

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 and also can 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 (T.A. Davis, “Algorithm 9xx, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization,” to appear ACM TOMS, 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 a warning flag indicates that the numerical rank should be correct. Reliable determination of numerical rank is critical to the other calculations in the package. The routines can work well for matrices with large, greater than 100,000, as well as small nullity.

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);

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20xx ACM 0098-3500/20xx/1200-0001 \$5.00

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

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

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

$$\|AN\| \leq \tau. \quad (2)$$

If we wish to solve the least squares problem

$$\min_x \|Ax - b\| \quad (3)$$

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

$$\min_x \|\hat{A}x - b\|. \quad (4)$$

where \hat{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, which is described below. 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 new routines calculate estimates of upper and lower bounds for singular values of A and use these estimates to return an error flag that warns the user when the calculated numerical rank may be incorrect. The algorithms reliably determine the numerical rank in the sense that, based on extensive testing with matrices from applications with a common choice for τ , the numerical rank is almost always accurately determined when the warning flag is 0, indicating that the numerical rank should be correct. In our experiments for one matrix the warning flag was, at times, zero while the numerical rank returned was incorrect. This matrix has singular values that decay gradually to zero and is discussed further in Section 3.5. Reliable determination of numerical rank is often critical to our other calculations.

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, $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(m,n)}$, 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 = the last $n - r$ columns of V and \mathcal{X} = the span of the

columns of X . It follows from the SVD definition, $A = UDV^T$, that

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

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. Therefore it is of interest to explore alternatives to the SVD for determining the numerical rank, constructing an orthonormal basis for a numerical null space of A and the approximate pseudoinverse solution to (3).

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 represent an orthogonal matrix, Q , in sparse format using Householder transformations and the ability to estimate the numerical rank of A . The sparse representation of Q will lead to a sparse representation for the orthonormal basis of the numerical null space. This makes it practical to determine such bases for null spaces of high dimension – we are successful in determining orthonormal bases for null spaces of dimension larger than 100,000.

The estimate of the numerical rank returned by SPQR is based on an idea due to Heath [Heath 1982] and is usually, but not always, accurate [Foster 1990]. We will 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 will be 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).

We should note that 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 in the case that there is a significant gap in the singular values so that $\sigma_r \gg \sigma_{r+1}$, τ lies between σ_{r+1} and σ_r and 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 [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 1998b, p. 227] we will use the term “well defined numerical rank” for matrices with a well-determined gap in the singular values. We will see that our routines often work better for matrices with a well defined numerical rank.

In the following section we discuss SPQR and our algorithms including the routine SPQR_BASIC which determines a basic solution to (3), the routine SPQR_NULL which constructs an orthonormal basis for the numerical null space of A , the rou-

tine SPQR_PINV which constructs an approximate pseudoinverse solution to (3) and the routine SPQR_COD which uses the complete orthogonal decomposition, which we define below, to construct the approximate pseudoinverse solution to (3) as well as orthonormal bases for the numerical null spaces of A and A^T . As is discussed in Subsection 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 will be used to determine the estimated bounds and we briefly mention these results here.

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

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

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

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 $k = 1, 2, \dots, \min(m, n)$,*

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

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 $|\lambda_i - \lambda| \leq \|Av - \lambda v\|$.*

The first three theorems will be used to provide rigorous error bounds for singular value locations. The last theorem will be used to estimate error bounds on certain singular values.

Our routines SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD are based on the routine SPQR [Davis 2011] and also involve subspace iteration applied to the inverse of certain matrices (routine SPQR_SSI) to estimate small singular values. In this section we will first discuss SPQR and SPQR_SSI and then the four routines SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD. A routine SPQR_SSP used by the last four routines to estimate matrix norms is discussed in Appendix B. Also in Subsection 2.7 we summarize the key differences in our algorithms (see Table I). In Subsection 2.8 we will briefly discuss 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 sparse QR factorization based on the multifrontal method. The SPQR routine returns an estimate of the numerical rank ℓ of A based on the idea of Heath [Heath 1982]. SPQR constructs a decomposition

$$AP_1 = Q_1R + E_1 = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix} + E_1 = Q_1 \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} + E_1 \quad (6)$$

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 a $\ell \times n$ right trapezoidal matrix, the bottom block row of zeros below R_1 is $(m - \ell) \times n$, R_{11} is a $\ell \times \ell$ triangular matrix whose diagonal entries are larger in magnitude than a user chosen tolerance τ and R_{12} is $\ell \times (n - \ell)$. SPQR includes the option, which we will choose in our use of SPQR, to represent the orthogonal matrix Q in the QR factorization in a sparse format by storing the elementary Householder transformations that form Q . Note that 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 $r < \ell$. This possibility is discussed further in the next section.

Given a user defined tolerance τ the technique of Heath 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 can be permuted to the right of the current partly triangularized matrix and the new i^{th} column is processed with Householder transformations. In practice it is not necessary to physically permute the original i^{th} column to the right at this stage of the computations, rather one can simply label the column and move it after the triangularization is complete. We should add that although Heath originally described the procedure in terms of zeroing out rows of the current R matrix using Givens transformations, the above description is equivalent [Barlow and Vemulapati 1992, p. 1281].

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*

$$\|E_1\|_F = \|w\| \quad \text{and} \quad (7)$$

$$\|E_1\| \leq \|w\| \leq \sqrt{n - \ell} \tau \quad (8)$$

where $\| \cdot \|$ indicates the usual two norm and $\| \cdot \|_F$ indicates the Frobenius norm.

PROOF. Assume that we apply the orthogonal transformations Q_1 and P_1 in (6) to A without dropping any diagonal entries in the factorization and define the unperturbed factorization:

$$\tilde{R} = Q_1^T AP_1. \quad (9)$$

We will compare \tilde{R} in the unperturbed factorization (9) with R in (6). Suppose in the SPQR algorithm for some i , $r_{ii} \leq \tau$. As described above we can move the

i^{th} column of R to the right and set r_{ii} to zero so that the entire column on or below element i will be zero. As the SPQR algorithm proceeds the subsequent Householder transformations will act only on (a subset of) rows i to m of this column. Therefore in the perturbed matrix, with r_{ii} set to zero, the column will remain zero on or below element i . When the orthogonal transformations used in SPQR 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} . The subsequent orthogonal Householder transforms will 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 \tilde{R} will be $|r_{ii}|$. The other columns of R and \tilde{R} will be 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. \square

Due to equation (6) and the singular value perturbation bounds, Theorem 1, the singular values of A and of R will differ by at most $\|E_1\|$. Theorem 5 implies $\sqrt{(n - \ell)\tau}$ is an a priori bound on this difference and $\|w\|$ is an a posteriori bound. We should note that the a posteriori bound is, in some cases, much smaller than the a priori bound. In the current version of SPQR [Davis 2011], the value of $\|w\|$ is returned to the caller.

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) will have a low dimensional numerical null space. Let R represent any $n \times n$ (where n perhaps is ℓ) 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 an R and the corresponding singular vectors using an appropriate iterative method applied to, for example, $R^{-1}R^{-T}$. 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:

Data:

- R : a $n \times n$ nonsingular, triangular matrix
- τ : the tolerance defining the numerical rank of R
- other options described in the code including `init_block_size`, the initial block size, and `block_size_increment`, the increment for the block size

Result:

- U : an $n \times k$ matrix containing estimates of the left singular vectors of R corresponding to estimated singular values in S . When stats.flag (see below) is zero then $r = n - k + 1$ is the estimated numerical rank of R . Also $U(:,2:k)$ is an orthonormal basis for the numerical null space of R^T .
- S : a $k \times k$ diagonal matrix whose diagonal entries $s_1 \geq s_2 \geq \dots \geq s_k$ are the estimated smallest k singular values of R , with k as described above.
- V : an $n \times k$ matrix containing estimates of the right singular vectors of R corresponding to estimated singular values in S . When stats.flag is 0, $V(:,2:k)$ is an orthonormal basis for the numerical null space of R .
- stats: a structure containing additional information including the estimated numerical rank of R , $e_j, j = 1, \dots, k$ or, optionally, for just $j = 1$, which are estimates for bounds on the errors in s_j (see Appendix A) and flag. Flag = 0 indicates that the numerical rank r is correct. Also when flag is zero, $s_1 > \tau$ and $s_2 \leq \tau$.

