Sparse stretching for solving sparse-dense linear least-squares problems

SPARSE STRETCHING FOR SOLVING SPARSE-DENSE LINEAR LEAST-SQUARES PROBLEMS

JENNIFER SCOTT* AND MIROSLAV TŮMA†

Abstract. Large-scale linear least-squares problems arise in a wide range of practical applications. In some cases, the system matrix contains a small number of dense rows. These make the problem significantly harder to solve because their presence limits the direct applicability of sparse matrix techniques. In particular, the normal matrix is (close to) dense, so that forming it is impractical. One way to help overcome the dense row problem is to employ matrix stretching. Stretching is a sparse matrix technique that improves sparsity by making the least-squares problem larger. We show that standard stretching can still result in the normal matrix for the stretched problem having an unacceptably large amount of fill. This motivates us to propose a new sparse stretching strategy that performs the stretching so as to limit the fill in the normal matrix and its Cholesky factor. Numerical examples from real problems are used to illustrate the potential gains.

Key words. sparse matrices, linear least-squares problems, dense rows, matrix stretching, Cholesky factorization, normal matrix.

1. Introduction. Large-scale linear least-squares (LS) problems occur in a wide variety of practical applications, both in their own right and as subproblems of nonlinear least-squares problems. Our interest lies in solving the real $m \times n$ ($m > n$) mixed sparse-dense LS problem

$$\min_x \|Ax - b\|_2 = \min_x \left\| \begin{pmatrix} A_s \\ A_d \end{pmatrix} x - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2, \tag{1.1}$$

in which each row of the $p \times n$ block A_d is considered to be dense and A_s is $m_s \times n$ with $m_s \gg p$ ($m = m_s + p$); the vector b is partitioned conformally. The presence of dense rows causes the normal matrix $C = A^T A$ to be very (or even completely) dense, and this greatly limits the effectiveness of the straightforward application of sparse matrix techniques for solving (1.1). A number of authors in the 1980s and 1990s studied direct methods for tackling this problem, including George and Heath [18], Heath [21], Björck [7, 8], and Sun [33, 34]. More recently, preconditioning strategies for (1.1) have been proposed. A simple approach is that of Avron, Ng and Toledo [5] in which LSQR [27] is preconditioned by the complete QR factorization of A_s (A_d is dropped from the factorization) while Scott and Tůma [32] explore transforming the problem and using a reduced augmented form with either direct or iterative solvers. In a separate paper, Scott and Tůma [31] propose processing the rows of A_d separately within a conjugate gradient method using an incomplete factorization preconditioner combined with the factorization of a dense matrix of size equal to the number of dense rows.

Here, we revisit the idea of matrix stretching for handling dense rows. Stretching aims to split each of the rows of A_d into several sparser parts and to formulate a (larger) modified problem that provides the solution to the original one. The approach was introduced in 1990 by Grcar [20], who proposed both row and column stretching as an effective way of treating sparse matrices with dense rows or columns before performing an LU factorization. The technique was subsequently used for solving linear systems by Alvarado [3], Ferris and Horn [17], Aykannat, Pinar and Çatalyürek [6], and Duff and Scott [15]. For LS problems, a theoretical analysis of the stretching of dense rows was presented in 2000 by Adlers and Björck [1, 2], who proposed row splitting as an alternative to updating methods and presented some preliminary numerical results illustrating the potential of the approach.

A key issue with matrix stretching is deciding how to choose the number of parts the dense rows should be split into. While a small number can significantly improve sparsity of the normal matrix, increasing this number can adversely effect the condition of the problem. Moreover, the fill in the Cholesky factor of the

*STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, Oxfordshire, OX11 0QX, UK and School of Mathematical, Physical and Computational Sciences, University of Reading, Reading RG6 6AQ, UK. Correspondence to: jennifer.scott@stfc.ac.uk. Partially supported by EPSRC grant EP/M025179/1.

†Department of Numerical Mathematics, Faculty of Mathematics and Physics, Charles University, Czech Republic, (mirektuma@karlin.mff.cuni.cz.) Supported by the projects 17-04150J and 18-12719S of the Grant Agency of the Czech Republic.

normal matrix for the stretched problem can still be unacceptably large, limiting the suitability of sparse direct solvers and leading to poor quality incomplete factorization preconditioners. To overcome this, we introduce a new idea of *sparse stretching*. Sparse stretching looks at the interaction between the stretched rows and the non-dense rows A_s when deciding how to split the dense rows. This leads to the need to solve an auxiliary combinatorial problem. In practice, this combinatorial problem is solved approximately and its solution determines the number of parts to split the dense rows into as well as how to split the dense rows.

The rest of the paper is organised as follows. In Section 2, we recall stretching of dense rows and present a result on the condition of the stretched problem. We also look at the structural properties of the stretched problem for three simple test cases with a single dense row. These highlight the potential weaknesses of the existing stretching strategy. In Section 3, we introduce sparse stretching and present our algorithm for determining how to split dense rows so as to limit fill in the resulting stretched normal matrix. Numerical results are presented in Section 4 and concluding remarks and possible future directions are discussed in Section 5.

2. Matrix stretching for LS problems.

2.1. Standard stretching. We start by briefly recalling the simplest case of matrix stretching for LS problems based on binary splitting. The notation used here is based on that of Adlers and Björck [2]. We initially assume that A_d represents a single dense row, which we denote by f^T. The stretching procedure has two steps. In the first, a larger problem is constructed by splitting the dense row f^T into two parts $f^T = (f_a^T \;\; f_b^T)$ and introducing a linking variable $s \in \mathbb{R}$. Let us split the sparse block A_s and the solution vector x as $A_s = (A_{sa} \;\; A_{sb})$, $x = (x_a^T \;\; x_b^T)^T$ to conform with the splitting of f^T. It is straightforward to observe [2] that the component x of the solution $(x^T \;\; s)^T$ of the extended LS problem

$$\min_{(x^T \;\; s)^T} \left\| \begin{pmatrix} A_{sa} & A_{sb} & 0 \\ f_a^T & f_b^T & 0 \\ f_a^T & -f_b^T & \sqrt{2} \end{pmatrix} \begin{pmatrix} x_a \\ x_b \\ s \end{pmatrix} - \begin{pmatrix} b_s \\ b_d \\ 0 \end{pmatrix} \right\|_2 \tag{2.1}$$

is the same as the solution x of the original problem (1.1). The second step applies an orthogonal transformation to the extended system matrix in (2.1) to replace f_b^T in the second block row and f_a^T in the third block row by zeros (see [2] for details). Orthogonal invariance of the norm leads to the equivalent stretched problem

$$\min_z \|\hat{A}z - \hat{b}\|_2$$

with

$$\hat{A} = \begin{pmatrix} A_{sa} & A_{sb} & 0 \\ \sqrt{2}\, f_a^T & 0 & 1 \\ 0 & \sqrt{2}\, f_b^T & -1 \end{pmatrix}, \qquad z = \begin{pmatrix} x_a \\ x_b \\ s \end{pmatrix}, \qquad \hat{b} = \begin{pmatrix} b_s \\ b_d/\sqrt{2} \\ b_d/\sqrt{2} \end{pmatrix}.$$

Now consider splitting f^T into $k \geq 2$ parts. Generalising the above gives

$$\hat{A} = \begin{pmatrix} A_s & \\ F^T & \gamma S \end{pmatrix}, \quad A_d \equiv f^T = \frac{1}{\sqrt{k}}\, e^T F^T, \quad F^T \in R^{k \times n}, \quad z = \begin{pmatrix} x \\ s \end{pmatrix}, \quad \hat{b} = \begin{pmatrix} b_s \\ b_d e/\sqrt{k} \end{pmatrix}, \tag{2.2}$$

where $e \in R^k$ is the vector of ones. Here the *linking matrix* S is given by

$$S = \begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & -1 & \ddots & \\ & & \ddots & 1 \\ & & & -1 \end{pmatrix} \in R^{k \times (k-1)}$$

and $s \in R^{k-1}$ is the *linking vector* [2, 35]. The parameter γ offers additional scaling that is applied to the linking vector and can be chosen to improve the condition estimate (see below).

Generalising the splitting further, let $A_d = \begin{pmatrix} f_1, \ldots, f_p \end{pmatrix}^T$ have $p \geq 1$ rows. Let each dense row f_i^T be stretched into a matrix F_i^T with $k_i > 1$ rows and let the right-hand side vector $b_d = \begin{pmatrix} b_1, \ldots, b_p \end{pmatrix}^T \in R^{p \times n}$ be transformed into $\hat{b}$. The matrix $\hat{A}$ can be written as in (2.2) with S replaced by $\hat{S} = \mathrm{diag}(S_1, \ldots, S_p)$ and $F = \begin{pmatrix} F_1 \ldots F_p \end{pmatrix}$ where

$$f_i^T = \frac{1}{\sqrt{k_i}}\, e_i^T F_i^T, \quad F_i^T \in R^{k_i \times n}, \quad z = \begin{pmatrix} x \\ s \end{pmatrix}, \quad s = \begin{pmatrix} s_1 \\ \vdots \\ s_p \end{pmatrix}, \quad \hat{b}^T = \begin{pmatrix} b_s^T, \, \frac{b_1}{\sqrt{k_1}}\, e_1^T & \cdots & \frac{b_p}{\sqrt{k_p}}\, e_p^T \end{pmatrix},$$

with $S_i \in R^{k_i \times (k_i - 1)}$, $s_i \in R^{k_i - 1}$, $e_i \in R^{k_i}$.

We refer to this matrix stretching for LS problems as *standard stretching*. In particular, we assume that standard stretching splits the row indices of the entries in the dense rows into sets of equal (or almost equal) contiguous segments and then bases the stretching on these sets.

2.2. Condition of the stretched system. Adlers and Björck [2] showed that the stretched LS problem has the same set of solutions x as the original problem. In addition, they analysed the condition of the stretched problem in the case that A is of full rank. Since their derivation of an upper bound on the condition number of $\hat{A}$ contains a small error, we state a slightly modified conditioning result but omit full details of the proof. In particular, in deriving an upper bound on the norm of $(S^T S)^{-1} S^T$, Adlers and Björck use the inequality $\sin x \geq x, \ x \geq 0$ but this inequality is incorrect. Instead, the following relation may be used.

