Journal of Computational and Applied Mathematics 344 (2018) 346–355

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

An efficient LDU algorithm for the minimal least squares solution of linear systems

I. Fernández de Bustos [a], V. García-Marina [b,*], G. Urkullu [a], M. Abasolo [a]

[a] Department of Mechanical Engineering, University of the Basque Country, Faculty of Engineering of Bilbao, Alameda de Urquijo s/n, 48013, Bilbao, Spain

[b] Department of Mechanical Engineering, University of the Basque Country, University School of Engineering of Vitoria-Gasteiz , Nieves Cano 12, 01006, Vitoria-Gasteiz, Spain

ARTICLE INFO

Article history:
Received 19 September 2017
Received in revised form 5 March 2018

MSC:
65F05
65F20
1504
6504

Keywords:
Minimum linear least squares
LDU
Null subspaces

ABSTRACT

The minimal least squares solutions is a topic of interest due to the broad range of applications of this problem. Although it can be obtained from other algorithms, such as the Singular Value Decomposition (SVD) or the Complete Orthogonal Decomposition (COD), the use of LDU factorizations has its advantages, namely the computational cost and the low fill-in that can be obtained using this method. If the right and left null-subspaces (which can also be named as Null and Image subspaces, respectively) are to be obtained, the use of these factorizations leads to fundamental subspaces, which are sparse by definition. Here an algorithm that takes advantage of both the Peters–Wilkinson method and Sautter method is presented. This combination allows for a good performance in all cases. The method also optimizes memory use by storing the right null-subspace and the left null-subspace in the factored matrix.

1. Introduction

In this paper the minimal least squares problem known as $\min_x \|\mathbf{b} - A\mathbf{x}\|_2$ is considered, with $\|\mathbf{x}\|_2$ minimal. Although not quite popular, both Peters–Wilkinson [1] and Sautter [2] methods have their advantages when solving this minimal least squares solution of linear systems of equations. When compared to other general methods, such as Singular Value Decomposition (SVD) or Complete Orthogonal Decomposition (COD), they feature low memory consumption and high speed, while keeping reasonable accuracy. They also have advantages when applied to sparse matrices. The main drawback of these methods is the difficulty to develop block implementations, which can be faster when the matrix size is considerable. In this paper an algorithm combining Peters–Wilkinson and Sautter methods will be presented, which computes not only the minimal least squares solution, but also the right and left null-subspaces, in the form of a fundamental subspace, with no noticeable increase in the overall cost. This allows the method to be used as an efficient way for introducing linear restrictions in a system of equations, which can be of interest in short to medium scale optimization problems.

The Peters–Wilkinson method is a general way for the resolution of the minimal least squares problem which can be of use with any factorization method. It can be applied to trapezoidal factorizations such as LU o, even better, LDU, delivering greater precision in the last alternative [1]. The advantages of this particular implementation are low memory cost and high speed, along with a good behavior when applied to sparse matrices with small modifications (see Bjork, [3]). Some of the

* Corresponding author.
E-mail addresses: igor.fernandezdebustos@ehu.eus (I. Fernández de Bustos), vanessa.garcia@ehu.eus (V. García-Marina), gorka.urkullu@ehu.eus (G. Urkullu), mikel.abasolo@ehu.eus (M. Abasolo).

https://doi.org/10.1016/j.cam.2018.05.037

alternatives of this method are the SVD factorization, which should be the most accurate alternative, but at cumbersome cost both in terms of memory and computations, and the COD, which sacrifices some precision but delivers a very low memory consumption and cost when dealing with dense matrices. A great advantage of both of these algorithms is that blocked algorithms can easily be derived for them, which improves efficiency for large matrices. An additional advantage of the COD appears in the solution of weighted least squares problems (see [4]). For a good comparison of SVD and QR methods, along with others, (see [3] and [5]). If the problem includes sparse matrices, the Peters–Wilkinson method can be successfully applied just by taking into account the pivoting strategy, as shown for example in [3]. On the other hand, if the aim of the problem is to introduce linear restrictions in a system of equations for an optimization, where the system can be derived into an incompatible system for a particular linear approximation of the restrictions, both minimal least squares solution of the system of equations along with a fundamental expression of the right null-subspace can be of interest. Here an algorithm which combines a modified approach of Peters–Wilkinson and Sautter methods is presented. This algorithm allows one to obtain not only the minimal least squares solution, but also the right and left null-subspaces introducing no noticeable additional cost.

2. Formulation of the method

Although the presented algorithm can be demonstrated as a combination of the Peters–Wilkinson approach along with Sautter method, including some other algorithms, here a different approach will be used. In order to do so, first the least squares problem will be solved. This will lead to (in the general case) an underdetermined system of equations where the minimum norm solution will be obtained.

