Algorithm xxx: Reliable Calculation of Numerical Rank, Null Space Bases, Pseudoinverse Solutions, and Basic Solutions using SuiteSparseQR

LESLIE V. FOSTER
San Jose State University
and
TIMOTHY A. DAVIS
University of Florida

The SPQR_RANK package contains routines that calculate the numerical rank of large, sparse, numerically rank-deficient matrices. The routines can also calculate orthonormal bases for numerical null spaces, approximate pseudoinverse solutions to least squares problems involving rank-deficient matrices, and basic solutions to these problems. The algorithms are based on SPQR from SuiteSparseQR (ACM Transactions on Mathematical Software 38, Article 8, 2011). SPQR is a high-performance routine for forming QR factorizations of large, sparse matrices. It returns an estimate for the numerical rank that is usually, but not always, correct. The new routines improve the accuracy of the numerical rank calculated by SPQR and reliably determine the numerical rank in the sense that, based on extensive testing with matrices from applications, the numerical rank is almost always accurately determined when our methods report that the numerical rank should be correct. Reliable determination of numerical rank is critical to the other calculations in the package. The routines work well for matrices with either small or large null space dimensions.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classification—*MATLAB*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: numerical rank, null space, rank revealing, QR factorization, pseudoinverse, sparse matrices

1. INTRODUCTION

For an $m \times n$ matrix A, the numerical rank of A can be defined as the number of singular values larger than a specified tolerance τ. Calculation of numerical rank is

This work was supported in part by the Woodward bequest to the Department of Mathematics, San Jose State University and the National Science Foundation, under grants 0619080 and 02202286.

Authors' addresses: L. V. Foster, Department of Mathematics, San Jose State University, San Jose, CA 95192-0103 (foster@math.sjsu.edu); T. A. Davis, CISE Department, University of Florida, Gainesville, FL 32611-6120 (davis@cise.ufl.edu);

important in the presence of rounding errors and fuzzy data [Golub and Van Loan 1996, p. 72]. If the numerical rank of a matrix A is r and $\mathcal{X}$ is a subspace of dimension $n-r$, we say that $\mathcal{X}$ is a numerical null space of A for tolerance τ if

$$\max_{x\in\mathcal{X},x\neq 0} ||Ax||/||x|| \leq \tau. \tag{1}$$

Throughout the paper $||\cdot||$ indicates the Euclidean norm. It follows that if N is an $n\times(n-r)$ matrix whose columns form an orthonormal basis for $\mathcal{X}$, then

$$||AN|| \leq \tau. \tag{2}$$

If we wish to solve the least squares problem

$$\min_x ||Ax-b|| \tag{3}$$

and if the numerical rank r of A is less than n, then it is useful to find the minimal norm or pseudoinverse solution [Björck 1996, p. 15] to

$$\min_x ||\widehat{A}x-b||. \tag{4}$$

where $\widehat{A}$ has (exact) rank r and is close to A. We call the minimum norm solution to (4) the approximate pseudoinverse solution to (3). Another solution to (3) that can be useful is a basic solution. Here, by basic solution we mean an approximate solution to (3) where the number of nonzero components in x is less than or equal to the estimated rank of A determined by the routine SPQR. Calculations of numerical ranks, numerical null spaces, approximate pseudoinverse solutions, and basic solutions are useful in many applications [Chan and Hansen 1992], [Enting 2002], [Gotsman and Toledo 2008], [Hansen 1998], [Li and Zeng 2005]. The focus of this paper is on the calculation of these quantities for sparse and potentially large matrices in the presence of computer arithmetic or other errors.

The routines in SPQR_RANK calculate estimates of upper and lower bounds for singular values of A and use these estimates to report a warning when the calculated numerical rank may be incorrect. Section 3.5 demonstrates that either the rank is accurately determined or an accurate warning is returned (with one rare exception).

The most widely used method for determination of the numerical rank, an orthonormal basis for the numerical null space, and an approximate pseudoinverse solution to (3) is the singular value decomposition (SVD): $A=UDV^T$, where U is an $m\times m$ orthogonal matrix, V is an $n\times n$ orthogonal matrix, and D is an $m\times n$ diagonal matrix whose diagonal entries, $\sigma_1\geq\sigma_2\geq\ldots\geq\sigma_{\min(m,n)}$, are the singular values of A. Let r be the numerical rank of A for some tolerance τ, so that $\sigma_r>\tau$ and $\sigma_{r+1}\leq\tau$. Also let X be the last $n-r$ columns of V and $\mathcal{X}$ be the span of the columns of X. It follows from the SVD definition, $A=UDV^T$, that

$$\max_{x\in\mathcal{X},x\neq 0} ||Ax||/||x|| = \sigma_{r+1} \quad \text{and} \quad ||AX|| = \sigma_{r+1}. \tag{5}$$

Furthermore, by the minimax characterization of singular values [Björck 1996, p. 14] for any other subspace $\mathcal{X}$ of dimension $n-r$

$$\max_{x\in\mathcal{X},x\neq 0} ||Ax||/||x|| \geq \sigma_{r+1}.$$

Although the singular value decomposition is, in the above sense, the most accurate method for calculating the numerical rank and a numerical null space, the SVD is expensive to compute for a large matrix because sparsity cannot be exploited in $\mathcal{X}$. Therefore, it is of interest to explore alternatives to the SVD for these problems.

A critical tool for our algorithms is the SuiteSparseQR package [Davis 2011]. The SPQR routine in SuiteSparseQR is a high-performance, sparse QR factorization based on the multifrontal method. In addition to the speed of SPQR, two features that are important for our use are the ability to estimate the numerical rank of A and the ability to represent an orthogonal matrix, Q, in sparse format using Householder transformations. The sparse representation of Q leads to a sparse representation for the orthonormal basis of the numerical null space, enabling the computation of null spaces of high dimension.

The numerical rank estimated by SPQR relies on Heath's method [Heath 1982], which is often but not always accurate [Foster 1990]. We estimate bounds on certain singular values of A to determine if the numerical rank calculated by SPQR is correct. Furthermore, if the numerical rank returned by SPQR is incorrect, our routines are able, in most cases, to correct the numerical rank and determine a corresponding orthonormal basis for the numerical null space, basic solution, and an approximate pseudoinverse solution to (3).

It can be difficult to accurately determine the numerical rank if the tolerance τ is near a singular value of A. In this case, changes in the tolerance or in A can change the numerical rank. However, when there is a significant gap in the singular values so that $\sigma_r >> \sigma_{r+1}$, τ lies between σ_{r+1} and σ_r, and τ is near neither, then small changes in A or the tolerance τ will not affect the numerical rank. Hansen [Hansen 1998, p. 2] uses the term "rank-deficient problem" for matrices with a cluster of small singular values with a well-determined gap between large and small singular values. [Pierce and Lewis 1997, p. 177, 179] use the term "well-posed" for problems with a significant gap in the singular values. Following [Bischof and Quintana-Ortí 1998, p. 227], we use the term "well defined numerical rank" for matrices with a well-determined gap in the singular values. Our routines often work better for matrices with a well defined numerical rank.

In the following section we discuss SPQR and our new algorithms:

(1) SPQR_BASIC, which determines a basic solution to (3);
(2) SPQR_NULL, which constructs an orthonormal basis for the numerical null space of A;
(3) SPQR_PINV, which constructs an approximate pseudoinverse solution to (3);
(4) and SPQR_COD, which uses the complete orthogonal decomposition, defined below, to construct the approximate pseudoinverse solution to (3).

As is discussed in Section 2.7, the routines require varying amounts of computations and memory use. In Section 3 we discuss numerical experiments and Section 4 contains conclusions.

2. ALGORITHMS AND SINGULAR VALUE ERROR BOUNDS

The four algorithms mentioned above all estimate upper and lower bounds on certain singular values to determine whether the numerical ranks determined by the

algorithms appear to be correct. Four well known results below are used to determine the estimated bounds; the first three provide rigorous error bounds for singular value locations, and the last theorem is used to estimate error bounds on certain singular values:

THEOREM 1. **(Singular Value Perturbation Bounds)** *[Golub and Van Loan 1996, p. 449] If A and E are m by n matrices, then for $k = 1, 2, \ldots, \min(m, n)$ $|\sigma_k(A + E) - \sigma_k(A)| \leq ||E||$ where $\sigma_k(A)$ is the k^{th} singular value of matrix A.*

THEOREM 2. **(Singular Value Interlace Property)** *[Golub and Van Loan 1996, p. 449] Let $A = [a_1, \ldots, a_n]$ be a column partition of the m by n matrix A with $m \geq n$. If $A_r = [a_1, \ldots, a_r]$, then for $r = 1, \ldots, n-1$*

$$\sigma_1(A_{r+1}) \geq \sigma_1(A_r) \geq \sigma_2(A_{r+1}) \geq \ldots \geq \sigma_r(A_{r+1}) \geq \sigma_r(A_r) \geq \sigma_{r+1}(A_{r+1}).$$

THEOREM 3. **(Singular Value Minimax Property)** *[Björck 1996, p. 14] If A is an m by n matrix and if S is a subspace of R^n, then for $k = 1, 2, \ldots, \min(m, n)$*

$$\sigma_k(A) = \min_{\dim(S)=n-k+1} \max_{x \in S,\ x \neq 0} \frac{||Ax||}{||x||}.$$

THEOREM 4. **(Eigenvalue Error Bound)** *[Demmel 1997, p. 205] Let A be an n by n symmetric matrix, v be a unit vector in R^n, and λ be a scalar. Then A has an eigenpair $Av_i = \lambda_i v_i$ satisfying $|\lambda_i - \lambda| \leq ||Av - \lambda v||$.*

Our routines SPQR_BASIC, SPQR_NULL, SPQR_PINV, and SPQR_COD (discussed in Sections 2.3 to 2.6) are based on SPQR (see Section 2.1) and subspace iteration applied to the inverse of certain matrices to estimate small singular values (SPQR_SSI, in Section 2.2). These methods rely upon our SPQR_SSP routine to estimate matrix norms (see the Appendix). Section 2.7 summarizes the key differences between our algorithms (see Table I). Section 2.8 briefly discusses other algorithms in the literature that can be used for calculation of numerical rank, pseudoinverse solutions, basic solutions, or numerical null spaces of a matrix.

