Received: 9 July 2015 | Revised: 24 October 2016 | Accepted: 1 March 2017

DOI: 10.1002/nla.2103

RESEARCH ARTICLE

WILEY

A comparative study of null-space factorizations for sparse symmetric saddle point systems

Tyrone Rees | Jennifer Scott

STFC Rutherford Appleton Laboratory, Chilton Didcot, UK

Correspondence
Tyrone Rees, STFC Rutherford Appleton Laboratory, Chilton, Didcot, UK.
Email: tyrone.rees@stfc.ac.uk

Funding information
EPSRC, Grant/Award Number: EP/I013067/1

Summary

Null-space methods for solving saddle point systems of equations have long been used to transform an indefinite system into a symmetric positive definite one of smaller dimension. A number of independent works in the literature have identified that we can interpret a null-space method as a matrix factorization. We review these findings, highlight links between them, and bring them into a unified framework. We also investigate the suitability of using null-space factorizations to derive sparse direct methods and present numerical results for both practical and academic problems.

KEYWORDS

direct methods, null-space methods, saddle point systems, sparse symmetric indefinite systems

1 | INTRODUCTION

A saddle point system is an indefinite linear system of equations of the form

$$\mathcal{A}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}. \tag{1}$$

Here, we will assume that $B \in \mathbb{R}^{m\times n}(n > m)$ has full rank and $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite on the null space of B. We are particularly interested in the case where A and B are large, sparse matrices, and our discussion will focus on A symmetric and positive semidefinite.

Saddle point systems (also known as KKT systems or augmented systems) arise frequently in scientific applications, particularly when solving constrained optimization problems. Sometimes the optimization framework is explicit, for example, when applying an interior point method to solve a linear or quadratic program, or when solving a least squares problem. More often, the optimization context is not so obvious, as is the case with incompressible fluid flow, electronic circuit simulation, and structural mechanics. The survey paper of Benzi et al.[1] describes a wide range of applications that require the solution of systems of the form 1.

One approach for solving Equation 1 is to use a null-space method.[1, section 6] These methods, which we will describe in detail below, have been used in the fields of optimization (where they are known as reduced Hessian methods), structural mechanics (where they are known as a "force" method or "direct elimination"), fluid mechanics (where they are known as the "dual variable" method), and electrical engineering (where they are known as "loop analysis"). This approach remains particularly popular in the large-scale optimization literature.[2-7]

There has been a sizable body of work—some historical but much recent—that has (directly or indirectly) revisited null-space methods and put them into the context of a matrix factorization, see, for example, the references.[8-18] Our main contribution is

to bring these factorizations together within a unified framework, highlighting the relationships between them (some of which do not appear to be well known). In addition to drawing links between different null-space factorizations, we compare their relative merits. We focus on two issues that are vital for determining the potential of a sparse direct solver, namely the sparsity of the factors and the stability of the factorization.

Null-space methods require a basis for the null space of B. This typically comes from either finding an invertible subblock of B or performing a QR factorization of B; we discuss different choices in Section 2, drawing them into a unified framework. The choice of null-space basis affects the conditioning of the resulting factorization and thus its stability. In Section 3, we highlight stability results that have appeared in the literature.

We also observe that null-space factorizations allow us to predict the level of fill a priori, which is not the case for a general sparse symmetric indefinite solver that employs numerical pivoting for stability. However, as a null-space factorization predetermines an elimination ordering, it may or may not produce sparser factors than a general sparse solver.

Section 4 presents our numerical experiments that explore the effectiveness of null-space factorizations and compare their performance with that of a general sparse solver. We consider a range of problems, both artificial and from real-world applications, and investigate the stability of the factorization and the sparsity of the factors. We give concluding remarks in Section 5.

2 | NULL-SPACE METHODS AS A FACTORIZATION

Suppose we are given a matrix $Z \in \mathbb{R}^{n\times(n-m)}$ whose columns form a basis for the null space of B, that is, $BZ = 0$. Suppose additionally that we have a particular solution for the second equation, that is, a vector $\hat{\mathrm{x}}$ such that

$$B\hat{\mathrm{x}} = \mathrm{g}.$$

Then solving Equation 1 is equivalent to solving

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \bar{\mathrm{x}} \\ \mathrm{y} \end{bmatrix} = \begin{bmatrix} \mathrm{f} - A\hat{\mathrm{x}} \\ 0 \end{bmatrix},$$

where $\mathrm{x} = \hat{\mathrm{x}} + \bar{\mathrm{x}}$. The second equation in this system is equivalent to finding a vector $\mathrm{z} \in \mathbb{R}^{(n-m)}$ such that $\bar{\mathrm{x}} = Z\mathrm{z}$. Substituting this into the first equation we have

$$\begin{aligned} & AZ\mathrm{z} + B^T\mathrm{y} = \mathrm{f} - A\hat{\mathrm{x}} \\ \iff\ & Z^TAZ\mathrm{z} = Z^T(\mathrm{f} - A\hat{\mathrm{x}}) \end{aligned} \tag{2}$$

Therefore, by solving the reduced system (Equation 2), we can straightforwardly recover $\mathrm{x} = \hat{\mathrm{x}} + Z\mathrm{z}$. We can then obtain y by solving the overdetermined system $Ax + B^Ty = f$. This is a null-space method, which we summarise as Algorithm 1.

Algorithm 1 Null-space method for solving Equation 1

Choose Z so that its columns form a basis for the null space of B.
Find $\hat{\mathrm{x}}$ such that $B\hat{\mathrm{x}} = \mathrm{g}$.
Solve $Z^TAZ\mathrm{z} = Z^T(\mathrm{f} - A\hat{\mathrm{x}})$
Set $\mathrm{x} = Z\mathrm{z} + \hat{\mathrm{x}}$
Find y such that $B^T\mathrm{y} = \mathrm{f} - A\mathrm{x}$

The approach outlined in Algorithm 1 is well known. What is less well known is that we can interpret this as a matrix factorization. Consider again Equation 1. The primal variable x can be expressed in the form

$$\mathrm{x} = Z\mathrm{x}_N + Y\mathrm{x}_R, \tag{3}$$

where $Y \in \mathbb{R}^{n\times m}$ is chosen so that $\begin{bmatrix} Z & Y \end{bmatrix}$ spans $\mathbb{R}^n$. Thus,

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} Y & Z & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \mathrm{x}_R \\ \mathrm{x}_N \\ \mathrm{y} \end{bmatrix} = \begin{bmatrix} \mathrm{f} \\ \mathrm{g} \end{bmatrix},$$

and hence,

$$\begin{bmatrix} Y^T & 0 \\ Z^T & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} Y & Z & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \mathrm{x}_R \\ \mathrm{x}_N \\ \mathrm{y} \end{bmatrix} = \begin{bmatrix} Y^T & 0 \\ Z^T & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathrm{f} \\ \mathrm{g} \end{bmatrix},$$

and so,

$$\begin{bmatrix} Y^TAY & Y^TAZ & Y^TB^T \\ Z^TAY & Z^TAZ & 0 \\ BY & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{x}_R \\ \mathrm{x}_N \\ \mathrm{y} \end{bmatrix} = \begin{bmatrix} Y^T & 0 \\ Z^T & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathrm{f} \\ \mathrm{g} \end{bmatrix}. \tag{4}$$

It is clear that this is a matrix representation of Algorithm 1. First, $\hat{\mathrm{x}} = Y\mathrm{x}_R$ is found by solving the linear system $BYx_R = g$. Then, the component x_N of x in the null space of B is found by solving the linear system

$$Z^TAZ\mathrm{x}_N = Z^T(\mathrm{f} - AY\mathrm{x}_R) = Z^T(\mathrm{f} - A\hat{\mathrm{x}}).$$

Finally, y is recovered by solving

$$Y^TB^T\mathrm{y} = Y^T\mathrm{f} - Y^TAY\mathrm{x}_R - Y^TAZ\mathrm{x}_N = Y^T(\mathrm{f} - A\mathrm{x}).$$

Note that the matrix $\begin{bmatrix} Y^T & 0 \\ Z^T & 0 \\ 0 & I \end{bmatrix}$ is square and nonsingular and so, using Equations 3 and 4, we can rewrite Equation 1 as

$$\begin{bmatrix} Y^T & 0 \\ Z^T & 0 \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} Y^TAY & Y^TAZ & Y^TB^T \\ Z^TAY & Z^TAZ & 0 \\ BY & 0 & 0 \end{bmatrix} \begin{bmatrix} Y & Z & 0 \\ 0 & 0 & I \end{bmatrix}^{-1} \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \end{bmatrix} = \begin{bmatrix} \mathrm{f} \\ \mathrm{g} \end{bmatrix}.$$

Thus, the factorization

$$\mathcal{A} = \begin{bmatrix} \begin{bmatrix} Y^T \\ Z^T \end{bmatrix}^{-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} Y^TAY & Y^TAZ & Y^TB^T \\ Z^TAY & Z^TAZ & 0 \\ BY & 0 & 0 \end{bmatrix} \begin{bmatrix} \begin{bmatrix} Y & Z \end{bmatrix}^{-1} & 0 \\ 0 \; 0 & I \end{bmatrix} \tag{5}$$

is a block LTL^T factorization, with L lower triangular and T reverse block triangular, that is equivalent to the null-space method. Because there are infinitely many potential bases Y, Z, this factorization is nonunique and the main difficulty of the null-space method is choosing these bases. In the Sections 2.1 to 2.5, we discuss some special cases that have been proposed in the literature.

