Marín Mateos-Aparicio, J.; Mas Marí, J.; Guerrero-Flores, DJ.; Hayami, K. (2017). Updating preconditioners for modified least squares problems. Numerical Algorithms. 75(2):491-508. doi:10.1007/s11075-017-0315-z

Updating preconditioners for modified least squares problems *

J. Marín
Instituto de Matemática Multidisciplinar
Universitat Politècnica de València
València, España
jmarinma@imm.upv.es

J. Mas
Instituto de Matemática Multidisciplinar
Universitat Politècnica de València
València, España
jmasm@imm.upv.es

D. Guerrero
Departamento de Ciencias Matemáticas
Universidad Pedagógica Nacional Francisco Morazán
Tegucigalpa, Honduras
danguefl@doctor.upv.es

K. Hayami
National Institute of Informatics,
SOKENDAI (The Graduate University for Advanced Studies)
Tokyo, Japan
hayami@nii.ac.jp

April, 2017

Abstract

In this paper we analyze how to update incomplete Cholesky preconditioners to solve least squares problems using iterative methods when the set of linear relations is updated with some new information, a new variable is added or, contrarily, some information or variable is removed from the set. Our proposed method computes a low-rank update of the preconditioner using a bordering method which is inexpensive compared with the cost of computing a new preconditioner. Moreover the numerical experiments presented show that this strategy gives, in many

*Partially supported by Spanish Grants MTM2014-58159-P and MTM2015-68805-REDT.

cases, a better preconditioner than other choices, including the computation of a new preconditioner from scratch or reusing an existing one.

Keywords: Least squares problems, Iterative methods, Preconditioners, Low-rank updates, Sparse matrices

Mathematics Subject Classification: 93E24, 65F08, 65F10, 65F50

1 Introduction

Iterative methods are used for solving large and sparse linear least squares (LS) problems because they often require much less storage than their direct counterparts. One of the most used iterative methods for LS problems is CGLS [5]. CGLS is equivalent to applying the Conjugate Gradient method (CG) to the normal equations. To improve the convergence of the iterative method very often a preconditioner is needed. Among other choices like Incomplete QR factorizations preconditioners, we will focus on Incomplete Cholesky (IC) preconditioners. These preconditioners have been successfully employed in different applications, see [18, 19], and allow for the computation of robust preconditioners for full rank overdetermined least squares problems [3, 6].

The problem of updating a preconditioner arise in some applications from statistics and optimization, where it is necessary to solve a sequence of modified least squares problems. An example can be found in [8], where an efficient and stable method for adding and deleting equations to a regression model is required. In signal processing applications near real-time solutions are required. Thus, methods that allow to modify LS problems with few operations and little storage requirements are needed, see [1]. The same problem is present if some information is added to or deleted from the data set. On some occasions it may be convenient to add or to remove some variables. Such situations are usually referred to as updating or downdating least squares problems. Chapter 3 of the reference text [5] is devoted to analyzing how to deal with these modifications when the least squares problem is solved by a direct method, including full and rank revealing QR decomposition, Cholesky factorization and singular value decomposition. More recently other algorithms to update Cholesky factorizations have been proposed, see [10, 11, 12]. More efforts seem to be addressed to updating the QR factorization, see [2, 14, 15].

In this paper we present a method to modify an existing incomplete factorization with low computational cost. We note that when some columns are removed from an overdetermined system, obtaining a preconditioner for the modified LS problem can be done without additional cost by taking a block from the existing one. A similar situation occurs when some columns are added to an overdetermined system, or when new relations are added to an underdetermined one. In both cases the old preconditioner is the top left block of the new one. Thus, it is a preconditioner completion problem. The final result is equivalent to computing a new preconditioner from scratch. These trivial cases are not considered in this paper.

The cases in which we are interested correspond to LS modified problems whose normal equations have a coefficient matrix that does not change in size but their entries do. These problems can be formulated as a low-rank update of the original normal equations and we propose updating the preconditioner following the ideas presented in

[7]. The goal is computing the update with smaller cost than obtaining a new preconditioner from scratch, but with comparable performance.

The paper is organized as follows. In Section 2 we describe the bordering technique used to update an existing preconditioner by using an equivalent augmented system. In Section 3 we consider adding or deleting equations to an overdetermined least squares problem. The opposite case, that is when the system is underdetermined, is analyzed in Section 4. We will see that there is a duality between both groups of problems. In Section 5 we present the results of the numerical experiments that show that the proposed strategy is effective.

2 Preconditioner update computation and application

Suppose that the least squares solution of the overdetermined linear system

$$Ax = b, \tag{1}$$

where A is a large and sparse $m \times n$ matrix, $m > n$, has been computed using a preconditioned iterative method. We assume that A has full rank, n, that is, its columns are linearly independent. As it is well known, the LS solution is given by the vector x that minimizes $\| b - Ax \|_2$, and can be obtained by solving the normal equations corresponding to (1) given by

$$A^T Ax = A^T b. \tag{2}$$

We are interested in computing the least squares solution of a new linear system obtained after the original system has been modified by adding or removing k equations. As it is shown in the next section, the normal equations for the modified linear system can be written in these cases as

$$(A^T A \pm B^T B)x = c. \tag{3}$$

where B is a $k \times n$ matrix.

