*Article Submitted to ***Journal title****

Solving a sequence of sparse compatible systems

Pablo Guerrero-García*[1] and Ángel Santos-Palomo[1]

[1] *Dpt. Matemática Aplicada, University of Málaga (Spain), Complejo Tecnológico, Campus Teatinos, 29071 Málaga (Spain),*
{pablito,santos}@ctima.uma.es

Abstract

We describe how to use an upper trapezoidal sparse orthogonal factorization to solve the sequence of sparse compatible systems needed to implement sparse reduced gradient versions of certain non-simplex active-set LP methods. For reasons of familiarity, we focus on the reduced gradient non-simplex active-set method of Gill and Murray, but other algorithms of its class like the second stage of Megiddo's cross-over can be implemented with the proposed technique, which clearly shows its suitability to be used in a combined interior-point simplex methodology. Besides two examples illustrating all the given formulae, we report the results obtained with our implementation on top of the sparse toolbox of Matlab 5 when solving the first 15 smallest Netlib problems with a highly-degenerate Phase I, and several paralellizability issues are remarked.

1. The algorithm and its historical perspective

In this paper we describe how to use the upper trapezoidal sparse orthogonal factorization described by Santos & Guerrero [39] to solve the sequence of sparse compatible systems needed to implement sparse reduced gradient versions of certain non-simplex active-set LP methods. For the sake of completeness, let us first describe the algorithm we are going to implement. It is the reduced gradient version of the non-simplex active-set method of Gill, Murray & Wright [21, §8.5], but it is worth noting that the same sparse technique can be used to implement reduced gradient versions of other methods of its class, see [21, p. 285] and [37, 38, 40, and references therein].

*October 5, 2003. Review of a previous version dated in July 1, 2001. Expanded version of a talk presented at 19th Biennial Conf. Numerical Analysis, Dundee, Scotland, June 2001, and at XXVI Con. Nal. Estadística & Investigación Operativa, Úbeda, Spain, November 2001.

The algorithm —given below using pseudo-MATLAB notation— is intended to solve the usual unsymmetric primal-dual pair of linear programs using a non-standard notation (we have deliberately exchanged the usual roles of b and c, x and y, n and m, and (P) and (D), as in e.g. [34, §2]):

$$(P)\quad \begin{array}{ll}\min & \ell(x) \doteq c^T x \\ \text{s.t} & A^T x \geq b\end{array}\ ,\ x \in \mathbb{R}^n \qquad (D)\quad \begin{array}{ll}\max & \mathcal{L}(y) \doteq b^T y \\ \text{s.t} & Ay = c\end{array}\ ,\ y \geq \mathrm{O}$$

where $A \in \mathbb{R}^{n\times m}$ with $m \geq n$ and $\mathrm{rank}(A) = n$. We denote with $\mathcal{F}$ and $\mathcal{G}$ the feasible regions of (P) and (D), respectively.

Non-simplex active-set method for LPs in inequality form

S0. *Initialization.* Set $k \leftarrow 0$ and let $x^{(0)}$ be a given feasible point with working set $\mathcal{B}_0$, working matrix A_0, vector $r^{(0)} \doteq A^T x^{(0)} - b$ of residues, and basis Z_0 of $\mathcal{N}(A_0^T)$.

S1. *Optimality check.* Test whether Lagrange multipliers $y_{\mathcal{B}}^{(k)}$ exist by checking the compatibility of the system $A_k y_{\mathcal{B}} = c$. If $Z_k^T c = \mathrm{O}$ it does, so $y_{\mathcal{B}}^{(k)}$ can be obtained and it can be checked if $y_{\mathcal{B}}^{(k)} \geq \mathrm{O}$ to stop with $x^{(k)}$ optimal, and if not go on to **S2**. If $Z_k^T c \neq \mathrm{O}$ such system is not compatible, but in this case a null-space search direction $d^{(k)} \in \mathbb{R}^n$ can be determined (i.e., it verifies $A_k^T d^{(k)} = \mathrm{O}$ and $c^T d^{(k)} < 0$), as for example a steepest-descent null-space search direction, or else any solution of

$$\left[\begin{array}{c} A_k^T \\ c^T \end{array}\right] d = \left[\begin{array}{c} \mathrm{O} \\ -1 \end{array}\right]; \tag{1}$$

then set $i \leftarrow 0$ to signal that in this case there is no constraint to delete from $\mathcal{B}^{(k)}$ in **S5** and go to **S3**.

S2. *Choose a constraint to delete from the working set and define the search direction.* Using some appropriate rule, choose an index $i \in 1\!:\!m_k$ (corresponding to constraint $\mathcal{B}_i \in 1\!:\!m$) such that $y_{\mathcal{B}_i}^{(k)} < 0$. Define the search direction $d^{(k)} \in \mathbb{R}^n$ as a solution of $A_k^T d = e_i$.

S3. *Calculate the maximum feasible step along the search direction.* Once defined the set $\mathcal{C}_k \doteq \{s \mid a_s^T d^{(k)} < 0\}$ of indices of the *contrary constraints* to the search direction, the maximum step length α from $x^{(k)}$ along $d^{(k)}$ until a contrary constraint is reached is

$$\alpha = \min_{s\in\mathcal{N}^{(k)}} \{\alpha_s\},$$

where the step α_s until the sth constraint along $d^{(k)}$ is

$$\alpha_s = \left\{\begin{array}{cl} r_s^{(k)}/(-a_s^T d^{(k)}) & , \text{ if } s \in \mathcal{C}_k \\ +\infty & , \text{ otherwise} \end{array}\right.$$

S4. *Test for unbounded solution.* If $\alpha = +\infty$ then stop, because $\ell(x)$ is unbounded below in $\mathcal{F}$.

S5. *Choose a constraint to add to the working set and prepare the next iteration.* Using some appropriate rule, choose an index $p \in 1\!:\!m$ (with index $j \in 1\!:\!(m - m_k)$ in $\mathcal{N}^{(k)}$) of a blocking constraint such that $\alpha = \alpha_p$; now, if $i \neq 0$ then set

$$q \leftarrow \mathcal{B}_i^{(k)}, \qquad \mathcal{B}_i^{(k)} \leftarrow \varnothing, \qquad \mathcal{N}_j^{(k)} \leftarrow \varnothing,$$

and delete the ith column from A_k. Finally, set

$$x^{(k+1)} \leftarrow x^{(k)} + \alpha d^{(k)}, \qquad \mathcal{B}^{(k)} \leftarrow [\mathcal{B}^{(k)},\ p], \qquad \mathcal{N}^{(k)} \leftarrow [\mathcal{N}^{(k)},\ q],$$

appending a_p to A_k and setting $k \leftarrow k + 1$. Update the residues and go back to **S1**.

