Comput. Methods Appl. Mech. Engrg. 198 (2009) 3084–3095

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

Computing the null space of finite element problems

Gil Shklarski [a], Sivan Toledo [a,b,*]

[a] School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel
[b] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge 02139, USA

ARTICLE INFO

Article history:
Received 22 June 2008
Received in revised form 13 May 2009
Accepted 14 May 2009
Available online 27 May 2009

Keywords:
Rigid-body motions
Null space
Linear constraints
Finite element models
Singular linear systems
Fretsaw preconditioner

ABSTRACT

We present a method for computing the null space of finite element models, including models with equality constraints. The method is purely algebraic; it requires access to the element matrices, but not to the geometry or material properties of the model.

Theoretical considerations show that under certain conditions, both the amount of computation and the amount of memory required by our method scale linearly with model size; memory scales linearly but computation scales quadratically with the dimension of the null space. Our experiments confirm this: the method scales extremely well on 3-dimensional model problems. In general, large industrial models do not satisfy all the conditions that the theoretical results assume; however, experimentally the method performs well and outperforms an established method on industrial models, including models with many equality constraints.

The accuracy of the computed null vectors is acceptable, but the method is usually less accurate than a more naive (and computationally much more expensive) method.

1. Introduction

The computation of the null space of finite element matrices is an important task in many industrial applications. In this paper we show a method to efficiently compute the null space of symmetric positive semidefinite finite-element matrices, possibly with additional linear equality constraints. Our method is strictly algebraic and does not use the geometry of the underlying model; the only information needed is the model in elemental form and a representation of the linear constraints. Our method works for any model with symmetric semidefinite elements. It works both when the constraints are symmetrically added to the finite-element matrix and when the constraints are appended to the symmetric finite-element matrix to create a rectangular matrix.

In structural mechanics the null space of a finite-element matrix corresponds to the zero energy modes of the structure. Such zero energy modes are important in static and dynamic analysis of floating structures [16], for example, where the computed null space is used to solve a consistent but rank-deficient system of linear equations. Certain domain decomposition methods also require the computation of the null space of floating substructures. This is the case for FETI [13,11], for the balancing domain decomposition method [26], and for some direct flexibility methods [14,6].

Although most finite-element analyses are performed on nonsingular matrices, a nonempty null space can also be introduced due to an error in the modeling phase. A model with too few constraints (boundary conditions) can lead to a singular coefficient matrix. If the linear solver factors the matrix with a backward stable factorization (such as LU with partial pivoting), the singularity can be detected fairly easily either as a breakdown in the factorization process or using inverse iteration [21]. The failed factorization process, is however, expensive. Other types of linear solvers, especially iterative ones, do not cope well with singular or nearly singular matrices, which can cause the solver to converge extremely slowly or not at all. Our method can address this issue by quickly checking if a finite element model is sufficiently constrained; the scaling of our null space computation routine is approximately linear.

Finally, algebraic multigrid methods such as smooth aggregation [19,35,36] also benefit from an efficient computation of the null space of semidefinite finite-element matrices.

The remainder of the paper is organized as follows. The next section gives an overview of our method and relate it to existing work. Section 3 describes our method in detail. We present numerical results in Section 4. Finally, we summarize our results in Section 5.

2. Overview and related work

Let $K = \sum_e K_e$ be an n-by-n finite-element matrix, where all the K_es are symmetric positive semidefinite element matrices. Let C be

* Corresponding author. Address: School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Tel.: +972 52 2424460; fax: +972 3 6350780.
E-mail address: stoledo@tau.ac.il (S. Toledo).

0045-7825/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.cma.2009.05.012

a c-by-n row block of linear constraints (C may be empty). Our ultimate goal is to compute the null space of the rectangular matrix

$$K_C = \begin{bmatrix} K \\ C \end{bmatrix},$$

or of the symmetric square matrix

$$\begin{bmatrix} K & C^T \\ C & \end{bmatrix}.$$

We compute the null space using a two phase process. In the first phase we efficiently compute an $(n+\ell)$-by-$(n+\ell)$ matrix $\mathscr{F}(K)$. The matrix $\mathscr{F}(K)$ is an easy-to-factor approximation of K that preserves its null space (in a specific technical sense that we describe later). The theory behind the approximation $\mathscr{F}(K)$ was developed in an earlier paper [33], and the algorithmic aspects are described below in Section 3. In the second phase we compute the null space of

$$\widehat{K} = \begin{bmatrix} \mathscr{F}(K) \\ \widehat{C} \end{bmatrix},$$

where $\widehat{C}$ is the matrix C padded with ℓ zero columns. We then extract the null space of K from the computed null space of $\widehat{K}$. Any reliable method for computing the null space of $\widehat{K}$ is appropriate here, including the methods in [7,17,31].

The key insight in our method is the construction of the matrix $\widehat{K}$ such that factoring it is significantly easier than factoring K_C. Once a matrix is factored using a backward-stable factorization, computing its null space is relatively easy using inverse power iterations. This technique works well even for rectangular matrices [18]. Unfortunately, factoring K_C (or even just K) is expensive for three-dimensional models. Our method addresses this by replacing K by $\mathscr{F}(K)$, which preserves the null space but is easier to factor (even though its dimension is higher).

Our method uses a combinatorial relationship called *mutual rigidity* between the K_es. This relationship, which we define in Section 3.1, is closely related to element relationships defined in several papers; two mutually rigid elements are defined as*mechanism free adjacent elements* in [29], as *elasticity connected elements* in [8], and as adjacent in the *natural association graph* in [22,23]. This relationship is also closely related to the *mechanism buster* algorithm in [11].

There is, however, a fundamental difference between the element-relationship definition in this paper and the definitions in the papers cited above. Our definition is purely algebraic; it uses only the information in the element matrices. All the other definitions use geometrical information, discretization specific information, or assumptions on the physics that the elements model. Therefore, our method is more general and can be used with many finite-element modeling techniques. Moreover, our method is not restricted to structural mechanics; it works equally well in electrostatic problems, for example.

Our method also *uses* the inter-element relationship differently than earlier methods. We use the rigidity relations and the underlying graph to combinatorially sparsify K. In essence, we form the stiffness matrix of another physical structure, which has the same null space as the given structure but is much easier to analyze (because its stiffness matrix is easier to factor).

The specific problem of computing the null space of large semidefinite sparse matrices arising in structural mechanics was explored in [12] and in [29]. In both cases the null space of rigid substructures was computed using additional geometrical information. The method in this paper works without this information, and therefore may be suitable for broader usage scenarios.

Several papers [7,17,31] proposed algorithms for finding the null space of another matrix arising in structural mechanics, the equilibrium matrix in the force method. More recent papers [22,23] presented efficient algorithms to compute the null space of the equilibrium matrix, when the matrix was computed using specific finite-element formulation and specific element types. The equilibrium matrix is underdetermined (has more columns than rows), whereas our paper focuses on computing the null space of another family of matrices, namely stiffness matrices and such matrices with appended constraints. These matrices are square or overdetermined (have more rows than columns) but possibly rank deficient.

Finally, we mention another related group of algorithms that can be used to compute the null space of sparse rectangular matrices such as K_C. These methods are based on rank revealing QR factorizations [28,30,1]. These works compute a QR factorization in a careful way to handle rank deficiency. The QR factorization can then be used to accurately compute a null space basis for the original matrix. Computing the QR factorization of K_C, however, is very slow. Nevertheless, these methods can be used to compute the null space of $\widehat{K}$. Our method uses an *LU* factorization rather than a QR, since sparse *LU* factorizations are usually much faster than sparse *QR* factorizations.

3. The method in detail

We start with some notation. We represent a finite element model as collection of elements of the form $e = (k_e, I_e)$ where k_e is an n_e-by-n_e element matrix in local variables and I_e is a vector of size n_e containing the global indices corresponding to the local variables. The matrix K_e is the n-by-n element matrix in global coordinates. In other words, $K_e(I_e, I_e) = k_e$. The linear constraints of the model, if there are any, are given in a c-by-n block of rows denoted by C. This is all the information that we need for our method. Let $K = \sum_e K_e$, the steps for computing the null space of $K_C = \begin{bmatrix} K^T & C^T \end{bmatrix}^T$ are:

(1) Compute the *rigidity graph*, a combinatorial graph in which vertices represent elements and edges represent rigidity relationships.
(2) Use the rigidity graph to extend every element $e = (k_e, I_e)$ into an extended element $\hat{e} = (k_e, \widehat{I}_e)$.
(3) Assemble the elements $\{\hat{e}\}$ into an $(n+\ell)$-by-$(n+\ell)$ matrix $\mathscr{F}(K)$.
(4) Pad C into $\hat{C} = [C \quad 0]$ and append to $\mathscr{F}(K)$, to create
$$\widehat{K} = \begin{bmatrix} \mathscr{F}(K) \\ \widehat{C} \end{bmatrix}.$$
(5) Compute the *LU* factorization of $\widehat{K}$ with partial pivoting.
(6) Compute the null space of $\widehat{K}$ using subspace inverse iteration, using its *LU* factors.
(7) Restrict the null space of $\widehat{K}$ to its first n coordinates and orthogonalize.