2.1 SPQR

SPQR [Davis 2011] is a high-performance, multifrontal sparse QR factorization method that appears as the built-in `qr` in MATLAB. It returns an estimate of the numerical rank ℓ of A [Heath 1982], and decomposes A into

$$AP_1 = Q_1 R + E_1 = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix} + E_1 = Q_1 \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} + E_1 \tag{6}$$

where P_1 is an $n \times n$ permutation matrix, Q_1 is an $m \times m$ orthogonal matrix, E_1 is an $m \times n$ error matrix, R is a $m \times n$ right trapezoidal matrix (assuming $\ell < n$), R_1 is an $\ell \times n$ right trapezoidal matrix, R_{11} is a $\ell \times \ell$ triangular matrix whose diagonal entries are larger in magnitude than a given tolerance τ, and R_{12} is $\ell \times (n-\ell)$. SPQR can return Q as a sparse matrix or as a set of sparse the elementary Householder transformations. We rely upon the latter, which takes much less memory but is not available via the built-in MATLAB `qr` function. In most cases ℓ equals the true

numerical rank r of A, but in some cases R_{11}, although technically nonsingular, is very ill-conditioned. In such a case $r < \ell$, which we discuss in in the next section. SPQR also returns $||E_1||_F$, where $||\cdot||_F$ indicates the Frobenius norm.

Given a tolerance τ, Heath's technique estimates the numerical rank of A by comparing the magnitude of diagonal entries in R to τ. In SPQR the implementation of Heath's idea is done by triangularizing the matrix A using Householder transformations. If, after the i^{th} column is forced to zero below the diagonal, the magnitude of the diagonal entry is less than or equal to τ, then the diagonal entry r_{ii} is set to zero. The i^{th} column is implicitly permuted to the right of the current partly triangularized matrix and the new i^{th} column is processed with Householder transformations.

THEOREM 5. *Let A be an $m \times n$ matrix, let (6) be the decomposition produced by SPQR, and let w be a vector consisting of the diagonal entries that are set to zero during the QR factorization in SPQR. Then*

$$||E_1||_F = ||w|| \quad \textit{and} \tag{7}$$

$$||E_1|| \leq ||w|| \leq \sqrt{n-\ell}\ \tau. \tag{8}$$

PROOF. Assume orthogonal transformations Q_1 and P_1 in (6) are applied to A without dropping diagonal entries and define the unperturbed factorization

$$\widetilde{R} = Q_1^T A P_1. \tag{9}$$

Compare $\widetilde{R}$ in the unperturbed factorization (9) with R in (6). Suppose for some i, $r_{ii} \leq \tau$. The i^{th} column of R is moved to the right and r_{ii} is set to zero so that the entire column on or below element i is zero. Subsequent Householder transformations act only on a subset of rows i to m of this column. Thus, in the perturbed matrix (with $r_{ii} = 0$) the column remains zero on or below element i. When the orthogonal transformations are applied in the unperturbed factorization (9), at step i of the algorithm the norm of rows i to m of column i has magnitude r_{ii}. Subsequent orthogonal Householder transforms do not change this norm. Therefore, when the algorithm is complete, for each column with a diagonal entry set to zero, the norm of the difference in the column of R and $\widetilde{R}$ is $|r_{ii}|$. The other columns of R and $\widetilde{R}$ are identical. $||E||_F = ||w||$ in (7) follows. The inequalities in (8) follow by norm properties and since w can have at most $n - \ell$ nonzero entries. □

2.2 Subspace Iteration to Estimate Singular Values of a Nonsingular Triangular Matrix

Usually ℓ, the estimated numerical rank determined by SPQR, is a good estimate of the correct numerical rank r of A: in most cases $\ell = r$ and in many other cases ℓ and r are relatively close. If so, the $\ell \times \ell$ matrix R_{11} (and also the $\ell \times \ell$ matrix T defined in Section 2.6) has a low dimensional numerical null space. Let R represent any $\ell \times \ell$ nonsingular, triangular matrix with a low dimensional numerical null space. For example R could be R_{11}. We can efficiently estimate the smallest singular values of such a matrix R and the corresponding singular vectors using an appropriate iterative method applied to $R^{-T}R^{-1}$. One could use, for example, Lanczos method as implemented in ARPACK [Lehoucq et al. 1998]. We found,

however, that subspace iteration (sometimes called orthogonal iteration [Golub and Van Loan 1996, p. 332]) is more reliable and more suitable for our use.

There is a variety of potential implementations of subspace iteration when calculating eigenvalues of symmetric matrices [Parlett 1980, pp. 289-293] and these can be adapted to finding singular values of nonsymmetric matrices [Berry 1994], [Vogel and Wade 1994], [Gotsman and Toledo 2008]. Our implementation of subspace iteration, which we call algorithm SPQR_SSI, is given on the following page.

The key to the stopping criteria in SPQR_SSI is to continue the algorithm until the estimates of singular value r and $r+1$ are sufficiently accurate so that

$$\sigma_r(A) > \tau \quad \text{and} \quad \sigma_{r+1}(A) \leq \tau. \tag{10}$$

We describe some of the details in terms of the final U, V and $s_1, \ldots, s_k$. The code uses equivalent conditions involving $\dot{s}_1, \ldots, \dot{s}_k$, U, and V in the repeat loop (except for condition (2), which is calculated after leaving the loop for efficiency):

(1) (a) $s_1 > \tau$ and (b) $s_2 \leq \tau$
(2) (a) $||RV(:, 2:k)|| \leq \tau$ and (b) $||R^T U(:, 2:k)|| \leq \tau$ (using MATLAB notation)
(3) $e_1 \leq f|s_1 - \tau|$
(4) $e_1 \leq f s_1$

The algorithm fails if the maximum number of iterations or maximum block size is exceeded without meeting all of these four conditions for success. By the singular value minimax property (Theorem 3) in exact arithmetic each of (1b), (2a) or (2b) imply the second condition in (10). In finite precision arithmetic, it is useful to require all three conditions to insure that $\sigma_{r+1}(A) \leq \tau$.

Theorem 4 applied to columns of $\begin{pmatrix} 0 & R \\ R^T & 0 \end{pmatrix} \begin{pmatrix} U \\ V \end{pmatrix} - \begin{pmatrix} U \\ V \end{pmatrix} S = \begin{pmatrix} RV - US \\ 0 \end{pmatrix}$ implies that, for each j, $1 \leq j \leq k$, $|s_j - \alpha| \leq e_j$ for some singular value α of R. Therefore, for each j, $1 \leq j \leq k$, e_j is a bound on the error in using s_j to approximate some singular value of R. In our testing usually $|s_j - \sigma_{\ell-k+j}| \leq e_j$. However, since this is not guaranteed by the theory, we consider e_j to be an estimate for a bound on the error in approximating $\sigma_{\ell-k+j}$ with s_j.

If we assume that $|s_1 - \sigma_r| \leq e_1$, then this inequality, (1a), and (3) with $f = 1$ imply the first condition in (10). Since e_1 is only an estimate for a bound on $|s_1 - \sigma_r|$, in our stopping criteria we require that (3) and (4) are true with $f < 1$. The default value in our code, $f = 0.1$, works well in the testing described in Section 3.

In most cases the estimated rank determined by SPQR is identical or close to the true numerical rank. Therefore, the block size b in Algorithm SPQR_SSI is usually small. At each iteration the singular value decompositions take $O(\ell b^2)$ operations, which is thus normally a small amount of work. The extra work for increasing the size of U also requires $O(\ell b^2)$ operations per step, when necessary. Because b is increased as needed, the choice of the initial block size is not critical.

Algorithm SPQR_SSI is related to Algorithm SI in [Vogel and Wade 1994, p. 741] and Algorithm SPIT in [Chan and Hansen 1990, p. 525]. Neither of these algorithms dynamically increase the block size and Algorithm SI estimates large singular values, not small singular values.

Theorem 3.1 of [Vogel and Wade 1994, p. 742] describes the convergence rates of the approximate singular values in Algorithm SI to the singular values of R. Since

SPQR_SSI: subspace iteration to estimate singular values
Input:
• R: an $\ell \times \ell$ nonsingular, triangular matrix
• τ: the tolerance defining the numerical rank of R
• other options including init_block_size, the initial block size (default 3); block_size_increment, the increment for the block size (default 5); max_block_size, the maximum block size (default 10); max_iters, the maximum number of iterations (default 100); and convergence factor $f \leq 1$ (default 0.1), discussed above
Output:
• U: an $\ell \times k$ matrix containing estimates of the left singular vectors of R corresponding to estimated singular values in S. Upon success, $r = \ell - k + 1$ is the estimated numerical rank of R and $U(:, 2:k)$ is an orthonormal basis for the numerical null space of R^T.
• S: a $k \times k$ diagonal matrix whose diagonal entries $s_1 \geq s_2 \geq \ldots \geq s_k$ are the estimated smallest k singular values of R, with k as described above.
• V: an $\ell \times k$ matrix containing estimates of the right singular vectors of R corresponding to estimated singular values in S. Upon success, $V(:, 2:k)$ is an orthonormal basis for the numerical null space of R.
• `stats`: a structure containing additional information including the estimated numerical rank, r, of R; $e_j, j = 1, \ldots, k$, which are estimates for bounds on the errors in approximating singular value $\sigma_{\ell-j+k}(R)$ with s_j; and a flag indicating if the method is successful. If successful, the numerical rank appears to be correct, $s_1 > \tau$, and $s_2 \leq \tau$.
Initialize:
• b = the current block size = init_block_size
• U = a random $\ell \times b$ matrix with orthonormal columns
repeat
  • Solve the triangular system $RV_1 = U$ to calculate $V_1 = R^{-1}U$.
  • Determine the compact SVD of V_1: $VD_1X_1^T = V_1$, where V is $\ell \times b$, D_1 is a $b \times b$ diagonal matrix, and X_1 is $b \times b$
  • Solve the triangular system $R^TU_1 = V$ to calculate $U_1 = R^{-T}V$.
  • Determine the compact SVD of U_1: $UD_2X_2^T = U_1$, where U is $\ell \times b$, D_2 is a $b \times b$ diagonal matrix, and X_2 is $b \times b$
  • let $\dot{s}_i$, $i = 1, 2, \ldots, b$, be diagonal entries of D_2
  if $\dot{s}_b \geq 1/\tau$ **then**
    • Y = orthonormal basis for a block_size_increment dimensional subspace orthogonal to the column space of U
    • Redefine $U = [U, Y]$ and increase b
  end
until *until stopping criteria is met, see the previous page* ;
if *stopping criteria for success is met* **then**
  • Let k be the smallest i with $\dot{s}_i < 1/\tau$ and let $r = \ell - k + 1$
  • Redefine V: let $V = VX_2$ and reorder, in reverse order, columns 1 to k of V
  • Redefine U: reorder, in reverse order, columns 1 to k of U
  • $s_j = 1/\dot{s}_{\ell-r-j+2}$, $j = 1, 2, \ldots, k$
  • For $j = 1, 2, \ldots, k$ let $e_j = ||Rv_j - u_j s_j||/\sqrt{2}$ where v_j is the j^{th} column of V and u_j is the j^{th} column of U
  • Report success
else
  • Report failure
end

