On iterative algorithms for linear least squares problems with bound constraints

by M. Bierlaire, Ph.L. Toint and D. Tuyttens

October 3, 1995

Abstract. Three new iterative methods for the solution of the linear least squares problem with bound constraints are presented and their performance analyzed. The first is a modification of a method proposed by Lötstedt, while the two others are characterized by a technique allowing for fast active set changes resulting in noticeable improvements on the speed at which constraints active at the solution are identified. The numerical efficiency of these algorithms is experimentally studied, with particular emphasis on dependence of the starting point choice and the use of preconditioning for ill-conditioned problems.

Department of Mathematics
Facultés Universitaires ND de la Paix
B-5000 Namur, Belgium

Keywords : Linear least squares, statistical regression, bound constraints, iterative methods, active sets, preconditioning, numerical experiments.

On iterative algorithms for linear least squares problems with bound constraints

by M. Bierlaire, Ph.L. Toint and D. Tuyttens

Report 89/5 October 3, 1995

Abstract. Three new iterative methods for the solution of the linear least squares problem with bound constraints are presented and their performance analyzed. The first is a modification of a method proposed by Lötstedt, while the two others are characterized by a technique allowing for fast active set changes resulting in noticeable improvements on the speed at which constraints active at the solution are identified. The numerical efficiency of these algorithms is experimentally studied, with particular emphasis on dependence of the starting point choice and the use of preconditioning for ill-conditioned problems.

Department of Mathematics
Facultés Universitaires ND de la Paix
B-5000 Namur, Belgium

Keywords : Linear least squares, statistical regression, bound constraints, iterative methods, active sets, preconditioning, numerical experiments.

1 Introduction

The linear least squares problem, that is the problem of finding a vector $x \in \mathbf{R}^n$ solving

$$\min Q(x) \stackrel{\text{def}}{=} \frac{1}{2}\|Ax - b\|_2^2 \tag{1}$$

for a given matrix $A \in \mathbf{R}^{m \times n}$ and a given vector $b \in \mathbf{R}^m$, has always enjoyed a lot of attention from researchers in numerical analysis and statistics. This interest is justified by the fact that solving (1) often plays a crucial role in fitting a theoretical model to real world observations, a fundamental preoccupation in many sciences. We refer the reader to [15] and [4] for recent surveys on the related algorithmic methods. We will focus our attention on a particular case of (1) that often arises in practice: the case where the variables (the components of x) are restricted to lie in certain prescribed intervals. This amounts to adding the constraints

$$l_i \leq x_i \leq u_i \quad (i = 1, \ldots, n) \tag{2}$$

to the minimization (1). Not only these constraints do appear in many practical examples (tomography [26], for instance), but it is even sometimes argued that any minimization problem, in particular (1), is only realistic when its variables are within meaningful intervals [11]. Special methods for this very important case have therefore been considered by several authors (see [3], [14], [18] and [28], for instance).

From the computational point of view, algorithms for the solution of (1) (with or without (2)) fall into two classes : direct methods, usually based on some matrix factorization, and iterative ones, amongst which conjugate gradients techniques play an important role. When comparing these two classes, it has been observed that iterative methods can be very advantageous when the matrix A in (1) is well-conditioned, or when an efficient preconditioner is known, such that the original problem can be transformed into a well-conditioned one. This is especially true in a number of practical applications (for example, geodetic positioning [13] and mechanics [19]), where the number of variables n and the number of equations m are large and the matrix A is sparse. Solving such bound constrained problems by iterative methods is also the subject of this paper.

In this context, special attention must clearly be given to the part of the algorithm that handles the status of variables with respect to their bounds. In particular, some known methods ([3], [18]) do only allow one variable to leave a bound at a given iteration. This is a very undesirable feature when the number of variables is large, because a large number of iterations may then be required when many variables must leave their bound. This restriction is however understandable in the case of direct methods based on the QR factorization (as in the first method cited above) because each variable leaving its bound corresponds to increasing the dimension of the upper triangular factor R by one, which can be efficiently handled. It is less obvious to us why the same restriction is imposed on iterative algorithms that do not maintain an upper triangular matrix R at all. Furthermore, even if the theory allows several variables to hit one of their bounds at the same iteration, this is very seldom observed in practice: most iterations indeed only fix one single variable at its bound. Because of these features, the number of iterations required to compute the solution of (1)–(2) may be unacceptably high for large scale problems when the set of constraints active at this solution is not guessed well by the user.

On the other hand, recent developments in general nonlinear optimization with bound constraints have produced algorithms where the number of active constraints (that is the number of variables exactly at one of their bounds) can change very rapidly from one iteration to the next (see [6], [9], [20], [21], [25], [27]). The first of these algorithms has shown to be quite efficient on general nonlinear and quadratic problems [7]. Our purpose in the present paper is to specialize the class of algorithms described in [6], [7], [20] and [25] to the particular case of (1)–(2).

Section 2 will introduce a variant of the algorithm proposed by Lötstedt in [18] and two new methods based on the nonlinear techniques cited above. Section 3 is devoted to the presentation of some comparative numerical results involving these three methods. Finally, conclusions and further perspectives are discussed in Section 4.

2 The algorithms

We now introduce the three algorithms that we will consider in more detail. All three are based on a conjugate gradient related iterative least squares solver, which is formally equivalent to a step by step Lanczos decomposition of the matrix A^tA followed by Cholesky factorization of the resulting tridiagonal. As all such methods, they are potentially advantageous when A is relatively well conditioned, or when an efficient preconditioner can be applied.

The considered methods involve two distincts algorithmic layers.

- The outer layer handles the status of the variables with respect to their bounds, by declaring each variable either *free*, that is strictly between its bounds, or *fixed*, that is at one of its bounds. This layer proceeds in an iterative fashion, and the outer layer iterations will be called *major* iterations. We will also denote by $I^{(k)}$ the set of fixed variables at major iteration k, and by $J^{(k)}$ its complement, that is the set of free variables at this iteration. The kth major iterate will be denoted by $x^{(k)}$.

- The inner layer will attempt, at every major iteration, to solve the problem (1) by a conjugate gradient algorithm applied while keeping the variables of $I^{(k)}$ fixed and its current approximate solution feasible. The iterations of this conjugate gradient method will be called *minor* iterations and denoted by superscripts of the form $[i]$.

2.1 Minor iterations : solving the subproblem in the free variables

For solving the restricted least-squares subproblem, we have chosen to use a variant of the LSQR algorithm [23]. This variant differs from the original algorithm in the following points.

- It is applied only in the subspace of the free variables, the fixed variables remaining, by definition, fixed.

- The iterations are stopped as soon as one or more of the free variables violate their bounds.

- The starting point is not necessarily the origin, which may well be infeasible, but a feasible point given by the current major iteration.

Its formal description otherwise follows that of [23].

More precisely, assume that $J^{[0]} = J^{(k)}$ is the set of free variables at major iteration k. A starting point $x^{[0]} = x^{(k)}$ is also given. We also assume, without loss of generality, that

$$J^{[0]} = \{1, \ldots, p\}. \tag{3}$$

We then partition the vector $x^{[0]}$ into two parts :

$$x^{[0]} = \begin{pmatrix} y^{[0]} \\ z^{[0]} \end{pmatrix}, \tag{4}$$

where the vector $y^{[0]}$ contains the first p components of $x^{[0]}$ (the free part), while $z^{[0]}$ contains the $n-p$ last components (the fixed part). The subproblem then reduces to finding a new vector y of free variables solving

$$\min_{y \in \mathbf{R}^p} \frac{1}{2} \|By - d\|_2^2 \tag{5}$$

where the matrix $B \in \mathbf{R}^{m \times p}$ contains the first p columns of A and where

$$d = b - Ax^{[0]}. \tag{6}$$

We also assume that the stopping rule parameters ATOL and BTOL are given, according to [23].

Restricted LSQR

Step 1 : Initialization.

Given $x^{[0]}$ and $J^{[0]}$, compute $y^{[0]}$ and d from (4) and (6). Also define

$$\begin{aligned} \beta^{[1]} u^{[1]} &= d, \\ \alpha^{[1]} v^{[1]} &= B^t u^{[1]}, \\ w^{[1]} &= v^{[1]}, \\ \bar{\phi}^{[1]} &= \beta^{[1]}, \\ \bar{\rho}^{[1]} &= \alpha^{[1]} \end{aligned} \tag{7}$$

and set $i = 1$.