The theoretical idea behind this is that $x \in \mathrm{null}(\widehat{K})$ if and only if

$$Q^T x \in \mathrm{null}\left(\begin{bmatrix} K \\ C \end{bmatrix} \right),$$

where Q is the $(n+\ell)$-by-n identity matrix (an identity extended with ℓ zero rows).

The rest of this section describes these steps in detail and explains the theoretical justification for the method.

3.1. Rigidity graphs

We define an undirected weighted combinatorial graph which we call the *rigidity graph* [33, Definition 6.1]. The vertices in this graph correspond to the elements and the edges connect certain pairs of elements. Informally, this graph is the dual of the finite-element mesh. Fig. 3.1c illustrates the rigidity graph of the model in Fig. 3.1b.

More formally, the edges of the graph represent a local relation between pairs of elements. We call this relation *mutual rigidity* relationship. The rigidity relationship can be defined and computed in a purely algebraic way (but if the element types are known, not just the (k_e, I_e)s, then mutual rigidity can be determined by the element type and adjacency in the mesh). We next show how to algebraically determine whether two given element matrices are mutually rigid.

We denote by ℓ_e the dimension of the null space of k_e, and we let N_e be an n_e-by-ℓ_e matrix whose columns are a basis for the null space of k_e. In structural analysis in 2 and 3 dimensions, ℓ_e is 3 and 6 respectively; in electrostatic problems, ℓ_e is 1 in any dimension.

Let e and f be two elements. Let C_{ef} be the common variables that appear both in I_e and I_f. We denote by $C_{e\to f}$ and $C_{f\to e}$ the $|C_{ef}|$-vectors that contain the indices of the common variables in I_e and I_f, respectively. In other words, $I_e(C_{e\to f}) = I_f(C_{f\to e}) = C_{ef}$. We also define $N_{e\to f}$ and $N_{f\to e}$ as the rows subset matrices $N_e(C_{e\to f}, :)$ and $N_f(C_{f\to e}, :)$, respectively (the colon selects all the columns, as in Matlab).

The elements e and f are mutually rigid if the following conditions hold:

(1) The element matrices k_e and k_f have the same null space dimension (i.e. $\ell_e = \ell_f$).
(2) The number of the shared variables $|C_{ef}| \geqslant \ell_e$.
(3) The columns of $N_{e\to f}$ and $N_{f\to e}$ are linearly independent.
(4) The columns of $N_{e\to f}$ span the columns of $N_{f\to e}$ and vice versa.

If e and f are mutually rigid, we add an edge (e,f) to the rigidity graph. We assign a weight to the edge: the number of shared variables $|C_{ef}|$. (In other applications of the rigidity graph other weights may be appropriate.) These weights will be used in the second phase of the algorithm. Informally and intuitively they indicate that a pair of elements that share more nodes than another pair of elements, are more strongly connected.

We determine the set of mutually-rigid element pairs as follows. We begin by computing the eigen-decomposition of each k_e, which gives us the ℓ_es and the N_es. The next challenge is to find all the element pairs that satisfy conditions 1 and 2 above; that is, the pairs that share both the dimension of the null space and

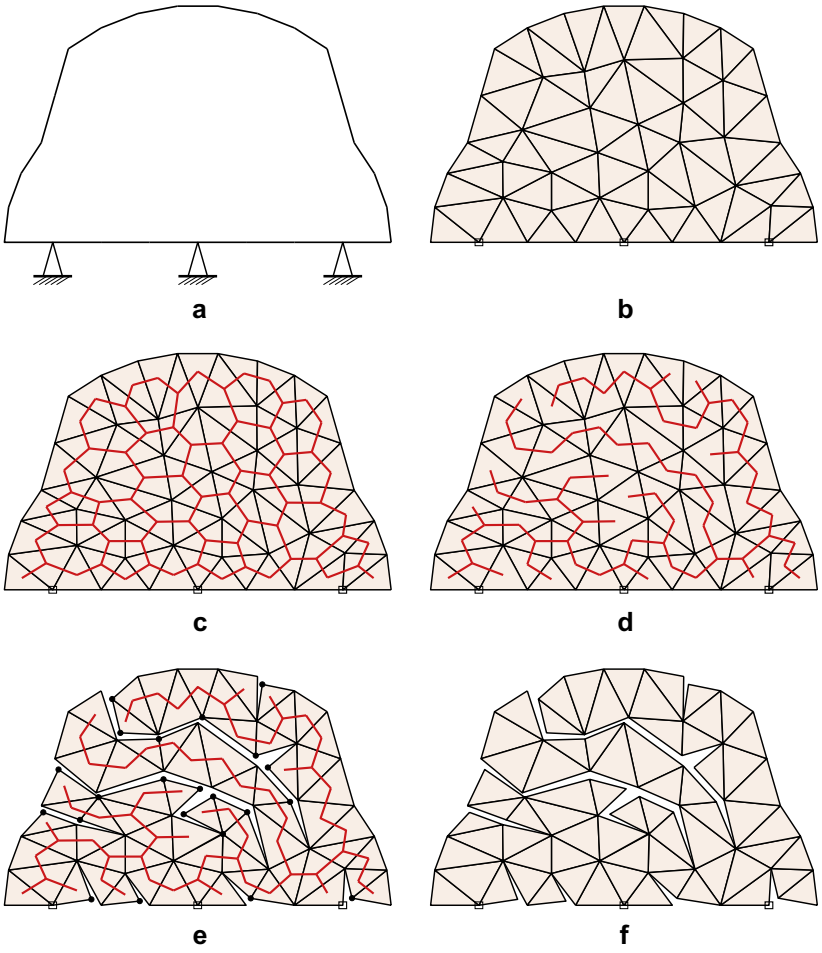

Fig. 3.1. The construction of the fretsaw extension for simply supported planar elastic triangles. (a) The original model problem. (b) Its finite element discretization. The small rectangles represent boundary conditions in the form of simply supported grid points. (c) The rigidity graph. (d) The spanning forest (tree in this case). (e) The addition of slack variables. New slack variables are marked with black circles. The elements where slightly shrunk to illustrate the addition of slack variables. In practice the element matrices are the same. (f) The resulting finite element model.

enough unknowns. A naive approach is to first find all the pairs that have the same null-space dimension and share at least one unknown, and then to remove from this list all the pairs that do not satisfy condition 2. We use a more sophisticated algorithm that is guaranteed to run in time that is linear in the problem size with a small constant. The algorithm is somewhat complex, but it does not employ data structures other than arrays and lists. We omit the details, which appear in [32].

In practice we first find the common ℓ_e for the given model, then we only process the elements that share this specific ℓ_e. This means that every element f with $\ell_f \neq \ell_e$ will be a singleton in the rigidity graph.

We now need to test conditions 3 and 4. We compute the reduced singular values decompositions (SVDS) of $N_{e\to f}$ and $N_{f\to e}$: $N_{e\to f} = U_{e\to f}\Sigma_{e\to f}V_{e\to f}^T$ and $N_{f\to e} = U_{f\to e}\Sigma_{f\to e}V_{f\to e}^T$. We test whether $N_{e\to f}$ is full rank by checking whether the ratio of its smallest to largest singular values is less than machine ϵ ($\approx 10^{-16}$ in double precision), and similarly for $N_{f\to e}$. We test condition 4 by computing $\|U_{f\to e}U_{f\to e}^T N_{e\to f} - N_{e\to f}\|_F$ and $\|U_{e\to f}U_{e\to f}^T N_{f\to e} - N_{f\to e}\|_F$. If both are small enough, then the condition 4 holds. By definition, $U_{f\to e}U_{f\to e}^T = N_{f\to e}N_{f\to e}^+$ and $U_{e\to f}U_{e\to f}^T = N_{e\to f}N_{e\to f}^+$. Therefore, this criteria determines whether the following two equalities hold:

$$N_{f\to e}N_{f\to e}^+ N_{e\to f} = N_{e\to f}$$
$$N_{e\to f}N_{e\to f}^+ N_{f\to e} = N_{f\to e}\,.$$

By [2, Proposition 6.1.7], a vector b is in the range of a matrix A if and only if $AA^+b = b$. Therefore, condition 4 holds if and only if the last two equalities hold.

In some models there are element pairs (e,f) such that $I_e \subseteq I_f$. We detect such pairs during the computation of the rigidity graph and sum each pair into a single element by summing the two original element matrices. This process can be performed efficiently using a single pass on the data [32].

We close this section with a short discussion about the mutual rigidity relationship in the context of structural mechanics. The relationship is computed algebraically, but a rule of thumb that works for common element types is that two elements that share grid points are mutually rigid if and only no mechanism exists when examining a body consisting of only these two elements. For example, two struts connected at one joint are never mutually rigid [33, Section 7.2]; Two elastic triangles in the plane constructed by three struts each are mutually rigid if they share a side (share two grid points) [33, Section 7.3]; elastic tetrahedrons in space built out of six struts each are mutually rigid if they share a face (share three grid points); solid tetrahedral elements that are connected in a single grid point are mutually rigid if they share the three displacement variables and three rotation variables in that grid point, but are not mutually rigid if they share only the displacement variables.

3.2. Fretsaw extensions and null-space preservation

In the second phase of the algorithm, we compute a maximum weight spanning forest of the rigidity graph and form a finite-element model of a structure whose rigidity graph is this forest. We use the given element matrices but drop some of the inter-element continuity constraints. Fig. 3.1d illustrates the spanning forest (tree in this case) of the rigidity graph in Fig. 3.1c.

