BIT
2000, Vol. 40, No. 4, pp. 001–004
0006-3835/00/4004-0001 $15.00

SOLVING A SEQUENCE OF SPARSE LEAST SQUARES PROBLEMS *

ÁNGEL SANTOS-PALOMO and PABLO GUERRERO-GARCÍA

Department of Applied Mathematics, University of Málaga, Complejo Tecnológico, Campus Teatinos, 29071 Málaga, Spain, emails: {santos,pablito}@ctima.uma.es

Abstract.

We describe how to maintain an explicit sparse orthogonal factorization in order to solve the sequence of sparse least squares subproblems needed to implement an active-set method to solve the nonnegative least squares problem for a matrix with more columns than rows. In order to do that, we have adapted the sparse direct methodology of Björck and Oreborn of late 80s in a similar way to Coleman and Hulbert, but without forming the hessian matrix that is only positive semidefinite in this case. We comment on our implementation on top of the sparse toolbox of MATLAB 5, and we emphasize the importance of Lawson and Hanson's NNLS active-set method as an alternative to Dax's constructive proof of Farkas' lemma; these methods can be used as Phase I for a non-simplex active-set linear programming method.

AMS subject classification: 90C05, 90C60.

Key words: Sparse orthogonalization, Givens rotations, active-set methods, Farkas' lemma, linear programming.

1 The problem we want to solve.

In the linear least squares problem with nonnegative variables:

$$\text{(1.1)}\quad \text{minimize} \quad \frac{1}{2}\|Ay - c\|_2^2, \quad A \in \mathbb{R}^{n\times m},\ y \in \mathbb{R}^m, \quad \text{subject to} \quad y \geq \mathrm{O},$$

it is usual [1, 5, 25] to assume $n > m$ and that the matrix A has full column rank, so the problem is strictly convex and its solution is unique. To compute such solution, Lawson and Hanson's NNLS method [21, §23.3] can be used. However, they did not make any assumption about the dimensions of the matrix A in their original presentation; indeed, the proof of the convergence of the NNLS method only needs that, in the sequence of least squares subproblems of the form $\min \|A_k z - c\|_2$ that this method has to solve, every matrix A_k has full column rank.

In this paper we shall only deal with the case $n < m$, with A being a sparse full row rank matrix. It is also a convex program (although not strictly, so there is no unique solution) and we call it the *convex nonnegative least squares*

*Review (September 2003) of the first version submitted to BIT in October 2001.

(NNLS) problem. In fact there may be infinitely many vectors y that produce the (unique) minimum value of the length of the residual. The one we are interested in does not have any particular distinguishing property as in algorithm 1 of Lötstedt [22] (i.e., it does not necessarily have the least possible number of nonzero components, nor the minimum euclidean norm among all solutions).

The availability of a sparse algorithm for (1.1) makes possible an elegantly simple way (due to Cline) of solving sparse least distance programming (LDP) problems [21, §23.4]; moreover, least squares full column rank problems with linear inequality constraints like those arising in constrained curve fitting [21, §23.7] can be transformed into an LDP as described in [21, §23.5]. Anyhow, our main interest in problem (1.1) lies in the fact that its solution method constitutes a suitable Phase I for a non-simplex active-set linear programming method, in a similar way to the active-set algorithm P1 of Dantzig et al. [8]; even so we do not want to fail to emphasize in this paper that *to obtain a constructive proof of the Farkas' lemma it would suffice to apply to problem (1.1) the NNLS method itself in the original Lawson and Hanson's [21] form*.

LEMMA 1.1. *(Farkas' lemma) Let $A \in \mathbb{R}^{n\times m}$ and $c \in \mathbb{R}^n$, then exactly one of the following propositions must be true: or*

$$c = Ay \qquad \text{for some} \qquad y \geq O, \tag{1.2}$$

or else there exists a vector d verifying

$$A^T d \geq O \qquad \text{and} \qquad c^T d < 0. \tag{1.3}$$

In fact, Dax [10] has given another active-set method (different from NNLS) to solve the same problem (1.1), thus providing a different constructive proof of Farkas' lemma. First, he proves the following theorem, whose corollary acts as the separating hyperplane theorem.

THEOREM 1.2. *(Kuhn-Tucker) Let $y^* \in \mathbb{R}^m$, $d^* = Ay^* - c \in \mathbb{R}^n$ and $w^* = A^T d^* \in \mathbb{R}^m$; then y^* is a solution of (1.1) if and only if $y^* \geq O$, $w^* \geq O$ and $(y^*)^T w^* = 0$.*

COROLLARY 1.3. *Let $y^* \in \mathbb{R}^m$ be a solution of (1.1) and $d^* = Ay^* - c \in \mathbb{R}^n$. It holds that if $d^* = O$ then y^* is a solution of $Ay = c$ and $y \geq O$; otherwise, d^* is a solution of $A^T d \geq O$ and $c^T d < 0$.*

In this way, the only thing that must be established is the existence of a solution y^* of (1.1); to do this, Dax [10] has developed his active-set method using the Kuhn-Tucker conditions, but the NNLS method itself can be used instead.

The algorithms of Dantzig et al. [8], Portugal et al. [25, pp. 627–628] and Dax [10] can be thought of as variants of Lawson and Hanson's NNLS method [21, §23.3]. Since NNLS is the most well-known among them, it has been chosen in this paper to illustrate how its sparse implementation can be accomplished, attending in this way a claim made in 1993 in the "Future work" section of [8]:

> *(Algorithm) P1 should be implemented using sparse matrix methods (...) However, this would require substantial effort, as our algorithm uses QR factorization, and would thus be more complicated to update than the sparse LU factorization*

as well as another one by Saunders in the same paper [8, p. 429]:

> *These results seem significant enough to spark interest in a sparse NNLS implementation*

This claim has just been done again recently (in October, 2002) by Barnes et al. in the "Conclusions and future research" section of [3], where (1.1) occurs as a subproblem in their primal-dual active-set LSPD method (closely related to that of Dax [9], as we show in [19]) for linear programming:

> *An efficient implementation of the LSPD algorithm using QR update techniques for solving the NNLS problem is crucial in order to make it competitive*

This serves to highlight the importance of a sparse implementation of an active-set method for solving the convex NNLS problem, which constitutes the main concern of this paper.