Observe that the solution of (3) can be obtained from the solution of the equivalent linear system

$$\begin{bmatrix} A^T A & B^T \\ B & \mp I \end{bmatrix} \begin{bmatrix} x \\ \pm Bx \end{bmatrix} = \begin{bmatrix} c \\ 0 \end{bmatrix}. \tag{4}$$

One has the following relations between the linear operators in (3) and (4),

$$A^T A \pm B^T B = \begin{bmatrix} I & O \end{bmatrix} \begin{bmatrix} A^T A & B^T \\ B & \mp I \end{bmatrix} \begin{bmatrix} I \\ \pm B \end{bmatrix} \tag{5}$$

and their inverses

$$(A^T A \pm B^T B)^{-1} = \begin{bmatrix} I & O \end{bmatrix} \begin{bmatrix} A^T A & B^T \\ B & \mp I \end{bmatrix}^{-1} \begin{bmatrix} I \\ O \end{bmatrix}. \tag{6}$$

The preconditioner update technique consists in computing an incomplete factorization for the augmented matrix in (4) that is used to approximate the inverse linear operator in

(6) by direct preconditioning, i.e., solving the corresponding upper and lower triangular systems. Therefore we avoid the computation of a new preconditioner for the updated matrix $A^TA \pm B^TB$ from scratch.

To be precise, let $A^TA \approx R^TR$ be an IC factorization of A^TA, where R is an upper triangular matrix. Then one gets a block LDL^T (almost Cholesky) factorization of the augmented matrix in (5) given by

$$\begin{bmatrix} A^TA & B^T \\ B & \mp I \end{bmatrix} = \begin{bmatrix} R^T & 0 \\ R_{12}^T & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \mp S \end{bmatrix} \begin{bmatrix} R & R_{12} \\ 0 & I \end{bmatrix}, \tag{7}$$

where $R_{12} = R^{-T}B^T$ is a $n \times k$ matrix and $S = I \pm R_{12}^T R_{12}$ is a $k \times k$ matrix. To maintain sparsity in these factors some dropping strategy can be used when computing R_{12} and an incomplete factorization of the Schur complement $S \approx R_S^T R_S$ as well, but if k is small enough this block can be factorized exactly. Note that although the (approximate) inverse operator in the form of (6) is symmetric and positive definite (spd), it is not stored nor can it be applied in factorized form. Therefore, only left or right preconditioning can be used when applying the conjugate gradient method to the normal equations (or the mathematically equivalent CGLS method). The preconditioning step for a Krylov subspace iterative method typically consists of obtaining the preconditioned vector $s = M^{-1}r$ where M^{-1} is the preconditioner and r is the residual. M^{-1} should be a good sparse approximation of the inverse of the coefficient matrix of the linear system (3). Thus, the preconditioning strategy proposed computes the preconditioned residual by applying equation (6) with an incomplete factorization of the augmented matrix. That is, the preconditioned residual s is given by

$$s = \begin{bmatrix} I & O \end{bmatrix} \begin{bmatrix} A^TA & B^T \\ B & \mp I \end{bmatrix}^{-1} \begin{bmatrix} I \\ O \end{bmatrix} r,$$

and it is computed from the solution of

$$\begin{bmatrix} R^T & 0 \\ R_{12}^T & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \mp S \end{bmatrix} \begin{bmatrix} R & R_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} s \\ s' \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}. \tag{8}$$

The preconditioning is done in three steps as Algorithm 1 shows.

Algorithm 1 Preconditioner update application

Input: Matrices R, R_{12}, R_S and residual vector r.
Output: Preconditioned vector s
1. Solve the linear system $R^T\tilde{r} = r$.
2. Update $\tilde{r} \leftarrow \tilde{r} \mp R_{12}(R_S^TR_S)^{-1}R_{12}^T\tilde{r}$.
3. Solve the linear system $Rs = \tilde{r}$.

Step 2 in Algorithm 1 represents the extra cost in the application of the preconditioner with respect to the case of non-updating an existing one. If R_{12} and R_S are kept sparse and the number of added or removed equations is small compared with the problem size, this overhead is small and can be amortized even for moderate reductions on the number of iterations, see [7].

3 Updating preconditioners: Overdetermined case

In this section we consider the modification of the overdetermined linear system (1) adding and/or removing equations. We analyze and propose strategies to get a preconditioner for the new normal equations when adding or/and removing equations.

3.1 Adding equations to an overdetermined system

It may happen that some new relations among the unknowns are considered. If these relations are given as the system of k linear equations

$$Bx = c,$$

then we have the $m+k$ system of linear equations

$$\begin{bmatrix} A \\ B \end{bmatrix} x = \begin{bmatrix} b \\ c \end{bmatrix}.$$

If A has full rank, the new coefficient matrix $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$ has also full rank, and the corresponding normal equations are

$$(A^T A + B^T B)x = A^T b + B^T c. \tag{9}$$

That is, the new normal equations are the result of a low-rank update of the initial ones. If we put $f = A^T b + B^T c$, the preconditioner update technique proposed in Section 2 can be applied with the augmented linear system

$$\begin{bmatrix} A^T A & B^T \\ B & -I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}.$$

3.2 Removing equations from an overdetermined system

This is just the opposite case. Suppose that instead of adding new information, some linear equations are removed from the initial linear system $Ax = b$. After a suitable row permutation, the new system can be written as

$$\begin{bmatrix} A_1 \\ B \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{10}$$

where $A_1 \in \mathbb{R}^{(m-k)\times n}$, $m-k > n$ and $B \in \mathbb{R}^{k\times n}$, and it is assumed that rank $\left[\begin{smallmatrix} A_1 \\ B \end{smallmatrix}\right] = n$. Assume the information corresponding to the bottom block must be removed. The normal equations corresponding to (10) are

$$(A_1^T A_1 + B^T B)x = A_1^T b_1 + B^T b_2. \tag{11}$$

Observe that the row permutation is irrelevant when forming the normal equations. In fact if M is a matrix and P is a permutation matrix, $(PM)^T(PM) = M^T(P^T P)M = M^T M$.

