ON THE COMPUTATION OF NULL SPACES OF SPARSE RECTANGULAR MATRICES

CRAIG GOTSMAN AND SIVAN TOLEDO

ABSTRACT. Computing the null space of a sparse matrix, sometimes a rectangular sparse matrix, is an important part of some computations, such as embeddings and parametrization of meshes. We propose an efficient and reliable method to compute an orthonormal basis of the null space of a sparse square or rectangular matrix (usually with more rows than columns). The main computational component in our method is a sparse LU factorization with partial pivoting of the input matrix; this factorization is significantly cheaper than the QR factorization used in previous methods. The paper analyzes important theoretical aspects of the new method and demonstrates experimentally that it is efficient and reliable.

1. INTRODUCTION

We propose a new method for computing an orthonormal basis for the null space of a rectangular m-by-n matrix A with $m \geq n$. The main computational component of the new method is a conventional LU factorization with partial pivoting of A. For many classes of sparse matrices, an appropriate preordering of the columns leads to sparse factors, usually significantly sparser than the QR factors and than the factors of any rank-revealing factorization [21, 23]. There are several recent high-quality sparse LU codes that can perform partial pivoting [2, 17, 18, 19, 27, 28]. Experts on sparse-matrix factorizations, including experts on sparse QR, believe that sparse LU factorizations with partial pivoting is intrinsically cheaper than sparse QR[1]. Therefore, the new method is particularly suitable for large sparse matrices with a small-dimensional null space. (Because we compute an orthonormal basis, in the computation of a high-dimensional null space, the cost of orthogonalizing the null vectors dominates.)

We also revisit a somewhat more expensive method, which is based on a QR factorization. This method is not new, but is not widely known either. Because of this, and because it can be used to compute not only the null space, but also additional singular triplets corresponding to small singular values, we mention it here briefly too.

There are several applications to the computation of the null space, and more generally, to the computation of singular vectors or singular subspaces associated with small singular values. The first application that we describe involves computation with matrices derived from graphs. We start with a discussion of the square and normal case, and then describe extensions of these applications to rectangular matrices.

[1]Private discussions with John Gilbert, Pontus Matstoms, and Esmond Ng in April and June 2005.

The spectrum of matrices related to the adjacency matrix of a graph is extremely interesting. Typically the lowest eigenvalues and eigenvectors are the most useful, as they characterize various properties of the graph. The most famous example is the second smallest eigenvalue of the graph Laplacian, which characterizes how strongly it is connected, hence its mixing rate [11]. The smallest eigenvalue is zero, corresponding to a fixed-valued eigenvector. The eigenvector corresponding to the second eigenvector (the so-called "Fiedler vector") is also useful for ordering the vertices of the graph (using the components of this vector) for embedding [29] and partitioning [1] purposes. In general, the d eigenvectors corresponding to the d smallest nonzero eigenvalues may be used to form partitions and embed in $\mathbb{R}^d$. Singular vectors associated with the smallest singular values also play an important role in techniques for embedding graphs in $\mathbb{R}^d$. For example, the null space of the so-called Colin de Verdière matrices [14, 25, 33] are used in convex embeddings of closed manifold genus-0 graphs in $\mathbb{R}^d$, and the null space of stress matrices [16] is used in unique (up to rigid transformations) embeddings of a graph with given edge lengths. When the graphs arise from a 3D mesh structure, as frequently happens in computer graphics applications, or from a k-nearest-neighbor graph, as in feature-learning applications, the graphs tend to be very sparse, and the sparsity should be exploited in the computation of the small singular subspace.

Another area where the null space of rectangular matrices arises is the parameterization of manifold 3D meshes of genus $g > 0$ [24, 26, 39]. In this application, a graph is considered a discrete version of a vector field on a surface, and a discrete version of the one-form is defined for it. Of particular interest are the so-called harmonic one-forms, which satisfy certain balance conditions. The search for harmonic one-forms on a given mesh graph results in the formulation of a set of linear equations for unknowns corresponding to the edges of a graph. Some of the equations are derived for the edges incident on vertices, and some are derived from the edges bounding faces. The size of the matrix is N by E, where E is the number of edges in the graph, and N is typically close to E, but is also influenced by the genus. These matrices are sparse, rectangular, and the nullity (the dimension of the nullspace) is typically $2g$.

Approximate null vectors are also used in at least two areas of numerical linear algebra. One area is condition-number estimation. The spectral condition number of a matrix A is the ratio of its extreme singular values $\sigma_{\max}/\sigma_{\min}$. Estimating the largest singular value is relatively easy; the hard part is estimating $\sigma_{\min}$. This is done almost invariably by trying to find a vector v with a unit norm such that $\|Av\|$ is small. When A is nearly singular, this problem is roughly equivalent to finding an approximate null vector. Note that in condition-number estimation, the estimate can be accurate enough even when v is not a good approximation of the singular vector associated with the smallest singular value. Therefore, the methods for finding such a v are not very similar to the methods that we describe here; condition-number estimators usually favor speed over accuracy; many of them are reliable in practice but may fail on some matrices. For further details, see [30, Chapter 15] and the references therein.

Approximate null vectors are also used in algorithms that compute rank-revealing factorizations and algorithms that solve rank-deficient least-squares problems without a rank-revealing factorization [9, 10, 20, 35]. Here too, a relatively inaccurate approximate null vector is good enough. On the other hand, some of the rank-revealing factorizations require approximate null vector of a sequence of nested upper-triangular matrices (the incrementally-constructed factor of A); There are specialized incremental condition-number estimators for these applications [6, 7]; our approach is not efficient enough for such applications.

We focus on the $m \geq n$ matrices because the problems of computing a basis for the null space when $m > n$ and when $m < n$ are fundamentally different. When $m < n$ the nullity is at least $n-m$. When n is much larger than m, the nullity is high, and the space and time costs of computing an orthogonal basis for the null space are dominated by the usually-dense basis vectors and by the cost of orthogonalizing them. Thus, when $m < n$ the standard approach is to compute a sparse but not necessarily orthogonal basis. This topic has been researched extensively and is outside the scope of this paper [5, 12, 13, 22]. When $m \geq n$ or when m is only slightly larger than n the nullity can be small and the orthogonal-basis algorithms that we discuss are appropriate.

The rest of the paper is organized as follows. The next section introduces inverse iteration. Section 3 introduces normalized inverse iterations for non-normal and for rectangular matrices, and in particular, normalized R iteration. Section 4 presents our main contribution, an LU-based normalized inverse iteration. Section 5 describes the results of numerical experiments, and Section 6 compares our work to previously-published work. Section 7 presents some conclusions and open questions.