LEMMA 2.1. *For all $x \geq 0$, $\sin x > x - x^3/3$.*

Proof. Setting $g(x) = x - x^3/3 - \sin x$, we can see that $g(0) = 0$, $g'(x) = 1 - x^2 - \cos x = 2\sin^2(x/2) - x^2 < 2x^2/4 - x^2 = -x^2/2 < 0$. Consequently, g is decreasing for $x \geq 0$ and the result follows. ∎

Based on the previous lemma, we state another simple relation as follows.

LEMMA 2.2. *For $k \geq 2$, $\sin \pi/(2k) > 1/k$.*

Proof. From Lemma 2.1 it follows that $\sin(\pi/(2k)) > \pi/(2k) - \pi^3/(24k) > 1/k(\pi/2 - \pi^2/96) > 1/k$. ∎

Using this and the identity

$$\|S^\dagger\|_2^2 = \|(S^T S)^{-1} S^T\|_2^2 = \|((S^T S)^{-1}) S^T ((S^T S)^{-1} S^T)^T\|_2 = \|(S^T S)^{-T} S^T S (S^T S)^{-1}\|_2 = \|(S^T S)^{-1}\|_2,$$

we obtain a bound for the pseudoinverse

$$\|S^\dagger\|_2^2 = 1/2(1 - \cos(\pi/k))^{-1} = 1/4 \sin^{-2}(\pi/(2k)) < k^2/4. \tag{2.3}$$

We now state a bound on the condition of the stretched system. Let $k = \max\{k_i, 1 \leq i \leq p\}$. Let $\hat{C} = \hat{A}^T \hat{A}$ be the *stretched normal matrix*. Assume the singular values $\sigma_i(A)$ of A and the eigenvalues $\lambda_i(\hat{C})$ of $\hat{C}$ are sorted in descending order of their absolute values. Then we have the following result.

THEOREM 2.3. *(modified from [2]) If the stretching parameter γ is given by $1/2\sqrt{p\,k}\|A_d\|_2$ then the largest eigenvalue $\lambda_1(\hat{C})$ satisfies*

$$\lambda_n(\hat{C})^{-1} \leq \sigma_n^2(A)^{-1} \left(\frac{\|S^\dagger\|_2^2}{\gamma^2} p\,k \|A_d\|_2^2 + \frac{\|S^\dagger\|_2^2}{\gamma^2} \sigma_n^2(A) + \frac{\|S^\dagger\|_2 \sqrt{p\,k}}{\gamma} \|A_d\|_2 \right). \tag{2.4}$$

Furthermore,

$$\lambda_1(\hat{C}) \leq \left(\sigma_1^2(A) + p\,k\,\|A_d\|_2^2 + 2\gamma\sqrt{p\,k}\|A_d\|_2 \right) = \left(\sigma_1^2(A) + 2p\,k\,\|A_d\|_2^2 \right).$$

Substituting the bound (2.3) for $\|S^\dagger\|_2$ into (2.4) yields the following result.

THEOREM 2.4. *An upper bound for the condition number of the stretched matrix $\hat{A}$ with $\gamma = 1/2\sqrt{p\,k}\|A_d\|_2$ is*

$$\kappa^2(\hat{A}) \leq \kappa^2(A)k\left(1+\frac{2\,p\,k\|A_d\|_2^2}{\|A\|_2^2}\right)\left(k+1+\frac{\sigma_n(A)^2}{\|A_d\|_2^2}\right).$$

2.3. Structural properties of the stretched system. In this section, we consider structural effects caused by standard stretching. We focus on three simple cases: A_s diagonal, A_s tridiagonal and A_s with a simple Laplacian structure based on the five-point two-dimensional stencil. We demonstrate that the stretched rows can adversely influence the density of the stretched normal matrix and of its factors, even if the number of stretched parts is small. This will help understand the experimental results in Section 4 for practical examples and motivates the sparse stretching that we propose in Section 3. With the stretched matrix given by

$$\hat{A} = \begin{pmatrix} A_s \\ F^T & \gamma S \end{pmatrix},$$

the stretched normal matrix is

$$\hat{C} = \hat{A}^T\hat{A} = \begin{pmatrix} A_s^T & F \\ & \gamma S^T \end{pmatrix}\begin{pmatrix} A_s \\ F^T & \gamma S \end{pmatrix} = \begin{pmatrix} A_s^TA_s + FF^T & \gamma FS \\ \gamma S^TF^T & \gamma^2 S^TS \end{pmatrix}. \tag{2.5}$$

The number of entries in $\hat{C}$ is denoted by $nz(\hat{C})$.

2.3.1. A_s diagonal. We start by considering A with a single dense row and A_s diagonal. The sparsity pattern of the stretched matrix $\hat{A}$ and the stretched normal matrix $\hat{C}$ is illustrated in Figure 2.1. Here, $n = 64$ and the dense row f^T is stretched into $k = 8$ parts.

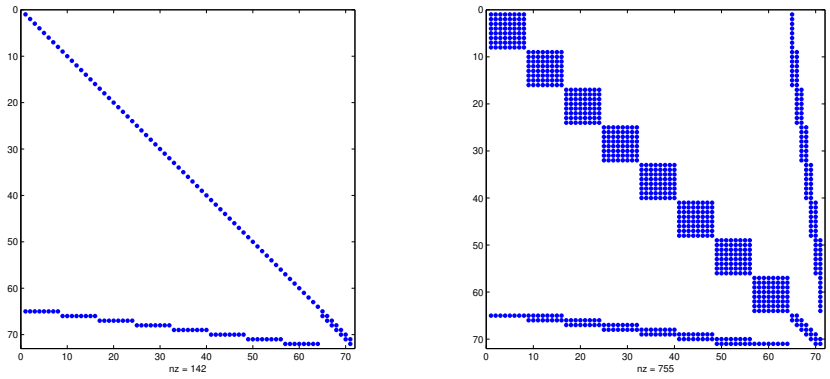

FIG. 2.1. *Structure of $\hat{A}$ (left) and $\hat{C}$ (right) for the matrix A with a single dense row and A_s diagonal with n=64 and the dense row stretched into 8 parts.*

We have the following result.

LEMMA 2.5. *Consider the $m \times n$ matrix $A = \begin{pmatrix} A_s \\ f^T \end{pmatrix}$ with a single dense row, $m = n+1$, $n = 2^r$ $(r > 1)$ and A_s diagonal. If the dense row is split into $k = 2^l$ equal parts $(0 \leq l \leq r-1)$ then the number $nz(\hat{C})$ of entries in the stretched normal matrix $\hat{C}$ is*

$$nz(\hat{C}) = 2^{2r-l} + 2^{r+2} - 2^{r-l+2} + 3\,(2^l - 1) - 2.$$

Proof. The stretched matrix $\hat{A}$ is of order $(n + 2^l) \times (n + 2^l - 1)$. The dense row is split into 2^l parts, each of length 2^{r-l}. $A_s^T A_s + FF^T$ is block diagonal with blocks of dimension 2^{r-l}, having a total of $2^l \times 2^{2(r-l)}$ entries. FS has $2^l - 1$ blocks each of size $2(2^{(r-l)}) \times 1$. Finally, $S^T S$ is tridiagonal with $3\,(2^l - 2) - 2$ nonzeros. Combining yields

$$nz(\hat{C}) = 2^l \times 2^{2(r-l)} + 2(2^l - 1)2(2^{(r-l)}) + 3(2^l - 1) - 2,$$

from which the result is obtained. ∎

In this case with A_s diagonal, there is no fill in the factors of $\hat{C}$.

LEMMA 2.6. *Under the same assumptions as in Lemma 2.5, the Cholesky factorization of $\hat{C}$ generates no fill.*

Proof. Since $A_s^T A_s + FF^T$ is block diagonal, there is no fill when it is factorized. Furthermore, there is no fill in the off-diagonal block γFS. All the updates of entries during the Cholesky factorization belong to the tridiagonal matrix $\gamma^2 S^T S$. But, because the construction of $\hat{C}$ always couples only two neighbouring parts of the stretched matrix $\hat{A}$, the block $\gamma^2 S^T S$ embeds all the updates within its tridiagonal structure. ∎

Figure 2.2 plots the number of entries in $\hat{A}$ and $\hat{C}$ for the matrix A with a single dense row that is split into an increasing number of parts and A_s diagonal ($n = 64$). Even in this simple case we see that, after an initial reduction, $nz(\hat{C})$ (and hence the number of entries in the Cholesky factor of $\hat{C}$) stagnates as the number of parts increases. This indicates that it may not be worthwhile to split dense rows into a large number of parts.

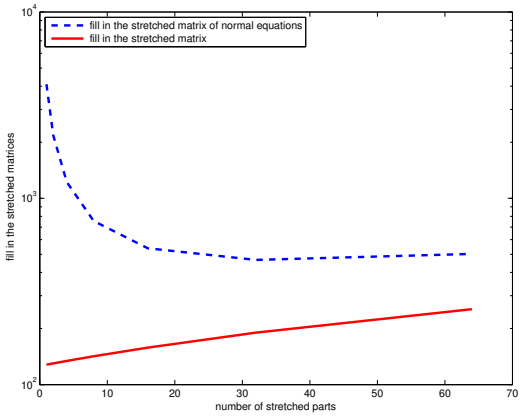

FIG. 2.2. *Number of entries in $\hat{A}$ and $\hat{C}$ when A has a single dense row that is split into an increasing number of parts and A_s is diagonal ($n = 64$).*

2.3.2. A_s tridiagonal. We now consider A_s tridiagonal and again we assume A has a single dense row. The following lemma shows that the Cholesky factorization $\hat{C} = \hat{L}^T \hat{L}$ of the stretched normal matrix can suffer from significant fill caused by the interaction of the stretched row with A_s.