Step 2 : Minor iterations.

Step 2.a : Bidiagonalization.

Set

$$\begin{aligned} \beta^{[i+1]} u^{[i+1]} &= Bv^{[i]} - \alpha^{[i]} u^{[i]}, \\ \alpha^{[i+1]} v^{[i+1]} &= B^t u^{[i+1]} - \beta^{[i+1]} v^{[i]}. \end{aligned} \tag{8}$$

Step 2.b : Orthogonal transformation.

Compute

$$\begin{aligned}
\rho^{[i]} &= (\bar{\rho}^{[i]2} + \beta^{[i+1]2})^{\frac{1}{2}}, \\
c^{[i]} &= \bar{\rho}^{[i]}/\rho^{[i]}, \\
s^{[i]} &= \beta^{[i+1]}/\rho^{[i]}, \\
\theta^{[i+1]} &= s^{[i]}\alpha^{[i+1]}, \\
\bar{\rho}^{[i+1]} &= -c^{[i]}\alpha^{[i+1]}, \\
\phi^{[i]} &= c^{[i]}\bar{\phi}^{[i]}, \\
\bar{\phi}^{[i+1]} &= s^{[i]}\bar{\phi}^{[i]}.
\end{aligned} \tag{9}$$

Step 2.c : Update.

Compute the maximum feasible step from $y^{[i-1]}$ along $w^{[i]}$:

$$\lambda^{[i]} = \min_{j=1,\ldots,p} \lambda_j^{[i]}, \tag{10}$$

where

$$\lambda_j^{[i]} = \begin{cases} \frac{l_j - y_j^{[i-1]}}{w_j^{[i]}} & \text{if } w_j^{[i]} < 0, \\ \frac{u_j - y_j^{[i-1]}}{w_j^{[i]}} & \text{if } w_j^{[i]} > 0, \\ +\infty & \text{otherwise.} \end{cases} \tag{11}$$

If $\lambda^{[i]} \leq \frac{\phi^{[i]}}{\rho^{[i]}}$ then set

$$y^{[i]} = y^{[i-1]} + \lambda^{[i]} w^{[i]}, \tag{12}$$

define $S^{[*]}$ as the set of indices of the variables that are at their bounds at $y^{[i]}$ but not at $y^{[i-1]}$ and go to Step 2.e.

Else set

$$y^{[i]} = y^{[i-1]} + \frac{\phi^{[i]}}{\rho^{[i]}} w^{[i]} \tag{13}$$

$$w^{[i+1]} = v^{[i+1]} - \frac{\theta^{[i+1]}}{\rho^{[i]}} w^{[i]}. \tag{14}$$

Step 2.d : Stopping criteria.

If

$$\begin{gathered} \|r^{[i]}\| \leq \text{BTOL}\|d\| + \text{ATOL}\|B\|\|y^{[i]}\| \\ \text{or} \\ \|B^t r^{[i]}\| \leq \text{ATOL}\|B\|\|r^{[i]}\| \end{gathered} \tag{15}$$

where

$$r^{[i]} = By^{[i]} - d, \tag{16}$$

then define $S^{[*]} = \emptyset$ and go to Step 2.e.

Else increment i by one and go to Step 2.a.

Step 2.e : Return to the n-dimensional space.

Set

$$x^{[*]} = \begin{pmatrix} y^{[i]} \\ z^{[0]} \end{pmatrix} \tag{17}$$

and STOP.

As in [23], the scalars $\alpha^{[i]}$ and $\beta^{[i]}$ are chosen in (7) and (8) to normalize the vectors $v^{[i]}$ and $u^{[i]}$ respectively.

For a given problem (1)–(2) and for a consistent choice of ATOL and BTOL, we will refer to the application of this algorithm, starting from $x^{[0]}$ with the indices of the free variables $J^{[0]}$ and computing the result $x^{[*]}$, by the statement

$$(x^{[*]}, S^{[*]}) = \mathrm{LSQR}(x^{[0]}, J^{[0]}), \tag{18}$$

where $S^{[*]}$ is the set of indices of the free variables that hit one of their bounds in Step 2.c. $S^{[*]} = \emptyset$ therefore means that a feasible unconstrained solution has been found in the space of the free variables, and $S^{[*]} \neq \emptyset$ means that one or more bound(s) have been hit before any of the optimality conditions of Step 2.d is satisfied.

We end this paragraph by noting that a totally similar adaptation of the conjugate gradient CGLS (see [16], [23]) is also possible. The same three modifications (see above) are made to the original algorithm. This will not, however, be analyzed in full detail.

2.2 Major iterations : handling the bound constraints

2.2.1 A variant of Lötstedt's algorithm

In [18], Lötstedt introduced a method designed to solve a sequence of related problems of the form (1)–(2). The idea is to perform a QR factorization for the first problem of the sequence, and then to solve the remaining problems of the sequence using a conjugate gradient based algorithm with this first factorization as preconditioner.

We note that the iterative algorithm used for solving all problems of the sequence but the first is of the *active set* type, in the sense that it only allows one variable to leave its bound per iteration. The variant considered below shares this feature, but introduces a few modifications to improve efficiency when the number of variables is large.

This variant can be formalized as follows.

Active set algorithm

Step 1 : Initialization.

Let $x^{(0)}$ be a feasible point and $I^{(0)}$ the set of indices of the components of $x^{(0)}$ that are exactly at one of their bounds and $J^{(0)}$ its complement. Set $k = 0$.
If $J^{(0)} \neq \emptyset$, then go to Step 2.d.

Step 2 : Major iterations.

Step 2.a : Calculate the gradient of Q (which is the Lagrange multipliers associated with the bound constraints with opposite sign) by

$$g^{(k)} = A^t(Ax^{(k)} - b). \tag{19}$$

Step 2.b : Check if $x^{(k)}$ is a constrained solution.

Compute

$$\Gamma^{(k)} = \max_{j \in I^{(k)}} \gamma_j^{(k)} g_j^{(k)} \tag{20}$$

with

$$\gamma_j^{(k)} = \begin{cases} 1 & \text{if } x_j^{(k)} = u_j \\ -1 & \text{if } x_j^{(k)} = l_j \end{cases} \tag{21}$$

If $\Gamma^{(k)} \leq 0$ then STOP with $x^{(*)} = x^{(k)}$.

Step 2.c : Drop a bound.

Update the sets of free and fixed variables by

$$I^{(k)} = I^{(k)} \setminus \{t\} \quad \text{and} \quad J^{(k)} = J^{(k)} \cup \{t\}, \tag{22}$$

where $1 \leq t \leq n$ is such that

$$\Gamma^{(k)} = \gamma_t^{(k)} g_t^{(k)}. \tag{23}$$

Step 2.d : Solve the problem on the free variables using an iterative algorithm while keeping the fixed variables unchanged, that is compute

$$(x^{(k+1)}, S^{(k+1)}) = \text{LSQR}(x^{(k)}, J^{(k)}). \tag{24}$$

Then set

$$I^{(k+1)} = I^{(k)} \cup S^{(k+1)} \quad \text{and} \quad J^{(k+1)} = J^{(k)} \setminus S^{(k+1)}. \tag{25}$$

Step 2.e : Restart if one or more bounds have been hit.

If $S^{(k+1)} \neq \emptyset$ and $J^{(k+1)} \neq \emptyset$, increment k by one and go to Step 2.d.

Step 2.f : Check if $x^{(k+1)}$ is an unconstrained solution.

If $I^{(k+1)} = \emptyset$ then STOP with $x^{(*)} = x^{(k+1)}$.

Else increment k by one and go to Step 2.a.

This algorithm actually differs from that proposed by Lötstedt in [18] in that the solution of the least squares problem in the free variables is abandoned as soon as one of these variables violates a bound. In the original algorithm, no bound constraint is imposed on the free variables in the subproblem: the iterative algorithm first completes the subproblem's solution in the space of the free variables irrespective of feasibility, and then chooses $x^{(k+1)}$ as the feasible point on the segment $[x^{(k)}, x^{[*]}]$ that is closest to $x^{[*]}$. The original algorithm is formally equivalent to that presented in [4], where a direct solver is used instead of an iterative method. Our modification has a (sometimes very) significant advantage when n is large: the solution of the subproblem can be stopped as soon as it is detected that the fixed variables are incorrectly guessed, sparing unnecessary minor iterations.