To the best of our knowledge, its first occurrence in the literature traces back to Rosen's 1960 projected gradient method for nonlinear programming [36]. In 1971, Stoer included it too at the end of the last section of a lengthy paper on constrained least-squares problems [41, pp. 408–409]. Two years later, the algorithm appears alone in a celebrated paper of Gill & Murray [18], and steepest-edge rules was adapted for it in a 1983 technical report by Gould [22]. A recent revival has been done in 1998 by Pan [35] under the name (projective) basis-deficiency-allowing dual simplex variation. In all these papers the proposal is a projected gradient version, using the dense QR factorization of A_k, or equivalently, the dense LQ factorization of A_k^T.

The algorithm has also appeared in a 1985 textbook by Osborne [34, §2.5–2.6], where he gave a reduced gradient version (besides the same projected gradient one described above) in terms of *a column* of a matrix Z_k whose columns form a basis of $\mathcal{N}(A_k^T)$. The main improvement that we have found in the description given in the 1991 textbook of Gill, Murray & Wright [21] is that it allows to work with *all the columns* of Z_k, yielding a reduced gradient search direction $-Z_k Z_k^T c$ (although this possibility was already pointed out in Gill & Murray's 1977 survey [20]). Furthermore, a particular case of this algorithm (namely, when $m_k = n$ for all k) is a fairly elegant description of the dual simplex method, whose steepest-edge version has less computational cost than its primal counterpart and hence seems to be the preferred simplex variant nowadays.

But the algorithm has also appeared in others contexts. Dax [12] extends it to deal with additional equality constraints and uses the matrix A_A of active constraints as the working set matrix A_k, and hence m_k can be greater than n or less than n in the algorithm. This leads to the possibility of losing the full column rank feature of A_k when reaching degenerate points, but the way he solve this problem constitute on their own a new constructive proof of Farkas' lemma and hence a new implementation of the primal-dual method of Dantzig, Ford

and Fulkerson [13]. The details can be found in [23], but its least-squares nature rules out an implementation based on [39].

There are other algorithms in which m_k can be greater or less than n. Perhaps the most well-known is the dual vertex-finding algorithm (also known as a *purification* or *reduction* algorithm) described by Murty [33, §11.5] (see also [32, §3.5.4]), but the most subtle occurrence of this kind of algorithms is in the superb three-page paper by Megiddo [30]. He described how to cross-over from an interior point solution to an optimal basis. After a careful reading, it turns out that he first applies Murty's vertex-finding algorithm to obtain a basic feasible solution of $Ay = c, y \geq \mathrm{O}$ from the interior-point feasible solution. As the basic feasible solution can have fewer columns than rows, Megiddo proceeds to a second stage in which a primal vertex-finding algorithm (similar to the one we are implementing, see [21, p. 285]) can be applied because *the first stage does not destroy the primal feasibility of the initial primal interior-point solution*. In this second stage, m_k remains less than or equal to n for all k.

At present, an efficient implementation of Megiddo's cross-over is under research. A promising "An implementation of a strongly polynomial time algorithm for basis recovery" by Bixby and Lustig cited as in preparation in [7, 8] have not seen the light of day yet. In our opinion, it is too costly to proceed as in [8], running the interior-point method first and then the simplex method. The key point is that this would entail the use of *two different sparse data structures*: the static one to hold the Cholesky factor of AA^T in the interior-point phase, and the dynamic one to maintain the LU factorization of the simplex basis. The proposal given in [39] allow us to use *the same data structure* for both phases, so Megiddo's cross-over can be implemented in an efficient sparse manner.

Summing up, in this paper we want to show how to implement sparse reduced-gradient versions of non-simplex active-set methods, that could lead to review the 2002 assertion of Forsgren, Gill & Wright [15, p. 526]

> *Although "nonsimplex" strategies for linear programming were suggested and tried from time to time, they could not consistently match the simplex method in overall speed and reliability.*

To achieve this goal, desirable added features of the sparse implementation must be its suitability to be used in combined interior-point simplex approaches and its parallelizability. We have already addressed the former and we shall address the latter in §5, after the description given in §2 of the way we solve the sequence of sparse compatible systems (with two illustrative examples in §3 and the computational results of §4 obtained in MATLAB with a highly-degenerate Phase I for some NETLIB problems).

2. Solving the systems

In a more general framework, the non-simplex active-set methods for linear programming in which we are interested need to solve a *sequence* of sparse

compatible systems of the form

$$A_k^T x = b^{(k)}, \quad \text{and} \quad A_k y = c^{(k)} \quad \text{or} \quad \begin{bmatrix} A_k^T \\ c^{(k)T} \end{bmatrix} d = \begin{bmatrix} \mathrm{O} \\ -1 \end{bmatrix},$$

where $b^{(k)}$ and $c^{(k)}$ are iteration-dependent vectors, and x, y, and d are unknown vectors. For example:

(i) In the algorithm of Gill, Murray & Wright [21, §8.5] given in §1,

$$y \doteq y_{\mathcal{B}}, \quad c^{(k)} \doteq c, \quad \text{and} \quad ((x \doteq d \text{ and } b^{(k)} \doteq e_{i(k)}) \text{ or } b^{(k)} \doteq b_{\mathcal{B}}).$$

(ii) In the vertex-finding algorithm given e.g. in [21, p. 285],

$$y \doteq \delta_{\mathcal{B}}, \quad c^{(k)} \doteq a_{p(k)}, \quad \text{and} \quad b^{(k)} \doteq b_{\mathcal{B}}.$$

(iii) In the algorithms of Santos [37], Santos & Guerrero [38, 40, and references therein],

$$(y \doteq y_{\mathcal{B}} \text{ and } c^{(k)} \doteq c) \text{ or } (y \doteq \delta_{\mathcal{B}} \text{ and } c^{(k)} \doteq a_{p(k)}), \quad \text{and} \quad b^{(k)} \doteq b_{\mathcal{B}}.$$

From now on, in this section we shall dispense with the superindex $^{(k)}$ in $b^{(k)}$ and $c^{(k)}$ for notational convenience. The matrix $A_k \in \mathbb{R}^{n \times m_k}$ is a "tall and thin" iteration-dependent full column rank matrix with $m_k \leq n$ and $\operatorname{rank}(A_k) = m_k$. Such matrix A_k is a subset of the columns of a *fixed* matrix $A \in \mathbb{R}^{n \times m}$ with $m \geq n$ and $\operatorname{rank}(A) = n$, and A_{k+1} is obtained by appending/deleting columns to/from A_k with $\operatorname{rank}(A_{k+1}) = \operatorname{rank}(A_k) \pm 1$.