LEMMA 2.7. *Consider the $m \times n$ matrix $A = \begin{pmatrix} A_s \\ f^T \end{pmatrix}$ with a single dense row, $m = n + 1$, $n = 2^r$ ($r > 1$) and A_s tridiagonal. If the dense row is split into $k = 2^l$ equal parts ($0 \le l \le r - 1$), then*

$$nz(\hat{C}) = 2^{2r-l} + 2^{r+2} - 2^{r-l+2} + 3\,(2^l - 1) - 2.$$

Furthermore, if $\hat{C} = \hat{L}^T \hat{L}$ then

$$nz(\hat{L} + \hat{L}^T) = 2^l \times 2^{2\,(r-l)} + 6(2^l - 1) + (2^l - 1)(2^l - 1) + 2\,(2^l - 1)2^{(r-l)} + 2^{(r-l)}(2^{(l-1)}(2^l - 1)).$$

Proof. The dense row is split into 2^l parts, each of length 2^{r-l}. $A_s^T A_s + FF^T$ has blocks on the diagonal of dimension 2^{r-l} plus nonzeros between the blocks because A_s is tridiagonal. This gives $2^l \times 2^{2(r-l)} + 6(2^l - 1)$ entries. The off-diagonal block FS and the remaining diagonal block $\hat{S}^T S$ then contribute $2\,(2\,(2^l - 2)2^{(r-l)} + 2 \times 2^{(r-l)})$ and $2^l - 1 + 2(2^l - 2)$ entries, respectively. From this we obtain

$$nz(\hat{C}) = 2^l \times 2^{2(r-l)} + 6(2^l - 1) + 2\,(2\,(2^l - 2) \times 2^{(r-l)} + 2 \times 2^{(r-l)}) + 2^l - 1 + 2\,(2^l - 2),$$

which gives the first result. To compute $nz(\hat{L} + \hat{L}^T)$, we need to consider the fill. There is no fill from factorizing $A_s^T A_s + FF^T$ and its contribution is thus $2^l \times 2^{2(r-l)} + 6(2^l - 1)$. The $(2,2)$ block fills in completely, contributing $(2^l - 1)(2^l - 1)$ entries. Finally, the full lower-trapezoidal block in $\hat{L}$ contributes $2 \times (2^l - 1) \times 2^{(r-l)} + 2^{(r-l)}(1 + \ldots + 2^l - 1) = 2 \times (2^l - 1) \times 2^{(r-l)} + 2^{(r-l)}(2^{(l-1)}(2^l - 1))$ entries and the result follows. ∎

Figure 2.3 illustrates the sparsity patterns of $\hat{C}$ and its Cholesky factor for the matrix in Lemma 2.7 with $n = 64$ and the dense row split into 8 parts. Figure 2.4 plots the number of entries in $\hat{A}$, $\hat{C}$ and its Cholesky factor as the number k of parts into which the dense row is split is increased. Here the fill in the Cholesky factor increases rapidly with k (even though the basic sparsity pattern of $\hat{C}$ is an arrowhead [12]). In this example, the fill can be reduced by reordering $\hat{C}$ prior to performing the Cholesky factorization. Figure 2.5 illustrates the effect of reordering using the approximate minimum degree (AMD) algorithm [4].

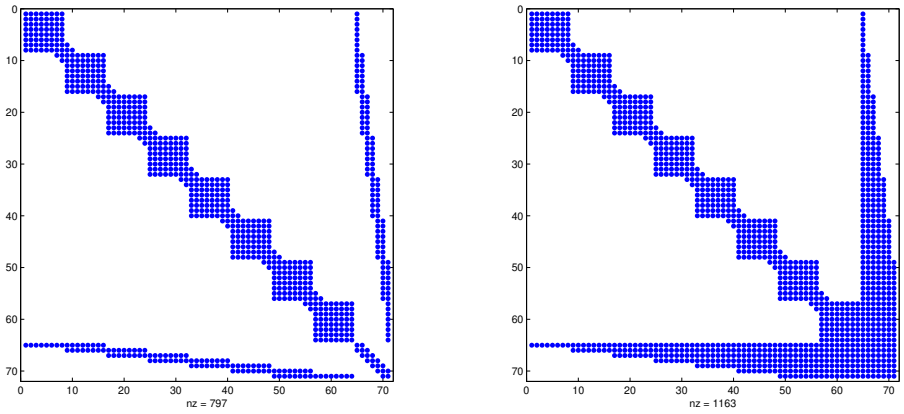

FIG. 2.3. *Structure of $\hat{C}$ and its Cholesky factor for the matrix A with A_s tridiagonal and a single dense row that is split into 8 parts ($n = 64$).*

2.3.3. A_s with Laplacian structure. The third example we consider is A_s having the structure of the two-dimensional 5-point discrete Laplacian matrix and one dense row is appended. In this case, as can be seen in Figures 2.6 and 2.7, reordering the stretched normal matrix offers some advantages but does not prevent the fill from growing as the number of parts the dense row is split into increases. We remark that these results are for AMD ordering but similar results are obtained using nested dissection ordering.

3. Sparse stretching. We have seen that standard stretching based on the simple splitting of the dense row into a number of contiguous segments can result in significant fill in the stretched normal matrix and, in particular, in its factor. In this section, we introduce a new sparse stretching strategy that aims to limit this fill. We use the same notation as in Section 2.2 and in particular, we initially assume a single dense row f^T and use the notation (2.5). We want to perform the splitting of f^T (that is, the construction of F^T) so as to limit the entries in $\hat{C}$. We make the following observations.

OBSERVATION 3.1. *$S^T S$ is tridiagonal and so the number of entries in the (2,2) block of $\hat{C}$ depends only on the order of S (that is, on the number of parts k the dense row is split into). Thus we want k to be small.*

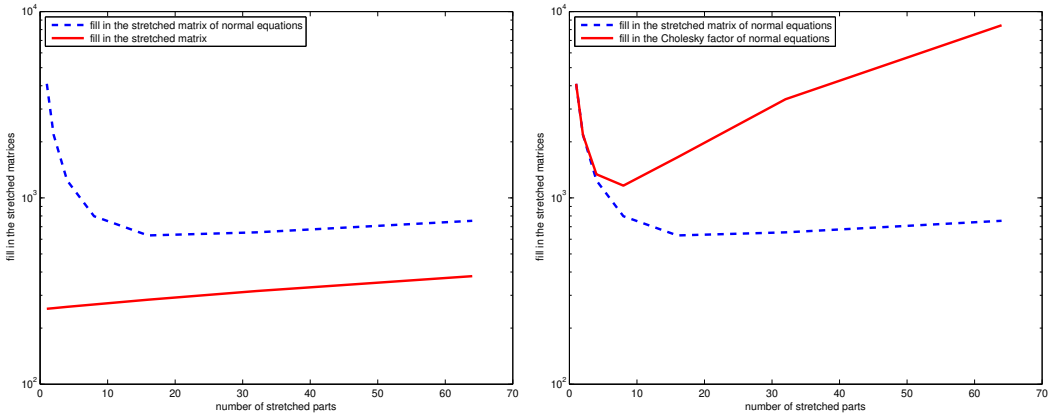

FIG. 2.4. *Number of nonzeros in $\hat{A}$ and $\hat{C}$ (left) and number of nonzeros in $\hat{C}$ and its Cholesky factor (right) for the matrix A with A_s tridiagonal and a single dense row that is split into an increasing number of parts ($n = 64$).*

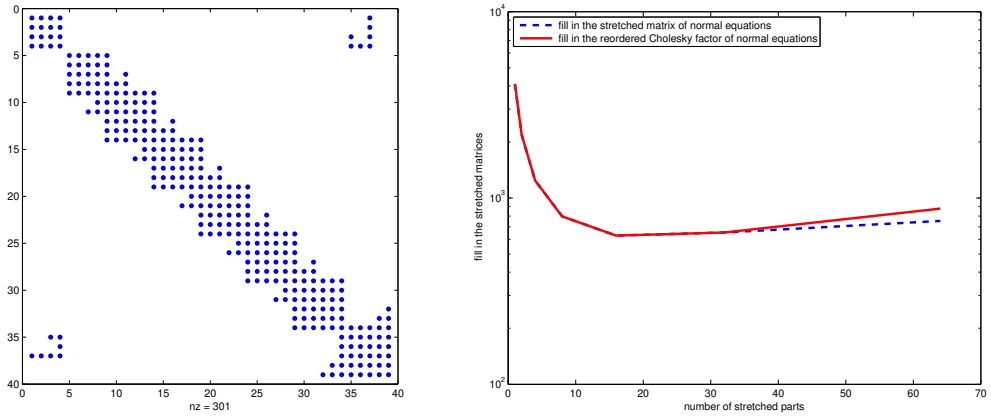

FIG. 2.5. *Structure of the Cholesky factor for the AMD reordered $\hat{C}$ for the matrix A with A_s tridiagonal and a single dense row that is split into 8 parts ($n = 64$) (left); the number of nonzeros in the reordered $\hat{C}$ and its Cholesky factor (right) for the same matrix with the dense row split into an increasing number of parts.*

OBSERVATION 3.2. *The fill in the principal leading block $A_s^T A_s + FF^T$ of $\hat{C}$ resulting from the dense row is a minimum if the structure of FF^T is contained within that of $A_s^T A_s$.*

Consider a sparse vector u. Define $Struct(u)$ to be the set of indices corresponding to the non zero entries in u. This can be generalised to a sparse matrix X by defining $Struct(X)$ to be the set of positions (i, j) of the non zero entries in X. The condition on embedding stated in Observation 3.2 can then be written as

$$Struct(FF^T) \subseteq Struct(A_s^T A_s). \tag{3.1}$$

We define the concept of dominance among rows of a sparse matrix.

DEFINITION 3.1. *Consider a sparse matrix X. Row v^T of X is said to* structurally dominate *row u^T of X if*

$$Struct(u) \subseteq Struct(v).$$

We have the following straightforward result.