The rest of this paper is structured as follows. A description (revealing some interesting geometrical features) of Lawson and Hanson's NNLS method and the analysis of its convergence for the problem mentioned above is given in §2, along with several remarks about closely related algorithms. In §3 we indicate how to maintain an explicit suitable sparse orthogonal factorization by adapting the sparse direct methodology of Björck [4] and Oreborn [23] in a similar way to Coleman and Hulbert [6], but without forming the hessian matrix that is only positive semidefinite in this case. Finally, we conclude in §4 with several algorithmic alternatives based on the dual problem of (1.1) that can also take advantage of the sparse technique described in §3.

From now on, we shall denote with $\|\cdot\|$ the euclidean vector norm. I and O are respectively the identity and the zero matrix of appropriate dimension. $\mathcal{R}(\cdot)$ and $\mathcal{N}(\cdot)$ are respectively the range and null space of a certain matrix. The orthogonal projector onto a subspace $\mathcal{S}$ is denoted by $P_{\mathcal{S}}$. Subindexing and construction of matrices is done using MATLAB notation.

2 Active-set methods for convex NNLS problem.

The NNLS method works with *complementary solutions*, namely those vectors $[y; w] \in \mathbb{R}^{2m}$ verifying $w = A^T(Ay - c) \in \mathbb{R}^m$ and $y^T w = 0$. Let us denote the working set by $\mathcal{B}_k \subset 1 : m$ (with $|\mathcal{B}_k| = m_k$) and its complement by $\mathcal{N}_k = (1 : m) \backslash \mathcal{B}_k$, partitioning accordingly the matrix $A = [A_k, N_k] \in \mathbb{R}^{n \times m}$ and the vectors $y = [y_B; y_N]$ and $w = [w_B; w_N]$. We include in $\mathcal{B}_k$ the indices of the strictly positive components of y, thus $\mathcal{B}_k$ denotes our set of free variables and $\mathcal{N}_k$ our set of fixed variables at their bounds (i.e., active constraints with respect to $y \geq$ O). To obtain a complementary solution we set to zero the so-called

nonbasic variables y_N and w_B, whereas to obtain the values of the so-called *basic variables* y_B and w_N we solve the unconstrained subproblem

$$\min \|A_k y_B - c\| \tag{2.1}$$

where $A_k \in \mathbb{R}^{n\times m_k}$ has full column rank, and then

$$w_N = N_k^T(A_k y_B - c). \tag{2.2}$$

Note that if we define $d^{(k)} \doteq A_k y_B - c$ then we have $w_B = A_k^T d^{(k)} = \mathrm{O}$, since $d^{(k)} \in \mathcal{N}(A_k^T)$ is the opposite of the residual of the linear least squares problem (2.1). As was pointed out in the refereeing process, the solution technique employed in (2.1) is crucial for the success of the algorithm; nevertheless, we shall defer this issue until the beginning of §3 to clearly differentiate between the description of the method (given below) and its sparse implementation.

ALGORITHM 2.1. *Convex NNLS method*

S0. *Initialization. Let* $k \leftarrow 0$ *and define* $\mathcal{B}_k = \varnothing$, $\mathcal{N}_k = 1:m$, $y^{(k)} = O$ *and* $w^{(k)} = -A^T c$.

S1. *Check for optimality and pick a constraint to add. Compute* $w_p = \min\{w_i : i \in \mathcal{N}_k\}$. *If* $w_p < 0$ *then add* p *to* $\mathcal{B}_k$ *and delete* p *from* $\mathcal{N}_k$; *otherwise, stop with* $y^{(k)}$ *as optimal solution.*

S2. *Determine the search direction, the maximum step length along such direction and the constraints to drop. Compute temporal variables* $\bar{y}_B$ *by solving (2.1). If* $\bar{y}_B > O$ *then let* $y^{(k+1)} \leftarrow [\bar{y}_B; O]$ *and go to* **S3**; *otherwise, let* q *be a constraint such that*

$$\alpha_k = \frac{y_q}{y_q - \bar{y}_q} = \min\left\{\frac{y_i}{y_i - \bar{y}_i} : (i \in \mathcal{B}_k) \text{ and } (\bar{y}_i \le 0)\right\}$$

and let $y^{(k+1)} \leftarrow [y_B + \alpha_k(\bar{y}_B - y_B); O]$, *delete from* $\mathcal{B}_k$ *all index* j *(*q *among them) such that* $y_j = 0$, *add them to* $\mathcal{N}_k$ *and go back to* **S2** *with* $k \leftarrow k+1$.

S3. *Prepare the next iteration. Compute* w_N *by (2.2) and go back to* **S1** *with* $k \leftarrow k+1$.

A complementary solution is feasible if $y_B \ge \mathrm{O}$ and $w_N \ge \mathrm{O}$. A *principal pivoting algorithm* computes successive infeasible complementary solutions until it reaches a feasible one, and such algorithm is called *single pivoting* if there is only one element changing in the involved sets in every iteration (if not, it is called *block pivoting*). The description of the NNLS method that we have done before is not the same as that given in Portugal et al. [25, pp. 627–628] as single principal pivoting algorithm, because although both algorithms add constraints one after another, NNLS can drop a block of constraints and also imposes that $y_B > \mathrm{O}$. The single principal pivoting algorithm of Portugal et al. [25] allows $y_B \ge \mathrm{O}$, which implies two additional changes in step **S2** of algorithm 2.1:

"If $\bar{y}_B \geq \mathrm{O}$" instead of "If $\bar{y}_B > \mathrm{O}$", and "$\bar{y}_i < 0$" instead of "$\bar{y}_i \leq 0$". These differences are important from both a theoretical and a practical point of view, because it is not possible to ensure in the latter case that $\alpha_k > 0$ and thus a strict improvement has to be ruled out; moreover, "If $\bar{y}_B \geq -\epsilon$" instead of "If $\bar{y}_B > \epsilon$" would be used in fixed-precision arithmetic, where ϵ is some prescribed tolerance, thus allowing small negative elements in y_B that can cause a change in the behaviour of the algorithm. Following Portugal et al. [25, p. 629], the problem we are dealing with cannot be solved with a block principal pivoting algorithm, because they cannot be applied to rank-deficient linear least squares problems with nonnegative variables. Although single addition is used in algorithm 2.1, note that we do not rule out the possibility of multiple deletion; nevertheless, the implementation given in §3 only deals with single deletion, and multiple deletion is accomplished by a sequence of single deletions.