2. Inverse Iteration

Given a square matrix A, inverse iteration repeatedly solves the equation $Ax^{(t)} = x^{(t-1)}/\|x^{(t-1)}\|$ for $x^{(t)}$. The starting vector $x^{(0)}$ can be random, although there are alternatives that often work a little better. When A is normal, the iteration converges to an eigenvector associated with the smallest eigenvalue of A (in absolute value). If the equation is solved using a backward stable factorization, such as QR or LU with partial pivoting, the iteration converges even if A is singular. In fact, if A is singular then the iteration converges very quickly, in most cases in one or two iterations.

It is easy to see why the iteration converges even when A is singular. When we compute an LU factorization with partial pivoting, say, of A, what we actually obtain is an exact factorization of a nearby matrix, $A + E = P^T LU$, with $\|E\|/\|A\| = O(\epsilon_{\text{machine}})$. When we iterate, we converge to an approximation of the smallest eigenvector of $A + E$. But since the eigenvalues and eigenvectors of a normal matrix are relatively insensitive to perturbations, the resulting vector is also an approximation to a null vector of A.

If the nullity of A is larger than one, we can start with an n-by-k matrix $Y^{(0)}$, and in each iteration we solve $AY^{(t)} = X^{(t-1)}$ for $Y^{(t)}$ and then orthonormalize the columns of $Y^{(t)}$ to produce $X^{(t)}$. If k is at least as large as the dimension of null(A), then the first $n - \text{rank}(A)$ columns of $X^{(t)}$ converge to an orthonormal basis of null(A). This technique is essentially the inverse version of simultaneous iteration or subspace iteration. The same idea applies to all the inverse iterations that we describe in the rest of the paper.

When A is square but not normal, the method often works, but it may also fail [31]. Fundamentally, the reason for the failure is that the eigenvectors and eigenvalues of a general matrix may be sensitive to small perturbation, so even if the iterations converge to a small eigenvector of $A + E$, it may be far from a null vector of A. Although some techniques for utilizing inverse iteration in the non-normal case do exist [31], the method is inherently unreliable.

When A is not even square, standard inverse iteration does not apply at all.

One issue that is outside the scope of this paper, but should be mentioned, is overflows in inverse iteration. Suppose that we apply inverse iteration using a QR factorization, such that the exact R factor of A is

$$R = \begin{bmatrix} 0 & 1 & & & \\ & 0 & \ddots & & \\ & & \ddots & 1 & \\ & & & 0 & 1 \\ & & & & 0 \end{bmatrix},$$

and that due to rounding errors, the computed factor is

$$\tilde{R} = \begin{bmatrix} \epsilon & 1 & & & \\ & \epsilon & \ddots & & \\ & & \ddots & 1 & \\ & & & \epsilon & 1 \\ & & & & \epsilon \end{bmatrix}.$$

Consider solving $Rx = \vec{1}$ (the right hand-side is the vector of all ones). As ϵ shrinks, the solution converges to $[\epsilon^{-n}\ \epsilon^{-(n-1)}\ \cdots\ \epsilon^{-1}]^T$. If there are no overflows, this is a good approximation of the exact null vector $[1\ 0\ \cdots\ 0]^T$, as expected. But obviously, ϵ^{-n} is extremely likely to overflow. There are techniques to mitigate this danger [31], but they are difficult to apply in the case of subspace iteration. We shall see examples of this behavior in the numerical results below.

3. Normalized Iterations and Normalized Inverse R Iteration

When A is not normal or not even square, a variant of inverse iteration can still be applied reliably. This variant is not new, but not widely appreciated either. We call this variant *normalized* inverse iteration.

Normalized power iteration repeatedly applies A^*A to a starting vector, and normalized inverse iteration repeatedly solves equations of the form $A^*Ax^{(t)} = x^{(t-1)}/\|x^{(t-1)}\|$ for $x^{(t)}$. The Gram matrix A^*A is normal, so inverse iteration works on it reliably. Normalized iterations, both power and inverse, work without ever computing A^*A. The eigenvalues of A^*A are the squares of the singular values of A, so small singular values become a lot smaller: singular values near or below $\|A\|\sqrt{\epsilon_{\text{machine}}}$ become eigenvalues near or below $\|A\|\epsilon_{\text{machine}}$. If we compute A^*A explicitly, rounding errors usually make these small but nonzero singular values indistinguishable from the zero singular values of A, and inverse iteration will always produce linear combinations of the corresponding singular vectors. In other words, inverse iteration on an explicit A^*A may be unstable and may produce vectors that are far from null vectors of A.

But iterating on A^*A implicitly does not suffer from this instability. If A is square, we solve $A^*Ax^{(t)} = x^{(t-1)}/\|x^{(t-1)}\|$ by factoring A using any backward-stable factorization, say LU with partial pivoting, and solving

$$\begin{aligned} A^*w &= x^{(t-1)}/\|x^{(t-1)}\| \\ Ax^{(t)} &= w \,. \end{aligned}$$

As far as we can tell, this idea is due to Stewart [38].

When A is not even square, we can still solve $A^*Ax^{(t)} = x^{(t-1)}/\|x^{(t-1)}\|$ if we compute the reduced QR factorization of A. In this factorization, Q is m-by-n (like A) with orthonormal columns, and R is n-by-n and upper triangular. Here too, we need a backward-stable, but not rank-revealing, factorization. (So when A is sparse, we can use an arbitrary row and column preorderings to minimize fill and work.) Since $A = QR$, we have $A^*A = R^*Q^*QR = R^*R$. To solve $A^*Ax^{(t)} = R^*Rx^{(t)} = x^{(t-1)}/\|x^{(t-1)}\|$, we perform two triangular solves,

$$\begin{aligned} R^*w &= x^{(t-1)}/\|x^{(t-1)}\| \\ Rx^{(t)} &= w \,. \end{aligned}$$

Björck [8, Page 109] credits Chan [10] with this technique, although Chan's paper is not explicit about how to carry out the inverse iteration. We refer to this technique as *normalized inverse R iteration.* The simultaneous/subspace version of this technique can be used to compute multiple singular vectors associated with the smallest singular values of A.

