Ignore $\boldsymbol{u}$ – i.e. ignore the constraints in (1.1). The unconstrained minimizer of q is

$$\widetilde{\boldsymbol{y}} = \boldsymbol{H}^{-1}\boldsymbol{c} \ . \tag{2.3}$$

If $\|\widetilde{\boldsymbol{y}}\|_\infty \le 1$, then we are through: No constraint is active at the minimizer; $\boldsymbol{y}^* = \widetilde{\boldsymbol{y}}$ and $q(\boldsymbol{y}^*) = -\frac{1}{2}\boldsymbol{c}^T\widetilde{\boldsymbol{y}}$.

Otherwise, we generalize the relation (1.9a) and the definition (1.3c): Our first guess is $\widetilde{\boldsymbol{s}} = \boldsymbol{S}(-\widetilde{\boldsymbol{y}}, 1)$, given by

$$S_i(\boldsymbol{v}, \tau) = \begin{cases} 0 & \text{if } |v_i| \le \tau \\ \operatorname{sign}(v_i) & \text{otherwise} \end{cases} \ . \tag{2.4}$$

The starting point $\boldsymbol{x}_{\widetilde{\mathbf{s}}}$ found by (2.2) with $\boldsymbol{s} = \widetilde{\boldsymbol{s}}$ works well for the original Moré–Toraldo problems, where $\boldsymbol{c}$ is dominated by $\boldsymbol{H}\boldsymbol{y}^*$, cf. the discussion after (1.11). For the modified Moré–Toraldo problems ($desc > 0$), however, the active set given by $\widetilde{\boldsymbol{s}} = \boldsymbol{S}(-\widetilde{\boldsymbol{y}}, 1)$ will often be empty. Even worse: the signs in $\widetilde{\boldsymbol{s}}$ will be random, as compared with $\boldsymbol{s}(\boldsymbol{x}_\gamma)$, and the vector obtained by (2.2) with $\boldsymbol{W} = \boldsymbol{0}$, $\boldsymbol{x} = -\boldsymbol{A}\widetilde{\boldsymbol{s}}$, is a poor starting point.

We experimented with a number of other starting strategies. To shorten the presentation, let $\mathcal{N}(\boldsymbol{v}, \tau)$ denote the number of active equations, i.e. the number of $S_i(\boldsymbol{v}, \tau) = 0$. Further, let τ_m denote the mth smallest $|v_i|$, so that $\mathcal{N}(\boldsymbol{v}, \tau_m) \ge m$, where $0 \le m \le n$.

In all the experiments we used the starting guess given by $\widetilde{\boldsymbol{s}}$, provided that $\mathcal{N}(-\widetilde{\boldsymbol{y}}, 1) \ge p \cdot n$, where the parameter p was chosen as one of the values $\frac{1}{10}$, $\frac{1}{4}$ or $\frac{1}{2}$. When this condition was not satisfied, we tried another way of obtaining the initial point. Among the strategies tried were

1° Compute the initial $\boldsymbol{x}$ by (2.2) with $\boldsymbol{s} = \boldsymbol{S}(-\widetilde{\boldsymbol{y}}, \tau_{p \cdot n})$.

2° Compute the initial $\boldsymbol{x}$ by (2.2) with $\boldsymbol{s} = \boldsymbol{S}(-\boldsymbol{c}, \tau_{p \cdot n})$. Note that $\boldsymbol{c}$ is the steepest descent direction for $q(\boldsymbol{y})$ at $\boldsymbol{y} = \boldsymbol{0}$.

3° Start the Huber iterations (Section 2.2) with $\boldsymbol{x} = \boldsymbol{0}$ and use gradient directions ($\boldsymbol{h} = -\boldsymbol{A}(\frac{1}{\gamma}\boldsymbol{W}\boldsymbol{r} + \boldsymbol{s}) + \boldsymbol{x}$) in the first few iterations.

4° Start the Huber iterations with the "least squares solution" as given by (2.2) with $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{W} = \boldsymbol{I}$.

5° Compute the initial $\boldsymbol{x}$ by (2.2) with $\boldsymbol{s} = \boldsymbol{S}(-\boldsymbol{y}, \tau_{p \cdot n})$, where $\boldsymbol{y}$ is computed by a Lagrange modification of (1.1): $\boldsymbol{y} = \operatorname{argmin}\{q(\boldsymbol{y}) + \mu\boldsymbol{y}^T\boldsymbol{y}\}$, with μ chosen to reflect the influence of $\boldsymbol{u}$ in (1.10). Omitting details, we used $\mu = \gamma + 0.1 + \frac{\|\boldsymbol{c}\|_\infty}{\|\widetilde{\boldsymbol{y}}\|_\infty}$ and $\boldsymbol{y} = (\boldsymbol{A}^T\boldsymbol{A} + \mu\boldsymbol{I})^{-1}\boldsymbol{c}$.

6° Find an integer m such that $\|f^m\boldsymbol{c}\|_\infty \simeq \|\boldsymbol{H}\|_\infty$, where $f \in]0, 1[$ is a chosen number. Solve problem (1.1) with $\boldsymbol{c}$ replaced by $f^m\boldsymbol{c}$ and starting vector given by $\widetilde{\boldsymbol{s}}$. Then, for $p = m-1, \ldots, 1, 0$ use the warm start possibilities to solve (1.1) with $\boldsymbol{c}$ replaced by $f^p\boldsymbol{c}$.

None of these strategies came out a clear winner, but generally 1°, 2° or 5° was best, while strategy 4° surprisingly often was much worse than the others.

We finally settled for the starting algorithm given in (2.6). The algorithm aims at giving a gradual change from the cases where $\boldsymbol{u}$ in (1.10) has little influence to the cases where it dominates. In these extreme cases good approximations to the direction of the primal solution is given by $\widetilde{\boldsymbol{y}}$ and $\boldsymbol{c}$, respectively, and we use the interpolation

$$\boldsymbol{y} := \frac{f}{\|\boldsymbol{c}\|_2}\boldsymbol{c} + \frac{1-f}{\|\widetilde{\boldsymbol{y}}\|_2}\widetilde{\boldsymbol{y}} \quad \text{with} \quad f := 0.9 \cdot \left(1 - \frac{\mathcal{N}(-\widetilde{\boldsymbol{y}}, 1)}{n}\right)^4 \ . \tag{2.5}$$

The factor 0.9 and the exponent 4 were decided experimentally.

```
Compute ỹ by (2.3)                                              (2.6)
if N(−ỹ, 1) ≥ n/2 then
  Compute x by (2.2) with s = S(−ỹ, 1)
  r := Aᵀx − c;   s = S(r, γ)
else
  Compute y by (2.5)
  Compute x by (2.2) with s = S(−y, τ_{n/2})
  s̃ = s;   r := Aᵀx − c;   s = S(r, γ)
  if N(r, γ) < n/2 then
    s := s + s̃;   where |s_i| > 1,  s_i := ½ s_i
    Compute x by (2.2)
    r := Aᵀx − c;   s = S(r, γ)
  end
end
```

In the inner modification of the algorithm an equation is considered active only if both $\boldsymbol{S}(-\boldsymbol{y}, \tau_{n/2})$ and $\boldsymbol{S}(\mathbf{r}(\boldsymbol{x}), \gamma)$ agree on this or if they show opposite sign.

Compared with the other strategies we found that (2.6) generally gave the smallest number of iterations with the algorithm of Section 2.2. The dominant part of the computation is one factorization of $\boldsymbol{AWA}^T + \gamma \boldsymbol{I}$ and 2 or 3 updates (or refactorizations), cf. Section 3.3.

2.2. Huber Solution

The Huber solution $\mathbf{x}_\gamma$ minimizes F, (1.3). The algorithm for computing $\mathbf{x}_\gamma$ is described elsewhere, [6], [13]. We shall briefly repeat it here so that we can stress some features that are special for the present application.

Given $\boldsymbol{x}$ and the corresponding $\boldsymbol{r}$, $\boldsymbol{s}$, $\boldsymbol{W}$. The Newton step aiming at making $\boldsymbol{g} = \boldsymbol{0}$ is determined by

$$(\boldsymbol{AWA}^T + \gamma \boldsymbol{I})\boldsymbol{h} = -\gamma \boldsymbol{g} \ . \tag{2.7}$$

Note, that the coefficient matrix is symmetric and positive definite. This implies that $\boldsymbol{h}$ exists and is unique.