After deleting the bottom block, one gets the linear system $A_1 x = b_1$, whose normal equations, $A_1^T A_1 x = A_1^T b_1$, can be related to (11) by

$$\big(A^T A - B^T B\big)x = A_1^T b_1.$$

This system is again the result of a rank k modification of the initial normal equations, and it has the same solution as component x in the solution of the augmented linear system

$$\begin{bmatrix} A^T A & B^T \\ B & I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A_1^T b_1 \\ 0 \end{bmatrix},$$

which allows for the application of the preconditioner update strategy described in Section 2, provided that A_1 has full rank. Otherwise, the singularity of $A_1^T A_1$ may produce poor preconditioners or even a breakdown during the computation of the preconditioner. Since the new coefficient matrix has less rows than the original one, it may happen that the remaining set of equations has rank less than n. In this case the square matrix $\left[\begin{smallmatrix} A^T A & B^T \\ B & I \end{smallmatrix}\right]$ is singular. To prove it, let $\operatorname{rank} A_1 = r < n$. Then, $\operatorname{rank} A_1^T A_1 = r < n$, and let $B \in \mathbb{R}^{k \times n}$ of rank k. Let us do a symmetric permutation, $\left[\begin{smallmatrix} O & I \\ I & O \end{smallmatrix}\right]$, to the matrix so that the permuted matrix is $\left[\begin{smallmatrix} I & B \\ B^T & A^T A \end{smallmatrix}\right]$. After eliminating the left bottom block by Gaussian elimination obtaining $\left[\begin{smallmatrix} I & B \\ 0 & A_1^T A_1 \end{smallmatrix}\right]$, which has rank $k + r < k + n$. Of course, computing a new preconditioner from scratch in this case can be difficult for the same reasons. This is illustrated with an example with the matrix ASH219 in the numerical experiments section, see Figures 2, 3 and 4.

3.3 Adding and removing equations from an overdetermined system

Now suppose that both things occur simultaneously, that is, some equations are added and some others are deleted. To fix the notation, starting with the linear system $Ax = b$ written as

$$\begin{bmatrix} A_1 \\ B \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{12}$$

one wants to solve the linear system obtained after removing the bottom equations $Bx = b_2$, and then adding some new equations $Cx = c$, such that the new linear system is

$$\begin{bmatrix} A_1 \\ C \end{bmatrix} x = \begin{bmatrix} b_1 \\ c \end{bmatrix}. \tag{13}$$

The normal equations for systems (12) and (13) are

$$(A_1^T A_1 + B^T B)x = A_1^T b_1 + B^T b_2 \tag{14}$$

and

$$(A_1^T A_1 + C^T C)x = A_1^T b_1 + C^T c, \tag{15}$$

respectively. If we consider the augmented system

$$\begin{bmatrix} A^T A & C^T & B^T \\ C & -I & 0 \\ B & 0 & I \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} A_1^T b_1 + C^T c \\ 0 \\ 0 \end{bmatrix} \tag{16}$$

we obtain

$$A^TAx + C^Ty + B^Tz = A_1^Tb_1 + C^Tc$$
$$Cx - y = 0$$
$$Bx + z = 0.$$

Hence, $y = Cx$ and $z = -Bx$ and substituting in the first equation we obtain

$$(A^TA + C^TC - B^TB)x = (A_1^TA_1 + C^TC)x = A_1^Tb_1 + C^Tc.$$

Therefore, problem (15) is a low-rank update of the initial problem (14) and the proposed strategy can be used provided that $\left[\begin{smallmatrix} A_1 \\ C \end{smallmatrix}\right]$ has full rank. Note that the new coefficient matrix can have full rank even when A_1 has not, depending on C. Observe also that the coefficient matrix in (16) is nonsingular if and only if the matrix $\left[\begin{smallmatrix} A_1 \\ C \end{smallmatrix}\right]$ has full rank since the Schur complement of the $(1,1)$ block is $A_1^TA_1 + C^TC$.

4 Updating preconditioners: Underdetermined case

Consider now the LS problem

$$\min\|x\|_2 \qquad \text{subject to } Ax = b, \tag{17}$$

where $A \in \mathbb{R}^{m\times n}$, is a large and sparse full rank matrix with $m < n$. Problem (17) is solved using the second kind normal equations

$$AA^Tz = b, \qquad y = A^Tz. \tag{18}$$

Since $y = A^T(AA^T)^{-1}b$, y belongs to the row subspace of A which is orthogonal to the Kernel of A. Thus, y is the solution of (17).

As in the overdetermined case, it is assumed that an incomplete Cholesky factorization of the symmetric positive definite matrix AA^T has been computed. Then, new unknowns are added to the linear system or some of them are deleted, or both. In the following, we will see that the preconditioner can be updated under the same conditions and with similar techniques as for the cases studied in Section 3.

4.1 Adding columns

Now we consider the problem of adding unknowns to (17), so that the new LS problem is

$$\min\|x\|_2 \qquad \text{subject to } \begin{bmatrix} A & B \end{bmatrix} x = b. \tag{19}$$

The normal equations of the second kind in this case are

$$(AA^T + BB^T)z = b, \tag{20}$$

that corresponds to a low-rank update similar to the one described in subsection 3.1 since the augmented linear system

$$\begin{bmatrix} AA^T & B^T \\ B & -I \end{bmatrix} \begin{bmatrix} z \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

provides the solution for (20). Therefore, the updating strategy proposed in Section 2 can be applied.

4.2 Removing columns

Assume that the linear system $Ax = b$ is splitted as

$$\begin{bmatrix} A_1 & B \end{bmatrix} x = b,$$

where B represents the block of columns to be removed. The corresponding normal equations of the second kind are

$$(A_1 A_1^T + BB^T)z = b.$$

To solve the new normal equations $A_1 A_1^T z = b$, one can consider the augmented system

$$\begin{bmatrix} AA^T & B \\ B^T & I \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.$$

This is similar to the situation in subsection 3.2. Then, the proposed strategy to update the preconditioner can be applied.

