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

Updating and downdating an upper trapezoidal sparse orthogonal factorization

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

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

Abstract

We describe how to update and downdate an upper trapezoidal sparse orthogonal factorization, namely the sparse QR factorization of A_k^T, where A_k is a "tall and thin" full column rank matrix formed with a subset of the columns of a fixed matrix A. In order to do that, we have adapted to rectangular matrices (with fewer columns than rows) Saunders' techniques of early 70s for square matrices, by using the static data structure of George and Heath of early 80s but allowing row downdating on it. An implicitly determined column permutation allow us to dispense with computing a new ordering after each update/downdate; it fits well into the Linpack downdating algorithm and ensures that the updated trapezoidal factor will remain sparse. We give all the necessary formulae even if the orthogonal factor is not available, and we comment on our implementation using the sparse toolbox of Matlab 5.

1. Aims, difficulties and related work

In certain non-simplex active-set methods (also currently known as basis-deficiency-allowing simplex variations) for linear programming we need to solve a *sequence* of sparse compatible systems of the form:

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

where b and c are iteration-dependent vectors, and x, y, and d are unknown vectors. Furthermore, 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

*September 30, 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.

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 $\operatorname{rank}(A_{k+1}) = \operatorname{rank}(A_k) \pm 1$; when exchanging, deletion is done before appending and $\operatorname{rank}(A_{k+1}) = \operatorname{rank}(A_k)$.

In dense implementations, the QR factorization of A_k (or equivalently, the LQ factorization of A_k^T) is usually maintained [12, pp. 172,234] due to the fact that it can be computed by a numerically stable method, for $0 < A_k^T A_k \in \mathbb{R}^{m_k\times m_k}$. ($M > 0$ ($M \geq 0$) stands for "M is symmetric positive (semi)definite".) Updating is done column-wise as indicated e.g. by Golub & Van Loan [19, §12.5.2]. But, as Gilbert & Peierls pointed out in [17, p. 862],

> *Translating a matrix algorithm from a dense setting to a sparse setting may involve more than just generalizing indices and using lists instead of dense arrays.*

The sparse implementation of a non-simplex active-set method has its own difficulties, whose solution is not a trivial matter as will be shown in this and a forthcoming article [21].

It is well-known that nowadays efficient "black-boxes" exist to compute the sparse QR factorization using multifrontal techniques, but such a factorization can only be used when a *single* least-squares problem is to be solved because this factorization is not (column-wise) updatable, as Adlers [1, p. 43] recognized in 1998:

> *Multi-frontal techniques are more efficient for factorization of sparse matrices. However, there are no efficient ways yet, to update the QR factorization using multifrontal techniques.*

With respect to this subject, Mitra & Tamiz [30, p. 280] claimed

> *Considerable progress continues to be made in sparse equation solving methods which may be looked upon (as) "blackbox" procedures. Not all of these provide iterative update procedures. Hence methods for sparse and stable updates to work with sparse and stable solvers can be of considerable value.*

George, Heath & Ng [14] adapted the sparse QR factorization of A_k to solve *a single* underdetermined system (hence they did not downdate); furthermore, they admit [14, pp. 995–996] that this factorization requires more work than the QR factorization of A_k^T when $m_k \ll n$. Note that we do not have to assume a restrictive $n - m_k$ being relatively small when we the minimum norm solution (see [26, p. 229]) of the underdetermined system is not required.

The main point is that working with $A_k A_k^T$ in a sparse setting is more suitable than with $A_k^T A_k$; indeed, Hager, Shih & Lundin [24, p. 23] propose to work with the factorization of $A_k A_k^T \in \mathbb{R}^{n\times n}$, although it is positive semidefinite, rather than with $A_k^T A_k \in \mathbb{R}^{m_k\times m_k}$, which is positive definite:

> *The matrix $A_F^T A_F$ is difficult to work with since its dimension grows*

> *and contracts during the iterations, and the sparsity structure is difficult to predict. The matrix $\sigma I_n + A_F A_F^T$ is quite amenable to numerical work since it has fixed dimensions and its sparsity is predictable.*

In their work, the matrix A_F has the same meaning as our A_k, *but $m_k \geq n$ is allowed*. Thus, when A_F is full row rank, $A_F A_F^T > 0$ and the QR factorization of A_F^T yields a regular triangular factor (hence $\sigma = 0$); when A_F is not full row rank, $A_F A_F^T \geq 0$ and they regularize to compute the Cholesky factor of $\sigma I_n + A_F A_F^T$. The dynamic work with the Cholesky factor of $A_F A_F^T$ has been recently published by Davis & Hager [10], where they explicitly impose $m_k \geq n$.

The result given above has been exploited in interior point methods for linear programming (see, e.g., [39]) nearly since their inception; in these methods, the full row rank matrix A plays the role of the matrix A_F above. Interior point methods based on the normal equations approach work with a sequence of weighted systems whose matrix is AW_kA^T, where $W_k > 0$ is an iteration-dependent diagonal matrix. As the number of iterations in an interior point method is small, the Cholesky factor of $AW_{k+1}A^T$ is usually recomputed from scratch rather than updated from that of AW_kA^T, although some recent sparse updating proposals have been done by Baryamureeba & Steihaug [3] in a robust linear regression context, also with $W_k > 0$.

We could modify the interior point updating techniques cited above if we were able to deal with a binary diagonal matrix $W_k \geq 0$ with $\mathrm{sum}(\mathrm{diag}(W_k)) \leq n$. In fact, Edlund [11] has been working with $\mathrm{sum}(\mathrm{diag}(W_k)) \geq n$ and he can (row-wise) update and downdate the QR factorization of A_k^T in a dynamic multifrontal sparse setting, using hyperbolic rotations to perform the downdating. When A_k is not full row rank or when $m_k < n$, Edlund also regularize to yield the Cholesky factor of $\sigma I_n + A_k A_k^T$. Furthermore, note that Adlers' claim of a (column-wise) updatable sparse multifrontal QR factorization of A_k remains at present unsolved in spite of Edlund's work.