The algorithm is given below. If $\boldsymbol{s}(\boldsymbol{x}+\boldsymbol{h}) = \boldsymbol{s}(\boldsymbol{x})$, then we have found the final active set (see [6]), and the Huber solution is $\mathbf{x}_\gamma = \mathbf{x} + \boldsymbol{h}$. We also stop the iteration if the computed gradient may be dominated by rounding

errors: More specifically, if

$$\|\boldsymbol{g}\|_\infty \le \varepsilon_M (\|\boldsymbol{A}\|_\infty + \|\mathbf{x}\|_\infty) \equiv \eta \ . \tag{2.8}$$

Algorithm Huber

```
k := 0
Get starting point x and corresponding r and s      {Section 2.1}
while  k < n  do
  k := k+1
  Compute g by (1.5)
  if ‖g‖∞ ≤ η  then   stop := true
  else
    Find h from (2.7)
    stop := (s(x+h) = s(x))
  end
  if stop then
    Compute y                                         {Section 2.3}
    return
  else
    x := x + t*h                          {Line search; Section 3.4}
  end
end
Error return
```

In the computation we also take rounding errors into account by modifying the "sign vector" $\boldsymbol{s}(\boldsymbol{x})$ to $\widehat{s}(\boldsymbol{x})$ defined by

$$\widehat{s}_i = \begin{cases} 0 & \text{if } |r_i| \le \gamma - \tau \\ \frac{1}{4}\operatorname{sign}(r_i) & \text{if } \gamma - \tau < |r_i| \le \gamma \\ \operatorname{sign}(r_i) & \text{if } |r_i| > \gamma \end{cases} \ . \tag{2.9a}$$

Here,

$$\tau = \varepsilon_M \cdot \min\{n, 10\} \cdot (\|\boldsymbol{A}\|_1 \|\mathbf{x}\|_\infty + \|\boldsymbol{c}\|_\infty) \tag{2.9b}$$

is an estimate of the effect of rounding errors during the computation of $\mathbf{r}$, (1.3b). The corresponding active set is defined by

$$w_i = \begin{cases} 1 & \text{if} \quad |\widehat{s}_i| < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.9c}$$

We consider $\boldsymbol{s}(\boldsymbol{x}+\boldsymbol{h}) = \boldsymbol{s}(\boldsymbol{x})$ if $\|\widehat{s}(\boldsymbol{x}+\boldsymbol{h}) - \widehat{s}(\boldsymbol{x})\|_\infty \le 0.75$.

The work is dominated by solution of the system (2.7) in each iteration. For large values of n the number of changes in active set between two consecutive iterations often is small compared with n. In Section 3.3 we show how this can be exploited so that the determination of each Newton step is an $O(n^2)$ process.

2.3. Solution to the QP Problem

When the Huber algorithm stops, we have the information needed to compute the solution to the QP–problem,

$$\boldsymbol{y} = -\left(\frac{1}{\gamma}\boldsymbol{W}(\mathbf{x})\boldsymbol{r}(\mathbf{x}) + \boldsymbol{s}(\mathbf{x})\right) , \tag{2.10}$$

where $\mathbf{x}=\mathbf{x}_\gamma$, cf. (1.6). The corresponding value of the objective function is

$$q(\boldsymbol{y}) = \frac{1}{2}\boldsymbol{y}^T\left(\boldsymbol{H}\boldsymbol{y} - 2\boldsymbol{c}\right) = \frac{1}{2}\boldsymbol{y}^T\left(\mathbf{A}^T\mathbf{A}\boldsymbol{y} + \gamma\boldsymbol{y} - 2\boldsymbol{c}\right) . \tag{2.11}$$

Exploiting (1.6), this can be written in the simpler form

$$q(\boldsymbol{y}) = \frac{1}{2}\mathbf{x}^T\mathbf{x} + \frac{\gamma}{2}\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{c}^T\boldsymbol{y} .$$

The optimal value of q is always non positive, however, so with the latter formulation there is risk of severe cancellation error. Therefore we use the more expensive formulation (2.11).

If the value of the gap function (1.7) is too big,

$$F(\mathbf{x}) + q(\boldsymbol{y}) > \min\{n, 20\} \cdot \varepsilon_M \cdot F(\mathbf{x}) , \tag{2.12}$$

then we refine the solution

$$\begin{aligned} &\text{refactorize} \;\; \boldsymbol{A}\boldsymbol{W}\boldsymbol{A}^T + \gamma\boldsymbol{I} , \\ &\boldsymbol{h} = \gamma(\boldsymbol{A}\boldsymbol{W}\boldsymbol{A}^T + \gamma\boldsymbol{I})^{-1}\boldsymbol{g} , \\ &\mathbf{x} := \mathbf{x} - \boldsymbol{h}; \quad \mathbf{r} := \mathbf{r} - \boldsymbol{A}^T\boldsymbol{h} , \end{aligned} \tag{2.13}$$

and use $\mathbf{x}$, $\mathbf{r}$ and the corresponding $\boldsymbol{s}$ to restart the Huber iteration. This restart is allowed once, only. The condition (2.12) is rarely satisfied, but when it is, this refinement drastically improves the accuracy of $\boldsymbol{y}$.

3. Auxiliary Algorithms

In the presentation of these algorithms we shall use a Matlab® inspired notation. For instance $\boldsymbol{A}_{1:k,p}$ and $\boldsymbol{A}_{r,q:n}$ denote the submatrices

$$\boldsymbol{A}_{1:k,p} = \begin{bmatrix} a_{1,p} \\ \vdots \\ a_{k,p} \end{bmatrix} \quad \text{and} \quad \boldsymbol{A}_{r,q:n} = \begin{bmatrix} a_{r,q} & \cdots & a_{q,n} \end{bmatrix} .$$

Further, all vectors are defined as column vectors, and $\boldsymbol{z} = [\mathbf{x};\boldsymbol{y}]$ means $\mathbf{x}$ extended by $\boldsymbol{y}$; i.e. $\boldsymbol{z} = \begin{bmatrix} \mathbf{x} \\ \boldsymbol{y} \end{bmatrix}$.

3.1. Compute γ

The shift parameter γ in (1.2) should be smaller than the smallest eigenvalue of $\boldsymbol{H}=\boldsymbol{R}^T\boldsymbol{R}$. We choose it as

$$\gamma = f \cdot \overline{\lambda}_1 , \tag{3.1}$$

where $0<f<1$, and $\overline{\lambda}_1$ is an estimate of the smallest eigenvalue of $\boldsymbol{H}$. In Section 4.3 we give experimental results about the choice of f. They show that generally $f=0.5$ is a good choice.

To explain our algorithm for computing $\overline{\lambda}_1$ we introduce the singular value decomposition (SVD) for $\boldsymbol{R}$,

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \quad \Longleftrightarrow \quad \boldsymbol{R}^T = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T , \tag{3.2a}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal, and

$$\boldsymbol{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n) \quad \text{with} \quad \sigma_1 \geq \ldots \geq \sigma_n > 0 . \tag{3.2b}$$

Equations (2.1) and (3.2a–b) imply

$$\boldsymbol{H} = \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T, \quad \text{i.e.} \quad \lambda_1 = \sigma_n^2 . \tag{3.2c}$$

The estimate $\overline{\lambda}_1$ is computed in two major steps:

1° Compute $(\mathbf{x},\boldsymbol{y})$: estimates of $(\mathbf{u}_n, \sigma_n^{-1}\mathbf{v}_n)$.

2° Refine the estimate by one step of simultaneous inverse iterations.

The pair $(\mathbf{x},\boldsymbol{y})$ is computed by using ideas from some well-known condition estimators, [4]: They satisfy

$$\boldsymbol{R}^T\boldsymbol{y} = \mathbf{x} \quad \text{with} \quad \|\mathbf{x}\|_2 = 1 \;, \tag{3.3}$$

and are constructed with the aim of maximizing $\|\boldsymbol{y}\|_2$: Assume that we have computed $\mathbf{x}^{(k-1)}_{1:k-1}$ and $\boldsymbol{y}^{(k-1)}_{1:k-1}$ with $\|\mathbf{x}^{(k-1)}_{1:k-1}\|_2 = 1$. We let

$$\mathbf{x}^{(k)}_{1:k} = [s_k\mathbf{x}^{(k-1)}_{1:k-1};\; c_k] \quad \text{with} \quad c_k^2 + s_k^2 = 1 \;.$$

Then, also $\|\mathbf{x}^{(k)}_{1:k}\|_2 = 1$, and (c_k, s_k) is chosen to maximize

$$\|\boldsymbol{y}^{(k)}_{1:k}\|_2 = \|[s_k\boldsymbol{y}^{(k-1)}_{1:k-1};\; y^{(k)}_k]\|_2 \;.$$

Now, the kth equation in (3.3) has the form

$$r_{kk}y^{(k)}_k = c_k - s_k b_k \quad \text{with} \quad b_k = \boldsymbol{R}^T_{1:k-1,k}\boldsymbol{y}^{(k-1)}_{1:k-1} \;,$$

and it follows that

$$\begin{aligned} N_k \equiv \|\boldsymbol{y}^{(k)}_{1:k}\|_2^2 &= s_k^2 N_{k-1} + \Big(\frac{c_k - s_k b_k}{r_{kk}}\Big)^2 \\ &= \frac{1}{r_{kk}^2}\big(c_k^2 - 2c_k s_k b_k + s_k^2(b_k^2 + r_{kk}^2 N_{k-1}\big) \;. \end{aligned}$$

It can be shown that N_k is maximized by chosing

$$c_k = \sqrt{0.5 - t_k}, \qquad s_k = -\mathrm{sign}(b_k)\sqrt{0.5 + t_k} \;, \tag{3.4a}$$

where

$$t_k = \frac{d_k}{2\sqrt{4b_k^2 + d_k^2}} \quad \text{with} \quad d_k = b_k^2 + r_{kk}^2 N_{k-1} - 1 \;. \tag{3.4b}$$

In the special case $b_k = 0$ we let $(c_k, s_k) = (0,1)$ if $d_k \geq 0$; otherwise $(c_k, s_k) = (1,0)$. In the code the condition $b_k = 0$ is replaced by $\mathsf{fl}(d_k^2 + 4b_k^2) = \mathsf{fl}(d_k^2)$.