A further modification to the original method is the possibility to start the algorithm from a non-interior point. We note that this modification can be useful when the set of variables active at the solution is reasonably well guessed, because the algorithm will not need to set these variables to their bound explicitely.

2.2.2 Methods using the projected gradient

The two new methods that we would like to introduce both belong to the class of algorithms that use information from the projected gradient to guess the set of bounds that are active at the solution. Such algorithms have been analyzed in the context of general nonlinear optimization by many authors : originally introduced by Goldstein [12], Levitin and Polyak [17], this class of algorithms has enjoyed continued development (see [1] and [10], for example). More recently, the concept of guessing the set of active bounds using a projected gradient step has been incorporated in efficient trust region algorithms by Conn, Gould and Toint [6], [7], while alternative methods in the same context were developed by Toint [25] and Moré and Toraldo [20].

We now consider the specialization of these general methods to the least squares problem and note that the approaches of [6] and [25] differ in the detailed strategy used for activating the bound constraints, that is for deciding which variables must be fixed before an attempt to solve (1) is made. Both strategies do however share the following feature : they consider that this set of variables is determined by the active constraints at a specially computed point, called the *generalized Cauchy point* (GCP). If we define, for $t \geq 0$,

$$s^{(k)}(t) = P[x^{(k)} - tg^{(k)}] - x^{(k)}, \tag{26}$$

where $P[\cdot]$ is the orthogonal projection onto the convex set of feasible points and $g^{(k)}$ is defined by (19), this GCP is defined by

$$x_C^{(k)} = x^{(k)} + s^{(k)}(t_C^{(k)}) \tag{27}$$

for some $t_C^{(k)} \geq 0$. The method of choice for $t_C^{(k)}$ is what differs between the two mentioned approaches.

We first describe the strategy used in [6]. The idea is to determine $t_C^{(k)}$ such that $x_C^{(k)}$ is exactly the first local minimizer of the univariate piecewise quadratic

$$q(t) = Q(x^{(k)} + s^{(k)}(t)) \tag{28}$$

for $t \geq 0$. Computing the adequate value of t is achieved by successively examining the intervals between breakpoints of (26), that is the points where one or more variable(s) hit(s) its bound along the piecewise linear trajectory defined by (26) for $t \geq 0$. The inner iterations generated by this successive examination will be denoted by superscripts of the form $\{\ell\}$ and called *GCP iterations* . This technique for finding $x_C^{(k)}$ will be called *exact linesearch* and is detailed below.

Exact linesearch

Step 1 : Initialization.

Given an initial point $x^{\{0\}}$, $g^{\{0\}}$ the gradient of Q at this point and the corresponding active set $I^{\{0\}}$, compute the direction

$$d_j^{\{0\}} = \begin{cases} -g_j^{\{0\}} & \text{if } j \notin I^{\{0\}}, \\ 0 & \text{if } j \in I^{\{0\}}. \end{cases} \tag{29}$$

and the directional derivatives

$$q'^{\{0\}} = g^{\{0\}t} d^{\{0\}} \text{ and } q''^{\{0\}} = d^{\{0\}t} A^t A d^{\{0\}} = \|Ad^{\{0\}}\|^2. \tag{30}$$

Set $\ell = 0$.

Step 2 : Check if the GCP is at the current breakpoint.
If $q'^{\{\ell\}} \geq 0$, then STOP with $x^{\{*\}} = x^{\{\ell\}}$ and $I^{\{*\}} = I^{\{\ell\}}$.

Step 3 : Determine the next breakpoint.
Compute the maximum feasible step along $d^{\{\ell\}}$:

$$\alpha^{\{\ell\}} = \min_{j \notin I^{\{\ell\}}} \alpha_j^{\{\ell\}}, \tag{31}$$

where

$$\alpha_j^{\{\ell\}} = \begin{cases} \frac{l_j - x_j^{\{\ell\}}}{d_j^{\{\ell\}}} & \text{if } d_j^{\{\ell\}} < 0, \\ \frac{u_j - x_j^{\{\ell\}}}{d_j^{\{\ell\}}} & \text{if } d_j^{\{\ell\}} > 0, \\ +\infty & \text{otherwise.} \end{cases} \tag{32}$$

Step 4 : Check if the GCP is before the next breakpoint.
If $q''^{\{\ell\}} > 0$ and $(-q'^{\{\ell\}}/q''^{\{\ell\}}) < \alpha^{\{\ell\}}$, then STOP with

$$x^{\{*\}} = x^{\{\ell\}} - \frac{q'^{\{\ell\}}}{q''^{\{\ell\}}} d^{\{\ell\}} \tag{33}$$

and $I^{\{*\}} = I^{\{\ell\}}$.

Step 5 : Updates.
Set

$$x^{\{\ell+1\}} = x^{\{\ell\}} + \alpha^{\{\ell\}} d^{\{\ell\}}, \tag{34}$$

and define $S^{\{\ell\}}$ as the set of indices of the variables that are at one of their bounds at $x^{\{\ell+1\}}$ and were not at $x^{\{\ell\}}$. Then set

$$\begin{array}{lcl} I^{\{\ell+1\}} & = & I^{\{\ell\}} \cup S^{\{\ell\}}, \\ z^{\{\ell\}} & = & A^t \sum_{j \in S^{\{\ell\}}} d_j^{\{\ell\}} A e_j, \\ q'^{\{\ell+1\}} & = & q'^{\{\ell\}} + \alpha^{\{\ell\}} q''^{\{\ell\}} - z^{\{\ell\}t} x^{\{\ell+1\}} + \sum_{j \in S^{\{\ell\}}} d_j^{\{\ell\}} (A^t b)_j, \\ q''^{\{\ell+1\}} & = & q''^{\{\ell\}} + \sum_{j \in S^{\{\ell\}}} d_j^{\{\ell\}} z_j^{\{\ell\}} - 2 z^{\{\ell\}t} d^{\{\ell\}} \end{array} \tag{35}$$

where e_j is the jth vector of the canonical basis and $z^{\{\ell\}} \in \mathbf{R}^n$, and finally

$$d_j^{\{\ell+1\}} = \begin{cases} d_j^{\{\ell\}} & \text{if } j \notin S^{\{\ell\}}, \\ 0 & \text{if } j \in S^{\{\ell\}}. \end{cases} \tag{36}$$

Increment ℓ by one and go to Step 2.

We will refer to the application of this algorithm to determine $x^{\{*\}}$ and $I^{\{*\}}$ from $x^{\{0\}}$, $g^{\{0\}}$ and $I^{\{0\}}$ by the statement

$$(x^{\{*\}}, I^{\{*\}}) = \text{EXACT}(x^{\{0\}}, g^{\{0\}}, I^{\{0\}}). \tag{37}$$

We note that the recurrences (35) can be implemented very efficiently: it is indeed possible to write them in such a way that the lengths of the necessary loops do only depend on the density of A and the size of $S^{\{l\}}$, as opposed to the dimensions n and m. The explicit recomputation of the breakpoint's curvilinear abscissas (32) at each GCP iteration can also be avoided. More detail on this implementation will be given in a forthcoming paper.

The second approach, introduced in [25] and refined in [20], does not require the GCP, $x_C^{(k)}$, to be an exact minimizer of the piecewise quadratic $q(t)$, but merely that this point satisfies a condition guaranteeing a sufficient decrease in $q(t)$. We then have an *inexact linesearch* in the usual sense, where the termination conditions are of the Armijo-type (see [2, pp. 20 ff.], for instance), and depend on a parameter $\mu \in (0,1)$ given a priori.

The implementation given below is inspired by [20], but does not follow this source exactly. In particular, the search starts backtracking from the last breakpoint as soon as it is detected that the first line minimum is past the first breakpoint. This strategy is only used by Moré and Toraldo if negative curvature is observed, which would never happen in the least squares context. This modification appears to be important in certain examples that we discuss in the next section.

Inexact linesearch

Step 1 : Minimization in the first interval.

Given an initial point $x^{\{0\}}$, $g^{\{0\}}$ the gradient of Q at this point and the corresponding active set $I^{\{0\}}$, compute the line coordinate of the minimizer of q in the first interval by

$$t^{\{0\}} = \frac{\|d^{\{0\}}\|^2}{\|Ad^{\{0\}}\|^2}, \tag{38}$$

where $d^{\{0\}}$ is given by (29). Also set $\ell = 0$ and compute $\alpha^{\{0\}}$, the line coordinate of the first break point, from (31) and (32).