Let us consider the general linear system of equations factored using an LDU approach, together with pivoting strategies:

$$A\mathbf{x} = P_f^T LDUP_c\mathbf{x} = \mathbf{b} \tag{1}$$

where the relevant dimensions are the following:

$$A_{m\times n};\ \mathbf{x}_n;\ P_{f,m\times m};\ L_{m\times m};\ D_{m\times n};\ U_{n\times n};\ P_{c,n\times n};\ \mathbf{b}_m.$$

Let D, L, and U be the LDU factoring matrices defined above. Keeping generality, one can write:

$$D = \begin{bmatrix} D_1 & [0] \\ [0] & [0] \end{bmatrix} \quad ; \quad L = \begin{bmatrix} L_{11} & [0] \\ L_{21} & I \end{bmatrix} \quad ; \quad U = \begin{bmatrix} U_{11} & U_{12} \\ [0] & I \end{bmatrix} \tag{2}$$

where D_1, L_{11} and U_{11} are of dimensions $r \times r$. In the general case, the problem will be incompatible and, even if it is not, the least squares solutions can be obtained from:

$$A^T A\mathbf{x} = A^T\mathbf{b}. \tag{3}$$

Introducing the factorization:

$$P_c^T U^T D^T L^T P_f P_f^T LDUP_c\mathbf{x} = P_c^T U^T D^T L^T P_f\mathbf{b}. \tag{4}$$

Obviously, one can write:

$$D^T L^T LDUP_c\mathbf{x} = D^T L^T P_f\mathbf{b}. \tag{5}$$

As usual, the following change will be first introduced:

$$\mathbf{z} = DUP_c\mathbf{x} = \begin{Bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2. \end{Bmatrix} \tag{6}$$

So one needs to solve:

$$D^T L^T L\mathbf{z} = D^T L^T P_f\mathbf{b} = D^T L^T\mathbf{c}. \tag{7}$$

With the partition:

$$\begin{bmatrix} D_1^T & [0] \\ [0] & [0] \end{bmatrix}\begin{bmatrix} L_{11}^T & L_{21}^T \\ [0] & I \end{bmatrix}\begin{bmatrix} L_{11} & [0] \\ L_{21} & I \end{bmatrix}\begin{Bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{Bmatrix} = \begin{bmatrix} D_1^T & [0] \\ [0] & [0] \end{bmatrix}\begin{bmatrix} L_{11}^T & L_{21}^T \\ [0] & I \end{bmatrix}\begin{Bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2. \end{Bmatrix} \tag{8}$$

The last $m - r$ equations are identically zero. From here one can only obtain $\mathbf{z}_1$. It is possible to divide the problem into two parts: the first is related to the compatibility, while the second is related to underdetermination. First $\mathbf{z}_1$ will be obtained and afterwards, $\mathbf{x}$.

One can reach:

$$(L_{11}^T L_{11} + L_{21}^T L_{21})\mathbf{z}_1 = L_{11}^T\mathbf{c}_1 + L_{21}^T\mathbf{c}_2. \tag{9}$$

This equation can be used to obtain the least squares solution of any system of equations and, due to the fact that L is well conditioned, it is easy to demonstrate that $L_{11}^T L_{11} + L_{21}^T L_{21}$ is also well conditioned. Furthermore, it is also positive definite,

which allows one to use PCG or Cholesky methods to solve the $r \times r$ system. One can also modify the system in the following way:

$$(L_{11}^T)^{-1}(L_{11}^T L_{11} + L_{21}^T L_{21})L_{11}^{-1} L_{11}\mathbf{z}_1 = (L_{11}^T)^{-1}(L_{11}^T \mathbf{c}_1 + L_{21}^T \mathbf{c}_2). \tag{10}$$

And, thus:

$$(I + (L_{11}^T)^{-1} L_{21}^T L_{21} L_{11}^{-1})L_{11}\mathbf{z}_1 = \mathbf{c}_1 + (L_{11}^T)^{-1} L_{21}^T \mathbf{c}_2. \tag{11}$$

The interest of this equation is that the matrix $L_{21}L_{11}^{-1}$ is a representation of the left null-subspace of A, due to the fact that:

$$S^T = \begin{bmatrix} -L_{21}L_{11}^{-1} & I \end{bmatrix} = \begin{bmatrix} S_1^T & I \end{bmatrix}. \tag{12}$$

This is usually known as a fundamental representation of the left null-subspace of A. In order to reduce storage, one can store S_1 in the place of L_{21} (which is usually where A_{21} was stored). So the only increase in memory required is to store the $r \times r$ matrix $I + S_1 S_1^T$. Obviously, the dimensions of S^T are $(m - r) \times m$. This approach is quite useful when dealing with matrices of low rank, where the $r \times r$ resultant matrices are small, but if $r \simeq m$, this approach is not optimal. In this case one can take a totally different approach, leading to similar results. Let us consider the change:

$$\mathbf{u} = P_f A \mathbf{x}. \tag{13}$$

Going back to (3), one can write:

$$A^T P_f^T \mathbf{u} = A^T \mathbf{b} \tag{14}$$

$$P_c^T U^T D^T L^T P_f P_f^T \mathbf{u} = P_c^T U^T D^T L^T P_f \mathbf{b}. \tag{15}$$