4.3 Adding and removing columns

The last problem that we study in this section corresponds to the case of removing the last set of columns in the underdetermined linear system $Ax = b$ given by

$$\begin{bmatrix} A_1 & B \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = b \tag{21}$$

and adding a new block C to get a new underdetermined problem given by

$$\begin{bmatrix} A_1 & C \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} = b. \tag{22}$$

The normal equations of the second kind for (22) are

$$(A_1 A_1^T + CC^T)z = b,$$

that can be written as

$$(AA^T - BB^T + CC^T)z = b.$$

The solution of these equations, z, can be obtained from the solution of the augmented system

$$\begin{bmatrix} AA^T & B & C \\ B^T & I & 0 \\ C^T & 0 & -I \end{bmatrix} \begin{bmatrix} z \\ v \\ w \end{bmatrix} = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}.$$

Observe that the coefficient matrix of this system is similar to the one in (16). Then, the same comments and strategies used in subsection 3.3 apply to this case.

5 Numerical experiments

In this section we study the numerical performance of the preconditioner update method proposed. We present results obtained with matrices arising in different areas of scientific computing. The performance of the method is compared with other preconditioning strategies. The first one is reusing the initial preconditioner computed for the normal equations of the unmodified matrix. The second strategy corresponds to the computation of a new almost Cholesky preconditioner for the updated matrix from scratch. In addition, non-preconditioned iterations are also reported.

We present results for the modifications described in Sections 3 and 4 that correspond to adding and removing equations or columns.

Matrix name	rows	cols	nnz	Application
PHOTOGRAMMETRY2	4472	936	37056	Computer graphics/vision problem
TESTBIG	17613	31223	61639	Linear programming problem
CAT_EARS_4_4	19020	44448	132888	Combinatorial problem
DELTAX	68600	21961	247424	High fillin with exact partial pivoting
FOME13	48568	97840	285046	Linear programming problem
LP_KEN_18	105127	154699	358171	Linear programming problem
FLOWER_8_4	55081	125361	375266	Combinatorial problem
FXM3_16	41340	85575	392252	Linear programming problem
LP_OSA_30	4350	104375	604488	Linear programming problem
MESH_DEFORM	234023	9393	853829	Image mesh deformation problem
WATSON_1	201155	386992	1055093	Linear programming problem
TS-PALKO	22002	47235	1076903	Linear programming problem
LP_NUG30	52260	379350	1567800	Linear programming problem
LARGEREGFILE	2111154	801374	4944201	Circuit simulation problem
SLS	1748122	62729	6804304	Statistics
TP-6	142752	1014301	11537419	Linear programming problem

Table 1: Set of test matrices

The tested matrices are shown in Table 1. All the matrices can be downloaded from the University of Florida Sparse Matrix Collection [9]. For each matrix we provide its number of rows and columns, the number of its nonzero entries, nnz, and the application field. The matrices with more rows than columns were used to obtain the numerical results corresponding to the overdetermined case, while the rest, mainly matrices arising from linear programming problems, were used for the undetermined one.

The preconditioned CGLS [5] or CGNR [19] for the over determined problems, and the preconditioned (CGNE [19]) were used for a relative initial residual norm decrease of 10^{-8}, allowing a maximum number of $3,000$ iterations. The right hand side vector was computed as a random vector. The initial approximation to the solution x was the vector of all zeros. The experiments where done with MATLAB version 2016a running on an Intel 5 CPU with 8 Gb of RAM in a Windows operating system. We used MATLAB's function **ilu**() to compute the incomplete factorizations since, for some matrices, the computation of a Cholesky factorization with the MATLAB's function **ichol**() stopped with a breakdown. Moreover, we found that permuting the coefficient matrix to block triangular form before computing the normal equations improved the quality of the preconditioner. Thus, all the matrices were permuted using the MATLAB's function **dmperm**() that obtains the Dulmage-Mendelsohn decomposition

[17]. Symmetric diagonal scaling was applied to the matrices. The dropping parameter for managing the fill-in of the preconditioners was set to 0.1 except for the matrices DELTAX and MESH_DEFORM for which a value of 0.01 was used. We avoided fine tuning of the drop tolerance and with these values we computed very sparse preconditioners.

Tables 2 and 3 report the results for the cases of adding and removing equations or columns, depending of the problem. In these tables, k represents the rank of the update, i.e., the number of equations added or removed. This parameter is given in absolute number and also in percentage compared with the largest dimension of the matrix. We tested several values, but in the Tables we only report three results for each matrix that correspond to small, medium and large modifications up to a maximum of five percent. The relative density of the preconditioner with respect to the updated matrix is indicated in the column ρ. For simplicity the minimum and maximum density values observed for the preconditioners considered are shown. Normally, the minimum value corresponds to the non-updated preconditioner while the maximum was achieved for either, the recomputed or the updated preconditioner. The number of iterations and CPU solution time, measured in seconds, are indicated with Its. and Time, respectively. We recall that the application of the updated and the recomputed preconditioners have, with respect to the two other strategies, an extra cost due to the computation of the preconditioner update or the computation of the full preconditioner from scratch, respectively. Therefore, in the tables the value Time reports the total CPU time corresponding to the preconditioner computation and the iterative solution spent by these two strategies. As recommended in MATLAB's documentation, CPU times reported are the mean value of 10 successive runs of the experiment performed after 3 initial runs that were discarded. The maximum standard deviation observed relative to the mean value was 3 percent, and frequently less than 1 percent.