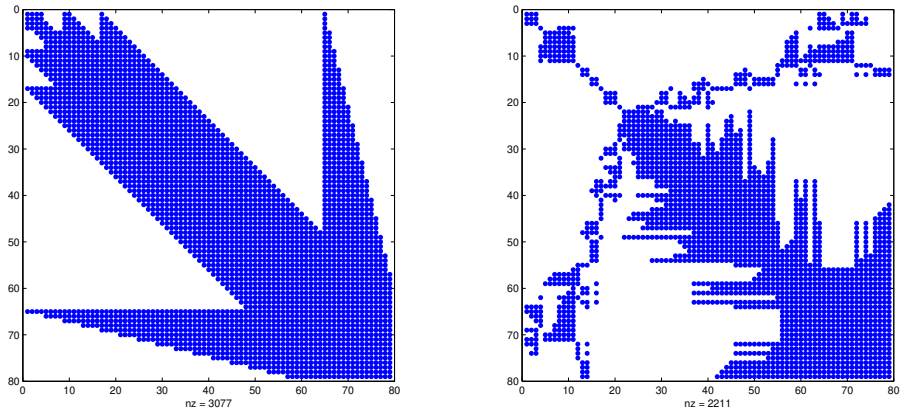

FIG. 2.6. *Structure of the Cholesky factor of the stretched normal matrix with no reordering (left) and AMD ordering (right). A_s has the structure of the two-dimensional 5-point discrete Laplacian matrix; one dense row is appended and split into 8 parts ($n = 64$).*

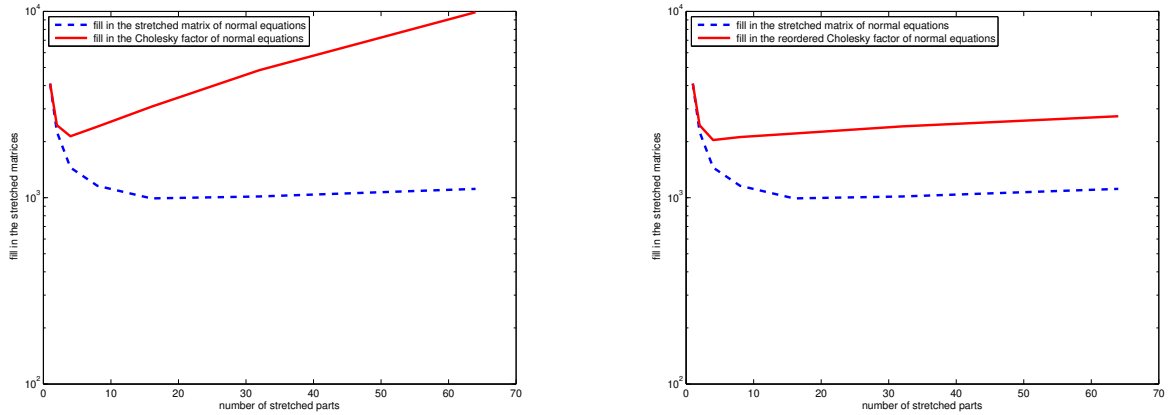

FIG. 2.7. *Number of nonzeros in the Cholesky factor of $\hat{C}$ with no reordering (left) and with AMD ordering (right) as the number of stretched parts increases. A_s has the structure of the two-dimensional 5-point discrete Laplacian matrix and one dense row is appended ($n = 64$).*

LEMMA 3.2. *Consider an $m \times n$ sparse matrix X and let u^T be a $1 \times n$ sparse row vector. If there exists a row v^T of X that structurally dominates u^T then*

$$Struct(\begin{pmatrix} X^T & u \end{pmatrix} \begin{pmatrix} X \\ u^T \end{pmatrix}) \subseteq Struct(X^T X).$$

As a simple example, consider the 4×4 matrix

$$X = \begin{pmatrix} * & * & & * \\ & * & & \\ * & * & & \\ & & * & \end{pmatrix}.$$

The sparsity patterns of rows 2 and 3 are both contained within that of row 1 and so row 1 structurally dominates rows 2 and 3. If $u^T = \begin{pmatrix} * & 0 & 0 & * \end{pmatrix}$ then again row 1 structurally dominates u^T and it is

straightforward to verify that the sparsity pattern of $\begin{pmatrix} X^T & u \end{pmatrix} \begin{pmatrix} X \\ u^T \end{pmatrix}$ and of $X^T X$ is

$$\begin{pmatrix} * & * & & * \\ * & * & & * \\ & & * & \\ * & * & & * \end{pmatrix}$$

The idea behind sparse stretching is to find a splitting of $Struct(f)$ into k disjoint non-empty index sets $t_1, \ldots, t_k$ $(k \leq r)$ and to construct F^T so that its i-th row contains the $|t_i|$ entries of $Struct(f)$ corresponding to the i-th part of the splitting. Any such splitting that satisfies (3.1) is said to define a *correct stretching*. From Lemma 3.2, finding a correct stretching involves finding index sets so that each row of F^T is dominated by one or more rows of A_s. This problem is closely related to the well-known problem of minimizing a set cover, which we discuss below.

DEFINITION 3.3. *Consider a set U and a collection $\mathcal{R}$ of subsets of U whose union is equal to U. A subcollection $\mathcal{Q} \subseteq \mathcal{R}$ is a set cover for $B \subseteq U$ if*

$$B \subseteq \bigcup_{q \in \mathcal{Q}} q. \tag{3.2}$$

$\mathcal{Q}$ is a minimum set cover for B if it is the smallest subcollection of $\mathcal{R}$ satisfying (3.2). $\mathcal{Q}$ is a minimal set cover for B if there is no proper subset of $\mathcal{Q}$ satisfying (3.2).

To express the relationship between a minimum vertex cover for B and a correct stretching we use the following bipartite graph.

DEFINITION 3.4. *Denote the i-th row of A_s by $(A_s)_{i*}$ $(1 \leq i \leq m_s)$ and let $Struct(f) = \{j_1, \ldots, j_r\}$ where $r = |Struct(f)|$. The bipartite graph $G = (R, B, E)$ with vertex sets $R = \{1, \ldots, m_s\}$ and $B = \{j_1, \ldots, j_r\}$ and edges given by*

$$(i, j) \in E \Longleftrightarrow j \in Struct((A_s)_{i*}) \tag{3.3}$$

is called the bipartite row intersection graph *of A_s and f^T.*

As an example, consider the matrix A on the left of Figure 3.1. Here $A_s \in R^{14 \times 12}$ and $Struct(f) = \{1, 2, 3, 4, 5, 6, 7, 8, 10, 12\}$ $(r = 10)$. The bipartite row intersection graph of A_s and f^T is given (middle) and a subgraph that contains only the edges involving the vertices in the vertex cover for B (right). The vertex cover $\mathcal{Q}'$ depicted here is given by the subset $\{2, 5, 8, 9\}$ of the vertices in the set R corresponding to rows $(A_s)_{2*}$, $(A_s)_{5*}$, $(A_s)_{8*}$ and $(A_s)_{9*}$ of A. Thus $\mathcal{Q}' = \{t'_1, t'_2, t'_3, t'_4\}$ with

$$t'_1 = \{2, 8, 10\}, \quad t'_2 = \{1, 2, 5, 7\}, \quad t'_3 = \{3, 12\}, \quad t'_4 = \{3, 4, 6\}.$$

LEMMA 3.5. *Let $G = (R, B, E)$ be the bipartite row intersection graph of A_s and f^T and let $\mathcal{Q}' = \{t'_1, \ldots, t'_k\}$ be a minimal vertex set cover for B. Consider the transformation of $\mathcal{Q}'$ to another vertex cover $\mathcal{Q} = \{t_1, \ldots, t_k\}$ such that*

$$Struct(t_u) \cap Struct(t_v) = \emptyset,\ u, v \in \{1, \ldots, k\},\ u \neq v. \tag{3.4}$$

Then the sets of column indices in $\mathcal{Q}$ represent a splitting that gives a correct stretching.

Proof. From the definition of a vertex set cover of B it follows that the union of the sets in $\mathcal{Q}'$ contains all indices in $Struct(f)$. Removal of entries from $\mathcal{Q}'$ makes its members disjoint. Since the cover is minimal, there is no subcollection of $\mathcal{Q}'$ that is a vertex set cover of B. This means that none of the members of $\mathcal{Q}$ is empty. Consequently, $\mathcal{Q}$ defines a correct stretching. ∎

Recall Observations 3.1 and 3.2. From the above, the structure of FF^T is embedded within that of $A_s^T A_s$ when the stretching is based on the transformation of a minimal vertex set cover of B to obtain

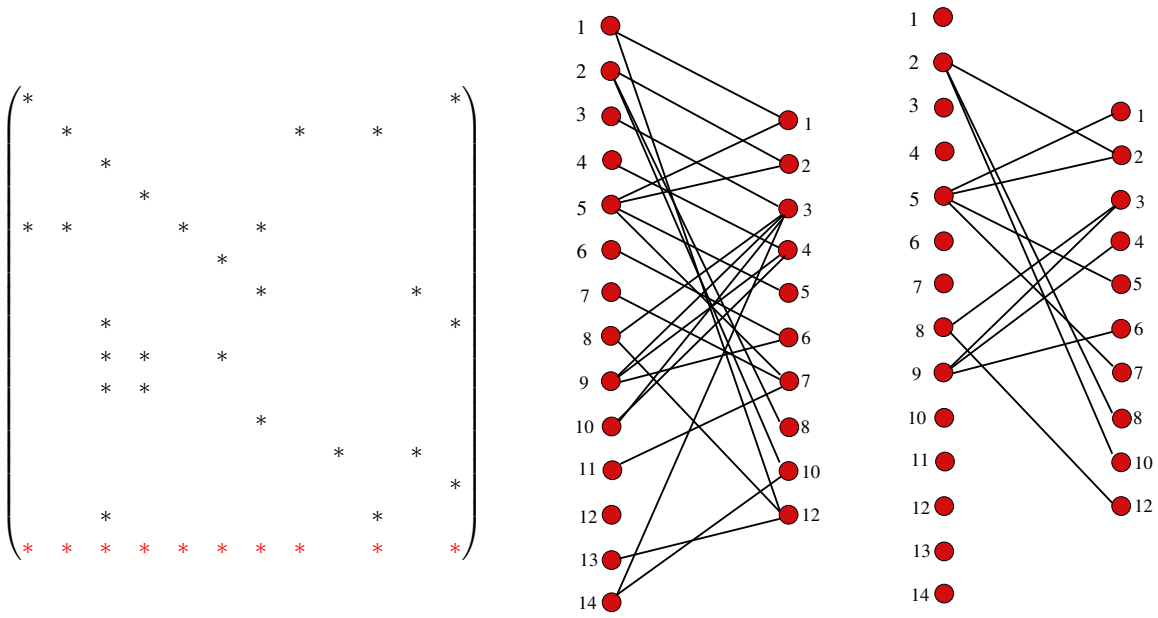

FIG. 3.1. *A matrix of order* 15×12 *with one dense row (left); the bipartite row intersection graph* $G = (R, B, E)$ *of* A_s *and* f^T *(middle); a subgraph containing only the edges involving the vertices in the vertex cover for* B *(right).*

disjoint sets. Thus to minimise the fill in $\hat{C}$ (2.5), it remains to consider fill in FS. Recall that S is tridiagonal and so it is easy to see that the fill in FS is given by

$$|t_1| + |t_k| + 2\sum_{i=2}^{k} |t_i|. \tag{3.5}$$

This is minimized by reordering the sets within $\mathcal{Q}$ so that

$$|t_1| \geq |t_k| \geq \max_{2 \leq i \leq k-1} |t_i|. \tag{3.6}$$

Our algorithm for determining the splitting on which sparse stretching is based can now be presented (Algorithm 3.1). We remark that the number k of parts into which the dense row is split is determined by step 2 of the algorithm as the number of sets in the minimal vertex set cover.