And, thus:

$$D^T L^T \mathbf{u} = D^T L^T P_f \mathbf{b}. \tag{16}$$

Here, again, the last $n - r$ equations are identically null. In the general case, the system is underdetermined. To get a particular result, one can write:

$$\mathbf{u}_p = P_f \mathbf{b} = \mathbf{c}. \tag{17}$$

A general solution can be obtained using the Null-Subspace of A^T which is obviously the Image Subspace of A.

$$\mathbf{u} = \mathbf{u}_p + S\boldsymbol{\alpha}. \tag{18}$$

The interest lies in obtaining those u leading to a solution in the equation $A\mathbf{x} = \mathbf{u}$. This means that the left null-subspace of A multiplied by $\mathbf{u}$ equals $\mathbf{0}$. Thus:

$$S^T \mathbf{u} = \mathbf{0} \rightarrow S^T \mathbf{u}_p + S^T S\boldsymbol{\alpha} = \mathbf{0} \rightarrow S^T S\boldsymbol{\alpha} = -S^T \mathbf{c} \tag{19}$$

which, again, is a linear system with positive definite matrix, which can easily be solved using Cholesky or PCG methods. The advantage of this approach is that the matrix is of size $(m-r)\times(m-r)$ instead of $r \times r$. If one considers the fundamental left null-subspace of A,

$$S^T S = S_1^T S_1 + I. \tag{20}$$

One can write:

$$\mathbf{u} = \mathbf{c} - S(S_1^T S_1 + I)^{-1} S^T \mathbf{c}. \tag{21}$$

Once $\boldsymbol{\alpha}$ is obtained, one can get $\mathbf{u}$ from (18). Now, resorting to Eq. (13), one reaches:

$$P_f A \mathbf{x} = P_f P_f^T L D U P_c \mathbf{x} = \mathbf{u}. \tag{22}$$

And, from (6):

$$DUP_c \mathbf{x} = \mathbf{z} = L^{-1}\mathbf{u}. \tag{23}$$

Obviously, depending on the particular solution chosen, one gets a different $\mathbf{z}$. The relevant part is $\mathbf{z}_1$, which, as shown before, does not depend on $\mathbf{z}_2$, which must be equal to zero due to Eq. (23). So one can write:

$$\mathbf{z}_1 = L_{11}^{-1}\mathbf{u}_1. \tag{24}$$

And:

$$\mathbf{u}_1 = \mathbf{c}_1 - S_1(S_1^T S_1 + I)^{-1}(S_1^T \mathbf{c}_1 + \mathbf{c}_2). \tag{25}$$

Thus, there are two alternatives to obtain $\mathbf{z}_1$. Both can be obtained by using the fundamental left null-subspace. In both formulations, the operation delivering most cost is the Cholesky factorization, (or PCG resolution of the system) which is of dimension $r \times r$ in the first case and $(m-r) \times (m-r)$ in the second. Obviously, the best bet is to use the first algorithm if $r < (m-r) \rightarrow r < m/2$ and, in the other case, the second.

Once one has $\mathbf{z}_1$, it is time to get the minimal solution.

$$DUP_c\mathbf{x} = \mathbf{z} = \begin{Bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2. \end{Bmatrix} \tag{26}$$

For the sake of simplicity, one can write:

$$P_c\mathbf{x} = \mathbf{w}. \tag{27}$$

So:

$$DU\mathbf{w} = \mathbf{z}. \tag{28}$$

And:

$$\begin{bmatrix} D_1 & [0] \\ [0] & [0] \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ [0] & I \end{bmatrix} \begin{Bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{Bmatrix} = \begin{Bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2. \end{Bmatrix} \tag{29}$$

The similitude with Eq. (8) is evident. The last $m-r$ equations are irrelevant. As for the other, one can write:

$$\begin{bmatrix} U_{11} & U_{12} \end{bmatrix} = U_1. \tag{30}$$

So:

$$U_1\mathbf{w} = D_1^{-1}\mathbf{z}_1. \tag{31}$$

Using the usual minimum norm expression:

$$\mathbf{w} = U_1^T(U_1U_1^T)^{-1}D_1^{-1}\mathbf{z}_1. \tag{32}$$

And, thus:

$$\mathbf{w} = U_1^T[U_{11}U_{11}^T + U_{12}U_{12}^T]^{-1}D_1^{-1}\mathbf{z}_1 \tag{33}$$

One can also use a formulation based on a fundamental right null-subspace:

$$\mathbf{w} = U_1^T[U_{11}(I + U_{11}^{-1}U_{12}U_{12}^T(U_{11}^T)^{-1})U_{11}^T]^{-1}D_1^{-1}\mathbf{z}_1 \tag{34}$$

or:

$$\mathbf{w} = \begin{bmatrix} U_{11}^T \\ U_{12}^T \end{bmatrix}(U_{11}^T)^{-1}[I + U_{11}^{-1}U_{12}U_{12}^T(U_{11}^T)^{-1}]^{-1}U_{11}^{-1}D_1^{-1}\mathbf{z}_1. \tag{35}$$

The product $U_{11}^{-1}U_{12}$ can be stored in the place of U_{12} (which usually was the place where A_{12} was at the beginning), and a representation of the right null-subspace of A can be written as:

$$N = \begin{bmatrix} -U_{11}^{-1}U_{12} \\ I \end{bmatrix} = \begin{bmatrix} N_1 \\ I \end{bmatrix}. \tag{36}$$

And:

$$\mathbf{w} = \begin{bmatrix} I \\ -N_1^T \end{bmatrix}[I + N_1N_1^T]^{-1}U_{11}^{-1}D_1^{-1}\mathbf{z}_1. \tag{37}$$

One can use (33) or (37) and, in both cases, one needs to solve a rxr system of equations, which can be addressed by using Cholesky or PCG, due to the positive definiteness of both systems.