the singular values of R^{-1} are the reciprocals of the singular values of R, the results of [Vogel and Wade 1994] can be adapted to inverse iteration. The convergence rates of the approximate singular values in Algorithm SPQR_SSI are described by the adapted results. Let σ_j, $j = 1, \ldots, \ell$, be the singular values of R and note that upon termination of Algorithm SPQR_SSI, s_{j-r+1} is an approximation to σ_j for $j = r, r+1, \ldots, \ell$. Assume the algorithm is successful, terminating with $b > \ell - r$. In addition, assume that $\sigma_{\ell-b} > \sigma_{\ell-b+1}$. Let p be the number of iterations in the algorithm. Then the relative errors for the singular value approximations obtained by Algorithm SPQR_SSI converge to zero at the asymptotic rate

$$\frac{\sigma_j - s_{j-r+1}}{\sigma_j} = O\left(\left(\frac{\sigma_j}{\sigma_{\ell-b}} \right)^{4p} \right), \quad j = r, r+1, \ldots, \ell. \tag{11}$$

Note that $\sigma_j/\sigma_{\ell-b} \leq \sigma_{r+1}/\sigma_r$ for $j = r+1, \ldots, \ell$. It follows that when the numerical rank is well defined, in the sense that $\sigma_{r+1} << \sigma_r$, convergence will be rapid to singular values $\sigma_{r+1}, \sigma_{r+2}, \ldots, \sigma_\ell$.

2.3 Basic Solution to a Rank Deficient Least Squares Problem

To determine a basic solution (3), we initially apply SPQR to A and use SPQR_SSI to determine if the estimated numerical rank returned by SPQR is correct, and then correct it if necessary. See the following page for the SPQR_BASIC pseudocode.

As discussed in the proof of Theorem 5, no perturbations are made to the columns of A that are used in the construction of R_{11}. From this fact and Theorem 2, it follows that $\sigma_i(R_{11}) \leq \sigma_i(A)$ for $i = 1, 2, \ldots, \ell$. This is the justification for the estimated lower bounds calculated in the first part of step (5). It follows from (6), Theorem 5, the singular value perturbation bound (Theorem 1), and the singular value minimax property (Theorem 3), that the values returned in the second part of step (5) are upper bounds for singular values $r, r+1, \ldots, \min(m, n)$ of A. This is the justification of the upper bound in step (5) when the null space of A^T is not requested. By the singular value minimax property (Theorem 3) the largest singular value of $A^T\widehat{N}$ is an upper bound on singular value $r+1$ of A. This is the justification for estimated upper bound on singular value $r+1$ of A in step (5) when the null space of A^T is requested. In step (5) the cost of calculating the singular values of $U^T R_1$ is usually relatively small since k is usually small: the default parameter choice restricts $k \leq 10$.

Since our codes use finite precision arithmetic, the numerical rank returned by SPQR_BASIC corresponds to exact calculation applied to $A+E$ where E is $O(\epsilon||A||)$ and ϵ relative machine precision. Thus, the tolerance used to define the numerical rank should be larger than $\epsilon||A||$. The default choice in our code is the MATLAB expression `max(m,n)*eps(normest(A,0.01))` where `eps(x)` is the spacing in the floating point number system near `x` and `normest(A,0.01)` estimates $||A||$. This choice is essentially the same as the choice in the MATLAB `rank` command and it works well in the numerical experiments described in Section 3. Since SPQR_BASIC uses estimates of bounds on the singular values to determine if it is successful, the algorithm does not guarantee that the calculated numerical rank is correct when it reports success. However, the numerical experiments in Section 3 indicate that in practice the method provides a very reliable indicator of its success or failure.

SPQR_BASIC: approximate basic solution to $\min ||b - Ax||$
Input:
• an $m \times n$ matrix A and an $m \times p$ right hand side B
• the tolerance τ defining the numerical rank of A
• other user selectable parameters with default values supplied by the routine
Output:
• an $n \times p$ matrix x. The j^{th} column of x contains a basic solution to (3) with $b =$ the j^{th} column of of B
• a structure `stats` that contains the calculated numerical rank and estimates of upper and lower bounds for singular values of A.
• optionally, SPQR_BASIC can return an orthonormal basis for the null space of A^T
Calculations:

(1) Apply SPQR, with tolerance set to τ, to A, returning P_1, R_1, R_{11} and R_{12} in (6) and $||w||$ where w is the vector of perturbations described in Theorem 5. If an orthonormal basis for A^T is requested, also return Q_1. Let ℓ be the estimated numerical rank returned by SPQR so that R_{11} is $\ell \times \ell$.

(2) Apply SPQR_SSI to R_{11} with tolerance set to τ.

(3) Let $c = Q_1^T b$ and $\widehat{c} =$ the first ℓ components of c. If $r = \ell$, let $\widehat{z}$ be the solution to $R_{11}\widehat{z} = \widehat{c}$. If $r < \ell$, let $U_2 = U(:, 2:k)$, $V_2 = V(:, 2:k)$, and calculate $\widehat{z}$ using deflation [Stewart 1981], [Chan and Hansen 1990]: $\widehat{z} = (I - V_2 V_2^T) R^{-1} (I - U_2 U_2^T)\widehat{c}$. Let $z = [\widehat{z}; 0]$ with zeros added so that z has n components and let $x = P_1 z$.

(4) Calculate a null space basis, $\widehat{N}$, of A^T, if requested:
if $r = \ell$ **then**
$\widehat{N} = Q_1 * [0; I]$ where, here and below, I is an $n - \ell$ by $n - \ell$ identity matrix.
else if $r < \ell$ **then**
$\widehat{N} = Q_1 * [U_2, 0; 0, I]$.

(5) Check that the calculated numerical rank, r, is correct:
• Lower bounds: Estimated lower bounds for $\sigma_i(A), i = r, r+1, \ldots, \ell$, are $s_j - e_j, j = 1, 2, \ldots, k$, respectively. Zero is a lower bound to $\sigma_i(A)$, $i = \ell + 1, \ell + 2, \ldots, \min(m, n)$.
• Upper bounds:
if *the nullspace of* A^T *is not requested* **then**
For upper bounds for singular values $r, r+1, \ldots, \ell$ of A, return $||w||$ plus the singular values $1, 2, \ldots, k$ of $U^T R_1$ (the latter computed by `svd` in MATLAB). For upper bounds on $\sigma_i(A)$, $i = \ell + 1, \ell + 2, \ldots, \min(m, n)$, return $||w||$.
else
For singular value $r + 1$ of A return the minimum of the above upper bound for this singular value and an estimated upper bound calculating using SPQR_SSP (see the Appendix). The sum of the estimated largest singular value of $A^T \widehat{N}$ and the error bound on this estimate, as calculated by SPQR_SSP, is an estimate of an upper bound on singular $r + 1$ of A.

• Check the numerical rank: If the estimated lower bound for $\sigma_r(A) > \tau$ and if the estimated upper bound for $\sigma_{r+1}(A) \leq \tau$, then the routine is successful. If the estimated lower bound for $\sigma_r(A)$ is greater than the estimated upper bound for $\sigma_{r+1}(A) > \tau$, then raise a warning and return τ_{alt}, an alternate tolerance value, set equal to the estimated upper bound for $\sigma_{r+1}(A)$. The calculated numerical rank with tolerance equal to τ_{alt} appears to be correct. Otherwise, report an error.

2.4 Null Space Calculation using SPQR_NULL

SPQR_NULL calculates an orthonormal basis for the null space of A. It applies SPQR_BASIC to A^T, rather than A, using the option to return a null space basis to A and skipping the computation of the solution vector (step (2) of SPQR_BASIC).

Our calculation of a null space basis requires storage of Q_1. The default option for SPQR_NULL is to represent the null space basis in an implicit form using a feature of SPQR that represents Q_1 in terms of sparse Householder transformations. The null space is then represented implicitly by storing the orthogonal matrix Q_1 and the matrix

$$X = \begin{pmatrix} U(:, 2:k) & 0 \\ 0 & I \end{pmatrix}. \tag{12}$$

The matrix X is stored as a sparse matrix. This is usually a good storage mode since the number of columns of $U(:, 2:k)$ is usually small. A routine SPQR_NULL_MULT is supplied to multiply an implicit null space basis by another matrix. The implicit storage mode makes it practical to calculate and represent null spaces of matrices with high dimensional null spaces.

2.5 Pseudoinverse solution using SPQR_PINV

The routine SPQR_PINV calculates an approximate pseudoinverse solution to (3) by calling SPQR_BASIC and SPQR_NULL. Step 3 of SPQR_PINV can, in some cases, lead to loss of precision in the calculated x, for example if $||x_B||$ is significantly larger than $||x||$. This potential problem is discussed further in Section 3.2.4.