In terms of the SVD (3.2) we can write

$$\mathbf{x} = \sum_{j=1}^{n} \alpha_j \mathbf{u}_j \;, \quad \boldsymbol{y} = \sum_{j=1}^{n} \alpha_j \sigma_j^{-1} \mathbf{v}_j \;, \tag{3.5}$$

where we have tried to make $\alpha_n \mathbf{u}_n$ the dominant term in $\mathbf{x}$, and $\alpha_j\sigma_j^{-1}\mathbf{v}_j$ even more dominant in $\boldsymbol{y}$.

The refinement in step 2° above involves the computation of

$$\begin{aligned} &\boldsymbol{g} = \boldsymbol{R}^{-1}\boldsymbol{y} \;, \\ &\boldsymbol{h} = \boldsymbol{g} - (\mathbf{x}^T\boldsymbol{g})\mathbf{x}, \quad \boldsymbol{w} = \boldsymbol{h}/\|\boldsymbol{h}\|_2 \;, \\ &\boldsymbol{z} = \boldsymbol{R}^{-T}\boldsymbol{w} \;. \end{aligned} \tag{3.6a}$$

Note that $\boldsymbol{g} = \boldsymbol{R}^{-1}\boldsymbol{R}^{-T}\mathbf{x} = \boldsymbol{H}^{-1}\mathbf{x}$, and that $(\mathbf{a}_1, \mathbf{a}_2) = (\boldsymbol{y}, \boldsymbol{z})$ are solutions to $\boldsymbol{R}^T\mathbf{a}_k = \boldsymbol{b}_k$ with orthonormal right hand sides $\boldsymbol{b}_1 = \mathbf{x},\; \boldsymbol{b}_2 = \boldsymbol{w}$.

Now, the estimate is found as

$$\overline{\lambda}_1 = \overline{\sigma}_n^2, \quad \text{where} \quad \overline{\sigma}_n = \min_{\alpha,\beta\in\mathbb{R}} \Big\{\frac{\|\alpha\boldsymbol{y} + \beta\mathbf{z}\|_2}{\|\alpha\mathbf{x} + \beta\mathbf{w}\|_2}\Big\} \;. \tag{3.6b}$$

A little algebra shows that

$$\overline{\lambda}_1 = \frac{1}{\max\{\lambda_j(\boldsymbol{B})\}}, \quad \text{where} \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{y}^T\boldsymbol{y} & \boldsymbol{y}^T\boldsymbol{z} \\ \boldsymbol{y}^T\mathbf{z} & \boldsymbol{z}^T\mathbf{z} \end{bmatrix} \;. \tag{3.6c}$$

The total cost of this estimator is about $4n^2$ flops. In Section 4.2 we give experimental results about the accuracy. Note, that we do not need a high accuracy: We use

$$\gamma = f \cdot \overline{\lambda}_1 = f\cdot(\lambda_1 + \delta) \quad \Longleftrightarrow \quad \lambda_1 - \gamma = (1{-}f)\lambda_1 - f\delta \;.$$

This is positive if $\delta < \dfrac{1-f}{f}\lambda_1$. We recommend to use $f = 0.5$. In this case it suffices to estimate λ_1 with a relative error less than 100%.

At the end of Section 3.2 we describe a "safety valve": If the matrix $\boldsymbol{H}{-}\gamma\boldsymbol{I}$ is found to be indefinite – i.e. $\gamma \geq \lambda_1$ – then we replace γ by $\gamma := 0.1{\cdot}\gamma$.

3.2. Compute A

The matrix $\boldsymbol{A}$ in the Huber function is defined by

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{R}^T\boldsymbol{R} - \gamma\boldsymbol{I} \;, \tag{3.7}$$

where $\boldsymbol{R}$ is upper triangular, cf. (2.1). We choose also to let $\boldsymbol{A}$ have this property. We might compute it simply as the upper triangular Cholesky factor of $\boldsymbol{H} - \gamma\boldsymbol{I}$. If, however, the problem is given directly by $\boldsymbol{R}$, this approach would lead to unnecessary loss of accuracy. Instead we compute

$\boldsymbol{A}$ via orthogonal transformation: From (3.7) we see that

$$\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{A}^T\boldsymbol{A} + \gamma\boldsymbol{I} \ , \tag{3.8a}$$

which is equivalent with

$$\begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{bmatrix} = \boldsymbol{Q}\begin{bmatrix} \boldsymbol{A} \\ \sqrt{\gamma}\boldsymbol{I} \end{bmatrix} \ , \tag{3.8b}$$

where $\boldsymbol{Q}$ is orthogonal. The transformation is computed via a series of Householder reflections. Before the kth transformation the partly transformed matrix has the structure shown in figure 3.1 for $n=5$, $k=3$.

$$\left[\begin{array}{ccccc} * & * & * & * & * \\ & * & * & * & * \\ & & \star & \star & \star \\ & & & \star & \star \\ & & & & \star \\ & & \star & \star & \star \\ & & \star & \star & \star \\ & & \star & & \\ & & & \star & \\ & & & & \star \end{array}\right] \begin{array}{l} \left.\vphantom{\begin{array}{c} * \\ * \end{array}}\right\} \text{Part of } \boldsymbol{R} \ \ (\text{overwritten by } \boldsymbol{A}) \\ \left.\vphantom{\begin{array}{c} * \\ * \\ * \end{array}}\right\} \text{Unknown part of } \boldsymbol{A} \\ \left.\vphantom{\begin{array}{c} * \\ * \\ * \\ * \\ * \end{array}}\right\} \boldsymbol{B} = \text{Partly transformed } \sqrt{\gamma}\boldsymbol{I} \end{array}$$

Figure 3.1

The kth reflection involves row k of $\boldsymbol{A}$ and $\boldsymbol{R}$ and rows $1,\ldots,k$ of $\boldsymbol{B}$. Let

$$\mathbf{x}_k = \begin{bmatrix} a_{kk} \\ \boldsymbol{B}_{1:k,k} \end{bmatrix} \quad \text{and} \quad \widetilde{\mathbf{x}}_k = \begin{bmatrix} r_{kk} \\ \boldsymbol{0} \end{bmatrix} \tag{3.9a}$$

denote the active parts of the kth column of the two matrices in (3.8b). The condition $\|\mathbf{x}_k\|_2 = \|\widetilde{\mathbf{x}}_k\|_2$ implies

$$a_{kk}^2 = r_{kk}^2 - \|\boldsymbol{B}_{1:k,k}\|_2^2 \ .$$

If the right-hand side is not significantly positive, we give an error return, indicating that γ has been chosen too large. If the condition

$$r_{kk}^2 - \|\boldsymbol{B}_{1:k,k}\|_2^2 > \left(\min\{2k, 20\}\cdot\varepsilon_M\cdot r_{kk}\right)^2 \tag{3.9b}$$

is satisfied, then we proceed by letting

$$a_{kk} = -\mathrm{sign}(r_{kk})\sqrt{r_{kk}^2 - \|\boldsymbol{B}_{1:k,k}\|_2^2} \ . \tag{3.9c}$$

Further,

$$\boldsymbol{Q}_k = \boldsymbol{I} - \frac{1}{\beta_k}\mathbf{w}_k\mathbf{w}_k^T \quad \text{with} \quad \begin{cases} \mathbf{w}_k = \mathbf{x}_k - \widetilde{\mathbf{x}}_k \\ \beta_k = -r_{kk}(a_{kk} - r_{kk}) \end{cases} \tag{3.9d}$$

is an orthogonal matrix that gives $\boldsymbol{Q}_k\mathbf{x}_k = \widetilde{\mathbf{x}}_k$.

Now, for $j \neq k$ let $\boldsymbol{y}_j = \begin{bmatrix} a_{kj} \\ \boldsymbol{B}_{1:k,j} \end{bmatrix}$ and $\widetilde{\boldsymbol{y}}_j = \boldsymbol{Q}_k\boldsymbol{y}_j$. For $j<k$ both $\boldsymbol{y}_j$ and $\widetilde{\boldsymbol{y}}_j$ are zero. For $j=k+1,\ldots,n$ we find

$$\widetilde{\boldsymbol{y}}_j = \boldsymbol{y}_j - \left(\mathbf{w}_k^T\boldsymbol{y}_j/\beta_k\right)\mathbf{w}_k \ .$$

The first element of $\widetilde{\boldsymbol{y}}_j$ should be equal to r_{kj}, leading to the condition

$$r_{kj} = a_{kj} - \frac{(a_{kk} - r_{kk})a_{kj} + h_{kj}}{-r_{kk}(a_{kk} - r_{kk})}(a_{kk} - r_{kk}) \ , \tag{3.10a}$$

where

$$h_{kj} = \boldsymbol{B}_{1:k,k}^T\boldsymbol{B}_{1:k,j} \ . \tag{3.10b}$$

Thus, we get

$$a_{kj} = (r_{kk}r_{kj} - h_{kj})/a_{kk} \ , \tag{3.11a}$$

and

$$\widetilde{\boldsymbol{B}}_{1:k,j} = \boldsymbol{B}_{1:k,j} - d_{kj}\boldsymbol{B}_{1:k,k} \ , \tag{3.11b}$$

where

$$d_{kj} = \left((a_{kk} - r_{kk})a_{kj} + h_{kj}\right)/\beta_k \ . \tag{3.11c}$$

The total cost of the transformation (3.8b) is about $\frac{2}{3}n^3$ flops. This is the same as if we computed the upper triangle of $\boldsymbol{R}^T\boldsymbol{R} - \gamma\boldsymbol{I}$ ($\frac{1}{3}n^3$ flops) followed by Cholesky factorization (also $\frac{1}{3}n^3$ flops).