Initialize:

- U = a random $n \times (\text{init_block_size})$ matrix with orthonormal columns
- b = the current block size = init_block_size

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 $n \times b$, D_1 is a $b \times b$ diagonal matrix, and X_1 is $b \times b$
- Solve the triangular system $R^T U_1 = V$ to calculate $U_1 = R^{-T}V$.
- determine the compact SVD of U_1 : $UD_2X_2^T = U_1$, where U is $n \times b$, D_2 is a $b \times b$ diagonal matrix, and X_2 is $b \times b$
- let $\hat{s}_i, i = 1, 2, \dots, b$ be diagonal entries of D_2
- if** $\hat{s}_b \geq 1/\tau$ **then**
 - Y = orthonormal basis for a block_size_increment dimensional subspace orthogonal to the column space of U
 - redefine $U = [U, Y]$
- end**

until *until stopping criteria is met, see Appendix A* ;

if *stopping criteria for success is met* **then**

- let k be the smallest i with $\hat{s}_i < 1/\tau$ and let $r = n - k + 1$
- Redefine V : let $V = VX_2$ (see Appendix A) and reorder, in reverse order, columns 1 to k of V
- Redefine U : reorder, in reverse order, columns 1 to k of U
- $s_j = 1/\hat{s}_{n-r-j+2}, j = 1, 2, \dots, n - r + 1$
- estimate e_j using Theorem 4 as outlined in Appendix A
- Set flag = 0

else

- Set flag > 0

end

When the matrix R is a large matrix it is often the case the solutions to the triangular systems are the dominant computations in Algorithm SPQR_SSI. In most case 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 not large. At each iteration the singular value decompositions can be done in $O(nb^2)$ operations. Therefore, in most cases, the work for the SVD calculations is not large. The extra work for increasing the size of U also requires $O(nb^2)$ operations per step, when it is necessary.

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 SI or SPIT dynamically increase the block size and the stopping criteria in these algorithms differ from our stopping criteria. Also SI is configured to use one singular value decomposition per iteration. Our use of two SVD's can be more expensive but has the advantage that the normalization implicit in each SVD reduces the risk of overflow. Finally we use inverse iteration (applying the iterations to $R^{-1}R^{-T}$); as does SPIT whereas Algorithm SI as presented in [Vogel and Wade 1994] is applied to RR^T .

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 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 , $j = 1, \dots, n$ 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 $j = r, r+1, \dots, n$. If $\text{flag} = 0$ the algorithm terminates with $b > n - r$, which we will assume. In addition we will assume the $\sigma_{n-b} > \sigma_{n-b+1}$, or that these two singular values are distinct. Let p be the number of iterations in the algorithm. The relative errors for the singular value approximations obtained by Algorithm SPQR_SSI converge to zero at the asymptotic rate

$$\frac{\sigma_j - s_{j-r+1}}{\sigma_j} = O\left(\left(\frac{\sigma_j}{\sigma_{n-b}}\right)^{4p}\right), \quad j = r, r+1, \dots, n. \quad (10)$$

Note that $\frac{\sigma_j}{\sigma_{n-b}} \leq \frac{\sigma_{r+1}}{\sigma_r}$ for $j = r+1, \dots, n$. It follows that in the case that the numerical rank is well defined in the sense that $\sigma_{r+1} \ll \sigma_r$, convergence will be rapid to singular values $\sigma_{r+1}, \sigma_{r+2}, \dots, \sigma_n$.

2.3 Basic Solution to a Rank Deficient Least Squares Problem

The routine SPQR_BASIC determines a basic solution to (3). As mentioned earlier 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 SPQR. To determine a basic solution we will initially apply SPQR to A and use routine SPQR_SSI to determine if the estimated numerical rank returned by SPQR is correct. If the estimated rank returned by SPQR is not correct we can often determine the correct numerical rank and still construct a satisfactory basic solution to (3).

Input to the routine includes A and b in (3), a tolerance τ defining the numerical rank and user selectable parameters with default values supplied by the routine. The routine returns the basic solution x and a structure stats that contains the calculated numerical rank, estimates of upper and lower bounds for singular values

of A , an error flag, which is zero when the algorithm succeeds, and other information. Optionally, SPQR_BASIC can return an orthonormal basis for the null space of A^T . The steps of the SPQR_BASIC algorithm are:

- (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_{11} is $\ell \times \ell$.
- (2) Apply SPQR_SSI to R_{11} , with tolerance set to τ . Let r be the numerical rank returned by SPQR_SSI.
 - If $r = \ell$, let $c = Q_1^T b$, \hat{c} = the first ℓ components of c , \hat{z} be the solution to $R_{11}\hat{z} = \hat{c}$, $z = [\hat{z}; 0]$ (using MATLAB notation) where 0 represents a vector of zeros so that z has n components and finally let $x = P_1 z$
 - If $r < \ell$, do the same constructions as in the case that $r = \ell$ except let \hat{z} be the solution to $R_{11}\hat{z} = \hat{c}$ that is produced by deflation, [Chan 1984],[Chan and Hansen 1990],[Stewart 1981] as described in Appendix C.
- (3) To check that the calculated numerical rank, r , is correct we first estimate upper and lower bounds for singular values of A .
 - Lower bounds: 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} . It follows from this fact and the singular value interlace property, Theorem 2, that $\sigma_i(R_{11}) \leq \sigma_i(A)$, for $i = 1, 2, \dots, \ell$. Algorithm SPQR_SSI applied to the $\ell \times \ell$ matrix R_{11} returns $s_j, j = 1, 2, \dots, k$ with $k = \ell - r + 1$, which are estimates for singular values, $\sigma_i(R_{11}), i = r, r+1, \dots, \ell$, respectively. Furthermore SPQR_SSI returns $e_j, j = 1, 2, \dots, k$ which are estimated error bounds for the accuracy of $s_j, j = 1, 2, \dots, k$. Therefore, $s_j - e_j, j = 1, 2, \dots, k$ are estimated lower bounds for $\sigma_i(R_{11}), i = r, r+1, \dots, \ell$, respectively, and also for $\sigma_i(A), i = r, r+1, \dots, \ell$, respectively. For $i = \ell + 1, \ell + 2, \dots, \min(m, n)$ we use 0 for a lower bounds for $\sigma_i(A)$.
 - Upper bounds: Algorithm SPQR_SSI applied to R_{11} returns approximate left singular vectors of R_{11} in the $\ell \times k$ matrix U . It follows from the singular value minimax property, Theorem 3, that the singular values $1, 2, \dots, k$ of $U^T R_1$ provide upper bounds for singular values $r, r+1, \dots, \ell$ of R_1 , respectively. It follows from (6), Theorem 5 and the singular value perturbation bound, Theorem 1, that $\|w\|$ plus the singular values $1, 2, \dots, k$ of $U^T R_1$, provide upper bounds for singular values $r, r+1, \dots, \ell$ of A , respectively. By the singular value perturbation bounds, Theorem 1, for $i = \ell + 1, \ell + 2, \dots, \min(m, n)$ it follows that $\|w\|$ provides an upper bound for singular values $\sigma_i(A)$. If the option is selected to calculate an orthonormal basis for the numerical null space of A^T the upper bounds can, in some cases, be improved by using techniques similar to those described in step 3 of algorithm SPQR_NULL.
 - Checking the numerical rank: If the estimated lower bound for $\sigma_r(A) > \tau$ and if the upper bound for $\sigma_{r+1}(A) \leq \tau$ then we set flag to zero. If the estimated lower bound for $\sigma_r(A)$ is greater than the estimated upper bound for $\sigma_{r+1}(A) > \tau$ then we set flag to one 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. If these

conditions are not met then the routine returns flag > 1 .