SPQR_PINV: pseudoinverse solution to $\min ||b - Ax||$
Input: same input as algorithm SPQR_BASIC
Output:
• an $n \times p$ matrix x. The j^{th} column of x contains the pseudoinverse solution to (3) with $b =$ the j^{th} column of of B
• the structure `stats` described in SPQR_BASIC
• optionally, return an orthonormal basis for the null space of A and of A^T
Calculations:

(1) Apply SPQR_BASIC with tolerance set to τ to A producing basic solution x_B
(2) Apply SPQR_NULL with tolerance τ to A producing an orthonormal basis N for the numerical null space of A.
(3) The pseudoinverse solution is $x = x_B - N(N^T x_B)$. This can be calculated using the routine SPQR_NULL_MULT mentioned following equation (12).
(4) Optionally, if the user requests, return N, an orthonormal basis for the numerical null space of A calculated by SPQR_NULL, and $\widehat{N}$, an orthonormal basis for the numerical null space of A^T calculated by SPQR_BASIC.
(5) Report success if both SPQR_BASIC and SPQR_NULL report success. Return estimates of upper bounds for singular values of A by choosing the maximum of the estimated upper bounds returned by SPQR_BASIC and SPQR_NULL and return estimates of lower bounds for singular values of A by choosing the minimum of the estimated lower bounds returned by SPQR_BASIC and SPQR_NULL.

2.6 Calculations using a Complete Orthogonal Decomposition and SPQR_COD

The routines SPQR_BASIC, SQPR_NULL, and SPQR_PINV often work well in practice, but occasionally they fail or they return upper and lower singular value bounds that are significantly different. In these cases an algorithm using the complete orthogonal decomposition can be useful. For an $m \times n$ matrix A, consider the decomposition

$$A = U \begin{pmatrix} T & 0 \\ 0 & 0 \end{pmatrix} V^T + E \tag{13}$$

where U is an $m \times m$ orthogonal matrix; V is an $n \times n$ orthogonal matrix; T is an $r \times r$ nonsingular, triangular matrix; and E is a small error matrix. If E is zero, then (13) is the complete orthogonal decomposition (COD) [Golub and Van Loan 1996, p. 250], [Björck 1996, p. 23] and r is the rank of A. If E is small but not zero, we call (13) the approximate complete orthogonal decomposition. The COD is potent since it can be used to determine the rank of A, the fundamental subspaces of A, and the pseudoinverse solution to (4) [Golub and Van Loan 1996, p. 256], [Björck 1996, pp. 110-111]. The approximate COD can determine the numerical rank, numerical fundamental subspaces, and an approximate pseudoinverse solution to (3). Pseudocode for SPQR_COD is given on the following page.

Applying Theorem 5 to equation (15), it follows that $||E_2||_F = ||w||$ and $||E_2|| \leq ||w||$. Therefore, by the singular value perturbation property, Theorem 1, $|\sigma_i(A) - \sigma_i(T)| \leq ||w||$ for $i = 1, \ldots, \ell$. Also, $s_j - e_j, j = 1, 2, \ldots, k$, are estimated lower bounds for $\sigma_i(T), i = r, r+1, \ldots, \ell$, respectively. These comments provide justification for the estimated lower bounds returned by SPQR_COD. Justification of the upper bounds follow from similar arguments.

2.7 Comparison of Algorithms

Table I compares the routines SPQR_BASIC, SPQR_NULL, SPQR_PINV, and SPQR_COD. The last column reflects the results of the experiments in Section 3.2.

The memory requirements and computational work listed in Table I are approximate. For example, in the computation work column the work to calculate the error estimates is not included and the memory requirements column does not include the memory needed for storing permutation matrices (which are stored as vectors in the code). The orthogonal matrices are stored in terms of sparse Householder factors which can be a significant savings in memory.

The table suggests that SPQR_BASIC requires the least memory and computation work. SPQR_NULL requires approximately the same work but more memory. SPQR_PINV calls both of these algorithms and requires more memory and work. SPQR_COD begins with a sparse QR factorization of A which results in fill-in while calculating R_1. The algorithm follows this with a sparse QR factorization of R_1^T. These sequential factorizations can compound the fill-in, leading to larger memory requirements and work than is required by SPQR_PINV, which factors A and A^T separately.

2.8 Other Algorithms

There are many algorithms [Davis 2011, p. 2] that solve the least squares problem (3) using sparse QR factorizations. However, only SPQR [Davis 2011] and the

SPQR_COD: approximate pseudoinverse solution to $\min ||b - Ax||$
Input: same input as algorithm SPQR_PINV
Output: same output as algorithm SPQR_PINV
Calculations:

(1) Apply SPQR, with tolerance set to τ, to A, producing P_1, R_1, R_{11} and R_{12} in (6) and $||w||$, where w is the vector of perturbations described in Theorem 5. Let ℓ be the estimated numerical rank returned by SPQR so that R_1 is $\ell \times n$.

(2) Apply SPQR with tolerance set to zero to R_1^T, constructing an $\ell \times \ell$ permutation matrix $\widehat{P}_2$, an $n \times n$ orthogonal matrix Q_2, and an $\ell \times \ell$ right triangular matrix T such that

$$R_1^T \widehat{P}_2 = Q_2 \begin{pmatrix} T \\ 0 \end{pmatrix}. \tag{14}$$

Let the $m \times m$ permutation matrix $P_2 = \begin{pmatrix} \widehat{P}_2 & 0 \\ 0 & I \end{pmatrix}$. It follows from (6) and (14) that

$$A = Q_1 P_2 \begin{pmatrix} T^T & 0 \\ 0 & 0 \end{pmatrix} Q_2^T P_1^T + E_2. \tag{15}$$

Here $E_2 = E_1 P_1^T$. Since $Q_1 P_2$ and $P_1 Q_2$ are orthogonal then (15) is an approximate complete orthogonal decomposition of A.

(3) Apply SPQR_SSI to T with tolerance set to τ.

(4) Let $c = P_2^T Q_1^T b$ and $\widehat{c} =$ the first ℓ components of c. If $r = \ell$, let $\widehat{z}$ be the solution to $T^T \widehat{z} = \widehat{c}$. If $r < \ell$, let $U_2 = U(:, 2:k)$, $V_2 = V(:, 2:k)$, and calculate $\widehat{z}$ using deflation [Stewart 1981], [Chan and Hansen 1990]: $\widehat{z} = (I - U_2 U_2^T) T^{-T} (I - V_2 V_2^T) \widehat{c}$. Let $z = [\widehat{z}; 0]$ adding zeros so that z has n components. Finally, let $x = P_1 Q_2 z$.

(5) Calculate a null space basis, N, of A, if requested:
if $r = \ell$ **then**
$N = P_1 * Q_2 * [0; I]$ where I is an $n - \ell$ by $n - \ell$ identity matrix
else if $r < \ell$ **then**
$N = P_1 * Q_2 * [U_2, 0; 0, I]$
Calculate a null space basis for A^T, if requested, in a similar manner.

(6) Check if the calculated numerical rank, r, is correct:
• Lower bounds: Return $s_j - e_j - ||w||, j = 1, 2, \ldots, k$, for estimated lower bounds for $\sigma_i(A), i = r, r+1, \ldots, \ell$, respectively. Return zero for an lower bound to singular values $\sigma_i(A)$, $i = \ell + 1, \ell + 2, \ldots, \min(m, n)$.
• Upper bounds: Return $s_j + e_j + ||w||, j = 1, 2, \ldots, k$, for estimated upper bounds for $\sigma_i(A), i = r, r+1, \ldots, \ell$, respectively. Return $||w||$ for an upper bound to singular values $\sigma_i(A)$, $i = \ell + 1, \ell + 2, \ldots, \min(m, n)$. The basis N for the null space of A can be used to improve the upper bound for $\sigma_{r+1}(A)$.
• Check the numerical rank with the same procedure used in SPQR_BASIC.

algorithm discussed in [Pierce and Lewis 1997], which is based in part on ideas from [Foster 1986], can handle rank deficient matrices and also implement the efficient multifrontal approach [Davis 2011, p. 9]. [Pierce and Lewis 1997] use a dynamic mixture of Householder and Givens rotations, which would make Q difficult to keep. Thus, their method does not keep a representation of Q, but rather discards the transformations as they are computed. Our methods presented in this paper, except for SPQR_BASIC, require Q. Least squares problems with rank deficient

Algorithm	Primary Application	Principal Memory Requirements	Principal Computational Work	Accuracy
SPQR_BASIC	Basic Solution to (3)	R_1 in (6), U, V from SPQR_SSI	SPQR to A, SPQR_SSI to R_{11} in (6)	usually good
SPQR_NULL	Orthonormal Null Space Basis	R_1 and Q_1 in (6), U, V from SPQR_SSI	SPQR to A^T, SPQR_SSI to R_{11} in (6), SPQR_SSP to (AN)	usually good
SPQR_PINV	Approximate pseudoinverse solution to (3)	maximum of SPQR_BASIC, SPQR_NULL memory	sum of SPQR_BASIC, SPQR_NULL work	usually good
SPQR_COD	Approximate pseudoinverse solution to (3)	R_1 in (6), Q_2, T in (14), U, V from SPQR_SSI	SPQR to A, SPQR to R_1^T in (6), SPQR_SSI to T, SPQR_SSP to (AN)	good

Table I. Comparison of SPQR_BASIC, SPQR_NULL, SPQR_PINV, and SPQR_COD

matrices can also be solved using iterative techniques such as LSQR [Paige and Saunders 1982] or LSMR [Fong and Saunders 2010]. Investigation of these iterative techniques is beyond the scope of this paper.

There are other algorithms that can potentially construct an orthonormal null space basis of a sparse matrix A. These include the algorithm discussed in [Gotsman and Toledo 2008] and the MATLAB `svds` based on ARPACK [Lehoucq et al. 1998]. The algorithm of [Gotsman and Toledo 2008] uses symmetric inverse iteration with an LU factorization of A. Symmetric inverse iteration is inverse iteration used with solutions to $A^T Ax = y$. `svds` uses Arnoldi/Lanczos type algorithms on

$$B = \begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}. \tag{16}$$

To find a numerical null space basis `svds` uses the "shift and invert" technique [Lehoucq et al. 1998] with a shift of zero.

The database described in Section 3 includes matrices whose nullity is very large (hundreds of thousands). Gotsman and Toledo's algorithm is practical only when the nullity is small or moderate [Gotsman and Toledo 2008, p. 447]. This is also true for Arnoldi/Lanczos based methods such as `svds`. In comparison, our algorithms can successfully calculate null space bases of matrices with large nullity – larger than 100,000. `svds` often fails to produce an acceptable null space [Gotsman and Toledo 2008, p. 460]. Finally, for both `svds` and the algorithm of [Gotsman and Toledo 2008] it is not clear what to choose for the nullity or subspace dimensions in the code. The algorithms presented in this paper automatically select the appropriate numerical rank and nullity in most cases.