ALGORITHM 3.1. **Sparse stretching of A with one dense row**

Input: *$A \in R^{m \times n}$ with $m \geq n$ split into two parts A_s and A_d where $A_s \in R^{m_s \times n}$ and $A_d \equiv f^T \in R^{1 \times n}$*

Output: *k and disjoint index sets $t_1, t_2, \ldots, t_k$ such that $Struct(f) = \cup_{1 \leq i \leq k} t_i$*

1. *Construct the bipartite row intersection graph $G = (R, B, E)$ of A_s and f^T.*
2. *Find a minimum vertex set cover $\mathcal{Q}' = \{t'_1, \ldots, t'_k\}$ for B. (This determines k.)*
3. **for** $i = 1, \ldots, k$ **do**
4. *Set $t_i = \text{argmax}_{\tau \in \mathcal{Q}'} |\tau|$, update $\mathcal{Q}' \Leftarrow \mathcal{Q}' \setminus t_i$.*
5. **for** $q \in \mathcal{Q}'$ **do**
6. $q \Leftarrow q \setminus (q \cap t_i)$
7. **end do**
8. **end do**
9. *Set $t = t_2$.*

10. **for** $i = 2, \ldots, k-1$ **do**

11. *Set* $t_i = t_{i+1}$

12. **end do**

13. *Set* $t_k = t$.

We illustrate Algorithm 3.1 for the example in Figure 3.1. Starting with $\mathcal{Q}' = \{t'_1, t'_2, t'_3, t'_4\}$, the algorithm finds the vertex set cover $\mathcal{Q} = \{t_1, t_2, t_3, t_4\}$ with disjoint sets

$$t_1 = \{1, 2, 5, 7\}, \quad t_2 = \{8, 10\}, \quad t_3 = \{12\}, \quad t_4 = \{3, 4, 6\}.$$

We illustrate this graphically in Figure 3.2. On the left, we have the $k = 4$ rows of A chosen by solving the minimal vertex cover problem. Then, on the right, the sets of indices corresponding to these rows are made pairwise disjoint by removing some of the entries and then reordering so that the sets with the largest number of entries are the first and last ones. This gives the required sparse stretching of the dense row into 4 parts, that is, it gives *Struct*(F). Observe that the rows of F^T are, respectively, dominated by rows 5, 2, 8 and 9 of A_s.

$$\begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \left(\begin{array}{cccccccccccc} & * & & & & & & * & & * & & \\ * & * & & & * & & * & & & & & \\ & & * & & & & & & & & & * \\ & & * & * & & * & & & & & & \end{array}\right). \end{array} \qquad F^T = \begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \left(\begin{array}{cccccccccccc} * & * & & & * & & * & & & & & \\ & & & & & & & * & & * & & \\ & & * & * & & * & & & & & & * \end{array}\right). \end{array}$$

FIG. 3.2. *For the problem in Figure 3.1, the rows of A that solve the minimal vertex cover problem are on the left and on the right is the sparsity pattern of the stretched block* F^T.

We now consider the complexity of the preprocessing based on Algorithm 3.1. We focus on step 2 because the other steps pass through the selected rows only once and hence their complexity is low. The decision version of the set cover problem is one of twenty-one classical NP-complete problems introduced by Karp [25]. There are a number of ways to construct a minimum vertex set cover; we would like the cardinality k to be small. A classical treatment of this problem and its extensions can be found, for example, in [36]. In our experiments, we adopt a greedy approach. Given the bipartite row intersection graph $G(R, B, E)$, for each $i \in R$ we define the adjacency set

$$adj_i = \{j \in B | (i, j) \in E\},$$

that is, adj_i is the set of neighbours of i in B. For each $j \in B$, the adjacency set adj_j is defined analogously. If we store the indices of the entries in A_s by both rows and columns then adj_i and adj_j are readily available. The degrees deg_i and deg_j are the numbers of entries in adj_i and adj_j, respectively. The greedy algorithm starts by marking all the vertices in B as uncovered and, for each $i \in R$, initialises the count of uncovered neighbours in B to $count_i = deg_i$; the m vertices in R are then sorted into decreasing order of their counts. Starting with the first vertex in the sorted sequence, at each step, the next vertex $i \in R$ in the sequence is selected. The uncovered neighbours of i are marked as covered, and for each of the remaining vertices $r \in R$, $count_r$ is updated and the sorted sequence amended. It follows that an upper bound on the operation count of the greedy algorithm is given by

$$m + k \max_{i \in R} deg_i \max_{j \in B} deg_j.$$

To illustrate that this complexity is comparable with other standard preprocessing steps used by sparse solvers, consider the algorithms that are used to make a sparse matrix more diagonally dominant. These are commonly employed within modern sparse solvers to help limit the need for pivoting that can adversely

effect performance and limit the scope for parallelism (see, for example, [11, 22, 26, 28]) and are based on maximum matching [13, 14, 16]. For an $m \times n$ sparse matrix M, computing a maximum matching involves employing a bipartite graph G_M in which the vertex sets Row and Col correspond to the rows and columns of M with an edge between $r \in Row$ and $c \in Col$ if $M_{ij} \neq 0$. Each step of the maximum matching algorithm searches for augmenting paths in G_M. This involves passing through the adjacency sets for both the row and column vertices and the complexity can be shown [16] to be $n\,|M|$ ($|M|$ denotes the number of entries in M), although in practice maximum matching algorithms are typically much faster than this.

We observe that another interesting aspect of the greedy algorithm is its approximation quality, see, for example, [24]. It can be shown that if k_{min} is the smallest possible number of rows in the set cover, that is, the cardinality of the optimum solution, then k in Algorithm 3.1 satisfies $k \leq k_{min} \ln n$, where ln denotes the natural logarithmic function (see [9]).

Finally, we remark that we have discussed stretching of a single row f^T. In the case of more than one such row, we stretch each separately (the number of parts k is typically different for each row). The matrix is extended by a block F_i^T for each dense row f_i^T that is stretched. While the initial classification of rows is straightforward since it is enough to mark the rows and not to move them, separate constructions of the graph model by Algorithm 3.1 for each row can result in significant increases in the preprocessing time. In our future research we plan to try and overcome this by handling blocks of dense rows as a single entity.

4. Numerical results. In this section, we demonstrate that sparse stretching finds a suitable number of parts for the splitting and can significantly reduce the fill in both the stretched normal matrix and its Cholesky factor compared to standard stretching.

Our first set of examples do not initially contain dense rows; instead we append one or more dense rows. This allows us to explore the effect of varying the number of dense rows as well as the density of these rows. The problems are listed in Table 4.1. They are taken from the University of Florida Sparse Matrix Collection [10] and, if necessary, are transposed so that $m_s > n_s$.

TABLE 4.1
Statistics for Test Set 1. m_s, n_s and $nz(A_s)$ are the row and column counts and the number of nonzeros in A_s.

Problem	m_s	n_s	$nz(A_s)$
WM1	277	207	2909
LP_AGG	615	488	2862
GAMS60AM	1071	714	2607
MARAGAL_5	4654	3320	93091
CEP1	4769	1521	8233
STORM2-8	11322	4409	28553
IG5-15	11369	6146	323509
KEMELMACHER	28452	9693	100875
BAXTER	30733	27441	11576
TESTBIG	31223	17613	61639
STORMG2-27	37485	14441	94274
WORLD	67147	34506	198883
MRI1	147456	65536	589824

4.1. Two small examples. We first consider two examples WM1 and LP_AGG that are small enough for us to present plots of the stretched matrix and the stretched normal matrix and its factors. In both cases, one fully dense row is appended to the given matrix. These problems illustrate the potential advantages of sparse stretching over standard stretching. For example WM1, Figure 4.1 reports the entries in the stretched normal matrix (left) and its Cholesky factor (right). Results for standard stretching are plotted for an increasing number of stretched parts while for sparse stretching a single point is plotted, corresponding to the number of parts determined by Algorithm 3.1.