The basic step in the computation of the new model is removing a rigidity edge between two adjacent elements. In order to detach such elements we add new slack variables to the model. Fig. 3.2 illustrates this step. We now show how to efficiently allocate those new slack variables to the elements so that the modified model will be the exact forest we computed.

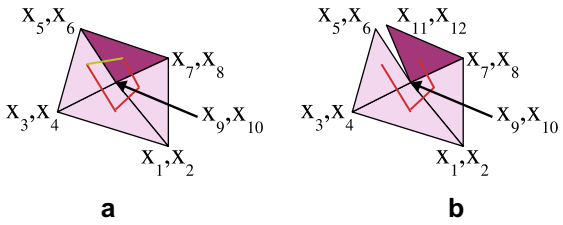

Fig. 3.2. Detaching a rigidity edge by adding slack variables. The element matrices remain unchanged (in local coordinates), but in the local-to-global index mapping of the dark element, new slack variables X_{11} and X_{12} replaced the original variables X_5 and X_6.

Let F be the maximum weight spanning forest of the rigidity graph. We first compute the connected components of F. We store for each element e the index C_e of its connected component. We also randomly select a representative element for each connected component. These elements will not be altered during the entire process. We define r to be the highest variable index used so far in the process. We initialize it to n (the highest variable index in K).

In many cases, we know an a-priori grouping of the variables into grid points. This knowledge can accelerate the following part of our method. If it is not known (or not passed to the software layer that performs this operation) we treat each variable as a different grid point. In any case, the acceleration is only by a bounded constant factor, namely the number of variables per grid point. Next, we perform the following operation for every grid point p.

(1) Let E_p be the set of elements that are incident on p. We partition E_p according to the connected components of F (stored for every $e \in E_p$ in C_e). Let $\{E_{p,i}\}_i$ be the parts of E_p ($\uplus_i E_{p,i} = E_p$).
(2) For each part $E_{p,i}$ of E_p we perform the following steps:
   (a) We construct the induced subgraph $G_{p,i}$ of F with the vertex set $E_{p,i}$. The edges of $G_{p,i}$ are
   $$\{(e,f)|e,f \in E_{p,i}, (e,f) \in F\}.$$
   (b) We compute the connected components of $G_{p,i}$. Let $V_{p,i}^{(1)}, \ldots, V_{p,i}^{(n_{p,i})}$ be the vertex sets of the connected components. ($n_{p,i}$ is the number of connected components.)
   (c) If $n_{p,i} > 1$ then we need to detach the elements in different $V_{p,i}^{(j)}$s. We do that by adding new slack variables and exchanging the variables corresponding to p in all of the $V_{p,i}^{(j)}$s except for one. We first select the set $V_{p,i}^{(k)}$ of elements that are not changed. If $E_{p,i}$ contains a representative element of a global connected component, then we select the k such that $V_{p,i}^{(k)}$ contains this representative. Otherwise, we select k arbitrarily. Let D_p be the number of variables that correspond to p (D_p can be 1 if no knowledge of variable grouping to grid points exist). For every $j \neq k$, we do the following:
      (i) We go over all $e \in V_{p,i}^{(j)}$, we renumber the D_p variables corresponding to p in I_e to the new slack variables $(r+1), (r+2), \ldots, (r+D_p)$.
      (ii) We let $r \leftarrow r + D_p$.

We denote the resulting I_e by $\widehat{I}_e$. See Fig. 3.3 for an illustrated example for this process. Steps 1, 2(a) and 2(b) can be fused into a single pass on the data.

Let ℓ be the number of new variables that were introduced in the process above. We now assemble the resulting finite-element model into the $(n+\ell)$-by-$(n+\ell)$ matrix $\mathscr{F}(A)$. This matrix is called

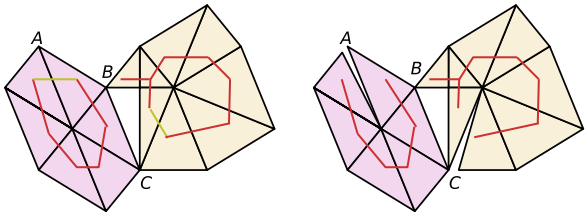

Fig. 3.3. Slack variable allocation mechanism. Parts (a) and (b) illustrate a model before and after the addition of slack variables respectively. The rigidity graph is illustrated in yellow and its spanning forest is illustrated in red. The rigidity forest has two connected components illustrated in pink and light yellow. The rigidity forest around grid points A, B and C is not connected. Grid point A represents the basic case. There is only one connected component of the forest adjacent to A. The subgraph incident on A is not connected. Therefore, slack variables are added. We do not add slack variables for point B since the induced rigidity forest for every connected component of the rigidity forest is connected. We add one set of slack variables for point C since the rigidity forest is connected in the pink connected component and not connected in the light yellow connected component.

a *fretsaw-forest extension* [33, Definition 8.8]. Fig. 3.1f reveals the reasoning behind this name.

The next theorem shows that under a fairly natural condition, the null space of the fretsaw extension is a subset of the null space of the original stiffness matrix. The condition that guarantees this is called *element compatibility*. Let $\mathbb{S} \subseteq \mathbb{R}^n$ be a linear space. An element matrix k_e is called $\mathbb{S}$-*compatible* if every vector v_e in null (k_e) has a unique extension into a vector v in $\mathbb{S}$ (by extension we mean that $v(I_e) = v_e$), and if the restriction $v(I_e)$ of every vector v in $\mathbb{S}$ is always in null (k_e). In linear elasticity, element matrices are $\mathbb{S}_6$-compatible with the space $\mathbb{S}_6$ of rotations and translations. In electrostatic problems, element matrices are $\mathbb{S}_1$-compatible with the space $\mathbb{S}_1$ of constant vectors.

Theorem 3.1. *Let $K = \sum_e K_e$, where $\{K_e\}$ is a collection of symmetric positive semidefinite n-by-n matrices that are compatible with some $\mathbb{S} \subseteq \mathbb{R}^n$, let Q be the $(n+\ell)$-by-n identity matrix, and let $\mathscr{F}(K)$ be a fretsaw-forest extension of K, as constructed by the algorithm above. Then,*

(1) *There exist an $(n+\ell)$-by-n matrix P such that $P^T\mathscr{F}(K)P = K$ and $Q^TP = I$.*
(2) *null $(\mathscr{F}(K)) \subseteq$ null(QKQ^T).*

The first part of the theorem is exactly [33, Lemma 8.9]. The second part of the theorem is a slight extension of Theorem 8.11 in [33]. We omit the proof because it requires a considerable amount of notation and definitions. The complete proof appears in [32].

The sum $K = \sum_e K_e$ of n-by-n SPSD finite-element matrices is usually singular, so $Kd = f$ has an infinite number of solutions or none at all. In order to be able to form a numerical problem with a unique solution, linear equality constraints are usually added to K. Let C be a c-by-n matrix. A common way to get a nonsingular system is symmetrically adding the block C to K. It can be easily shown ([32]) that

$$\text{null}\left(\begin{bmatrix} K & C^T \\ C & 0 \end{bmatrix}\right) = \left\{ \begin{bmatrix} u \\ v \end{bmatrix} | u \in \text{null}(K) \cap \text{null}(C) \text{ and } v \in \text{null}(C^T) \right\}.$$

Another possibility is adding a c-by-n block of new rows C to K. Clearly, null $\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right) = \text{null}(K) \cap \text{null}(C)$.

The following lemma shows that the fretsaw-forest extension also preserves the null space under equality constraints.

Lemma 3.2. *Let $K = \sum_e K_e$, where $\{K_e\}$ is a collection of symmetric positive semidefinite n-by-n matrices, let Q be the $(n+\ell)$-by-n identity matrix, let $\mathscr{F}(K)$ be a fretsaw-forest extension of K, and let C be a c-by-n matrix. Then,*

$$\text{null}\left(\begin{bmatrix} \mathscr{F}(K) \\ CQ^T \end{bmatrix}\right) \subseteq \text{null}\left(\begin{bmatrix} K \\ C \end{bmatrix} Q^T\right).$$

Proof. Let

$$x \in \text{null}\left(\begin{bmatrix} \mathscr{F}(K) \\ CQ^T \end{bmatrix}\right).$$

By definition, $\mathscr{F}(K)x = 0$ and $CQ^Tx = 0$. By Theorem 3.1, since $x \in \text{null}(\mathscr{F}(K))$, then $x \in \text{null}(QKQ^T)$. Therefore, $QKQ^Tx = 0$ and $KQ^Tx = Q^TQKQ^Tx = 0$. Therefore,

$$\begin{bmatrix} K \\ C \end{bmatrix} Q^Tx = \begin{bmatrix} KQ^Tx \\ CQ^Tx \end{bmatrix} = 0,$$

which concludes the proof of the lemma. □

The containment in the null-space and the special padding structures enable us to get a stronger characteristic which is summarized in the next lemma.