If, for some $k \le n$ the condition (3.9b) is not satisfied, then the process is repeated with $\gamma := 0.1\gamma$. If this also fails, then the QP–algorithm gives an error return: The problem is too ill conditioned.

3.3. Factorization

During the iterations for computing the Huber solution we have to solve problems of the form

$$(\boldsymbol{A}\boldsymbol{W}\boldsymbol{A}^T + \gamma\boldsymbol{I})\boldsymbol{h} = -\gamma\boldsymbol{g} \ , \tag{3.12}$$

where $\boldsymbol{W} = \mathrm{diag}(w_i)$ with $w_i \in \{0,1\}$; i.e. we can write

$$\boldsymbol{A}\boldsymbol{W}\boldsymbol{A}^T = \boldsymbol{A}_{\mathcal{A}}\boldsymbol{A}_{\mathcal{A}}^T \ ,$$

where $\boldsymbol{A}_{\mathcal{A}}$ consists of the columns $\{\boldsymbol{A}_{:,j}\}$ with $j \in \mathcal{A}$, the active set. Between iterations there is usually only a few changes in $\mathcal{A}$, and we are interested in a cheap (but accurate) updating of a triangular factorization of the coefficient matrix in (3.12).

We have chosen to use an untraditional factorization, viz.

$$\boldsymbol{A}\boldsymbol{W}\boldsymbol{A}^T + \gamma\boldsymbol{I} = \boldsymbol{L}^T\boldsymbol{L} \ , \tag{3.13}$$

where $\boldsymbol{L}$ is a lower triangular matrix. This implies that the solution of (3.12) is done by a back substitution followed by a forward substitution.

This choice is made because it leads to simple updatings: Let the active set be augmented by $\mathcal{E}$ (for "enter"). Then

$$\widetilde{\boldsymbol{L}}^T\widetilde{\boldsymbol{L}} = \boldsymbol{L}^T\boldsymbol{L} + \boldsymbol{A}_{\mathcal{E}}\boldsymbol{A}_{\mathcal{E}}^T = \begin{bmatrix} \boldsymbol{L}^T & \boldsymbol{A}_{\mathcal{E}} \end{bmatrix} \begin{bmatrix} \boldsymbol{L} \\ \boldsymbol{A}_{\mathcal{E}}^T \end{bmatrix} \ .$$

This shows that we can compute $\widetilde{\boldsymbol{L}}$ by an orthogonal transformation

$$\begin{bmatrix} \widetilde{\boldsymbol{L}} \\ \boldsymbol{0} \end{bmatrix} = \boldsymbol{Q} \begin{bmatrix} \boldsymbol{L} \\ \boldsymbol{A}_{\mathcal{E}}^T \end{bmatrix} \ . \tag{3.14}$$

The structure of the rightmost matrix is shown below for $n = 8$ and columns 2 and 5 entering the active set

$$\begin{bmatrix} \star \\ \star\star \\ \star\star\star \\ \star\star\star\star \\ \star\star\star\star\star \\ \star\star\star\star\star\star \\ \star\star\star\star\star\star\star \\ \star\star\star\star\star\star\star\star \\ \star\star \\ \star\star\star\star\star \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}1\\1\\1\\1\\1\\1\\1\\1\end{matrix}}\right\} \boldsymbol{L} \\ \left.\vphantom{\begin{matrix}1\\1\end{matrix}}\right\} \boldsymbol{A}_{\mathcal{E}}^T \end{matrix}$$

Figure 3.2

As in Section 3.2 the transformation is done by a series of Householder reflections. Now, however, we start from $k = n$ and go back.

For the example shown in figure 3.2 the first change occurs for $k = 5$. For $k = 5, 4, 3$ the transformations involve row k of $\boldsymbol{L}$ and the last row of $\boldsymbol{A}_{\mathcal{E}}^T$. For $k = 2, 1$ the kth row of $\boldsymbol{L}$ and both rows of $\boldsymbol{A}_{\mathcal{E}}^T$ are involved.

We use standard Householder reflections, and a short presentation of the algorithm is provided by the following MATLAB function `upqr`.


```
function L = upqr(Lo,A,i)
   n = size(Lo,1);   L = Lo;   q = length(i);
   i = sort(i(:));   ii = [0; i];
   B = A(:,i)';   w = zeros(q,1);
   for  r = q : -1 : 1
     for  k = ii(r+1) : -1 : ii(r)+1
       w(r:q) = B(r:q,k);   rk = L(k,k);
       L(k,k) = -sign(rk)*sqrt(rk^2 + norm(w(r:q))^2);
       if  k > 1    %  Transform leading columns
         wk = rk - L(k,k);   beta = -L(k,k)*wk;   k1 = k - 1;
         d = (wk*L(k,1:k1) + w(r:q)'*B(r:q,1:k1))/beta;
         L(k,1:k1) = L(k,1:k1) - d*wk;
         B(r:q,1:k1) = B(r:q,1:k1) - w(r:q)*d;
       end  % Transform
     end  %  Add next column of  A
   end  %  Columns of  A
```

Function for updating factorization. `Lo` *and* `L` *represent* $\boldsymbol{L}$ *and* $\widetilde{\boldsymbol{L}}$*, respectively, and the vector* `i` *represents* $\mathcal{E}$*.*

The changes in active set can also imply that columns of $\boldsymbol{A}$ leave (3.13). Therefore we are also interested in downdating the factorization; i.e. to compute

$$\widetilde{\boldsymbol{L}}^T\widetilde{\boldsymbol{L}} = \boldsymbol{L}^T\boldsymbol{L} - \boldsymbol{A}_{\mathcal{L}}\boldsymbol{A}_{\mathcal{L}}^T \ . \tag{3.15}$$

This can be done as shown in the following MATLAB function `ddqr`, where we combine ideas from Section 3.2 and algorithm `upqr`. A value of `r` larger than zero indicates that severe loss of accuracy has occured – remember that the columns of $\boldsymbol{A}$ that are removed have contributed to the current $\boldsymbol{L}$.

Typically a change in active set involves that some columns of $\boldsymbol{A}$ leave and some enter. This means that we seek

$$\widetilde{\boldsymbol{L}}^T\widetilde{\boldsymbol{L}} = \boldsymbol{L}^T\boldsymbol{L} - \boldsymbol{A}_{\mathcal{L}}\boldsymbol{A}_{\mathcal{L}}^T + \boldsymbol{A}_{\mathcal{E}}\boldsymbol{A}_{\mathcal{E}}^T \ . \tag{3.16}$$

The transformation can be made in two steps as indicated in the following MATLAB function `updfac`.

For both `upqr` and `ddqr` the cost of entering (deleting) m columns to (from) the active set is between $\frac{2}{3}m^3$ flops (the first m columns) and $2mn^2$ flops (the last m columns). Thus, in the typical case, where $m_{\mathcal{E}} + m_{\mathcal{L}} \ll n$, the cost of adjusting the factorization is $O(n^2)$ flops.

```
function [L,r] = ddqr(Lo,A,o)
  n = size(Lo,1);   m = length(o);
  o = sort(o(:));   oo = [0; o];
  L = Lo;   B = A(:,o)';   w = zeros(m,1);   r = 0;
  for  t = m : -1 : 1
    for  k = oo(t+1) : -1 : oo(t)+1
      w(t:m) = B(t:m,k);   nw = norm(w);
      rk = L(k,k);   lk2 = (rk + nw)*(rk - nw);
      if  lk2 < (min(2*k,20)*eps*rk)^2
        r = k+1;  return
      end
      L(k,k) = -sign(rk)*sqrt(lk2);
      if  k > 1    %  Transform leading columns
        wk = L(k,k) - rk;   beta = -rk*wk;
        e = w(t:m)'*B(t:m,1:k-1);
        L(k,1:k-1) = (rk*L(k,1:k-1) - e)/L(k,k);
        d = (L(k,1:k-1)*wk + e)/beta;
        B(t:m,1:k-1) = B(t:m,1:k-1) - w(t:m)*d;
      end  % Transform
    end  %  Subtract next column of  A
  end  %  Leaving columns of  A
```