Now let us analyze the convergence of algorithm 2.1. It is straightforward to prove that A_k has full column rank (since the constraints added are contrary to the null-space descent direction $d^{(k)} \doteq A_k y_B - c$ with respect to $c^T x$). It is also easy to prove[1] that if $A_{k+1} = [A_k, a_p]$ has full column rank and $d^{(k)}$ verifies

$$A_{k+1}^T d^{(k)} = w_p \cdot e_{m_k+1}, \qquad w_p < 0,$$

then $\tilde{y}_p$ is stricly positive, where $\tilde{y}_p$ is the last component of the solution $\tilde{y}_B$ of $\min \|A_{k+1} y_B - (-d^{(k)})\|$. Lastly we need the following theorem too, which is not proven in [21].

THEOREM 2.1. *Let $A_{k+1} = [A_k, a_p]$ be a full column rank matrix, $d^{(k)} = -P_{\mathcal{N}(A_k^T)} c$ and $a_p^T d^{(k)} = w_p < 0$. Then the last component $\tilde{y}_p$ of the solution $\tilde{y}_B$ of $\min \|A_{k+1} y_B - (-d^{(k)})\|$ coincides with the last component $\hat{y}_p$ of the solution $\hat{y}_B$ of $\min \|A_{k+1} y_B - c\|$.*

PROOF. Using Schur complements there exist block-triangular matrices L_S and R_S such that $L_S A_{k+1}^T A_{k+1} = R_S$, namely

$$\begin{bmatrix} (A_k^T A_k)^{-1} & \mathrm{O} \\ -a_p^T A_k^{T\dagger} & 1 \end{bmatrix} \begin{bmatrix} A_k^T A_k & A_k^T a_p \\ a_p^T A_k & a_p^T a_p \end{bmatrix} = \begin{bmatrix} I & A_k^\dagger a_p \\ \mathrm{O}^T & a_p^T (a_p - A_k A_k^\dagger a_p) \end{bmatrix}.$$

Now we can premultiply by L_S both sides of the equalities

$$(A_{k+1}^T A_{k+1}) \tilde{y}_B = -A_{k+1}^T d^{(k)} \qquad \text{and} \qquad (A_{k+1}^T A_{k+1}) \hat{y}_B = A_{k+1}^T c,$$

leading to the systems

$$R_S \tilde{y}_B = \begin{bmatrix} \mathrm{O} \\ -w_p \end{bmatrix} \qquad \text{and} \qquad R_S \hat{y}_B = \begin{bmatrix} A_k^\dagger c \\ a_p^T (c - A_k A_k^\dagger c) \end{bmatrix}.$$

From the leftmost system we obtain that $\tilde{y}_p = -w_p / (a_p^T (a_p - A_k A_k^\dagger a_p))$, and from the rightmost one

$$\hat{y}_p = \frac{a_p^T (c - A_k A_k^\dagger c)}{a_p^T (a_p - A_k A_k^\dagger a_p)} \doteq \frac{-a_p^T d^{(k)}}{a_p^T (a_p - A_k A_k^\dagger a_p)} \doteq \frac{-w_p}{a_p^T (a_p - A_k A_k^\dagger a_p)} = \tilde{y}_p.$$

[1] It is an alternative wording of lemma (23.17) of Lawson and Hanson [21].

□

From theorem 2.1 we have that if $\bar{y}_B$ is the solution of $\min \|A_{k+1} y_B - (-d^{(k)})\|$ or the solution of $\min \|A_{k+1} y_B - c\|$, then $\bar{y}_p > 0$. The termination of the inner loop **S2** can be obtained from the facts that at least the index q is deleted from $\mathcal{B}_k$ in each iteration of this step and that at least a positive component has to be left in $\mathcal{B}_k$, namely that corresponding to $\bar{y}_p$. Hence, if $\mathcal{B}_k$ has t indices when **S2** is reached, at most $t-1$ iterations can be achieved before exiting towards **S3**. The finite termination of the whole algorithm is proved from the fact that $\|d^{(k)}\|$ has a strictly lower value each time **S1** is reached, so $y^{(k)}$ and its associated set $\mathcal{B}_k$ will be different from every one occurred before; since $\mathcal{B}_k$ is a subset of $1:m$ and there is only a finite number of such subsets, the whole algorithm finishes after a finite number of iterations.

Another issue not treated in [21] is the occurrence of ties in **S1**; an illustrative example can be found in [18, pp. 78–80]. Although this possibility cannot be ruled out, the strict decrease of $\|d^{(k)}\|$ is not compromised at all. In practice, a least-index tie-breaker would suffice.

Dax [10] studied the convergence of another active-set method to solve (1.1); here we only point out that it also has to solve a sequence of least squares problems (in this case of the form $\min \|A_k \Delta y_B - d^{(k)}\|$), hence it can also be implemented with the same sparse techniques we are going to introduce in the next section. The same is true for the variants of the NNLS method given by Dantzig et al. [8] and Portugal et al. [25, pp. 627–628].