Lemma 3.3. *For the above $K, \mathscr{F}(K), Q$ and C, let $\widehat{N}$ be an $(n+\ell)$-by-s matrix whose columns form a basis for null $\left(\begin{bmatrix} \mathscr{F}(K) \\ CQ^T \end{bmatrix}\right)$, then $(Q^T\widehat{N}) = \text{null}\left(\begin{bmatrix} K \\ C \end{bmatrix}\right)$.*

Proof. Let $r \in \mathbb{R}^s$. We first show that $Q^T\widehat{N}r \in \text{null}\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right)$. By definition, $\widehat{N}r \in \text{null}\left(\begin{bmatrix} \mathscr{F}(K) \\ CQ^T \end{bmatrix}\right)$. By Lemma 3.2, $\widehat{N}r \in \text{null}\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T Q^T\right)$. Therefore, $\begin{bmatrix} K^T & C^T \end{bmatrix}^T Q^T\widehat{N}r = 0$ so $Q^T\widehat{N}r \in \text{null}\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right)$. This shows that null $\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right) \supseteq \{Q^T\widehat{N}r | r \in \mathbb{R}^s\}$.

We now show the containment in the other direction. Let $x \in \text{null}\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right)$. By Theorem 3.1, there exists an $(n+\ell)$-by-n matrix P such that $K = P^T\mathscr{F}(K)P$ and $Q^TP = I_n$. By the definition of x,

$$\begin{bmatrix} K \\ C \end{bmatrix} x = 0$$
$$\begin{bmatrix} P^T\mathscr{F}(K)P \\ CQ^TP \end{bmatrix} x = 0$$
$$\begin{bmatrix} P^T\mathscr{F}(K) \\ CQ^T \end{bmatrix} Px = 0.$$

Since $\mathscr{F}(K)$ is SPSD, there exist a matrix U, such that $\mathscr{F}(K) = U^TU$. This implies that $Px \in \text{null}(\mathscr{F}(K))$, since

$$P^T\mathscr{F}(K)Px = 0$$
$$x^TP^T\mathscr{F}(K)Px = 0$$
$$x^TP^TU^TUPx = 0$$
$$\|UPx\|^2 = 0$$
$$UPx = 0$$
$$U^TUPx = 0$$
$$\mathscr{F}(K)Px = 0.$$

Therefore,

$$\begin{bmatrix} \mathscr{F}(K) \\ CQ^T \end{bmatrix} Px = 0 .$$

By the definition of $\widehat{N}$, there exists a vector r such that $Px = \widehat{N}r$. Multiplying the last equality by Q^T, we get $x = Q^T Px = Q^T \widehat{N} r$. This shows the other containment direction and concludes the proof of the lemma. □

3.3. Rectangular inverse iteration

Once we construct the augmented fretsaw-forest matrix $\widehat{K}$, we compute a basis $\widehat{N}$ for its null space using rectangular inverse iteration [18]; see [21] for additional background on inverse iteration. This process requires the sparse *LU* factorization with partial pivoting of $\widehat{K}$. Once this factorization is computed, the algorithm usually performs a few iterations involving solving linear systems of equations of the form $UX = B$ where X and B have only a few columns; between every two iterations, the algorithm orthogonalizes X. In some rare cases the algorithm becomes somewhat more complex, but not more expensive. As long as the dimension of the computed null space is small, the orthogonalization is not a dominant cost.

In practice, there are cases where there are zero columns and rows in K and therefore also in $\mathscr{F}(K)$ and $\widehat{K}$. In these cases, we first remove the zero rows and columns from $\widehat{K}$. This has no algebraic or theoretical significance, but was needed for our specific implementation of the rectangular inverse iteration.

Another issue that arises in practice is the treatment of single point constraints. Single point constraints correspond to rows in C (and in $\widehat{K}$) with only a single nonzero each. In exact arithmetic, the corresponding variables in the null space are always zero. In practice, the inverse iteration process assigns small values to these variables that depend on the scaling of the single nonzero rows in C. In order to handle this numerical behavior, and to avoid the question of proper scaling of the single nonzero rows in C, we chose to force a zero in the corresponding variables by removing those columns from $\widehat{K}$.

The inverse iteration result $\widehat{N}$ is an $(n+\ell)$-by-$\hat{s}$ matrix. Our method truncates the last ℓ rows of $\widehat{N}$ and then orthonormalize it. The result is an n-by-s matrix N. (Note that s can be smaller than $\hat{s}$). Lemma 3.3 guarantees that the computed null space is exactly the null space of $\begin{bmatrix} K^T & C^T \end{bmatrix}^T$.

It is intuitive that the sparsified structure is weaker than the original structure. This can also be quantified theoretically in a purely algebraic way: By [33, Lemma 8.14] and using [4, Proposition 6.1] the generalized eigenvalues $\lambda(\mathscr{F}(K), QKQ^T)$ are bounded by 1. This behavior affects the inverse iteration process. For a given threshold, the inverse iterations outputs vectors which are linear combinations of null vectors of $\widehat{K}$ and of vectors that correspond to small singular values. In other words, the generated $\widehat{N}$ spans a linear space that contains the null space of $\widehat{K}$ and some other vectors. Fortunately, we can still recover the null space of $\begin{bmatrix} K^T & C^T \end{bmatrix}^T$. The following lemma shows exactly how to do this.

Lemma 3.4. *For the above $K, \mathscr{F}(K), Q$ and C, let $\widehat{N}$ be an $(n+\ell)$-by-$\hat{s}$ matrix whose columns span a linear space that contains null $\left(\begin{bmatrix} \mathscr{F}(K) \\ CQ^T \end{bmatrix}\right)$. Let V be a matrix, such that span $(V) = \text{null}\left(\begin{bmatrix} K \\ C \end{bmatrix} Q^T \widehat{N}\right)$, then span $(Q^T \widehat{N} V) = \text{null}\left(\begin{bmatrix} K \\ C \end{bmatrix}\right)$.*

Proof. Let U be an $\hat{s}$-by-s matrix such that span $(\widehat{N}U) = \text{null}\left(\begin{bmatrix} \mathscr{F}(K) \\ CQ^T \end{bmatrix}\right)$. By Lemma 3.3, span $(Q^T \widehat{N} U) = \text{null}\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right)$.

Let $x \in \text{null}\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right)$. By definition, there exists a vector r such that $x = Q^T \widehat{N} U r$. By the definition of x,

$$\begin{bmatrix} K \\ C \end{bmatrix} x = 0$$
$$\begin{bmatrix} K \\ C \end{bmatrix} Q^T \widehat{N} U r = 0 .$$

Since $Ur \in \text{null}\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T Q^T \widehat{N}\right)$, there exist a vector t such that $Ur = Vt$. Therefore, $x = Q^T \widehat{N} V t$. This shows that null $\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right)$ is contained in span $(Q^T \widehat{N} V)$.

In order to complete the proof, we need to show that every vector in span $(Q^T \widehat{N} V)$ is also in null $\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right)$. Let $x = Q^T \widehat{N} V t$ for an arbitrary t. By the definition of V,

$$\begin{bmatrix} K \\ C \end{bmatrix} x = \begin{bmatrix} K \\ C \end{bmatrix} Q^T \widehat{N} V t = 0t = 0 .$$

This shows that $x \in \text{null}\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T\right)$ and concludes the proof of the lemma. □

The Lemma implies the following final step in the algorithm: compute an orthonormal basis for $\left(\begin{bmatrix} K^T & C^T \end{bmatrix}^T Q^T \widehat{N}\right)$. This is usually inexpensive since $\widehat{N}$ only has a few columns. If we denote by V the matrix whose columns spans this null space, then the null space of $\begin{bmatrix} K^T & C^T \end{bmatrix}^T$ is exactly $Q^T \widehat{N} V$.

In general, triangular factorizations (Cholesky, LDL^T, LU, or QR) of large finite-elements problems are expensive. But our method is efficient because triangular factorizations of forest-based fretsaw extensions are sparse and cheap to compute. In particular, if C consists of single-unknown constraints and the rigidity graph of K is connected, then $\widehat{K}$ can be factored using sparse symmetric factorizations with no fill outside the original element matrices [33, Lemma 8.13].

The analysis of the cost of the factorization in [33] cannot be directly applied, however, the *LU* factorization of $\widehat{K}$ is usually cheap to compute and produce factors with small amount of fill. There are three gaps between that analysis and the computation that we use here. First, the analysis in [33] assumes that the rigidity graph is connected. But in many real-world models, even models with many solid elements, the rigidity graph is disconnected. As the number of connected components in the rigidity graph increases, so does the fill in the *LU* factors that our method computes. Second, the analysis in [33] assumes that C consists of single-unknown equality constraints; if this is not the case, the structure of C introduces additional fill that is hard to characterize theoretically. Third, the analysis in [33] bounded fill and work in symmetric factorization (Cholesky or LDL^T), whereas rectangular inverse iteration requires an *LU* factorization. Existing theory on fill in sparse *LU* factorization [3,5,10,15,20] cannot be directly applied to our forest or tree-based fretsaw extensions. Nevertheless, the existing body of theory suggests that the *LU* factorization of tree-based fretsaw extension should be cheap relative to the original finite element model.

Our experimental results, which are reported below, indicate that in spite of the lack of complete theoretical analysis, the *LU* factorization of $\widehat{K}$ is indeed inexpensive.