In the case of r being large, another approach is possible. One can write:

$$U_{11}\mathbf{w}_1 + U_{12}\mathbf{w}_2 = D_1^{-1}\mathbf{z}_1. \tag{38}$$

One can obtain a particular solution by assuming $\mathbf{w}_{p2} = \mathbf{0}$, thus,

$$\mathbf{w}_p = \begin{Bmatrix} U_{11}^{-1}D_1^{-1}\mathbf{z}_1 \\ \mathbf{0} \end{Bmatrix} = \begin{Bmatrix} \mathbf{w}_{p1} \\ \mathbf{0} \end{Bmatrix}. \tag{39}$$

A general solution can be obtained from:

$$\mathbf{w} = \mathbf{w}_p + N\boldsymbol{\beta}. \tag{40}$$

So:

$$\|\mathbf{w}\|^2 = (\mathbf{w}_p + N\boldsymbol{\beta})^T(\mathbf{w}_p + N\boldsymbol{\beta}) = \mathbf{w}_p^T\mathbf{w}_p + 2\mathbf{w}_p^T N\boldsymbol{\beta} + \boldsymbol{\beta}^T N^T N\boldsymbol{\beta}. \tag{41}$$

Introducing the minimal norm condition:

$$N^T N\boldsymbol{\beta} = -N^T\mathbf{w}_p. \tag{42}$$

Going back to (40), one gets $\mathbf{w}$ and, afterwards, using (27), $\mathbf{x}$. This alternative requires the resolution of a positive definite $(n-r)\times(n-r)$ system of equations. It is easy to find that this alternative is of interest when $r > n/2$. One can also write:

$$\mathbf{w} = \mathbf{w}_p + N(N_1^T N_1 + I)^{-1}(-N_1^T)\mathbf{w}_{p1}. \tag{43}$$

With:

$$\mathbf{w}_2 = (N_1^T N_1 + I)^{-1}(-N_1^T)\mathbf{w}_{p1}. \tag{44}$$

And, finally:

$$\mathbf{w}_1 = \mathbf{w}_{p1} + N_1\mathbf{w}_2. \tag{45}$$

3. Proposed algorithm

The aim is to reduce memory storage as much as possible. To solve the problem, the following steps are followed:

Step 1. perform LDU factorization, with complete or Rook pivoting strategies (rigorous partial pivoting can lead to failure of the algorithm).

$$A, \mathbf{f}, \mathbf{c} \leftarrow P_f^T LDUP_c \tag{46}$$

being f and c the pivot vectors, which can be reserved at the beginning with a size equal to the minimum between m and n.

Step 2. Compute the right and left null-subspaces:

$$A_{12} \leftarrow U_{11}^{-1}A_{12} \tag{47}$$
$$A_{21} \leftarrow A_{21}L_{11}^{-1} \tag{48}$$
$$\mathbf{b} \leftarrow P_f\mathbf{b}. \tag{49}$$

One should take into account the fact that L_{11}, D_{11}, and U_{11} are usually stored in A_{11}.

Step 3. Knowing the rank (derived at a marginal cost from D):

3.1. If $r \le m/2$:

$$\mathbf{b}_1 \leftarrow \mathbf{b}_1 - S_1\mathbf{b}_2 = \mathbf{b}_1 + A_{21}\mathbf{b}_2. \tag{50}$$
$$\mathbf{b}_1 \leftarrow (I + S_1S_1^T)^{-1}\mathbf{b}_1 = (I + A_{21}^TA_{21})^{-1}\mathbf{b}_1 \tag{51}$$

3.2. If $r > m/2$:

$$\mathbf{b}_2 \leftarrow -S_1^T\mathbf{b}_1 - \mathbf{b}_2 = A_{21}\mathbf{b}_1 - \mathbf{b}_2 \tag{52}$$
$$\mathbf{b}_2 \leftarrow (S_1^TS_1 + I)^{-1}\mathbf{b}_2 = (A_{21}A_{21}^T + I)^{-1}\mathbf{b}_2 \tag{53}$$
$$\mathbf{b}_1 \leftarrow \mathbf{b}_1 + S_1\mathbf{b}_2 = \mathbf{b}_1 - A_{21}^T\mathbf{b}_2 \tag{54}$$

Note:. After either Eqs. (50)–(51) or (52)–(54):

$$\mathbf{b}_1 \leftarrow L_{11}^{-1}\mathbf{b}_1 \tag{55}$$
$$\mathbf{b}_1 \leftarrow D_{11}^{-1}\mathbf{b}_1 \tag{56}$$
$$\mathbf{b}_1 \leftarrow U_{11}^{-1}\mathbf{b}_1. \tag{57}$$

Step 4. Deal with the least norm problem.