All the algorithms mentioned so far in this section are "build-up" methods [8, p. 429] in the sense that they avoid the calculation of an initial factorization of A, which otherwise may well dominate the rest of the computation. Following Clark and Osborne [7, p. 26], these algorithms are grouped under the name RLSA and they have the advantage of being self-starting and avoiding a possibly badly conditioned full system if the subproblems leading to an optimal solution are well conditioned. By exchanging the roles of y and w, Clark and Osborne developed the RLSD class of algorithms, which also dispense with the initial factorization of A; although A was restricted to have full column rank in the original description of these algorithms [7, and references therein], the convex case considered here has been addressed in [24, p. 833]. Henceforth, the sparse counterpart, which is not treated in these papers, can also be dealt with the sparse technique of §3.

There exist alternative approaches based upon identifying (1.1) as a quadratic programming (QP) problem of a rather simple special kind and particularizing general QP *primal* techniques. As an example, the celebrated general QP primal algorithm of Gill and Murray [13] has been particularized to obtain "build-down" algorithms for (1.1), like algorithm 1 of Lötstedt [22] and LSSOL (LS1 option) of Gill et al. [15]:

(i) Lötstedt [22, p. 210] updates a *dense* QR decomposition of a matrix A_F that consists of the columns of A corresponding to his set $\mathcal{F}$ of free variables. Moreover, $\mathcal{F}$ is selected in such a way that A_F has full column rank initially and such that $\operatorname{rank}(A) = \operatorname{rank}(A_F)$; thus, our assumption that

rank$(A) = n$ would imply that $|\mathcal{F}| = n$ and then his procedure would need an initial QR factorization of A to choose the initial $\mathcal{F}$.

(ii) The LS1 option of LSSOL maintains the *dense* QR factorization of A [8, p. 427]. As pointed out in [15, §2] and [5, §5.2.3], in the full rank case this method is essentially equivalent to Stoer's 1971 algorithm, which is not restricted to the full-column rank case [31, p. 405]. Björck and Oreborn have developed a *sparse* implementation for a particularization of Stoer's algorithm to simple bounds [4] and non-negative constraints [23], but their algorithms only consider the positive definite case [6, p. 375] and would not be efficient in the $n \ll m$ context [8, p. 429].

On the other hand, we can also particularize general QP *dual* algorithms to obtain "build-up" methods for (1.1). In fact, Dantzig et al. pointed out in [8, p. 421, and references therein] that Lawson and Hanson's NNLS method can be obtained in this way from that of van de Panne and Whinston, and NNLS has been generalized to deal with simple bounds [21, pp. 279–283] or general linear inequality constraints [32]. The sparse case for this generalizations, not treated in these works, could be addressed as described in §3.

A numerical comparison between a "build-up" (Dantzig et al.'s P1) and a "build-down" (Gill et al.'s LSSOL) method for the dense version of the convex NNLS problem corresponding to the smallest NETLIB linear programs is included in [8, §5.2], where a clear advantage was obtained by the former in both iteration counts and run times, and so we do not treat this issue further here. What we are going to show now is how the sparse methodology (not the algorithm!) of Björck [4] and Oreborn [23] to implement a "build-down" method in the full-column rank case can also be adapted to implement a "build-up" one in the $n \ll m$ full-row rank context.

3 A sparse implementation.

In this section we are going to adapt the sparse direct methodology of Björck [4] and Oreborn [23] to be able to apply the sparse NNLS algorithm with a "short-and-fat" matrix A, i.e., with more columns than rows. They proposed an active-set algorithm for the sparse least squares problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} y^T C y + d^T y, \quad y \in \mathbb{R}^m \\ \text{subject to} & l \leq y \leq u \end{array}$$

with C positive definite. In our problem

$$C = A^T A \quad \text{and} \quad d = -A^T c \quad \text{and} \quad \forall i \in 1:m \ , \ l_i = 0 \text{ and } u_i = +\infty$$

but C is positive semidefinite, hence to maintain a sparse QR factorization of the working set matrix we proceed in a similar way as in Coleman and Hulbert [6], but without forming the hessian matrix C.

Given a fixed matrix $A \in \mathbb{R}^{n \times m}$ of full row rank with $m \geq n$, we recur the triangular factor R_k of the sparse QR factorization of $A_k \in \mathbb{R}^{n \times m_k}$ with

$m_k \leq n$, where A_k is a linearly independent subset of the columns of A and it is generated from A_{k-1} by adding/dropping a column. Furthermore, A is not restricted to be triangular as in [4]. It is worth noting that in a sparse MATLAB NNLS implementation due to Adlers [1, §5.7], R_k is not updated but recomputed from scratch every iteration by calling a sparse Cholesky routine after explicitly forming the matrix of the normal equations. To be as close to the implementation as possible, we shall use $L_k \doteq R_k^T$ to explain implementation details and reserve R_k for more theoretical explanations.

Since we are going to do without the orthogonal factor due to sparsity considerations, the solution of the least squares subproblem (2.1) is done by the CSNE method (see, for example, [5, p. 126]) as suggested by Björck [4, §5], which amounts to perform a step of iterative refinement after having found the solution $\bar{y}_B$ of the seminormal equations

$$R_k^T R_k y_B = A_k^T A_k y_B = A_k^T c,$$

and then computing $d^{(k)} = A_k \bar{y}_B - c$ and $w_N = A_k^T d^{(k)}$. This contrasts with the dense updatable QR-based approach used by Lötstedt [22, p. 210]. The reason why we have chosen a direct approach to solve (2.1) is that we only have to solve one *isolated* convex NNLS problem in the Phase I linear programming application described in §1. However, a *sequence* of closely related convex NNLS problems has to be solved in the primal-dual active-set application, and an iterative approach would perhaps be more efficient in this case as reported by Lötstedt [22] and recommended by Dax [9]. Nevertheless, Dantzig et al. [8, §5.3] and Barnes et al. [3] solve each convex NNLS problem with a dense NNLS direct subroutine, thus avoiding slow convergence when dealing with ill-conditioned problems.