We know that there exists an *implicitly defined* permutation Π_k^T of the columns of A_k^T such that

$$A_k^T \Pi_k^T = Q_k R_k \doteq Q_k \begin{bmatrix} R_{\mathrm{i}} & R_{\mathrm{d}} \end{bmatrix}, \quad (Q_k, R_{\mathrm{i}} \in \mathbb{R}^{m_k \times m_k}) \text{and} (R_{\mathrm{d}} \in \mathbb{R}^{m_k \times (n-m_k)}),$$

with R_k upper trapezoidal, R_{i} upper triangular and nonsingular, and Q_k orthogonal; note that $R_k \Pi_k$ is a staircase-shaped, permuted upper trapezoidal matrix, where *the permutation* Π_k *is implicitly defined by the staircase shape of the structure*. The updating and downdating of this upper trapezoidal sparse orthogonal factorization after adding or deleting columns to A_k has been described by Santos & Guerrero [39]. We have used R_{i} and R_{d} rather than a more standard notation R_1 and R_2 to avoid subindex clash problems with R_k. Note that a reordering of the columns of A_k has no impact in R_k, for

$$S_k^T A_k^T \Pi_k^T = (S_k^T Q_k) R_k,$$

thus only a reordering of the rows of Q_k would take place.

Since the columns of R_k^T form a (not necessarily orthogonal) basis for $\mathcal{R}(\Pi_k A_k)$, we categorize this methodology as range-space; anyway, from this information

we can construct a matrix Z whose columns form a (not necessarily orthogonal) fundamental basis for $\mathcal{N}(A_k^T)$:

$$Z = \Pi_k^T \begin{bmatrix} Z_1^T \\ I \end{bmatrix}, \qquad \text{where} \qquad Z_1 \doteq -R_{\mathrm{d}}^T R_{\mathrm{i}}^{-T},$$

where we have also dispensed with the subindex k in Z_k for notational convenience. Note that the submatrices to construct Z are obtained from R_k, so we obtain a sparse Z if we keep R_k sparse and we do not compute Z_1 explicitly.

When $c \notin \mathcal{R}(A_k)$, a first option to obtain a solution of the compatible system $(A_k^T d = 0)\text{and}(c^T d = -1)$ is to project $-c$ onto $\mathcal{N}(A_k^T)$, in the form $-Z(Z^TZ)^{-1}Z^Tc$, but $Z(Z^TZ)^{-1}Z^T$ is a complicated expression. (This solution is a steepest-descent null-space direction for $\ell(x)$.) Fortunately, we have an alternative choice because $-ZZ^Tc$ is also a solution, even if the columns of Z are not orthogonal (It is also a steepest-descent null-space direction for $\ell(x)$, see [21, p. 378].) Note that in this case we have to perform multiplications only by Z_1 and Z_1^T, which implies multiplications by the sparse matrices R_{d} and R_{d}^T, and to solve systems with matrices R_{i} and R_{i}^T (which are triangular sparse matrices and we suppose well conditioned). Thus,

$$d = -ZZ^Tc = \Pi_k^T \begin{bmatrix} -R_{\mathrm{i}}^{-1}R_{\mathrm{d}}w \\ w \end{bmatrix}, \quad \Pi_k c \doteq \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \text{ and } w \doteq R_{\mathrm{d}}^T R_{\mathrm{i}}^{-T} c_1 - c_2.$$

As soon as a matrix Z whose columns are a basis for $\mathcal{N}(A_k^T)$ is available, we can check whether

$$v \in \mathcal{R}(A_k) \qquad \text{iff} \qquad \forall s \in \mathcal{N}(A_k^T),\ v^T s = 0 \qquad \text{iff} \qquad Z^T v = \mathrm{O},$$

so $v \notin \mathcal{R}(A_k)$ if and only if $Z^Tv \neq \mathrm{O}$. Note that if $\Pi_k v = [v_1; v_2]$ with $v_1 \in \mathbb{R}^{m_k}$, then $Z^Tv = Z_1v_1 + v_2 = -R_{\mathrm{d}}^T R_{\mathrm{i}}^{-T} v_1 + v_2$. This is the way that Björck & Duff [9] in 1980 determined if $v \in \mathcal{R}(A_k)$ without computing the solution of $\min_y \|A_k y - v\|_2$ and analyzing the residual when the sparse LU factorization of A_k was available (modified Peters-Wilkinson method, see e.g. [26, Alg. 2]). Numerically we check if $\|Z^Tv\| < \epsilon$ for an ϵ conveniently fixed.

To find a solution of the underdetermined system $A_k^T x = b$, we have $R_k\hat{x} = Q_k^T b$ from $A_k^T = Q_k R_k \Pi_k$, so

$$\hat{x} = \begin{bmatrix} R_{\mathrm{i}}^{-1}(Q_k^T b) \\ \mathrm{O} \end{bmatrix} \quad \text{and} \quad x \doteq \Pi_k^T \hat{x}, \qquad \text{then} \qquad x = \Pi_k^T \begin{bmatrix} R_{\mathrm{i}}^{-1}(Q_k^T b) \\ \mathrm{O} \end{bmatrix}.$$

We have obtained a basic solution, which is not the minimum norm solution unless $R_{\mathrm{d}} = \mathrm{O}$. If Q_k is not available, we can resort to some kind of seminormal equations, premultiplying both sides of the equation by A_k:

$$A_k A_k^T x = \Pi_k^T R_k^T R_k \Pi_k x = A_k b, \qquad \text{then} \qquad R_k^T(R_k(\Pi_k x)) = \Pi_k(A_k b). \tag{2}$$

Now we first solve the compatible system (with unique solution) whose matrix is R_k^T, and then we find a basic solution for the underdetermined system whose (upper trapezoidal) matrix is R_k.