4. Numerical results

We have implemented a prototype of the fretsaw extension method. Our implementation runs under MATLAB version 7.4 [27], although all its computation-intensive parts are implemented in C (and are invoked by MATLAB through its CMEX interface). The code factors $\widehat{K}$ using MATLAB's `lu` function with pivoting threshold 1.0 (partial pivoting), which uses UMFPACK version 5.0 [9]. However,

Table 1
Model statistics for the scaling experiments. The parameter n stands for the number of unknowns, $|\{e\}|$ is the number of elements, nnz (K) is the number of nonzeros in the stiffness matrix, and $|\{\Omega\}|$ is the computed number of subdomains used in the Schwarz domain decomposition preconditioner experiments.

Cube side size	n	$\lvert\{e\}\rvert$	$\frac{\mathrm{nnz}(K)}{10^3}$	$\lvert\{\Omega\}\rvert$
11	264	245	5	15
28	2646	4048	90	59
36	5739	9458	210	111
53	15,576	27,853	618	263
69	34,179	63,763	1414	576
82	54,069	103,052	2286	827
91	75,507	145,423	3228	1151
108	121,023	237,513	5269	1757
121	169,449	336,087	7456	2402
132	218,838	437,026	9695	3094
155	351,828	711,269	15,777	4833
174	492,729	1000,395	22,253	6646

we use a C function to solve sparse triangular linear systems; it is much faster than matlab's \ operator. Running times were measured on a 64-bit Linux 2.6 system with a 1.6 GHz AMD Opteron 242 processor and an 8 GB of RAM. The reported running times are wall-clock times.

The numerical experiments whose results are reported in this section all compute the null space of $K_C = \begin{bmatrix} K^T & C^T \end{bmatrix}^T$. The inverse iteration code is the same as in [18 Section 5], with threshold 10^{-8} and the initial estimate of the dimension of the null space is set to the common ℓ_e of the model.[1] We refer to our method in this section as the *fretsaw-forest* method. Its output is the n-by-s matrix N. We compare the fretsaw-forest method with a naive method to compute the null space of K_C, which performs inverse iteration on the triangular factors of K_C itself. We refer to this algorithm as the *direct method* below. The result of this experiment is also an n-by-s matrix N whose columns are an orthonormal basis of the null space of K_C.

In the first part of the experiments we also compare the fretsaw-forest method to locally optimal block preconditioned conjugate gradient (LOB-PCG) with multiplicative Schwarz domain decomposition preconditioner [24]. We use the implementation within Hypre 2.0 [25]. The LAPACK and BLAS that are used as subroutines in this process (and also within UMFPACK) are MATLAB's (in this case, ACML).

We used the Hypre 2.0 default parameters for both the LOB–PCG solver and the Schwarz preconditioner. The subdomains in the Schwarz preconditioner were computed using domain agglomeration; the specific number of subdomains for each problem is detailed in Table 1. The overlap between subdomains was set to minimal (one layer), and a direct solver was used within the subdomains. In experiments not reported here, we verified that for our specific problems these parameters performed better than other reasonable parameter sets.

We compare the accuracy of the methods using two metrics. The first is discrete: the dimension s of the discovered null-space. The second is continuous and contains more information: the relative error $\|K_C N\|_2 / \|K_C\|_2$. We compute the numerator $\|K_C N\|_2$ but approximate the denominator $\|K_C\|_2$ by the maximal absolute value of elements in K_C.

4.1. Performance scaling on a model problem

In the first set of experiments we compute the null space of a uniform symmetric elastic k-by-k-by-k cubes, with no constraints (namely, $K = K_C$). The cubes are discretized using a nonuniform mesh of tetrahedra. Each tetrahedron is built out of 6 struts. Each vertex of a tetrahedron is a joint in this struts structure. Fig. 4.1 shows such a cube. We used TETGEN version 1.4 [34] to generate the mesh. The tetrahedra have a minimum radius-edge ratio of 2 and a volume bounded by 10. Each tetrahedron is modeled by a single 12-by-12 element matrix. Table 1 summarizes the models' characteristics.

Fig. 4.2a and e present the running times of the fretsaw-forest method, the direct method, and the preconditioned LOB–PCG method as a function of the problem size. They show that the fretsaw-forest method is highly efficient for problems of modest to large sizes. It is significantly faster than the other methods, reaching up to an order of magnitude faster for the larger problems. Fig. 4.2a–d are limited to smaller problems where there was enough memory for the direct method to run. Fig. 4.2a demonstrates that the direct method is comparable and even better then the preconditioned LOB–PCG for problems where the factors of the matrix fit into the memory. This shows that our comparison with this fairly naive method is reasonable.

Fig. 4.2c demonstrates the linear scaling behavior of the fretsaw-forest method; The running time of the entire process is linear in the problem size and so are the independent running times of the fretsaw-forest construction, factorization and inverse iterations operations. The construction of the fretsaw-forest extension takes most of the running time. Measurements not presented here show that about 20% of this time is spent on computing the null spaces of the element matrices k_es, and more than 70% is spent on computing the rigidity graph. This means that when additional domain-specific information is used to construct the rigidity graph, the running time of this part might be reduced by an order of magnitude. The scaling behavior for the direct method is clearly super-linear, as demonstrated by Fig. 4.2d.

The reason for the linear factorization and inverse-iteration times in the fretsaw-forest case is presented in Fig. 4.2b. This figure presents the number of nonzeros in the LU factors in both the fretsaw-forest and the direct method. The nonzeros number in the LU factors of the fretsaw-forest sparsification appears to be linear in the problem size. This demonstrates that the fretsaw sparsification is effective with respect to the number of nonzeros in LU factorizations, not only with respect to nonzeros in symmetric factorizations. A closer inspection of the data indicates that the LU factors

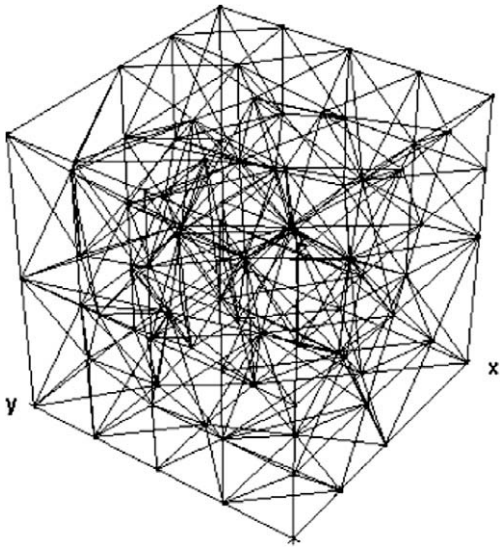

Fig. 4.1. Model problem for the scaling experiments.

[1] This code (but using Matlab's triangular solver) is publicly available at http://www.tau.ac.il/~stoledo/research.html.