In this paper we describe how to update and downdate the sparse orthogonal QR factorization of the full row rank matrix A_k^T when $m_k \leq n$ without using regularization. As we work on top of a static structure, we do not have to deal with intricate data structures as Edlund does. Row updating and downdating of A_k^T is done as in the dense case [19, §12.5.3], but we have had to consider an *implicitly defined column permutation* and several additional sparse issues. This factorization is naturally updatable by adding rows [13, 27], the row order in A_k^T does not affect the density of the triangular factor, small non-zeros which actually would be *structural zeros* [2] can be avoided, and the sequence of downdates is expected to have a good numerical behaviour because we dispense with hyperbolic rotations, as pointed out by Stewart [36].

Accurate algorithms to update and downdate in the $m_k \geq n$ full rank case was given by Björck, Eldén & Park [6], where they explicitly assume that no change in range occurs after the update or the downdate. They did not consider a static structure in spite of dispensing with the orthonormal factor [37, p. 56] by

combining corrected seminormal equations as a previous stage for the LINPACK downdate. Also in the $m_k \geq n$ full rank case, Chan & Brandwajn [7] have been working with the static sparsity structure in the field of power system analysis.

To the best of our knowledge, the only (dense) downdating experiments with a QR factorization of A_k^T in the $m_k \leq n$ case was done by Powell [32], but the dense nature of the matrix leads to do not have to deal with the column permutation, or else to permute only for numerical purposes rather than for sparsity purposes. Saunders [34] already used this sparse idea of factorizing A_k^T orthogonally for the case $m_k = n$; his (still unpublished) work was pushed aside because the fill-in tends to be worse than those of LU-based methods [8, p. 43], but note that this assertion was done *in terms of dynamic factors*. The static alternative proposed by Coleman [8, p. 41], which consists on symmetrically permuting the sparsity structure of $A_k A_k^T$ for its Cholesky factor to be as sparse as possible, would not be efficient since it would need a new ordering *in each iteration*. On the other hand, Coleman [8, pp. 43,45] proposed the following research problem:

> *The Cholesky-based method of Saunders attempts to maintain a sparse Cholesky factor. This general approach does have some appeal from a stability point of view and probably warrants further examination. Suggest and investigate ways in which a structured Cholesky factor can be maintained.*

What we do here is to adapt Saunders' methodology to the case $m_k < n$ (in fact, his proposal to do an exchange was to *delete before adding*, and he proclaimed the numerical stability of this exchange *in spite of the rank reduction after the deletion*). We use the data structure of George & Heath [13, 26], but we allow row downdating on such static structure, even if the orthogonal factor is not available. As we shall see in the following sections, the key point is the management of an *implicitly defined column permutation* in order to dispense with determining a new ordering in each step; we shall also show that it fits well into the LINPACK downdating algorithm and that ensures that the updated trapezoidal factor will remain sparse. We also provide mathematically rigorous descriptions of both the updating and downdating processes, which are usually avoided when dealing with static structures in sparse settings, as well as illustrative examples.

2. Initial factorization and sparse issues

Let us consider the sparse QR factorization of a "short and fat" matrix A_k^T, e.g., with $m_k = 3$ rows and $n = 7$ columns. When we compute this factorization with the MATLAB 5 `qr` "black-box" we would obtain, say, the Wilkinson's diagram

$$\underbrace{\mathtt{qr}\left(\begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}\right)}_{A_k^T} = \underbrace{\begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}}_{Q_k} \cdot \underbrace{\begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} \end{bmatrix}}_{R_k \cdot \Pi_k}.$$

Note that $R_k\Pi_k$ is a staircase-shaped, permuted upper trapezoidal matrix, where *permutation* Π_k *is implicitly defined by the staircase shape of the structure*; in this case

$$\Pi_k^T = [e_1,\ e_3,\ e_6,\ e_2,\ e_4,\ e_5,\ e_7], \qquad \text{or} \quad \Pi_k = [e_1,\ e_4,\ e_2,\ e_5,\ e_6,\ e_3,\ e_7],$$

where $e_i \in \mathbb{R}^7$ is the ith column of the identity matrix I_7. Even it could be the case that

$$\texttt{qr}\underbrace{\left(\begin{bmatrix} \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} \\ \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} \\ \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} \end{bmatrix}\right)}_{A_k^T} = \underbrace{\begin{bmatrix} \texttt{X} & \texttt{X} & \texttt{X} \\ \texttt{X} & \texttt{X} & \texttt{X} \\ \texttt{X} & \texttt{X} & \texttt{X} \end{bmatrix}}_{Q_k} \cdot \underbrace{\begin{bmatrix} \texttt{0} & \texttt{0} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} \\ \texttt{0} & \texttt{0} & \texttt{0} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} \\ \texttt{0} & \texttt{0} & \texttt{0} & \texttt{0} & \texttt{0} & \texttt{0} & \texttt{X} \end{bmatrix}}_{R_k\cdot\Pi_k},$$

$$\Pi_k^T = [e_3,\ e_4,\ e_7,\ e_1,\ e_2,\ e_5,\ e_6], \qquad \text{or} \quad \Pi_k = [e_4,\ e_5,\ e_1,\ e_2,\ e_6,\ e_7,\ e_3].$$

In a more general framework, let $A_k \in \mathbb{R}^{n\times m_k}$ be sparse and $\operatorname{rank}(A_k) = m_k \le n$. We know that there exists an *implicitly defined* permutation Π_k^T of the columns of A_k^T such that

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

with R_k upper trapezoidal, R_{i} upper triangular and nonsingular, and Q_k orthogonal. (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.) Nevertheless, the permutation Π_k^T is not explicitly performed, thus $R_k\Pi_k$ is what we are going to maintain. Note also that a reordering of the columns of A_k has no impact in R_k, for

$$S_k^TA_k^T\Pi_k^T = (S_k^TQ_k)R_k,$$

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

This factorization is computable with the black-box `qr` in MATLAB 5 (but not every A_k^T can be factorized in this way with the sparse `sqr` packages of Matstoms [29] and Adlers [1], because they are intended for matrices with more rows than columns). However, this factorization is not updatable with `qrupdate` in MATLAB 5 because that only works with dense matrices. Also, there are no efficient ways yet to update this QR factorization using multifrontal techniques.