Function for downdating factorization, (3.15). `Lo`, `L`, `o` *represent* $\boldsymbol{L}$, $\widetilde{\boldsymbol{L}}$ *and* $\mathcal{L}$, *respectively, and* $\mathtt{r}=0$ *indicates success.*

```
function L = updfac(Lo,so, A,s, gamma)
  n = size(Lo,1);   d = s.^2 - so.^2;
  o = find(d == 1);
  if  length(o)    %  Columns leave
    [L,r] = ddqr(Lo,A,o);
    if  r > 0    %  Refactorize
      a = find(abs(s) < 1);    % New active set
      if  length(a),  L = upqr(sqrt(gamma)*eye(n,n),A,a);
      else,  L = sqrt(gamma)*eye(n,n);  end
      return
    end
  else,  L = Lo; end
  i = find(d == -1);
  if  length(i)    %  Columns enter
    L = upqr(L,A,i);
  end
```

Function for adjusting factorization, (3.16). `so` *and* `s` *represent "old" and "new" active set, respectively, and* `Lo` *and* `L` *hold the corresponding triangular factors.*

3.4. Line Search

Given $\boldsymbol{x}$ and $\boldsymbol{r}=\boldsymbol{r}(\boldsymbol{x})$ and a search direction $\boldsymbol{h}$, that satisfies

$$(\boldsymbol{A}\boldsymbol{W}\boldsymbol{A}^T+\gamma\boldsymbol{I})\boldsymbol{h}=-\gamma\boldsymbol{g}\ , \quad \text{with} \quad \boldsymbol{g}=\boldsymbol{A}(\frac{1}{\gamma}\boldsymbol{W}\boldsymbol{r}+\boldsymbol{s})+\boldsymbol{x}\ . \tag{3.17}$$

We seek the minimum of the function

$$\varphi(t)=F(\boldsymbol{x}+t\boldsymbol{h})=c_0(t)+c_1(t)\cdot t+\frac{1}{2}c_2(t)\cdot t^2\ , \tag{3.18a}$$

where

$$c_0(t)=\frac{1}{2\gamma}\boldsymbol{r}^T\boldsymbol{W}\boldsymbol{r}+\boldsymbol{s}^T(\boldsymbol{r}-\frac{\gamma}{2}\boldsymbol{s})+\frac{1}{2}\boldsymbol{x}^T\boldsymbol{x}\ , \tag{3.18b}$$

$$c_1(t)=\boldsymbol{h}^T\left(\boldsymbol{A}(\frac{1}{\gamma}\boldsymbol{W}\boldsymbol{r}+\boldsymbol{s})+\boldsymbol{x}\right)\ , \tag{3.18c}$$

$$c_2(t)=\boldsymbol{h}^T(\frac{1}{\gamma}\boldsymbol{A}\boldsymbol{W}\boldsymbol{A}^T+\boldsymbol{I})\boldsymbol{h}\ . \tag{3.18d}$$

The coefficients depend on $\boldsymbol{s}$ and $\boldsymbol{W}$; i.e. on t since the residual $\boldsymbol{A}^T(\boldsymbol{x}+t\boldsymbol{h})-\boldsymbol{c}=\boldsymbol{r}+t\boldsymbol{A}^T\boldsymbol{h}$ varies with t. The coefficients are piecewise constant, and change at the *kink values*, where one or more residual components pass the threshold γ. The kink values are the positive α_j–values defined by

$$|r_k+\alpha_j\cdot(\boldsymbol{A}^T\boldsymbol{h})_k|=\gamma \quad \text{for some} \quad k=k_j\ . \tag{3.19}$$

We assume that the kink values are ordered so that

$$0<\alpha_1\le\alpha_2\le\cdots\le\alpha_K\ . \tag{3.20}$$

We seek t^*, the minimizer of φ. It satisfies $\varphi'(t^*)=0$. The function φ' is continuous and a piecewise first order polynomial, which we shall write in the form

$$\varphi'(t)=c_{1,j}+c_{2,j}\cdot(t-\alpha_{j-1}) \quad \text{for} \quad \alpha_{j-1}<t\le\alpha_j \tag{3.21a}$$

(with $\alpha_0=0$). From the continuity it follows that

$$c_{1,j+1}=c_{1,j}+c_{2,j}\cdot(\alpha_j-\alpha_{j-1})\ , \tag{3.21b}$$

and from (3.18) we see that

$$c_{2,j+1}=c_{2,j}\pm\frac{1}{\gamma}\big((\boldsymbol{A}\boldsymbol{h})_k\big)^2\ , \tag{3.22}$$

where $+$ $(-)$ is used if $\boldsymbol{r}_k$ enters (leaves) the active set at α_j. The initial values are

$$c_{1,1} = c_1(0) = \boldsymbol{h}^T\boldsymbol{g}\ , \qquad c_{2,1} = c_2(0) = -\boldsymbol{h}^T\boldsymbol{g}\ . \tag{3.22d}$$

In (3.20) the number of kink values is bounded: $K \le 2n$. Experience shows, however, that we get faster convergence of the Huber iteration (Section 2.2) by restricting the search so that we allow at most 20 changes in active set.

4. Testing

The algorithm has been tested on a large number of problems generated as described in Section 4.1. In Section 4.2 we give results for the algoritthm from Section 3.1 to compute estimates $\overline{\lambda}_1$. In Section 4.3 we give results for the influence of the factor f in $\gamma = f \cdot \overline{\lambda}_1$. In Section 4.4 we present two competing algorithms, and finally, in Section 4.5 we give computational results.

4.1. Test Problems

Our generator for test problems is similar to the generator introduced by Moré and Toraldo in [12], and also used in [1], [5], [8], [10]. The matrices have the form

$$\boldsymbol{H} = \boldsymbol{M}^T\boldsymbol{M} \quad \text{with} \quad \boldsymbol{M} = \boldsymbol{D}^{\frac{1}{2}}\boldsymbol{Z}\ , \tag{4.1a}$$

where

$$\boldsymbol{D} = 10^{-desc}\mathrm{diag}(d_1, \ldots, d_n) \quad \text{with} \quad \log_{10} d_i = \frac{i-1}{n-1} ncond\ . \tag{4.1b}$$

Here *desc* and *ncond* are prescribed numbers. The original Moré–Toraldo generator corresponds to $desc = 0$. Further, $\boldsymbol{Z}$ is a Householder matrix,

$$\boldsymbol{Z} = \boldsymbol{I} - \frac{2}{\boldsymbol{z}^T\boldsymbol{z}}\boldsymbol{z}\boldsymbol{z}^T \tag{4.1c}$$

where $\boldsymbol{z} \in I\!R^n$ has elements that are uniform random in]-1,1[. Since $\boldsymbol{Z}$ is orthogonal, it follows that the condition number $\kappa_2(\boldsymbol{H}) = \kappa_2(\boldsymbol{D}) = 10^{ncond}$.

A little algebra shows that

$$\boldsymbol{H} = \boldsymbol{D} - 2\big(\boldsymbol{u}\boldsymbol{v}^T + \boldsymbol{v}\boldsymbol{u}^T\big), \tag{4.1d}$$

where

$$\boldsymbol{u} = \boldsymbol{z}/\|\boldsymbol{z}\|_2, \quad \boldsymbol{v} = \boldsymbol{D}\boldsymbol{u} - (\boldsymbol{u}^T\boldsymbol{D}\boldsymbol{u})\boldsymbol{u}\ . \tag{4.1e}$$

The generator involves two more parameters:

nb Controls the number of "large" components in $\boldsymbol{y}^*$, i.e. $|y_i^*| = 1$.

deg Controls near–degeneracy: The non active residuals are computed as $r_i = s_i \cdot 10^{-\nu \cdot deg}$, where ν is uniform random in $[0, 1]$.

In MATLAB the code for generating $\boldsymbol{y}$ and $\boldsymbol{c}$ takes the form

```
for i = 1 :  n
   mu = 2*rand(1) - 1;
   if n * abs(mu) < nb
     y(i) = (-1)^i;  nu = rand(1);
     u(i) = y(i) * 10^(-nu*deg);
   else
     y(i) = mu;  u(i) = 0;
   end
end
c = H*y + u;
```
(4.2)

4.2. Estimation of γ

We have used the matrix generator (4.1) with $desc = 0$, $ncond = 1$, 6, 12 and $n = 5 : 5 : 200$. For each matrix the smallest eigenvalue was estimated by the algorithm of Section 3.1.

It follows from (4.1) that in all cases $\lambda_1(\boldsymbol{H}) = 10^{-desc} = 1$, so the relative error is equal to the absolute error $\overline{\lambda}_1 - 1$. In Figure 4.1 we show the results. We also give the critical error 33% corresponding to the choice $f = 0.75$, cf. the end of Section 3.1.

The error seems to be a decreasing function of n. Even for the ill–conditioned matrices with $\kappa_2(\boldsymbol{H}) = 10^{12}$ the error is sufficiently small for all $n \ge 40$ and $f = 0.75$. With $f = 0.5$ we can accept a relative error of 100%. This is satisfied in all cases.