4.1. If $r \le n/2$, it is better to use:

$$\mathbf{b}_1 \leftarrow [I + N_1N_1^T]^{-1}\mathbf{b}_1 = [I + A_{12}A_{12}^T]^{-1}\mathbf{b}_1 \tag{58}$$
$$\mathbf{b}_2 \leftarrow -N_1^T\mathbf{b}_1 = A_{12}^T\mathbf{b}_1. \tag{59}$$

4.2. If $r > n/2$, one should use:

$$\mathbf{b}_2 \leftarrow -N_1^T\mathbf{b}_1 = A_{12}^T\mathbf{b}_1 \tag{60}$$

$$\mathbf{b}_2 \leftarrow [N_1^TN_1 + I]^{-1}\mathbf{b}_2 = [A_{12}^TA_{12} + I]^{-1}\mathbf{b}_2 \tag{61}$$

$$\mathbf{b}_1 \leftarrow \mathbf{b}_1 + N_1\mathbf{b}_2 = \mathbf{b}_1 - A_{12}\mathbf{b}_2. \tag{62}$$

Step 5. Permute elements:

$$\mathbf{b} \leftarrow P_c^T\mathbf{b} \tag{63}$$

Note:. It is easy to generalize this algorithm for the case of matrix problems, in the form:

$$AX = B. \tag{64}$$

4. Relation with the Peters–Wilkinson and Sautter methods

Equations used when $r < m/2$ (9) and when $r < n/2$ (33) are exactly the same when using Peters–Wilkinson approach, but the use of the variants (11) and (37) allows one to obtain at no storage cost and at a very little computational cost expressions for the right and left null-subspaces, which can be stored in the same space used by the original factorization. The Sautter approach was derived to obtain the least squares solution when using LU factorization. Here it is developed for LDU factorization, which is better suited for the minimal least squares solution. Also, an easy way to generalize it to the minimal norm solution is shown.

5. Theoretical performance

The maximum possible cost of the algorithm will be considered, which corresponds to that of a full pivoting situation. Due to the fact that currently the cost of operations on modern computers is comparable to the cost of memory fetches, the calculation of a value for the computational cost is not quite easy. Also, one has to have in mind that triangular factorizations shine specially in the case of sparse matrices, where the cost is heavily dependent on the percentage of zeros and their position. Thus, only the dense case is considered, and only those operations which yield a cost of order 3 will be pondered. Comparisons used for pivoting will be taken into account as the other flops.

Lemma. *The maximum flop cost of the algorithm can be expressed as follows:*

$$O\left(\frac{m^3}{12} + \frac{n^3}{12} - r^3 + 3mnr - \frac{(m+n)r^2}{2}\right).$$

Proof. The different steps considered in the algorithm described in Section 3 will be dealt with.

The cost of *Step 1* for the factorization with full pivoting leads to a cost of:

$$O_1 = O\left(3mnr - \frac{3}{2}(m+n)r^2 + r^3\right). \tag{65}$$

The cost of *Step 2* for the computation of the right and left null-subspaces takes, respectively, $O(r^2(m-r))$ flops and $O(r^2(n-r))$ flops, which gives an overall cost for the computation of these subspaces of:

$$O_2 = O\left(r^2(m+n-2r)\right). \tag{66}$$

As for the cost in *Step 3*, it has to be said that in Eqs. (50)–(56), the only operations of relevant cost are (51) and (53) for $r \leq \frac{m}{2}$ or $r > \frac{m}{2}$, respectively. Using standard algorithms the cost of computing $(I + S_1S_1^T)$ is $O\left(\frac{r^2(m-r)}{2}\right)$ flops.

Furthermore, using Cholesky for the resolution yields an additional $O\left(\frac{r^3}{6}\right)$, which leads to a total cost of $O\left(\frac{r^2m}{2} - \frac{r^3}{3}\right)$.

The alternative algorithm presented in this paper yields a total cost of $O\left(\frac{(m-r)^2r}{2} + \frac{(m-r)^3}{3}\right)$. Taking into account the two algorithms, the maximum cost will be when $r = \frac{m}{2}$, that is the case (***3.1***). It is easy to find that, under these conditions, the total cost is:

$$O_{3max} = O\left(\frac{m^3}{12}\right). \tag{67}$$

For the cost in *Step 4*, using the same considerations as above, one obtains, for the minimum norm stage, a maximum cost of:

$$O_{4max} = O\left(\frac{n^3}{12}\right). \tag{68}$$

Table 1
Comparison between different algorithms in relation to time and error.