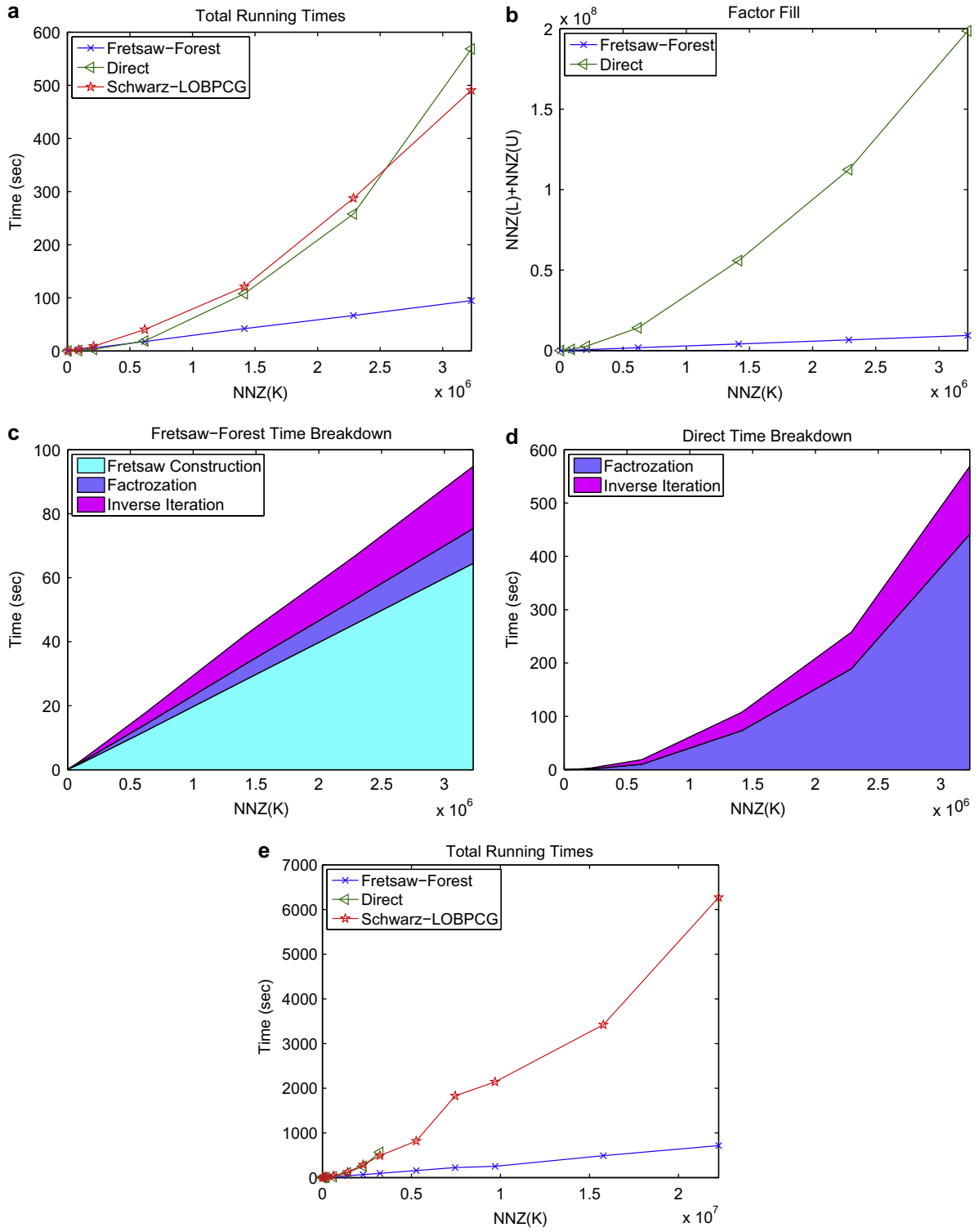

Fig. 4.2. Performance of scaling experiments set. Parts (a)–(d) focus on small sizes, where the direct method had enough memory to complete. Part (a) shows the total running time of the null space computation. Part (b) shows the total number of nonzeros in the *LU* factors in the direct and fretsaw-forest algorithms. Parts (c) and (d) show the breakdown of the running times into the different phases of the direct and fretsaw-forest algorithms. Part (e) shows the total running time of the null space computation for the complete set of test problems.

do fill somewhat. This is not surprising given that the ordering that UMFPACK computes is different from the ordering that [33, Lemma 8.13] assumes, and given that we use partial pivoting.

The null space dimension in all the experiments was correctly computed to be 6. Fig. 4.3 shows the relative errors in the scaling experiment. The relative error of the direct method was always 10^{-10} or less (when there was enough memory to run it). The relative error in the fretsaw forest method is worse: the method loses between 3 and 5 digits of accuracy relative to the direct method, but in all cases it is still smaller than 10^{-4}. The relative error of the iterative method was always smaller than 10^{-7} and was less affected by the problem size. The loss of accuracy in the fretsaw-for-

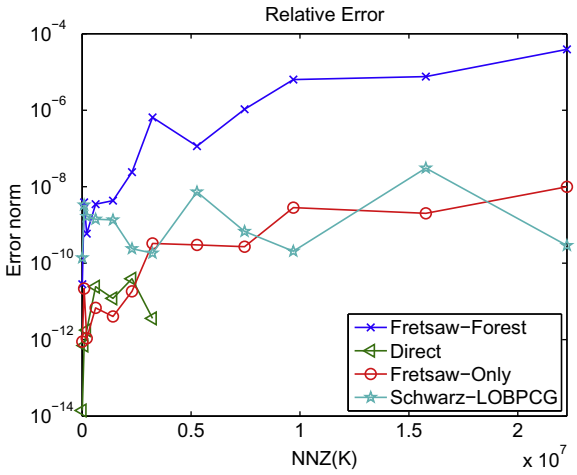

Fig. 4.3. Relative errors in the scaling experiment. The blue (Xs), green (triangles), and cyan (stars) lines present $\|K_C N\|_2 / \max_{i,j}|K_C(i,j)|$. The red (circles) line present $\|\widehat{K}\widehat{N}\|_2 / \max_{i,j}|\widehat{K}(i,j)|$. Note there are no data points for the direct method for nnz $(K) > 3.228e6$ where there was not enough memory to complete the computation.

Table 2
Model statistics for the industrial experiments set. The parameter n stands for the number of unknowns with a nonempty column in K, $|\{e\}|$ is the number of elements, nrows (C) is the number of rows in C, #SPC is the number of single point constraints that correspond to nonempty columns in K, nnz (K) is the number of nonzeros in the stiffness matrix (without the constraints), and the parameter #CC stands for the number of connected components in the rigidity graph.

Name	Description	n	$\|\{e\}\|$	$\frac{\text{nnz}(K)}{10^3}$	nrows (C)	#SPC	#CC
cyl	Cylinder	648	150	41	0	108	1
lg0d	Half Cylinder bracket	51,264	8665	2149	1248	512	2
lg0m	Transmission part	99,927	18,681	7322	540	466	3
lg0q	Cube	31,125	28,260	786	0	0	1
lg0r	Brake	90,738	21,024	5341	1338	144	338
md0c	Exhaust Manifold	52,938	9231	2769	4212	0	142
vl0d	Crank Shaft	149,775	45,200	10,715	96	337	50
vl0g	T-Joint	136,242	27,081	10,314	0	1909	2
vl0r	Brake	126,399	27,365	8598	4851	132	82
plan10g	Unknown	660	100	28	0	72	1
sm0d	Unknown	12,444	2276	572	0	0	2
nxn_plm	Aircraft fuselage section	39,432	9162	1991	11,640	386	16

est method is caused primarily by the fretsaw approximation itself, not because we compute the null vectors of $\widehat{K}$ inaccurately: the relative error in the null vectors of $\widehat{K}$ is almost always under 10^{-10}. The approximate values of $\|\widehat{K}\|_2$ and $\|K_C\|_2$ were very close. Because of the missing continuity relations in the fretsaw approximation, $\mathscr{F}(K)$ has smaller nonzeros eigenvalues than K. The small eigenvectors of $\mathscr{F}(K)$ mix with the true null vectors of the structure, which $\mathscr{F}(K)$ and K share, and cause the computed eigenvectors to be less accurate. In other words, the loss of accuracy

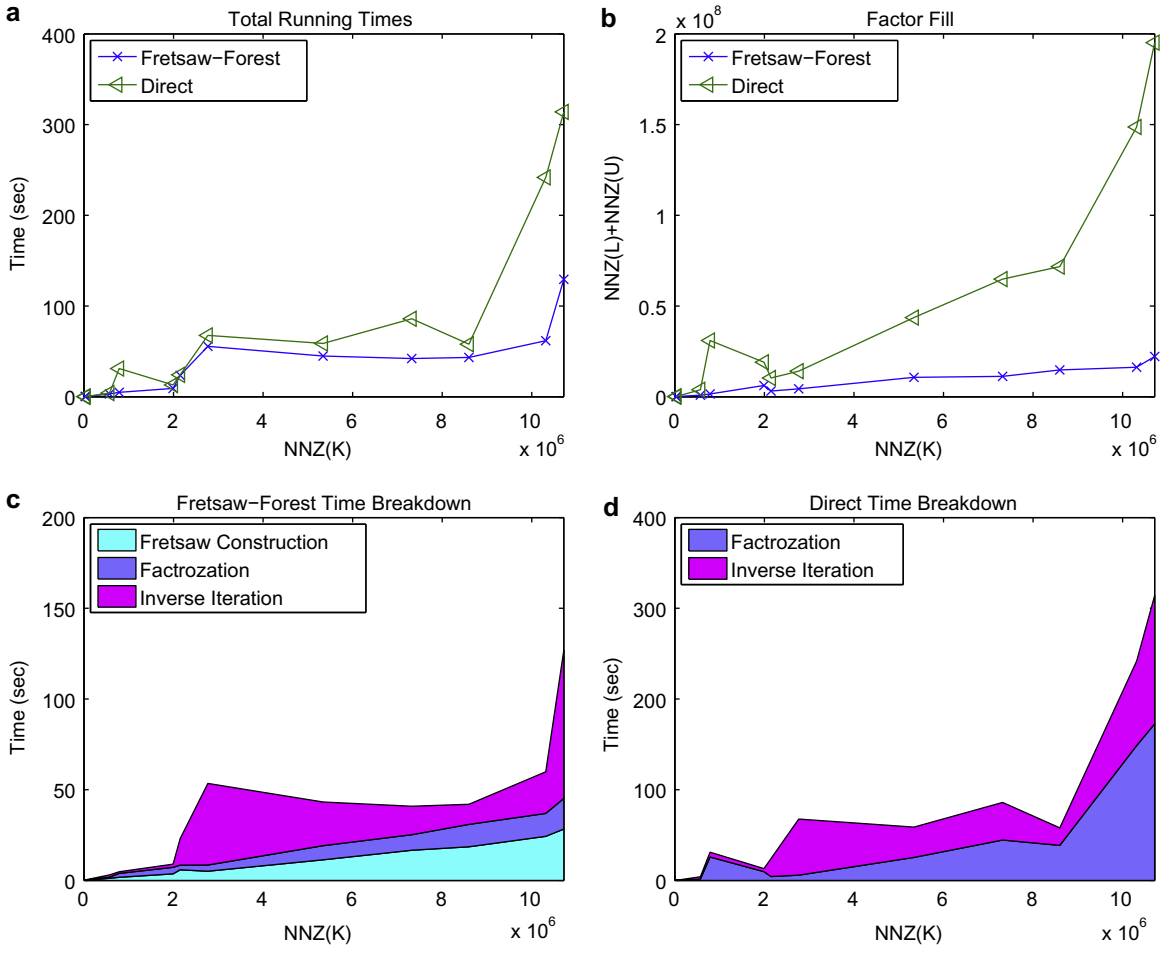

Fig. 4.4. Performance of industrial experiments set without constraints. Part (a) shows the total running time of the null space computation. Parts (c) and (d) show the breakdown of the null space computation times. Part (b) shows the total number of nonzeros in the *LU* factors.

appears to be due to the ill conditioning of $\mathscr{F}(K)$, which shrinks the spectral gap between numerically zero and numerically nonzero eigenvalues.