When $c \in \mathcal{R}(A_k)$, the overdetermined system $A_k y = c$ to be solved is equivalent to $\Pi_k A_k y = \Pi_k c = [c_1; c_2]$, which can be solved by $R_k^T(Q_k^T y) = \Pi_k c$ for

$$\hat{y} = R_{\text{i}}^{-T} c_1 \quad \text{and} \quad \hat{y} \doteq Q_k^T y, \qquad \text{then} \qquad y = Q_k(R_{\text{i}}^{-T} c_1).$$

Moreover, when Q_k is not available, we define $y \doteq A_k^T z$ and proceed as in (2) but with $A_k A_k^T z = c$:

$$A_k A_k^T z = \Pi_k^T R_k^T R_k \Pi_k z = c, \qquad \text{then} \qquad R_k^T(R_k(\Pi_k z)) = \Pi_k c. \tag{3}$$

Fixed precision iterative refinement will improve the results obtained in both cases when Q_k is not available; an application of this fact will be given at the end of the next section. Note that (2) and (3) are not the usual seminormal equations, because the systems we are solving are singular but compatible systems with positive semidefinite matrix. Although these systems are usually solved by iterative methods (see Higham [27, §16.4, and references therein]), using direct methods as above to solve them is a novel feature to the best of our knowledge, since the usual way to proceed in the singular case relies on regularization.

3. Two examples

To illustrate the sparse calculations described in §2, let us first consider the application of the algorithm given in §1 to solve the following linear program

$$\begin{array}{lrrrrrrcr}
\text{minimize} & 5x_1 & & -2x_3 & +x_4 & & +4x_6 & , & x \in \mathbb{R}^6 \\
\text{subjet to} & -x_1 & & & & & -x_6 & \geq & -7 \\
& & x_2 & & -x_4 & & & \geq & -1 \\
& x_1 & & -x_3 & & & & \geq & 3 \\
& & x_2 & & -x_4 & & +x_6 & \geq & -1 \\
& & & & & x_5 & +x_6 & \geq & -1 \\
& x_1 & & & & & +2x_6 & \geq & 6 \\
& & & -x_3 & -x_4 & & & \geq & 7 \\
& & x_2 & & +3x_4 & & & \geq & -6
\end{array}$$

which is an LP in inequality form with

$$\left[\begin{array}{c|c} c^T & \\ \hline A^T & b \end{array}\right] = \left[\begin{array}{cccccc|c}
5 & -2 & 1 & & 4 & & \\ \hline
-1 & & & & -1 & & -7 \\
& 1 & & -1 & & & -1 \\
1 & & -1 & & & & 3 \\
& 1 & & -1 & & 1 & -1 \\
& & & & 1 & 1 & -1 \\
1 & & & & & 2 & 6 \\
& & -1 & -1 & & & 7 \\
& 1 & & 3 & & & -6
\end{array}\right].$$

Starting from $x^{(0)} = [6; 0; -7; 0; 0; 0]$, the initial vector of residues is

$$r^{(0)} = A^T x^{(0)} - b = [1; 1; 10; 1; 1; 0; 0; 6].$$

Constraints 6 and 7 are active at this point, so we choose $\mathcal{B}^{(0)} = \{6, 7\}$ as the initial working set. Thus, the initial factorization is such that

$$R_0\Pi_0 \doteq [R_\mathrm{i}\ R_\mathrm{d}]\Pi_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 2 \\ & & -1 & -1 & 0 & 0 \end{bmatrix},$$

and then

$$\Pi_0^T = [e_1, e_3, e_2, e_4, e_5, e_6] = \Pi_0, \quad R_\mathrm{i} = \begin{bmatrix} 1 & 0 \\ & -1 \end{bmatrix}, \quad R_\mathrm{d} = \begin{bmatrix} 0 & 0 & 0 & 2 \\ 0 & -1 & 0 & 0 \end{bmatrix}.$$

The point $x^{(0)} = [6; 0; -7; 0; 0; 0]$ such that $A_0^T x^{(0)} = b_\mathcal{B} = [6; 7]$ could have been obtained from

$$R_0^T(R_0(\Pi_0 x^{(0)})) = \Pi_0(A_0 b_\mathcal{B}) = [6; -7; 0; -7; 0; 12],$$

first solving $R_0(\Pi_0 x^{(0)}) = [6; 7]$ and then $\Pi_0 x^{(0)} = [6; -7; 0; 0; 0; 0]$. Observe that we do not have to explicitly permute the rows of A_0, and that we would have obtained the same result if we maintain $R_0\Pi_0$ in its correct position into the original structure $R \in \mathbb{R}^{6\times 6}$ for the Cholesky factor of AA^T (namely, in the first and third rows) and then set to 1 the free diagonal elements (i.e., superpose $R_0\Pi_0$ on I_6) and compute $R^{-1}(R^{-T}(A_0 b_\mathcal{B})) = R^{-1}(R^{-T}[6; 0; -7; -7; 0; 12])$.

The computation of the first search direction $d^{(0)}$ is as follows:

$$c_1 = [5; -2], \quad c_2 = [0; 1; 0; 4], \quad w = R_\mathrm{d}^T(R_\mathrm{i}^{-T} c_1) - c_2 = [0; -3; 0; 6],$$

$$\Pi_0 d^{(0)} = [-R_\mathrm{i}^{-1}(R_\mathrm{d} w); w] = [-12; 3; 0; -3; 0; 6],$$

hence $d^{(0)} = [-12; 0; 3; -3; 0; 6]$. Now $A^T d^{(0)} = [6; 3; -15; 9; 6; 0; 0; -9]$, so only constraints 3 and 8 are contrary to $d^{(0)}$, but both are blocking constraints. Tie-breaking in favour of the least index constraint we get $p = 3$ after a step length $\alpha^{(0)} = 2/3$. Then $x^{(1)} = [-2; 0; -5; -2; 0; 4]$ with $\mathcal{B}^{(1)} = \{3, 6, 7\}$ and vector $r^{(1)} = [5; 3; 0; 7; 5; 0; 0; 0]$ of residues, so constraint 8 is idle (since it is active in the current iteration point but it is not in the working set).