One goal of this paper is an algorithm that finds a sparse representation of an orthonormal basis of the numerical null space of a matrix A. Other research investigates sparse matrix algorithms for construction of null space bases that are not orthogonal [Berry et al. 1985], [Coleman and Pothen 1986], [Coleman and Pothen

1987], [Gilbert and Heath 1987], [Heath 1982], [Gill et al. 2005]. If numerical considerations are not included in the basis construction, the resulting basis has the potential to be ill conditioned which may lead to error growth.

The optimization package SNOPT [Gill et al. 2005] uses rook pivoting [Saunders 2006] to construct, for sparse matrices, nonorthogonal null space bases that are usually well conditioned. For some matrices a null space basis may be calculated more quickly and be represented more compactly using an LU based algorithm rather than a QR based algorithm such as that used in our code. However, this is not always the case as illustrated by the matrix Mallya/lhr07c from [Davis and Hu 2011],[Foster and Botev 2009]. In our tests SPQR_COD required 0.5 seconds to construct a null space basis and the implicit representation of the basis required 0.2 MB. For the same matrix the LU based algorithm from LUSOL [Saunders 2006] required 5 seconds and 6.5 MB were used in the implicit representation of the basis. A systematic comparison of LUSOL with our routines is beyond the scope of this research. Also, the LU algorithms based on rook pivoting (or on complete pivoting) can fail to correctly determine the rank for some matrices. A classic example is a triangular matrix with ones on the diagonal and negative ones above the diagonal [Gill et al. 2005, p. 113]. Therefore, tests would be needed in an LU based algorithm to warn the user when the estimated rank may be incorrect [Foster 2007].

There are many algorithms for constructing rank revealing factorizations of dense matrices (see [Foster and Kommu 2006] and its references), but they are not efficient when applied to large, sparse matrices.

3. NUMERICAL EXPERIMENTS

In this section we apply our four methods to a test set of 767 matrices and compare these routines with from SuiteSparseQR, as well as `svd` and the dense `qr` in MATLAB.[1] The MATLAB `svd` is designed for dense matrices and is an implementation of LAPACK's routine DGESVD [Anderson et al. 1999]. We have also compared our codes with `svds` in MATLAB. Although in principle `svds` can be used to find bases for null spaces, we found that it frequently fails to construct an acceptable null space basis and that it is often slow. Thus, `svds` is not discussed further.

3.1 The Test Set

Our test set includes 699 matrices that are collected in the San Jose State University (SJSU) Singular Matrix Database [Foster and Botev 2009] and 68 additional matrices from the University of Florida Sparse Matrix Collection [Davis and Hu 2011]. The matrices in this set arise from applications or have characteristic features of problems from practice. The online databases contain information about specific applications for the matrices and for groups of matrices. The SJSU Singular Matrix Database is a subset of the University of Florida Sparse Matrix Collection with the exception that 40 additional matrices from Regularization Tools [Hansen 1994] are also in the SJSU Singular Matrix Database. The 767 matrices in our test set have

[1]We wish to acknowledge the critical assistance of Nikolay Botev from San Jose State University in developing our database of matrices and its interface. Also, Lars Johnson and Miranda Braselton from San Jose State University made valuable contributions to the analysis and experiments related to routines SPQR_SSP and SPQR_SSI.

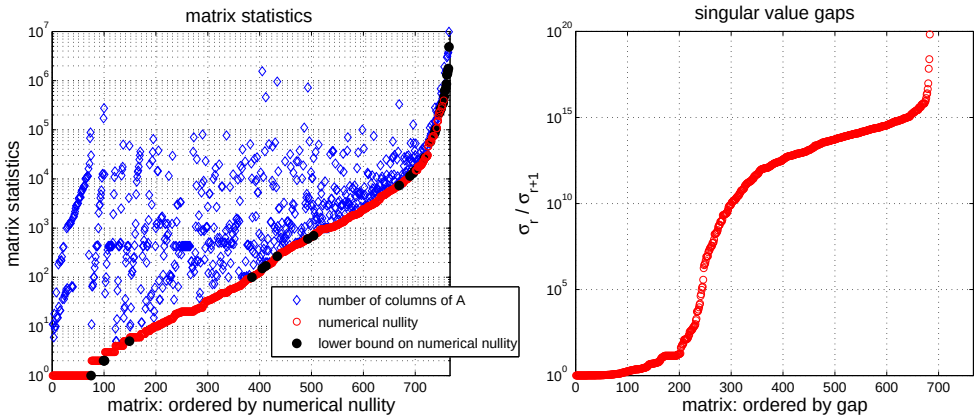

Fig. 1. Properties for matrices on our test set. The left hand plot pictures the numbers of columns and the dimensions of the numerical null spaces. The right hand plot pictures the gaps (σ_r/σ_{r+1}) in the singular value spectrum at the calculated numerical rank, r.

been selected so that each matrix in the set is numerically singular in the sense that, for an $m \times n$ matrix A, the numerical rank of A is smaller than $\min(m, n)$. Some properties of the matrices in the test set are summarized in Figure 1. The tolerance used to define the numerical rank is essentially the same as the default tolerance in the MATLAB `rank` function: $\tau =$ `max(m,n)*eps(norm(A))` for smaller matrices and $\tau =$ `max(m,n)*eps(normest(A,0.01))` for larger matrices.

To determine the true numerical rank for 563 matrices in our test set, singular values were calculated using `svd` in MATLAB and for another 136 matrices the numerical rank was calculated using matrix inertia [Parlett 1980, pp. 46-47] as implemented in SPNRANK [Foster 2009]. The remaining 68 matrices (see `http://www.math.sjsu.edu/singular/matrices/html/additional_matrices.html`) are structurally rank deficient and, therefore, are also numerically rank deficient [Davis 2006, p. 9].

3.2 Accuracy

We test the accuracy of our algorithms by examining the accuracy of the calculated numerical ranks, the quality of the calculated null space bases, the quality of the calculated basic solutions, and the accuracy of the calculated pseudoinverse solutions. In addition to these tests which are discussed in Sections 3.2.1 to 3.2.4 we also wrote testing routines that tested every line (essentially 100% coverage) of our computational code and code that exhaustive tested the option choices in our routines.

For some matrices the error bounds in the algorithms cannot confirm that the numerical rank is correct for our tolerance τ, but the algorithms can confirm that the numerical rank is correct with a larger tolerance. In this case the code returns a warning and returns the value of the larger tolerance.

The computations for these experiments were done on a computer with 32 Gbytes RAM with an Intel Xeon E5404 Quad Processor using 64 bit MATLAB 7.9b. The SuiteSparse Package was used via its MATLAB interface, using `mex` with 64 bit Visual Studio 2005 for the C++ compiler.

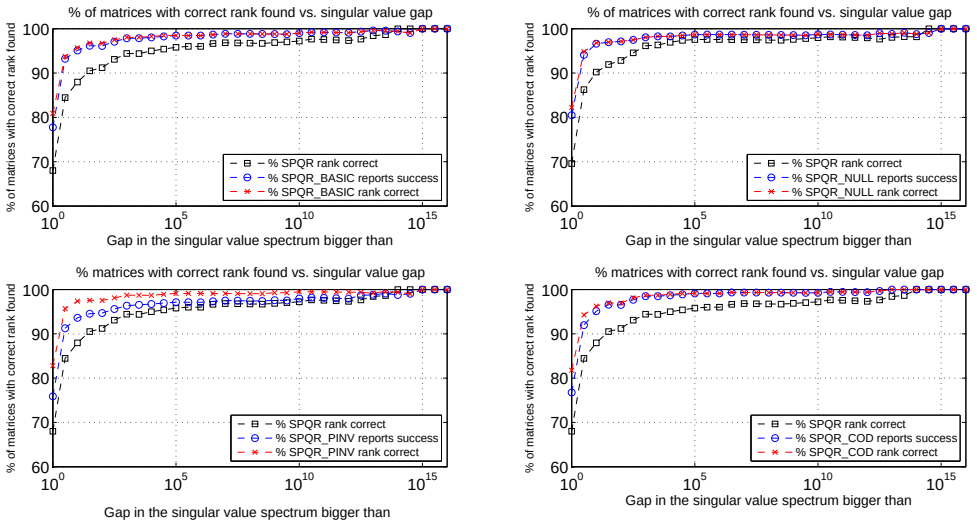

Fig. 2. For each of SPQR_BASIC, SPQR_NULL, SPQR_PINV, SPQR_COD, and for SPQR the plots summarize the percent of matrices where the calculated numerical rank is correct and the percent of the matrices where the routine indicates calculated numerical rank is correct (with the original or a modified tolerance) versus the singular value gap, σ_r/σ_{r+1}, where r is the calculated numerical rank.

Of the 767 matrices tested 729 successfully ran. In 38 cases memory limitations prevented completion for one of our algorithms. The largest matrix for which our code successfully calculated the numerical rank and constructed an implicit representation of an orthonormal basis for the null space was a $321,671 \times 321,671$ matrix with nullity 222,481. The 38 matrices where memory was insufficient had at least 171,395 rows or columns and up to 12,360,060 rows.

3.2.1 *Numerical Rank Calculations.* Figure 2 shows, for the 699 matrices with known numerical ranks, that SPQR calculates the numerical rank correctly for 68% of the matrices and that the other routines do so for more than 80% of the matrices. As the gap in the singular values increases, all the algorithms calculate the numerical rank correctly more frequently. For example, for the 466 matrices with a singular value gap of at least 1000, SPQR determines the correct numerical rank for 95% of matrices and the other routines are correct for 98% to 99% of the matrices. We feel that the inaccuracy of the algorithms for matrices with small gaps in the singular values is not a serious concern since in the small gap case the numerical rank is not well defined.