Enlarging R_k below with $n - m_k$ zero rows we obtain the sparse Cholesky factor of $\Pi_kA_kA_k^T\Pi_k^T \ge 0$ (see Higham [28, Thm. 10.9, p. 210]). Computing this factor from the sparse QR given above makes unnecessary both the formation of the product $A_kA_k^T$ and the complete pivoting that we would have to perform whether a dense semidefinite Cholesky method on $A_kA_k^T$ were used; in other words, our Π_k is only defined for sparsity purposes rather than for numerical purposes. This is not expected to cause numerical difficulties because it amounts to a row-skipping strategy in a modified Cholesky factorization (see [39, §11.5]), which is a successful device to mimic diagonal pivoting in $A_kA_k^T$ (i.e., column

pivoting in the QR factorization of A_k^T) that is used in interior point methods in order to avoid the modification of the sparsity structure that such pivoting would imply.

It is well known that when A_k is a subset of the columns of a fixed matrix $A \in \mathbb{R}^{n\times m}$ with $m \geq n$ and $\operatorname{rank}(A) = n$, a static structure (i.e., *a priori* set up) can be used to try to minimize fill in the triangular factor. The sparse structure of $A_k A_k^T$ is a subset of that of AA^T [13], hence an *a priori* permutation P of the rows of A can be chosen (e.g., with `colperm(A')` in MATLAB 5) to sparsify the Cholesky factor R of PAA^TP^T. This permutation P is set up at the beginning and *it is explicitly applied to the original A matrix*, and it is not related with the permutation Π_k that is implicitly defined *every time* by the staircase shape of the trapezoidal factor. In other words, we do not have to analyze which Π_k is most suitable in a given iteration for the upper trapezoidal factor R_k of $A_k^T\Pi_k^T$ to be as sparse as possible *because it is determined by the static structure itself*. On the other hand, the principal aim in papers like [13, 27] was to solve a single problem $\min \|A^Tx-b\|_2$, so none of them considered the case of deleting a row of this least squares problem. However, it is reasonable to think that when a row is deleted there must be space in the static structure, due to the formal equivalence (already established by Golub [18] in 1969) between deleting a real row a^T and adding a complex row $\sqrt{-1}a^T$ (both rows have the same structure and hyperbolic rotations modify the structure in the same way as Givens rotations).

The column order of A does not affect the density of the Cholesky factor R of PAA^TP^T, but does affect the amount of intermediate work to compute it. If the columns of A do not vary widely in norm, this column order does not affect the numerical stability and can be chosen based solely in sparse issues [5, p. 244–245]. We append rows to the bottom of A_k^T in order to take advantage of the work already done, but when we have to refactorize A_k^T (or to calculate an initial factorization of A_0^T), an *a priori* ordering of the columns of A can be beneficial, using the corresponding subordering for A_k. A refactorization is triggered with respect to a fixed tolerance [31, p. 34], and can be performed with the QR black-box mentioned above.

It is worth pointing out that an advantage of this static sparse approach is the great simplicity of the data structures involved (unlike those occurring in sparse dynamic techniques). In linear programming we usually reorder the constraint matrix A of a linear program in standard form to be in *lower block triangular form* (LBTF) at the beginning; in fact, it is considered the canonical form of a sparse matrix [23, p. 280]. We can use the Dulmage-Mendelsohn decomposition (`dmperm(A')` in MATLAB 5) to compute the *upper block triangular form* of A^T. In this way we obtain right from the beginning both a suitable row ordering (to be used in refactorizations) and a suitable column ordering P^T, and hence we have an *upper bound for the static structure* needed for $\Pi_k^T R_k^T$, correctly predicted for R^T in the sense of Coleman, Edenbrandt & Gilbert [9] if A^T has the strong Hall property. Note that we do not pursuit that the A_k matrix to be factorized

is in block triangular form for all iteration k, and that we could also have used cheaper minimum degree routines like `colmmd` to obtain P^T.

Both A and R^T are stored by a simple compressed column scheme, because we are going to access them by columns and thus we can take advantage of the locality of reference (see Stewart [37, p. 110]) in MATLAB 5 sparse packed storage scheme [16]. As pointed out by Davis & Hager [10, §7.1], in MATLAB every change in the structure of a sparse matrix would entail a new copy of the entire matrix. Hence we take advantage of the fact that when MATLAB annihilates an element of a sparse matrix, the operating system does not deallocate the memory locations previously allocated for that element of the sparse matrix. Thus, rather than overlaying a data structure on the triangular factor (as did Vempati, Slutsker & Tinney [38]), we mark with an special value (e.g., `NaN` in MATLAB) the elements not used in the structure previously set up, and we convert them into zeros just before selecting the submatrix to operate with. This allow us to take advantage of the sparse numerical linear algebra primitives of the sparse toolbox, but we have to recover the structure after the algebraic operation involved, reinserting `NaN`s in those elements where fill-in did not occur yet; this overhead would be avoided with low-level programming. It is reasonable to think that orderings that correctly predict the sparsity structure of R (like that obtained with `dmperm`) outperform those that only yield an upper bound (like those obtained with `colmmd` or `colperm`) when trying to minimize the overhead mentioned above.

The astute reader will have realized how we take advantage of the following fact. Suppose that we start the computation of the QR factorization of A^T (e.g., to solve $\min \|A^T x - b\|_2$) by a row-sequential algorithm (that of George & Heath), but we stop the computation after $m_k \leq n$ rows of A^T have been rotated into the structure for R. *This intermediate result obtained in the kth stage of this QR factorization of A^T is just the QR factorization of $A_k^T \Pi_k^T$*, and the indices of the rotated rows can be just the indices of the columns of A being in the working set (hence $m_k \leq n$) of a non-simplex active-set method in that iteration k. To the best of our knowledge, Hanson & Wisniewski [25] were the first authors to point out this analogy between minimizing $\|A^T x - b\|_2$ by processing $A^T \in \mathbb{R}^{m \times n}$ row by row, and solving the linear program in standard form that consists of maximizing $b^T y$ subject to $Ay = c, y \geq \mathrm{O}$, by processing $A \in \mathbb{R}^{n \times m}$ column by column in the fixed order dictated by an active-set method, thus fully exploiting that A_k is a subset of the columns of a given fixed matrix A when their dense work is translated to a sparse setting.