The updating of the current factorization when constraint 3 is added was illustrated in [39, §3.1], where we got

$$\Pi_1^T = [e_1, e_3, e_4, e_2, e_5, e_6] \neq [e_1, e_4, e_2, e_3, e_5, e_6] = \Pi_1,$$

$$R_\mathrm{i} = \begin{bmatrix} \sqrt{2} & -1/\sqrt{2} & 0 \\ & -\sqrt{6}/2 & -\sqrt{6}/3 \\ & & 1/\sqrt{3} \end{bmatrix}, \quad R_\mathrm{d} = \begin{bmatrix} 0 & 0 & \sqrt{2} \\ 0 & 0 & -\sqrt{6}/3 \\ 0 & 0 & -2/\sqrt{3} \end{bmatrix},$$

and since $Z^T c = [0;0;0]$ in **S1**, system $A_1 y_{\mathcal{B}} = c$ is compatible. Its solution yields the vector $y_{\mathcal{B}}$ of Lagrange multipliers, and it was obtained by solving

$$R_1^T(R_1(\Pi_1 z)) = \Pi_1 c = [5;-2;1;0;0;4],$$

first obtaining $R_1(\Pi_1 z)$, then $\Pi_1 z$ and finally $y_{\mathcal{B}}^{(1)} = A_1^T(\Pi_1^T(\Pi_1 z)) = [3;2;-1]$. Choosing the most negative multiplier to leave the working set implies $q = 7$ and $i = 3$, but before the downdating we have to deal with the underdetermined system $A_1^T d^{(k)} = e_3 = [0;0;1]$ by solving

$$R_1^T(R_1(\Pi_1 d^{(1)})) = \Pi_1(A_1 e_3) = \Pi_1 a_7,$$

first obtaining $R_1(\Pi_1 d^{(1)})$ and then $\Pi_1 d^{(1)} = [0;0;-1;0;0;0]$, so $d^{(1)} = [0;0;0;-1;0;0;0]$.

The downdating of the current factorization when constraint 7 is deleted was illustrated in [39, §4.1]. Since $A^T d^{(1)} = [0;1;0;1;0;0;1;-3]$, constraint $p = 8$ is chosen to be added to the working set. However, computing in **S3** the step length to such idle constraint yields $\alpha^{(1)} = 0/3 = 0$. Thus $x^{(2)} = x^{(1)}$ and then $\ell^{(2)} = \ell^{(1)}$, so there is not a strict decrease in ℓ in spite of $x^{(1)}$ being non-degenerate, for $\operatorname{rank}(A_A) \doteq \operatorname{rank}([a_3,\ a_6,\ a_7,\ a_8]) = 4$.

The new working set is $\mathcal{B}^{(2)} = \{3,6,8\} \neq \mathcal{B}^{(1)}$, and after the addition of constraint 8 we get

$$\Pi_2^T = I_6 = \Pi_2, \qquad R_{\mathrm{i}} = \begin{bmatrix} \sqrt{2} & 0 & -1/\sqrt{2} \\ & 1 & 0 \\ & & 1/\sqrt{2} \end{bmatrix}, \quad R_{\mathrm{d}} = \begin{bmatrix} 0 & 0 & \sqrt{2} \\ 3 & 0 & 0 \\ 0 & 0 & \sqrt{2} \end{bmatrix}.$$

Now we go back to **S1** and $c \notin \mathcal{R}(A_2)$, so we can use (1) to compute the search direction $d^{(2)}$. Such computation was performed by

$$c_1 = [5;0;-2], \quad c_2 = [1;0;4], \quad w = R_{\mathrm{d}}^T(R_{\mathrm{i}}^{-T} c_1) - c_2 = [-1;0;2],$$

$$\Pi_2 d^{(2)} = d^{(2)} = [-R_{\mathrm{i}}^{-1}(R_{\mathrm{d}} w); w] = [-4;3;-4;-1;0;2],$$

which verifies $A_2^T d^{(2)} = [0;0;0]$ and $c^T d^{(2)} = -5$. It turns out that $d^{(2)}$ is a (descent) direction of $\mathcal{F}$, and hence we can conclude that ℓ is unbounded below, because

$$A^T d^{(2)} = [2;4;0;6;2;0;5;0] \geq \mathrm{O} \qquad \text{and} \qquad c^T d^{(2)} = -5 < 0.$$

The example has been designed to deal with the most subtle issues that can be encountered when applying a non-simplex active-set method like that described in §1, and thus it is more illustrative than the one given in [21, §8.5.5]. Furthermore, if the first and last constraint were $x_1 + 0.5x_4 + x_6 \geq -7$ and $x_2 + 3x_4 \geq -16$, respectively, then the algorithm would obtain a solution

$$x^{(3)} = [-\frac{182}{15}; \frac{38}{5}; -\frac{227}{15}; -\frac{118}{15}; 0; \frac{136}{15}], \qquad r^{(3)} = [0; \frac{247}{15}; 0; \frac{383}{15}; \frac{151}{15}; 0; 16; 0],$$

with $\ell^{(3)} = -2$, $\mathcal{B}^{(3)} = \{1, 3, 6, 8\}$ and $y_{\mathcal{B}}^{(3)} = [2; 2; 1; 0]$, starting from the same $x^{(0)}$ and $\mathcal{B}^{(0)}$ and using the same pivoting rules. The details of its computation are left to the reader, but note that strict complementarity does not hold (for $y_8^{(3)} = r_8^{(3)} = 0$), and that $x^{(3)}$ is not a vertex of $\mathcal{F}$.

As an application of the fixed precision iterative refinement cited in §2, let us now give a second example to show how we can adapt the technique of Björck, Eldén & Park [10] of combining corrected seminormal equations as a previous stage for a LINPACK-like downdate of the ith row a_q^T of A_k^T to improve its accuracy. Note that $A_k(Q_k q) = a_q$ is compatible with only one solution, because $a_q \in \mathcal{R}(A_k)$ for A_k and a_q are original data. Then

$$A_k(Q_k q) = \Pi_k^T R_k^T q = a_q \quad \Rightarrow \quad R_k^T q = \Pi_k a_q,$$

which is the same situation as (3). Hence

$$R_k \Pi_k z = q \qquad \text{and} \qquad u = e_i - A_k^T z,$$

where u measures the deviation of $Q_k R_k \Pi_k z = Q_k q$ from e_i when using floating-point arithmetic. Now we have to perform one step of iterative refinement by solving

$$A_k(Q_k \Delta q) = \Pi_k^T R_k^T \Delta q = A_k u \qquad \Rightarrow \qquad R_k^T \Delta q = \Pi_k (A_k u),$$

which is the same situation as (2). Hence

$$R_k(\Pi_k \Delta v) = \Delta q,$$

and thus we can enter the LINPACK-like algorithm given in [39] with the improved quantities

$$q = q + \Delta q \qquad \text{and} \qquad \delta_n = \|u - A_k^T \Delta v\|_2.$$

4. Computational results

In this section we report the results obtained with our implementation on top of the sparse toolbox of MATLAB 5 [17] when solving the first 15 smallest NETLIB [16] problems, namely those with less than 1500 nonzeros. From this set we have only solved those problems with neither BOUNDS nor RANGES sections, since we have not specialized our code to deal with such cases. After reading the problem from NETLIB, we add slacks to convert it into our (D) format and then apply the proposed algorithm to its corresponding (P) format. Table 1 contains the characteristics of the resulting test problems: its name, the optimal value as reported in NETLIB and opposed sign, the dimensions of A, the number of nonzeros in A, b and c, and the total number of nonzeros. The problems were ordered by the right-most column as recommended by Bixby [6]; we have also included as the left-most column what we have called the Bixby number of the problem, which correspond to his order in [6].