As seen in Figure 2, the number of cases where the algorithm is successful closely tracks the number of cases where the calculated numerical rank is correct, especially for matrices with a larger singular value gap. For one matrix (Sandia/osci_dcop_33 [Davis and Hu 2011],[Foster and Botev 2009]) SPQR_NULL calculates the wrong numerical rank but reports success. For the other matrices in our experiments, there are no "false positives," that is no cases where the routines report success

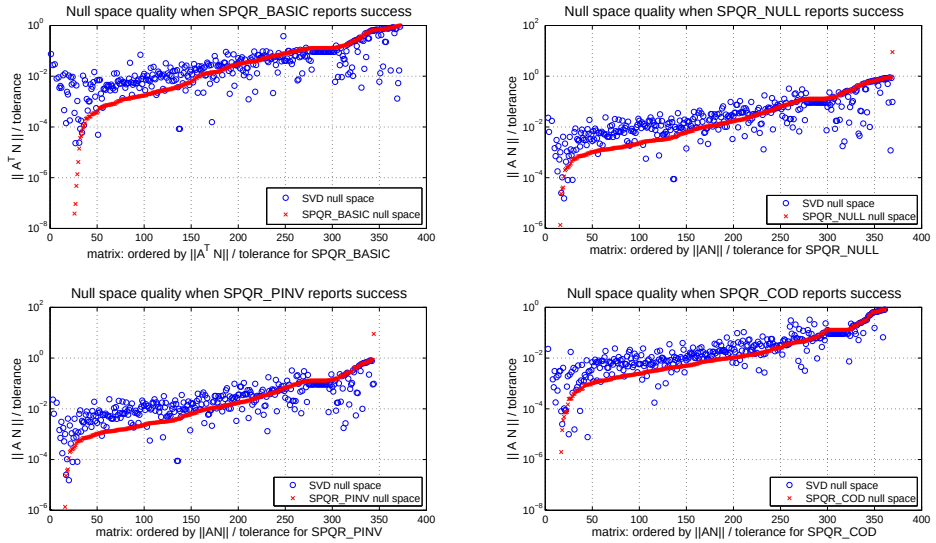

Fig. 3. $||AN||$, where N is a calculated orthonormal basis for the numerical null space, or, in the case of SPQR_BASIC, $||A^T N||$, normalized by the tolerance defining the numerical rank, is plotted for null space bases calculated by `svd` in MATLAB and by our four functions. SPQR_BASIC, SPQR_NULL, SPQR_PINV, and SPQR_COD return a smaller $||AN||$ than `svd` for 58%, 66%, 68%, and 76% of the matrices, respectively.

while the calculated numerical rank is incorrect. The singular values spectrum for Sandia/osci_dcop_33 decays gradually to zero and, for this reason, the potential failure of SPQR_NULL for this matrix is not a significant problem. The routines' indication of success or failure is thus reliable in practice.

3.2.2 *Numerical Null Space Bases.* To judge the quality of the calculated null space bases, note that an orthonormal basis for the numerical null space $\mathcal{X}$ is stored in N. The size of $||AN|| = \max_{x \in \mathcal{X}, x \neq 0} ||Ax||/||x||$ measures how well vectors in $\mathcal{X}$ are annihilated by A and, therefore, is a measure of the quality of the numerical null space. For the 446 $m \times n$ matrices in our test set with $\max(m,n) \leq 5000$, we also calculated null space bases of A and A^T using `svd` in MATLAB. For each of the 446 matrices, we can use the results to compare, relative to the quality of null space basis calculated using `svd` in MATLAB, the quality of the null space basis of A^T, which is an optional output parameter in SPQR_BASIC, and to test the quality of the null space basis for A, which is an output parameter of SPQR_NULL and an optional output parameter for SPQR_PINV and SPQR_COD.

Figure 3 summarizes these calculations. Except potentially for one matrix for SPQR_NULL and one matrix for SPQR_PINV, `svd` in MATLAB and the other routines produce excellent null space bases whenever they report success. The tolerance used to normalize $||AN||$ in Figure 3 is $0(\epsilon||A||)$, where ϵ is relative machine precision. Furthermore, the plots show that the null space bases produced by SPQR_BASIC and SPQR_COD are, for these matrices and choice of

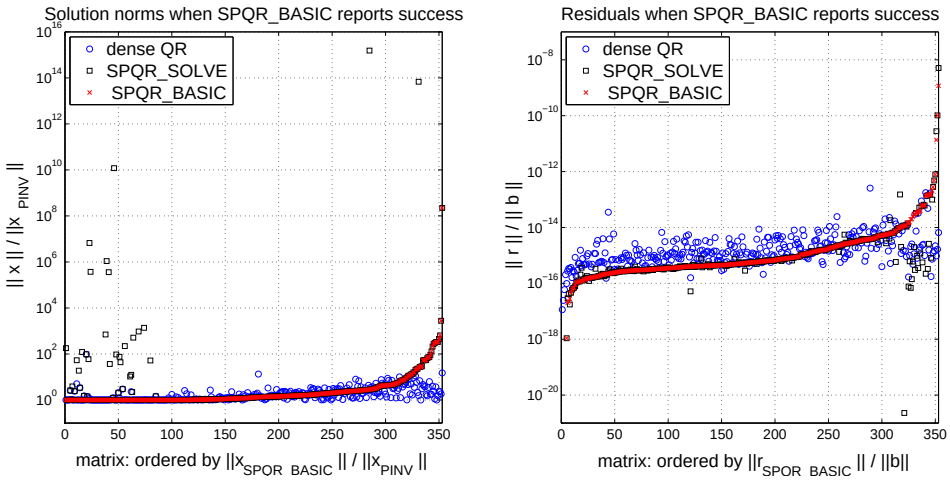

Fig. 4. The left plot pictures $||x||/||x_{PINV}||$ where x is a basic solution to (3) calculated by the dense `qr` in MATLAB, by SPQR_SOLVE, or by SPQR_BASIC and x_{PINV} is computed using `pinv` in MATLAB. In the left hand plot, the vectors b in (3) are random vectors. The right plot pictures $||r||/||b|| = ||b - Ax||/||b||$ for the x vectors calculated using `qr`, SPQR_SOLVE, or SPQR_BASIC. In the right hand plot the vectors b in (3) are of the form $b = Ax$ where x is a random vector. Both plots just show the 353 matrices for which SPQR_BASIC reported success.

tolerance, overall as good as those produced by `svd` in MATLAB. The one matrix for SPQR_PINV and for SPQR_NULL where $||AN||/(\text{tolerance}) = 8.9 > 1$ has $||AN||/||A|| = 4.8 \times 10^{-12}$ which is still small.

3.2.3 *Basic Solutions.* When solving a rank-deficient least squares problem (3), it is often desirable to choose a solution where the residual norm $||r|| = ||b - Ax||$ is small and the solution $||x||$ is not large [Hansen 1998, pp. 90-94]. If A is exactly rank deficient, then the pseudoinverse solution to (3) is an excellent choice since the pseudoinverse solution minimizes $||x||$ from the set of solutions that minimize $||r||$. A basic solution to (3) will not minimize $||x||$ but often still produces an acceptable solution vector. We compare the quality of the calculated basic solutions by looking at the norm of the basic solutions produced by SPQR_BASIC, by SPQR_SOLVE, and by `qr` in MATLAB for dense matrices (calculated using `x = full(A)\b`, if A is not square, or `x = full([A,0*b])\b` otherwise) with the norm of the solution, x_{PINV}, calculated using `pinv` in MATLAB. SPQR_SOLVE is part of SuiteSparseQR and uses SPQR to return a basic solution. So that those calculations that involve dense matrix algorithms are practical, the experiments consist of the 446 matrices in our test set with $\max(m, n) \leq 5000$.

The left hand plot in Figure 4 demonstrates that in most cases SPQR_BASIC produces solutions as good as the solutions produced by the dense `qr` in MATLAB. For some matrices, even when it reports success, SPQR_BASIC can calculate basic solutions to (3) with $||x||$ much larger ($||x_{SPQR\_BASIC}||/||x_{PINV}||$ is as large as 2.2×10^8) than the solution calculated using `pinv`. In these cases, the numerical rank determined by SPQR_BASIC is correct, but the estimated upper bound for $\sigma_r(A)$ is significantly larger than the estimated lower bound for $\sigma_r(A)$ where r is

the estimated numerical rank. These bounds are returned to the user, who may wish to consider use of SPQR_COD rather than SPQR_BASIC. The same plot also demonstrates that for some matrices SPQR_SOLVE can calculate basic solutions to (3) with $||x||$ much larger ($||x_{SPQR\_SOLVE}||/||x_{PINV}||$ is as large as 1.5×10^{15}) than the solution from `pinv` and also much larger than the solution calculated by SPQR_BASIC. In practice such solutions may not be acceptable. SPQR_SOLVE calculates such solutions when its estimate for the numerical rank is incorrect.

The right hand plot in Figure 4 demonstrates that, for consistent systems of equations, usually the residual produced using SPQR_BASIC is excellent and is often smaller than the residual corresponding to a solution calculated using the dense `qr` in MATLAB. However, occasionally in our experiments the residual using SPQR_BASIC or SPQR_SOLVE, although small, is significantly larger than the residual from a dense matrix algorithm. For SPQR_BASIC such cases arise when the calculated solution vector x is large and, as mentioned above, this occurs when the estimated upper bound for $\sigma_r(A)$ is significantly larger than the estimated lower bound for $\sigma_r(A)$.

3.2.4 *Approximate Pseudoinverse Solutions.* The accuracy of the pseudoinverse solution of by SPQR_PINV (x_{SPQR_PINV}) and SPQR_COD (x_{SPQR_COD}) is compared with solutions from `pinv` in MATLAB, (x_{PINV} = `pinv(full(A))*b`). According to the perturbation theory of pseudoinverse solutions [Stewart and Sun 1990, pp. 136-163], in general, we cannot expect that $||x-x_{PINV}||/||x_{PINV}||$ is $O(\epsilon)$ where ϵ is relative machine precision and where x is x_{SPQR_PINV} or x_{SPQR_COD}. To estimate a bound on accuracy we use

$$\frac{||x - x_{PINV}||}{||x_{PINV}||} \lesssim \left(\frac{\sigma_1(A)}{\sigma_r(A)}\right) \max(10\epsilon, ||w||/||A||) \tag{17}$$

where r is the numerical rank of A and $||w||$ (see Theorem 5) is the Frobenius norm of the perturbation in (6). The term 10ϵ in (17) is included since even if $||w||$ is zero there are $O(\epsilon)$ relative errors in storing A. The right hand side in (17) is, approximately, a bound on the first term on the right hand side of equation 5.3 in [Stewart and Sun 1990, p. 157]. We do not include the additional terms from their equation 5.3, since (17) provides a satisfactory description of our experiments.