2.1 | Factorizations based on the fundamental basis

One way of fixing Y in Equation 3 is to extend B^T to an $n \times n$ nonsingular matrix $\begin{bmatrix} B^T & V^T \end{bmatrix}$. If we choose Y and Z satisfying

$$\begin{bmatrix} B \\ V \end{bmatrix} \begin{bmatrix} Y & Z \end{bmatrix} = I$$

then it is easy to see that $BZ = 0$ and $BY = I$. The factorization in Equation 5 reduces to

$$\mathcal{A} = \begin{bmatrix} B^T & V^T & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} Y^TAY & Y^TAZ & I \\ Z^TAY & Z^TAZ & 0 \\ I & 0 & 0 \end{bmatrix} \begin{bmatrix} B & 0 \\ V & 0 \\ 0 & I \end{bmatrix}.$$

This factorization was given by Fletcher et al.,[19, eq. 3.6] who described it as "readily observed (but not well known)."

2.1.1 | A first null-space factorization

Suppose we have a nonsingular subset of m columns of B. We may then write, without loss of generality, $B = \begin{bmatrix} B_1 & B_2 \end{bmatrix}$, where $B_1 \in \mathbb{R}^{m \times m}$ is nonsingular. If, as suggested by Fletcher et al,[19] we make the choice of $V = \begin{bmatrix} 0 & I \end{bmatrix}$, then

$$\begin{bmatrix} B \\ V \end{bmatrix}^{-1} = \begin{bmatrix} B_1 & B_2 \\ 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} B_1^{-1} & -B_1^{-1}B_2 \\ 0 & I \end{bmatrix} \left(= \begin{bmatrix} Y & Z \end{bmatrix} \right).$$

This gives us the bases

$$Z_f = \begin{bmatrix} -B_1^{-1}B_2 \\ I \end{bmatrix}, \qquad Y_f = \begin{bmatrix} B_1^{-1} \\ 0 \end{bmatrix}. \tag{6}$$

This choice for Z is often called the fundamental basis,[1, section 6] and we consequently label it Z_f.

Substituting Equation 6 into Equation 5 gives the factorization

$$\mathcal{A} = \begin{bmatrix} B_1^T & 0 & 0 \\ B_2^T & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} B_1^{-T}A_{11}B_1^{-1} & B_1^{-T}X^T & I \\ XB_1^{-1} & N & 0 \\ I & 0 & 0 \end{bmatrix} \begin{bmatrix} B_1 & B_2 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}, \tag{7}$$

TABLE 1 Cost of forming Factorization 1 assuming that B_1, B_2 chosen. mat-mat, and mat-add denote a sparse matrix–matrix product and sparse matrix addition

Operation	Size	Details	How many?
sparse solve	$m \times m$ matrix with $n-m$ right hand sides	$B_1\mathcal{B} = B_2$	1
mat-mat	$((n-m)\times m)\times(m\times m)$	$\mathcal{B}^T A_{11}$	1
mat-mat	$((n-m)\times m)\times(m\times(n-m))$	$X\mathcal{B}$, $\mathcal{B}^T A_{12}$	2
mat-add	$(n-m)\times m$	forming X	1
mat-add	$(n-m)\times(n-m)$	forming N	2
factorize	$(n-m)\times(n-m)$ (possibly sparse)	N	1

where

$$N = Z_f^T A Z_f, \tag{8}$$

denotes the $(n-m)\times(n-m)$ null-space matrix and

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{and} \quad X = Z_f^T \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix},$$

with $A_{11} \in R^{m\times m}$. It is easy to see that Equation 7 is equivalent to

$$\mathcal{A} = \underbrace{\begin{bmatrix} I & 0 & 0 \\ B_2^T B_1^{-T} & I & 0 \\ 0 & 0 & I \end{bmatrix}}_{\mathcal{L}_1} \underbrace{\begin{bmatrix} A_{11} & X^T & B_1^T \\ X & N & 0 \\ B_1 & 0 & 0 \end{bmatrix}}_{\mathcal{T}_1} \underbrace{\begin{bmatrix} I & B_1^{-1}B_2 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}}_{\mathcal{L}_1^T}. \tag{9}$$

Indeed, this LTL^T factorization appeared in the survey paper by Benzi et al.,[1, eq. 10.35] where it was attributed to a personal communication from Michael Saunders and was described as being "related to the null-space method." We will refer to this decomposition as Factorization 1. The factor $\mathcal{L}_1$ is well conditioned provided B_1 is chosen appropriately; this is discussed in Section 3. Factorization 1 is then a stable factorization of $\mathcal{A}$ and the diagonal blocks of $\mathcal{T}_1$ (that is, B_1, N, and B_1^T) accurately reflect the condition of the full system.

In practice, once we have found the $m\times(n-m)$ matrix $\mathcal{B}$ by solving the sparse system

$$B_1\mathcal{B} = B_2, \tag{10}$$

the only additional calculations required to generate Factorization 1 are matrix–matrix products with $\mathcal{B}$, some matrix additions, and a factorization of the null-space matrix N. This is due to the relationships $X = -\mathcal{B}^T A_{11} + A_{21}$ and $N = -X\mathcal{B} - \mathcal{B}^T A_{12} + A_{22}$. The overall cost is summarised in Table 1. We point out that, because the subblocks involved are sparse, it is not possible to give explicit operation counts in Table 1, as these would depend on the density of the submatrices. Note further that, in some applications (for example, where N is large and dense) it may not be possible to explicitly compute N and form its factorization; in this case, it is necessary to use an iterative solver.

Having formed Factorization 1, Table 2 gives the costs of using it to solve Equation 1. Again, note that exact operation counts are not possible. Table 2 describes two variants: implicit and explicit. In the explicit version, we store the off-diagonal matrices (X, $\mathcal{B}$) that are formed in the construction of N and apply them via matrix-vector products to solve Equation 1. In the implicit version, we discard the off-diagonal matrices and recompute them as needed. This is computationally more expensive but saves storing the potentially dense matrices X and $\mathcal{B}$ of size $m\times(n-m)$. It is clear that the implicit version is exactly equivalent to the null-space method as presented in Algorithm 1.

The costs in Tables 1 and 2 are upper bounds, and they may be reduced in certain circumstances. For example, as we discuss in Section 3 below, it is usual to find B_1, B_2 by forming an LU factorization of B^T

$$B^T = (PLP^T)(PUQ) = \left(P\begin{bmatrix} L_a & 0 \\ L_b & I \end{bmatrix}P^T\right)\left(P\begin{bmatrix} U_a \\ 0 \end{bmatrix}Q\right),$$

where L_a is lower triangular, U_a is upper triangular, and P and Q are permutation matrices. Then $B_1^{-1}B_2 = PL_a^{-T}L_b^T P^T$, and so this can be calculated without reference to U_a, although L_b is needed and is likely to be less sparse than B_2. Furthermore, in this case $Z = L^{-T}P^T\begin{bmatrix} I \\ 0 \end{bmatrix}$, so Z^TAZ can also be formed efficiently. See, for example, Fletcher et al.[19] for more details.

TABLE 2 Cost of using Factorization 1 to solve Equation 1

	Operation	Size	Details	How many?
Explicit	sparse solve	$m \times m$ matrix	$B_1 x = y, B_1^T \mathrm{x} = \mathrm{y}$	2
	sparse solve	$(n-m) \times (n-m)$ lower triangular	Factors of N	2
	mat-vec	$(n-m) \times m$	$\mathcal{B}\mathrm{x}, Xx$	2
	mat-vec	$m \times m$	$A_{11}x$	1
	mat-vec	$m \times (n-m)$	$\mathcal{B}^T\mathrm{x}, X^T x$	2
	vec-add	$n-m$		2
	vec-add	m		3
Implicit			As explicit[a] plus:	
	sparse solve	$m \times m$ matrix	$B_1\mathrm{x} = \mathrm{y}, B_1^T\mathrm{x} = \mathrm{y}$ (in mult with $\mathcal{B}, \mathcal{B}^T, X, X^T$)	4
	mat-vec	$(n-m) \times m$		1
	mat-vec	$m \times m$		1
	mat-vec	$m \times (n-m)$		1
	vec-add	$n-m$		1

Note. mat-vec denotes the product of a sparse matrix with a vector, and vec-add denotes the addition of two vectors.

[a]The numbers of solves and matrix-vector products will be the same, but the matrices in the matrix-vector products will generally be sparser in the implicit case.

2.1.2 | A factorization due to Lungten, Schilders, and Maubach

Assume now that A is symmetric and positive semidefinite so that N is symmetric positive definite and a Cholesky factorization of the form $N = L_2 L_2^T$ exists, where L_2 is lower triangular. Then we can decompose the reverse triangular $\mathcal{T}_1$ matrix in Equation 9 as

$$\mathcal{T}_1 = \begin{bmatrix} A_{11} & X^T & B_1^T \\ X & N & 0 \\ B_1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} I & 0 & L_A \\ 0 & L_2 & X \\ 0 & 0 & B_1 \end{bmatrix} \begin{bmatrix} -D_A & 0 & I \\ 0 & I & 0 \\ I & 0 & 0 \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & L_2^T & 0 \\ L_A^T & X^T & B_1^T \end{bmatrix},$$

where

$$A_{11} = L_A - D_A + L_A^T, \tag{11}$$

with L_A a strictly lower triangular matrix and D_A a diagonal matrix. Combining the outer matrices here with the outer matrices in Equation 9 yields the alternative, but equivalent, LTL^T factorization

$$\mathcal{A} = \underbrace{\begin{bmatrix} I & 0 & L_A \\ B_2^T B_1^{-T} & L_2 & K \\ 0 & 0 & B_1 \end{bmatrix}}_{\mathcal{L}_2} \underbrace{\begin{bmatrix} -D_A & 0 & I \\ 0 & I & 0 \\ I & 0 & 0 \end{bmatrix}}_{\mathcal{T}_2} \underbrace{\begin{bmatrix} I & B_1^{-1}B_2 & 0 \\ 0 & L_2^T & 0 \\ L_A^T & K^T & B_1^T \end{bmatrix}}_{\mathcal{L}_2}, \tag{12}$$

where $K = X + B_2^T B_1^{-T} L_A$. This factorization was recently proposed both for use as a direct method and as the basis of a preconditioner for an iterative method by Lungten et al.[15] We refer to it as Factorization 2 or the LSM factorization.