Algorithm SPQR_BASIC

As mentioned earlier, often the numerical rank determined by SPQR is correct or is close to the true numerical rank. In such cases, especially if the numerical rank is well defined, algorithm SPQR_SSI will converge quickly. The additional calculations in steps 2 and 3 of the algorithm also are usually done quickly in this case. Occasionally the numerical rank determined by SPQR is substantially different than the true numerical rank and, in this case, algorithm SPQR_BASIC can be significantly slower than SPQR. To address this case we set the default maximum block size in algorithm SPQR_SSI to 10. If this block size is exceeded a flag larger than one is returned. Similarly there is a default maximum number of iterations set in SPQR_SSI. The user can override the default values.

We should note that it is not necessary to save the orthogonal matrix Q_1 to use SPQR_BASIC. SPQR has a feature to apply Q_1 to a b without saving Q_1 . For this reason, often the memory required to save R_1 is the largest memory requirement in SPQR_BASIC. If the option is selected to calculate an orthonormal basis for the numerical null space of A^T then it is necessary to save Q_1 . See Section 2.4 for more details about construction of a numerical null space basis.

The algorithm can, at times, fail to return the warning flag equal to zero and in this case the numerical rank returned by SPQR_BASIC may be incorrect. As will be explored in Section 3 this most frequently occurs when the tolerance defining the numerical rank does not correspond to a significant gap in the singular values. As illustrated in Section 3.2.3 even when the flag is zero that algorithm can still produce a poor solution vector x . For example $\|x\|$ returned by the algorithm can be much larger than necessary if the estimated lower bound and the estimated upper bound for $\sigma_r(A)$ are significantly different. If either of these issues appear to be a problem when using SPQR_BASIC one may choose to use algorithm SPQR_COD. SPQR_COD is usually more expensive but also more accurate than SPQR_BASIC.

We should also note that when the computations are done in finite precision arithmetic the numerical rank returned by SPQR_BASIC will correspond to exact calculation applied to $A + E$ where E is $O(\epsilon\|A\|)$ and ϵ relative machine precision. For this reason the tolerance used to define the numerical rank should be larger than $\epsilon\|A\|$. The default choice in our code is $\max(m, n)$ eps (an estimate of $\|A\|$) where $\text{eps}(x)$, as calculated by MATLAB, is the spacing in the floating point number system near x . This choice is essentially the same as the choice in MATLAB's RANK command and it worked well in the numerical experiments described in Section 3. Also we should note that calculation of the parameter flag in SPQR_BASIC is based on estimates of bounds on the singular values. Due to the use of estimates, the algorithm does not guarantee that the calculated numerical rank is correct when flag is 0. However the numerical experiments in Section 3 indicate that in practice the warning flag does provide a reliable indicator that the numerical rank is correct, in almost all cases.

2.4 Null Space Calculation using SPQR_NULL

The goal of the routine SPQR_NULL is to calculate an orthonormal basis, N , for the numerical null space of the $m \times n$ matrix A for tolerance τ . Input to the rou-

tine includes A , the tolerance τ and user selectable parameters with default values supplied by the routine. The routine returns an orthonormal basis, N , for the numerical null space. The routine has an option to return the basis either as an explicit $n \times n - r$ matrix or store N implicitly as a product of sparse matrices (see below). When the nullity $n - r$ is large the implicit form will require less memory. The routine also returns a structure stats that contains the calculated numerical rank, estimates of upper and lower bounds for singular values of A , an error flag, which is zero when the algorithm succeeds, and other information. In our code SPQR_NULL is implemented using a call to SPQR_BASIC. However for presentation purposes we describe it separately:

- (1) Apply SPQR, with tolerance set to τ , to A^T , producing Q_1 , P_1 , R_1 , R_{11} and R_{12} in

$$A^T P_1 = Q_1 R + E_1 = Q_1 \begin{pmatrix} R_1 \\ 0 \end{pmatrix} + E_1 = Q_1 \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} + E_1. \quad (11)$$

and $\|w\|$ where w is the vector of perturbations described in Theorem 5. If ℓ is the estimated numerical rank returned by SPQR then P_1 is an $m \times m$ permutation matrix, Q_1 is an $n \times n$ orthogonal matrix, R is a $n \times m$ right triangular (for $n \geq m$) or trapezoidal (for $n < m$) matrix, E_1 is an $n \times m$ error matrix, R_1 is an $\ell \times n$, R_{11} is an $\ell \times \ell$ triangular matrix and R_{12} is $\ell \times n - \ell$ (when $n > \ell$).

- (2) Apply SPQR_SSI to R_{11} , with tolerance set to τ . Let r be the numerical rank returned by SPQR_SSI.
 - If $r = \ell$, let $N = Q_1 * [0; I]$ (using MATLAB notation) where 0 represents an $\ell \times n - \ell$ matrix of zeros and I is an $n - \ell$ by $n - \ell$ identity matrix.
 - If $r < \ell$, let $k = \ell - r + 1$ and let U be the $\ell \times k$ matrix of approximate left singular vectors returned by SPQR_SSI. Then $N = Q_1 * [U(:, 2:k), 0; 0, I]$ (using MATLAB notation) where the first 0 represents an $\ell \times n - \ell$ matrix of zeros, the second 0 represents an $n - \ell \times \ell - r$ matrix of zeros and I is an $n - \ell$ by $n - \ell$ identity matrix.
- (3) The routine checks that the calculated numerical rank appears to be correct by estimating upper and lower bounds for singular values of A .
 - Lower bounds: SPQR_NULL applies SPQR to A^T . Since the singular values of A and A^T are the same, the lower bounds for singular values of A can be calculated using the procedure in algorithm SPQR_BASIC
 - Upper bounds: Since the singular values of A and A^T are the same the upper bounds on the singular values of A can be calculated using the procedure in algorithm SPQR_BASIC. These bounds can be improved, in some cases, by making use of the calculated null space. By the singular value minimax property, Theorem 3, the $n - r$ singular values of AN are upper bounds for the smallest $n - r$ singular values of A . Algorithm SPQR_SSP discussed in Appendix B calculates estimates of singular values of AN and, as well, estimated error bounds on these estimates. The sum of the singular value estimates and the error bound estimates provide estimates of upper bounds on the singular values of A . The upper bounds returned by algorithm SPQR_NULL are chosen as the

minimum of the upper bounds determined using the technique of algorithm SPQR_BASIC and by estimating singular values of AN .

- Checking the numerical rank: If the estimated lower bound for $\sigma_r(A) > \tau$ and if the upper bound for $\sigma_{r+1}(A) \leq \tau$ then we set flag to zero. The calculated numerical rank appears to be correct. If the conditions for flag = 0 are not met we set the flag to one or more as described in algorithm SPQR_BASIC.

Algorithm SPQR_NULL

Our comments about the efficiency of algorithm SPQR_BASIC also apply to SPQR_NULL. For many matrices, especially for matrices that are large and have a well defined numerical rank, the additional work required by SPQR_NULL, beyond the work required for SPQR, is not large. SPQR_NULL requires storage of Q_1 and therefore can require more memory than SPQR_BASIC. However 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

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

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 with our package of code to multiply a null space basis, represented in this implicit mode, by another matrix and, as well, the routine SPQR_EXPLICIT_BASIS will convert a null space basis stored in implicit form to an explicit matrix. Due to the implicit storage mode it is practical to calculate and represent null spaces of matrices with high dimensional null spaces. For example we have calculated the null space of a 321671 by 321671 matrix whose null space has dimension 222481.