Method	LDU Rook		SVD (dgesvd)	
Matrix size	Time	Error	Time	Error
64	$1.985E^{-04}$	$1.619E^{-12}$	$2.794E^{-03}$	$4.619E^{-13}$
128	$9.011E^{-04}$	$5.619E^{-10}$	$1.274E^{-02}$	$1.886E^{-10}$
256	$6.603E^{-03}$	$1.138E^{-10}$	$1.070E^{-01}$	$2.572E^{-11}$
512	$5.010E^{-02}$	$1.224E^{-09}$	$9.218E^{-01}$	$1.847E^{-10}$
1024	$4.072E^{-01}$	$4.446E^{-08}$	$7.477E^{+00}$	$2.798E^{-09}$
2048	$4.634E^{+00}$	$2.040E^{-07}$	$9.170E^{+01}$	$8.972E^{-09}$
4096	$4.183E^{+01}$	$2.758E^{-06}$	$7.296E^{+02}$	$8.094E^{-08}$
Method	SVD (dgelsd)		COD (dgelsy)	
Matrix size	Time	Error	Time	Error
64	$1.507E^{-03}$	$2.958E^{-12}$	$3.684E^{-04}$	$1.514E^{-12}$
128	$4.999E^{-03}$	$1.217E^{-09}$	$1.936E^{-03}$	$8.034E^{-10}$
256	$2.597E^{-02}$	$2.633E^{-11}$	$1.258E^{-02}$	$2.678E^{-10}$
512	$1.505E^{-01}$	$1.541E^{-10}$	$9.395E^{-02}$	$9.828E^{-10}$
1024	$1.271E^{+00}$	$2.795E^{-09}$	$6.706E^{-01}$	$3.858E^{-08}$
2048	$1.097E^{+01}$	$8.989E^{-09}$	$7.097E^{+00}$	$2.512E^{-07}$
4096	$8.448E^{+01}$	$8.086E^{-08}$	$5.644E^{+01}$	$2.392E^{-05}$

The total cost:

$$\begin{aligned} O = O_1 + O_2 + O_3 + O_4 &= O\left(3mnr - \frac{3}{2}(m+n)r^2 + r^3 + r^2(m+n-2r) + \frac{m^3}{12} + \frac{n^3}{12}\right) = \\ &= O\left(\frac{m^3}{12} + \frac{n^3}{12} - r^3 + 3mnr - \frac{(m+n)r^2}{2}\right). \end{aligned} \tag{69}$$

6. Experimental results

In order to get an idea of the performance of the algorithm, it has been compared with three methods, the first is a complete SVD factorization (using dgesvd), the second uses the same factorization, but saves time by discarding the computation of the orthogonal matrices (using dgelsd). The last method is the Complete Orthogonal Decomposition (using dgelsy). In this case, it is important to notice that this is a blocked implementation, which should deliver a great performance. For all methods the ATLAS implementation has been used, using full recompilation in the test platforms to get the most out of it.

First rectangular diagonal matrices of a given rank and dimensions are generated. The elements in the diagonal are randomly generated and, afterwards, a vector of independent values is also randomly generated. The solution for this system is calculated in this form.

Afterwards, two random orthogonal matrices are built. In order to build each one, the Stewart method [6] is used. The Mersenne Twister random number generator presented in [7] along with the basic form of the Box–Mueller transformation (see [8] and [9]) was employed to generate the random numbers with a Gaussian distribution. The first orthogonal matrix is left multiplied by the diagonal matrix and the right side vector and the second one is right multiplied by the matrix and its transpose left multiplied by the computed solution. This way a pseudo-random matrix with known rank is generated and a solution for a randomly built independent vector with small error is obtained.

It is quite difficult to evaluate the performance of the algorithm, due to the fact that there are many parameters involved. To get a reasonable idea of the performance, first is chosen a situation where $m = n$, $r = m/2$ and there are $r/2$ incompatibilities in the problem. Obviously, depending on the employed hardware, the results can noticeably vary.

The first experiments have been carried out in a XEON E5-2647 v3 (Haswell architecture), and are exposed in Table 1. In these experiments, the CPU has been forced to run at 3.5 GHz. A single processor has been used, but including SIMD extensions. The OS is Ubuntu Linux 16.10, and ATLAS implementation for the BLAS and LAPACK routines have been employed, recompiling the libraries with gcc, using options "-march = haswell - O3" to take the most out of the processor. Results are averaged on 5 runs.

These results translate in the following diagrams of time and precision versus size, both represented in logarithmic scale. See Figs. 1 and 2.