Note that forming Equation 12 is more expensive than Equation 9, as it requires one more matrix-matrix multiply of $\mathcal{B}^T$ (recall Equation 10) with an $m \times m$ matrix and one more $(n-m) \times m$ matrix addition, both coming from the formation of K. In terms of applying Equation 12 explicitly, one matrix-vector multiply with A_{11} is replaced by matrix-vector multiplies with its strictly upper, lower, and diagonal parts, and two extra $m \times m$ matrix additions; there is a similar increase in cost when applying the factorization implicitly. This suggests that, in terms of the computational cost, Factorization 1 is preferable; we perform tests with both versions in Section 3.

Lungten et al.[15] focus on problems for which the nonsingular matrix B_1 is also upper triangular (or it is easy to transform the problem into this form). In this case, if Factorization 2 is formed via an modified Cholesky algorithm, they show that, for the dense case, it takes

$$\frac{1}{3}(n^3 - m^3) + \frac{1}{2}(n^2 - 7m^2) - \frac{1}{6}(5n + m) + nm(n - m + 4)$$

flops to factorize the saddle point matrix $\mathcal{A}$.

2.1.3 | The Schilders factorization

We next consider the relationship of Factorizations 1 and 2 to the decomposition known in the literature as the Schilders factorization. Dollar et al.[10] were interested in developing constraint preconditioners for symmetric systems as in Equation 1. Such preconditioners represent the blocks B exactly but approximate the (1,1) block A. Dollar et al. choose symmetric matrices $E_1 \in \mathbb{R}^{m\times m}$, $E_2 \in \mathbb{R}^{(n-m)\times(n-m)}$ (E_2 nonsingular), together with matrices $F_1 \in \mathbb{R}^{m\times m}$, $F_2 \in \mathbb{R}^{(n-m)\times(n-m)}$ (F_2 nonsingular), and $M \in \mathbb{R}^{(n-m)\times m}$. To obtain an inexpensive preconditioner, E_2, F_2 are chosen so that they are easy to invert. The Schilders factorization is then given by

$$\begin{bmatrix} A_{11} & A_{21}^T & B_1^T \\ A_{21} & A_{22} & B_2^T \\ B_1 & B_2 & 0 \end{bmatrix} = \begin{bmatrix} B_1^T & 0 & F_1 \\ B_2^T & F_2 & M \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} E_1 & 0 & I \\ 0 & E_2 & 0 \\ I & 0 & 0 \end{bmatrix} \begin{bmatrix} B_1 & B_2 & 0 \\ 0 & F_2^T & 0 \\ F_1^T & M^T & I \end{bmatrix}, \tag{13}$$

where

$$\begin{aligned} A_{11} &= F_1B_1 + B_1^TF_1^T + B_1^TE_1B_1 \\ A_{21} &= B_2^TF_1^T + MB_1 + B_2^TE_1B_2 \\ A_{22} &= F_2E_2F_2^T + MB_2 + B_2^TM^T + B_2^TE_1B_2. \end{aligned}$$

Note that the (1,1) block A is implicitly defined by the choices of E_i, F_i. Nevertheless, we can use this construction to give a factorization for a given A.

One possible choice for E_1, F_1 is

$$E_1 = -B_1^{-T}D_AB_1^{-1}, \qquad F_1 = L_AB_1^{-1},$$

for the D_A, L_A in Equation 11. The matrices M, E_2, and F_2 are then given by the relations

$$\begin{aligned} M &= \left(A_{21} - B_2^TF_1^T - B_2^TE_1B_2\right)B_1^{-1}, \\ F_2E_2F_2^T &= MB_2 + B_2^TM^T + B_2^TE_1B_2 - A_{22}. \end{aligned}$$

With these choices, transferring a factor of the block diagonal matrix with diagonal blocks B_1^T, I, and B_1^{-1} from the left outer matrix to the central matrix in Equation 13 again gives Factorization 2.

The original Schilders factorization,[14] of which the formulation 13 is a generalization, was given only for matrices for which B_1 is upper triangular and used the choice

$$E_1 = \texttt{diag}\left(B_1^{-T}A_{11}B_1^{-1}\right), \quad F_1 = B_1^T\texttt{lower}\left(B_1^{-T}A_{11}B_1^{-1}\right),$$

where `diag()` and `lower()` denote the diagonal and strictly lower triangular parts of a matrix, respectively. Again, it is straightforward to show the equivalence of this factorization to Factorization 2. Generating this factorization is clearly significantly more work than Equation 9, not least because we are unable to reuse the matrix $\mathcal{B}$ in forming the subblocks.

There are, of course, other ways of rearranging the decompostion given by Equation 9. Dollar et al.[20] give a list of forms the factorization can take. As already observed, their focus was on implicit factorizations of constraint preconditioners and only five of their factorizations are applicable in the case of an arbitrary symmetric (1,1) block.

2.2 | Relationship to the Schur complement factorization

A commonly used block LDL^T factorization for generalized saddle point systems (which have a negative definite (2,2) block) where A is nonsingular is

$$\begin{bmatrix} A & B^T \\ B & -C \end{bmatrix} = \begin{bmatrix} I & 0 \\ BA^{-1} & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & -(C + BA^{-1}B^T) \end{bmatrix} \begin{bmatrix} I & A^{-1}B^T \\ 0 & I \end{bmatrix}, \tag{14}$$

see, for example, Benzi et al.[1, Equation 3.9] Approximating the terms of this factorization has proved an effective strategy for developing preconditioners for saddle point systems. It can also be used to develop another factorization that is equivalent to the null-space method. First, note that

$$\mathcal{A} = \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & I \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} A_{11} & B_1^T & A_{12} \\ B_1 & 0 & B_2 \\ A_{21} & B_2^T & A_{22} \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & I \\ 0 & I & 0 \end{bmatrix},$$

where we are again assuming, without loss of generality, that B_1 is a nonsingular $m \times m$ subblock of B. Applying Equation 14 with $C = A_{22}$, we obtain

$$\mathcal{A} = \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & I \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ B_2^T B_1^{-T} & A_{21}B_1^{-1} - B_2^T B_1^{-T} A_{11} B_1^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & B_1^T & 0 \\ B_1 & 0 & 0 \\ 0 & 0 & S \end{bmatrix} \begin{bmatrix} I & 0 & B_1^{-1}B_2 \\ 0 & I & B_1^{-T}A_{12} - B_1^{-T}A_{11}B_1^{-1}B_2 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & I \\ 0 & I & 0 \end{bmatrix}$$
$$= \begin{bmatrix} I & 0 & 0 \\ B_2^T B_1^{-T} & XB_1^{-1} & I \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} A_{11} & B_1^T & 0 \\ B_1 & 0 & 0 \\ 0 & 0 & S \end{bmatrix} \begin{bmatrix} I & B_1^{-1}B_2 & 0 \\ 0 & B_1^{-T}X^T & I \\ 0 & I & 0 \end{bmatrix}.$$

Here, S denotes the Schur complement, which satisfies

$$\begin{aligned} S &= A_{22} - \begin{bmatrix} A_{21} & B_2^T \end{bmatrix} \begin{bmatrix} A_{11} & B_1^T \\ B_1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} A_{12} \\ B_2 \end{bmatrix} \\ &= A_{22} - \begin{bmatrix} A_{21} & B_2^T \end{bmatrix} \begin{bmatrix} 0 & B_1^{-1} \\ B_1^{-T} & -B_1^{-T}A_{11}B_1^{-1} \end{bmatrix} \begin{bmatrix} A_{12} \\ B_2 \end{bmatrix} \\ &= A_{22} - B_2^T B_1^{-T} A_{12} - A_{21}B_1^{-1}B_2 + B_2^T B_1^{-T} A_{11} B_1^{-1} B_2 \\ &= Z_f^T A Z_f = N. \end{aligned}$$

It follows that the null-space matrix in Equation 8 is the Schur complement for an alternative blocking of the matrix, and we have the factorization

$$\mathcal{A} = \begin{bmatrix} I & 0 & 0 \\ B_2^T B_1^{-T} & XB_1^{-1} & I \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} A_{11} & B_1^T & 0 \\ B_1 & 0 & 0 \\ 0 & 0 & N \end{bmatrix} \begin{bmatrix} I & B_1^{-1}B_2 & 0 \\ 0 & B_1^{-T}X^T & I \\ 0 & I & 0 \end{bmatrix}. \tag{15}$$

Again, this can be derived from Equation 9 by simply noting that the reverse triangular matrix $\mathcal{T}_1$ is equal to the product

$$T = \begin{bmatrix} I & 0 & 0 \\ 0 & XB_1^{-1} & I \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} A_{11} & B_1^T & 0 \\ B_1 & 0 & 0 \\ 0 & 0 & N \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & B_1^{-T}X^T & I \\ 0 & I & 0 \end{bmatrix}.$$

This factorization is therefore not "new"—and it is significantly more expensive to form than Equation 9—but it highlights the connection between Schur complement methods and null-space factorizations. In particular, given the success in finding approximations to the Schur complement using techniques from functional analysis (see, e.g., Mardel et al.[21]) it is hoped that viewing the null-space matrix this way could yield alternative preconditioners for certain classes of saddle point systems.