3. Updating the factorization

When we add the pth constraint to the active set we have to add a new column to the right of A_k, i.e., we have to add a new row a_p^T to the bottom of A_k^T:

$$A_{k+1}^T = \begin{bmatrix} A_k^T \\ a_p^T \end{bmatrix}.$$

Then, given the QR factorization of $A_k^T\Pi_k^T$, we want to determine the new QR factorization after the addition of a_p^T; this problem is known as *updating* the QR factorization.

Since $A_k^T = Q_k R_k \Pi_k$, we can write

$$A_{k+1}^T = \begin{bmatrix} A_k^T \\ a_p^T \end{bmatrix} = \begin{bmatrix} Q_k & \mathrm{O} \\ \mathrm{O}^T & 1 \end{bmatrix} \begin{bmatrix} R_k\Pi_k \\ a_p^T \end{bmatrix},$$

so we have

$$\begin{bmatrix} Q_k^T & \mathrm{O} \\ \mathrm{O}^T & 1 \end{bmatrix} \begin{bmatrix} A_k^T \\ a_p^T \end{bmatrix} = \begin{bmatrix} R_k\Pi_k \\ a_p^T \end{bmatrix} = \begin{bmatrix} \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \end{bmatrix}. \tag{1}$$

Let f_j be the *index* of the column of Π_k^T where e_j appears; e.g., regarding the first Wilkinson diagram of §2,

$$f_1 = 1, f_3 = 2, f_6 = 3, \text{ and } f_2 = 4 > 3 \doteq m_k;$$

or even, with respect to the second Wilkinson diagram of §2,

$$f_3 = 1, f_4 = 2, f_7 = 3, \text{ and } f_1 = 4 > 3 \doteq m_k;$$

For each $j \in 1\colon n$, we proceed to annihilate the jth element (if nonzero) of the updated version of a_p^T; such annihilation is done by a Givens rotation $G(f_j, m_k+1)^T \in \mathbb{R}^{(m_k+1)\times(m_k+1)}$ [19, §5.1.8] only if $f_j \le m_k$, since otherwise we are done. Hence, after m_k rotations (in the worse case) we have

$$G^T \doteq G(f_n, m_k+1)^T \cdots G(f_2, m_k+1)^T G(f_1, m_k+1)^T,$$

$$F^T G^T \begin{bmatrix} R_k\Pi_k \\ a_p^T \end{bmatrix} \doteq R_{k+1}\Pi_{k+1}, \tag{2}$$

where Π_{k+1} is a permutation matrix such that R_{k+1} is trapezoidal with nonzero diagonal elements, and F^T is a suitable row permutation for $R_{k+1}\Pi_{k+1}$ to be staircase shaped. Then, from (1) and (2),

$$A_{k+1}^T\Pi_{k+1}^T = \begin{bmatrix} Q_k & \mathrm{O} \\ \mathrm{O}^T & 1 \end{bmatrix} GFF^TG^T \begin{bmatrix} R_k\Pi_k \\ a_p^T \end{bmatrix} \Pi_{k+1}^T \doteq Q_{k+1}R_{k+1}.$$

It is well known that in practical algorithms we can dispense with the orthogonal factor, since R_k can be updated even if Q_k is not available. Furthermore, as we shall illustrate in the following example, the static sparsity structure itself implicitly defines both the updated column permutation Π_{k+1} and the suitable row permutation F^T; in fact, the actual Givens rotation used to annihilate the jth element is $G(j, n+1) \in \mathbb{R}^{(n+1)\times(n+1)}$. This ensures that the updated trapezoidal factor will remain sparse.

3.1. Example of updating

Let us consider the matrix whose structure is given in [15, p. 104]. In MATLAB, `symbfact(A','col')` accomplishes the symbolic analysis of AA^T directly on A^T and consequently returns an upper bound structure for R^T:

$$A^T = \begin{bmatrix} \times & & & & & \times \\ & \times & & \times & & \\ 1 & & -1 & & & \\ & \times & & \times & & \times \\ & & & & 1 & 1 \\ 1 & & & & & 2 \\ & & -1 & -1 & & \\ & \times & & \times & & \end{bmatrix}, \qquad R^T = \begin{bmatrix} \times & & & & & \\ & \times & & & & \\ \times & & \times & & & \\ & \times & \times & \times & & \\ & & & & \times & \\ \times & \times & \times & \times & \times & \times \end{bmatrix}.$$

Note that we have considered R^T instead of R because of locality of reference issues (MATLAB stores the sparse matrices by columns). We have chosen to proceed row-wise in the mathematical description of the updating due to familiarity reasons; however, for the example to be as real as possible, we shall work column-wise, postmultiplying by Givens matrices rather than premultiplying by their transposes.

We have rotated rows a_6^T and a_7^T into R and now we are going to rotate a_3^T (i.e., $k = 2$, $A_k = [a_6\ a_7]$ and $A_{k+1} = [a_6\ a_7\ a_3]$, where a_i is column i of A):

$$[R^T|a_3] = \left[\begin{array}{cccccc|c} 1 & & & & & & 1 \\ & \times & & & & & \\ \times & & -1 & & & & -1 \\ & \times & -1 & \times & & & \\ & & & & \times & & \\ 2 & \times & \times & \times & \times & \times & \end{array}\right] \text{ and } [\Pi_2^T R_2^T|a_3] = \left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 0 & 0 \\ \mathbf{0} & -1 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \\ 2 & \mathbf{0} & 0 \end{array}\right],$$

where the working vector is located to the right of the vertical bar. Postmultiplying by $\begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$ and by $\begin{bmatrix} \sqrt{6}/3 & -1/\sqrt{3} \\ 1/\sqrt{3} & \sqrt{6}/3 \end{bmatrix}$, first columns 1 and 3 and then columns 2 and 3, the elements of the working vector are annihilated downhill using Givens rotations:

$$\left[\begin{array}{cc|c} \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -1 & -1/\sqrt{2} \\ 0 & -1 & 0 \\ 0 & 0 & 0 \\ \sqrt{2} & \mathbf{0} & -\sqrt{2} \end{array}\right] \text{ and } \Pi_3^T R_3^T = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ 0 & -\sqrt{6}/3 & 1/\sqrt{3} \\ 0 & 0 & 0 \\ \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} \end{bmatrix}.$$

Note that fill is restricted to the working vector and the column with respect to which we are rotating. The column just created has appeared in the last position

(hence $F = I$), but it does not happen so in general. Although we have shown here only the columns of R^T that occur in $\Pi_k^T R_k^T$, we actually work within the whole structure to avoid the explicit computation of the permutations Π_k and F, so the mathematical application of F amounts to a simple copy of the working vector to the right position within the sparsity structure. Finally, the addition of the 5th constraint entails no rotations:

$$\Pi_4^T R_4^T = \begin{bmatrix} \sqrt{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1/\sqrt{2} & -\sqrt{6}/2 & 0 & 0 \\ 0 & -\sqrt{6}/3 & 1/\sqrt{3} & 0 \\ 0 & 0 & 0 & 1 \\ \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} & 1 \end{bmatrix},$$

thus obtaining the permuted trapezoidal matrix for $A_4 = [a_6\ a_7\ a_3\ a_5]$ or any column permutation $A_4 S_4$.