We have developed a MATLAB toolbox based upon the following subsections, and it has been successfully used to implement linear programming methods [28, 18, 20]. The proof of the result (given in §3.1) needed to set up the static data structure, as well as the correctness of the techniques to drop (§3.2) and add (§3.3) a constraint, are essentially due to Björck [4] (see also [6, 5]). The reason why these details are included is that we want to emphasize the fact (not included in [4, 5, 6]) that a different choice in the order in which Givens rotations are used in the classical dense case (see, e.g., [17, §12.5.2]) leads us to an efficient sparse updating scheme. Moreover, in our sparse scheme A_k is not restricted to be a column-echelon submatrix (as $R_{\mathcal{F}_k}$ is in [4, 5]) nor $A_k^T A_k$ has to be formed (as in [6]). To describe these techniques we use the notation of Coleman and Hulbert [6], where $\ell_{\vee p}$ denotes the bottom part of $L(:, p)$ and $\ell_{\wedge p}$ the top part. The notation for the Givens rotations is given in [17, p. 216].

3.1 Setting up the static structure.

First we determine a permutation matrix P such that $P^T A^T A P$ has a Cholesky factor as sparse as possible. In MATLAB we can resort to:

```
>> p = colmmd(A);
```

```
>> A = A(:,p);
>> [count,h,parent,post,R]=symbfact(A,'col');
>> spy(R');
```

Note that we do not form $(AP)^TAP$ and that both `colmmd` and `symbfact` perform correctly even although $A^TA \in \mathbb{R}^{m\times m}$ is only positive semidefinite; furthermore, we *a priori* permute the columns of A in such a way that the natural order coincides with the computed order. *This static structure R^T has enough space to accommodate any L_k, so it constitutes an upper bound for the memory needed [4, Theorem 4.1].* To do that, it is crucial that instead of labeling each row and column of both $A_k^TA_k$ and L_k with consecutive numbers, we label them with the number of the row and column of A^TA from which it comes. As an example, if $m = 6$ and $\mathcal{B}_k = [2, 4, 5]$, then the matrix L_k will be stored in the triangular $m_k \times m_k$ matrix $R(\mathcal{B}_k, \mathcal{B}_k)^T$ constructed by intersecting the rows and columns of R^T whose indices are in $\mathcal{B}_k$:

$$\mathcal{B}_k = [2,4,5] \text{ and } R^T = \begin{bmatrix} \times & & & & & \\ & \otimes & & & & \\ \times & & \times & & & \\ & \otimes & \times & \otimes & & \\ & & & & \otimes & \\ \times & \times & \times & \times & \times & \times \end{bmatrix} \rightsquigarrow\ L_k = R(\mathcal{B}_k, \mathcal{B}_k)^T.$$

Therefore, the column ordering of A dictates that of A_k.

3.2 Dropping a constraint.

Let $\mathcal{B}_k = [B_1, B_2, \ldots, B_i, \ldots, B_{m_k}]$ be the current working set and let us drop the constraint $q \doteq B_i$ to get $\mathcal{B}_{k+1}$. Then, to obtain the factor L_{k+1} of $A_{k+1}^TA_{k+1}$ given the factor L_k of $A_k^TA_k$, we first delete the row q of L_k to get a lower Hessenberg matrix H_k such that $H_kH_k^T = A_{k+1}^TA_{k+1}$:

$$L_k = \begin{bmatrix} L_{11} & & \\ \ell_{\wedge q}^T & \ell_{qq} & \\ L_{21} & \ell_{\vee q} & L_{22} \end{bmatrix} \rightsquigarrow H_k = \begin{bmatrix} L_{11} & & \\ L_{21} & \ell_{\vee q} & L_{22} \end{bmatrix} = \begin{bmatrix} \mathtt{X} & & & & \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & & \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}.$$

To obtain L_{k+1} from H_k we could proceed [17, p. 608] as Gill and Murray [12] did in the dense case, annihilating the diagonal of L_{22} by applying Givens rotations to the column pairs (q, B_{i+1}), (B_{i+1}, B_{i+2}), …, (B_{m_k-1}, B_{m_k}) and then dropping the last column, namely

$$G_k \doteq G(q, B_{i+1})G(B_{i+1}, B_{i+2}) \ldots G(B_{m_k-1}, B_{m_k}),$$

$$H_kG_k = \begin{bmatrix} L_{11} & & \\ L_{21} & \tilde{L}_{22} & \mathrm{O} \end{bmatrix} \quad \rightsquigarrow \quad L_{k+1} = \begin{bmatrix} L_{11} & \\ L_{21} & \tilde{L}_{22} \end{bmatrix}.$$

However, using this procedure $\tilde{L}_{22}$ would be stored in the columns from q to B_{m_k-1}; nevertheless, $\tilde{L}_{22}$ must be stored *in the same place set up (and possibly*

only partially used) for L_{22}, i.e., in the columns from B_{i+1} to B_{m_k}. This can be done by annihilating $\ell_{\vee q}$ with Givens rotations applied to the column pairs (B_{i+1}, q), (B_{i+2}, q), ...,(B_{m_k}, q), namely

$$G_k \doteq G(B_{i+1}, q)G(B_{i+2}, q)\dots G(B_{m_k}, q),$$

$$H_k G_k = \begin{bmatrix} L_{11} & & \\ L_{21} & \mathrm{O} & \tilde{L}_{22} \end{bmatrix} \quad \rightsquigarrow \quad L_{k+1} = \begin{bmatrix} L_{11} & \\ L_{21} & \tilde{L}_{22} \end{bmatrix}.$$

Furthermore, as $\ell_{\vee q}$ is sparse we only have to perform some of the rotations described above.