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. Input to the routine includes A and b in (3), the tolerance τ and user selectable parameters with default values supplied by the routine. The routine returns x , the approximate pseudoinverse solution to (3). The routine also returns a structure stats that contains the calculated numerical rank, estimates of upper and lower bounds on certain singular values of A , an error flag, which is zero when the algorithm succeeds, and other information. Optionally SPQR_PINV can return an orthonormal basis N , for the numerical null space of A for tolerance τ and NT , an orthonormal basis for the numerical null space of the A^T for tolerance τ . The steps of the SPQR_PINV algorithm are:

- (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) Return an error flag of 0 if the error flags of both SPQR_BASIC and SPQR_NULL are zero. 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.
- (5) Optionally, if the user requests, return N , an orthonormal basis for the numerical null space of A calculated by SPQR_NULL, and NT , an orthonormal basis for the numerical null space of A^T calculated by SPQR_BASIC.

Algorithm SPQR_PINV

2.6 Calculations using a Complete Orthogonal Decomposition and SPQR_COD

The routines SPQR_BASIC, SPQR_NULL and SPQR_PINV often work well in practice but occasionally they fail, returning a nonzero flag or returning 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 decomposition

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

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 [Golub and Van Loan 1996, p. 256], [Björck 1996, pp. 110-111] to determine the rank of A , the fundamental subspaces of A and the pseudoinverse solution to (4). The approximate COD can determine the numerical rank, numerical fundamental subspaces and an approximate pseudoinverse solution to (3).

The routine SPQR_COD receives the same information as SPQR_PINV and also returns the same information. The steps in SPQR_COD are:

- (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 a $\ell \times \ell$ permutation matrix \hat{P}_2 , an $n \times n$ orthogonal matrix Q_2 and a $\ell \times \ell$ right triangular matrix T such that

$$R_1^T \hat{P}_2 = Q_2 \begin{pmatrix} T \\ 0 \end{pmatrix}. \quad (14)$$

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

(14) that

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

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 . Let r be the numerical rank returned by SPQR_SSI.
 - If $r = \ell$, let $c = P_2^T Q_1^T b$, \hat{c} = the first ℓ components of c , \hat{z} be the solution to $T^T \hat{z} = \hat{c}$, $z = [\hat{z}; 0]$ (using MATLAB notation) where 0 represents a vector of zeros so that z has n components and finally let $x = P_1 Q_2 z$
 - If $r < \ell$, do the same constructions as in the case that $r = \ell$ except let \hat{z} be the solution to $T^T \hat{z} = \hat{c}$ that is produced by deflation, [Chan 1984],[Chan and Hansen 1990],[Stewart 1981] as described in Appendix C.
- (4) To check that the calculated numerical rank, r , is correct we estimate upper and lower bounds for singular values of A .
 - Lower bounds: 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 $i = 1, \dots, \ell$. Algorithm SPQR_SSI applied to the $\ell \times \ell$ matrix T returns $s_j, j = 1, 2, \dots, k$ with $k = \ell - r + 1$, which are estimates for singular values, $\sigma_i(T), i = r, r + 1, \dots, \ell$, respectively. Furthermore SPQR_SSI returns $e_j, j = 1, 2, \dots, k$ which are estimated error bounds for the accuracy of $s_j, j = 1, 2, \dots, k$. Therefore, $s_j - e_j, j = 1, 2, \dots, k$ are estimated lower bounds for $\sigma_i(T), i = r, r + 1, \dots, \ell$, respectively. It follows that $s_j - e_j - \|w\|, j = 1, 2, \dots, k$ are estimated lower bounds for $\sigma_i(A), i = r, r + 1, \dots, \ell$, respectively. For $i = \ell + 1, \ell + 2, \dots, \min(m, n)$ we use 0 for a lower bounds for $\sigma_i(A)$.
 - Upper bounds: Using the same arguments as for the lower bounds it follows that $s_j + e_j + \|w\|, j = 1, 2, \dots, k$ are estimated upper bounds for $\sigma_i(A), i = r, r + 1, \dots, \ell$, respectively. By the singular value perturbation bounds, Theorem 1, for $i = \ell + 1, \ell + 2, \dots, \min(m, n)$ it follows that $\|w\|$ provides upper bounds for singular values $\sigma_i(A)$. As discussed in Section 2.4 in some cases an orthonormal basis N for the numerical null space can be used to improve these upper bounds.
 - Checking the numerical rank: If the estimated lower bound for $\sigma_r(A) > \tau$ and if the upper bound for $\sigma_{r+1}(A) \leq \tau$ then we set flag to zero. The calculated numerical rank appears to be correct. If the conditions for flag = 0 are not met we set the flag to one or more as described in algorithm SPQR_BASIC.
- (5) Calculate and return, if requested, orthonormal bases for the numerical null space of A , using P_1, Q_2 and U returned by routine SPQR_SSI, and A^T , using P_2, Q_1 and V returned by routine SPQR_SSI, in a manner similar to that described in Section 2.4. For example, when $r < \ell$ the nullspace basis for A is $N = P_1 * Q_2 * [U(:, 2 : k), 0; 0, I]$, using notation similar to that in step (2) of Algorithm SPQR_NULL.

Algorithm SPQR_COD

It is also possible to calculate an approximate complete orthogonal decomposition by beginning with a QR factorization of A^T rather than a QR factorization of A as

in step (1) of the above algorithm. The SPQR_COD code includes this alternative as an option. Over the entire data set discussed in Section 3 the two alternatives are similar in performance. We won't discuss the alternative further except to note that it can be worthwhile to try the alternative for matrices where the default code in SPQR_COD returns a nonzero warning flag.

2.7 Comparison of Algorithms

Table I compares the routines SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD. The last column of the table reflects the results of the experiments described in Section 3.2. We can summarize these experiments as follows:

- All the algorithms accurately determine the numerical rank when warning flags returned by the algorithms are zero, except SPQR_NULL returns, rarely, an incorrect numerical rank when the warning flag is zero for one matrix (see Section 3.5).
- When the warning flag is zero the null spaces calculated by SPQR_COD are overall as good as those produced by MATLAB's dense matrix SVD routine and the approximate pseudoinverse solutions calculated by SPQR_COD are as accurate as predicted by the perturbation theory in [Stewart and Sun 1990].
- For some matrices SPQR_COD is more accurate than SPQR_BASIC, SPQR_NULL or SPQR_PINV. SPQR_COD uses T in (15) to estimate bounds for singular values of A . By the singular value perturbation property, Theorem 1, and by Theorem 5 the singular values of A and T differ by at most $\|E_1\| \leq \sqrt{n - \ell}\tau$. When the tolerance is small the singular values of A and T will be close and the estimates of singular values of A will be very good. SPQR_BASIC, SPQR_NULL and SPQR_PINV use estimates, which are returned in stats, of singular values of R_{11} in (6) or (11) to estimate lower bounds for singular values of A and use $R_1 = (R_{11}R_{12})$ in (6) or (11) to estimate upper bounds on these singular values. Although often these bounds are satisfactory, the singular values of R_{11} can, in some cases, be several orders of magnitude different than those of R_1 . In these cases SPQR_COD can provide better estimates of the numerical rank, bases for the numerical null space of A and approximate pseudoinverse solutions.

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). Note that, as mentioned earlier, 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.