In Figure 4.2 we illustrate the efficiency of the estimator: We give the number of flops needed to find the estimate. For comparison we also show

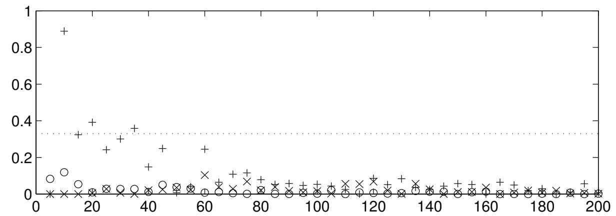

Figure 4.1. *Relative error in estimated* λ_1. *ncond* $= 1(\circ),\ 6(\times),\ 12(+)$

the number of flops used by the MATLAB function `eig`, which is a very good eigenvalue algorithm. This figure clearly illustrates the difference between an $O(n^2)$ and an $O(n^3)$ proces.

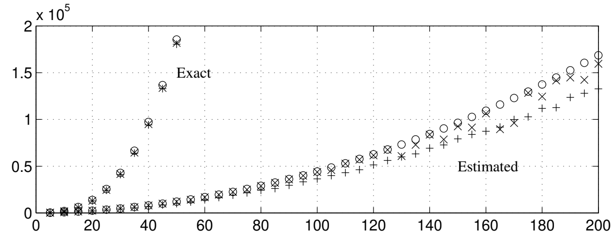

Figure 4.2. *Number of* flops *for estimated and "exact"* λ_1.
ncond $= 1$ (o), 6 (x), 12 (+)

4.3. Choice of f

The choice of the factor f in

$$\gamma = f \cdot \overline{\lambda}_1 \tag{4.3}$$

must consider several aspects:

1° The condition of the matrix $\boldsymbol{A}$ depends on γ:

$$\kappa_2^2(\boldsymbol{A}) = \frac{\max\{\lambda(\boldsymbol{A}^T\boldsymbol{A})\}}{\min\{\lambda(\boldsymbol{A}^T\boldsymbol{A})\}} = \frac{\max\{\lambda(\boldsymbol{H}) - \gamma\}}{\min\{\lambda(\boldsymbol{H}) - \gamma\}} \simeq \frac{1}{1-f}\kappa_2(\boldsymbol{H}) .$$

This shows that $f > 0$ should be chosen small.

2° For small γ it is more difficult to find the solution from a bad starting guess. This was shown in [6], confirmed in [7], [9], and – although not so clearly – in the present application of the Huber algorithm. Thus, $f < 1$ should be chosen as large as possible.

3° If f is chosen too large there is an increased risk that $f \cdot \overline{\lambda}_1 \geq \lambda_1$. Then $\boldsymbol{H} - \gamma\boldsymbol{I}$ is not positive definite, and the recomputation of $\boldsymbol{A}$ (see the end of Section 3.2) with $\gamma := 0.1\gamma$ implies a waste of $O(n^3)$ flops, and the new γ–value may be too small – cf. 2° above.

In Figure 4.3 we give results for problems generated as described in Section 4.1 with $deg = 3$, $nb = \frac{3}{4}n$ and two different choices of condition. Each problem is solved with three different values of f.

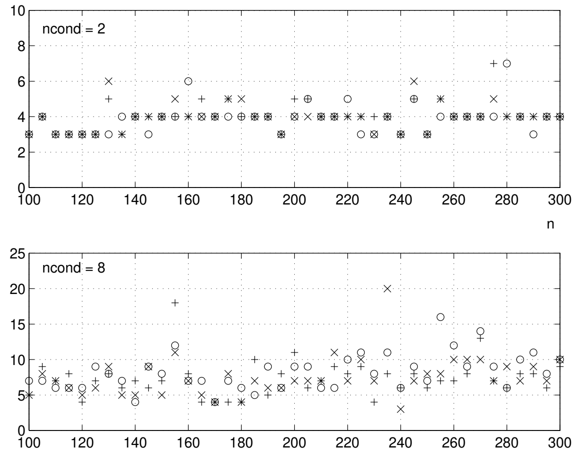

Figure 4.3. *Influence of* γ*-factor.* $f = .25$ (o), $.50$ (x), $.75$ (+)

In the well conditioned case ($ncond = 2$) the choice of f seems of no importance. For $ncond = 8$ the value of f can have a larger influence on the number of iterations. The best choice seems, however, to be as random as the test matrices.

We recommend to use $f = 0.5$, and this is the value chosen in the remainder of this report.

4.4. Competing Methods

First, we compare with an interior point method. As a typical example we take the primal-dual approach used by Han et al. [5]. They use the standard formulation

$$\begin{aligned} &\min\{f(\mathbf{x}) \equiv \tfrac{1}{2}\mathbf{x}^T\boldsymbol{H}\mathbf{x} + \boldsymbol{d}^T\mathbf{x}\} \\ &\text{subject to} \quad \boldsymbol{0} \le \mathbf{x} \le \mathbf{e} \;. \end{aligned} \tag{4.4}$$

This is equivalent with our formulation (1.1) when we set

$$\boldsymbol{d} = -\frac{1}{2}(\boldsymbol{c} + \boldsymbol{H}\mathbf{e}) \;, \tag{4.5a}$$

and

$$\boldsymbol{y} = 2\mathbf{x} - \mathbf{e}, \quad q(\boldsymbol{y}) = f(\mathbf{x}) + \frac{1}{2}\mathbf{e}^T(2\boldsymbol{c} + \boldsymbol{H}\mathbf{e}) \;. \tag{4.5b}$$

The method of Han et al. is derived from the reformulation

$$\begin{aligned} &\min\{f(\mathbf{x}) \equiv \tfrac{1}{2}\mathbf{x}^T\boldsymbol{H}\mathbf{x} + \boldsymbol{d}^T\mathbf{x}\} \\ &\text{subject to} \quad \mathbf{x} + \boldsymbol{z} = \mathbf{e} \quad \text{and} \quad \mathbf{x}, \boldsymbol{z} \ge \boldsymbol{0} \;, \end{aligned} \tag{4.6a}$$

and the dual

$$\begin{aligned} &\max\{-\mathbf{e}^T\mathbf{v} - \tfrac{1}{2}\mathbf{x}^T\boldsymbol{H}\mathbf{x}\} \\ &\text{subject to} \quad \mathbf{v} \ge \boldsymbol{0} \quad \text{and} \quad \boldsymbol{u} \equiv \boldsymbol{H}\mathbf{x} + \boldsymbol{d} + \mathbf{v} \ge \boldsymbol{0} \;. \end{aligned} \tag{4.6b}$$

The duality gap is

$$\Delta \equiv \mathbf{x}^T\boldsymbol{u} + \boldsymbol{z}^T\mathbf{v} \ge 0 \;. \tag{4.6c}$$

The solution is found by minimizing the potential function

$$\varphi(\mathbf{x}, \boldsymbol{z}, \boldsymbol{u}, \mathbf{v}) \equiv \rho \log(\Delta) - \sum_{i=1}^{n} \log(x_i u_i) - \sum_{i=1}^{n} \log(z_i v_i) \;, \tag{4.7}$$

where the scalar $\rho \ge 2n + \sqrt{2n}$.

Let $(\mathbf{x},\boldsymbol{z},\boldsymbol{u},\mathbf{v})$ denote the current iterate, and let $\boldsymbol{X},\boldsymbol{Z},\boldsymbol{U},\boldsymbol{V}$ denote the diagonal matrices with $(\boldsymbol{X})_{ii} = x_i$ etc. The next iterate is found as follows,

1° Find $\boldsymbol{h}_{\mathbf{X}}$ as the solution to

$$\begin{aligned} &(\boldsymbol{DHD} + \boldsymbol{UZ} + \boldsymbol{VX})(\boldsymbol{D}^{-1}\boldsymbol{h}_{\mathbf{X}}) \\ &\qquad = \tfrac{\Delta}{\rho}(\boldsymbol{D}\boldsymbol{X}^{-1} - \boldsymbol{D}\boldsymbol{z}^{-1})\mathbf{e} - \boldsymbol{D}(\boldsymbol{u} - \mathbf{v}) \end{aligned} \tag{4.8}$$

with $\boldsymbol{D} = \mathrm{diag}(\sqrt{x_i z_i})$, and compute

$$\boldsymbol{h}_{\mathbf{V}} = \boldsymbol{Z}^{-1}(\boldsymbol{V}\boldsymbol{h}_{\mathbf{X}} + \frac{\Delta}{\rho}\mathbf{e}) - \mathbf{v} \;.$$

2° Line search: Find $\overline{\theta}$, the largest value for which

$$\mathbf{x} + \theta\boldsymbol{h}_{\mathbf{X}} \ge \boldsymbol{0}, \quad \boldsymbol{z} - \theta\boldsymbol{h}_{\mathbf{X}} \ge \boldsymbol{0}, \quad \mathbf{v} + \theta\boldsymbol{h}_{\mathbf{V}} \ge \boldsymbol{0}$$
$$\text{and} \quad \boldsymbol{u} + \theta(\boldsymbol{H}\boldsymbol{h}_{\mathbf{X}} + \boldsymbol{h}_{\mathbf{V}}) \ge \boldsymbol{0} \;.$$