If $t^{\{0\}} \leq \alpha^{\{0\}}$, then STOP with

$$x^{\{*\}} = x^{\{0\}} + t^{\{0\}} d^{\{0\}} \tag{39}$$

and $I^{\{*\}} = I^{\{0\}}$.

Else, compute the line coordinate of the last breakpoint as

$$t^{\{1\}} = \max_{j \notin I^{\{0\}}} \alpha_j^{\{0\}} \tag{40}$$

where the $\alpha_j^{\{0\}}$ are again computed by (32), and set $\ell = 1$.

Step 2 : Check if the decrease in q is sufficient.

Compute

$$x^{\{\ell\}} = P[x^{\{0\}} + t^{\{\ell\}} d^{\{0\}}]. \tag{41}$$

and define $S^{\{\ell\}}$ as the set of indices of the bounds that are violated at $x^{\{0\}} + t^{\{\ell\}} d^{\{0\}}$.

If

$$q(x^{\{\ell\}}) \leq q(x^{\{0\}}) + \mu d^{\{0\}t}(x^{\{\ell\}} - x^{\{0\}}), \tag{42}$$

then STOP with $x^{\{*\}} = x^{\{\ell\}}$ and $I^{\{*\}} = I^{\{0\}} \cup S^{\{\ell\}}$.

Step 3 : Safeguarded quadratic interpolation.

Compute $\lambda^{\{\ell\}}$, the line coordinate of the minimizer of the univariate quadratic that interpolates $q(0)$, $q'(0)$ and $q(t^{\{\ell\}})$ from

$$\lambda^{\{\ell\}} = -\frac{q'(0)}{2[q(t^{\{\ell\}}) - q(0) - q'(0)]}. \tag{43}$$

Then set

$$t^{\{\ell+1\}} = \max\left[\alpha^{\{0\}}, \frac{t^{\{\ell\}}}{100}, \min(\lambda^{\{\ell\}}, \frac{t^{\{\ell\}}}{2})\right]. \tag{44}$$

Increment ℓ by one and go to Step 2.

We will refer to the application of this last algorithm to determine $x^{\{*\}}$ and $I^{\{*\}}$ from $x^{\{0\}}$, $g^{\{0\}}$ and $I^{\{0\}}$ by the statement

$$(x^{\{*\}}, I^{\{*\}}) = \text{INEXACT}(x^{\{0\}}, g^{\{0\}}, I^{\{0\}}). \tag{45}$$

The inner iterations of this procedure (denoted by superscripts of the form $\{\ell\}$) are also called *GCP iterations.*

We now describe the algorithmic framework that is common to both methods based on the projected gradient. In this framework, every major iteration consists of two successive phases. The first is the computation of the GCP, which will determine the fixed and free variables. The second phase then attempts to solve the least squares problem in the subspace of free variables.

A projected gradient framework

Step 1 : Initialization.

Let $x^{(0)}$ be a feasible point. Set $k = 0$.

Step 2 : Major iterations.

Step 2.a : Compute $g^{(k)}$, the gradient of Q at $x^{(k)}$ from (19).

Step 2.b : Compute the active set at $x^{(k)}$ and check if $x^{(k)}$ is a solution.

Define the set $I^{(k)}$ as the set of indices j between 1 and n such that

$$\left[x_j^{(k)} = l_j \text{ and } g_j^{(k)} > 0\right] \text{ or } \left[x_j^{(k)} = u_j \text{ and } g_j^{(k)} < 0\right]. \tag{46}$$

If $I^{(k)} = \{1, \ldots, n\}$ or if $k \geq 1$ and $I^{(k)} = I^{(k-1)}$, then STOP with $x^{(*)} = x^{(k)}$.

Step 2.c : Find the GCP.

Apply one of the methods described above, that is

$$(x_C^{(k)}, I_C^{(k)}) = \text{EXACT}(x^{(k)}, g^{(k)}, I^{(k)}) \text{ or } (x_C^{(k)}, I_C^{(k)}) = \text{INEXACT}(x^{(k)}, g^{(k)}, I^{(k)}). \tag{47}$$

Step 2.d : Attempt to solve the problem in the space of the free variables.
If $I_C^{(k)} = \{1, \ldots, n\}$, define $x^{(k+1)} = x_C^{(k)}$;
else, compute

$$(x^{(k+1)}, S^{(k+1)}) = \mathrm{LSQR}(x_C^{(k)}, \{1, \ldots, n\} \setminus I_C^{(k)}). \tag{48}$$

Increment k by one and go to Step 2.a.

It will also be useful in the next Section to consider a variant of this technique where the Step 2.d is replaced by the following (depending on an a priori given parameter $\nu \in (0, 1)$).

Step 2.d (variant) : Attempt to solve the problem in the space of the free variables.
If $I_C^{(k)} = \{1, \ldots, n\}$, define $x^{(k+1)} = x_C^{(k)}$.
Else, perform the following steps :

Step 2.d.1 : Initialization.
Define $x^{\langle 0 \rangle} = x_C^{(k)}$, $I^{\langle 0 \rangle} = I_C^{(k)}$. Set $p = 0$.

Step 2.d.2 : Apply LSQR.

$$(x^{\langle p+1 \rangle}, S^{\langle p+1 \rangle}) = \mathrm{LSQR}(x^{\langle p \rangle}, \{1, \ldots, n\} \setminus I^{\langle p \rangle}). \tag{49}$$

If $S^{\langle p+1 \rangle} = \emptyset$, then go to Step 2.d.4.

Step 2.d.3 : Check if the gradient of Q has been reduced enough.
If

$$\|A^t(Ax^{\langle p+1 \rangle} - b)\| < \nu \|A^t(Ax^{\langle 0 \rangle} - b)\|, \tag{50}$$

then go to Step 2.d.4.
Else, update

$$I^{\langle p+1 \rangle} = I^{\langle p \rangle} \cup S^{\langle p+1 \rangle}, \tag{51}$$

increment p by one, and go to Step 2.d.2.

Step 2.d.4 : Set $x^{(k+1)} = x^{\langle p+1 \rangle}$, increment k by one and go to Step 2.a.

This variant may be viewed as imposing a minimum progression towards the solution before allowing fixed variables to be freed by the algorithm. It also introduces the concept of incomplete problem solution, as proposed for general nonlinear problems in [24] and [8], in the least squares context. It is motivated by some preconditioned examples, as will be discussed below.

The iterations between Step 2.d.2 and Step 2.d.4 are called *restart iterations* and are denoted by superscripts of the form $\langle p \rangle$.

3 Some numerical experiments

This section provides some numerical results with the three methods described in the previous section. The purposes of these experiments are

- to prove the computational feasibility of the two new methods,

- to allow a comparison with a more classical algorithm, showing the advantages and drawbacks of the new proposals,
- to compare the two active set strategies used in the new approaches,
- to analyse the effect of preconditioning on performance.

The three algorithms described above were implemented in Fortran 77, and the numerical experiments reported were all run on a DEC VAX3500 at the Facultés Universitaires ND de la Paix (Namur, Belgium), under the VMS Fortran Compiler with optimization. All real quantities were stored in double precision ($\epsilon_M \approx 1.37 \times 10^{-17}$). Some care must be exercized when interpreting the cpu times (in seconds) quoted below : due to the busy character of this time-shared machine, their relative error margin may be as high as 8%. All test reported use the following default values for the algorithm's constants:

$$\mathrm{ATOL} = \mathrm{BTOL} = \epsilon_M, \quad \text{and} \quad \mu = \nu = \frac{1}{10}. \tag{52}$$

3.1 The test problems

The test problems used feature three different matrix structures:

- dense matrices of moderate size,
- random sparse matrices,
- block-angular matrices, arising from a Doppler positioning application [13].

All these matrices are relatively well conditioned ($\kappa(A) \leq 10^4$).

The dense matrices are generated from their singular value decomposition

$$A = USV^t, \tag{53}$$

where the positive diagonal entries of S are such that their logarithm is uniformly distributed in the interval $[-\frac{1}{2}\ln(\kappa(A)), \frac{1}{2}\ln(\kappa(A))]$, and where the orthogonal matrices U and V are the Q factors obtained from the QR decomposition of randomly generated matrices with elements in [-100,100].