Summing up, if we allocate a dense intermediate column vector as column $m+1$ in our static structure to act as accumulator, we first initialize it with the column q of H_k, restore the static structure of $L(q,:)$ and $L(:,q)$ (to be used later if needed), and start the annihilation process of the elements of $L(:, m+1)$ from top to bottom, by rotating with the corresponding column of L and taking into account the intermediate fill-in in the accumulator:

ALGORITHM 3.1. *Dropping a constraint*

$L(\mathcal{B}_k, m+1) \leftarrow [O; \ell_{\vee q}]$

while *there are nonzero elements in* $L(:, m+1)$ **do**

$j \leftarrow$ *index of first nonzero element of* $L(:, m+1)$

$G \leftarrow Givens(L(j,j), L(j, m+1))$ $\qquad \{G \in \mathbb{R}^{2\times 2}\}$

$L(j:m, [j\ m+1]) \leftarrow L(j:m, [j\ m+1]) \cdot G$ $\qquad \{Sparse\ product\}$

end while

3.3 Adding a constraint.

Adding a constraint to the working set implies adding a new row a_p^T to the matrix A_k^T. As the computational effort is minimized when the new row is added at the bottom of the matrix, we will perform this addition in two stages: first we will form the working set

$$\tilde{\mathcal{B}}_{k+1} = [B_1, B_2, \dots, B_{i-1}, B_{i+1}, \dots, B_{m_{k+1}}, B_i], \qquad p \doteq B_i$$

and then we will reorder it to obtain

$$\mathcal{B}_{k+1} = [B_1, B_2, \dots, B_{i-1}, B_i, B_{i+1}, \dots, B_{m_{k+1}}],$$

because the order is crucial to maintain the static structure (cf. §3.1).

Let $A_k^T A_k = L_k L_k^T$ and $\tilde{A}_{k+1} = [A_k,\ a_p]$; denoting with $[\ell^T,\ \sigma]$ the new row to add to the bottom of L_k we have that there exists an orthogonal matrix $\tilde{V}_{k+1}$ such that

$$\tilde{A}_{k+1}^T = \begin{bmatrix} A_k^T \\ a_p^T \end{bmatrix} = \begin{bmatrix} L_k & \mathrm{O} & \mathrm{O} \\ \ell^T & \sigma & 0 \end{bmatrix} \tilde{V}_{k+1} \doteq [\tilde{L}_{k+1}\ \mathrm{O}]\tilde{V}_{k+1},$$

so then

$$\tilde{A}_{k+1}^T \tilde{A}_{k+1} = \begin{bmatrix} A_k^T A_k & \mathrm{O} \\ \mathrm{O} & 0 \end{bmatrix} + \begin{bmatrix} \mathrm{O} & A_k^T a_p \\ a_p^T A_k & a_p^T a_p \end{bmatrix}.$$

However, it also holds that

$$\tilde{L}_{k+1} \tilde{L}_{k+1}^T = \begin{bmatrix} L_k L_k^T & \mathrm{O} \\ \mathrm{O} & 0 \end{bmatrix} + \begin{bmatrix} \mathrm{O} & L_k \ell \\ \ell^T L_k^T & \ell^T \ell + \sigma^2 \end{bmatrix},$$

and from a comparison of the two equations above we conclude

$$L_k \ell = A_k^T a_p \qquad \text{and} \qquad \sigma = \sqrt{a_p^T a_p - \ell^T \ell}.$$

Note that this technique, due to Gill and Murray [12], avoid the formation of the column p of the matrix $A^T A$ in [6]. Moreover, it is more accurate to proceed as in [5, §3.3.3] and consider $\ell \doteq L_k^T \delta$, so CSNE can be applied to $\min \|A_k \delta - a_p\|$ and then compute $\sigma = \|A_k \delta - a_p\|$.

Now we partition $\tilde{L}_{k+1}$ in blocks by grouping on one side the rows and columns with index less than p and those with index greater than p on the other side:

$$\tilde{L}_{k+1} = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ \ell_\wedge^T & \ell_\vee^T & \sigma \end{bmatrix} \rightsquigarrow H_{k+1} = \begin{bmatrix} L_{11} & & \\ \ell_\wedge^T & \sigma & \ell_\vee^T \\ L_{21} & \mathrm{O} & L_{22} \end{bmatrix} = \begin{bmatrix} \mathtt{X} & & & & \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{0} & \mathtt{X} & & \\ \mathtt{X} & \mathtt{0} & \mathtt{X} & \mathtt{X} & \\ \mathtt{X} & \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}.$$

Reordering in accordance with $\mathcal{B}_{k+1}$, we have obtained a matrix H_{k+1} that verifies $H_{k+1} H_{k+1}^T = A_{k+1}^T A_{k+1}$ and is lower triangular but has a horizontal spike in row p. So in order to get L_{k+1} from H_{k+1} we only have to apply Givens rotations to the column pairs $(p, B_{m_{k+1}})$, $(p, B_{m_{k+1}-1})$, …, (p, B_{i+1}) to annihilate the elements of $\ell_\vee^T$ from right to left using column p, thus

$$L_{k+1} = H_{k+1} G(p, B_{m_{k+1}}) G(p, B_{m_{k+1}-1}) \ldots G(p, B_{i+1}).$$

Furthermore, as $\ell_\vee^T$ is sparse we only have to perform some of the rotations described above. Note that in the dense case [17, p. 609–610] the usual way to proceed is

$$\tilde{L}_{k+1} = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ \ell_\wedge^T & \ell_\vee^T & \sigma \end{bmatrix} \rightsquigarrow H_{k+1} = \begin{bmatrix} L_{11} & & \\ \ell_\wedge^T & \ell_\vee^T & \sigma \\ L_{21} & L_{22} & \mathrm{O} \end{bmatrix} = \begin{bmatrix} \mathtt{X} & & & & \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & & & \mathtt{0} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & & \mathtt{0} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{0} \end{bmatrix},$$

$$L_{k+1} = H_{k+1} G(B_{m_{k+1}-1}, B_{m_{k+1}}) G(B_{m_{k+1}-2}, B_{m_{k+1}-1}) \ldots G(p, B_{i+1}),$$

but as in §3.2, the $\tilde{L}_{22}$ matrix obtained after the rotations would not occupy the same place as L_{22}.