2.3 | Connection with Cholesky decomposition

Schilders developed his original factorization and the subsequent variant (Equation 12), by considering what he terms a *micro-factorization*. In this formulation, the matrix in Equation 1 is reordered by pairing every entry on the diagonal of the (1,1) block A with a corresponding nonzero entry in the constraint block B so that (after permutations) the entries on the diagonal form micro saddle point systems. This is known as a tiling in the optimization community. Below is an illustrative example of this ordering for $n = 3, m = 2$:

$$P_1^T \mathcal{A} P_1 = \widetilde{\mathcal{A}} = \begin{bmatrix} a_{11} & b_{11} & a_{12} & b_{21} & a_{13} & b_{31} & a_{14} & a_{15} \\ b_{11} & 0 & b_{12} & 0 & b_{13} & 0 & b_{14} & b_{15} \\ a_{21} & b_{12} & a_{22} & b_{22} & a_{23} & b_{32} & a_{24} & a_{25} \\ b_{21} & 0 & b_{22} & 0 & b_{23} & 0 & b_{24} & b_{25} \\ a_{31} & b_{13} & a_{32} & b_{23} & a_{33} & b_{33} & a_{34} & a_{35} \\ b_{31} & 0 & b_{32} & 0 & b_{33} & 0 & b_{34} & b_{35} \\ a_{41} & b_{14} & a_{42} & b_{24} & a_{43} & b_{34} & a_{44} & a_{54} \\ a_{51} & b_{15} & a_{52} & b_{25} & a_{53} & b_{35} & a_{45} & a_{55} \end{bmatrix}.$$

Note that there is no requirement for a_{ij} to be combined with b_{ij} and b_{ji}; instead, a suitable pairing that preserves sparsity and is numerically stable should be chosen—see Section 3 for further discussion. This is an example of a constrained ordering.[22-24]

Because the entries on the (block) diagonal are now micro saddle point systems (chosen to be nonsingular) a modified sparse Cholesky code can be used to solve this system, and it is guaranteed (at least in exact arithmetic) that this will not break

down.[14, section 3] By this process, a factorization $\tilde{\mathcal{A}} = LDL^T$ can be computed, where D has 1×1 and 2×2 blocks on the diagonal in the appropriate places. Furthermore, uniqueness results in, for example, Maubach et al.,[12] show that the factors generated by the Cholesky process will be equivalent to those generated by the (macro) factorizations described earlier in this section.

In addition to the work by Schilders et al.,[12,14,15] this approach has been considered by Forsgren et al.[25] (whose focus was on inertia control), Gould,[17] and de Niet et al.[9]; each of these works, to varying degrees, made the connection between the microfactorization and the null-space method.

2.4 | The antitriangular factorization

An alternative matrix factorization can be obtained by assuming we have a QR factorization

$$B^T = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}, \tag{16}$$

where $Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ is an $n \times n$ orthogonal matrix and R is an $m \times m$ upper triangular and nonsingular matrix. Then Q_1 spans the range of B^T and Q_2 spans the null space of B. We can, therefore, substitute $Y = Q_1$ and $Z = Q_2$ into the factorization given by Equation 5 to obtain

$$\mathcal{A} = \underbrace{\begin{bmatrix} Q_1 & Q_2 & 0 \\ 0 & 0 & I \end{bmatrix}}_{\mathcal{L}_3} \underbrace{\begin{bmatrix} Q_1^TAQ_1 & Q_1^TAQ_2 & R \\ Q_2^TAQ_1 & Q_2^TAQ_2 & 0 \\ R^T & 0 & 0 \end{bmatrix}}_{\mathcal{T}_3} \underbrace{\begin{bmatrix} Q_1^T & 0 \\ Q_2^T & 0 \\ 0 & I \end{bmatrix}}_{\mathcal{L}_3}. \tag{17}$$

We refer to this as Factorization 3 or the AT (for antitriangular) factorization.

The reverse triangular matrix $\mathcal{T}_3$ in Equation 17 appeared in the proof of theorem 2.1 in Keller et al.[18] which established eigenvalue bounds for Equation 1 preconditioned with a constraint preconditioner, although the link to a factorization of $\mathcal{A}$ does not appear to have been made there. Recently, Pestana et al.[13] derived Factorization 3 in the case A is symmetric and B is of full rank and showed that it is (up to a permutation) the representation of the antitriangular factorization of Mastronardi et al.[11] applied to a saddle point system. They also gave a flop count for forming the factorization in the dense case. The work is dominated by performing the QR factorization (Equation 16), with additional work being two $n \times n$ matrix–matrix products and a subsequent factorization of $Q_2^TAQ_2$.

It is clear from the above formulation that—provided a rank-revealing QR is applied—Factorization 3 is well defined for a B with linearly dependent rows and also for nonsymmetric A.

An advantage of having an orthogonal null basis is that the reduced system (Equation 4) is guaranteed to be well conditioned if A is. However, even for sparse problems, Q_1 and Q_2 may be rather dense. The explicit version of the factorization requires storing $Q_1, Q_2, Q_1^TAQ_2$ and the lower triangular part of $Q_1^TAQ_1$ in addition to keeping the factors of $Q_2^TAQ_2$. For large systems this incurs prohibitive storage costs, see Section 3. Sparse QR routines—for example, SuiteSparseQR[26]—normally allow the user to compute the action of Q_i on a vector using the stored Householder reflectors, and this facility is the preferred option here.

Pestana et al.[13] give a complexity analysis of the factorization in Equation 17 in the dense case and show that the number of flops required is

$$8mn^2 - 2m^2 - \frac{2}{3}m^3.$$

An alternative way of obtaining the factorization in Equation 17 which again focuses on the dense case, is given by Mastronardi et al.,[27] and the number of flops there is dependent on the size of m relative to n; see Pestana et al.[13, Table 2.1] Details of an efficient implementation that uses Level 3 BLAS are given by Bujanović et al.[28]

2.5 | Other bases and converting between factorizations

Consider again the general factorization as given in Equation 5. As already observed, there is an infinite number of potential matrices Z (whose columns span the null space of B) and Y (whose columns are such that $\begin{bmatrix} Y & Z \end{bmatrix}$ span $\mathbb{R}^n$).

In fact, for any given basis matrix Z and for any nonsingular matrix $G \in \mathbb{R}^{(n-m)\times(n-m)}$, another factorization that is equivalent to Equation 5—and hence Algorithm 1—is

$$\mathcal{A} = \begin{bmatrix} \begin{bmatrix} \widehat{Y}^T \\ G^TZ^T \end{bmatrix}^{-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \widehat{Y}^TA\widehat{Y} & \widehat{Y}^TAZG & \widehat{Y}^TB^T \\ G^TZ^TA\widehat{Y} & G^TZ^TAZG & 0 \\ B\widehat{Y} & 0 & 0 \end{bmatrix} \begin{bmatrix} \begin{bmatrix} \widehat{Y} & ZG \end{bmatrix}^{-1} & 0 \\ 0\ 0 & I \end{bmatrix}, \tag{18}$$

where $\widehat{Y}$ is a matrix such that $\begin{bmatrix} \widehat{Y} & ZG \end{bmatrix}$ spans $\mathbb{R}^n$.

To highlight this fact, we will show explicitly how we can transform Factorization 1 into Factorization 3. Consider the fundamental basis Z_f (Equation 6), and assume that we have its "skinny" QR factorization $Z_f = Q_2 R_2$. Then, we see that

$$Q_2 = Z_f R_2^{-1} \in \mathbb{R}^{n\times(n-m)},$$

the columns of which clearly span the null-space of B. Now, let $B^T = Q_1 R_1$ be the skinny QR factorization of B^T. Because B is of full rank, we have obtained two matrices, $Q_1 \in \mathbb{R}^{m\times n}$ that spans the range of B^T and $Q_2 \in \mathbb{R}^{n\times(n-m)}$ that spans the null-space of B. We, therefore, know that the $n \times n$ matrix $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ spans $\mathbb{R}^n$.

Substituting $G = R_2^{-1}$ and $\hat{Y} = Q_1$ into Equation 18, we see that these choices transform Factorization 1 (Equation 9) into Factorization 3 (Equation 17). This construction may not have any practical use, but it is theoretically satisfying to explicitly connect two seemingly different methods. In the same way, by an appropriate choice of G and $\hat{Y}$, we can transform any null-space factorization into any other.

3 | STABILITY AND SPARSITY PROPERTIES

For a factorization to be effective for solving sparse linear systems, it must be stable in the classical sense[29, chapter 7] and the factors must be sparse. In this section, we bring together results from the literature that concern the viability of the factorizations described in Section 2 as the basis of a sparse direct solver.

Consider Factorization 3, the antitriangular factorization. This was shown to be backward stable by Mastronardi et al.[11] Stability is a direct consequence of the fact that the only calculation needed is a stable QR factorization, see also Gill et al.,[16] and Fletcher et al.[19] The penalty for this stability is that the factors may not be sparse, even when using a sparse QR factorization routine, as the Q matrices can fill in significantly; this is illustrated in Section 4.1.

An alternative approach is to choose the permutations in Factorizations 1 and 2 in such a way that we guarantee a stable factorization. Forsgren et al.[25]—thinking in terms of a microfactorization—describe a pivoting strategy to do this. Their method was refined further by Gould.[17]

Another way to ensure stability is to perform an LU factorization on B^T (incorporating pivoting to ensure the L factor is well conditioned). If we obtain B_1 using this, then $B_2^T B_1^{-T}$ is bounded, and the factorization can be proved to be backward stable, whereas the forward error in x and y are proportional to $\kappa([B^T V^T])\kappa(Z^T A Z)$ and $\kappa([B^T V^T])^2\kappa(Z^T A Z)$, respectively, where κ denotes the condition number—see Fletcher et al.[19] for details; an alternative description is given by Gill et al.[30] This is the approach we adopt in the tests in Section 4.