The sparse matrices are randomly generated with the constraint that each row should contain a prescribed number of nonzero elements. These elements are chosen using a uniform distribution in $[-100, 100]$.

The block-angular matrices are generated as for the structure described in [13], that is

$$A = \begin{pmatrix} B_1 & & & & C_1 \\ & B_2 & & & C_2 \\ & & \ddots & & \vdots \\ & & & B_{m_D} & C_{m_D} \end{pmatrix}, \tag{54}$$

where

$$B_i = \begin{pmatrix} B_{i1} \\ B_{i2} \\ \vdots \\ B_{in_{\mathrm{D}}} \end{pmatrix}, \quad C_i = \begin{pmatrix} C_{i1} & & & \\ & C_{i2} & & \\ & & \ddots & \\ & & & C_{in_{\mathrm{D}}} \end{pmatrix} \tag{55}$$

and where the blocks B_{ij} and C_{ij} ($i = 1, \ldots, m_{\mathrm{D}}; j = 1, \ldots n_{\mathrm{D}}$) are dense of dimension $(k_{\mathrm{D}}+p_{\mathrm{D}}) \times k_{\mathrm{D}}$ and $(k_{\mathrm{D}}+p_{\mathrm{D}}) \times p_{\mathrm{D}}$ respectively. The overall least squares problem has therefore $m_{\mathrm{D}}k_{\mathrm{D}} + n_{\mathrm{D}}p_{\mathrm{D}}$ variables and $m_{\mathrm{D}}n_{\mathrm{D}}(k_{\mathrm{D}} + p_{\mathrm{D}})$ equations. The dense blocks B_{ij} and C_{ij} are generated in the manner described above for dense matrices, with a different conditioning parameter for the blocks B_{ij} and C_{ij} (denoted $\kappa(B)$ and $\kappa(C)$ below). For an interpretation of the parameters associated with this structure in the context of Doppler geodetic positioning, we refer the reader to [13].

We summarize the properties of these examples in Table 1. The headings of this table have the following meaning :

t : problem type (dense = d, sparse = s, block-angular = b),

n : the number of variables,

m : the number of equations,

τ : the number of nonzero elements in A,

$\kappa(A)$: the condition number of A (estimated from LSQR for the sparse and block-angular matrices, see [23])

τ_{row} : the number of nonzero elements per row in A,

m_{D} : the number of block rows in (54),

n_{D} : the number of block rows in (55)

k_{D} : the number of columns in the blocks B_{ij},

p_{D} : the number of columns in the blocks C_{ij},

$\kappa(B)$: the condition number of the blocks B_{ij},

$\kappa(C)$: the condition number of the blocks C_{ij}.

The right-hand-side vectors b are generated, for each problem, from

$$b = Ax + \delta \tag{56}$$

where x and δ are m-dimensional vectors whose elements are randomly generated (with uniform distribution) in the ranges $[-100, 100]$ and $[-10, 10]$ respectively.

The bounds on the variables are uniformly chosen for all variables, namely

$$-l_j = u_j = \beta \quad (j = 1, \ldots, n), \tag{57}$$

Problem	t	n	m	τ	$\kappa(A)$	τ_{row}	m_{D}	n_{D}	k_{D}	p_{D}	$\kappa(B)$	$\kappa(C)$
P1	d	110	260	28600	1.0×10^2	110						
P2	d	110	260	28600	1.0×10^1	110						
P3	d	150	350	52500	1.0×10^2	150						
P4	d	200	500	100000	1.0×10^2	200						
P5	s	1000	3000	30000	1.18×10^2	10						
P6	s	700	5000	75000	5.19×10^1	15						
P7	s	2000	10000	110000	7.25×10^1	11						
P8	b	163	1050	26250	1.47×10^4	25	7	6	13	12	10^5	10^7
P9	b	244	2640	58080	1.12×10^3	22	12	10	12	10	10^4	10^5
P10	b	335	3150	110250	1.68×10^3	35	10	9	20	15	10^4	10^5

Table 1: Problem characteristics

where the constant β takes the three values 70, 50 and 30 for each of the problems P1 to P10.

The choice of the starting point is quite important for our algorithms. As one of the main feature of the discussed methods is the way in which changes of the active set are handled, the initial choice of active variables plays a significant role. For each problem, we distinguish five different starting points, according to the five following types.

Type A : The starting point is exactly in the center of the feasible region, with all variables being free.

Type B : The point is such that all variables are at one of their bounds, but the choice of these bounds is incorrect. More precisely, if a variable is at one of its bounds at the solution, then it is at the opposite bound at the starting point. Furthermore, if a variable is free at the solution, then it is set to a randomly chosen bound at the starting point.

Type C : Amongst the variables that are at a bound at the solution, 40% are chosen at random and set to this bound, 30% are set to the opposite bound and 30% to the midpoint between their bounds. Amongst the variables that are free at the solution, 70% are set to the midpoint between their bounds, 15% are set to their lower bound and 15% to their upper bound.

Type D : The starting point is chosen as in Type C, except that the proportions related to the variables at their bound at the solution are changed from 40, 30 and 30% to 70, 15 and 15% respectively.

Type E : Amongst the variables that are at a bound at the solution, 60% are chosen at random and set to this bound. All other variables are set to the midpoint between their lower and upper bounds.

Over 800 different tests have been run, using Lötstedt's original method, the Active Set method of Section 2.2.1 and the two projected gradient based methods of Section 2.2.2, where the type and dimension of the problem, the number of active variables at its solution and the starting point were allowed to vary. We will not report the results of these tests in detail, but rather extract from them some meaningful examples to illustrate the points that we discuss below.

In the tables presenting the selected numerical results, the following symbols are used.

Algo : The column with this heading indicates which algorithm has been used. In this column, one can find

- **LO :** the original method of Lötstedt, as proposed in [18],
- **AS :** the Active Set algorithm of Section 2.2.1, which is a variant of LO,
- **PGE :** the projected gradient method of Section 2.2.2, where the EXACT procedure is used for calculating the GCP,
- **PGI :** the projected gradient method of Section 2.2.2, where the INEXACT procedure is used for calculating the GCP,
- **PGER :** the projected gradient method of Section 2.2.2, using the EXACT procedure for calculating the GCP and the variant of Step 2.d,
- **PGIR :** the projected gradient method of Section 2.2.2, using the INEXACT procedure for calculating the GCP and the variant of Step 2.d.

$n_A^{(*)}$ **:** This column gives the number of variables that are at one of their bounds at the problem's solution.

$x^{(0)}$ **:** The type of the starting point used (A to E) is reported in this column.

a/d : This column gives two numbers separated by a "/". The first is the number of variables that are free at the starting point, but fixed at the solution (indicating the amount of activation), while the second is the number of variables that are at one of their bound at the starting point, but are free at the solution (indicating the amount of deactivation).

3.2 A comparison of the original Lötstedt's algorithm with the Active Set algorithm

Before comparing the two projected gradient based methods with the Active Set algorithm, it is useful to verify that the modifications to the original proposal by Lötstedt are beneficial. The expected improvement is twofold.

1. Because the minor iterations are stopped as soon as a bound is violated, one expects the total number of such iterations (and hence the total amount of cpu time) to be substantially lower for the modified method.

2. The possible choice of the starting point on the boundary of the feasible region can induce additional favourable effects due to a lower number of variables to activate (when the active set has been guessed well), and also to the smaller dimension of the successive restricted least squares problems.

Problem	Activities				Algo	Iterations		CPU	
	β	$n_A^{(*)}$	$x^{(0)}$	a/d		Major	Minor	Minor	Total
P2	70	35	A	35/0	LO	38	4032	1267	1275
					AS	36	124	35	42
			E	12/0	AS	15	182	44	48
	30	72	A	72/0	LO	88	6858	1826	1845
					AS	75	148	32	47
			E	24/0	AS	30	139	19	25

Table 2: Original vs modified Lötstedt algorithms

To illustrate these effects, we have chosen to present a typical situation that occurs when solving problem P2 with the two considered algorithms. The results are given in Table 2.

We first observe that both algorithms behave in a similar way as far as the major iterations are concerned. This is due to the fact that the activation/deactivation procedure is identical.

The number of minor iterations is however very different, reflecting the expected advantage to stop the restricted least squares solver as soon as a bound is violated. The modified version (AS) is clearly more efficient in this respect.