In terms of brute performance, the winner is the here presented algorithm (LDU Rook), followed by the COD. Nevertheless, in terms of Flops/sec throughput, one can easily find that the LDU decomposition has quite a room for improvement. This throughput is crippled mainly because of the non blocked nature of the decomposition. The use of a blocked version could noticeably improve the performance, but as it is known, it is not easy to develop stable triangular decompositions (see, for example, [10] and [11]), although not impossible. Another improvement could be derived from the use of modern approaches

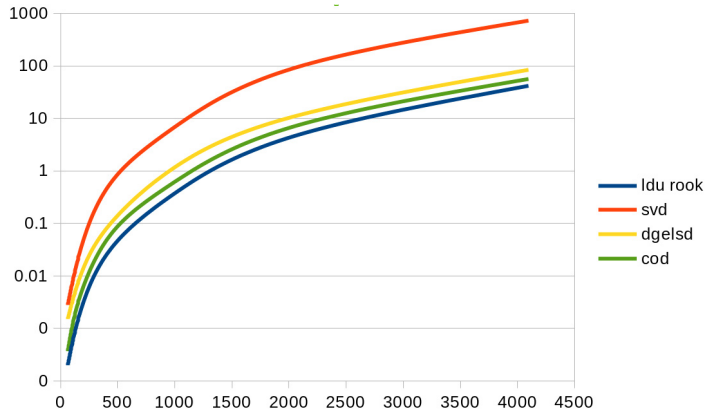

Fig. 1. Necessary time to yield the solution for different sizes of systems.

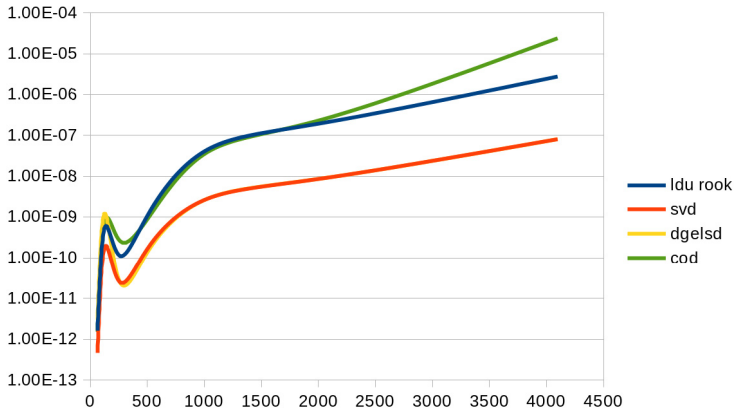

Fig. 2. Precision obtained in the solution for different sizes of systems.

for the storage of the symmetric matrices that appear in the algorithm. In the presented examples, the usual packed storage has been used, but, for example, the RFPF presented in [12] could make a moderate improvement in the throughput.

Speaking of error, the algorithms perform as expected. First of all, one must point out that there is no way for the curve in Fig. 2 to be used as a reference of the evolution of the error with the size of the matrices. This is due to the fact that only 5 experiments have been performed. Thus, the conditioning of the matrix is quite random, and the error in the methods is increased as the conditioning worsens. But it is easy to see a direct relation among the errors for the LDU decomposition with the presented algorithm, the SVD decomposition and the dgelsd algorithm which also uses a SVD algorithm. The Singular Value Decomposition is the most precise of them, and both in dgesvd and dgelsd forms, with, as expected, quite similar figures. Both LDU decomposition and Complete Orthogonal Decomposition deliver similar precision figures. It must be pointed out that the chart represented in Fig. 2 has been developed using only five experiments per size. This implies that the precision results are quite influenced by the conditioning of the resulting matrices in each experiment. This explains the strange behavior of the chart, with lower precision in smaller sized matrices in the beginning. Another unexpected result is in the 4096 sized experiments, where the COD leads to surprisingly noticeable bigger errors than the LDU.

To sort out these phenomena, an additional set of experiments has been developed. In this case, a group of problems with given estimated condition numbers have been generated. In order to do so, the same procedures have been followed, but now the diagonal elements have been generated in a different way. A maximum and a minimum positive value are introduced and, afterwards, their logarithms are taken. Then, a set of r random numbers with uniform distribution along these values is generated. The values in the diagonal are these numbers elevated to the power of 10, with a random sign. Finally, the maximum and minimum values are introduced with random signs and stored in a random situation, overwriting the previous ones. This assures an estimated condition number equal to the maximum value divided by the minimum value. A matrix size of 1024×1024, with rank 512 and 256 incompatible equations has been chosen.

The errors of LDU, SVD and its fast variant are presented in Fig. 3. They follow a reasonable pattern but the COD again shows a strange behavior, although in this set of experiments an explanation can be obtained. If one takes a look at the individual results, one can see that in most tests, the COD shows a behavior slightly worse, but very similar to that of the presented algorithm; in contrast, in some tests, the COD delivers a much greater error, of about 1 or 2 orders of magnitude.

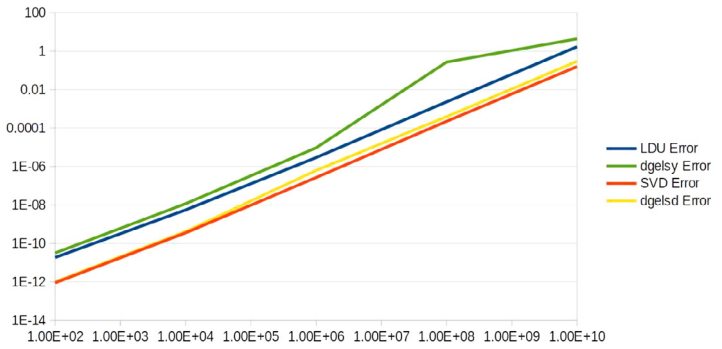

Fig. 3. Precision obtained in the solution.