A weakness of a general solver, such as an LDL^T factorization, is that, due to the need for pivoting to preserve stability, the fill in the computed factors may be significantly higher than predicted on the basis of the sparsity pattern of the matrix alone. This use of numerical pivoting not only leads to a higher operation count and denser factors but also prohibits the exploitation of parallelism (and significantly complicates the code development). On the other hand, null-space factorizations reduce the solution of Equation 1 to solving a positive definite matrix, and as such, they give a direct method with a predictable level of fill without pivoting (other than to find B_1). This is a potentially attractive feature. However, by identifying a nonsingular subblock B_1 of B we are essentially predetermining an ordering that may or may not produce sparser factors than that used by a general sparse indefinite direct solver.

Instead of choosing B_1 to give stability at the expense of sparsity, we can alternatively choose B_1 so that the fundamental basis Z_f is sparse, for example, by ensuring B_1^{-1} is sparse. Pinar et al.[31] describe several ways to do this using methods based on graph matchings and hypergraph partitioning. However, such approaches are reportedly time consuming, and no results about the stability of the resulting decomposition are given. Murray[32] describes a method for picking Z so that $Z^T A Z$ is diagonal. However, to quote Fletcher,[33, section 10.1] this "may be doubtful in terms of stability." Note also that B_1^{-1} can only be sparse when the graph of B_1 is disconnected,[34, section 12.6] which may not be possible to achieve in certain applications.

Favourable sparsity and stability properties are achievable for certain classes of matrices $\mathcal{A}$. In particular, de Niet et al.[9] prove some strong results for $\mathcal{F}$ matrices. An $\mathcal{F}$ matrix is a saddle point matrix (Equation 1) in which A is symmetric positive definite and B is a gradient matrix, that is, B has at most two entries per row, and if there are two entries, their sum is zero. Such matrices appear naturally in certain formulations of fluid flow problems (see, for example, Arioli et al.[35]) and also in electronic simulations, where they arise as the incidence matrix of the graph representing the circuit.[36] The original Schilders factorization[14] was developed specifically for solving systems with this structure.

De Niet et al. find the basis B_1 implicitly by considering a microfactorization (recall Section 2.3). They pair entries in A with entries in B in such a way as to ensure the stability of the factorization. Moreover, de Niet et al. show that the number of nonzeros

in the symbolic factorization of $F(A) \cup F(BB^T)$, where $F(\cdot)$ denotes the sparsity pattern, "provides a reasonable estimate" of the number of nonzeros in the factorization. We perform tests on this class of matrices in Section 4.3.

4 | NUMERICAL RESULTS

With the exception of the special case of $\mathcal{F}$ matrices, the literature focuses on proving either stability, with little said about sparsity or vice versa. In this section, we present results using Factorizations 1, 2, and 3—that is, Equations 9, 12, and 17, respectively—as the basis of a direct method. Our intention is to demonstrate the stability and sparsity of these factorizations when applied to a range of problems, and we compare their performance with that of a general sparse symmetric indefinite solver. We concentrate on the solution of a single saddle-point system and do not exploit the potential advantages null-space factorizations can offer over a general indefinite solver in terms of updating the factors after a row is added to (or removed from) $\mathcal{A}$.

We perform tests using MATLAB Version R2014a. We factorize the positive definite null-space matrix N using `cholmod`[37] and use the `SuiteSparseQR` algorithm,[26] both from Tim Davis' SuiteSparse package and applied via a MATLAB interface. To find an invertible subset of B, we use the package LUSOL[38] with partial pivoting (with a pivot tolerance of 1.9, as suggested for basis repair in the LUSOL documentation) to perform the sparse factorization of B^T. LUSOL is called via a MATLAB interface.[39]

For comparison, results for the MATLAB command `ldl` (with default settings) are also given. `ldl` computes an LDL^T factorization (where L is unit lower triangular and D is block diagonal with 1×1 and 2×2 blocks on the diagonal) of a sparse symmetric indefinite matrix by employing the direct solver[40] `MA57` from the HSL mathematical software library.[41] `MA57` first computes a sparsity-preserving ordering of $\mathcal{A}$ and a scaling that is designed to help with numerical stability. It does not explicitly exploit the block structure of $\mathcal{A}$ but is designed to efficiently and robustly solve general symmetric indefinite sparse linear systems. To ensure a fair comparison in our experiments, we always prescale $\mathcal{A}$ using the scaling for $\mathcal{A}$ calculated by `ldl`. We recognise, however, that, in practice, it may be more effective to scale the A and B blocks separately as x and y may represent very different types of quantity.

To measure stability, we report the scaled backwards error for Equation 1 given by

$$\|\mathcal{A}\mathbf{w} - \mathbf{b}\|_2 / \|\mathbf{b}\|_2, \tag{19}$$

where $\mathbf{w} = \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} \mathrm{f} \\ \mathrm{g} \end{bmatrix}$, see Higham.[29, Equation 7.23] Again, we note that in certain circumstances it may be more appropriate to consider the residual of the first and second rows of Equation 1 individually. In particular, note that we can bound Equation 19 by

$$\|A\mathrm{x} + B^T\mathrm{y} - \mathrm{f}\|_2^2 / \|\mathbf{b}\|_2^2 + \|B\mathrm{x} - \mathrm{g}\|_2^2 / \|\mathbf{b}\|_2^2 \leqslant \|\mathcal{A}\mathbf{w} - \mathbf{b}\|_2^2 / \|\mathbf{b}\|_2^2.$$

We highlight the fact that, due to the nature of null-space factorizations, $\|B\mathrm{x} - \mathrm{g}\|_2^2$ is generally well behaved and close to machine precision. This is because the LU or QR factorization used to find the null space of B will be accurate. When comparing null-space factorizations, we can, therefore, assume that most of the error in $\|\mathbf{w} - \mathbf{b}\|_2$ actually comes from the term $\|Ax + B^T y - f\|_2$. In contrast, `MA57` does not treat the two types of variable differently. Because we assume no prior knowledge of the source of the matrix in the following examples, we present the results as residuals for the unreduced matrix only.

We require a measure of the amount of fill that a method uses; this will be different in the explicit and implicit cases. For the explicit methods, the measure of fill we use is

$$\text{fill} = \frac{\mathcal{N}(\mathcal{L}_i) + \mathcal{N}(\mathcal{T}_i)}{\text{nnz}(\mathcal{A})}, \tag{20}$$

where $\mathcal{N}(\cdot)$ denotes the number of nonzeros needed to store the information required to solve with $\mathcal{T}_i$. For all the factorizations considered here, $\mathcal{N}(\mathcal{L}_i)$ denotes the number of nonzeros in the leftmost matrix in Factorization i. The central $\mathcal{T}_i$ is symmetric, and we only count the lower triangular part. Here, $\mathcal{N}(\mathcal{T}_i)$ denotes the number of nonzeros of all blocks that only need to be multiplied, plus the number of nonzeros in the factors of any matrix that needs to be solved. For example, for Factorization 1, if $B_1 = L_B U_B$ and $N = L_N L_N^T$, then

$$\mathcal{N}_{\text{explicit}}(\mathcal{L}_1) = \text{nnz}(\texttt{tril}(A_{11})) + \text{nnz}(X) + \text{nnz}(L_B) + \text{nnz}(U_B) + \text{nnz}(L_N),$$

where $\texttt{tril}(\cdot)$ denotes the lower triangular part of a matrix.

For implicit methods, we only store the number of nonzeros needed by the factorizations of the blocks we must solve for; we assume that we have access to the original matrix, and do not count entries that can be obtained from there. For example,

for Factorization 1, the measure of fill would be $\text{nnz}(L_B) + \text{nnz}(U_B) + \text{nnz}(L_N)$, as all the other blocks can be obtained from the original matrix. Recall that, although the storage is lower for implicit methods, the operation count is higher, as certain quantities are computed "on the fly"; see Table 2.

4.1 | An academic example

We start by considering some small academic matrices that illustrate some worst case behaviour. We form a system of the form given in Equation 1 using the MATLAB commands:

```
A = sprandsym(n,0.1,1e-1,2);
B = sprand(m,n,0.1);
B(1:m,1:m) = B(1:m,1:m) + 10*diag(rand(m,1));
b = [A B'; B sparse(m,m)]*rand(n+m,1);
```

We take $n = 1024$ and $m = \{100, 512, 900\}$ and construct the matrices so that the leading m columns of B form a nonsingular submatrix; these can be used without permutation as B_1.

Figure 1 presents plots showing the backward error and the fill. We should expect the norm of the residual (Equation 19) to be close to machine precision if the method is stable. However, we observe that, if B_1 taken as being the leading columns of B, Factorizations 1 and 2 are numerically unstable, with the measure of the backwards error spanning four orders of magnitude, and, at worst, greater than 10^{-9}. This indicates that the accuracy of the solution is variable and hence demonstrates that choosing an invertible subset for B_1 is not sufficient for stability. If, however, we choose B_1 using LUSOL with partial pivoting, as expected given the results quoted in Section 3, the norms of the residuals obtained using Factorizations 1 and 2 all lie below 10^{-13}, confirming that this method is stable. Factorization 3 (the antitriangular factorization) also behaves as expected, being the most stable of the null-space factorizations, although it also results in the largest amount of fill (Figure 1b).

4.2 | Optimization examples

We next consider examples from the optimization community; namely, a selection of relatively small quadratic programs from the CUTEst[42] test collection. In particular, the problems are taken from the Maros and Meszaros quadratic programming test set and are solved without consideration for any inequality constraints. These are of the form

$$\min \frac{1}{2}\mathrm{x}^T A\mathrm{x} - \mathrm{f}^T\mathrm{x}$$
$$\text{s.t.} \quad B\mathrm{x} = \mathrm{g}.$$

We chose the subset of problems for which A is symmetric positive semidefinite, has at least $n - m$ nonzeros on the diagonal, and where $n > m$. Our test problems, and the relative sizes of their submatrices, are listed in Table 3. Our tests showed that the families {HS21, HS35, HS35MOD, HS51, HS52, HS53}, {AUG2D, AUG2DC, AUG2DCQP, AUG2DQP}, and {AUG3D, AUG3DC, AUG3DCQP} all exhibited very similar behaviour, and so we only report results for a representative problem from each of these sets: HS51, AUG2DC, and AUG3DC.