We also note that algorithm AS can take advantage of a well guessed initial active set (Type E). In the abscence of reliable information on this set, choosing the starting point in the interior of the feasible domain (Type A) seems a good strategy. In this case indeed no variable needs an initial deactivation, which is quite costly as each deactivation requires a complete restricted least squares solution.

It seems therefore fair to consider algorithm AS as an improvement on the original proposal LO, especially when the problem is large or when a good initial guess of the active set is available.

3.3 Active set vs projected gradient methods

We now compare the active set method AS to the two projected gradient based algorithms PGE and PGI. We successively consider

- the influence of the number of variables that are at one of their bounds at the solution on the overall efficiency of the algorithms,
- the differences between exact and inexact linesearches in the projected gradient based algorithms PGE and PGI,
- a preconditioning strategy and its effect on the three methods.

Because the three algorithms (AS, PGE and PGI) mostly differ in the way activation and deactivation of variables is handled, it is natural to examine their relative performance as a function of the amount of activation/deactivation between the starting point and the problem's solution. In order to make the differences between the algorithms apparent, we consider several problems with varying number of variables at their bounds at the solution. Starting points are also chosen, such that the initial set of active variables differs to a varying extent from that at the solution. This amounts to varying the "a/d" characteristics of the test examples. We present in Table 3 three typical cases, for problems whose matrix structure is also different.

Problem	Activities				Algo	Iterations			CPU		
	β	$n_A^{(*)}$	$x^{(0)}$	a/d		Major	GCP	Minor	GCP	Minor	Total
P3	70	47	A	47/0	PGE	72	122	446	35	211	310
					PGI	80	179	456	93	217	381
					AS	60		2028		879	903
			B	47/150	PGE	71	155	460	43	215	321
					PGI	76	195	459	97	215	379
			C	31/49	PGE	70	130	458	37	217	316
					PGI	82	249	459	121	221	415
			D	16/43	PGE	67	132	445	37	205	303
					PGI	82	282	465	133	219	424
			E	16/0	PGE	69	121	427	35	201	298
					PGI	58	136	420	68	196	316
					AS	29		1942		813	827
	30	121	A	121/0	PGE	12	142	48	33	8	52
					PGI	15	51	50	20	8	42
					AS	123		157		68	110
			B	121/150	PGE	19	174	57	41	10	68
					PGI	14	62	49	22	8	43
			C	81/46	PGE	14	124	48	29	8	50
					PGI	15	57	48	21	8	42
			D	41/30	PGE	11	84	45	20	7	37
					PGI	17	68	53	25	9	49
			E	41/0	PGE	10	73	42	18	6	34
					PGI	12	55	46	19	7	38
					AS	42		75		16	31
P5	70	292	A	292/0	PGE	14	385	60	47	25	82
					PGI	13	154	58	31	23	64
					AS	297		440		242	312
			B	292/1000	PGE	16	468	62	53	26	90
					PGI	11	127	107	27	41	75
			C	195/330	PGE	15	347	60	41	25	77
					PGI	16	191	60	40	25	75
			D	98/282	PGE	12	257	57	33	24	66
					PGI	14	189	60	39	24	73
			E	98/0	PGE	13	211	60	28	25	63
					PGI	12	171	60	35	24	67
					AS	103		247		112	137
	30	705	A	705/0	PGE	8	784	62	79	12	97
					PGI	7	68	60	9	11	25
					AS	708		762		363	527
			B	705/1000	PGE	9	859	62	89	12	107
					PGI	11	106	37	15	7	29
			C	470/329	PGE	8	590	62	61	12	78
					PGI	11	109	38	15	8	30
			D	235/212	PGE	7	337	61	39	12	56
					PGI	11	80	63	12	12	31
			E	235/0	PGE	7	289	61	33	12	50
					PGI	8	85	64	11	12	29
					AS	238		293		93	148

Problem	Activities				Algo	Iterations			CPU		
	β	$n_A^{(*)}$	$x^{(0)}$	a/d		Major	GCP	Minor	GCP	Minor	Total
P9	70	73	A	73/0	PGE	28	83	246	22	245	316
					PGI	28	273	243	211	241	502
					AS	80		834		820	880
			B	73/244	PGE	41	94	258	31	260	363
					PGI	31	257	250	200	248	503
			C	49/86	PGE	32	82	239	24	238	320
					PGI	31	209	244	169	243	467
			D	25/74	PGE	32	71	244	24	242	323
					PGI	32	227	246	180	245	481
			E	25/0	PGE	29	59	241	22	239	312
					PGI	28	219	235	171	233	454
					AS	40		1465		1385	1421
	30	164	A	164/0	PGE	25	162	128	22	98	166
					PGI	15	102	117	73	85	185
					AS	165		269		268	384
			B	164/244	PGE	25	174	128	22	98	165
					PGI	18	132	119	93	87	212
			C	109/82	PGE	19	124	118	17	89	140
					PGI	16	64	117	51	85	165
			D	55/57	PGE	22	64	125	17	94	150
					PGI	17	99	118	72	86	189
			E	55/0	PGE	17	63	116	13	86	130
					PGI	16	60	117	49	85	163
					AS	64		512		374	424

Table 3: Active set vs projected gradient algorithms

In these tables, we only kept the more favourable starting points for algorithm AS (Types A and E), according to the conclusions of the previous subsection.

These results allow us to make to following comments.

- As already noted, AS is quite sensitive to ratio of the number of variables that are active at the solution to that of variables that are active at the starting point.
- In contrast with the situation for algorithms LO and AS, the choice of starting point has a reduced impact on the overall efficiency of the projected gradient methods PGE and PGI. This is especially noticeable for PGI.

  This effect is explained by the feature of both PGE and PGI, that allows activation and deactivation of large numbers of variables in a single iteration.

  Recommending one of the starting point types (A to E) is very difficult: which one produces the best result is very problem dependent.

- Comparing the active set method AS to the projected gradient based algorithms PGE and PGI, we note the poor performance of AS. AS is virtually never better than PGE or PGI, while the cases where their efficiency is similar are also unfrequent.

  The differences between PGE and PGI will be analyzed in more detail below.

- The comments made up to now do not seem to depend on the matrix structure (Recall that P3 is dense, P5 sparse and P9 block angular), except for the behaviour of method AS on problem P9, where starting point A is better than E. This situation however happens very seldom in our tests.

3.4 Exact vs inexact linesearches

We now analyze in more detail the relative merits of the two projected gradient based methods PGE and PGI. To illustrate our discussion, we have chosen a dense (P4) and a sparse problem (P7) and restricted our attention to starting points of types A, C and E only. The results of these runs are presented in Table 4. This table completes Table 3 with problems of a larger size, leaving more room for the activation/deactivation process.

In this table, we note the following points.

- The differences between PGE and PGI in term of major and minor iterations are surprisingly modest. Both methods are relatively equivalent from this point of view, but differ very strongly in their respective numbers of GCP iterations.

- A situation where many variables are activated overall and per major iteration is, on average, unfavourable for the PGE method. This is due to the method of computation of the GCP itself, which sequentially considers a potentially large number of intervals along the piecewise linear arc $x_k + s_k(t)$. PGI, on the other hand, is not impeded by this problem, because it uses an interpolation procedure that jumps over many breakpoints of this arc in one single GCP iteration.

- When the number of variables to activate is small, the differences between PGE and PGI are less marked, with a slight advantage for PGE. The exact linesearch is quite efficient in this case, because the number of successive intervals to analyze is small. We note however that PGI is never very much worse than PGE in term of cpu time.