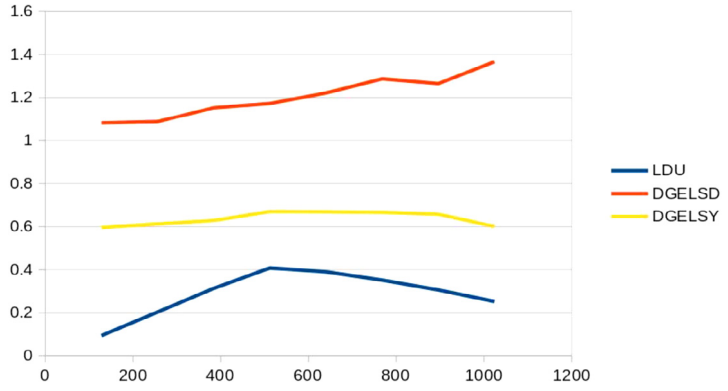

Fig. 4. Necessary time to yield the solution.

The effect of these particular cases is quite severe in the averaged values. The reason behind this problem could well be some optimization performed in the implementation algorithm, and should be studied, but it is out of the scope of this work.

Now the focus is set in the performance related to the matrix rank. The conditioning is fixed to an estimate value of 10 000 (minimum nonzero singular value of 0.01 and maximum of 100). The matrices are of size 1024×1024 and the number of incompatibilities, which should not affect performance, is set to half the dimension minus the rank.

Fig. 4 shows total time in sec. The LDU is the fastest in all situations. As expected, the worst situation for the LDU is at a rank of 512, where the algorithm changes from the algorithm derived from Peters–Wilkinson to that derived from Sautter. In this point, the algorithm is 1.64 times faster than the dgelsy algorithm (COD).

7. Conclusions

The use of LDU factorizations together with Rook pivoting along with a combination of the Peters–Wilkinson method and the Sautter method can lead to an approach to obtain the minimal least squares solution with quite a considerable performance, while keeping the precision of the calculation. This method has shown to be quite efficient in medium sized matrices, although it looses part of its advantages in speed when applied to very large matrices, due to data starvation. The precision figures are similar to those obtained by the COD algorithm, although it can be preferable in the cases of weighted least squares problems, as exposed in [4]. In order to improve speed, there are some tweaks that can directly be applied to the algorithm as, for example, using Rectangular Packed Format for the symmetric matrices, or using an iterative solver instead of Cholesky. But to get a considerable improvement one should solve the problem of stability for blocked LDU factorizations. Some authors have already performed some improvements in that sense, so this could be achievable in the mid term. In terms of memory, the algorithm is also quite efficient, and can also take advantage of sparsity. An additional advantage of this method is that it includes, at nearly no computational or memory cost, the computation of the fundamental right and left null-subspaces. This representation allows one to make fast computations with them.

Acknowledgments

The authors wish to thank the Spanish Ministry of Economy and Competitiveness for its support through grant DPI2016-80372-R, which also includes funding through European FEDER program; and the Education Department of the Basque Government for its support through grant IT947-16.

References

[1] G. Peters, J.H. Wilkinson, The least squares problem and pseudo-inverses, Comput. J. 13 (3) (1970) 309–316.
[2] W. Sautter, Fehleranalyse für die Gauß-Elimination zur Berechnung der Lösung minimaler Länge, Numer. Math. 30 (2) (1978) 165–184.
[3] Å. Björck, Numerical Methods for Least Squares Problems, SIAM, 1996.
[4] P.D. Hough, S.A. Vavasis, Complete orthogonal decomposition for weighted least squares, SIAM J. Matrix Anal. Appl. 18 (2) (1997) 369–392.
[5] G. Golub, C. Van Loan, Matrix Computations, fourth ed., Johns Hopkins, 2013.
[6] G.W. Stewart, The efficient generation of random orthogonal matrices with an application to condition estimators, SIAM J. Numer. Anal. 17 (3) (1980) 403–409.
[7] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. 8 (1) (1998) 3–30.
[8] G.E. Box, M.E. Muller, et al., A note on the generation of random normal deviates, Ann. Math. Stat. 29 (2) (1958) 610–611.
[9] M. MacDougall, Simulating computer systems: techniques and tools, 1987.
[10] A. Khabou, J.W. Demmel, L. Grigori, M. Gu, Lu factorization with panel rank revealing pivoting and its communication avoiding version, SIAM J. Matrix Anal. Appl. 34 (3) (2013) 1401–1429.
[11] J.W. Demmel, N.J. Higham, R.S. Schreiber, Stability of block LU factorization, Numer. Linear Algebra Appl. 2 (2) (1995) 173–190.
[12] F.G. Gustavson, J. Waśniewski, J.J. Dongarra, J. Langou, Rectangular full packed format for cholesky's algorithm: factorization, solution, and inversion, ACM Trans. Math. Softw. 37 (2) (2010) 18.