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 (11) + U, V from SPQR_SSI	SPQR to A^T + SPQR_SSI to R_{11} in (11) + SPQR_SSP to $A N$	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 $A N$	good

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

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 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 discard the transformations as they are computed. Our methods presented in this paper, except for SPQR_BASIC, require Q . Least squares problems with rank deficient 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 a number of other algorithms that can potentially be used to construct an orthonormal null space basis of a sparse matrix A . These include the algorithm discussed in [Gotsman and Toledo 2008] and SVDS which is available as part of MATLAB and is based on ARPACK [Lehoucq et al. 1998]. The algorithm of [Gotsman and Toledo 2008] uses symmetric inverse iteration making use of an LU factorization of A . Symmetric inverse iteration is inverse iteration used with solutions to $A^T A x = y$. SVDS uses Arnoldi/Lanczos type algorithms on

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

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). The algorithm of [Gotsman and Toledo 2008] 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. A problem for SVDS is that it 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.

PROPACK [Larsen 1998] and SVDPACK [Berry 1994] also potentially could be used to find orthogonal null space bases. These packages were developed to find large singular values. Adaptation and testing would be required to use the packages to find small singular values and the corresponding singular vectors.

One of the goals of this paper is to describe an algorithm that determines a sparse representation of an orthonormal basis of the numerical null space of a matrix A . There is also research that investigates sparse matrix algorithms for construction of null space bases that are not orthogonal [Berry et al. 1985],[Coleman and Pothén 1986],[Coleman and Pothén 1987], [Gilbert and Heath 1987],[Heath 1982], [Gill et al. 2005]. Note that 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 235,880 bytes. For the same matrix the LU based algorithm nullspaceLUSOLform from LUSOL [Saunders 2006] required 5 seconds and 6,533,376 bytes 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 we should note that 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 1s on the diagonal and -1s above the diagonal [Gill et al. 2005, p. 113]. Therefore tests would be needed to insure that an LU based algorithm is reliable in the sense that the user is warned when the estimated rank may be incorrect [Foster 2007].

We should add that there are many algorithms for constructing rank revealing factorizations of dense matrices [Bischof and Quintana-Ortí 1998a],[Bischof and Quintana-Ortí 1998b],[Chan 1987], [Chandrasekaran and Ipsen 1994], [Fierro et al. 1999],[Foster and Kommu 2006],[Foster 1986],[Li and Zeng 2005], [Quintana-Ortí et al. 1998] and these algorithms can be the basis for determining numerical rank and constructing null space bases. However since these are dense matrix algorithms they are not efficient when applied to large, sparse matrices.

We should also mention that algorithms that are designed to calculate the eigen-

values of a symmetric matrix, for example [Polizzi 2009], [Zhang et al. 2007] or the routine SPTARN in MATLAB's PDE toolbox, can be applied to the matrix B in (16) to calculate singular values and, potentially, singular vectors. However the necessary adaptations of an eigenvalue routine to calculate null spaces are not necessarily straightforward [Gotsman and Toledo 2008, p. 460].

Finally note that work is being done by Rajamanickam and Davis on the PIRO-SKY package (see <http://www.cise.ufl.edu/~srajaman/>) which can be used to calculate the singular values of sparse matrices. The software for this research have not been released yet.

3. NUMERICAL EXPERIMENTS

In this section we apply SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD to a test set of 767 matrices and compare these routines with SPQR and SPQR_SOLVE from SuiteSparseQR as well as MATLAB's SVD and MATLAB's dense QR factorization. Note that MATLAB's 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 MATLAB's SVDS. 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. For these reasons we will not discuss SVDS further in this section.

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.

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 real applications or have characteristic features of problems from practice. 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 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 $\min(m, n)$. The tolerance used to define the numerical rank is essentially the same as the default tolerance in MATLAB's RANK command: $\tau = \max(m, n)\text{eps}(\text{an estimate of } \|A\|)$ as discussed in Section 2.3.

To determine the true numerical rank for 563 matrices in our test set singular values were calculated using MATLAB's SVD and for another 136 matrices the numerical rank was calculated with routine 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. For these matrices the structural nullity provides a lower bound on the numerical nullity [Davis 2006, p. 9].

Some properties of the matrices in the test set are summarized in Figure 1. As seen in Figure 1 the matrices in the test set have a wide range of sizes and numerical properties. Therefore the test set is an excellent set for testing algorithms for singular matrices.

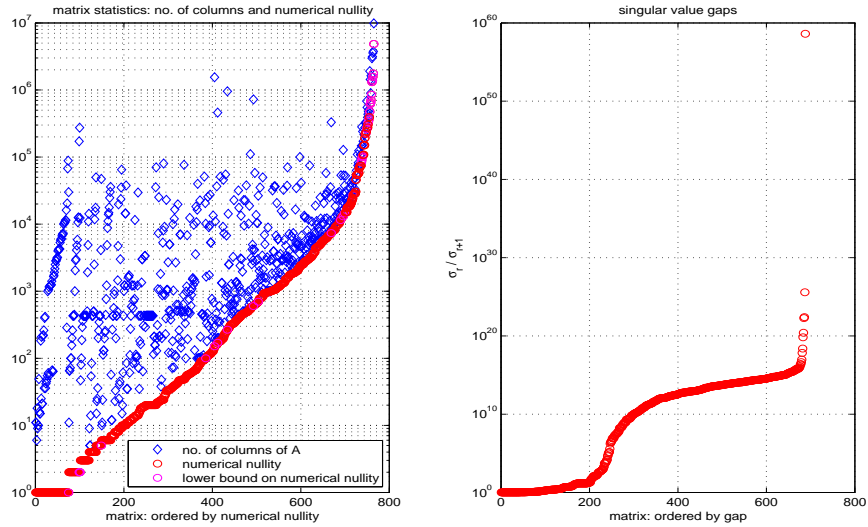


Fig. 1. Properties for matrices on our test set. The left hand plot pictures the numbers of columns and the dimensions of the numerical null spaces. The right hand plot pictures the gaps ($= \sigma_r / \sigma_{r+1}$) in the singular value spectrum at the calculated numerical rank, r .

3.2 Accuracy

We will 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. To test the numerical ranks we will use 697 matrices in the test set for which the numerical rank has been determined by other algorithms and for which memory limitations were not an issue. To test the accuracy and quality of the null space bases, the pseudoinverse solutions and the basic solutions we will use 446 matrices (all matrices in our test set with no more the 5000 rows or columns) for which it was practical to calculate null space bases and pseudoinverse solutions using MATLAB’s SVD. In addition to these tests which are discussed in Section 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 all our calculations we will use the tolerance $\tau = \max(m, n) \text{ eps}$ (an estimate of $\|A\|$) discussed earlier. There are applications where a larger tolerance would be appropriate. Our routines can be used in such cases, but to keep our experiments focused we use $\tau = \max(m, n) \text{ eps}$ (an estimate of $\|A\|$). As mentioned earlier for

some matrices the error bounds in the algorithms cannot confirm that the numerical rank is correct with the above 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 flag set to 1 and returns the value of the larger tolerance.

There are a number of other parameters in our routines. For algorithm SPQR_SSI discussed in Section 2.3 we set the initial block size to 3 and block size increment to be 5. For the SPQR_SSI stopping criteria discussed in Appendix A we set the maximum block size to 10 and the maximum number of iterations to 100. For the estimation of the norm of AN discussed in Appendix B we choose the stopping tolerance to be 0.1 and the maximum number of iterations to be 10. The routines SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD call SPQR and we chose `opts.econ = 0`, `opts.Q = 'Householder'`, `opts.permutation = 'vector'`, and `opts.tol = τ` , except in the second call to SPQR within SPQR_COD we set `opts.tol = 0`.

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 implemented, using MATLAB's MEX utility with 64 bit Visual Studio 2005 for the C++ compiler.