Summing up, once ℓ and σ has been calculated, firstly we locate $\ell_\wedge^T$ at the beginning of the row p of L_{k+1}. Next, we allocate as column 0 of our static structure a dense intermediate column vector initially zero to act as accumulator for column p of L_{k+1}, and as row 0 a sparse vector whose first element is initialized to σ and whose last part is initialized to $\ell_\vee^T$. Then we start the annihilation process of the elements of $L(0, p+1\colon m)$ from right to left, by rotating column 0 with the corresponding column of L and taking into account the intermediate fill-in in the accumulator; the sparsity of row 0 is exploited to know which columns have to be rotated. Finally we move the contents of column 0 *taking care of the static structure* of $L(:, p)$, since Barlow has proven in [2] that non-structural non-zero elements (i.e., those that do not fit into the predicted structure due to rounding errors) can be omitted without compromising the numerical accuracy.

ALGORITHM 3.2. *Adding a constraint*

$L(p, [0\ \mathcal{B}_k]) \leftarrow [0,\ \ell_\wedge^T,\ O^T]$

$L(0, [0\ \mathcal{B}_k]) \leftarrow [\sigma,\ O^T,\ \ell_\vee^T]$

while *there are nonzero elements in* $L(0, p+1\colon m)$ **do**

$j \leftarrow$ *index of last nonzero element of* $L(0, p+1\colon m)$

$G \leftarrow Givens(L(0,0), L(0,j))$ $\qquad \{G \in \mathbb{R}^{2\times 2}\}$

$L([0\ j\colon m], [0\ j]) \leftarrow L([0\ j\colon m], [0\ j]) \cdot G$ $\qquad \{$*Sparse product*$\}$

end while

$L([0\ p], 0) \leftarrow L([p\ 0], 0)$

$L(:, p) \leftarrow L(:, 0)$ $\qquad \{$*Sparse assignement*$\}$

4 Final remarks.

We have analyzed how to maintain an explicit sparse orthogonal factorization in order to solve the sequence of sparse least squares subproblems needed to implement an active-set method to solve the nonnegative least squares problem for a matrix with more columns than rows. These active-set methods constitute constructive proofs of Farkas' lemma, as well as suitable Phase I's for a non-simplex active-set method for linear programming [30]; in this sense, let us now see that there are more algorithmic alternatives that can take advantage of the sparse technique described in the previous section.

Using the definition of the ℓ_2 norm, it is straightforward to write (1.1) as a semidefinite quadratic program

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} y^T A^T A y - (A^T c)^T y \quad , \ y \in \mathbb{R}^m, \\ \text{subject to} & y \geq \mathrm{O}. \end{array} \tag{4.1}$$

Although there exist several active-set methods to treat directly this problem (e.g., [6]), (4.1) can be thought of as the dual problem of the quadratic program

$$\begin{array}{ll} \text{maximize} & \frac{1}{2}y^T A^T A y - (A^T c)^T y - w^T y \\ \text{subject to} & A^T A y - A^T c - w = \mathrm{O}\ ,\ w \geq \mathrm{O}, \end{array} \quad , \ y, w \in \mathbb{R}^m, \tag{4.2}$$

where we have considered the Wolfe dual of (4.1). If we proceed as in Fletcher [11, p. 249], we can eliminate $w = A^T(Ay - c)$ from the objective function of (4.2) and letting $u \doteq Ay$, we obtain the least distance problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}u^T u \\ \text{subject to} & A^T u \geq A^T c \doteq v, \end{array} \quad , \ u \in \mathbb{R}^n, \tag{4.3}$$

where the nonnegative variables y occurring in (1.1) and in (1.2) are Kuhn-Tucker multipliers for this dual problem of (1.1). Note that w can be eliminated even although $A^T A$ is singular, and that when $c \in \mathcal{R}(A)$ we can assume without loss of generality that $r = \mathrm{rank}(A) = \mathrm{rank}([A, c]) = n$, for if $r < n$ we can append additional linearly independent columns to A with associated zero multiplier to expand this matrix until its rank is n.

Since (4.3) is strictly convex, we can particularize any of the existing primal, dual and primal-dual methods (e.g., those of Goldfarb and Idnani (see [16] and the references therein), or those developed by Santos-Palomo [26] (see also [27])), where we consider (4.3) as the primal problem and (4.1) as the dual problem. *All of them would be constructive proofs of Farkas' lemma, as well as suitable Phase I's for a non-simplex active-set method for linear programming.* As a trivial primal feasible point we can take $u^{(0)} = c$ with the m constraints being active (primal degeneracy), whereas in a dual method we can use $u^{(0)} = \mathrm{O}$, $\mathcal{B}_0 = \varnothing$ and $y^{(0)} = \mathrm{O}$. Dual degeneracy (i.e., in the dual non-negative constraints $y \geq \mathrm{O}$) cannot occur in this strictly convex quadratic program, and primal degeneracy —which does not pose any difficulties in exact arithmetic— is under control in presence of round-off errors. In this case we also have to solve a sequence of least squares problems, so the techniques given in the previous section can be directly applied; we plan as future work to compare all these strategies to analyze whether any of them provides a dual feasible point more suitable than the others to solve $\min\{c^T x : A^T x \geq b\}$.

The vector $\bar{y}_B$ of §2 can be regarded as a first order estimation of the Lagrange multipliers [14, §5.1.5] for $\min\{c^T x : A^T x \geq b\}$. Then, if $A_k^T = [A_1^T,\ A_2^T]$ with A_1 nonsingular and c are partitioned accordingly, we can also use the variable reduction method to develop a reduced version[2] of algorithm 2.1; in this case the estimation is obtained by solving $A_1 \bar{y}_B = c_1$ and computing $w_N = N_k^T d^{(k)}$, where now $d^{(k)} = -Z_k Z_k^T c$ with $Z_k^T = [-A_2^T A_1^{-T},\ I]$. The advantage of this reduced version is that the implementation could be done solving sparse compatible systems (as in [29]) and not least squares problems.

[2]The version given in this paper can be regarded as projected.

Acknowledgements.

The authors thank Michael Saunders at Stanford who, with respect to the technical report prior to Dantzig et al. [8], commented us: "Their method is equivalent to the known (and very simple) NNLS algorithm for non-negative least squares. Their real contribution was to apply it to linear programs to find a feasible solution". We also thank Åke Björck at Linköping and Achiya Dax at Hydrological Service for providing us the references [4, 22] and [9] that we were not aware of. Finally, we gratefully acknowledge the referees for their helpful corrections and remarks, which have improved the quality of the paper.