We start analyzing the results for the case of adding equations or columns that are shown in Table 2. The equations added were obtained by selecting at random k rows of the original matrix, and ordering in reverse order their column entries to avoid duplicated rows. In the case of adding columns, the modification was obtained similarly but with the columns of the matrix. The preconditioner density is very small for most of the preconditioners and always below one. It is important to note that in our algorithm, to compute an update with moderate fill-in, element dropping was applied in three different steps. First, a sparsification of the new block of rows (equations) added to the matrix was done before computing the block column R_{12} in equation (7). Then, the computation of the block R_{12} itself was done incompletely by dropping small entries. Finally, an incomplete factorization was computed for the Schur-complement block S, see equation (7). The respective tolerances were 1.0, 1.0 and 0.1 for all the matrices. Although for small values of k exact factorization of the Shur complement S could be done, we avoided fine tuning and performed incomplete factorization with the same drop tolerance used for the normal equations. We note that, with this aggressive dropping the total solution time was reduced, because the application of the preconditioner is cheaper, and also in some cases the number of iterations needed to converge was reduced. We recall that adding fill-in is not directly correlated with fewer number of iterations and, actually an increment is possible as is reported for instance in [6] for incomplete Cholesky factorizations for LS problems, see also [4].

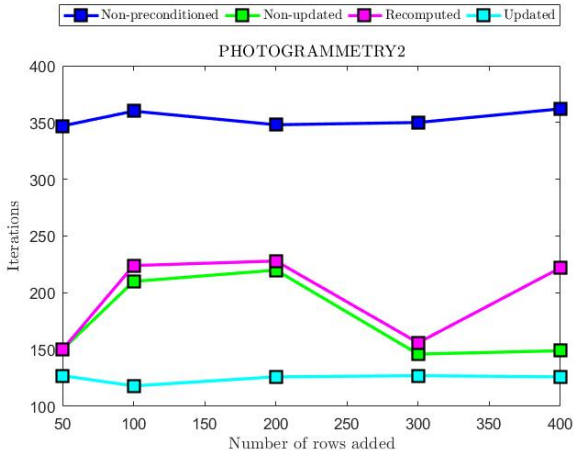

Figure 1: Effect of the number of equations added on the number of iterations for the matrix PHOTOGRAMMETRY2.

From the number of iterations we see that the updated preconditioner performed better than the non-updated one, and similar to the case of recomputing the preconditioner. Taking into account the overall time, our strategy performed similarly to the best of the other strategies in most of the cases, and it was the best in several cases. For example, Figure 1 shows the evolution of the number of iterations when the number of added rows increases, for the matrix PHOTOGRAMMETRY2. In this case the proposed strategy performed better than the others with almost constant number of iterations.

Analyzing the results in Table 3 we observe that, if instead of adding equations (columns) the modification consists of removing a block of them, the situation changes in favor of the proposed algorithm. In this case, when the number of equations removed increases, sometimes the preconditioner can not be computed or it is very poor. Therefore, the application of the recomputed preconditioner can even lead to a divergence of the iterative solution method. Recall that, after removing equations, it is not warranted that the new matrix keeps its full rank. This can explain the big increment in the number of iterations needed to converge, and even the failure to converge in some cases. Under these conditions the proposed updating strategy performed nicely, and surprisingly it kept an almost constant performance independently of the number of equations or columns removed.

To illustrate the comments above we did an experiment with the matrix ASH219, also from the University of Florida Sparse Matrix Collection [9]. Its size is 219×85 and has full rank. Figure 2 illustrates the condition numbers of the normal equations, of the bordered matrix and the Schur complement block S, when successive rows are deleted from the end of the matrix. We observe that when a small number of rows are deleted all condition numbers remain low and have the same order up to some point where all them increase similarly. But eventually as more rows are deleted the condi-

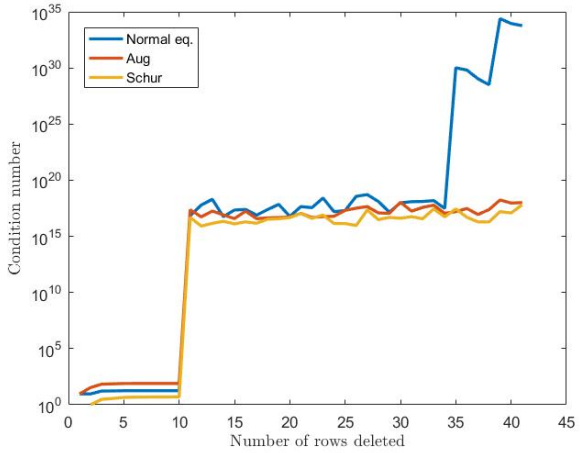

Figure 2: Condition numbers of the normal equations, equivalent bordered matrix and Schur complement matrix S for matrix ASH219 when removing rows from bottom.

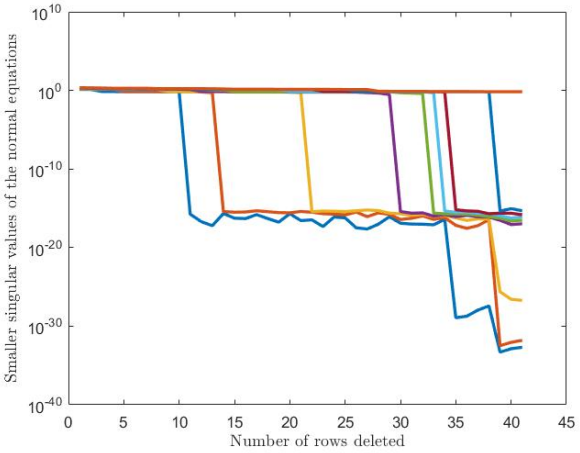

Figure 3: Decay evolution of the smallest 9 singular values of the normal equation matrix when removing rows from the bottom for matrix ASH219.