Both the implicit normalization idea and the use of the R factor are not new, but is also not widely appreciated. Virtually all the research on inverse iteration, surveyed by Ipsen [31], ignores normalization and focuses instead on less reliable and more complex methods to enhance inverse iteration for the non-normal case. In that literature, there is essentially no discussion of rectangular matrices. So although normalization and the use of the R factor are mentioned in the literature, they are not widely known. For example, LAPACK [3] uses unnormalized inverse iteration to compute eigenvalues of tridiagonal matrices [31].

4. LU-Based Normalized Inverse Iterations

This section presents the main contribution of the paper, normalized inverse iterations with the triangular factors computed by LU with partial pivoting. The algorithms start with a factorization

$$PA = LU = \begin{bmatrix} L' \\ L'' \end{bmatrix} U \,, \tag{4.1}$$

where P is an m-by-m permutation matrix, L is an m-by-n upper trapezoidal matrix, and U is an n-by-n upper triangular matrix. Thanks to partial pivoting, L has ones on the diagonal and the magnitude of all of its elements is bounded by 1. We partition L into pivot and non-pivot rows: L' is the square n-by-n triangular part of L, containing the pivot rows, and L'' is the subdiagonal block containing the remaining $m - n$ rows.

In exact arithmetic, A, U, and $L'U$ all have exactly the same null space. Therefore, we can try to compute the null space of A by performing normalized inverse iteration on U or on $L'U$, both of which are square. The matrix U is upper triangular, and once we compute the factorization (4.1), we have a triangular factorization of $L'U$. This allows us to perform normalized inverse iteration with either $L'U$ or with U without any additional pre-processing. The following trivial lemma proves that A, U, and $L'U$ all have the same null space. Once we prove it, we analyze the effect of rounding errors on this process.

Lemma 4.1. *Let*

$$PA = LU = \begin{bmatrix} L' \\ L'' \end{bmatrix} U$$

be an exact LU factorization of A such that L' has a nonzero diagonal (this covers the case of partial pivoting). Then null(A) = null($L'U$) = null(U).

Proof. The row permutation P is irrelevant for null vectors, so without loss of generality we assume that $P = I$. Because L' is nonsingular, $\text{null}(L'U) = \text{null}(U)$. Therefore, all we need to show is that $\text{null}(A) = \text{null}(L'U)$. If $Ux = 0$, then we also have $L''Ux = 0$, so $Ax = LUx = 0$. This shows that $\text{null}(A) \subseteq \text{null}(U) = \text{null}(L'U)$. On the other hand, if $Ax = LUx = 0$ then in particular $L'Ux = 0$. This shows that $\text{null}(A) \supseteq \text{null}(L'U) = \text{null}(U)$, which concludes the proof. □

Next, we analyze the effects of rounding errors on approximate null vectors of $L'U$.

Lemma 4.2. *Let*

$$A + E = PLU = P \begin{bmatrix} L' \\ L'' \end{bmatrix} U$$

be an LU factorization with partial pivoting of A such that

$$\|E\|_2 \leq \epsilon$$

for some small ϵ. Let x be an approximate null vector of $L'U$, such that

$$\|x\|_2 = 1 \text{ and } \|L'Ux\|_\infty \leq \delta$$

for some small δ. Then x is also an approximate null vector of A,

$$\|Ax\|_\infty \leq \epsilon + 2^n \delta \ .$$

(The choice of norms is fairly arbitrary up to multiplicative factors of order $\sqrt{m}$ in the bounds.)

Proof. We again assume without loss of generality that $P = I$.

We begin by bounding the size of elements of $y = Ux$. We have

$$|L'_{11} y_1| = |1 \cdot y_1| \leq \delta \ .$$

Next we have

$$|L'_{21} y_1 + L'_{22} y_2| \leq \delta \ .$$

Because the magnitude of elements of L is bounded by 1, we have $|L'_{21} y_1| \leq |y_1| \leq \delta$. Therefore,

$$|y_2| = |L'_{22} y_2| \leq \delta + |L'_{21} y_1| \leq \delta + \delta = 2\delta \ .$$

Similarly, we have

$$\begin{aligned} |y_3| &= |L'_{33} y_3| \\ &\leq \delta + |L'_{31} y_1| + |L'_{32} y_2| \\ &\leq \delta + |y_1| + |y_2| \\ &\leq \delta + \delta + 2\delta \\ &= 4\delta \ . \end{aligned}$$

In general, for $k \le n$ we have $|y_k| \le 2^{k-1}\delta$.

From

$$(LUx)_k = (Ly)_k = \sum_{j=1}^{\min(k,n)} L_{kj} y_j$$

and the triangle inequality we obtain

$$|(LUx)_k| \le \sum_{j=1}^{\min(k,n)} |L_{kj} y_j| \le \sum_{j=1}^{\min(k,n)} |y_j| \le 2^{\min(k,n)}\delta \; .$$

Therefore,

$$|(Ax)_k| = |(LUx)_k + (Ex)_k| \le |(LUx)_k| + |(Ex)_k| \le 2^{\min(k,n)}\delta + \epsilon \; .$$

□

The exponential factor in the bound is, of course, bad. It means that the lemma does not really guarantee that an approximate null vector of $L'U$ is also an approximate null vector of A. However, achieving the bound depends on a precise alignment of signs, which is unlikely to occur in practice. (Similarly, the element growth in Gaussian elimination with partial pivoting may be exponential, but it rarely high in practice.)

On the positive side, a null vector of A is always an approximate null vector of $L'U$. Suppose that $Ax = 0$. Since $Ex = (A+E)x = P^T LUx$, all the elements of LUx must be small, and in particular, the elements of $L'Ux$. But this is not enough. Suppose that A has rank $n-1$ but the numerical rank of $L'U$ is $n-2$. When we apply subspace normalized inverse iteration to $L'U$, we will get a basis of the two-dimensional numerical null space of $L'U$. In general, neither of these vectors is an approximate null vector of A. Therefore, this output does not allow us to determine the numerical rank of A at all.

We summarize this discussion. Suppose that we compute approximate null vectors of $L'U$ using an iteration of the form

$$\begin{aligned} U^* y &= x^{(t-1)}/\|x^{(t-1)}\| \\ L_1^* w &= y \\ L_1 z &= w \\ U x^{(t)} &= z \; . \end{aligned}$$

(Or a subspace version of it, to compute multiple independent null vectors.) Then the vectors that are computed are not necessarily null vectors of A, but they span a subspace that approximately contains the null space of A.

Normalized inverse iteration with U has slightly different behavior. It always produces good approximations to null vectors of A, but it may fail to detect some of the null vectors of A (that is, it may return a basis for a proper subspace of the null space, a subspace with a dimension strictly smaller than the nullity of A).