REFERENCES

1. M. Adlers, *Sparse Least Squares Problems with Box Constraints*, Licentiat thesis, Department of Mathematics, Linköping University, 1998.
2. J. L. Barlow, On the use of structural zeros in orthogonal factorization, *SIAM J. Sci. Statist. Comput.*, 11 (1990), pp. 600–601.
3. E. Barnes, V. Chen, B. Gopalakrishnan and E. L. Johnson, A least-squares primal-dual algorithm for solving linear programming problems, *Oper. Res. Lett.*, 30 (October, 2002), pp. 289–294.
4. Å. Björck, A direct method for sparse least squares problems with lower and upper bounds, *Numer. Math.*, 54 (1988), pp. 19–32.
5. Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM Publications, Philadelphia, USA, 1996.
6. T. F. Coleman and L. A. Hulbert, A direct active set algorithm for large sparse quadratic programs with simple lower bounds, *Math. Progr.*, 45 (1989), pp. 373–406.
7. D. I. Clark and M. R. Osborne, On linear restricted and interval least-squares problems, *IMA J. Numer. Anal.*, 8 (1988), pp. 23–36.
8. G. B. Dantzig, S. A. Leichner, and J. W. Davis, A strictly improving linear programming phase I algorithm, *Ann. Oper. Res.*, 46/47 (1993).
9. A. Dax, Linear programming via least squares, *Linear Algebra Appl.*, 111 (1988), pp. 313–324.
10. A. Dax, An elementary proof of Farkas' lemma, *SIAM Rev.*, 39 (1997), pp. 503–507.
11. R. Fletcher, *Practical Methods of Optimization, 2nd edition*, John Wiley and Sons, New York, NY, USA, 1987.
12. P. E. Gill, and W. Murray, A numerically stable form of the simplex algorithm, *Linear Algebra Appl.*, 7 (1973), pp. 99–138.
13. P. E. Gill, and W. Murray, Numerically stable methods for quadratic programming, *Math. Prog.*, 14 (1978), pp. 349–372.
14. P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London and New York, 1981.
15. P. E. Gill, and S. J. Hammarling, and W. Murray, and M. A. Saunders, and M. H. Wright, User's guide for LSSOL (version 1.0): A FORTRAN package for constrained linear least-squares and convex quadratic programming, Technical report, Department of Operations Research, Stanford University, USA, January 1986.
16. D. Goldfarb, Efficient primal algorithms for strictly convex quadratic programs, in *Lecture Notes in Mathematics*, 1230 (1986), pp. 11–25, Springer-Verlag, Berlin.

17. G. H. Golub, and C. F. Van Loan, *Matrix Computations, 3rd edition*, The Johns Hopkins University Press, Baltimore, MD, USA, 1996.
18. P. Guerrero-García, *Range-Space Methods for Sparse Linear Programs* (Spanish), Ph.D. thesis, Department of Applied Mathematics, University of Málaga, Spain, July 2002.
19. P. Guerrero-García and Á. Santos-Palomo, Gyula Farkas would also feel proud, Technical Report, Department of Applied Mathematics, University of Málaga, Spain, October 2002. Submitted for publication to OMS.
20. P. Guerrero-García and A. Santos-Palomo, A comparison of three sparse linear program solvers, Technical Report, Department of Applied Mathematics, University of Málaga, October 2003. Submitted for publication to BIT.
21. C. L. Lawson, and R. J. Hanson, *Solving Least Squares Problems*, revised republication in 1995 by SIAM of the original work published by Prentice-Hall, Englewood Cliffs, NJ, USA, 1974.
22. P. Lötstedt, Solving the minimal least squares problem subject to bounds on the variables, *BIT Numer. Math.*, 24 (1984), pp. 206–224.
23. U. Oreborn, *A Direct Method for Sparse Nonnegative Least Squares Problems*, Licentiat thesis, Department of Mathematics, Linköping University, Sweden, 1986.
24. M. R. Osborne, Degeneracy: Resolve or avoid?, *J. Opl. Res. Soc.*, 43:8 (1992), pp. 829–835.
25. L. F. Portugal, J. J. Júdice, and L. N. Vicente, A comparison of block pivoting and interior-point algorithms for linear least squares problems with nonnegative variables, *Math. Comp.*, 63 (1994), pp. 625–643.
26. A. Santos-Palomo, *New Quadratic Programming Methods* (Spanish), Ph.D. thesis, Department of Applied Econometrics, University of Málaga, Spain, 1995.
27. A. Santos-Palomo, New range-space active-set methods for strictly convex quadratic programming, in the proceedings of the III Conference on Operations Research, Universidad de La Habana, Cuba, P. Olivares-Rieumont (ed.), p. 27, March 1997
28. Á. Santos-Palomo, The sagitta method for solving linear programs, Technical Report, Department of Applied Mathematics, University of Málaga, April 1998. Accepted for publication to EJOR.
29. A. Santos-Palomo, and P. Guerrero-García, Solving a sequence of sparse compatible systems, in the proceedings of the 19th Biennial Conference on Numerical Analysis, Dundee, Scotland, D. Griffiths and G. A. Watson (eds.), Tech. report NA/201, pp. 23–24, June 2001. Submitted for publication to IMAJNA.
30. A. Santos-Palomo, and P. Guerrero-García, A non-simplex active-set method for linear programs in standard form, in the proceedings of the XXVI Congreso Nacional de Estadística e Investigación Operativa, Úbeda, Spain, J. C. Ruiz-Molina (ed.), p. 246(5), November 2001. Submitted for publication to CMA.
31. J. Stoer, On the numerical solution of constrained least squares problems, *SIAM J. Numer. Anal.*, 8 (1971), pp. 382–411.
32. J. Stoer, A dual algorithm for solving degenerate linearly constrained linear least squares problems, *J. Num. Alg. App.*, 1 (1992), pp. 103–131.