Problem	Activities				Algo	Iterations			CPU		
	β	$n_A^{(*)}$	$x^{(0)}$	a/d		Major	GCP	Minor	GCP	Minor	Total
P4	70	64	A	64/0	PGE	88	137	864	77	742	968
					PGI	95	197	854	195	736	1092
			C	43/61	PGE	104	235	531	121	477	774
					PGI	112	343	548	315	496	1000
			E	22/0	PGE	85	148	833	80	711	934
					PGI	87	213	835	204	714	1064
	50	97	A	97/0	PGE	54	124	459	64	307	463
					PGI	54	176	464	155	307	554
			C	65/64	PGE	51	159	283	77	195	359
					PGI	51	164	280	142	192	422
			E	33/0	PGE	46	149	446	72	287	438
					PGI	57	189	448	160	293	550
	30	147	A	147/0	PGE	17	176	85	77	34	141
					PGI	26	110	152	80	56	181
			C	98/70	PGE	25	193	151	86	57	185
					PGI	22	98	88	69	35	142
			E	49/0	PGE	14	117	80	54	30	109
					PGI	19	84	85	62	33	128
P7	70	608	A	608/0	PGE	9	800	45	292	63	380
					PGI	9	104	78	75	103	201
			C	405/692	PGE	11	702	46	258	66	354
					PGI	11	148	45	107	63	197
			E	203/0	PGE	9	415	44	158	62	245
					PGI	13	189	47	136	66	234
	50	990	A	990/0	PGE	8	1220	37	423	39	485
					PGI	10	111	66	65	67	156
			C	660/678	PGE	9	944	40	336	43	403
					PGI	12	162	41	95	44	168
			E	330/0	PGE	9	479	39	180	42	247
					PGI	9	112	66	65	67	154
	30	1400	A	1400/0	PGE	7	1505	32	522	22	565
					PGI	10	106	32	48	23	96
			C	933/662	PGE	7	1132	54	387	36	444
					PGI	11	111	35	52	24	103
			E	467/0	PGE	6	579	31	206	21	245
					PGI	11	121	55	54	37	119

Table 4: Comparison of PGE and PGI

3.5 Preconditioning

In this paragraph, we examine the behaviour of the methods on problems that are not so well conditioned. We first analyze the results of test runs for PGE, PGI and AS on problems P1 and P2, as given in Table 5. These two problems have identical matrix structure and dimensions, but differ in conditioning by one order of magnitude.

Problem	Activities				Algo	Iterations			CPU		
	β	$n_A^{(*)}$	$x^{(0)}$	a/d		Major	GCP	Minor	GCP	Minor	Total
P1	70	30	A	30/0	PGE	59	87	549	15	145	188
P2		35		35/0		14	39	106	6	26	39
P1	70	30	A	30/0	PGI	60	122	568	36	150	215
P2		35		35/0		21	51	107	14	27	52
P1	70	30	A	30/0	AS	45		1842		449	460
P2		35		35/0		36		124		35	42
P1	50	50	A	50/0	PGE	37	57	192	9	42	70
P2		48		48/0		13	61	80	8	17	32
P1	50	50	A	50/0	PGI	40	101	185	27	40	87
P2		48		48/0		17	38	84	10	18	37
P1	50	50	A	50/0	AS	67		1083		202	217
P2		48		48/0		53		255		58	69
P1	30	79	A	79/0	PGE	16	101	58	13	8	29
P2		72		72/0		9	99	82	13	10	27
P1	30	79	A	79/0	PGI	18	82	58	17	7	33
P2		72		72/0		10	18	45	5	6	16
P1	30	79	A	79/0	AS	82		159		31	47
P2		72		72/0		75		148		32	47

Table 5: Varying the conditioning

Two main effects are apparent in these results.

- Deteriorating the conditioning, even by a relatively modest amount, has a significant effect on the three considered algorithms. This is especially true for problems where the number of variables to activate is small. In these problems indeed a larger percentage of the computational effort goes in the solution of the restricted least squares subproblems, using a conditioning sensitive technique as LSQR. The number of minor iterations therefore increases substantially, causing a noticeable decrease of efficiency.

- The number of major iterations also increases for the algorithms PGE and PGI when the conditioning deteriorates. This can be expected from the following argument: in methods based on the projected gradient, this vector is used as an indicator of the set of variables that should be fixed at the solution, and this important role is best fulfilled when the gradient vector "points towards the solution", even when the current iterate is far from it. This geometric interpretation is obviously more valid for well conditioned problems, where the level sets of the objective Q are more circular. As soon as conditioning deteriorates, the quality of the prediction then decreases and the number of major iterations grows.

It should be noted however that this unfavourable effect for PGE and PGI is not severe enough to compromise their competitivity with the method AS.

Because modifying the conditioning has a substantial detrimental effect on algorithm performance, it seems important to keep this conditioning as low as possible. A traditional approach to this question is to use a preconditioner, that is a change of variables such that the modified problem has a (hopefully much) better conditioning than the original one.

Applying this idea in the framework of problems with bound constraints is however not as simple as it may seem at first sight. The reason of the complexity is that a good preconditioner (often derived from an analysis of the underlying model) usually transforms a bounded problem in a problem with general linear constraints, destroying in the process a very significant structure. This fact is recognized by Conn, Gould and Toint in [6], where they impose that the preconditioner should be diagonal, and hence transforms bound constraints into bound constraints. Clearly, if the projected gradient and the GCP are to play their role, this must be the case, but restricting the preconditioner to diagonal matrices is usually not the best choice when a restricted least squares subproblem has to be solved in a fixed subspace. The obvious idea is then to use a diagonal preconditioner in the GCP computation, as required by the theory, and a more general one in the subspace of free variables where the restricted least squares subproblem is being solved by LSQR.

In the context of the block angular problems P8 to P10, the preconditioner of choice is nondiagonal and is described in [13].

This simple strategy does not however necessarily produce the desired improvement, as is shown by the results presented in Table 6.

Problem	Activities				Algo	P	Iterations			CPU		
	β	$n_A^{(*)}$	$x^{(0)}$	a/d			Major	GCP	Minor	GCP	Minor	Total
P8	70	63	A	63/0	PGE		74	87	688	24	284	367
						•	***	***	***	***	***	***
					PGI		53	222	659	86	267	394
						•	***	***	***	***	***	***
					AS		74		2481		981	1007
						•	153		2206		874	942

Table 6: The effect of a nondiagonal preconditioner

In this table, the entries "***" mean that the solution was not found after 5000 major iterations!

This behaviour can be explained for PGE and PGI by the fact that two entirely different metrics are used together on the same problem: a diagonal metric is used for the bound handling process (deactivation and activation in the GCP determination) while a nondiagonal one is used in the restricted least squares solution. But this last step is stopped as soon as a new bound is violated, therefore also contributing to the activation process, but in a differently scaled space. The discrepancy between the diagonal and the nondiagonal scalings can and does cause an unwanted situation: a variable is declared free in the diagonally scaled space while its bound is immediately violated in the nondiagonaly scaled one, resulting in a fast oscillation along the relevant facet of the feasible region.

The situation is different for algorithm AS. The discrepancy between the scaling used in the deactivation and that used in LSQR (where the activation takes place) causes a number of variables to be fixed. Only a part of these are active at the solution and a complete subproblem calculation is required for deactivating each one of them. This also causes a significant decrease in efficiency.

This effect is quite unfortunate, but is apparently inherent to the use of a nondiagonal preconditioner in the subproblem solution, which is itself highly useful because it incorporates a substantial ammount of additional information on the problem. A "fix" was thus designed to cope with this situation. The idea is to prevent a fast oscillation between the restricted least squares solution and the active set management by preventing deactivation as long as some progress is not made in the restricted least squares solution.

More specifically, we used the variant of Step 2.d, that restarts the subproblem solver in a subspace with fewer free variables, without allowing deactivation as long as the gradient of Q is not sufficiently reduced, as tested by the new (and costly) condition (50). This is why algorithms PGER and PGIR have been introduced.

This technique is meaningless for algorithm AS, because this method already prevents deactivation of variables as long as the restricted least squares subproblem is not solved up to final accuracy.

If we apply the "restarted algorithms", that is PGER and PGIR, we obtain the results shown in Table 7.

Problem	Activities				Algo	P	Iterations			CPU		
	β	$n_A^{(*)}$	$x^{(0)}$	a/d			Major	GCP	Minor	GCP	Minor	Total
P8	70	63	A	63/0	PGER	•	5	5	224	2	114	210
					PGIR	•	5	5	224	3	114	212

Table 7: The effect of restart when using a nondiagonal preconditioner

We see in these results that the unwanted oscillation effect has disappeared. Both algorithms do perform much better with the preconditioner.

However, even using a restart, preconditioning is not always beneficial for the problems tested, as is shown by Table 8.

In this table, we note two effects.

- As the number of variables active at the solution increases, the efficiency of the preconditioned algorithms decreases. This is caused by the combined effect of the smaller size of the restricted least squares subproblems and their improved conditioning (because of the restriction).