Lemma 4.3. *Let*

$$A + E = PLU$$

be an LU factorization with partial pivoting of A such that

$$\|E\|_2 \le \epsilon$$

for some small ϵ. Let x be an approximate null vector of U, such that

$$\|x\|_2 = 1 \text{ and } \|Ux\|_\infty \le \delta$$

for some small δ. Then x is also an approximate null vector of A,

$$\|Ax\|_\infty \le \epsilon + n\delta \ .$$

Proof. We again assume without loss of generality that $P = I$. Denoting again $y = Ux$, we have $|y_k| \le \delta$ for all k. Because the magnitude of elements of L is at most 1,

$$|(LUx)_k| \le \sum_{j=1}^{\min(k,n)} |L_{kj} y_j| \le \sum_{j=1}^{\min(k,n)} |y_j| \le \min(k,n)\delta \ .$$

Therefore,

$$|(Ax)_k| = |(LUx)_k + (Ex)_k| \le |(LUx)_k| + |(Ex)_k| \le n\delta + \epsilon \ .$$

□

The $n\delta + \epsilon$ bound is acceptable and much better than the exponential bound we obtained for $L'U$. Here too, achieving the worst-case bound depends on a precise alignment of signs, and we can expect a much smaller residual in many cases.

Can inverse iteration on U fail to find a null vector of a singular A? This may happen, but it is extremely unlikely. Suppose that for some unit vector x, the product Ax has a tiny norm, so the product $P(A + E)x = LUx$ is also small, but the norm of $y = Ux$ is large. That means that L is ill conditioned, because Ly is small for some not-so-small y, and the norm of L is at least 1. Experience with LU with partial pivoting has shown that L is usually well conditioned, and that any ill-conditioning in A tends to be reflected in U, not in L. There are pathological cases where L is ill conditioned, but they are rare in practice (we show one such case later in the paper).

Better yet, we can easily check whether L is ill conditioned. First, note that if L is ill conditioned, then L' must be ill conditioned. Now apply normalized inverse iteration to L'. If the iteration finds an approximate null vector of L', then L' is ill conditioned. Otherwise, it almost certainly is not.

We now summarize the entire algorithm.

(1) Compute an LU factorization with partial pivoting $PA = LU = \begin{bmatrix} L' \\ L'' \end{bmatrix} U$.
(2) Perform normalized inverse iteration with U. The resulting vectors are approximate null vectors of A, but there may be more.
(3) Perform normalized inverse iteration with L' to determine whether it is ill conditioned. If it is, go to Step 5, otherwise, continue.
(4) L' is not ill conditioned: the vectors that we computed in Step 2 should be a basis for the null space of A. Report the numerical rank of A and the basis for the null space, and return.
(5) L' is ill conditioned, so inverse iteration with U may have failed to find some null vectors of A. Run normalized inverse iteration on $L'U$.
(6) If the iteration in Step 5 produced approximate null vectors of $L'U$, their number is an upper bound on the numerical rank deficiency of A (this number is possibly larger than the number of vectors found in Step 2).

(7) Determine which of the vectors produced in Step 5 is also an approximate null vector of A and linearly independent of the vectors produced in Step 2. Return these vectors, along with the vectors produced in Step 2. Also report the upper bound computed in Step 6.

As pointed out above, because L is usually well conditioned, we expect that the algorithm will usually perform steps 1–4 and stop there. If the algorithm does continue to steps 5–7, then the approximate null vectors that it returns may or may not constitute a basis for the null space of A. More specifically, if their number is smaller than the upper bound on the nullity, the vectors may span only a proper subspace of null(A), or they constitute a basis but the upper bound is loose.

5. Numerical Experiments

In this section we provide a few illustrative examples to demonstrate the behavior of the LU and QR-based algorithms, including pathological behaviors.

We carried out the experiments using Matlab version 7.0 on a 3 GHz Pentium 4 computer with 1 GB of main memory running Linux. The LU factorization was performed using the function call `[L,U,P,Q]=lu(A,1.0)`, which calls umfpack version 4.3 (`P` and `Q` are row and column permutations). This syntax enforces partial pivoting and allows the sparse factorization code to reorder rows and columns for sparsity (under the partial pivoting constraint). We also ran a normalized inverse R iteration, to compare the runtimes. We computed the R factor using the function call `R=qr(A(:,colamd(A)),0)`, which avoids the expensive computation of an explicit Q. The reordering of the columns tends to reduce fill and work, and is generally similar to the column ordering that umfpack uses.

The codes implement the algorithms from Sections 3 and 4. They apply the iterations first to 1 vector, then 2, then 4, and so on, until the dimension of the computed null space stops growing. The codes always run 3 iterations of the appropriate strategy, starting from a matrix consisting of uniformly-distributed random numbers between 0 and 1.[2]

5.1. **Accuracy.**

We created random matrices with singular values $1, \ldots, 1, \sigma_2, 0$ for $\sigma_2 = 10^{-16}, 10^{-15}, \ldots, 10^{0}$. The matrices are all 200-by-100, and they were computed by generating random orthonormal singular vectors and multiplying the singular vectors and singular values appropriately. We generated 100 random matrices for each σ_2. For each matrix, we used our algorithm to compute its null vector. We also used Matlab's singular value decomposition (`svd`) on both A and $A^T A$. On all of these matrices, our algorithm computed the null space of A by iterating on U; L was never ill conditioned. The results of the experiment, shown in Figure 5.1, show that our algorithm is less accurate than a full svd computation, but not significantly so. In particular, the results show that the qualitative behavior of our algorithm is similar to that of a full svd: the accuracy degrades smoothly as σ_2 approaches $\epsilon_{\text{machine}}\|A\|$.

We also ran experiments on matrices whose U factors have tiny diagonal values near the upper left corner. We did this by generating two independent columns, then a column that depends on the first two, and then another column that almost depends on the first two, but not exactly. We then completed the matrices with 96 additional linearly independent columns. This yielded matrices with norm around

[2]The main code, `nulls.m`, is publicly availalble at `http://www.tau.ac.il/~stoledo/research.html`.