We again select the basis B_1 using LUSOL with partial pivoting. We do not compare Factorization 3 here, and in the subsequent tests, as the cost of doing a QR factorization proved to be excessive (both in terms of timing and storage) for many of these problems.

The results are shown in Figures 2 and 3. The backwards errors for the null-space factorizations and the `ldl` factorization are generally similar, with the null-space factorizations giving slightly better accuracy on the CONT-xxx problems, which are known to be difficult for direct solvers. If we apply one step of iterative refinement (as is usual in practical applications) then, as we see in Figure 2b, all methods behave comparably.

In Figure 3, we compare the sparsity of the factors. The explicit versions of Factorizations 1 and 2 result in similar levels of fill, and, with a few exceptions, these are greater than for `ldl`, while the implicit version of Factorization 1 gives significant storage savings (at the cost of more computation). The latter needs the least storage for 38 of the 53 problems, whereas `ldl` performs best for the remaining 18 problems, in some cases, producing significantly sparser factors than the null-space factorizations. For example, about 100 times more storage is required by Factorization 1 (implicit) compared to `ldl` for the AUGxx problems, and over 1,500 times the storage is needed for HUES-MOD and HUESTIS.

We do not report timings, as we cannot give a fair comparison between our proof-of-concept MATLAB code and the compiled `ldl` command, but we observed the null-space factorizations to be significantly slower. They do, however, potentially offer

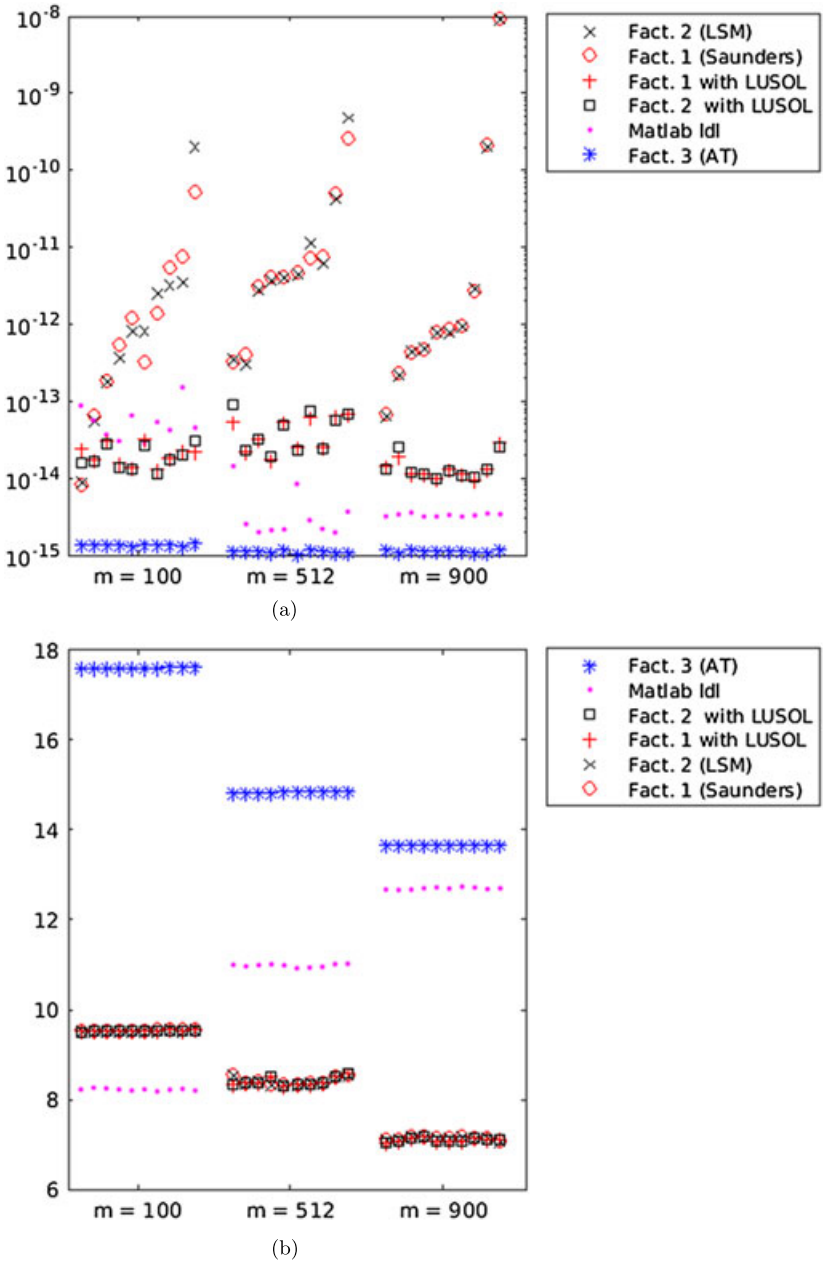

FIGURE 1 Measures of stability and sparsity for various factorizations. The results given for 10 runs with small random matrices and are reported ordered by the worst performer. (a) Backwards error (Equation 19), without iterative refinement. The legend gives the order of performance, running from least accurate to most accurate (on average). (b) Fill, as defined in Equation 20. The legend gives the order of performance, running from least sparse to most sparse (on average)

scope for parallelization, as the bulk of the computational work is from forming and factorizing N. The columns of N can be computed in parallel and, because the Cholesky factorization does not require pivoting, it can be parallelized more effectively than an indefinite sparse solver (see, e.g., Hogg et al.[43]). However, it is still necessary to factorize the nonsquare matrix B^T to obtain the basis B_1.

4.3 | $\mathcal{F}$ matrices

Recall that both stability and sparsity can be shown for $\mathcal{F}$ matrices. We give two examples of such problems below: one from resistor networks and one from fluid flow. In the former, the (1,1) block is a diagonal matrix, and in the latter, it is nondiagonal but symmetric positive definite.

TABLE 3 The CUTEst test matrices

Problem	m	n	$(n-m)/(n+m)$	Problem	m	n	$(n-m)/(n+m)$
LISWET1	10,000	10,002	9.999e-05	HS51	3	5	2.500e-01
LISWET10	10,000	10,002	9.999e-05	LOTSCHD	7	12	2.631e-01
LISWET11	10,000	10,002	9.999e-05	DPKLO1	77	133	2.667e-01
LISWET12	10,000	10,002	9.999e-05	STCQP2	2,052	4,097	3.326e-01
LISWET2	10,000	10,002	9.999e-05	CVXQP1_M	500	1,000	3.333e-01
LISWET3	10,000	10,002	9.999e-05	CVXQP1_S	50	100	3.333e-01
LISWET4	10,000	10,002	9.999e-05	TAME	1	2	3.333e-01
LISWET5	10,000	10,002	9.999e-05	GOULDQP3	349	699	3.334e-01
LISWET6	10,000	10,002	9.999e-05	AUG2DC	10,000	20,200	3.378e-01
LISWET7	10,000	10,002	9.999e-05	MOSARQP1	700	2,500	5.625e-01
LISWET8	10,000	10,002	9.999e-05	PRIMAL1	85	325	5.854e-01
LISWET9	10,000	10,002	9.999e-05	AUG3DC	1,000	3,873	5.896e-01
YAO	2,000	2,002	4.998e-04	CVXQP2_M	250	1,000	6.000e-01
LASER	1,000	1,002	9.990e-04	CVXQP2_S	25	100	6.000e-01
CONT-300	90,298	90,597	1.653e-03	PRIMAL3	111	745	7.407e-01
CONT-201	40,198	40,397	2.469e-03	PRIMAL2	96	649	7.423e-01
CONT-101	10,098	10,197	4.878e-03	PRIMAL4	75	1,489	9.041e-01
CONT-200	39,601	40,397	9.950e-03	PRIMALC1	9	230	9.247e-01
CONT-100	9,801	10,197	1.980e-02	PRIMALC2	7	231	9.412e-01
CONT-050	2,401	2,597	3.922e-02	PRIMALC5	8	287	9.458e-01
GENHS28	8	10	1.111e-01	PRIMALC8	8	520	9.697e-01
QPCSTAIR	356	467	1.349e-01	DUAL4	1	75	9.737e-01
CVXQP3_M	750	1,000	1.429e-01	DUAL1	1	85	9.767e-01
CVXQP3_S	75	100	1.429e-01	DUAL2	1	96	9.794e-01
HS76	3	4	1.429e-01	DUAL3	1	111	9.821e-01
MOSARQP2	600	900	2.000e-01	HUES-MOD	2	10,000	9.996e-01
DTOC3	9,998	14,999	2.001e-01	HUESTIS	2	10,000	9.996e-01

4.3.1 | Resistor networks

An important problem arising from the electronics industry is to find the voltage, V, and current, I, of a network of resistors. The current and voltages are related by the equation $AI + BV = 0$, where A is a diagonal matrix containing the values of the resistors, and B is the incidence matrix of the network. From Kirchhoff's law, we also have $B^T I = 0$.

One node is usually grounded (algebraically, we remove a column of B) so that B has full rank. It is clear that putting these two equations together we get a system of the form 1 that is also an $\mathcal{F}$ matrix. For further details, we refer the reader to Rommes et al.[44]

In order to analyse the behaviour of null space factorizations on such problems, we run tests on artificial systems with $n = 1,024$ resistors, joined at $m = \{100, 250, 512\}$ nodes at random (while forming a complete network). The resistor values are chosen at random from a uniform distribution between 0 and 10^{-2}. Plots showing backward errors and fill are given in Figure 4. A property of matrices of this type is that it is possible to permute B to make it upper trapezoidal, and so a well-conditioned, easy to invert, block B_1 is possible to achieve without arithmetic (see Benzi et al.[1, chapter 6] for a discussion and references).