#	Name	Optimal value	m	n	nnz(A)	nnz(b)	nnz(c)	nnz
1	AFIRO	$.46475314286E+3$	51	27	102	5	7	114
2	SC50B	$.70000000000E+2$	78	50	148	1	5	154
3	SC50A	$.64575077059E+2$	78	50	160	1	10	171
4	SC105	$.52202061212E+2$	163	105	340	1	20	361
8	STOCFOR1	$.41131976219E+5$	165	117	501	27	8	536
6	ADLITTLE	$-.22549496316E+6$	138	56	424	82	37	543
9	BLEND	$.30812149846E+2$	114	74	522	30	8	560
7	SCAGR7	$.23313898243E+7$	185	129	465	133	53	651
10	SC205	$.52202061212E+2$	317	205	665	1	38	704
12	SHARE2B	$.41573224074E+3$	162	96	777	36	24	837
14	LOTFI	$.25264706062E+2$	366	153	1136	8	49	1193
15	SHARE1B	$.76589318579E+5$	253	117	1179	31	103	1313

Table 1: Test problems for LPs in inequality form

We have tried neither to compare against any alternative algorithm (a good candidate would be a sparse updatable MATLAB implementation of the dual simplex), nor to adjust several tunable features. The sole purpose of the computational experience was to check the performance of the sparse technique under floating-point arithmetic. The computational results was obtained with an Intel Pentium II at 400 MHz with 64 MB RAM, and they are given in Tables 2–4 ordered by the Bixby number; for each problem we give, from left to right, its Bixby number, the density in % of the Cholesky factor of PAA^TP^T, the optimal value obtained, the number of Phase I over total iterations performed and the elapsed time, the minimum of the absolute values of both the residues and the multipliers, and the duality gap. All the tolerances were set up to $\sqrt{\epsilon}$, with ϵ being the machine epsilon. The Phase I was Gass' single artificial one as described e.g. in [21, p. 314] or [40, §3], starting Phase II with empty active set.

We considered two possible orderings and three scaling approaches. When the option `colmmd` was selected, the rows of A were *a priori* ordered using `colmmd` to obtain P above, whereas when `saunpm` was chosen, the columns of A were ordered for A to be in lower block triangular form. On the other hand, `noscale` indicated that no scaling had been applied and `cplex92` was used to select an ℓ_∞-based scaling as described in [6, p. 271], with `cplex93` being a slight variant of this technique.

The results obtained for three out of the six possible combinations are shown in Tables 2, 3 and 4. Note that problems ADLITTLE and BLEND failed to be solved with the combination shown in Table 2, problems SCAGR7 and SHARE1B did so with that shown in Table 3, and problem SHARE1B did so with that shown in Table 4. This results reveal the necessity of a careful implementation of a tie-breaker algorithmic step when using highly-degenerate Phase I's of this kind [25, p. 250], as well as the robustness of the sparse numerical linear algebra machinery employed.

#	Dns	Optimal value	Its	CtSc	MinRes	MinMul	DuaGap
1	56	$4.647531428571429E+2$	12/32	44	$-5E-18$	$-6E-29$	$0E+00$
2	38	$7.000000000000001E+1$	16/79	302	$0E+00$	$3E+01$	$4E-14$
3	38	$6.457507705857030E+1$	16/78	236	$-2E-14$	$-4E-28$	$6E-12$
4	38	$5.220206121170740E+1$	31/166	2850	$-1E-30$	$-2E-24$	$2E-13$
6							
7	44	$2.331389824330983E+6$	50/217	9612	$2E-03$	$1E+01$	$-9E-10$
8	52	$4.113197621944343E+4$	134/267	20772	$6E-01$	$-2E-15$	$7E-09$
9							
10	38	$5.220206121195902E+1$	58/365	229539	$-7E-16$	$-8E-19$	$3E-10$
12	50	$4.157322407414196E+2$	19/190	10573	$-8E-23$	$-8E-22$	$2E-13$
14	44	$2.526470606188035E+1$	27/353	125083	$0E+00$	$-3E-16$	$4E-13$
15	50	$7.658931857933567E+4$	102/315	54855	$1E-04$	$2E+00$	$2E-07$

Table 2: Results obtained with `'saunpm'` and `'noscale'` options

#	Dns	Optimal value	Its	CtSc	MinRes	MinMul	DuaGap
1	28	$4.647531428571429E+2$	9/30	27	$-8E-18$	$0E+00$	$6E-14$
2	19	$7.000000000000040E+1$	48/99	137	$0E+00$	$3E+01$	$4E-13$
3	19	$6.457507705856452E+1$	19/80	137	$0E+00$	$0E+00$	$1E-14$
4	11	$5.220206121170722E+1$	38/166	511	$-7E-19$	$-2E-27$	$-3E-14$
6	28	$-2.254949631623805E+5$	11/122	994	$-9E-13$	$0E+00$	$-9E-11$
7							
8	16	$4.113197621944636E+4$	118/247	1423	$4E+00$	$-5E-29$	$1E-08$
9	49	$3.081214984583058E+1$	83/187	2439	$3E-14$	$-1E-25$	$2E-12$
10	6	$5.220206121170735E+1$	75/349	20075	$-1E-20$	$-5E-22$	$1E-13$
12	28	$4.157322407414199E+2$	27/224	9535	$2E-16$	$-2E-22$	$3E-13$
14	17	$2.526470606200592E+1$	30/539	154615	$0E+00$	$-8E-15$	$1E-10$
15							

Table 3: Results obtained with `'colmmd'` and `'cplex92'` options

#	Dns	Optimal value	Its	CtSc	MinRes	MinMul	DuaGap
1	56	$4.647531428571431E+2$	12/33	44	$-3E-17$	$-1E-28$	$2E-13$
2	38	$6.999999999999999E+1$	16/76	259	$0E+00$	$3E+01$	$0E+00$
3	38	$6.457507705856457E+1$	16/76	198	$-3E-17$	$0E+00$	$7E-14$
4	38	$5.220206121170734E+1$	31/152	1631	$-7E-19$	$0E+00$	$9E-14$
6	79	$-2.254949631623752E+5$	71/180	4031	$-9E-13$	$0E+00$	$5E-09$
7	44	$2.331389824330985E+6$	52/251	35224	$2E-03$	$1E+01$	$9E-10$
8	52	$4.113197621946914E+4$	125/287	28896	$4E+00$	$-1E-16$	$3E-08$
9	85	$3.081215001922825E+1$	101/194	5844	$9E-16$	$-8E-27$	$2E-07$
10	38	$5.220206128785932E+1$	58/436	254794	$-5E-17$	$-4E-19$	$8E-08$
12	50	$4.157322407414188E+2$	18/192	9892	$0E+00$	$-5E-23$	$-1E-12$
14	44	$2.526470606187822E+1$	27/513	175294	$-5E-23$	$-3E-14$	$-2E-12$
15							