4. Downdating the factorization

When we delete the qth constraint from the active set we can assume that it is the first row of A_k^T:

$$A_k^T = \begin{bmatrix} a_q^T \\ A_{k+1}^T \end{bmatrix},$$

because if it were not the first row we only have to exchange beforehand the corresponding row and the first row in A_k^T. Then, given the QR factorization of $A_k^T \Pi_k^T$, we want to determine the new QR factorization after a_q^T is deleted; this problem is known as *downdating* the QR factorization.

It can be shown (see [20, §3.2.4]) that the presence of the permutation matrix does not modify in essence the classical downdating algorithm, although it turns out not to be suitable to be combined with the static structure *unless a separate working vector is introduced*. As was pointed out by one of the referees, the computations of the classical downdating method could be reorganized to *rotate into the bottom row* (more precisely, into the row corresponding to the bottom-most nonzero element of the vector $q \in \mathbb{R}^{m_k}$ defined below), with that row being the working vector. What we are (equivalently) going to do here is to reorganize the computation to *rotate into a new top row*, with that row being the working vector, to illustrate the suitability of the proposal to be used in a LINPACK setting.

In a sparse setting it is usual to dispense with Q_k. Let us show how the implicitly determined column permutation fits well into the LINPACK algorithm [34] to downdate the Cholesky factor, because *the system* $\Pi_k^T R_k^T q = a_q$ *is also compatible with only one solution*, and solving it we obtain q; furthermore, it does not destroy the predicted static structure. First we enlarge the matrix $R_k \Pi_k$ with

a suitable number δ_n to be determined below:

$$\bar{R}_k \doteq \begin{bmatrix} \delta_n & \mathrm{O}^T \\ q & R_k\Pi_k \end{bmatrix} = \begin{bmatrix} \mathtt{X} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} \\ \mathtt{X} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{X} \end{bmatrix}.$$

Now if we apply to $\bar{R}_k$ several Givens rotations $G(1,j)^T \in \mathbb{R}^{(m_k+1)\times(m_k+1)}$ with $j \in m_k+1\colon -1\colon 2$ to annihilate the nonzeros of q, we obtain

$$F^T G^T \bar{R}_k \doteq F^T G(1,2)^T \cdots G(1,m_k)^T G(1,m_k+1)^T \bar{R}_k \doteq \begin{bmatrix} \bar{R}_{k+1} \\ \mathrm{O}^T \end{bmatrix},$$

where $\bar{R}_{k+1}$ is the enlarged matrix corresponding to the new factor with R_{k+1} trapezoidal with nonzero diagonal elements:

$$\bar{R}_{k+1} \doteq \begin{bmatrix} \alpha & \alpha v^T \Pi_{k+1} \\ \mathrm{O} & R_{k+1}\Pi_{k+1} \end{bmatrix} \qquad (\alpha = \pm 1),$$

and F^T is the permutation needed to move the zero row just created to the m_k+1 position. Being G and F orthogonal matrices implies that $\bar{R}_{k+1}^T \bar{R}_{k+1} = \bar{R}_k^T \bar{R}_k$:

$$\begin{bmatrix} \alpha & \mathrm{O}^T \\ \alpha\Pi_{k+1}^T v & \Pi_{k+1}^T R_{k+1}^T \end{bmatrix} \begin{bmatrix} \alpha & \alpha v^T \Pi_{k+1} \\ \mathrm{O} & R_{k+1}\Pi_{k+1} \end{bmatrix} = \begin{bmatrix} \delta_n & q^T \\ \mathrm{O} & \Pi_k^T R_k^T \end{bmatrix} \begin{bmatrix} \delta_n & \mathrm{O}^T \\ q & R_k\Pi_k \end{bmatrix},$$

multiplying out and comparing blocks we have that $\Pi_{k+1}^T v = \Pi_k^T R_k^T q = a_q$ and that $\delta_n^2 = 1 - \|q\|_2^2 = 0$, because

$$\begin{bmatrix} 1 & v^T \Pi_{k+1} \\ \Pi_{k+1}^T v & \Pi_{k+1}^T (R_{k+1}^T R_{k+1} + vv^T)\Pi_{k+1} \end{bmatrix} = \begin{bmatrix} \delta_n^2 + \|q\|_2^2 & q^T R_k \Pi_k \\ \Pi_k^T R_k^T q & \Pi_k^T R_k^T R_k \Pi_k \end{bmatrix}.$$

Thus $\Pi_{k+1}^T R_{k+1}^T R_{k+1} \Pi_{k+1} = \Pi_k^T R_k^T R_k \Pi_k - a_q a_q^T$; that is what we wanted, for

$$A_{k+1}A_{k+1}^T = \Pi_{k+1}^T R_{k+1}^T R_{k+1} \Pi_{k+1} = \Pi_k^T R_k^T R_k \Pi_k - a_q a_q^T = A_k A_k^T - a_q a_q^T.$$

As in the algorithm given above, the Givens matrices are $(m_k+1)\times(m_k+1)$. Using the additional working row we can get useful information to trigger the refactorization when $\Pi_{k+1}^T v$ and a_q differ substantially; furthermore, the data structure always has enough space allocated to work, and the fill-in is confined to the working row, as in the updating algorithm but not in the classical downdating algorithm [20, §3.2.4].