3° Update:

$$\begin{aligned} &\theta = \beta\overline{\theta} \\ &\mathbf{x} := \mathbf{x} + \theta\boldsymbol{h}_{\mathbf{X}}; \qquad \boldsymbol{z} := \mathbf{e} - \mathbf{x} \\ &\mathbf{v} := \mathbf{v} + \theta\boldsymbol{h}_{\mathbf{V}}; \qquad \boldsymbol{u} := \boldsymbol{H}\mathbf{x} + \boldsymbol{d} + \mathbf{v} \\ &\Delta := \mathbf{x}^T\boldsymbol{u} + \boldsymbol{z}^T\mathbf{v} \end{aligned}$$

The iteration is stopped when

$$\Delta \le \min\{\epsilon_1, \epsilon_2(1 + |f(\mathbf{x})|)\} \;. \tag{4.9a}$$

The algorithm involves four iteration parameters: ρ, β, ϵ_1 and ϵ_2. Han et al. [5] found that the choices $\rho = n^{1.5}$, $\beta = 0.99$ were close to optimal, and these are the values used in our comparisons. Further, they recommend to use the stopping parameters: $\epsilon_1 = 10^{-5}$, $\epsilon_2 = 10^{-8}$. In an attempt to get more accurate results we use $\epsilon_1 = 10^{-8}$, $\epsilon_2 = 10^{-12}$, but have found that often this results in an infinite loop: After a certain number of steps the computed vaulue for Δ is not decreased further. To cure this problem we supplement the stopping criterion (4.9a) with

$$\textbf{if } \Delta_{\text{new}} \ge \Delta_{\text{old}} \textbf{ then } \text{STOP} \;. \tag{4.9b}$$

We also compare with a Simplex type method, viz. the `bqpd` package of Fletcher (1993). This package has a broader range of applications. The option used in our comparisons addresses a generalized version of (1.1),

$$\begin{aligned} &\min_{\boldsymbol{y}}\{q(\boldsymbol{y}) \equiv \tfrac{1}{2}\boldsymbol{y}^T\boldsymbol{H}\boldsymbol{y} - \boldsymbol{c}^T\boldsymbol{y}\} \\ &\text{subject to} \quad \boldsymbol{b}_l \le \boldsymbol{y} \le \boldsymbol{b}_u \quad \text{and} \quad \widehat{\boldsymbol{b}}_l \le \boldsymbol{G}\boldsymbol{y} \le \widehat{\boldsymbol{b}}_u \;, \end{aligned} \tag{4.10}$$

where $\boldsymbol{G}$ is an $m \times n$ matrix, and $\widehat{\boldsymbol{b}}_l$, $\widehat{\boldsymbol{b}}_u$ are m-vectors. By choosing $m = 0$ and $\boldsymbol{b}_l = -\mathbf{e}$, $\boldsymbol{b}_u = \mathbf{e}$ we see that (4.10) is identical with (1.1).

The method used is an active set strategy, and demands an initial, feasible point $\boldsymbol{y}_{(0)}$. In the comparisons we use $\boldsymbol{y}_{(0)} = \boldsymbol{0}$. Further, we use the parameter values

tol	*tolmin*	*fmin*	*nrep*	*npiv*
10^{-10}	10^{-14}	$-9.0 \cdot 10^{99}$	2	3

4.5. Computational Results

We implemented the Huber algorithm and the algorithm of Han et al. in Fortran77 with extensive use of BLAS, [2]. We used Fletcher's own Fortran77 implementation of `bqpd` except that we changed it to double precision.

The tests we performed on an HP9000/800–K460, and timings were done with the `-O` option of the `f77` compiler. The machine accuracy is $\varepsilon_M = 2^{-52} \simeq 2.22{\cdot}10^{-16}$.

Below we give results for varying values of the parameters of the problem generator described in Section 4.1. In e.g. [5] such results are presented in the form of tables, where each entry is the average over 10 problems with fixed value of the parameters, with a few, selected values of the parameter under discussion (e.g. the size of the problem with $n = 100,\ 200,\ \ldots,\ 500$). Instead, we have chosen to show a "more continuous" variation ($n = 100,\ 110,\ \ldots,\ 500$) with one instance of each. This presentation illustrates both the influence of the parameter and the stochasticity in the problem generation.

In Figures 4.4–8 we use the symbols

- o Results from `huber`: the Huber method described in Sections 1–3.
- x Results from `hpy`: the interior point method of Han et al. [5] as described in Section 4.4.
- \+ Results from `bqpd`: the Simplex type method of Fletcher [3] as described in Section 4.4. Here "iterations" is the number of Simplex bases, and we have multiplied the actual numbers by $\frac{1}{40}$.

First, we show the influence of the size of the problem. Note, that both for `huber` and `hpy` the number of iterations is almost constant, while in `bqpd` the number of Simplex steps seems to grow linearly with n. In none of the cases a refactorization was needed during the Huber iterations.

As regards accuracy of the computed results, we introduce

$$q_{err} = \frac{|q(\boldsymbol{y}) - q(\boldsymbol{y}^*)|}{|q(\boldsymbol{y}^*)|} \quad \text{and} \quad y_{err} = \|\boldsymbol{y} - \boldsymbol{y}^*\|_\infty\ , \tag{4.11}$$

where $\boldsymbol{y}^*$ is the solution generated as described in Section 4.1 and $\boldsymbol{y}$ is the computed solution. Since $\|\boldsymbol{y}^*\|_\infty = 1$, both q_{err} and y_{err} are relative errors.

In all the cases reported in Figure 4.4 we found that both `huber` and

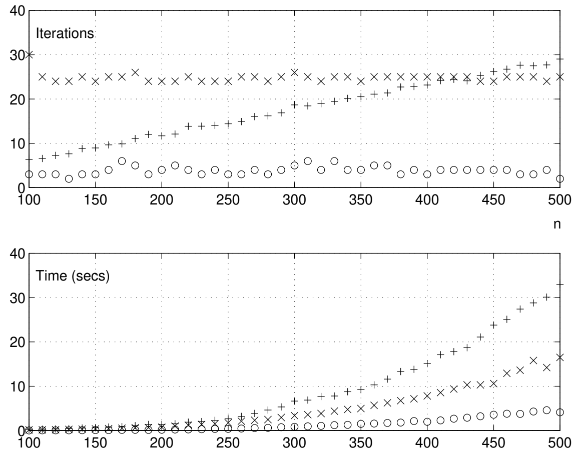

Figure 4.4. Varying size. (ncond, deg, nb, desc) = (3, 1, 50%, 0).

`bqpd` gave $q_{err} \simeq 10^{-16}$ and $y_{err} \simeq 10^{-15}$, i.e. full precision. With `hpy` we found $q_{err} \simeq 10^{-10}$ and y_{err} in the range 10^{-7} – 10^{-3}. This simple version of an interior point method does not deal with the problem of "extrapolating" from the interior to the boundary – see e.g. [11]. Therefore, we cannot expect full accuracy, but even with this in mind, the accuracy is worse than expected.

Next, in Figure 4.5 we consider the influence of the condition number $\kappa_2(\boldsymbol{H})$. Here, the number of iterations grows slightly with *ncond* for `huber` and `hpy`, but is constant for `bqpd`. At most one refactorization was used in the Huber iterations, and it is seen that the increasing number of iterations is not reflected in the computing time. Each Huber iteration is an $O(n^2)$ proces, wheras each iteration with `hpy` is an $O(n^3)$ proces, cf. (4.8).

In Figure 4.5b we show the accuracy obtained. Both the Huber and the Simplex method determine the minimum value of q with a relative error which is small multiple of the machine accuracy, and the error in the computed $\boldsymbol{y}$ grows proportional with $\kappa_2(\boldsymbol{H})$, which is to be expected. The results from `hpy` are orders of magnitude worse.

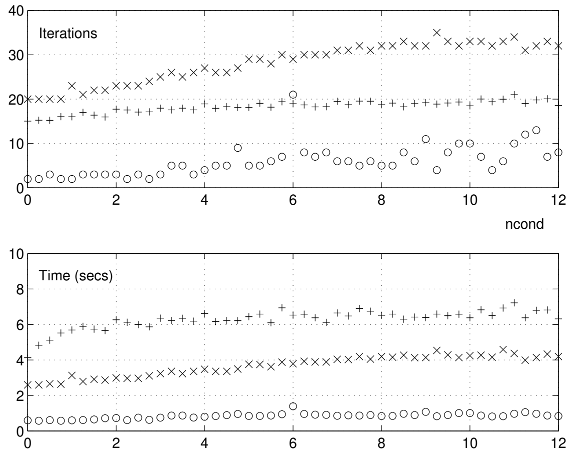

Figure 4.5a. *Varying condition. Iterations and timing.*
(n, deg, nb, desc) = (300, 1, 50%, 0).