Table 4: Results obtained with `'saunpm'` and `'cplex93'` options

5. Parallelizability issues and future work

We have described how to use an updatable upper trapezoidal sparse orthogonal factorization in order to solve the sequence of sparse compatible systems needed to implement the reduced gradient version of a well-known integrant of the class of non-simplex active-set methods for linear programming.

We have not resorted to the usual practice of maintaining a sparse LU factorization despite the remark in [20, p. 355]

> *The effort involved in producing a linear-programming package capable of solving large problems on a day-to-day basis is so great that it has understandably discouraged the implementation of new methods which depart radically from existing practice.*

The reason is that we are interested in solving sparse linear programs with numerical difficulties [29, p. 4]

> *Though the present day numerical tools in the simplex algorithm, first of all the LU factorization of the basis, are believed to be very safe, there can always be some newly arising problems that cause serious numerical troubles and prevent solution.*

Indeed, there is a class of degenerate problems that cannot be solved with robust interior point implementations (e.g., [42, §5]). So there are more important reasons than the mere fact of abandoning the usual practice to prefer a QR factorization instead of an LU: it always exists, it can be computed by a robust and numerically stable method, it fits well in a combined interior-point simplex environment and it can be easily used to solve a compatible system. All of these are well-known advantages to add to that pointed out by Gill, Murray & Saunders [19, p. 1057]

> *For a general sparse matrix B of the type encountered in LP it is possible to compute an orthogonal factorization $B = LDV$ in product form whose density is only slightly greater than that of the triangular factorization $B = LU$. This is a surprising result.*

As future work we plan the description of the orthogonal LV factorization in product form, where the LV factorization of $\Pi_k A_k$ is nothing else than the column-oriented version of our upper trapezoidal sparse QR factorization:

$$\Pi_k A_k = \begin{bmatrix} R_{\mathrm{i}}^T \\ R_{\mathrm{d}}^T \end{bmatrix} Q_k^T \doteq \begin{bmatrix} L_{\mathrm{i}} \\ L_{\mathrm{d}} \end{bmatrix} V_k \doteq L_k V_k,$$

with L_k lower trapezoidal, L_{i} lower triangular and non-singular and V_k orthogonal. The existing relationship is based on the LDV factorization [19, and references therein] of the matrix

$$B_k \doteq \begin{bmatrix} A_{\mathrm{i}} & \mathrm{O} \\ A_{\mathrm{d}} & I \end{bmatrix} = \begin{bmatrix} L_{\mathrm{i}} & \mathrm{O} \\ L_{\mathrm{d}} & I \end{bmatrix} \begin{bmatrix} D_{\mathrm{i}} & \mathrm{O} \\ \mathrm{O} & I \end{bmatrix} \begin{bmatrix} V_{\mathrm{i}} & \mathrm{O} \\ \mathrm{O} & I \end{bmatrix},$$

for the lower triangular matrix of this factorization is that implicitly used when solving systems with matrix $[L_{\mathrm{i}}; L_{\mathrm{d}}]$. The LDV factorization does not need square roots, columns can be added and deleted, and the orthogonal factor is free [19, p. 1055]:

> *The important feature of the matrices L_j and V_j is that both can be constructed from a pair of vectors.*

so we would use the formulae given above with the orthogonal factor available. This could compensate for the loss of numerical accuracy usually attributed to LU-based product forms, with the benefits in parallel computing pointed out by Alvarado, Pothen & Schreiber [2, 3] and exploited by Hall & McKinnon [24].

Curiously, current simplex implementations continue using dynamic LU techniques and they have not taken advantage of modern hardware architectures; in fact, the efficient treatment of sparse dynamic techniques is at present an active research area. On the other hand, working with a static structure allow us to accomplish this goal (one of the main objectives in 90's decade), as was pointed out by Aykanat, Özgü & Güven [4, p. 448]:

> *Efficient vectorization is only possible using static data structures.*

or else Coleman & Hulbert [11, p. 299]:

> *The major work at each iteration of our algorithm is the Cholesky factorization of a positive definite matrix with the size and structure of the Hessian of the quadratic. Hence, the algorithm is suitable for solving large sparse problems and for implementation on parallel computers.*

The papers of Bendtsen et al. [5] and Kratzer [28] are closely related to our sparse approach, both working with SIMD architectures. Bendtsen et al. [5] parallelize the updating and downdating processes in a QR factorization under the same rank assumptions as Björck, Eldén & Park [10]; nevertheless, they consider no static structure although they dispense with the orthonormal factor. On the other hand, Kratzer [28] only parallelizes the updating process in the computation of a sparse row-sequential QR factorization on top of the static structure of George & Heath (see [26, and references therein]).

We also plan to study *matrix enlarging and presolve techniques* [e.g., 1, 31, and references therein] to improve the sparsity of the static structure. In this way we attend an advise of Duff [14, §2]

> *Although we have stressed the importance of using higher level BLAS to obtain high performance, we should be aware that, if the matrix is very sparse and the factors are also, there will normally be no benefit in using BLAS kernels.*

Acknowledgements

The authors thank the referees for having put a lot of work into producing the reports and marking up our previous manuscript. We also want to thank Achiya

Dax at Hydrological Service for providing us the reference [12] that we were not aware of.

References

1. Adlers, M; Björck, Å. (2000). Matrix stretching for sparse least squares problems. *Numer. Linear Algebra Appl;* 7(2):51–65.

2. Alvarado, F.L; Pothen, A; Schreiber, R. (1993). Highly parallel sparse triangular solution. In George, A; Gilbert, J.R; Liu, J.W.H; eds. *Graph Theory and Sparse Matrix Computation*, pp. 141–157. Springer-Verlag, Berlin.

3. Alvarado, F.L; Schreiber, R. (1993). Optimal parallel solution of sparse triangular systems. *SIAM J. Sci. Comput;* 14(2):446–460.