4.2. Industrial structural analysis models experiments

In the second set of experiments we used industrial real-world structural mechanics models. The models were produced by two different versions of Nastran, MSC.Nastran and NX.Nastran. All the models contain multiple element types and most of them contain single-point and multi-point constraints. Table 2 summarizes the main properties of the models.

We first computed the null space of these models without the constraints. In such experiments the null-space is not empty but contains at least the rigid-body motions of the body. In these experiments the rigidity graph is often not connected, so the fretsaw-forest methods indeed generate a forest and not a tree. The number of connected components is shown in Table 2.

Fig. 4.4 shows the performance of the fretsaw-forest method and the direct method in this case. We first focus on the factor-fill graph (part (b) of the figure). As in the scaling experiments, the fill in the LU factors in the fretsaw-forest method is close to linear, and in the direct-method is clearly not linear; in most cases it is an order of magnitude higher. The slightly nonlinear behavior of the fill in the fretsaw-forest method may be due to the fact that the rigidity graph is not connected, which makes the fretsaw sparsification less effective.

We now direct our attention to the running times breakdown graphs in Fig. 4.4c and d. The factorization times in both figures are consistent with the data in the fill graphs. The inverse iterations time is also related to the size of the fill, but is not always monotone and smooth as in the scaling experiments. A closer inspection reveals that the nonmonotonicity is caused by differences in the null space dimensions and in the number of inverse iterations. Models with larger null space dimensions demand more time in every iteration and more iterations to converge.

Finally, the overall performance of the fretsaw-forest method, shown in Fig. 4.4c, demonstrates that this new method is faster than the direct method, especially on large models.

Fig. 4.5 shows the performance of the methods with constraints. The added constraints increase the fill in the LU factors and increase the running times. See, for example, the bump around 2×10^6, which corresponds to the matrix 'nxn_plm', which has a large number of constraints. On one matrix, 'vl0d', the direct method ran out of memory whereas the fretsaw-forest method ran to completion. The graphs in this figure do not include the results for this matrix. Nevertheless, the qualitative behavior is similar to the behavior in Fig. 4.4: even with constraints, the fretsaw method is faster and requires less memory (especially on larger models).

Table 3 presents the accuracy and the relative errors of the methods, with and without constraints. The table indicates that the fretsaw-forest method returned a null space with the same dimension as that returned by the direct method in all but four

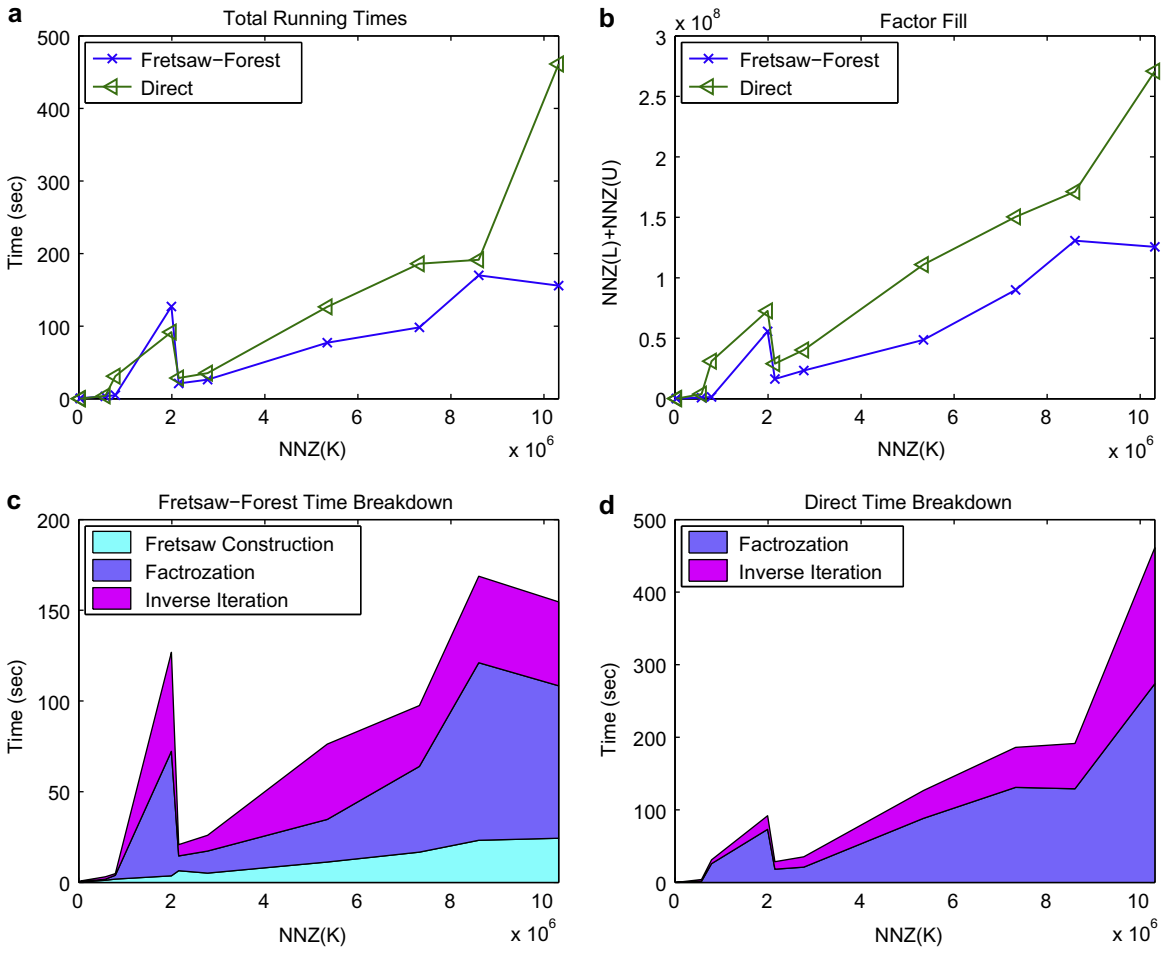

Fig. 4.5. Performance of industrial experiments set with constraints. Part (a) shows the total running time of the null space computation. Parts (c) and (d) show the breakdown of the null space computation times. Part (b) shows the total number of nonzeros in the *LU* factors.

Table 3
Computed null space sizes and relative errors in the industrial experiments set. The relative error (rel-err) is $\|K_C N\|_2 / \max_{i,j}|K_C(i,j)|$. In cases where the computed null space size was 0, the relative error is irrelevant and marked with a hyphen. In cases there was inconsistency between the returned results of the methods the result is marked with double asterisks.

	Without constraints				With constraints			
	Fretsaw-Forest		Direct		Fretsaw-Forest		Direct	
	Size	Rel-err	Size	Rel-err	Size	Rel-err	Size	Rel-err
plan10g	6**	4.4e−10	6	1.4e−12	0	–	0	–
cyl	5**	4.2e−7	6	3.6e−14	0	–	0	–
sm0d	6	6.3e−7	6	6.1e−10	No constraints available			
lg0q	1	2.1e−11	1	9.5e−13	1	2.1e−11	1	9.5e−13
lg0d	12	1.7e−35	12	1.8e−10	0	–	0	–
md0c	36	1.2e−7	36	5.9e−10	3**	1.4e−10	3	3.9e−14
lg0r	8	2.3e−18	8	4.5e−18	5**	2.6e−8	4	3.7e−8
lg0m	6	3.3e−7	6	5.3e−11	0	–	0	–
vl0r	4	5.9e−18	4	1.7e−18	0	–	0	–
vl0g	6**	8.3e−8**	6	2.3e−9	0	–	0	–
vl0d	10**	4.1e−7**	6	7.6e−10	11**	2.3e−1	Out of memory	
nxn_plm	1	3.5e−22	1	3.5e−22	8**	1.1e−5	0	–

cases. The relative errors behaved in a similar way to the relative errors in the scaling experiments; in most cases the method loses 2–5 digits of accuracy relative to the direct method.

We now explore the four cases where the fretsaw-forest method and the direct method computed different number of null space basis vectors (marked with double asterisks in the table). In all the cases, the relative error that was achieved by both methods was very small. Therefore, the inconsistency is clearly a threshold setting issue. The output of the inverse iteration method is highly sensitive to the threshold. Further experiments with different inverse iteration thresholds produced different numbers of null-space vectors both in the direct and fretsaw-tree methods, all with small relative errors.