There is a subtle difference with the $m_k > n$ LINPACK case, in which q is the first row of an *orthonormal* matrix and hence $\|q\|_2 \neq 1$ in general. On the other hand, we can ensure $\|q\|_2 = 1$ in exact arithmetic when $m_k \leq n$, since q is the first row of an *orthogonal* matrix. This case was excluded from LINPACK because it leads to a reduction in rank after the downdate but, as was pointed out in 1972 by Saunders [34], numerical difficulties are not expected.

4.1. Example of downdating

Let us go on with the example of §3.1; now we want to delete a_7^T. First we solve the compatible system $\Pi_4^T R_4^T q = a_7$ to obtain $q = [0; \sqrt{6}/3; -1/\sqrt{3}; 0]$; as was pointed out by Heath [26, p. 229] (with his R_1 being the same as our $R_{\bar{1}}$),

> *It is unnecessary to extract R_1 from the data structure or write a special back substitution routine to skip over the zero rows, since the same effect may be obtained by simply setting the "zero" diagonal elements equal to 1 and noting that the corresponding components of the right-hand sides will already be zero.*

and hence solving this system does not need any special back substitution routine, since we can proceed by overlaying $\Pi_4^T R_4^T$ on I_6 to obtain $[0; 0; \sqrt{6}/3; -1/\sqrt{3}; 0; 0]$ and then select the first, third, fourth and fifth components. Now we enlarge $\Pi_4^T R_4^T$ and rotate with respect to the first column to annihilate the nonzero elements of q^T westbound:

$$\bar{R}_4^T \doteq \begin{bmatrix} \delta_n & q^T \\ \mathrm{O} & \Pi_4^T R_4^T \end{bmatrix} = \left[\begin{array}{c|cccc} 0 & 0 & \sqrt{6}/3 & -1/\sqrt{3} & 0 \\ \hline 0 & \sqrt{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1/\sqrt{2} & -\sqrt{6}/2 & 0 & 0 \\ 0 & 0 & -\sqrt{6}/3 & 1/\sqrt{3} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} & 1 \end{array}\right],$$

where the working vector is now located to the left of the vertical bar; postmultiplying by $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ and by $\begin{bmatrix} -1/\sqrt{3} & \sqrt{6}/3 \\ -\sqrt{6}/3 & -1/\sqrt{3} \end{bmatrix}$, first columns 1 and 4 and then columns 1 and 3, we obtain

$$\left[\begin{array}{c|cccc} 1/\sqrt{3} & 0 & \sqrt{6}/3 & 0 & 0 \\ \hline 0 & \sqrt{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1/\sqrt{2} & -\sqrt{6}/2 & 0 & 0 \\ -1/\sqrt{3} & 0 & -\sqrt{6}/3 & \mathbf{0} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2/\sqrt{3} & \sqrt{2} & -\sqrt{6}/3 & \mathbf{0} & 1 \end{array}\right], [\bar{R}_5^T \ \mathrm{O}]F \doteq \left[\begin{array}{c|cccc} -1 & 0 & 0 & 0 & 0 \\ \hline 0 & \sqrt{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & -1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \\ 1 & 0 & \mathbf{0} & \mathbf{0} & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & \sqrt{2} & \sqrt{2} & \mathbf{0} & 1 \end{array}\right].$$

Note that the result is the same (barring signs and the last column) as just after having annihiled the first element when we were rotating a_3^T, that the vector a_7 has been recovered in the additional column, and that in this case $\alpha = -1$. We have to reset to `NaN` the zeros shown in bold in order to conserve the predicted structure for R^T in the columns involved. The zero column just created has not appeared in the last position, hence $F \neq I$ in general. Once again, the work with the static structure avoids the explicit computation of the permutation F. We can check the result using MATLAB with `[Q,R]=qr(sparse([1 0 0 0 0 2; 1 0 -1 0 0 0; 0 0 0 0 1 1]))`.

5. Conclusions and future work

We have described how to update and downdate an upper trapezoidal sparse orthogonal factorization. We have adapted to rectangular matrices (with fewer columns than rows) the techniques of Saunders [34] for square matrices, by using the static data structure of George & Heath [13, 26] but allowing sparse row downdating on it. An implicitly determined column permutation allow us to dispense with its recomputing; we have shown how it fits well into the LINPACK downdating algorithm and ensures that the updated trapezoidal factor will remain sparse.

The updatable sparse factorization described in this paper has been implemented in a MATLAB toolbox, which has been used to solve the *sequence* of sparse compatible systems needed to implement certain non-simplex active-set methods for linear programming. The formulae needed and the computational results obtained when solving the first 15 smallest NETLIB problems with a highly-degenerate Phase I are the subject of a forthcoming article [21], in which we also comment on several advantages of using QR rather than LU, its suitability to a combined interior-point simplex methodology and several paralellizability issues.

The numerical analysis of the behaviour in floating-point arithmetic when solving *dense* quadratic programs with an orthogonal factorization of A_k^T similar to the one given in §2 was developed by Powell [32, and references therein]. We have devised [21] a combination with corrected seminormal equations to obtain an accurate downdating in the spirit of [6], an estimate of the condition number can be maintained (see [35]) and, when numerical difficulties with the rank arise, the matrix R_k can be postprocessed as indicated by Heath [26] to obtain a rank-revealing factorization. Postprocessing R_k for R_{l} to be well-conditioned can also be the subject of future research, as well as several paralellizable algorithmic modifications of the LINPACK algorithm given by Bischof et al. [4].

At present we only manage problems in which we can choose an *a priori* permutation P of the rows of A such that the Cholesky factor of PAA^TP^T is sparse. When this cannot be done (e.g., when A has dense columns), it would be better to use dynamic techniques as in [10, 11], adapting them to the semidefinite case to work with $A_kA_k^T$ or else regularizing the problem to work with $\sigma I_n + A_kA_k^T$. To deal with this case, we are developing [22] the necessary formulae to use the Cholesky factor of $\sigma I_n + A_kA_k^T$ within non-simplex active-set algorithms. We have also devised (see [33]) sparse orthogonal techniques to deal with $A_k^TA_k$, but they work on top of the static structure of $A^TA \in \mathbb{R}^{m\times m}$ rather than in that of $AA^T \in \mathbb{R}^{n\times n}$, so the combination with interior-point techniques is more difficult; anyway, they can be used when A has dense columns.