Before presenting a summary of our results we should note that 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 3.2 shows, for the 697 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 increase 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 3.2, the number of cases where the flag is zero or one closely tracks the number of cases where the calculated numerical rank is correct, especially for matrices with a larger singular value gap. In our data set there are cases where the calculated numerical rank is correct but the warning flag is not zero. For one matrix, Sandia/osci_dcop_33 [Davis and Hu 2011],[Foster and Botev 2009], which is discussed in Section 3.5, for SPQR_NULL the warning flag was, at times, zero and the calculated numerical rank was incorrect. For the other matrices in our experiments there are no “false positives,” that is no cases where the warning flag is zero or one and the calculated numerical rank is incorrect. As discussed in Section 3.5 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

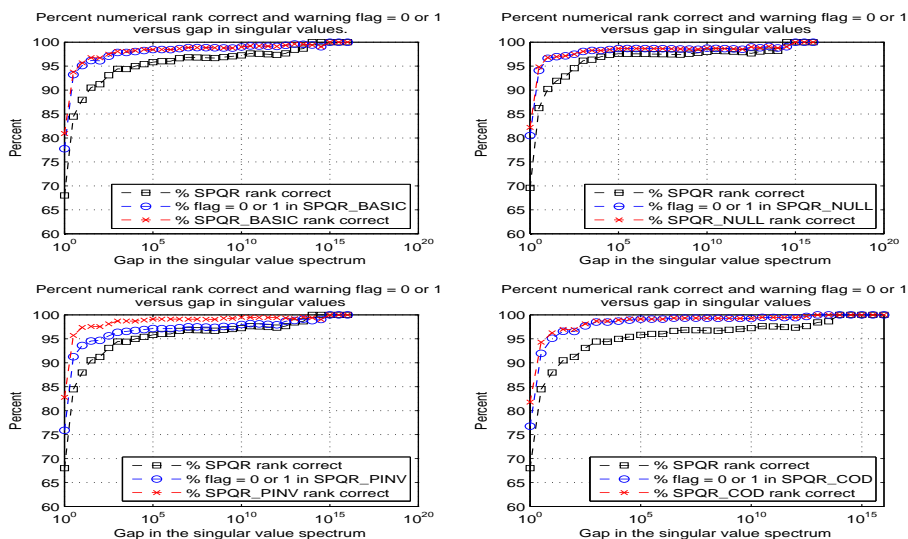


Fig. 2. For each of SPQR_BASIC, SPQR_NULL, SPQR_PINV, SPQR_COD and for SPQR the plots summarize the percent of matrices where the calculated numerical rank is correct and the percent of the matrices where the warning flag indicates that the calculated numerical rank is correct with a warning flag either 0 or 1 versus the singular value gap, σ_r/σ_{r+1} , where r is the calculated numerical rank.

significant problem. For these reasons we feel that the warning flag is a useful tool 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 446 $m \times n$ matrices in our test set with $\max(m, n) \leq 5000$ we also calculated a null spaces of A and A^T using MATLAB's SVD routine. We can use the results for these 446 matrices to compare, relative to the quality of numerical null spaces calculated using MATLAB's SVD, the quality of the null space bases 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. The main conclusion from Figure 3 is that, except potentially for one matrix for SPQR_NULL and one matrix for SPQR_PINV, MATLAB's SVD and the other routines, when the warning flag is zero, produce excellent null space bases. Note 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 tolerance, overall as good as those produced by the SVD.

The one matrix for SPQR_PINV and for SPQR_NULL where $\|AN\|/(\text{tolerance}) =$

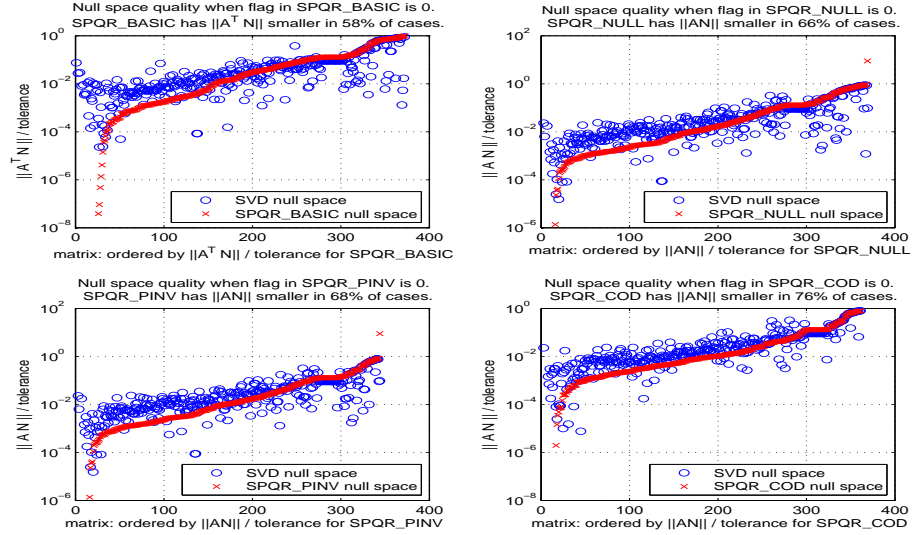


Fig. 3. $\|AN\|$ where N is a calculated orthonormal basis for the numerical null space or, in the case of SPQR_BASIC, $\|A^T N\|$, normalized by the tolerance defining the numerical rank is plotted for null space bases calculated by MATLAB’s SVD and by SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD.

$8.9 > 1$ has $\|AN\|/\|A\| = 4.8 \times 10^{-12}$ which is still small. Also for this matrix 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 estimated bounds are returned in the structure stats and if the bounds differ significantly the user may wish to use the routine SPQR_COD which can be more accurate than the other routines.

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 will 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 MATLAB QR factorization for dense matrices (calculated using $x = \text{full}(A) \setminus b$, if A is not square, or $x = \text{full}([A, 0*b]) \setminus b$, if A is square) with the norm of the solution, x_{PINV} , calculated using MATLAB’s PINV. 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 MATLAB’s dense matrix

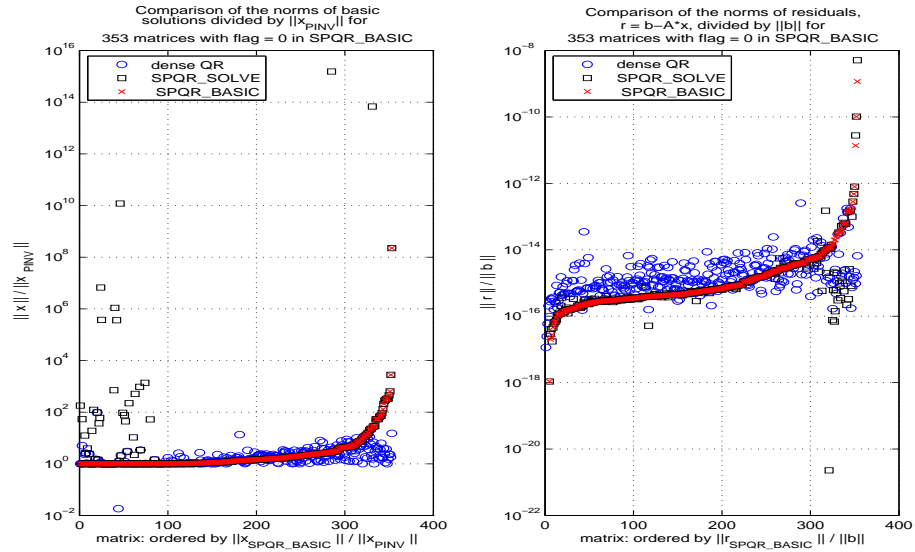


Fig. 4. The left plot pictures $\|x\|/\|x_{PINV}\|$ where x is a basic solution to (3) calculated by MATLAB’s dense matrix QR algorithm, by SPQR_SOLVE or by SPQR_BASIC and x_{PINV} is computed using MATLAB’s PINV. In the left hand plot the vectors b in (3) are random vectors. The right plot pictures $\|r\| = \|b - Ax\|$ for the x vectors calculated using MATLAB’s dense matrix QR algorithm, by SPQR_SOLVE or by SPQR_BASIC. In the right hand plot the vectors b in (3) are of the form $b = Ax$ where x is a random vector.

code. For some matrices even when flag is returned as zero 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 MATLAB’s 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 estimated bounds are returned in the structure stats and if the bounds differ significantly the user may wish to consider use of SPQR_COD rather than SPQR_BASIC.