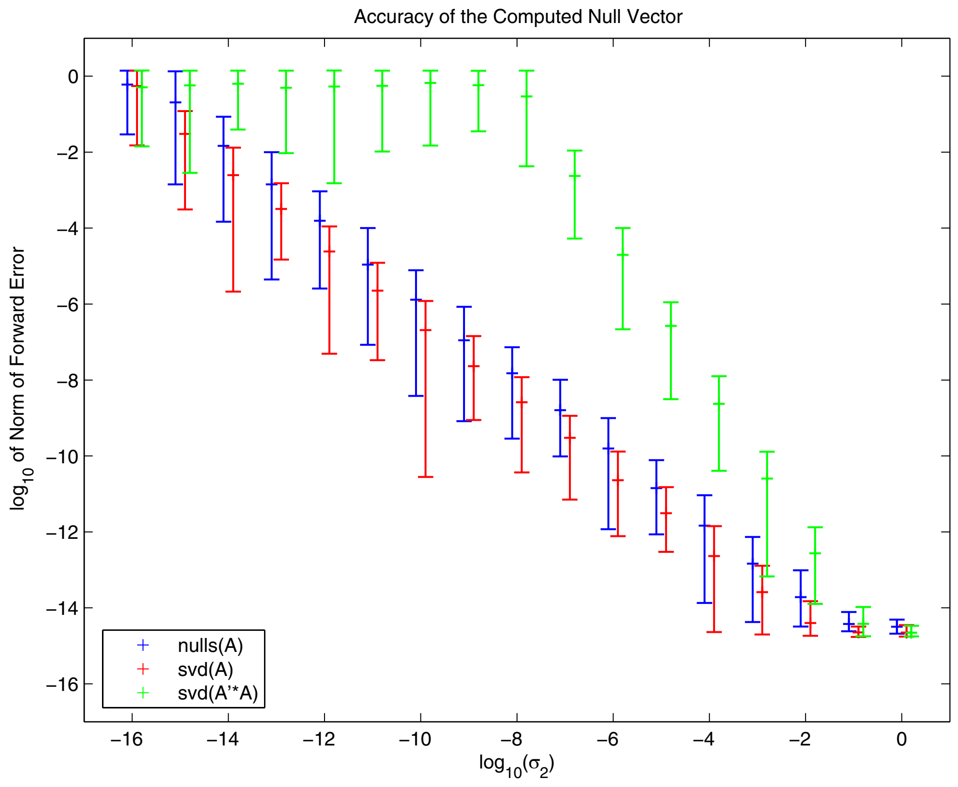

FIGURE 5.1. The results of the accuracy experiments. The bars show the range of accuracies for each σ_2 in 100 experiments; the mark along each range is the mean log accuracy. For each value of σ_2, the graph shows the accuracy of our algorithm (denoted `nulls`), of MATLAB's SVD implementation, and of the SVD applied to A^TA. The three bars for each σ_2 are slightly offset so that they don't overlap, but they all represent experiments with exactly the same σ_2.

1, one zero singular value, and one singular value near 10^{-8}. On the diagonal of U we have U_{33} close to $\epsilon_{\text{machine}}$ and U_{44} is small. We have also conducted experiments in which the dependent and almost-dependent columns, to swap the small and numerically-zero elements on the diagonal of U. The accuracy in these experiments was similar to the accuracy achieved in the previous experiments. From this experiment it appears that the position of small elements on the diagonal of U does not have a significant influence on the accuracy of the algorithm.

5.2. **Large Matrices.** We conduced experiments on a few of large sparse matrices from Davis's sparse matrix collection[3]. More precisely, we took matrices from this collection and modified them slightly to make them rectangular and singular. This experiment serves three purposes. First, it shows that the algorithm runs reasonably quickly even on large matrices. Second, it shows that our LU-based

[3]`http://www.cise.ufl.edu/research/sparse/matrices/`

TABLE 1. Our test matrices and the sizes of the computed null spaces. The columns denoted d displays the dimensions of the computed null spaces, and the $\overset{\text{NaN}}{\infty}$ columns shows whether any overflows or NaN's were detected during the iterations. The QR factorization ran out of memory on three matrices.

		LU		QR	
Base Matrix	n	d	$\overset{\text{NaN}}{\infty}$	d	$\overset{\text{NaN}}{\infty}$
fpga_trans_02	1220	2		2	
shyy41	4720	4		0	Y
utm5940	5940	2		2	
poisson3Da	13514	2		2	
mult_dcop_01	25187	1		—	
mult_dcop_02	25187	2		—	
mult_dcop_03	25187	0	Y	—	
wang4	26068	2		2	
onetone1	36057	2		2	
twotone	120758	2		2	

algorithm is much faster than normalized inverse R iteration. Third, it shows that the algorithm can fail; the failures are not specific to the LU-based algorithm but to inverse iteration in general. We believe that the failures are mostly due to scaling and overflow problems, similar to the ones discussed in Section 2. In principle, these problems can be addressed by exploiting the capabilities of floating-point hardware better, but we have not implemented such measures.

We constructed the matrices as follows. All the matrices were initially square. From each matrix we dropped the first and last row, and then duplicated rows 11 to 20 at the bottom of the matrix. This created $(n+8)$-by-n rectangular matrices with rank at most $n-2$.

The results of the experiments are summarized in Tables 1 and 2. Table 1 lists the matrices and the dimensions of the computed null spaces. Since the matrices were constructed to have null spaces of dimension at least 2, any dimension less than 2 indicates failure. Dimensions larger than 2 reflect matrices that were originally singular. One group of matrices, mult_dcop, caused difficulties to the LU-based algorithm, resulting in two failures. One of the failures led to overflows, but the other was silent. (The QR factorization of these matrices ran out of memory). Two of these matrices, mult_dcop_02 and 03, have highly skewed row scaling, which may contribute to the difficulty: the ratio between the extreme ∞-norms of rows is 10^{12} for mult_dcop_03, and even large for 02. Another matrix, shyy41, which was originally singular, caused similar difficulties to the QR-based algorithm. This shows that this class of numerical difficulties is not associated with our new LU-based algorithm, but with inverse iteration in general.

Table 2 shows the performance of the two algorithms. On all the matrices, the LU-based algorithms ran in less than 30 seconds. On several large matrices it ran in less than 10 seconds. We argue that these are acceptable running times. The table also shows that in all the experiments the QR-based algorithm was slower, in most cases substantially slower. This is probably due both to the fact that MATLAB 7 uses a state-of-the-art sparse LU factorization but a much older sparse QR, and

TABLE 2. Runtimes and the size of the factors. The columns denoted T displays the total running times, and the columns denoted T_f show the running time of the factorization alone. The columns η_L, η_U, and η_R show the number of nonzeros in the computed factors. The QR factorization ran out of memory on three matrices.