Analogous results for LP_AGG are summarized in Figure 4.2. In this case, the value of k returned by Algorithm 3.1 is 55. For this k, the sparsity patterns of $\hat{C}$ for standard stretching (left) and sparse

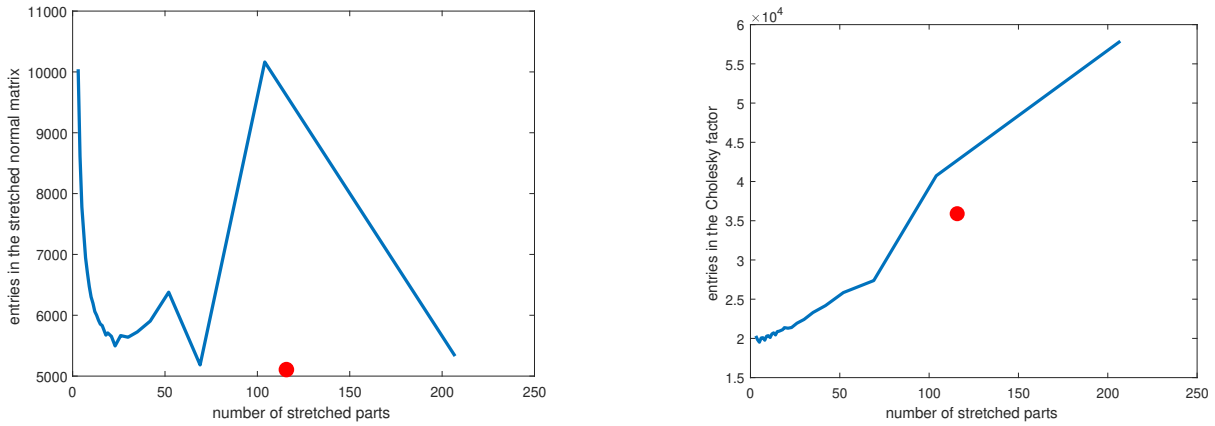

FIG. 4.1. *Comparison of the entries in the stretched normal matrix (left) and its Cholesky factor (right) for problem* `WM1` *with one dense row appended. The curve corresponds to the number of entries varying with the number of parts into which the dense row is stretched (standard stretching); the dot is for sparse stretching with the number of stretched parts determined by Algorithm 3.1 (*$k = 116$*).*

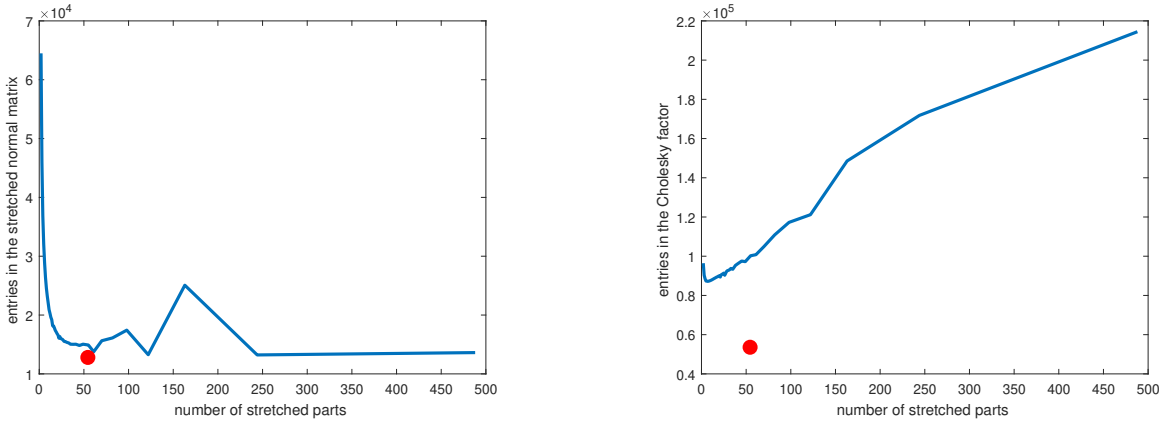

FIG. 4.2. *Comparison of the entries in the stretched normal matrix (left) and its Cholesky factor (right) for problem* `LP_AGG` *with one dense row appended. The curve corresponds to the number of entries varying with the number of parts into which the dense row is stretched (standard stretching); the dot is for sparse stretching with the number of stretched parts determined by Algorithm 3.1 (*$k = 55$*).*

stretching (right) are given in Figure 4.3 and its Cholesky factors with and without reordering $\hat{C}$ before it is factorized are given in Figures 4.4 and 4.5, respectively.

The emphasis of our study is on the structural effects of sparse stretching and the potential this has to make LS problems with a few dense rows more tractable for both direct and iterative solvers. Nevertheless, it is of interest to consider how stretching effects conditioning. In Figure 4.6, for examples `WM1` and `LP_AGG` we plot the condition number (computed using the Matlab function `condest`) of the normal matrix corresponding to the original matrix with one dense row appended and to the stretched matrix (the former has zero stretched parts). As expected, the condition number increases with the number of stretched parts. But for `WM1`, we observe that stretching the dense row into a small number of parts improves the conditioning compared to the unstretched problem.

4.2. Comparison of standard and sparse stretching. We now compare standard and sparse stretching when applied to the remaining problems in Test Set 1; again, a single dense row is added. Results are given in Table 4.2. Here $Ratio(\hat{C})$ is the ratio of the number of entries in the stretched normal matrix $\hat{C}$ using sparse stretching to the number using standard stretching. Similarly, $Ratio(\hat{L})$ is the ratio of the number of entries in the Cholesky factor of $\hat{C}$ using sparse stretching to the number using standard stretching; $nz_{AMD}(\hat{L})$ and $Ratio_{AMD}(\hat{L})$ indicate that $\hat{C}$ is reordered before being factorized. We see

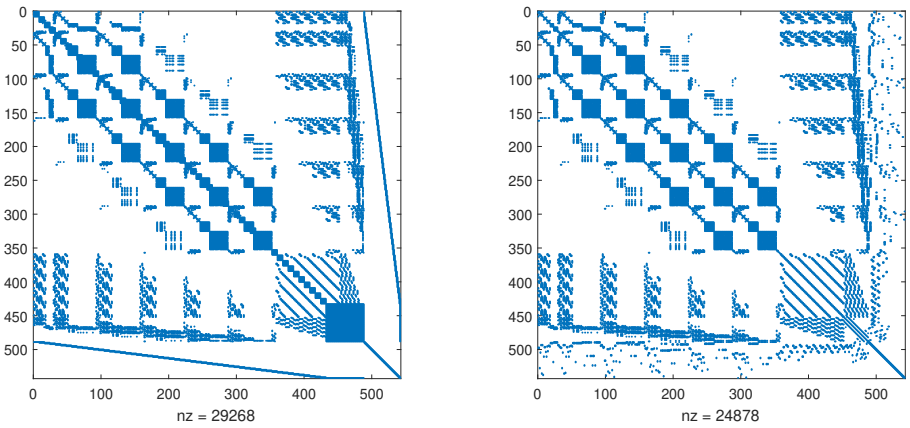

FIG. 4.3. *For problem* LP_AGG *with one dense row appended, the sparsity pattern of the stretched normal matrix for standard stretching (left) and sparse stretching (right) with the number of stretched parts determined by Algorithm 3.1.*

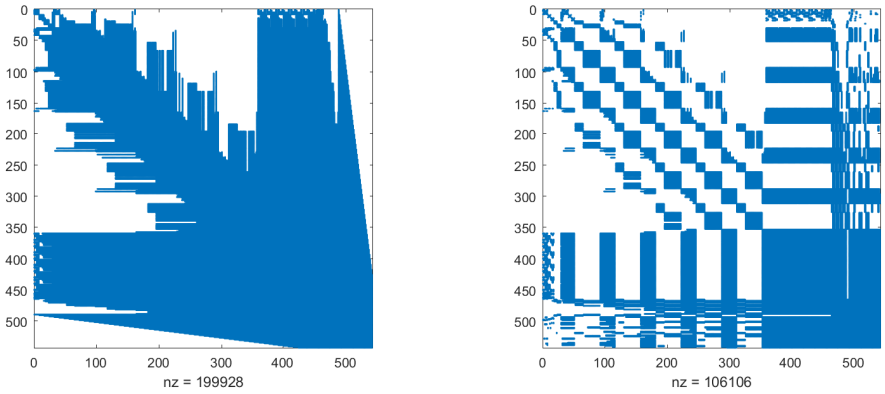

FIG. 4.4. *For problem* LP_AGG *with one dense row appended, the sparsity pattern of the Cholesky factor of the stretched normal matrix (without reordering) for standard stretching (left) and sparse stretching (right) with the number of stretched parts determined by Algorithm 3.1.*

that, in general, $\hat{C}$ is significantly sparser when sparse stretching is used in place of standard stretching ($Ratio(\hat{C})$ is less than 1.0 for all problems except IG5-15). This, in turn, generally leads to sparser factors, although this is not guaranteed (particularly if $\hat{C}$ is reordered).

4.3. The effects of varying the density. It is not necessary for a row to be fully dense for it to be advantageous to treat it as dense. Given the density $\rho < 1$, we randomly generate the sparsity pattern of the appended row. In Figure 4.7, for problem WORLD, we plot the number of entries in the Cholesky factor of $\hat{C}$ when the appended row has density $\rho = 0.01$, 0.05 and 0.1. As expected, as ρ increases, so too does the number of parts it needs to be split into to retain sparsity in the factors and the greater the advantage gained from using sparse stretching.

In Figure 4.8, we plot the number of entries in the Cholesky factor of the *AMD* reordered stretched normal matrix for an increasing number of added rows. Each added row is of density $\rho = 0.05$ (with the patterns generated randomly). Results are given for standard and sparse stretching. In each case, the number of stretched parts is determined by Algorithm 3.1. We observe the significant advantage of using sparse stretching.

4.4. Problems containing some dense rows. We now consider examples where the supplied matrix has a number of dense rows. These are taken from the Meszaros subcollection of the University of Florida Sparse Matrix Collection. In these experiments, we define a row of A to be dense if the number of entries either exceeds 100 times the average number of entries per row or is more than 4 times greater

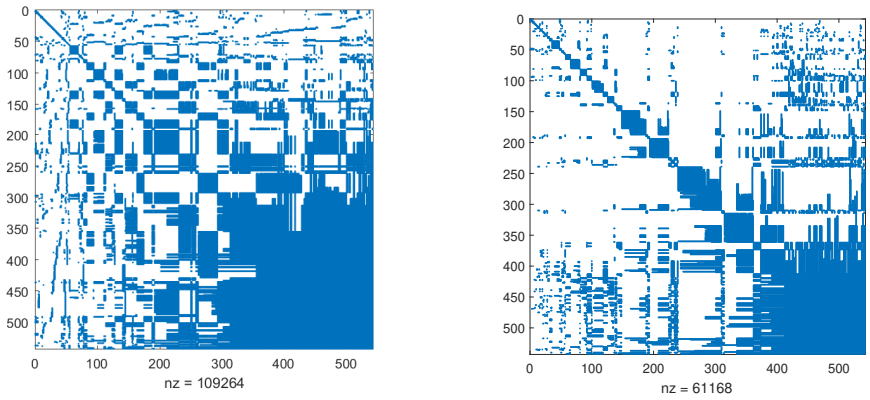

FIG. 4.5. *For problem* LP_AGG *with one dense row appended, the sparsity pattern of the Cholesky factor of the AMD reordered stretched normal matrix for standard stretching (left) and sparse stretching (right) with the number of stretched parts determined by Algorithm 3.1.*