The left hand plot in Figure 4 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 calculated using MATLAB’s 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 MATLAB’s dense QR factorization. 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. We can examine the accuracy of the pseudoinverse solution calculated by SPQR_PINV (x_{SPQR_PINV}) and SPQR_COD (x_{SPQR_COD}) by comparing solutions produced by these routines with the solutions calculated by using MATLAB's PINV ($x_{PINV} = \text{pinv}(\text{full}(A)) * b$). 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} . As an estimate of a bound on the accuracy that we should expect for $\|x - x_{PINV}\|/\|x_{PINV}\|$ we will use

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

where r is the numerical rank of A and $\|w\|$, as discussed in 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]. There are additional terms in the right hand side of equation 5.3 in [Stewart and Sun 1990, p. 157]. We will not include these terms in our discussion since (17) provides a satisfactory description of our numerical experiments for our tolerance choice.

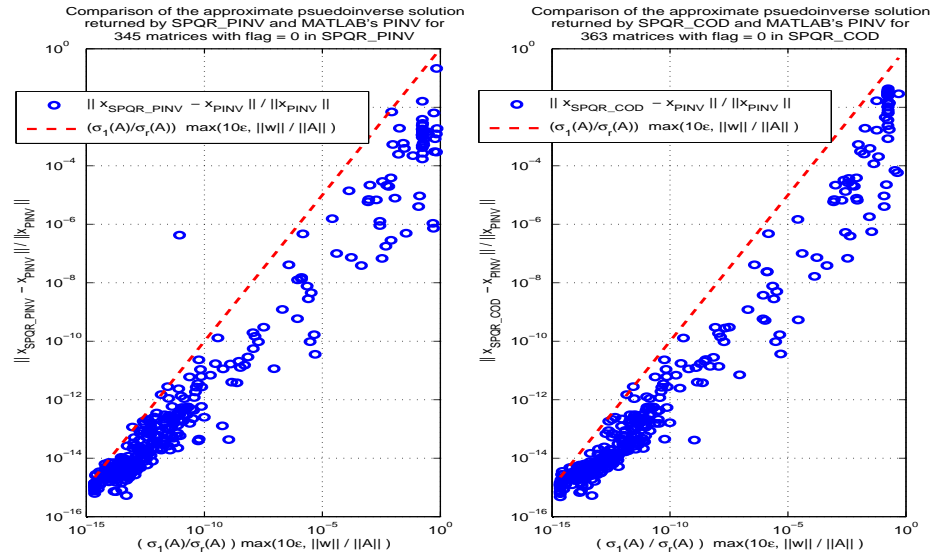


Fig. 5. The left graph plots $\|x - x_{PINV}\|/\|x_{PINV}\|$ for x produced by SPQR_PINV for 345 matrices where SPQR_PINV returns a flag of 0. The vectors b in (3) are random vectors. Also the right hand side of the perturbation theory result (17) is plotted. The right graph is the same plot for x produced by SPQR_COD for 362 matrices where SPQR_COD returns a flag of 0.

Figure 5 indicates that SPQR_COD and, except for one case, SPQR_PINV 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 the a value of $\|x - x_{PINV}\|/\|x_{PINV}\|$ larger than predicted (17). As discussed in the last section in such a case the estimates of upper and lower bounds for $\sigma_r(A)$ from SPQR_BASIC, which are returned in `stats.stats_spqr_basic`, will be orders of magnitude different. Users of SPQR_PINV can check these bounds and use SPQR_COD when the bounds differ significantly.

The routine SPQR_SOLVE, which is part of the SuiteSparseQR package, has an option to calculate approximate pseudoinverse or minimal norm solutions if the matrix A has fewer rows than columns. The calculations are based on SPQR without verifying that the numerical rank determined by SPQR is correct. In cases where the numerical rank determined by SPQR is incorrect the calculated solution can be poor. The matrix JGD_SL6/D_8 from [Davis and Hu 2011],[Foster and Botev 2009] is an 1172×1271 matrix with numerical rank 641 that provides an example of this. Using a tolerance of $\max(m,n)\text{eps}(\|A\|) = 4.5 \times 10^{-12}$ and a random vector b , SPQR_SOLVE with the “min2norm” option produces a solution to (3) with $\|x_{SPQR_SOLVE} - x_{PINV}\|/\|x_{PINV}\| = 6.1 \times 10^{13}$ whereas SPQR_COD calculates a solution with $\|x_{SPQR_COD} - x_{PINV}\|/\|x_{PINV}\| = 1.9 \times 10^{-14}$. Note the sign difference in the powers of 10.

3.3 Efficiency

Figure 6 compares the run times of SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD with the run time of SPQR for the 729 matrices in our data set that ran to completion.

As can be seen from the figure, over the entire data set, the average run time of SPQR_BASIC is approximately 22% larger than the average run time of SPQR applied to A , the average run time of SPQR_NULL is approximately 40% larger than the average run time of SPQR applied to A^T , the average run time of SPQR_PINV is approximately 136% larger than the average run time of SPQR applied to A , and the average run time of SPQR_COD is approximately 4.5 times the average run time of SPQR applied to A . On individual matrices the run times for SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD relative to the run time for SPQR can vary significantly from these averages, especially for smaller matrices.

Finally, we should note that we can estimate, extrapolating from the run times of matrices that are 5000 by 5000 or smaller, that MATLAB’s SVD 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. SPQR_COD is faster than MATLAB’s SVD by a factor larger than 10,000!

3.4 Memory use

Figure 6 compares the approximate memory requirement for SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD with the approximate memory requirements of SPQR for the 729 matrices in our data set that ran to completion. For SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD the memory pictured in the figure

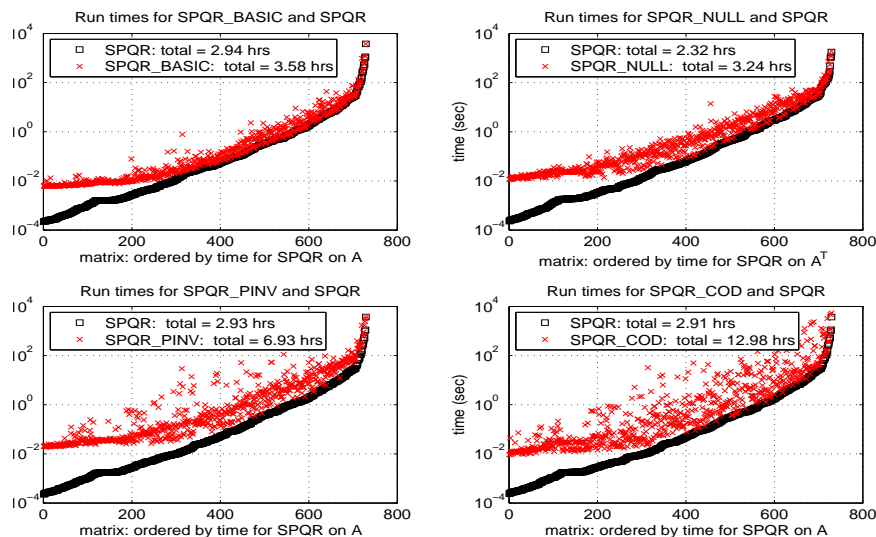


Fig. 6. Run times of SPQR_BASIC, SPQR_NULL, SPQR_PINV, and SPQR_COD and for SPQR for 729 matrices.

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. We should note that there is a potential for the routines to require additional memory as part of intermediate calculations. However for simplicity we will focus our discussion on the memory described above.