		LU				QR		
Base Matrix	n	T	T_f	η_L	η_U	T	T_f	η_R
FPGA_TRANS_02	1220	0.09	0.03	5.7e3	6.1e3	0.14	0.05	2.8e4
SHYY41	4720	0.76	0.13	5.9e4	7.1e4	5.24	0.34	1.8e5
UTM5940	5940	1.87	0.50	3.8e5	4.9e5	5.04	3.27	8.8e5
POISSON3DA	13514	23.84	7.27	5.9e6	6.0e6	290.67	255.85	1.7e7
MULT_DCOP_01	25187	2.18	1.06	1.4e5	3.5e5	—	—	—
MULT_DCOP_02	25187	2.82	0.96	1.0e5	3.2e5	—	—	—
MULT_DCOP_03	25187	3.56	1.26	1.2e5	3.4e5	—	—	—
WANG4	26068	47.94	15.48	1.1e7	1.1e7	478.16	431.35	2.3e7
ONETONE1	36057	14.94	5.03	1.8e6	2.5e6	45.62	36.42	4.3e6
TWOTONE	120758	29.77	12.75	3.2e6	4.8e6	132.66	98.60	1.6e7

to the intrinsic differences in the costs of sparse LU and QR factorizations. A comparison of the fill in the LU factors and the fill in the QR factor shows that the R factor is denser, but not significantly more than L and U combined.

5.3. **Extreme Examples.**

We now describe matrices that cause extreme behaviors in inverse-iteration algorithms. Experiments with these matrices constitute a partial coverage test of our implementation, because they exercise parts of the code that are rarely reached on real-world matrices.

We start with a particularly pathological matrix, suggested to us as an example by G. W. Stewart. This matrix has the form

$$A_S = \begin{bmatrix} 1 & & & & \\ -1 & 1 & & & \\ \vdots & & \ddots & & \\ -1 & & & 1 & \\ -1 & -1 & \cdots & -1 & 1 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix} .$$

The matrix A_s is $(n+1)$-by-n, has 1's on the diagonal, -1 below the diagonal in rows 1 through n, and all the entries in row $n+1$ are 0.5. The LU factorization with partial pivoting of A_S is $A_S = A_S I$, because A_S is already lower trapezoidal and its subdiagonal entries are bounded by 1 in absolute value. This matrix is well conditioned, so normalized inverse iterations should not find any approximate null vectors. Indeed, if we perform normalized inverse iteration with the upper triangular factor, we find no approximate null vectors. However, L', consisting of the first n rows of A_S, is very ill conditioned, since $L'(1\ 2\ 4\ \ldots\ 2^n)^T = (1\ 1\ 1\ \ldots\ 1)^T$. This implies that the condition number of L' is exponential in n. When we run normalized inverse iteration on either L' or $L'U$, we find an approximation of the small singular vector of L'. In this particular case, the large condition number of L' will cause our algorithm to iterate on $L'I$. This will return a single candidate vector, and an upper bound of 1 on the rank deficiency. In this particular case, since the upper bound is 1, there is only one candidate vector x that is easy to rule out by observing that $\|A_S x\|$ is large. But this example shows that the exponential bound shown in Lemma 4.2 can be attained, and it shows that L' can be ill conditioned. The exponential bound implies that the dimension of the null space computed by iterating on $L'U$ is only an upper bound, and the ill conditioning of L' shows that the dimension of the null space computed by iterating on U is only a lower bound. Put together, this means that the method may fail to reliably estimate the rank deficiency (but it will report this failure explicitly, because it will detect the ill conditioning of L').

We also ran the algorithms on a block matrix of the form

$$\begin{bmatrix} A_S & 0 \\ 0 & A_R \end{bmatrix},$$

Where A_S is the matrix describe above, and A_R is a random matrix with given singular values: all 1 except for four, which are three 0's and one 10^{-8}. On this matrix the QR-based algorithm correctly computes the rank deficiency, 3, and null space correctly, which is essentially the null space of A_R. The LU-based algorithm performs all the steps in the algorithm (that is, it does not stop in step 4 because it correctly detects that L' is ill conditioned). It finds three null vectors using inverse iteration on U, but since $L'U$ has four approximate null vectors, the algorithm returns the three null vectors but reports that the rank deficiency might be 4. In this particular case it is possible to determine the null space correctly, of course, but the example shows that the algorithm may need to resort to reporting a too-lax upper bound on the deficiency.

The next example shows that normalization may be necessary. The following class of square matrices,

$$A_I = \begin{bmatrix} 1 & \eta & & \\ & 1 & \ddots & \\ & & \ddots & \eta \\ & & & 1 \end{bmatrix},$$

where $\eta > 1$ is a parameter, were used by Ipsen [31] to show that without normalization, inverse iteration may fail. Their inverses are

$$A_I^{-1} = \begin{bmatrix} 1 & \eta & \eta^2 & \cdots & \eta^{n-1} \\ & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & \eta^2 \\ & & & \ddots & \eta \\ & & & & 1 \end{bmatrix}.$$

The norm of the matrices is $O(1+\eta)$ but the norm of the inverses is $O(1+\eta^{n-1})$. Therefore, the matrices are highly ill conditioned, so inverse iteration methods should find a vector x such that $A_I x$ has a small norm. However, without normalization, inverse iteration fails. With normalization, inverse iteration works. (Since A_I is upper triangular, iterating with A_I or with its R factor or with its U factor are exactly equivalent methods.)

5.4. **Embeddings Graphs on Surfaces.** We have also performed experiments on the following class of matrices. We describe the application where they arise, but we do not provide detailed results, since we detected surprising or interesting behaviors on these matrices.

An instance of a non-normal matrix whose null space is of interest is derived from a graph $G = (V, E, F)$ which has been embedded on a closed manifold surface of genus $g > 0$ in $\mathbb{R}^3$ (e.g., a torus). The graph G has $|V|$ vertices, $|E|$ edges and $|F|$ faces. A value x_h may be attached to each half-edge h of G, such that $x_h = -x_{t(h)}$, where $t(h)$ is the (opposite) twin half-edge of h. Given an orientation for each edge, the vector x of the values corresponding to the half-edges in this orientation is known as a *discrete one-form*, or just one-form for short, of G [26]. A *harmonic* one-form is one which satisfies some balance conditions, derived from each vertex and face of G. For a set of symmetric weights $w_h = w_{t(h)}$, each vertex v induces the following *co-closedness* linear equation on x,

$$\sum_{h\in\delta v} w_h x_h = 0 \ ,$$

where δv is the set of half-edges emanating from v. Each face f induces the following closedness linear equation,

$$\sum_{e\in\partial f} x_h = 0 \ ,$$

where ∂f is the set of half-edges bounding f.