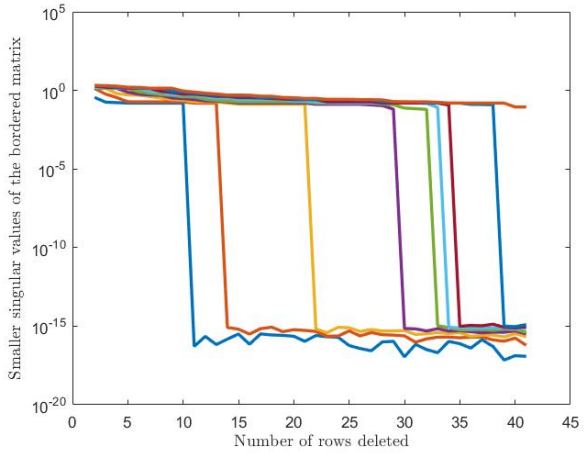

Figure 4: Decay evolution of the smallest 9 singular values of the augmented matrix when removing rows from the bottom for matrix ASH219.

tion number of the normal equations rises quickly while the other condition numbers remain almost constant. Figure 3 shows the decay evolution of the smallest singular values of the matrix when rows are deleted. It is observed that the increment of the condition number is acompanied by a progressive increment of the number of singular values that are clustered closer and closer to zero, while the decay in the singular values for the augmented system is less pronounced as Figure 4 shows. This may explain the convergence's degradation of the iterative method and why the updating technique gives better results than recomputing the preconditioner from scratch for some problems.

We note than as more rows (columns) are removed the matrix may loose its full rank, we observed this situation for example in the case of the matrix FXM3_16. CGLS then converges to the pseudo-inverse solution if the initial approximation x^0 is in the range of A^T, as in our choice of x^0 as the vector of all zeros [5, p. 291]. In practice this convergence can be very slow, or even can stagnate. Also it may be very difficult to compute the preconditioner due to breakdowns, as happen in some cases when recomputing it from scratch. Using our strategy the preconditioner was computed successfully in all cases and preconditioned CGLS converged quite fast.

Although for some matrices, for example TESTBIG and LP_OSA_30, the results are comparable in terms of time, we recall that the reduction of the number of iterations spent by the iterative method may have a bigger impact in the overall solution time when increasing the problem size, as the matrix SLS shows, see Figure 5.

Overall, we can conclude that the proposed algorithm is competitive and robust since it was able to successfully solve all the problems. The number of iterations and time spent was the best, or close to it, for the majority of cases. Another conclusion is that, in general, it is better to apply the update or recompute a new preconditioner from scratch instead of reusing the original one. In any case, these three strategies

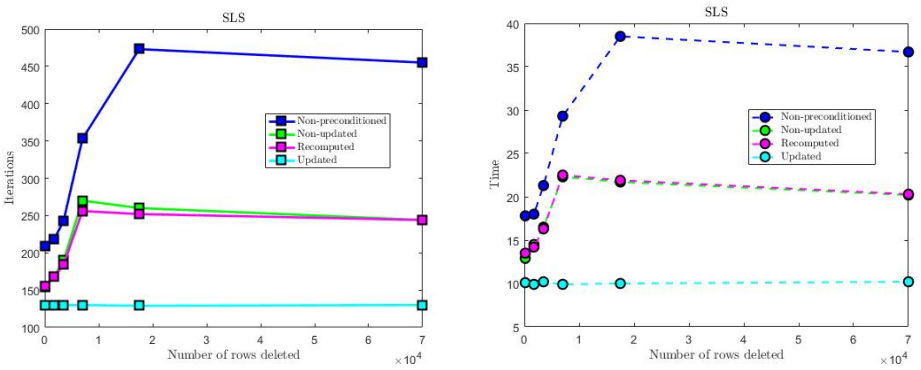

Figure 5: Effect of the number of equations deleted in the number of iterations (left) and in the total time to get the solution (right) for the matrix SLS.

are better than non-preconditioned iterations. Computing a new preconditioner from scratch may have two drawbacks. The first one is an increment on the set-up time that usually only pays off in the case of adding equations when the size of the update is quite large. The second one is that when removing equations, the preconditioner computation may become unstable probably due to an increment of the condition number of the coefficient matrix of the normal equations.

6 Conclusions and future work

The main conclusion of this work is that adding equations, or removing them or both, as well as adding or removing or simply changing some variables is feasible when solving least squares problems with preconditioned iterative methods, provided that the resulting coefficient matrix has full rank.

In the most difficult cases, that is, when the coefficient matrix of the normal equations are modified by a low-rank matrix, we have introduced a technique, based on bordering, that allows to update the preconditioner in an inexpensive way. This technique has moderate memory and computational requirements, as demanded in [1]. Moreover, it is effective and robust as our numerical experiments show.

In the future we plan to combine the proposed strategy with other regularization techniques to compute the least squares solution of rank deficient linear systems.

ACKNOWLEDGEMENTS We would like to acknowledge the anonymous referees for their helpful comments that substantially contributed to improve this paper.