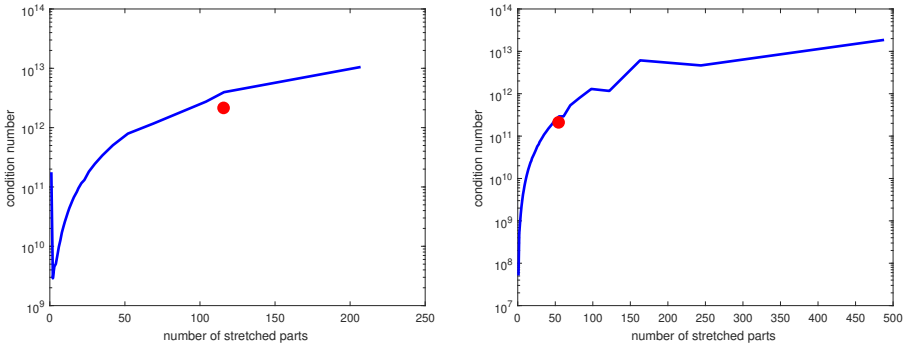

FIG. 4.6. *The condition number of the normal matrix for the original matrix with one dense row appended and for the stretched matrix for problems* WM1 *(left) and* LP_AGG *(right). The curves correspond to the condition number varying with the number of parts into which the dense row is stretched (standard stretching); the dot is for sparse stretching with the number of stretched parts determined by Algorithm 3.1.*

than the maximum number of entries in a row in the sparse part A_s [31]. In Table 4.3, we report results for standard and sparse stretching. The ratios are defined as in Section 4.2. Algorithm 3.1 is applied to each dense row; the total number of parts the dense rows are split into is reported. We see that, when there are multiple dense rows, the dimensions of the stretched matrix can be much greater than for the original matrix. Moreover, compared to standard stretching, sparse stretching can significantly reduce the number of entries in the factor $\hat{L}$, although the use of reordering can limit the differences between the two approaches.

4.5. Use with an iterative solver. So far, we have presented results for complete factorizations. However, we can also perform an incomplete Cholesky (IC) factorization of the stretched normal matrix and use it as as a preconditioner for an iterative solver. Here we illustrate the effectiveness of combining standard and sparse stretching with an iterative solver; in a future study, more extensive results will be given and comparisons made to other approaches for handling dense rows within an iterative solver, such as have been recently proposed in [31, 32]. To perform the IC factorization, we use the package HSL_MI35 from the HSL mathematical software library [23]. HSL_MI35 implements a limited memory IC algorithm; details are given in [29, 30]. It requires the user to set the parameters $lsize$ and $rsize$ that respectively control the number of entries in each column of the IC factor and the memory required to compute the factorization. In general, increasing these parameters improves the quality of the preconditioner (so that the number of iterations of the preconditioned iterative solver is reduced) at the cost of more time and memory to compute the factorization and increased cost for each preconditioner application. In our experiments we set

TABLE 4.2

Results for Test Set 1 with a single dense row appended. k is the number of parts the dense row is split into, determined by Algorithm 3.1. The ratios are for sparse stretching to standard stretching.

Identifier	k	standard stretching			Ratios		
		$nz(\hat{C})$	$nz(\hat{L})$	$nz_{AMD}(\hat{L})$	$Ratio(\hat{C})$	$Ratio(\hat{L})$	$Ratio_{AMD}(\hat{L})$
GAMS60AM	203	1.1×10^4	1.3×10^5	1.6×10^4	0.48	0.60	1.38
MARAGAL_5	17	2.7×10^6	5.5×10^6	4.4×10^6	0.94	1.00	0.69
CEP1	652	1.3×10^5	1.1×10^6	3.9×10^5	0.82	0.90	1.02
STORMG2-8	2448	1.9×10^6	1.3×10^7	2.3×10^6	0.02	0.68	0.26
IG5-15	1116	1.1×10^7	6.9×10^7	5.7×10^7	1.00	1.04	1.01
KEMELMACHER	4701	1.4×10^5	3.8×10^7	3.2×10^6	0.71	0.95	0.83
BAXTER	7662	1.1×10^7	4.2×10^8	5.9×10^7	0.07	0.41	0.44
TESTBIG	8008	3.0×10^6	1.5×10^8	9.3×10^6	0.57	0.93	1.04
STORMG2-27	8002	2.0×10^7	1.4×10^8	2.2×10^7	0.01	0.65	0.14
WORLD	13443	2.9×10^7	2.0×10^8	3.3×10^7	0.01	0.68	0.10
MRI1	19721	2.1×10^7	1.6×10^9	5.3×10^7	0.04	1.00	0.40

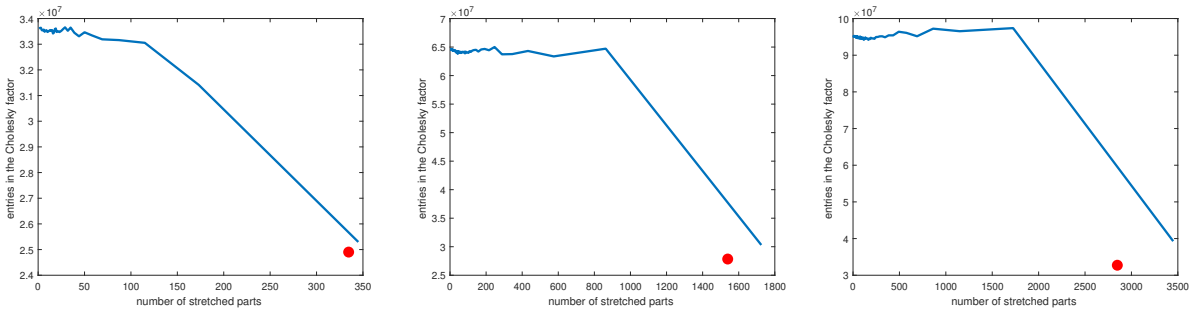

FIG. 4.7. *Problem* WORLD *with one appended row of density ρ =0.01, 0.05 and 0.1. In each case, the number of entries in the Cholesky factor of the stretched normal matrix is reported. The curves correspond to the number of entries varying with the number of parts into which the dense row is stretched (standard stretching); the dot is for sparse stretching with the number of stretched parts determined by Algorithm 3.1.*

$lsize = rsize$ (and so just report $lsize$). We employ preconditioned CGLS (an extension of the conjugate gradient method to least-squares problems) and terminate it using the following stopping rule from [19]:

$$\frac{\|\hat{A}^T\hat{r}\|_2}{\|\hat{r}\|_2} < \frac{\|\hat{A}^T\hat{r}^{(0)}\|_2}{\|\hat{r}^{(0)}\|_2} * \delta,$$

where $\hat{r} = \hat{b} - \hat{A}z$ is the residual of the stretched problem, $\hat{r}^{(0)} = \hat{b} - \hat{A}\hat{z}^{(0)}$ is the initial residual and the convergence tolerance is set to 10^{-6}. The (unstretched) vector b is taken to be the vector of all 1's and we take the initial solution guess for CGLS to be $\hat{z}^{(0)} = 0$. Note that the stopping rule is for the stretched system. Once it is satisfied, we perform a check that it is also satisfied for the original system.

In Figures 4.9 and 4.10, we present results for problem LP_AGG with a single dense row appended. Here we set the HSL_MI35 parameter $lsize$ to 25 and 50. Using sparse stretching, the iteration counts are 63 and 7, respectively. For standard stretching, the iteration counts as the number of parts increases do not form a smooth curve and it is not possible to predict the number of parts that will lead to a low iteration count. This can be explained from the structural point of view because the stretched segments that result from using different numbers of parts cause very different fill in the corresponding stretched normal matrix.

Examination of the condition number of the preconditioned stretched normal matrix shows that sparse stretching does not always offer an advantage over standard stretching. Despite this, sparse stretching leads to a lower iteration count, from which we conclude that the condition number can be a poor indicator of preconditioner quality. For this example, increasing the number of entries in the preconditioner by setting $lsize = 50$ significantly reduces the condition number along with the iteration counts.

In Figure 4.10, we plot the eigenvalues of the preconditioned stretched normal matrix ($lsize = 25$).

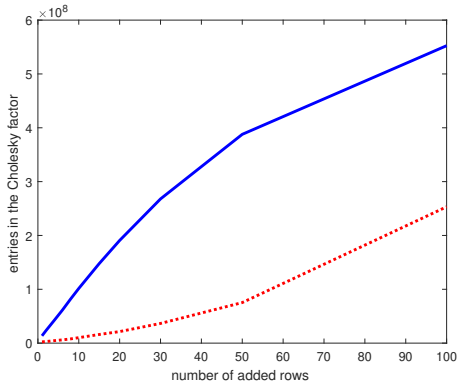

FIG. 4.8. *Problem* `WORLD` *with an increasing number of added rows, each of density* $\rho = 0.05$. *The solid curve is the entries in the Cholesky factor of the stretched normal matrix for standard stretching and the dotted curve is for sparse stretching.*

TABLE 4.3

Results for problems containing some dense rows. p *is the number of dense rows.* $\sum_i k_i$ *is the total number of parts the dense rows are split into.* m_{str} *and* n_{str} *are the dimensions of the stretched matrix* $\hat{A}$. *The ratios are for sparse stretching to standard stretching.*

Identifier	p	$\sum_i k_i$	m	n	m_{str}	n_{str}	Standard stretching $nz(\hat{L})$	Standard stretching $nz_{AMD}(\hat{L})$	Ratios $Ratio(\hat{L})$	Ratios $Ratio_{AMD}(\hat{L})$
`scrs8-2r`	22	12828	27669	14357	40497	27163	9.9×10^7	3.8×10^7	0.80	1.01
`sctap1-2b`	34	14420	33824	15390	48244	29776	2.1×10^8	2.5×10^7	0.49	0.83
`scsd8-2r`	50	21640	60500	8645	82140	30235	2.4×10^8	3.0×10^7	0.95	0.45
`scagr7-2r`	7	12966	46672	32846	59638	45805	1.3×10^8	2.2×10^7	0.62	0.20
`sctap1-2r`	34	26964	63392	28830	90356	55760	7.3×10^8	6.3×10^7	0.49	0.75
`scfxm1-2r`	58	15825	65885	37973	81710	53740	1.2×10^8	1.6×10^7	1.00	0.93

Following sparse stretching ($k = 55$), the number of entries in the incomplete factorization preconditioner is $13,741$ and the iteration count is 63. Standard stretching with $k = 163$ parts yields a preconditioner of a similar size ($16,575$ entries) but this fails to give convergence within 2000 iterations. The large number of eigenvalues close to zero may explain this poor performance.