The results illustrate that the null-space factorizations are stable (as predicted by the theory in de Niet et al.[9]); indeed, they produce smaller backward errors than `ldl` (without iterative refinement). In terms of sparsity, the antitriangular factorization (Factorization 3) is again the poorest, whereas Factorization 1 gives the sparsest factors. As we do not need to factorize a block of B, the fill for the factorizations based on the fundamental basis is negligible.

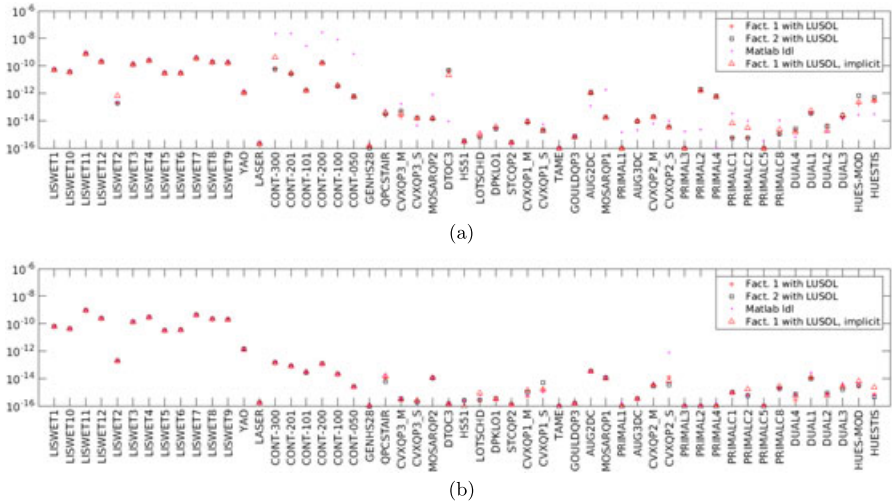

FIGURE 2 Stability results for matrices from the CUTEst test set. We have rounded up values less than 10^{-16} where neceesary. (a) Backwards error (Equation 19), no iterative refinement. (b) Backwards error (Equation 19) after one step of iterative refinement

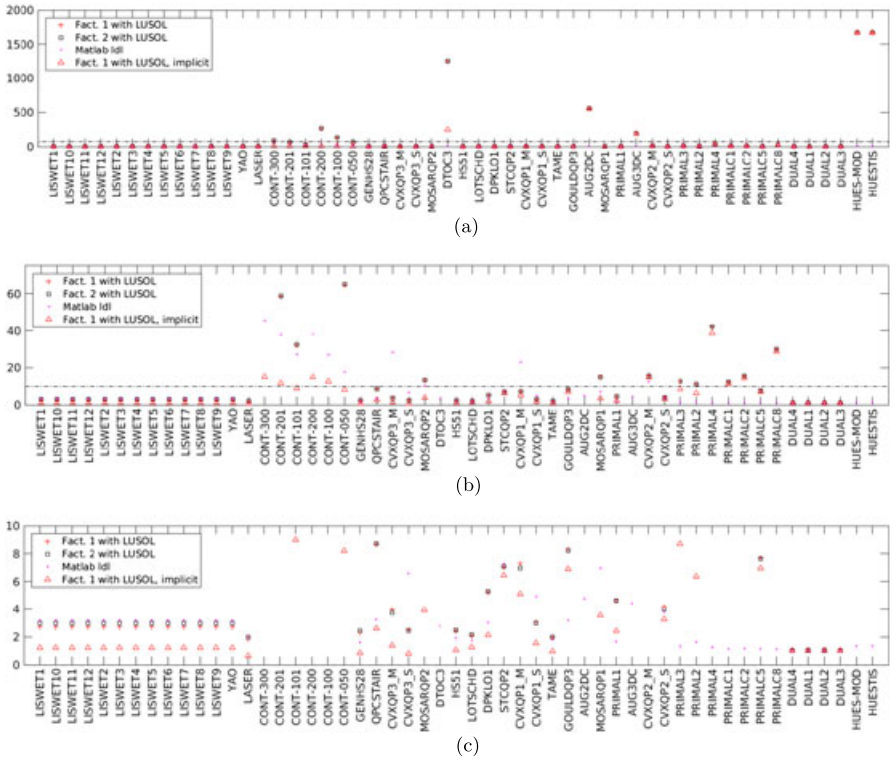

FIGURE 3 Sparsity results for matrices from the CUTEst test set. (a) Fill, as defined in Equation 20. (b) Detail from Figure 3a. (c) Detail from Figure 3b

4.3.2 | Fluid flow problems

Fluid flow problems provide another source of $\mathcal{F}$ matrices. In particular, we use some matrices derived from the mixed hybrid finite element approximation of Darcy's law and the continuity equation, which describes fluid flow through porous media.[45] Our test matrices* are described in Table 4; the same examples were used as test cases by, for example, Tůma[46] and de Neit et al.[9] Sparsity and stability results are given in Figure 5. As with example 4.2, we do not test Factorization 3 here, as computing a QR factorization proved to be too expensive.

*We thank Miroslav Tůma for providing us with these matrices.

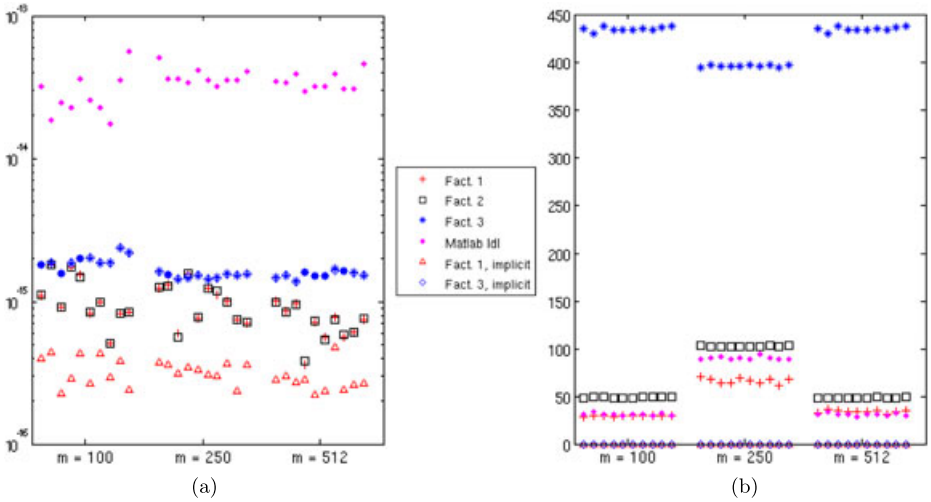

FIGURE 4 Stability and sparsity results for the resistor network problem, as described in Section 4.3.1. (a) Backwards error (Equation 19), no iterative refinement. (b) Fill, as defined in Equation 20

TABLE 4 $\mathcal{F}$ matrices from a problem in fluid flow

Problem	m	n	$(n-m)/(n+m)$
S3P	207	270	0.13208
M3P	1,584	2,160	0.15385
L3P	12,384	17,280	0.16505
DORT	9,607	13,360	0.16341
DORT2	5,477	7,515	0.15687
dan2	46,661	63,750	0.15478

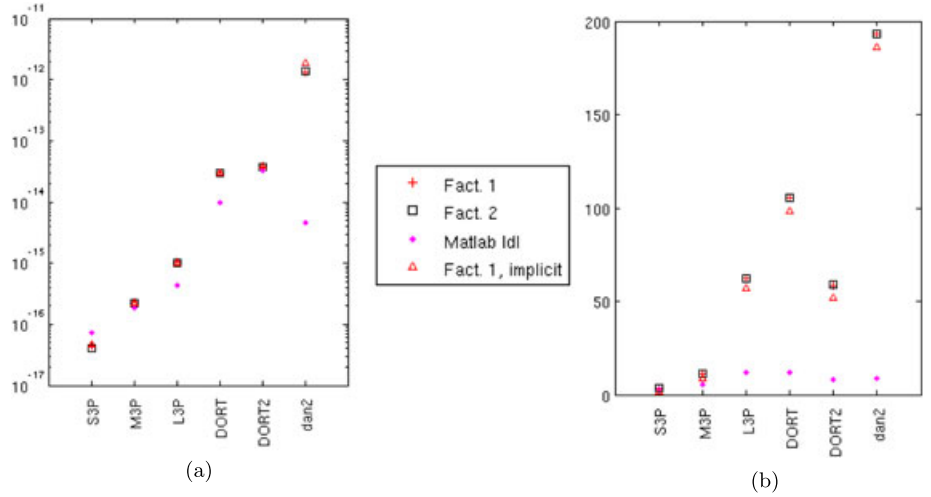

FIGURE 5 Stability and sparsity results for $\mathcal{F}$ matrices from a problem in fluid flow, as described in Section 4.3.2. (a) Backwards error (Equation 19), no iterative refinement. (b) Fill, as defined in Equation 20.

For these real-word problems, the null-space factorizations perform markedly worse than `ldl`. The fill, in particular, is very high. Consider the problem `dan2`, which has 318,750 nonzeros in A and 127,054 nonzeros in B. For Factorization 1, if $Z^TAZ = LL^T$, then the number of nonzeros in L is 59,146,699. This is an order of magnitude larger than any of the other terms in the fill calculation and is due to the fact that the null-space matrix, Z^TAZ, is significantly denser than A. This means that the null-space factorizations can need more than 20 times the storage of `ldl`, as `ldl` has greater freedom to choose appropriate

pivots to preserve sparsity in the LDL^T factorization. Furthermore, null-space factorizations also result in larger backwards errors (although a single step of iterative refinement reduces all the backwards errors to the order of machine precision).