References

[1] Alexander, S.T., Pan, C.T., Plemmons, R.J.: Analysis of a recursive least squares hyperbolic rotation algorithm for signal processing. Linear Algebra Appl. **98**, 3–40 (1988)

[2] Andrew, R., Dingle, N.: Implementing QR factorization updating algorithms on GPUs. Parallel Comput. **40**(7), 161 – 172 (2014). DOI http://dx.doi.org/10.1016/j.parco.2014.03.003. http://www.sciencedirect.com/science/article/pii/S0167819114000337. 7th Workshop on Parallel Matrix Algorithms and Applications

[3] Benzi, M., Tůma, M.: A robust incomplete factorization preconditioner for positive definite matrices. Numer. Linear Algebra Appl. **10**(5-6), 385–400 (2003)

[4] Benzi, M., Szyld, D. B., Van Duin, A.: Orderings for Incomplete Factorization Preconditioning of Nonsymmetric Problems. SIAM J. Sci. Comput. **20**(5), 1652–1670 (1999)

[5] Björck, Å.: Numerical methods for Least Squares Problems. SIAM, Philadelphia (1996)

[6] Bru, R., Marín, J., Mas, J., Tůma, M.: Preconditioned iterative methods for solving linear least squares problems. SIAM J. Sci. Comput. **36**(4), A2002–A2022 (2014)

[7] Cerdán, J., Marín, J., Mas, J.: Low-rank updates of balanced incomplete factorization preconditioners. Numer. Algorithms (2016). DOI http://dx.doi.org/10.1007/s11075-016-0151-6

[8] Chambers, J.M.: Regression updating. J. Amer. Statist. Assoc. **66**, 744–748 (1971)

[9] Davis T. A. and Hu Y.: The University of Florida Sparse Matrix Collection. ACM Trans. Math. Software, **38**(1), 1–25 (2011)

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

[11] Davis, T.A., Hager, W.W.: Multiple-rank modifications of a sparse Cholesky factorization. SIAM J. Matrix Anal. Appl. **22**, 997–1013 (2001)

[12] Davis, T.A., Hager, W.W.: Row modification of a sparse Cholesky factorization. SIAM J. Matrix Anal. Appl. **26**, 621–639 (2005)

[13] Golub, G.H., Van Loan, C.F.: Matrix Computations. Johns Hopkins University Press, Baltimore, Maryland (1983)

[14] Hammarling, S., Lucas, C.: Updating the QR factorization and the least squares problem. Tech. rep., The University of Manchester, http://www.manchester.ac.uk/mims/eprints (2008)

[15] Olsson, O., Ivarsson, T.: Using the QR factorization to swiftly update least squares problems. Thesis report, Centre for Mathematical Sciences. The Faculty of Engineering at Lund University, LTH (2014)

[16] Ortega, J.M.: Introduction to Parallel and Vector Computing. Plenum Press, New York (1988)

[17] Pothen, A., Fan, C.J.: Computing the block triangular form of a sparse matrix. ACM Trans. Math. Software **16**, 303–324 (1990)

[18] Saad, Y.: ILUT: a dual threshold incomplete *LU* factorization. Numer. Linear Algebra Appl. **1**(4), 387–402 (1994)

[19] Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing Co., Boston (1996)

Matrix	k	ρ	non-prec Its./Time	non-updated Its./Time	recomputed Its./Time*	updated Its./Time*
TESTBIG	300/0.96	[0.54,0.57]	86/0.1	72/0.1	66/0.2	70/0.1
	600/1.92		89/0.1	73/0.1	69/0.2	67/0.1
	1200/3.84		87/0.1	72/0.1	66/0.2	72/0.1
CAT_EARS_4_4	400/0.90	[0.70,0.77]	151/0.2	83/0.3	82/0.4	82/0.2
	1100/2.47		158/0.2	87/0.3	87/0.3	86/0.3
	2200/4.95		149/0.2	81/0.3	81/0.4	81/0.3
DELTAX	300/0.44	[0.95,0.98]	768/1.4	919/3.3	920/3.8	873/3.0
	1200/1.75		787/1.6	914/3.1	829/3.6	788/2.8
	3400/4.96		903/1.8	899/3.1	894/3.7	863/2.9
FOME13	200/0.20	[0.74,0.78]	457/1.5	272/1.5	289/1.9	270/1.4
	1000/1.02		494/1.6	303/1.6	301/2.0	297/1.6
	4000/4.08		527/1.8	309/1.8	304/2.0	306/1.7
LP_KEN_18	500/0.32	[0.75,0.78]	511/2.6	167/1.4	167/1.4	152/1.2
	1000/0.65		509/2.6	154/1.3	155/1.4	154/1.2
	5000/3.23		519/2.6	169/1.4	168/1.5	169/1.4
FLOWER_8_4	500/0.40	[0.77,0.80]	177/0.6	104/0.7	103/1.2	103/0.7
	1000/0.80		190/0.7	94/0.7	99/1.2	93/0.7
	6000/4.79		182/0.8	102/0.8	97/1.2	101/0.8
FXM3_16	100/0.12	[0.34,0.38]	1995/4.9	616/2.9	605/2.4	608/2.4
	1000/1.17		2870/7.1	754/2.8	712/2.8	714/2.8
	4000/4.68		†	781/3.1	802/3.1	794/3.1
LP_OSA_30	500/0.48	[0.01,0.02]	132/0.4	66/0.2	55/0.2	60/0.2
	1000/0.96		134/0.4	67/0.2	57/0.2	57/0.2
	5000/4.79		142/0.4	98/0.4	57/0.3	56/0.3
MESH_DEFORM	230/0.10	[0.07,0.11]	473/2.9	228/1.7	228/1.8	196/1.3
	2300/0.98		474/2.9	230/1.7	229/1.8	200/1.3
	9200/3.94		462/2.9	236/1.8	217/1.8	207/1.4
WATSON_1	500/0.13	[0.45,0.48]	638/7.0	342/6.4	420/7.7	342/6.5
	5000/1.29		622/6.9	343/6.8	576/9.8	342/6.8
	15000/3.88		627/7.1	333/6.7	612/11.1	332/6.6
TS-PALKO	500/1.06	[0.02,0.03]	48/0.3	48/0.3	47/0.3	44/0.3
	1000/2.12		49/0.2	48/0.3	48/0.4	44/0.3
	2000/4.23		48/0.2	48/0.3	48/0.4	44/0.3
LP_NUG30	500/0.13	[0.13,0.15]	13/0.3	13/0.3	13/0.4	13/0.3
	5000/1.32		14/0.3	14/0.3	14/0.4	16/0.3
	10000/2.64		16/0.3	16/0.3	16/0.4	16/0.3
LARGEREGFILE	5000/0.24	[0.42,0.45]	68/5.2	48/5.7	48/6.6	42/4.9
	10000/0.47		68/5.0	49/5.3	50/5.5	44/5.3
	50000/2.37		69/5.3	51/5.9	51/5.7	48/5.5
SLS	1750/0.10	[0.01,0.02]	149/12.1	118/9.7	118/9.8	114/9.1
	17500/1.00		125/10.2	101/8.4	101/8.6	103/7.9
	70000/4.00		124/10.3	100/8.4	100/8.6	101/7.9
TP-6	1400/0.14	[0.02,0.03]	20/1.7	17/1.6	17/1.7	16/1.6
	14000/1.38		21/1.8	18/1.7	18/1.8	19/1.7
	28000/2.76		21/1.8	18/1.7	18/1.7	19/1.8