In total, there are $|V| + |F|$ equations in $|E|$ unknowns, whose rank turns out to be $|V| + |F| - 2$. The Euler-Poincare formula for manifold graphs asserts that $|V| + |F| - |E| = 2 - 2g$, so this rank is $|E| - 2g$. Thus, solving for a basis for the subspace of harmonic one-forms involves computation of the $2g$-dimensional nullspace of a non-normal matrix of size $|E| + 2 - 2g$ by $|E|$.

By integrating harmonic one-forms, it is possible to parameterize manifold mesh data very efficiently. This has many applications in computer graphics and geometry processing [24, 26, 39].

6. Related Work

Unnormalized inverse iteration for square matrices is a well researched area. The method was invented by Wielandt in 1944 and was studied by Wilkinson, who published his findings in several papers and books over a period of almost 30 years. For a comprehensive survey of these results, along with many newer results, see Ipsen's survey [31].

Normalized inverse iteration for square matrices seems to have been first proposed by Stewart [38]. Normalized inverse R iteration is due to Chan [10] (see also [8, Page 109]).

Normalized inverse iteration with U or with $L'U$ is, to the best of our knowledge, new. It is remotely related to an idea by Saunders [36] to use U to precondition an iterative least-squares solver. There are additional least-squares preconditioners that are based on an LU factorization, but they all use the L'' block as well, so they are not really related to our proposed method (see [8, Section 7.5.3] and the references cited there). Our algorithm is also related to the Peters-Wilkinson family of methods for solving least squares problems using an LU factorization [34], in that both rely on the fact that L is usually well conditioned.

For square matrices, Schwetlick and Schnabel [37] proposed a bordering iteration as an alternative to inverse iteration. The advantage of their method is that the linear systems that their method solves in each iteration is nonsingular, so they can potentially be solved by an iterative linear solver, such as GMRES. However, the method is limited to square matrices that are numerically rank deficient by only one.

The standard way to compute a basis for the null space of a rectangular matrix is using a rank-revealing factorization, such as a rank-revealing LU or QR factorizations. For dense matrices, sophisticated rank-revealing factorizations are only slightly more expensive than backward-stable but non-rank-revealing ones. However, for sparse matrices, such factorizations can be significantly more expensive to compute than LU with partial pivoting or QR, because rank-revealing factorizations require column pivoting. In sparse QR and LU with partial pivoting, the column ordering is chosen so as to minimize fill and computation, so pivoting to reveal the rank typically leads to more fill and more work. Furthermore, sparse rank-revealing factorizations have not been implemented much, and those that have are not widely available. (The state-of-the-art in this area is an algorithm by Lewis and Pierce [35], but the code is not publicly available; an earlier method proposed by Foster [20] uses similar techniques to detect dependent columns and to re-triangularize R; see also [4].) In contrast, several recent and high-quality sparse LU with partial pivoting codes, which lie at the heart of our method, are publicly available [2, 17, 18, 19, 27, 28]. Some of these can exploit parallel computers and/or clusters. Even general-purpose interactive numerical engines, such as MATLAB, now contain excellent sparse LU codes (MATLAB 7 uses UMFPACK 4.3 [17]).

One method that might seem relevant but is not is inverse iteration on an augmented matrix

$$H = \begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix} .$$

The augmented matrix is normal, its eigenvalues are the singular values of A with both signs and additional $m-n$ zero eigenvalues. The eigenvectors of H associated with nonzero eigenvalues are concatenations of left and right singular vectors of A. The difficulty is that the dimension of null(H) is larger by $m-n$ than the dimension of null(A). There is no straightforward way to compute a basis from null(A) from a basis for null(H). Also, if m is significantly larger than n, then H has a high-dimensional null space that is expensive to compute. This method is appropriate for computing bases for singular subspaces of A associated with a singular value $\sigma \gg 0$, by inverse iteration on $H - \sigma I$ (for σ close to zero, the null space of H causes inaccuracies in the computed singular vectors). MATLAB, for example, uses the augmentation idea in its sparse SVD routine `svds` (which fails when applied to the computation of the null space).

Finally, we mention that our ideas also apply to iterative Arnoldi/Lanczos-type algorithms. When these algorithms are used to find the smallest eigenpairs, they usually iterate on a representation of the inverse. This is the case, for example, in ARPACK [32], an Arnoldi-based package (ARPACK is the code that MATLAB's `eigs` calls). Therefore, as in other forms of inverse iteration, the cost of these algorithms is likely to be dominated by the cost of factoring A. MATLAB's `eigs`, for example, calls exactly the same sparse LU factorization routine that our code calls. Also, if the inversion scheme is unnormalized, these methods can suffer from the same problems that simple inverse iteration suffers from. This implies that applying Lanczos to the implicitly normalized inverse is likely to be more reliable than applying Arnoldi to an unnormalized inverse (this is possible when the application can use singular triplets rather than eigenpairs, which is the case when computing the null space).

We summarize the discussion of Arnoldi/Lanczos algorithms as follows. First, when applied to an unnormalized inverse, they can be unreliable. We recommend that a normalized inverse be used when using these algorithms to compute the null space. Second, our analysis in Section 4 is also applicable when the null spaces of U, L', and possibly $L'U$ are computed using a normalized Lanczos procedure rather than simultaneous inverse iteration.

7. Conclusions

We have shown how to utilize an LU factorization with partial pivoting of a non-normal and possibly rectangular matrix to compute its null space. The algorithm is usually reliable and accurate. Furthermore, if the case of failure is ill conditioning in L', then it reports that it failed (rather than fail silently) and it provides a reliable upper bound on the nullity, possibly along with a basis for a subspace of the null space.

Our new algorithm can also fail due to overflows or scaling problems, but this is a property of inverse iterations in general, not of this particular variant. These problems can be addressed by exploiting the capabilities of floating-point hardware, but our implementation does not take these measures. This makes the algorithm somewhat less reliable than rank-revealing factorizations, but it is also much cheaper.

Because our algorithm uses an LU factorization, it can be easily applied to large sparse matrices, using one of several available factorization codes. Relying on an LU rather than a QR factorization reduces the total cost, especially in the sparse case, where a QR factorization can be substantially more expensive to compute.