Figure 5 indicates that SPQR_COD and SPQR_PINV (with one exception) do as good a job in calculating approximate pseudoinverse solutions as expected by the perturbation theory. In the one case where SPQR_PINV is significantly less accurate than predicted by (17), SPQR_BASIC, which is called by SPQR_PINV, produces a solution vector x that is much larger than x_{PINV} and this leads to a value of $||x-x_{PINV}||/||x_{PINV}||$ larger than predicted by (17). In this case the estimates of upper and lower bounds for $\sigma_r(A)$ from SPQR_BASIC are orders of magnitude different. Users of SPQR_PINV can check these bounds and use SPQR_COD when the bounds differ significantly.

3.3 Efficiency

Figure 6 compares the run times of our four methods with the run time of SPQR for the 729 matrices in our data set that ran to completion. The average run time of SPQR_BASIC, SPQR_NULL, SPQR_PINV, and SPQR_COD is 22%, 40%, 136%,

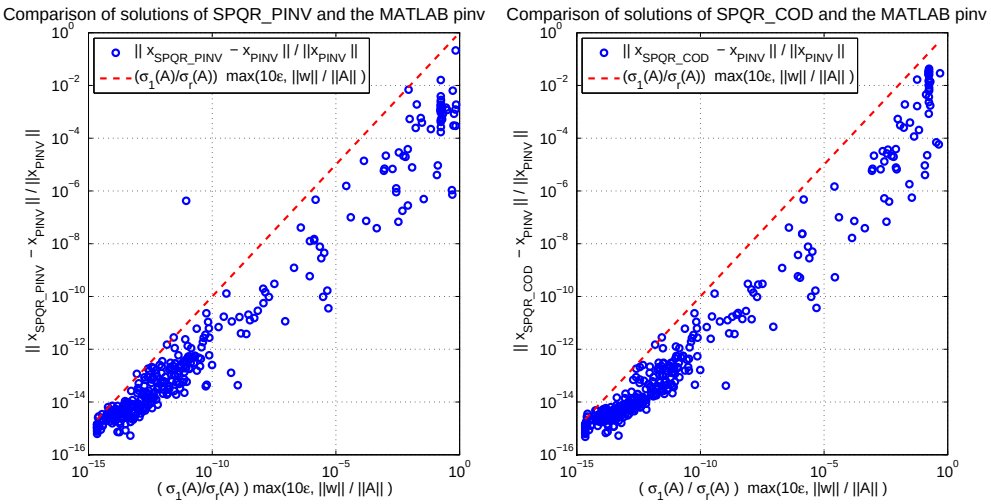

Fig. 5. The left graph plots $||x - x_{PINV}||/||x_{PINV}||$ for x produced by SPQR_PINV for 345 matrices where SPQR_PINV reports success. The vectors b in (3) are random vectors. Also the right hand side of the perturbation theory result (17) is plotted. The right graph is the same plot for x produced by SPQR_COD for 362 matrices where SPQR_COD reports success.

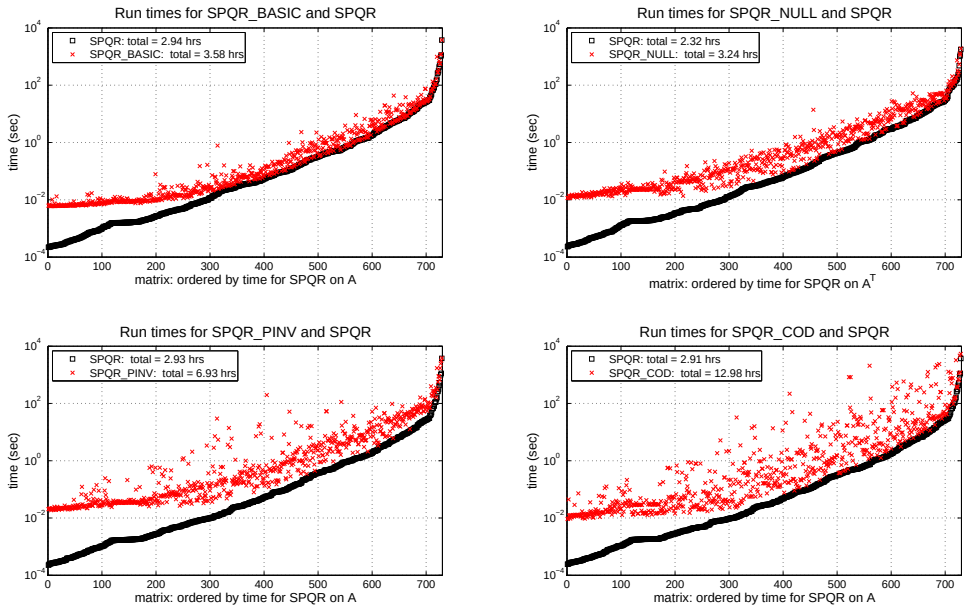

Fig. 6. Run times of our four methods, and SPQR, for 729 matrices.

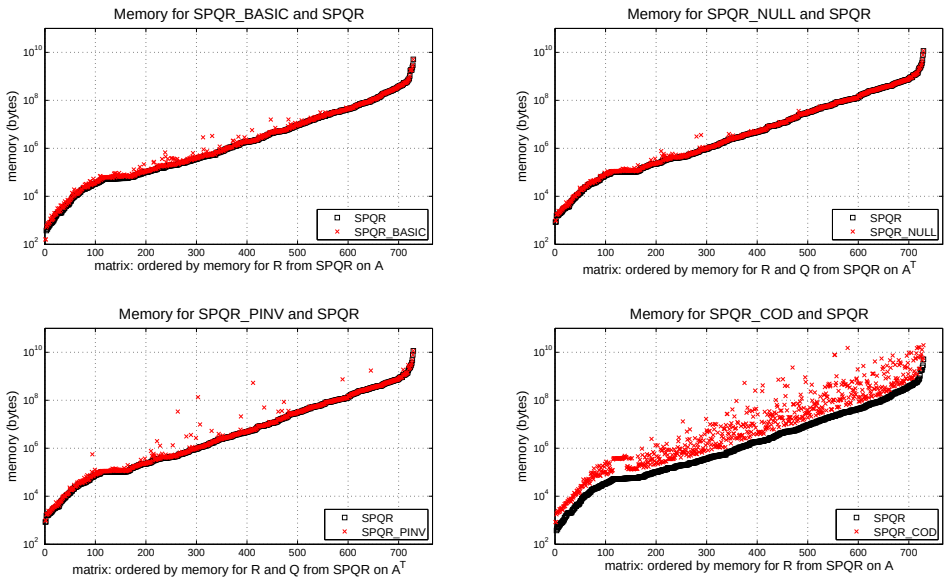

Fig. 7. Approximate memory requirements for our four methods and SPQR.

and 4.5x higher than SPQR, respectively (applied to A or A^T as appropriate). On individual matrices the relative run times for our routines can vary significantly from these averages, especially for smaller matrices.

By extrapolating from the run times of matrices that are 5000 by 5000 or smaller, we can estimate that `svd` in MATLAB would require more than fifteen years to find pseudoinverse solutions for all 729 matrices in this data set, assuming that memory limitations were not an issue. The sum total time of SPQR_COD for these problems is more than 10,000 times smaller.

3.4 Memory use

Figure 7 compares the approximate memory requirement for our four methds with the those of SPQR for the 729 matrices in our data set that ran to completion. For our methods, the memory pictured in the figure includes memory for the matrices listed in the Principal Memory Requirements column of Table I as well as any permutation matrices (stored as vectors) used in the algorithms. The memory pictured for SPQR is described beneath each plot. There is a potential for the routines to require additional memory as part of intermediate calculations. However, for simplicity we focus our discussion on the memory described above.

Figure 7 indicates that, in almost all cases, our four methods, except for SPQR_COD, require little additional memory beyond the memory required by SPQR. On average SPQR_COD requires approximately 7.5 times the amount of memory needed to store R returned by SPQR applied to A. The successive QR factorizations in SPQR_COD compound the fill-in and increases the memory required.

3.5 Challenging Examples

We also tested our code on other challenging test matrices including examples from [Gotsman and Toledo 2008, p. 457] and matrices such as the Kahan matrix [Kahan 1966] from Higham's `gallery` set in MATLAB [Higham 1991; 2002]. With the exception of a single matrix, our methods correctly warn the user of a potential problem when an algorithm did not return the correct numerical rank. For the Sandia/oscii_dcop_33 matrix [Davis and Hu 2011], [Foster and Botev 2009], SPQR_NULL can return an incorrect numerical rank but report success. The accuracy of the error bounds depends on randomly chosen starting vectors in SPQR_SSI and in approximately one in a thousand runs of SPQR_NULL, which calls SPQR_SSI, for this matrix the starting vector choice would lead to an incorrect rank but with no warning, i.e. "false convergence." This matrix has singular values that decay very gradually to zero and therefore the numerical rank is not well defined.

Our singular value bounds are estimated bounds and for this reason the difficulty illustrated with matrix Sandia/oscii_dcop_33 and SPQR_NULL could occur for other matrices. However, based on our experiments, we feel that the probability of the difficulty is exceedingly small. We have never observed it in extensive testing for matrices with a well defined numerical rank. SPQR_SSI converges faster (see (11)) and, in our testing, the risk of false convergence approaches zero as the gap in the singular values increases. A theoretical probabilistic justification of this last comment is beyond the scope of this paper. However, the theory in [Kuczyński and Woźniakowski 1992] (e.g. Theorem 4.1c) is potentially relevant.

4. CONCLUSIONS

We have described a set of algorithms which can be used to calculate the numerical rank, a basic solution to the least squares problem (3), an approximate pseudoinverse solution to this problem, and an orthonormal basis for the numerical null space of a matrix or its transpose. Our implicit, sparse representation allows us to find orthonormal null space bases for matrices with large nullity including examples whose numerical null space dimensions are larger than 100,000 on a computer with 32 Gbytes of RAM. We have estimated that for large matrices the new code can be faster than `svd` in MATLAB by a factor larger than 10,000.