Table 2: Effect of the rank of the update when adding equations or columns. k is the rank of the update in absolute number and percentage, ρ is the density range for all the preconditioners. * indicates total CPU time corresponding to the preconditioner computation and the iterative solution. A † means that the iterative method was unable to converge.

Matrix	k	ρ	non-prec Its./Time	non-updated Its./Time	recomputed Its./Time*	updated Its./Time*
PHOTOGRAMMETRY2	50/1.12	[0.03,0.05]	357/0.04	143/0.02	142/0.02	70/0.01
	100/2.24		461/0.05	182/0.03	191/0.03	70/0.01
	200/4.47		859/0.11	495/0.07	497/0.12	70/0.01
TESTBIG	300/0.96	[0.55,0.59]	142/0.1	112/0.1	101/0.2	56/0.1
	600/1.92		141/0.1	113/0.1	99/0.2	56/0.1
	1200/3.84		149/0.1	112/0.1	104/0.2	57/0.1
CAT_EARS_4_4	400/0.9	[0.78,0.84]	146/0.2	78/0.2	78/0.9	87/0.3
	1100/2.47		163/0.3	92/0.3	88/0.9	87/0.3
	2200/4.95		369/0.5	213/0.5	144/0.9	109/0.4
FOME13	200/0.20	[0.79]	460/1.5	276/1.5	272/1.9	230/1.3
	1000/1.02		512/1.8	306/1.6	296/2.0	239/1.4
	4000/4.08		712/3.8	612/1.9	604/2.4	237/1.3
LP_KEN_18	500/0.32	[0.77,0.81]	531/2.6	217/1.7	216/1.8	132/1.1
	1000/0.65		532/2.6	231/1.8	229/1.9	132/1.1
	5000/3.23		573/2.7	227/1.7	216/1.8	134/1.1
FLOWER_8_4	500/0.40	[0.80,0.86]	175/0.7	102/0.7	101/1.2	107/0.8
	1000/0.80		181/0.7	93/0.7	92/1.4	95/0.7
	6000/4.79		315/1.2	186/1.3	144/1.7	104/0.8
FXM3_16	100/0.12	[0.38,0.42]	2115/5.3	1200/4.7	668/3.1	462/1.9
	1000/1.17		2212/5.4	1974/7.2	2228/8.5	470/1.8
	4000/4.68		2670/6.3	2067/7.4	1592/6.7	473/1.8
LP_OSA_30	500/0.48	[0.01,0.02]	132/0.4	60/0.2	55/0.2	56/0.2
	1000/0.96		134/0.4	68/0.2	61/0.3	56/0.2
	5000/4.79		142/0.4	111/0.4	68/0.3	56/0.2
MESH_DEFORM	230/0.10	[0.09,0.19]	482/2.9	261/1.9	230/1.9	198/1.3
	2300/0.98		518/3.3	508/3.6	680/5.1	192/1.2
	9200/3.94		1554/9.1	2464/17.1	†	197/1.4
WATSON_1	500/0.13	[0.48,0.56]	1136/12.4	705/13.0	332/7.0	330/6.5
	5000/1.29		†	†	414/7.9	315/6.4
	15000/3.88		†	†	381/7.1	331/6.5
TS-PALKO	500/1.06	[0.03,0.04]	75/0.4	73/0.4	72/0.8	44/0.3
	1000/2.12		842/3.7	831/3.8	744/3.7	44/0.3
	2000/4.23		2665/10.6	2620/12.3	2346/11.8	44/0.3
LP_NUG30	500/0.13	[0.13,0.15]	20/0.3	32/0.4	18/0.7	17/0.3
	5000/1.32		40/0.6	57/0.9	22/0.8	18/0.3
	10000/2.64		62/1.1	91/1.3	27/1.0	19/0.3
LARGEREGFILE	5000/0.24	[0.44,0.48]	91/6.7	53/5.7	48/6.6	50/5.4
	10000/0.47		93/6.7	53/5.7	47/6.2	56/6.2
	50000/2.37		98/7.0	57/6.2	46/6.4	60/6.3
SLS	175/0.01		209/17.8	154/12.9	155/13.5	129/10.1
	17500/1.00		473/38.5	260/21.7	252/21.9	129/10.0
	70000/4.00		455/36.7	244/20.2	244/20.3	130/10.2
TP-6	1400/0.14	[0.02,0.03]	191/15.3	182/16.0	137/12.3	18/1.7
	14000/1.38		†	†	†	20/1.8
	28000/2.76		†	†	†	21/1.9

Table 3: Effect of the rank of the update when removing equations or columns. k is the rank of the update in absolute number and percentage, ρ is the density range for all the preconditioners. * indicates total CPU time corresponding to the preconditioner computation and the iterative solution. A † means that the iterative method was unable to converge.