5 | CONCLUSION

Null-space methods for solving systems of the form given by Equation 1 can be thought of in terms of matrix factorizations. We have presented a number of such decompositions that have been proposed in the literature. By placing these in the unified framework of null-space factorizations we see that, although they may appear to be different, they are in fact fundamentally related.

By highlighting such relationships it is clear that, for example, stability results proved for one type of factorization are equally applicable to another. In particular, we can say that all factorizations relying on a fundamental basis enjoy a certain degree of stability and sparsity when applied to $\mathcal{F}$ matrices. Similarly, the stability of Factorization 2 can be shown if B_1 is chosen appropriately, a result that is only clear by putting it in the null-space factorization framework.

In the literature we find null-space factorizations that are known to be either stable or sparse, but (with the exception of $\mathcal{F}$ matrices) not both. However, a practical factorization must possess favourable properties in both these areas. In Section 4, we investigated numerically the stability and sparsity properties of various null-space factorizations using matrices derived from a number of academic and practical applications. Our results suggest that these factorizations—particularly the form in Equation 9—have the potential to be competitive with a sparse symmetric indefinite solver for some classes of problems.

However, we stress that any set of numerical tests can only be indicative of typical behaviour, and cannot predict worse case performance. In the future, we would like to see the development a null-space factorization for general saddle point systems that gives, as much as possible, a theoretical guarantee of a certain measure of both stability and sparsity. Our hope is that, by exposing connections between previously disparate solution methods, we have brought such an algorithm one step closer.

ACKNOWLEDGEMENTS

We would like to thank Nick Gould, Chen Greif, Sangye Lungten, Jen Pestana, Wil Schilders, and Andy Wathen for reading and commenting on a draft manuscript. We are particularly grateful to Cleve Ashcraft and Michael Saunders for their detailed and constructive feedback and encouragement. This work was supported by the EPSRC grant EP/I013067/1.

REFERENCES

1. Benzi M, Golub GH, Liesen J. Numerical solution of saddle point problems. Acta Numer. 2005;14:1–137.
2. Biegler LT, Nocedal J, Schmid C. A reduced Hessian method for large-scale constrained optimization. SIAM J Opt. 1995;5(2):314–347.
3. Biros G, Ghattas O. A Lagrange-Newton-Krylov-Schur method for PDE-constrained optimization. SIAG/OPT Views and News. 2000;11(2):1–6.
4. Bochev P, Ridzal D. Additive operator decomposition and optimization-based reconnection with applications. In: Lirkov I, Margenov S, Waśniewski J, editors. Proceedings of the 7th international conference on Large-Scale Scientific Computing (LSSC′09). Berlin, Heidelberg: Springer-Verlag; 2009. p. 645–652.
5. Jiang Q, Geng G. A reduced-space interior point method for transient stability constrained optimal power flow. IEEE Trans Power Syst. 2010;25(3):1232–1240.
6. Kim HK, Flexman M, Yamashiro DJ, Kandel JJ, Hielscher AH. PDE-constrained multispectral imaging of tissue chromophores with the equation of radiative transfer. Biomed Opt Express. 2010;1(3):812–824.
7. Moghaddam PP, Herrmann FJ. Randomized full-waveform inversion: A dimensionality-reduction approach. SEG Tech Program Expanded Abstr. 2010;29: 977–982.
8. Ashcraft C, Grimes R. On direct elimination of constraints in KKT systems. Toulouse: Sparse Days 2011, CERFACS; 2011.
9. de Niet AC, Wubs FW. numerically stable LDL^T-factorization of $\mathcal{F}$-type saddle point matrices. IMA J Numer Anal. 2009;29(1):208–234.
10. Dollar HS, Wathen AJ. Approximate factorization constraint preconditioners for saddle-point matrices. SIAM J Sci Comput. 2006;27(5):1555–1572.
11. Mastronardi N, Van Dooren P. The antitriangular factorization of symmetric matrices. SIAM J Matrix Anal Applics. 2013;34(1):173–196.
12. Maubach J, Schilders W. Micro- and macro-block factorizations for regularized saddle point systems. Eindhoven: Technische Universiteit Eindhoven; 2012.
13. Pestana J, Wathen AJ. The antitriangular factorization of saddle point matrices. SIAM J Matrix Anal Applics. 2014;35(2):339–353.
14. Schilders WHA. Solution of indefinite linear systems using an LQ decomposition for the linear constraints. Linear Algebra Appl. 2009;431(3):381–395.
15. Lungten S, Schilders WHA, Maubach JML. Sparse block factorization of saddle point matrices. Linear Algebra Appl. 2016;502: 214–242.
16. Gill PE, Murray W. Newton-type methods for unconstrained and linearly constrained optimization. Math Program. 1974;7(1):311–350.

17. Gould NIM. On modified factorizations for large-scale linearly constrained optimization. SIAM J Opt. 1999;9(4):1041–1063.
18. Keller C, Gould NIM, Wathen AJ. Constraint preconditioning for indefinite linear systems. SIAM J Matrix Anal Appl. 2000;21(4):1300–1317.
19. Fletcher R, Johnson T. On the stability of null-space methods for KKT systems. SIAM J Matrix Anal Appl. 1997;18(4):938–958.
20. Dollar HS, Gould NIM, Schilders WHA, Wathen AJ. Implicit-factorization preconditioning and iterative solvers for regularized saddle-point systems. SIAM J Matrix Anal Appl. 2006;28(1):170–189.
21. Mardal KA, Winther R. Preconditioning discretizations of systems of partial differential equations. Numer Linear Algebra Appl. 2011;18(1):1–40.
22. Bridson R. An ordering method for the direct solution of saddle-point matrices. Available from: http://www.cs.ubc.ca/~rbridson/kktdirect/; 2007. [Accessed 8 July 2015]
23. Scott JA. A note on a simple constrained ordering for saddle-point systems: STFC Rutherford Appleton Laboratory; 2009. Tech. Report RAL-TR-2009-007.
24. Scott J, Tůma M. On signed incomplete Cholesky factorization preconditioners for saddle-point systems. SIAM J Sci Comput. 2014;36(6):A2984–A3010.
25. Forsgren A, Murray W. Newton methods for large-scale linear equality-constrained minimization. SIAM J Matrix Anal Appl. 1993;14(2):560–587.
26. Davis TA. Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization. ACM Trans Math Softw. 2011;38(1):8:1–8:22.
27. Mastronardi N, van Dooren P. An algorithm for solving the indefinite least squares problem with equality constraints. BIT Numer Math. 2014;54(1):201–218.
28. Bujanović Z, Kressner D. A block algorithm for computing antitriangular factorizations of symmetric matrices. Numer Algorithms. 2016;71(1):41–57.
29. Higham NJ. Accuracy and Stability of Numerical Algorithms. 2nd edn.: SIAM; 2002.
30. Gill PE, Murray W, Saunders MA. SNOPT: An SQP algorithm for large-scale constrained optimization. SIAM Rev. 2005;47(1):99–131.
31. Pinar A, Chow E, Pothen A. Combinatorial algorithms for computing column space bases that have sparse inverses. Elec Trans Numer Anal. 2006;22: 122–145.
32. Murray W. An algorithm for finding a local minimum of an indefinite quadratic program; 1971. Tech. report, NPL Report NAC 1.
33. Fletcher R. Practical Methods of Optimization. Chichester: John Wiley & Sons; 2013.
34. Duff I, Erisman AM, Reid JK. Direct Methods for Sparse Matrices. Oxford: Oxford University Press; 1986.
35. Arioli M, Manzini G. A null space algorithm for mixed finite-element approximations of Darcy's equation. Commun Numer Meth Engng. 2002;18(9):645–657.
36. Vavasis SA. Stable numerical algorithms for equilibrium systems. SIAM J Matrix Anal Applics. 1994;15(4):1108–1131.
37. Chen Y, Davis TA, Hager WW, Rajamanickam S. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. ACM Trans Math Softw. 2008;35(3):22–.
38. Saunders M. LUSOL: A basis package for constrained optimization: SOL, Stanford University; 2013. http://web.stanford.edu/group/SOL/software/lusol/. [Accessed 8 July 2015]
39. Henderson N. Mex interface to LUSOL. Available from: http://github.com/nwh/lusol; 2014. [Accessed 8 July 2015]
40. Duff IS. MA57—A code for the solution of sparse symmetric definite and indefinite systems. ACM Trans Math Softw. 2004;30(2):118–144.
41. HSL. A collection of Fortran codes for large-scale scientific computation. Available from: http://www.hsl.rl.ac.uk; 2013. [Accessed 8 July 2015]
42. Gould NIM, Orban D, Toint PL. CUTEst: A constrained and unconstrained testing environment with safe threads. Comput Optim Appl 2015;60: 545–557.
43. Hogg JD, Reid JK, Scott JA. Design of a multicore sparse Cholesky factorization using DAGs. SIAM J Sci Comput. 2010;32: 3627–3649.
44. Rommes J, Schilders WHA. Efficient methods for large resistor networks. IEEE Trans Comput-Aided Design Integr Circuits Syst. 2010;29(1):28–39.
45. Maryska J, Rozložník M, Tůma M. Schur complement systems in the mixed-hybrid finite element approximation of the potential fluid flow problem. SIAM J Sci Comput. 2000;22(2):704–723.
46. Tůma M. A note on the LDL^t decomposition of matrices from saddle-point problems. SIAM J Matrix Anal Applic. 2002;23(4):903–915.

How to cite this article: Rees T, Scott J. A comparative study of null-space factorizations for sparse symmetric saddle point systems. *Numer Linear Algebra Appl*. 2017;e2103. https://doi.org/10.1002/nla.2103