The algorithms were tested on a database of 767 numerically singular matrices, most of which are sparse and which have a wide variety of matrix properties (see Figure 1). The matrices come from real world applications or have characteristic features of real world problems. The algorithms were successful for most of the matrices in the database and the success rate approached 100 percent when the gap in the singular values at the numerical rank was large (see Figure 2). The routines calculate and return estimates of upper and lower bounds for singular values of A and the bounds are used to warn the user if the calculated numerical rank may be incorrect. Our experiments indicate that this warning reliably indicates that the estimated numerical rank is correct for matrices with a well defined numerical rank and, indeed, for almost all the matrices in the entire data set.

For most of the matrices in our tests, when our methods report success our basic solutions (see Figure 4), null space bases (see Figure 3), and approximate

pseudoinverse solutions (see Figure 5) are as good as the corresponding results calculated by a dense matrix routine. In a few cases, when the estimated upper and lower bounds on the smallest non-trivial singular value are orders of magnitude different, the solutions returned by SPQR_BASIC, SPQ_NULL, or SPQR_PINV are inferior to the corresponding results calculated by dense matrix routines, even when our methods report success. If the bounds differ significantly, the user can use SPQR_COD instead, which does not have these difficulties. SPQR_COD can be more accurate but requires more time and memory (see Figures 6 and 7) than the other routines.

Our new routines extend SPQR from the SuiteSparseQR package [Davis 2011] by providing reliable calculation of numerical rank, pseudoinverse solutions, orthonormal bases for numerical null spaces, and basic solutions.

Acknowledgement

We are grateful to the referees and editors whose suggestions led to substantial improvements.

Appendix: Estimating singular values of AN using SPQR_SSP

When one of our four primary routines returns an orthonormal basis N for the null space of A or A^T, they also return estimates of $||AN||$ or $||A^TN||$. The routine SPQR_SSP does this via subspace iteration to calculate estimates of the large singular values and corresponding singular vectors of a matrix A or AN where N is a orthonormal basis for the null space of A as returned by our four primary routines.

Algorithm SPQR_SSP is similar to Algorithm SI in [Vogel and Wade 1994, p. 741] and SISVD in [Berry 1994]. Therefore, we will not present the details of SPQR_SSP except to describe our stopping criteria. Our stopping criteria is based estimates for the accuracy of the calculated singular values and is different from the stopping criteria in [Vogel and Wade 1994, p. 741] and in [Berry 1994].

The error bounds and the stopping criteria in SPQR_SSP are based on the eigenvalue error bound. If $B = AN$, it follows by Theorem 4 that for each $i = 1, \ldots, k$ there exist an eigenvalue α of $C = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}$ such that

$$|s_i - \alpha| \leq ||B^T u_i - v_i s_i||/\sqrt{2} \equiv e_i, \ i = 1, \ldots, k. \tag{18}$$

Here u_i is an estimate of a left singular vector of B and v_i is an estimate of a right singular vector of B. The singular values of B are also eigenvalues of C [Golub and Van Loan 1996, p. 448]. SPQR_SSP uses e_i in (18) as estimates of the errors in using s_i as an approximations for $\sigma_i(A)$. Our stopping criteria is $e_k \leq cs_k$ (where c is a fixed convergence factor) or a maximum number of iterations have been reached. The calculated e_i is only an estimate of a bound in the error in using s_i to approximate $\sigma_i(A)$ because the theory does not insure that α is the i^{th} singular value of A. However, it is usually the case that $|s_i - \sigma_i(A)| \leq e_i$ and, as discussed in Section 3, our methods that use SPQR_SSP work well in practice.

REFERENCES

Anderson, E., Bai, Z., Bischof, S., Blackford, L. S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, S., Hammarling, S., McKenney, A., and Sorensen, D. 1999. *LAPACK users' guide, third edition*. SIAM, Philadelphia, PA.

Berry, M. W. 1994. Computing the sparse singular value decomposition via SVDPACK. In *Recent advances in iterative methods*. IMA Vol. Math. Appl., vol. 60. Springer, New York, 13–29.

Berry, M. W., Heath, M. T., Kaneko, I., Lawo, M., Plemmons, R. J., and Ward, R. C. 1985. An algorithm to compute a sparse basis of the null space. *Numer. Math. 47,* 4, 483–504.

Bischof, C. H. and Quintana-Ortí, G. 1998. Computing rank-revealing QR factorizations of dense matrices. *ACM Trans. Math. Software 24,* 2, 226–253.

Björck, Å. 1996. *Numerical methods for least squares problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Chan, T. F. and Hansen, P. C. 1990. Computing truncated singular value decomposition least squares solutions by rank revealing QR-factorizations. *SIAM J. Sci. Statist. Comput. 11,* 3, 519–530.

Chan, T. F. and Hansen, P. C. 1992. Some applications of the rank revealing QR factorization. *SIAM J. Sci. Statist. Comput. 13,* 3, 727–741.

Coleman, T. F. and Pothen, A. 1986. The null space problem I. complexity. *SIAM Journal on Algebraic and Discrete Methods 7,* 4, 527–537.

Coleman, T. F. and Pothen, A. 1987. The null space problem II. algorithms. *SIAM Journal on Algebraic and Discrete Methods 8,* 4, 544–563.

Davis, T. A. 2006. *Direct methods for sparse linear systems*. Fundamentals of Algorithms, vol. 2. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Davis, T. A. 2011. Algorithm 915, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization. *ACM Trans. Math. Software 38*, Article 8.

Davis, T. A. and Hu, Y. F. 2011. The University of Florida sparse matrix collection. *ACM Trans. Math. Software 38*, Article 1.

Demmel, J. W. 1997. *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Enting, I. 2002. *Inverse Problems in atmospheric constituent transport*. Cambridge University Press, Cambridge.

Fong, D. C. and Saunders, M. A. 2010. LSMR: An iterative algorithm for sparse least-squares problems. Tech. Rep. SOL-2010-2, see http://www.stanford.edu/group/SOL/reports/SOL-2010-2.pdf, Stanford.

Foster, L. V. 1986. Rank and null space calculations using matrix decomposition without column interchanges. *Linear Algebra Appl. 74*, 47–71.

Foster, L. V. 1990. The probability of large diagonal elements in the QR factorization. *SIAM J. Sci. Statist. Comput. 11,* 3, 531–544.

Foster, L. V. 2007. Row echelon form is (usually) accurate after all. International Linear Algebra Society Annual Conference, Shanghai, China, July 17, 2007. See http://www.math.sjsu.edu/~foster/ilas-07-17-2007.pdf.

Foster, L. V. 2009. Calculating the rank of a matrix using SPNRANK. *See http://www.math.sjsu.edu/singular/matrices/software/SJsingular/Doc/spnrank.pdf* .

Foster, L. V. and Botev, N. B. 2009. San Jose State Univeristy Singular Matrix Data Base. *See http://www.math.sjsu.edu/singular/matrices/* .

Foster, L. V. and Kommu, R. 2006. Algorithm 853: an efficient algorithm for solving rank-deficient least squares problems. *ACM Trans. Math. Software 32,* 1 (Mar.).

Gilbert, J. R. and Heath, M. T. 1987. Computing a sparse basis for the null space. *SIAM Journal on Algebraic and Discrete Methods 8,* 3, 446–459.

Gill, P. E., Murray, W., and Saunders, M. A. 2005. SNOPT: an SQP algorithm for large-scale constrained optimization. *SIAM Rev. 47,* 1, 99–131 (electronic).

Golub, G. H. and Van Loan, C. F. 1996. *Matrix computations*, Third ed. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD.

Gotsman, C. and Toledo, S. 2008. On the computation of null spaces of sparse rectangular matrices. *SIAM Journal on Matrix Analysis and Applications 30,* 2, 445–463.

Hansen, P. C. 1994. Regularization tools: a Matlab package for analysis and solution of discrete ill-posed problems. *Numer. Algorithms 6,* 1-2, 1–35.

Hansen, P. C. 1998. *Rank-deficient and discrete ill-posed problems.* SIAM Monographs on Mathematical Modeling and Computation. SIAM, Philadelphia, PA.

Heath, M. T. 1982. Some extensions of an algorithm for sparse linear least squares problems. *SIAM J. Sci. Statist. Comput. 3,* 2, 223–237.

Higham, N. J. 1991. Algorithm 694: a collection of test matrices in MATLAB. *ACM. Trans. Math. Softw. 17,* 3, 289–305.

Higham, N. J. 2002. *Accuracy and stability of numerical algorithms*, Second ed. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Kahan, W. 1966. Numerical linear algebra. *Canad. Math. Bull. 9*, 757–801.

Kuczyński, J. and Woźniakowski, H. 1992. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. *SIAM J. Matrix Anal. Appl. 13,* 4, 1094–1122.

Lehoucq, R. B., Sorensen, D. C., and Yang, C. 1998. *ARPACK users' guide.* Software, Environments, and Tools, vol. 6. SIAM, Philadelphia, PA.

Li, T. Y. and Zeng, Z. 2005. A rank-revealing method with updating, downdating and applications. *SIAM J. Matrix Anal. Appl. 26,* 4, 918–946.

Paige, C. C. and Saunders, M. A. 1982. LSQR: an algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software 8,* 1, 43–71.

Parlett, B. N. 1980. *The symmetric eigenvalue problem.* Prentice-Hall Inc., Englewood Cliffs, N.J. Prentice-Hall Series in Computational Mathematics.

Pierce, D. J. and Lewis, J. G. 1997. Sparse multifrontal rank revealing QR factorization. *SIAM Journal on Matrix Analysis and Applications 18,* 1, 159–180.

Saunders, M. 2006. LUSOL: A basis package for constrained optimization. Tech. rep., SCCM, Stanford University. February. See http://www.stanford.edu/group/SOL/talks/saunders-LUSOL-linopt2006.pdf.

Stewart, G. W. 1981. On the implicit deflation of nearly singular systems of linear equations. *SIAM J. Sci. Statist. Comput. 2,* 2, 136–140.

Stewart, G. W. and Sun, J. G. 1990. *Matrix perturbation theory.* Computer Science and Scientific Computing. Academic Press Inc., Boston, MA.

Vogel, C. R. and Wade, J. G. 1994. Iterative SVD-based methods for ill-posed problems. *SIAM J. Sci. Comput. 15,* 3, 736–754.

Received xxxx; xxxx; accepted xxxx