Our method can use a Lanzcos iteration, rather than simple inverse iteration (to compute the null spaces of U, L', and possibly $L'U$). The issue that our algorithm addresses is not the iteration itself, but the representation of the inverse, and the representations that we proposed are also applicable to Lanczos iterations.

Acknowledgements. Thanks to Pete Stewart and to Chen Greif for extensive comments on early drafts of this paper.

Sivan Toledo was partially 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] C. J. Alpert and S.-Z. Yao, *Spectral partitioning: the more eigenvectors, the better*, in DAC '95: Proceedings of the 32nd ACM/IEEE conference on Design automation, ACM Press, 1995, pp. 195–200.

[2] P. R. Amestoy and C. Puglisi, *An unsymmetrized multifrontal LU factorization*, SIAM Journal on Matrix Analysis and Applications, 24 (2002), pp. 553–569.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, SIAM, 2nd ed., 1994. Also available online from `http://www.netlib.org`.

[4] J. Barlow and U. Vemulapati, *Rank detection methods for sparse matrices*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 1279–1297.

[5] M. W. Berry, M. T. Heath, I. Kaneko, M. Lawo, R. J. Plemmons, and R. C. Ward, *An algorithm to compute a sparse basis of the null space*, Numerische Mathematik, 47 (1985), pp. 483–504.

[6] C. H. Bischof, *Incremental condition estimation*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 312–322.

[7] C. H. Bischof, J. G. Lewis, and D. J. Pierce, *Incremental condition estimation for sparse matrices*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 644–659.

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

[9] T. F. Chan, *Deflated decomposition of solutions of nearly singular systems*, SIAM Journal on Numerical Analysis, 21 (1984), pp. 738–754.

[10] ———, *Rank revealing QR factorizations*, Linear Algebra and its Applications, 88/89 (1987), pp. 67–82.

[11] F. R. K. Chung, *Spectral Graph Theory*, CBMS Regional Conference Series on Mathematics, AMS, 1997.

[12] T. F. Coleman and A. Pothen, *The null space problem i. complexity*, SIAM Journal Algebraic Discrete Methods, 7 (1986), pp. 527–537.

[13] ———, *The null space problem ii. algorithms*, SIAM Journal Algebraic Discrete Methods, 8 (1987), pp. 544–563.

[14] Y. Colin de Verdière, *Sur un nouvel invariant des graphes et un critère de planarité*, Journal of Combinatorial Theory Series B, 50 (1990), pp. 11–21. An English translation is available as [15].

[15] Y. Colin de Verdière, *On a new graph invariant and a criterion for planarity*, in Graph Structure Theory, N. Robertson and P. D. Seymour, eds., vol. 147 of Contemporary Mathematics, AMS, 1993, pp. 137–148. English translation of [14].

[16] R. Connelly, *Ridigity and energy*, Inventiones Mathematicae, 66 (1982), pp. 11–33.

[17] T. A. Davis, *A column pre-ordering strategy for the unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software, 30 (2004), pp. 165–195.

[18] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 720–755.

[19] J. W. Demmel, J. R. Gilbert, and X. S. Li, *An asynchronous parallel supernodal algorithm for sparse Gaussian elimination*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 915–952.

[20] L. V. Foster, *Rank and null space calculations using matrix decompositions without column interchanges*, Linear Algebra and its Applications, 74 (1986), pp. 47–71.

[21] A. George and E. Ng, *On the complexity of sparse QR and LU factorization on finite-element matrices*, SIAM Journal on Scientific and Statistical Computation, 9 (1988), pp. 849–861.

[22] J. Gilbert and M. Heath, *Computing a sparse basis for the null space*, SIAM Journal Algebraic Discrete Methods, 8 (1987), pp. 446–459.

[23] J. R. Gilbert and E. Ng, *Predicting structure in nonsymmetric sparse matrix factorizations*, in Graph Theory and Sparse Matrix Computation, A. George, J. R. Gilbert, and J. W. H. Liu, eds., Springer-Verlag, 1993.

[24] S. J. Gortler, C. Gotsman, and D. Thurston, *One-forms on meshes and applications to 3D mesh parameterization*, Tech. Report CS TR-12-04, Harvard University, June 2004. To appear in Computer Aided Geometric Design, 2005.

[25] C. Gotsman, X. Gu, and A. Sheffer, *Fundamentals of spherical parameterization for 3d meshes*, ACM Transactions on Graphphics, 22 (2003), pp. 358–363.

[26] X. Gu and S.-T. Yau, *Computing conformal structures of surfaces*, Communications in Information and Systems, 2 (2002), pp. 121–146.

[27] A. Gupta, *Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices*, SIAM Journal on Matrix Analysis and Applications, 24 (2002), pp. 529–552.

[28] ———, *Recent advances in direct methods for solving unsymmetric sparse systems of linear equations*, ACM Transactions on Mathematical Software, 28 (2002), pp. 301–324.

[29] K. M. Hall, *An r-dimensional quadratic placement algorithm*, Management Science, 17 (1970), pp. 219–229.

[30] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, 2nd ed., 2002.

[31] I. C. F. Ipsen, *Computing an eigenvector with inverse iteration*, SIAM Review, 39 (1997), pp. 254–291.

[32] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK User's Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, 1998.

[33] L. Lovász and A. Schrijver, *On the null space of a Colin de Verdière matrix*, Annales de l'institut Fourier, 49 (1999), pp. 1017–1026.

[34] G. Peters and J. H. Wilkinson, *The least-squares problem and pseudo-inverses*, Computer Journal, 13 (1970), pp. 309–316.

[35] D. J. Pierce and J. G. Lewis, *Sparse multifrontal rank revealing QR factorization*, SIAM Journal on Matrix Analysis and Applications, 18 (1997), pp. 159–180.

[36] M. A. Saunders, *Sparse least squares by conjugate gradients: A comparison of preconditioning methods*, in Proceedings of Computer Science and Statistics: 12th Annual Symposium on the Interface, J. F. Gentleman, ed., University of Waterloo, Waterloo, Ontario, Canada, May 1979, pp. 15–20. Cited by [8].

[37] H. Schwetlick and U. Schnabel, *Iterative computation of the smallest singular value and the corresponding singular vectors of a matrix*, Linear Algebra and its Applications, 371 (2003), pp. 1–30.

[38] G. Stewart, *Rank degeneracy*, SIAM Journal on Scientific and Statistical Computing, 5 (1981).

[39] G. Tewari, C. Gotsman, and S. Gortler, *Meshing point clouds with genus 1 using discrete one-forms*. Preprint, 2005.