5. Conclusions. In this paper, we have considered employing matrix stretching to tackle the problem of dense rows in otherwise sparse LS problems. Stretching for LS problems has been used in the past [1, 2] but, as we have illustrated using some problems with very basic sparsity structures, standard stretching can lead to unacceptably large amounts of fill in the stretched normal matrix and its factor; simply increasing the number of parts into which the dense rows are split does not necessarily alleviate the problem and can lead to a poorly conditioned problem. The novelty of our approach lies in performing the splitting so as to limit the fill in the stretched normal matrix. A practical algorithm for obtaining the proposed splitting has been presented and the new sparse stretching strategy has been shown to perform well when compared with standard stretching on a range of practical problems. Thus we have a new tool that can be used to help solve LS problems with dense rows.

The main focus has been on complete factorizations but we have also illustrated that it is possible to consider incomplete factorizations of the stretched normal matrix for use as a preconditioner with an iterative solver. Preliminary results suggest that sparse stretching is again preferable to standard stretching. Further investigations into using incomplete factorizations with sparse stretching will be part of a study in which we also explore combining stretching with updating techniques [32]. Updating techniques aim to use a factorization of the sparse reduced normal matrix $C_s = A_s^T A_s$ combined with the factorization of a dense subproblem involving A_d. However, even if A is of full column rank, removing the dense block A_d can result in the sparse part A_s being rank-deficient. In particular, A_s may contain one or more null columns and the Cholesky factorization of C_s breaks down. A possible option is to select some of the

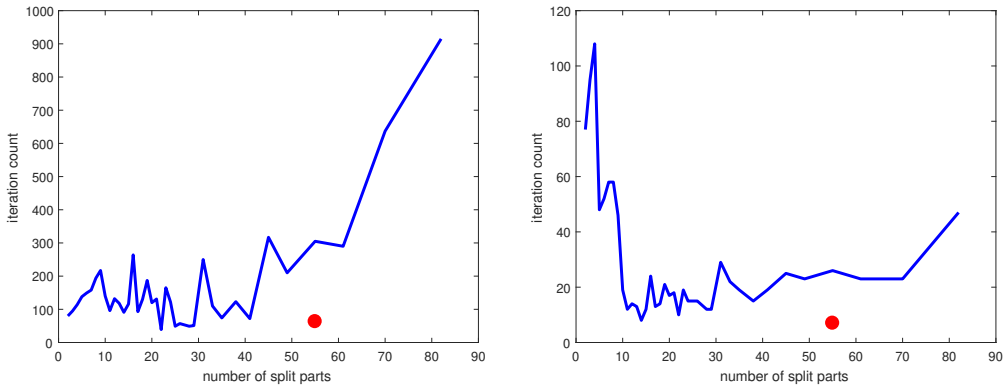

FIG. 4.9. *Comparison of iteration counts for standard and sparse stretching for problem* `LP_AGG` *with one dense row appended. The dot is for sparse stretching. Here* $lsize = 25$ *(left) and* $lsize = 50$ *(right).*

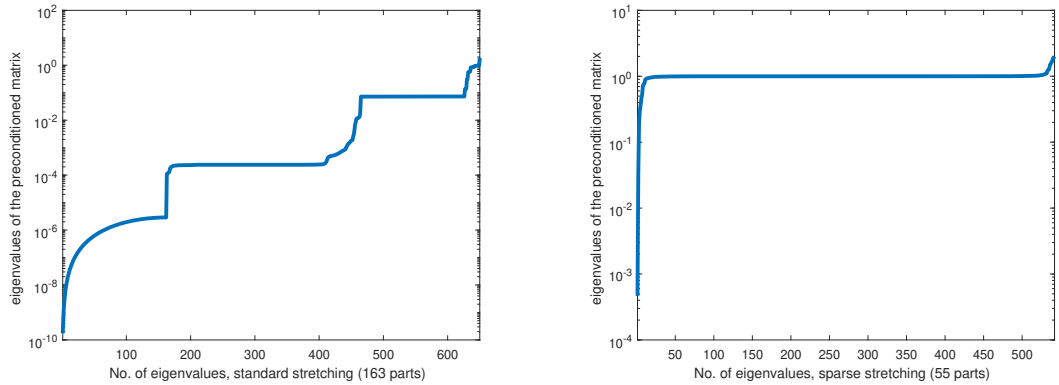

FIG. 4.10. *Spectrum of the preconditioned stretched normal matrix. Standard stretching with* $k = 163$ *(left) and sparse stretching (*$k = 55$*) (right). Here* $lsize = 25$*.*

rows of A_d, stretch them and then add the sparse parts to A_s to recover full rank. Again, the original problem will be replaced by a larger one but the hope is that the increase will be modest and will allow us to exploit the advantages of both updating and stretching in a single approach.

Acknowledgements. We are grateful to both Michael Saunders and an anonymous reviewer for their constructive feedback that led to improvements in this paper.

REFERENCES

[1] M. Adlers. *Topics in Sparse Least Squares Problems.* PhD thesis, Department of Mathematics, Linköpings Universitet, SE-581 83 Linköping, Sweden, available as Linköping Studies in Science and Technology. Dissertations No. 634, 2000.

[2] M. Adlers and Å. Björck. Matrix stretching for sparse least squares problems. *Numerical Linear Algebra with Applications*, 7(2):51–65, 2000.

[3] F. L. Alvarado. Matrix enlarging methods and their application. *BIT Numerical Mathematics*, 37(3):473–505, 1997.

[4] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. on Matrix Analysis and Applications*, 17:886–905, 1996.

[5] H. Avron, E. Ng, and S. Toledo. Using perturbed QR factorizations to solve linear least-squares problems. *SIAM J. on Matrix Analysis and Applications*, 31(2):674–693, 2009.

[6] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek. Permuting sparse rectangular matrices into singly-bordered block-diagonal form for parallel solution of LP problems. Technical Report BU-CE-0203, Computer Engineering Department, Bilkent Univeristy, Ankara, Turkey, 2002.

[7] Å. Björck. A general updating algorithm for constrained linear least squares problems. *SIAM J. on Scientific and Statistical Computing*, 5(2):394–402, 1984.

[8] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[9] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2008.

[10] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1–28, 2011.

[11] I. S. Duff. MA57– a new code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30:118–154, 2004.

[12] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, London, 1986.

[13] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. on Matrix Analysis and Applications*, 20:889–901, 1999.

[14] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. on Matrix Analysis and Applications*, 22:973–996, 2001.

[15] I. S. Duff and J. A. Scott. Stabilized bordered block diagonal forms for parallel sparse solvers. *Parallel Computing*, 31:275–289, 2005.

[16] I. S. Duff and T. Wiberg. Remarks on implementations of $O(n^{1/2}\tau)$ assignment algorithms. *ACM Transactions on Mathematical Software*, 14(3):267–287, 1988.

[17] M. C. Ferris and J. D. Horn. Partitioning mathematical programs for parallel solution. *Mathematical Programming*, 80:35–62, 1998.

[18] A. George and M. T. Heath. Solution of sparse linear least squares problems using Givens rotations. *Linear Algebra and its Applications*, 34:69–83, 1980.

[19] N. I. M. Gould and J. A. Scott. The state-of-the-art of preconditioners for sparse linear least squares problems. *ACM Transactions on Mathematical Software*, 43:Art. 36, 35 pages, 2017.

[20] J. F. Grcar. Matrix stretching for linear equations. Technical Report SAND90-8723, Sandia National Laboratories, 1990.

[21] M. T. Heath. Some extensions of an algorithm for sparse linear least squares problems. *SIAM J. on Scientific and Statistical Computing*, 3(2):223–237, 1982.

[22] J. D. Hogg and J. A. Scott. Pivoting strategies for tough sparse indefinite systems. *ACM Transactions on Mathematical Software*, 40:Art. 4, 19 pages, 2013.

[23] HSL. A collection of Fortran codes for large-scale scientific computation, 2019. `http://www.hsl.rl.ac.uk`.

[24] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974. Fifth Annual ACM Symposium on the Theory of Computing (Austin, Tex., 1973).

[25] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103. Plenum, New York, 1972.

[26] X.S. Li and J W. Demmel. SuperLU DIST: A scalable distributed-memory sparse direct solver or unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2):110–140, 2003.

[27] C. C. Paige and M. A. Saunders. LSQR: an algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, 1982.

[28] O. Schenk, A. Wächter, and M. Hagemann. Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization. *Journal of Computational Optimization and Applications*, 36(2-3):321–341, 2007.

[29] J. A. Scott and M. Tůma. `HSL_MI28`: an efficient and robust limited-memory incomplete Cholesky factorization code. *ACM Transactions on Mathematical Software*, 40(4):Art. 24, 19 pages, 2014.

[30] J. A. Scott and M. Tůma. On positive semidefinite modification schemes for incomplete Cholesky factorization. *SIAM J. on Scientific Computing*, 36(2):A609–A633, 2014.

[31] J. A. Scott and M. Tůma. Solving mixed sparse-dense linear least-squares problems by preconditioned iterative methods. *SIAM J. on Scientific Computing*, 39(6):A2422–A2437, 2017.

[32] J. A. Scott and M. Tůma. A Schur complement approach to preconditioning sparse linear least-squares problems with some dense rows. *Numerical Algorithms*, 79(4):1147–1168, 2018.

[33] C. Sun. Dealing with dense rows in the solution of sparse linear least squares problems. Research Report CTC95TR227, Advanced Computing Research Institute, Cornell Theory Center; Cornell University, 1995.

[34] C. Sun. Parallel solution of sparse linear least squares problems on distributed-memory multiprocessors. *Parallel Computing*, 23(13):2075–2093, 1997.

[35] R. J. Vanderbei. Splitting dense columns in sparse linear systems. *Linear Algebra and its Applications*, 152:107–117, 1991.

[36] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, 2001.