On 'nxn_plm' we obtained consistent results in the case without constraints, but the results varied when we changed the inverse-iteration threshold from 10^{-8} to 10^{-6} and to 10^{-4}. This is another example for the difficulty with the inverse-iterations method. Moreover, when we added constraints we got more null vectors which is really not what one expects. We explored the computation of null $\left(\begin{bmatrix} K^T & C^T \end{bmatrix} Q^T \widehat{N}\right)$. The singular values there are all very small (and indeed the relative error is small) and singular vectors with very small residuals where detected as null vectors. It is consistent with the inverse-iteration sensitivity to thresholds that we saw in the experiments with this matrix with no constraints.

The last experiment that we discuss is 'vl0d' with constraints. This is the only experiment with significantly large relative error. Further experiments with different thresholds for this matrix, with and without constraint revealed that the computed null space is very sensitive to the threshold. In the case without constraints, the direct method exhibited the same behavior.

We note that when the dimension of the computed null space is too large, this can be easily identified by calculating the relative error. A practical system can automatically detect this error and then try to run the same inverse iteration method with a different threshold.

We also note that the loss of 2–5 digits of accuracy in the computed null vectors is inherent in the fretsaw-forest approximation, but the reliability issues are due to the iterative solver that we use, which is based on inverse iteration. We comment on this further below.

5. Conclusions

We presented a novel method for computing the null space of finite element models, including models with equality constraints. The method is purely algebraic (but requires access to the element matrices, not just to the assembled matrix). It does not require access to the geometry of the model.

Our definition of rigidity relationship, which lies at the core of the new method, might be useful in generalizing the methods of [11,29,8,22,23] by eliminating the need for geometric or domain specific knowledge, but we did not check this in detail.

Our experiments show that the method scales extremely well on 3-dimensional model problems, leading to work and memory requirements that are linear in the size of the model. On this model problem, the new method outperforms two established methods, one based on inverse iteration the factors of the matrix, and the other based on a preconditioned iterative eigensolver (LOB–PCG with a Schwarz preconditioner). The method performs well and outperforms an established method (inverse iteration) on industrial models as well, including models with many equality constraints.

The computed null vectors are 2–5 decimal digits less accurate than those computed by established methods, which are computationally more expensive. Both the new method and one of the established methods are somewhat sensitive to a threshold parameter in the iterative solver that we use, which is based on inverse iteration.

We believe that an iterative solver that is more sophisticated than the simple inverse iteration solver may be more robust. Such a solver would use the fretsaw approximation as a preconditioner, but would use true residuals (whereas the inverse iteration solver that we use now uses residuals with respect to $\widehat{K}$, not K). We leave this to future research.

Acknowledgement

The industrial problems were provided to us by Tom Kowalski (MSC.Nastran) and Leonard Hofnang (NX.Nastran). Thanks to Ray Tuminaro for explaining to us the null-space issues in smoothed aggregation. Thanks to the two anonymous referees for numerous suggestions and corrections. This research was supported by an IBM Faculty Partnership Award, by Grant 848/04 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), and by Grant 2002261 from the United-States-Israel Binational Science Foundation.

References

[1] Haim Avron, Esmond Ng, Sivan Toledo. Using perturbed QR factorizations to solve linear least-squares problems. SIAM J. Matrix Anal. Comput., accepted for publication.
[2] Denis S. Bernstein, Matrix Mathematics: Theory, Facts, and Formulas with Applications to Linear Systems Theory, Princeton University Press, 2005.

[3] Jean R.S. Blair, Barry Peyton, An introduction to chordal graphs and clique trees, in: Alan George, John R. Gilbert, Joseph W.H. Liu (Eds.), Graph Theory and Sparse Matrix Computation, Springer-Verlag, 1993, pp. 1–29.
[4] Erik G. Boman, Bruce Hendrickson, Support theory for preconditioning, SIAM J. Matrix Anal. Appl. 25 (3) (2004) 694–717.
[5] Igor Brainman, Sivan Toledo, Nested-dissection orderings for sparse LU with partial pivoting, SIAM J. Matrix Anal. Appl. 23 (2002) 112–998.
[6] K.C. Park, C.A. Felippa, M.R. Justino Filho, The construction of free–free flexibility matrices as generalized stiffness inverses, Comput. Struct. 68 (4) (1998) 411–418.
[7] Coleman, Pothen, The null space problem I. Complexity, SIJADM: SIAM J. Algebr. Discrete Methods 7 (1986).
[8] G.M. Reese, D.M. Day, M.K. Bhardwaj, J.S. Peery, Mechanism free domain decomposition, Comput. Methods Appl. Mech. Engrg. 192 (2003) 763–776.
[9] Timothy A. Davis, Algorithm 832: UMFPACK V4.3 – an unsymmetric-pattern multifrontal method, ACM Trans. Math. Softw. 30 (2) (2004) 196–199.
[10] Alex Druinsky, Sivan Toledo, Factoring matrices with a tree-structured sparsity pattern, Electronic Transactions on Numerical Analysis, submitted for publication.
[11] C. Farhat, K.H. Pierson, M. Lesoinne, The second generation of FETI methods and their application to the parallel solution of large-scale linear and geometrically nonlinear structural analysis problems, Comput. Methods Appl. Mech. Engrg. 184 (2000) 333–374.
[12] Charbel Farhat, Michel Gérardin, On the general solution by a direct method of a large scale singular system of linear equations: application to the analysis of floating structures, Int. J. Numer. Methods Engrg. 41 (1997) 675–696.
[13] Charbel Farhat, Francois-Xavier Roux, A method of finite element tearing and interconnecting and its parallel solution algorithm, Int. J. Numer. Methods Engrg. 32 (1991) 1205–1227.
[14] C.A. Felippa, K.C. Park, A direct flexibility method, Comput. Methods Appl. Mech. Engrg. 149 (1997) 319–337.
[15] A. George, E. Ng, On the complexity of sparse QR and LU factorization of finite-element matrices, SIAM J. Sci. Statist. Comput. 9 (1988) 849–861.
[16] M. Geradin, D. Rixen, Mechanical Vibrations: Theory and Application to Structural Dynamics, Wiley and Sons, 1997.
[17] Gilbert, Heath, Computing a sparse basis for the null space, SIJADM: SIAM J. Algebr. Discrete Methods 8 (1987).
[18] Craig Gotsman, Sivan Toledo, On the computation of null spaces of sparse rectangular matrices. SIAM J. Matrix Anal. Appl., in press.
[19] Jonathan Hu, Charles Tong, Ray S. Tuminaro, ML2.0 Smoothed Aggregation User's Guide, June 04, 2000.
[20] Yifan Hu, Jennifer Scott, Ordering techniques for singly bordered block diagonal forms for unsymmetric parallel sparse direct solvers, Numer. Linear Algebra Appl. 12 (9) (2005) 877–894.
[21] Ilse C.F. Ipsen, Computing an eigenvector with inverse iteration, SIAM Rev. 39 (1997) 254–291.
[22] A. Kaveh, K. Koohestani, An efficient graph-theoretical force method for three-dimensional finite element analysis, Commun. Numer. Methods Engrg. (2007).
[23] A. Kaveh, K. Koohestani, N. Taghizadieh, Efficient finite element analysis by graph-theoretical force method, Finite Elements Anal. Design 43 (2007) 543–554.
[24] A.V. Knyazev, M.E. Argentati, I. Lashuk, E.E. Ovtchinnikov, Block locally optimal preconditioned eigenvalue xolvers (blopex) in hypre and petsc, SIAM J. Sci. Comput. 29 (5) (2007) 2224–2239.
[25] Lawrence Livermore National Laboratory, Center for Applied Scientific Computing (CASC), University of California. HYPRE User's Manual – Software Version 2.0.0, 2006.
[26] Jan Mandel, Balancing domain decomposition, Commun. Numer. Methods Engrg. 9 (1993) 233–241.
[27] The MathWorks Matlab version 7.4. Software Package, January 2007.
[28] Ng. Esmond, A scheme for handling rank-deficiency in the solution of sparse linear least squares problems, SIAM J. Sci. Statist. Comput. 12 (5) (1991) 1173–1183.
[29] M. Papadrakakis, Y. Fragakis, An integrated geometric–algebraic method for solving semi-definite problems in structural mechanics, Comput. Methods Appl. Mech. Engrg. 190 (2001) 6513–6532.
[30] Daniel J. Pierce, John G. Lewis, Sparse multifrontal rank revealing qr factorization, SIAM J. Matrix Anal. Appl. 18 (1) (1997) 159–180.
[31] Alex Pothen, Sparse null basis computations in structural optimization, Numer. Math. 55 (5) (1989) 501–519.
[32] Gil Shklarski, Combinatorial Preconditioners for Finite-Element Matrices and Other Contributions to Numerical Linear Algebra. Ph.D. Thesis, Tel-Aviv University, August 2008.
[33] Gil Shklarski, Sivan Toledo, Rigidity in finite-element matrices: Sufficient conditions for the rigidity of structures and substructures, SIAM J. Matrix Anal. Appl. 30 (1) (2008) 7–40.
[34] Hang Si. TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator: Users's Manual for Version 1.4, January 2006. Available in URL: <http://tetgen.berlios.de>.
[35] P. Vaněk, Acceleration of convergence of a two level algorithm by smoothing transfer operators, Appl. Math. 37 (1992) 265–274.
[36] P. Vaněk, J. Mandel, M. Brezina, Algebraic multigrid based on smoothed aggregation for second and fourth order problems, Computing 56 (1996) 179–196.