Acknowledgements

The authors thank the referees for having put a lot of work into producing the reports and marking up our previous manuscript, and for providing us

the references [11] and [3] that we were not aware of. We also want to thank John Gilbert at Xerox for confirming that the sparse orthogonal factorization in MATLAB 5 is row-oriented and Givens-based, and providing us technical details about the interface, as well as Christof Vömel at CERFACS for several remarks on the first sections.

References

1. Adlers, M. (1998). *Sparse least squares problems with box constraints.* Lic. thesis, Dpt. Mathematics, Linköping Univ.

2. Barlow, J.L. (1990). On the use of structural zeros in orthogonal factorization. *SIAM J. Sci. Statist. Comput;* 11:600–601.

3. Baryamureeba, V; Steihaug, T. (2000). Computational issues for a new class of preconditioners. In Griebel, M; Margenov, S; Yalamov, P; eds. *Large-Scale Scientific Computations of Engineering and Environmental Problems II*, Series Notes on Numerical Fluid Mechanics, pp. 128–135. Vieweg.

4. Bischof, C.H; Pan, C.-T; Tang, P.T.P. (1993). A Cholesky up- and downdating algorithm for systolic and SIMD architectures. *SIAM J. Sci. Comput;* 14(3):670–676.

5. Björck, Å. (1996). *Numerical Methods for Least Squares Problems.* SIAM Pubs; Philadelphia.

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

7. Chan, S.M; Brandwajn, V. (1986). Partial matrix refactorization. *IEEE Trans. Power Syst;* 1(1):193–200.

8. Coleman, T.F. (1984). *Large Sparse Numerical Optimization*, volume 165 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin.

9. Coleman, T.F; Edenbrandt, A; Gilbert, J.R. (1986). Predicting fill for sparse orthogonal factorization. *J. Assoc. Comput. Mach;* 33:517–532.

10. Davis, T; Hager, W. (1999). Modifying a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl;* 20:606–627.

11. Edlund, O. (2002). A software package for sparse orthogonal factorization and updating. *ACM Trans. Math. Soft;* 28(4):448–482.

12. Fletcher, R. (1987). *Practical Methods of Optimization*, second edition. John Wiley and Sons, Chichester.

13. George, A; Heath, M.T. (1980). Solution of sparse linear least squares problems using Givens rotations. *Linear Algebra Appl;* 34:69–83.

14. George, A; Heath, M.T; Ng, E. (1984). Solution of sparse underdetermined systems of linear equations. *SIAM J. Sci. Statist. Comput;* 5(4):988-997.

15. George, A; Liu, J; Ng, E. (1988). A data structure for sparse QR and LU factorizations. *SIAM J. Sci. Statist. Comput;* 9(1):100–121.

16. 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.

17. Gilbert, J.R; Peierls, T. (1988). Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Statist. Comput;* 9(5):862–874.

18. Golub, G.H. (1969). Matrix decompositions and statistical computation. In Milton, R.C; Nelder, J.A; eds. *Statistical Computation*, pp. 365–397. Academic Press, New York.

19. Golub, G.H; Van Loan, C.L. (1996). *Matrix Computations*. North Oxford Academic Publishing, Baltimore.

20. P. Guerrero-García. *Range-Space Methods for Sparse Linear Programs* (Spanish). Ph.D. thesis, Department of Applied Mathematics, University of Málaga, Spain, July 2002.

21. Guerrero-García, P; Santos-Palomo, Á. (2002). Solving a sequence of sparse compatible systems. Tech. report, Dpt. Applied Mathematics, Univ. Málaga. Submitted for publication to IMAJNA.

22. Guerrero-García, P; Santos-Palomo, Á (2003). Solving a sequence of sparse regularized least squares problems. Tech. report, Dpt. Applied Mathematics, Univ. Málaga.

23. Gustavson, F. G. (1976). Finding the block lower triangular form of a matrix. In Bunch, J.R; Rose, D.J; eds. *Sparse Matrix Computations*, pp. 275–289. Academic Press, New York.

24. Hager, W.W; Shih, C.-L; Lundin, E.O. (1996). Active set strategies and the LP dual active set algorithm. Tech. report, Dpt. Mathematics, Univ. Florida.

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. (1982). Some extensions of an algorithm for sparse linear least squares problems. *SIAM J. Sci. Statist. Comput;* 3(2):223–237.

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

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

29. Matstoms, P. (1994). *Sparse QR factorization with applications to linear least squares problems*. Ph.D. thesis, Dpt. Mathematics, Linköping Univ.

30. Mitra, G; Tamiz, M. (1991). Alternative methods for representing the inverse of linear programming basis matrices. In Kumar, S; ed. *Recent Developments in Mathematical Programming*, pp. 273-302. Gordon and Breach, Philadelphia.

31. Murtagh, B.A. (1981). *Advanced Linear Programming: Computation and Practice*. McGraw-Hill, New York.

32. Powell, M.J.D. (1981). An upper triangular matrix method for quadratic programming. In Mangasarian, O.L; Meyer, R.R; Robinson, S.M; eds. *Nonlinear Programming 4*, pp. 1–24. Academic Press, New York.

33. Santos-Palomo, Á; Guerrero-García, P. (2001). Solving a sequence of sparse least squares problems. Tech. report, Dpt. Applied Mathematics, Univ. Málaga. Submitted for publication to BIT.

34. Saunders, M.A. (1972). Large-scale linear programming using the Cholesky factorization. Tech. report CS-252, Computer Science Dpt; Stanford Univ.

35. Shroff, G.M; Bischof, C.H. (1992). Adaptive condition estimation for rank-one updates of QR factorizations. *SIAM J. Matrix Anal. Appl;* 13(4):1264–1278.

36. Stewart, G.W. (1995). On the stability of sequential updates and downdates. *IEEE Trans. Signal Proc;* 43:2642–2648.

37. Stewart, G.W. (1998). *Matrix Algorithms I: Basic Decompositions*. SIAM Pubs; Philadelphia.

38. Vempati, N; Slutsker, I.W; Tinney, W.F. (1991). Enhacements to Givens rotations for power system state estimation. *IEEE Trans. Power Syst;* 6(2):842–849.