4. Aykanat, C; Özgü, Ö; Güven, N. (1995). Algorithms for efficient vectorization of repeated sparse power system network computations. *IEEE Trans. Power Syst;* 10(1):448–456.

5. Bendtsen, C; Hansen, P.C; Madsen, K; Nielsen, H.B; Pinar, M. (1995). Implementation of QR up- and downdating on a massively parallel computer. *Parallel Computing*, 21(1):49–61.

6. Bixby, R.E. (1992). Implementing the simplex method: The initial basis. *ORSA J. Computing*, 4(3):267–284.

7. Bixby, R.E. (1994). Progress in linear programming. *ORSA J. Computing*, 6(1):15–22.

8. Bixby, R.E; Saltzman, M.J. (1994). Recovering an optimal LP basis from an interior point solution. *Oper. Res. Letters*, 15(4):169–178.

9. Björck, Å; Duff, I.S. (1980). A direct method for the solution of sparse linear least squares problems. *Linear Algebra Appl;* 34:43–67.

10. Björck, Å; Eldén, L; Park, H. (1994). Accurate downdating of least squares solutions. *SIAM J. Matrix Anal. Appl;* 15:549–568.

11. Coleman, T.F; Hulbert, L.A. (1993). A globally and superlinearly convergent algorithm for convex quadratic programs with simple bounds. *SIAM J. Optim;* 3:298–321.

12. Dax, A. (1988). Linear programming via least squares. *Linear Algebra Appl;* 111:313–324.

13. Dantzig, G.B; Ford, L.R; Fulkerson, D.R. (1956). A primal-dual algorithm for linear programs. In Kuhn, H.W; Tucker, A.W; eds. *Linear Inequalities and Related Systems*, pp. 171–181. Princeton University Press, Princeton.

14. Duff, I.S. (1997). Sparse numerical linear algebra: Direct methods and preconditioning. In Duff, I.S; Watson, G.A; eds. *The State of the Art in Numerical Analysis*, pp. 27–62. Oxford University Press, Oxford.

15. Forsgren, A; Gill, P.E; Wright, M.H. (2002). Interior methods for nonlinear optimization. *SIAM Rev;* 44(4):525–597.

16. Gay, D.M. (1985). Electronic mail distribution of linear programming test problems. *COAL Newsletter*, 13:10–12, 1985.

17. Gilbert, J.R; Moler, C.B; Schreiber, R.S. (1992). Sparse matrices in MATLAB: design and implementation. *SIAM J. Matrix Anal. Appl;* 13(1):333–356.

18. Gill, P.E; Murray, W. (1973). A numerically stable form of the simplex algorithm. *Linear Algebra Appl;*, 7:99–138.

19. Gill, P.E; Murray, W; Saunders, M.A. (1975). Methods for computing and modifying the LDV factors of a matrix. *Math. Comp;* 29:1051–1077.

20. Gill, P.E; Murray, W. (1977). Linearly-constrained problems including linear and quadratic programming. In Jacobs, D.A.H; ed. *The State of the Art in Numerical Analysis*, pp. 313–363. Academic Press, New York.

21. Gill, P.E; Murray, W; Wright, M.H. (1991). *Numerical Linear Algebra and Optimization, Vol. 1.* Addison-Wesley, Redwood City.

22. N. I. M. Gould. The generalized steepest-edge for linear programming, Part I: Theory. Tech. report CORR 83-2, Dept. Combinatorics & Optimization, Univ. Waterloo, Ontario, 1983.

23. Guerrero-García, P; Santos-Palomo, Á. (2002). Gyula Farkas would also feel proud. Submitted for publication to *Optim. Meth. Soft.*.

24. Hall, J.A.J; McKinnon, K.I.M. (1998). ASYNPLEX, an asynchronous parallel revised simplex algorithm. *Ann. Oper. Res;* 81:27–49.

25. Hanson, R.J; Wisniewski, J.A. (1979). A mathematical programming updating method using modified Givens transformations and applied to LP problems. *Comm. ACM;* 22(4):245–251.

26. Heath, M.T. (1984). Numerical methods for large sparse linear least squares problems. *SIAM J. Sci. Statist. Comput;* 5(3):497–513.

27. Higham, N.J. (1996). *Accuracy and Stability of Numerical Algorithms.* SIAM Pubs; Philadelphia.

28. Kratzer, S.G. (1992). Sparse QR factorization on a massively parallel computer. *The Journal of Supercomputing*, 6:237–255.

29. Maros, I; Mészáros, C. (1995). A numerically exact implementation of the simplex method. *Ann. Oper. Res;* 58:3–17.

30. Megiddo, N. (1991). On finding primal- and dual-optimal bases. *ORSA J. Computing*, 3(1):63–65.

31. Mészáros, C; Gondzio, J. (2001). Addendum to "Presolve analysis of linear programs prior to applying an interior point method". *INFORMS J. Computing*, 13(2):169–170.

32. Murty, K.G. (1983). *Linear Programming.* John Wiley and Sons, New York.

33. Murty, K.G. (1988). *Linear Complementarity, Linear and Nonlinear Programming.* Heldermann, New York.

34. Osborne, M.R. (1985). *Finite Algorithms in Optimization and Data Analysis.* John Wiley and Sons, Chichester.

35. P.-Q. Pan. A basis-deficiency-allowing variation of the simplex method for linear programming. *Computers Math. Applic.*, 36(3):33–53, August 1998.

36. J. B. Rosen. The gradient projection method for nonlinear programming, Part I: Linear constraints. *J. Soc. Indust. Appl. Math.*, 8:181–217, March 1960.

37. Santos-Palomo, Á. (1998). The sagitta method for solving linear programs. Accepted for publication in *Europ. J. Oper. Res.*

38. Santos-Palomo, Á; Guerrero-García, P. (1998). A feasible-point sagitta approach in linear programming. Submitted for publication.

39. Santos-Palomo, Á; Guerrero-García, P. (2001). Updating and downdating an upper trapezoidal sparse orthogonal factorization. Submitted for publication to *IMA J. Numer. Anal.*

40. Santos-Palomo, Á; Guerrero-García, P. (2001). A non-simplex active-set method for linear programs in standard form. Submitted for publication to *Computers Math. Applic.*.

41. Stoer, J. (1971). On the numerical solution of constrained least squares problems. *SIAM J. Numer. Anal;* 8(2):382–411.

42. Stojković, N.V; Stanimirović, P.S. (2001). Two direct methods in linear programming. *Europ. J. Oper. Res;* 131:417–439.