- When comparing the behaviour of the preconditioned PGER and PGIR, we note a striking

Problem	Activities				Algo	P	Iterations			CPU		
	β	$n_A^{(*)}$	$x^{(0)}$	a/d			Major	GCP	Minor	GCP	Minor	Total
P10	70	96	E	32/0	PGE		37	81	295	51	521	692
					PGER	•	4	4	138	5	287	514
					PGI		37	302	298	439	526	1085
					PGIR	•	4	4	138	11	286	518
	50	178	E	60/0	PGE		26	94	187	38	277	400
					PGER	•	3	3	113	4	211	477
					PGI		20	148	179	206	263	535
					PGIR	•	3	3	113	8	211	481
	30	247	E	83/0	PGE		22	116	110	34	137	244
					PGER	•	3	3	131	4	206	535
					PGI		17	151	172	189	207	452
					PGIR	•	3	3	131	8	206	539

Table 8: Using the preconditioner for varying number of active variables

similarity[1]. We explain this effect by the observation that the role of the GCP calculation (which is the only part where PGER and PGIR are distinct) is less important when the restart is used, because a significant number of variables are then activated by the sub-problem solver. The restart technique does in fact reduce the impact of using INEXACT instead of EXACT.

The last question we want to examine is the effect of the restart technique when there is no preconditioning. We present in Table 9 the results obtained when running PGE, PGI and their restarted variants on problem P6 without preconditioner.

Problem	Activities				Algo	Iterations			CPU		
	β	$n_A^{(*)}$	$x^{(0)}$	a/d		Major	GCP	Minor	GCP	Minor	Total
P6	70	192	E	64/0	PGE	7	128	34	93	30	137
					PGER	4	87	91	66	89	212
					PGI	10	118	35	54	31	101
					PGIR	5	72	131	32	107	238
	30	492	E	164/0	PGE	4	186	25	129	10	148
					PGER	2	163	44	116	21	173
					PGI	6	59	26	17	11	37
					PGIR	4	28	66	9	29	77

Table 9: The effect of restart without preconditioner

Because more variables are activated in LSQR when the restart is used, we note a decrease in the number of major and GCP iterations in these cases. The number of minor iterations then increases in counterpart.

[1]Their behaviour is indeed identical in the examples reported, but this is not a general feature. They may differ for other problems, but are still very similar.

We also note that using the restart is not really justified in the abscence of preconditioning. This reduces the freedom of the methods in their bound handling without necessity. It seems therefore that restart is only recommended when a nondiagonal preconditioner is used for better efficiency in the subproblem solution.

4 Conclusion and perspectives

In this paper, we have presented three new methods for iteratively solving well conditioned linear least squares problems with bound constraints on the variables. The first of these methods (AS) is an improvement on a known algorithm proposed by Lötstedt [18]. The improved behaviour is especially noticeable for large scale problems. Two new projected gradient based methods (PGE and PGI) have also been described. They are inspired by recent developments in trust region methods for nonlinear optimization with simple bounds. Their performance on a wide class of problems is shown to be even superior to that of the first algorithm.

Which of the two projected gradient methods is preferable is problem dependent, with PGI being more advantageous when the number of variables activated in one (major) iteration is large. On the whole, PGI may even be considered as slightly more efficient.

When the problem is less well conditioned and a preconditioner is used, a specific restart strategy is proposed, that alleviates the oscillation problems caused by different scalings of the same subspace.

More research is planned in the design of a method that would combine the best features of both PGE and PGI: it is indeed interesting to study an algorithm that would use exact or inexact linesearch in the GCP calculation, depending on the activation level in the past iterations.

5 Acknowledgements

The authors would like to acknowledge useful discussions with Å. Björck, I.S. Duff, G.H. Golub and N.I.M. Gould.

References

[1] D.P. Bertsekas, "On the Goldstein-Levitin-Polyak gradient projection method", IEEE Transactions on Automatic Control, vol. 21, pp. 174–184, 1976.

[2] D.P. Bertsekas, "Constrained Optimization and Lagrange Multiplier Methods", Academic Press, New York, 1982.

[3] Å. Björck, "A Direct Method for Sparse Least Squares Problems with Lower and Upper Bounds", Numerische Mathematik, vol. 54, pp. 19–32, 1988.

[4] Å. Björck, "Least squares methods", to appear in "Handbook of Numerical Analysis" (P.G. Ciarlet and J.L. Lions, eds), Vol. 1, Elsevier/North-Holland, 1989.

[5] D.I. Clark and M.R. Osborne, "On restricted and interval least squares problems", Department of Statistics, Australian National University, Canberra, Australia, 1986.

[6] A.R. Conn, N.I.M. Gould and Ph.L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds", SIAM Journal on Numerical Analysis, vol. 25, pp. 433–460, 1988. Correction, same journal, (to appear).

[7] A.R. Conn, N.I.M. Gould and Ph.L. Toint, "Testing a class of methods for solving minimization problems with simple bounds on the variables", Mathematics of Computation, vol. 50(182), pp. 399–430, 1988.

[8] R.S. Dembo, S.C. Eisenstat and T. Steihaug, "Inexact Newton methods", SIAM Journal on Numerical Analysis, vol. 19, nr. 2, pp. 400–408, 1982.

[9] R.S. Dembo and U. Tulowitski, "On the minimization of quadratic functions subject to box constraints", Technical Report, Department of Computer Science, Yale University, 1984.

[10] J.C. Dunn, "Global and asymptotic convergence rate estimates for a class of projected gradient process", SIAM Journal on Control and Optimization, vol. 19, pp. 368–400, 1981.

[11] P.E. Gill, W. Murray and M.H. Wright, "Practical Optimization", Academic Press, New York, 1981.

[12] A.A. Goldstein, "Convex programming in Hilbert space", Bull. Amer. Math. Soc., vol. 70, pp. 709–710, 1964.

[13] G.H. Golub, P.E. Manneback and Ph.L. Toint, "A comparison between some direct and iterative methods for large scale geodetic least squares problems", SIAM Journal on Scientific and Statistical Computing, vol. 7(3), pp. 799–816, 1986.

[14] R.J. Hanson, "Linear least squares with bounds and linear constraints", SIAM Journal on Scientific and Statistical Computing, vol. 7, pp. 826–834, 1986.

[15] M.T. Heath, "Numerical methods for large linear least squares problems", SIAM J. Sci. Stat. Comput., vol. 5, pp. 497–513, 1984.

[16] M.R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems", J. Res. N.B.S., vol. 49, pp. 409–436, 1952.

[17] E.S. Levitin and B.T. Polyak, "Constrained minimization problems", USSR Comput. Math. and Math. Phys., vol. 6, pp. 1–50, 1966.

[18] P. Lötstedt, "Solving the minimal least squares problem subject to bounds on the variables", B.I.T., vol. 24, pp. 206–224, 1984.

[19] P. Lötstedt, "Numerical simulation of time-dependent contact and friction problems in rigid body mechanics", SIAM Journal on Scientific and Statistical Computing, vol. 5, pp. 370–393, 1984.

[20] J.J. Moré and G. Toraldo, "Algorithms for bound constrained quadratic programming problems", Numerische Mathematik, vol. 55, pp. 377–400, 1989.

[21] D.P. O'Leary, "A generalized conjugate gradient algorithm for solving a class of quadratic programming problems", Linear Algebra and its Applications, vol. 34, pp. 371–399, 1980.

[22] U. Oreborn, "A direct method for sparse nonnegative least squares problems", Department of Mathematics, Thesis 87, Linköping University, Linköping, Sweden, 1986.

[23] C.C. Paige and M.A. Saunders, "LSQR : an algorithm for sparse linear equations and sparse least squares", ACM Transactions on Mathematical Software, vol. 8, pp. 43–71, 1982.

[24] Ph.L. Toint, "Towards an Efficient Sparsity Exploiting Newton Method for Minimization", in "Sparse Matrices and Their Uses" (I.S. Duff, ed.), Academic Press, London, 1981.

[25] Ph.L. Toint, "Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space", IMA Journal of Numerical Analysis, vol. 8, pp. 231–252, 1988.

[26] Y. Vardi, L.A. Shepp and L. Kaufman, "A statistical model for positron emission tomography", Working paper, Bell Laboratories, Murray Hill, USA, 1983.

[27] E.K. Yang and J.W. Tolle, "A class of methods for solving large convex quadratic programs subject to box constraints", private communication, 1989.

[28] P. Zimmermann, "Ein Algorithmus zur Lösung linearer Least Squares Probleme mit unteren und oberen Schranken als Nebenbedingungen", Diplomarbeit, Institut für Angewandte Mathematik, Universität Würzburg, Germany, 1977.