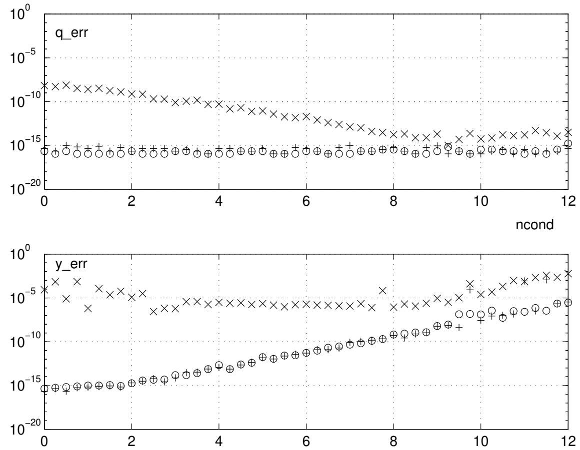

Figure 4.5b. *Varying condition. Accuracy.*
(n, deg, nb, desc) = (300, 1, 50%, 0).

In Figure 4.6 we give results for the influence of the parameter *deg*, i.e. how well active are distinguished from non active equations. Both `hpy` and `bqpd` show very little sensitivity, whereas the the number of Huber iterations grows slightly with *deg*. In none of the cases a refactorization was needed, and the accuracy is as described in connection with Figure 4.4.

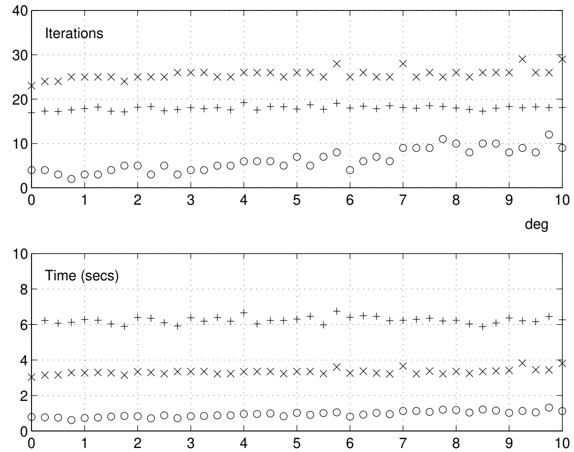

Figure 4.6. *Varying "near-degeneracy".*
(n, cond, nb, desc) = (300, 3, 50%, 0).

Figure 4.7 shows the effect of the parameter *nb,* i.e. the number of active constraints. The number of iterations seems to be independent of *nb* for `hpy`, but grows slightly with this parameter for `huber` and `bqpd`, although it does not reflect in the timings. In 8 (3) of the 41 cases shown one (two) refactorizations were used in `huber`.

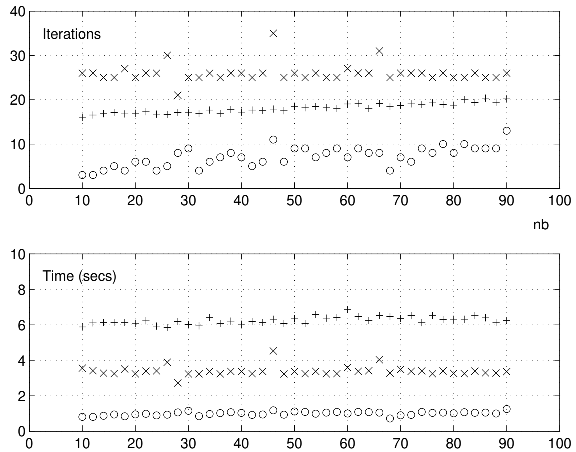

Figure 4.7. *Varying number of active constraints.*
(n, cond, deg, desc) = (300, 3, 6, 0).

Finally, Figure 4.8a–b shows the influence of the descaling factor. Here, `bqpd` needs an almost constant number of iterations, and its timings show a marked decrease as *desc* grows. The interior point method `hpy` also performs faster for increasing *desc*, while `huber` needs considerably more iterations (and up to three refactorizations). This reflects in the computing times, which are up to 0.7 times the `hpy`-time.

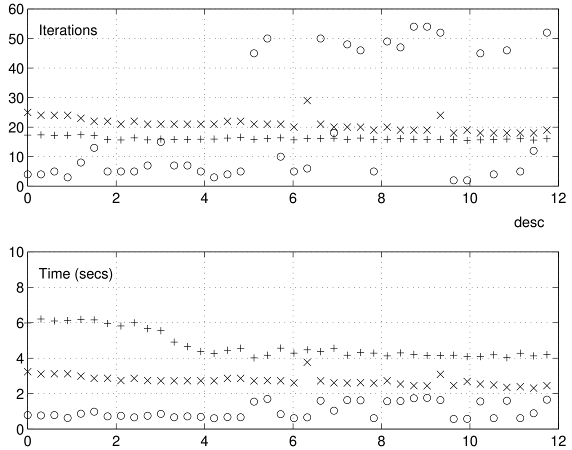

Figure 4.8a. *Varying descaling. Iterations and timing.*
(n, cond, deg, nb) = (300, 3, 1, 50%).

As regards accuracy, Figure 4.8b shows that it is doubtful whether the solution can be obtained by extrapolation from the results of `hpy`: For $desc \geq 9$ we find $0.967 \leq y_{err} \leq 1$. Also `bqpd` has trouble finding $\boldsymbol{y}$ with full accuracy, while the extra effort pays off with `huber`.

5. Conclusion

We have implemented a new method for solving quadratic programming problems with unit constraints. Careful attention to computational details has led to an efficient and accurate algorithm that compares favourably with both interior point and Simplex type methods. The details include

– Estimation of the smallest eigenvalue of a symmetric, positive definite matrix, exploiting its Cholesky factorization.

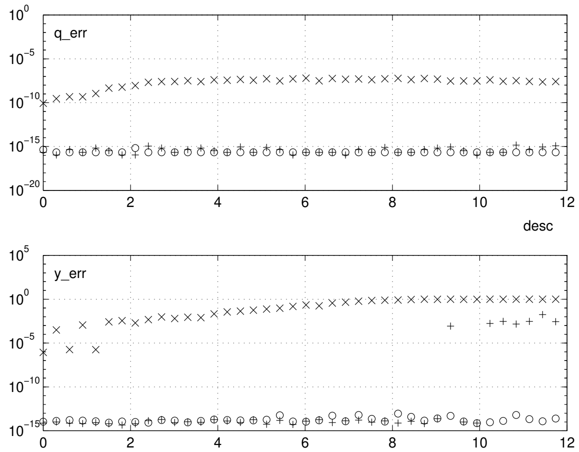

Figure 4.8b. *Varying descaling. Accuracy.*
(n, cond, deg, nb) = (300, 3, 1, 50%).

– Simple up– and downdating of triangular factors.

– Selection of a good starting point for Newton iteration.

The method is easily modified to non unit box constraints, and we are currently working on a sparse implementation of the algorithm (together with Wolfgang Hartmann from SAS). Future projects include modification of the algorithm to general linear constraints.

References

[1] T. Coleman and L. Hulbert. 1993. *A Globally and Superlinearly Convergent Algorithm for Quadratic Programming with Simple Bounds,* SIAM J. on Optimization **3**, 298–321.

[2] J. Dongarra, C.B. Moler, J.R. Bunch and G.W. Stewart. 1988. *An Extended Set of Fortran Basic Linear Algebra Subprograms,* ACM Trans. Math. Soft. **14**, 1–17.

[3] R. Fletcher. 1993. *Resolving Degeneracy in Quadratic Programming,* Annals of OR **47**, 307–334.

[4] N.J. Higham. 1987. *A Survey of Condition Number Estimation for Triangular Matrices,* SIAM Review **29**, 575–596.

[5] C-G. Han, P. Pardalos and Y. Ye. 1990. *Computational Aspects of an Interior Point Algorithm for Quadratic Programming Problems with Box Constraints,* pp 92–112 in T. Coleman and Y. Li (eds), *Large Scale Numerical Optimization*, SIAM, Philadelphia.

[6] K. Madsen and H.B. Nielsen. 1990. *Finite Algorithms for Robust Linear Regression,* BIT **30**, 682–699.

[7] K. Madsen, H.B. Nielsen. 1993. *A Finite Smoothing Algorithm for Linear L_1 Estimation,* SIAM J. on Optimization **3**, 223–235.

[8] K. Madsen, H. B. Nielsen and M.Ç. Pınar. 1995. *A New Finite Continuation Algorithm for Bound Constrained Quadratic Programming.* Report IMM-REP-95-22, Department of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, Denmark.

[9] K. Madsen, H.B. Nielsen and M.Ç. Pınar. 1996. *A New Finite Continuation Algorithm for Linear Programming,* SIAM J. on Optimization **6**, 600–616.

[10] K. Madsen, H.B. Nielsen and M.Ç. Pınar. 1996. *Bound Constrained Quadratic Programming Solved via Piecewise Quadratic Functions.* Forthcoming report from IMM, Department of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, Denmark.

[11] N. Megiddo. 1991. *On Finding Primal– and Dual– Optimal Bases.* ORSA J. on Comput. **3**(1), 63–65.

[12] J. Moré and G. Toraldo. 1989. *Algorithms for Bound Constrained Quadratic Programming Problems,* Numer. Math. **55**, 377–400.

[13] H.B. Nielsen. 1991. *Implementation of a Finite Algorithm for linear ℓ_1 Estimation,* Report NI-91-01, Institute for Numerical Analysis, Technical University of Denmark, DK-2800 Lyngby, Denmark.