39. Wright, S.J. (1997). *Primal Dual Interior Point Methods*. SIAM Pubs; Philadelphia.

Appendix

Let us show how the presence of the permutation matrix does not modify in essence the classical downdating algorithm, although as we shall see below it turns out not to be suitable to be combined with the static structure. If we add a column $e_1 = [1; 0; \ldots; 0] \in \mathbb{R}^{m_k}$, we have

$$Q_k^T \left[e_1 \; A_k^T\right] = [q \; R_k\Pi_k], \tag{3}$$

where $e_1^T Q_k \doteq q^T \in \mathbb{R}^{m_k}$. Now we use a Givens rotation $G(j, j+1)^T \in \mathbb{R}^{m_k \times m_k}$ with $j \in m_k - 1{:}\,{-1}{:}\,1$ to annihilate the $(j+1)$th element of q; after m_k rotations (in the worse case) we have

$$G^T q \doteq G(1,2)^T \cdots G(m_k - 2, m_k - 1)^T G(m_k - 1, m_k)^T q = \alpha e_1 \tag{4}$$

with $\alpha = \pm 1$, so

$$G^T [q \; R_k\Pi_k] = \begin{bmatrix} \alpha & \alpha v^T \Pi_{k+1} \\ \mathrm{O} & R_{k+1}\Pi_{k+1} \end{bmatrix}, \tag{5}$$

where $R_{k+1}\Pi_{k+1} \in \mathbb{R}^{(m_k-1)\times n}$ with R_{k+1} upper trapezoidal with nonzero diagonal elements. In this way, since the first row of the orthogonal matrix $Q_k G$ is $e_1^T Q_k G = \alpha e_1^T$ from (4), we can write —entering in (3) the identity matrix GG^T and applying (5)— that

$$\left[e_1 \; A_k^T\right] = Q_k G G^T [q \; R_k\Pi_k] = \begin{bmatrix} \alpha & \mathrm{O}^T \\ \mathrm{O} & Q_{k+1} \end{bmatrix} \begin{bmatrix} \alpha & \alpha v^T \Pi_{k+1} \\ \mathrm{O} & R_{k+1}\Pi_{k+1} \end{bmatrix},$$

so now we can compare terms on both sides of the equation to obtain, with $v^T \doteq a_q^T \Pi_{k+1}^T$, that

$$\begin{bmatrix} 1 & a_q^T \\ \mathrm{O} & A_{k+1}^T \end{bmatrix} = \left[e_1 \; A_k^T\right] = \begin{bmatrix} 1 & v^T \Pi_{k+1} \\ \mathrm{O} & Q_{k+1} R_{k+1} \Pi_{k+1} \end{bmatrix},$$

or, to put it differently, $A_{k+1}^T \Pi_{k+1}^T = Q_{k+1} R_{k+1}$.

We have assumed that q is dense in the worse case; when q is sparse, each rotation is done with respect to the last two rows in which q has nonzero elements, and finally a rotation is done with respect to the first row to obtain αe_1. The disadvantages of this algorithm in the sparse case is that the product $Q_k G$ has to be computed to obtain its submatrix Q_{k+1}, and the static structure cannot be maintained *unless a separate working vector is introduced*, see §4. To illustrate that the mere introduction of the implicitly defined column permutation on the classical downdating algorithm does not suffice to fit well into a static sparsity structure, it is instructive to compare the Linpack algorithm against the classical downdating algorithm. We start with $A_3 = (\Pi_3^T R_3^T) Q_3^T$ from §3.1:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & -1 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ 0 & -\sqrt{6}/3 & 1/\sqrt{3} \\ 0 & 0 & 0 \\ \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{6} & -1/\sqrt{3} \\ 0 & \sqrt{6}/3 & -1/\sqrt{3} \\ 1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3} \end{bmatrix}^T.$$

Since we want to delete a_7 we first have to exchange the first and second rows of A_3^T and Q_3:

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ 0 & -\sqrt{6}/3 & 1/\sqrt{3} \\ 0 & 0 & 0 \\ \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} \end{bmatrix} \begin{bmatrix} 0 & \sqrt{6}/3 & -1/\sqrt{3} \\ 1/\sqrt{2} & -1/\sqrt{6} & -1/\sqrt{3} \\ 1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3} \end{bmatrix}^T .$$

Using Givens rotations, we first postmultiply columns 2 and 3 by $\begin{bmatrix} \sqrt{6}/3 & 1/\sqrt{3} \\ -1/\sqrt{3} & \sqrt{6}/3 \end{bmatrix}$ and then columns 1 and 2 by $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$:

$$\begin{bmatrix} q^T \\ \Pi_3^T R_3^T \end{bmatrix} G(2,3)G(1,2) = \left[\begin{array}{ccc} 0 & \sqrt{6}/3 & -1/\sqrt{3} \\ \hline \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -\sqrt{6}/2 & 0 \\ 0 & -\sqrt{6}/3 & 1/\sqrt{3} \\ 0 & 0 & 0 \\ \sqrt{2} & -\sqrt{6}/3 & -2/\sqrt{3} \end{array}\right] G(2,3)G(1,2) =$$

$$= \left[\begin{array}{ccc} 0 & 1 & 0 \\ \hline \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ -1/\sqrt{2} & -1 & \mathbf{-1/\sqrt{2}} \\ 0 & -1 & 0 \\ 0 & 0 & 0 \\ \sqrt{2} & 0 & -\sqrt{2} \end{array}\right] G(1,2) = \left[\begin{array}{c|cc} -1 & 0 & 0 \\ \hline 0 & \sqrt{2} & 0 \\ 0 & 0 & 0 \\ 1 & -1/\sqrt{2} & -1/\sqrt{2} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \sqrt{2} & -\sqrt{2} \end{array}\right] .$$

Note that after the first rotation an unexpected fill-in (shown in bold) has appeared in the static structure. The new orthogonal factor results from

$$\begin{bmatrix} 0 & \sqrt{6}/3 & -1/\sqrt{3} \\ 1/\sqrt{2} & -1/\sqrt{6} & -1/\sqrt{3} \\ 1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sqrt{6}/3 & 1/\sqrt{3} \\ 0 & -1/\sqrt{3} & \sqrt{6}/3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= Q_3 G(2,3)G(1,2) = \begin{bmatrix} \alpha & \mathrm{O}^T \\ \mathrm{O} & Q_4 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} .$$