Figure 7 indicates that, in almost all cases, SPQR_BASIC, SPQR_NULL and SPQR_PINV require little additional memory beyond the memory required by SPQR. On average SPQR_COD requires approximately 3.4 times the amount of memory needed to store R and Q that result from SPQR applied to A. As mentioned earlier the successive QR factorizations in SPQR_COD compound the fill-in which leads to the additional memory requirements.

Figure 8 pictures the memory requirements for our implicit representations of orthonormal null space bases relative to the memory requirements for representing the bases explicitly as dense matrices. For matrices with a null space of low nullity usually less memory is required by an explicit representation. In such cases our routine SPQR_EXPLICIT_BASIS can be used to convert from implicit form to an explicit basis. If the nullity is low this conversion is usually quick. For matrices with a high dimension null space the savings in memory by using the implicit representation can be significant. As can be seen from the memory totals in the figures, on average, use of implicit representations for the bases requires an order of magnitude less memory than is required to represent the bases as explicit, dense matrices. For specific matrices the implicit representation can reduce the memory requirements by several orders of magnitude.

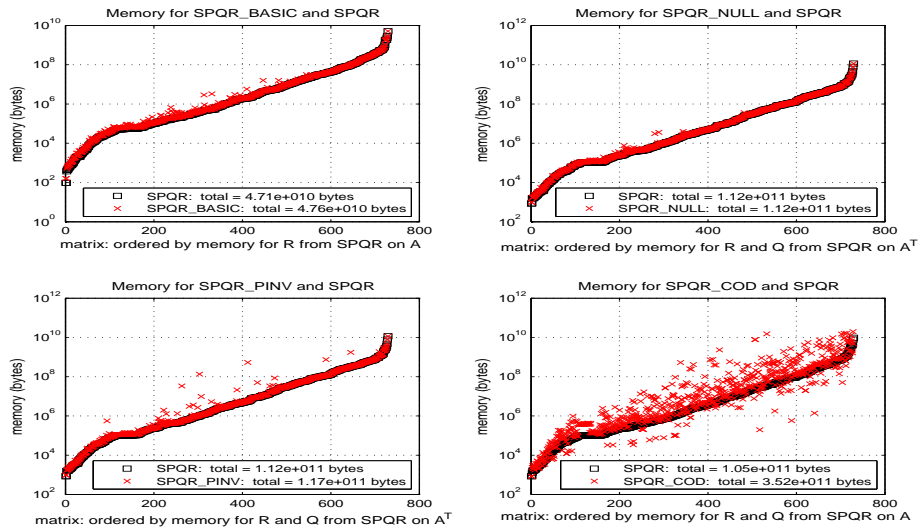


Fig. 7. Approximate memory requirements for SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD and for SPQR.

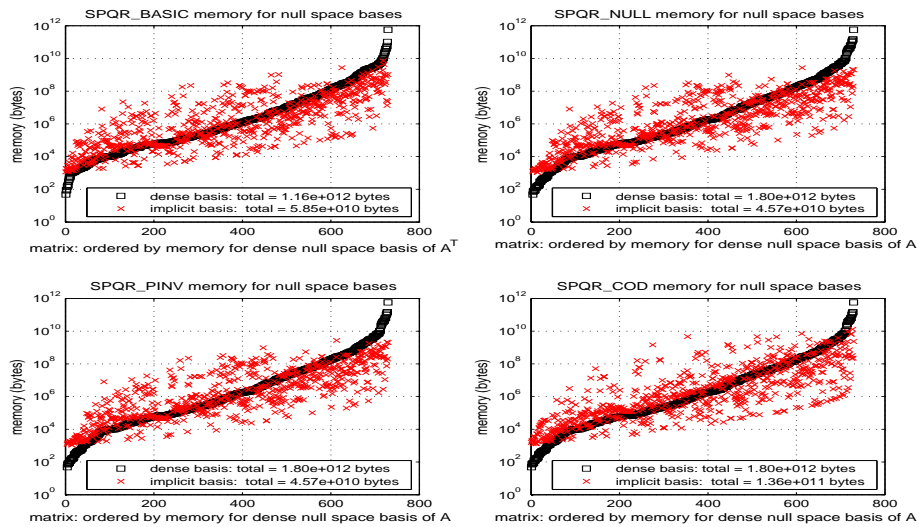


Fig. 8. Approximate memory requirements for implicit representations of the null space bases and explicit representations as dense matrices.

3.5 Challenging Examples

In this section we will briefly discuss examples that are challenging for our algorithms.

Gotsman and Toledo [Gotsman and Toledo 2008, p. 457] present several examples that are potentially difficult for their algorithm for calculating null spaces and also present challenges for some of our algorithms. Gotsman and Toledo note that the matrices exercise parts of their code that are rarely reached on real-world matrices. We consider two of these examples:

$$A_S = \begin{pmatrix} 1 & & & & & \\ -1 & 1 & & & & \\ \vdots & \ddots & \ddots & & & \\ -1 & & \ddots & & 1 & \\ -1 & -1 & \dots & -1 & 1 & \\ .5 & .5 & \dots & .5 & .5 & \end{pmatrix} \text{ and } A_I = \begin{pmatrix} 1 & \eta & & & & \\ & 1 & \ddots & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & \eta & \\ & & & & & 1 \end{pmatrix}. \quad (18)$$

A_S is an $(n+1) \times n$ matrix that has numerical rank n and is well conditioned. However the leading principal n by n submatrix is a triangular matrix with no small diagonal entries (they are all one) and yet has a condition number that is $O(2^n)$. If SPQR_NULL is applied to A_S or if SPQR_BASIC is applied to A_S^T with $n = 100$, for example, the routines return numerical ranks of 99 not 100 (and also return nonzero warning flags). The reason for this is that, since A_S^T has no small diagonal entries, the QR factorization that is produced by SPQR has $R = A_S^T$. In SPQR_BASIC the numerical rank is calculated by applying the routine SPQR_SSI to R_{11} in (6) and R_{11} has numerical rank 99.

For A_I with $\eta = 2$ and $n = 200$, for example, all our algorithms can fail to determine the numerical rank correctly. The matrix A_I has a condition number that is $O(2^n)$ [Gotsman and Toledo 2008, p. 458] and the resulting errors in inverse iteration lead to difficulties with the stopping criteria used in Algorithm SPQR_SSI. Note that the warning flags returned by the algorithms are nonzero and warn the user of the difficulty.

We will also discuss some of the examples in our test set where our algorithms were not successful. As pictured in Figure 1 many of the matrices in the test set do not have a significant gap in the singular value spectrum at the numerical rank and in these cases the numerical rank is not well determined. As pictured in Figure 2 most of the cases where the numerical ranks determined by SPQR_BASIC, SPQR_NULL, SPQR_PINV or SPQR_COD are inaccurate are for such matrices. However in a few cases these algorithms are not successful even when there is a significant gap in the singular value spectrum. For example, for the 466 matrices in our data set with a gap larger than one thousand, SPQR_BASIC does not return the correct numerical rank for 9 matrices, SPQR_NULL does not return the correct numerical rank for 9 matrices, SPQR_PINV does not return the correct numerical rank for 6 matrices and SPQR_COD does not return the correct numerical rank for 6 matrices. For most of these matrices choosing input options different from the default options will lead to successful calculation of the numerical rank. The reasons that the numerical ranks are incorrect include failure of subspace iterations, similar to the difficulty with A_I discussed above, and differences in the numerical rank of R_{11} and the numerical rank R_1 , similar to the difficulty with A_S . SPQR returns an incorrect estimated rank for 26 of these 466 matrices.

Note that the warning flag warns the user of a potential problem for every one of

the above examples when the algorithm does not return the correct numerical rank. For one other matrix, Sandia/oscii_dcop_33 [Davis and Hu 2011],[Foster and Botev 2009], one of our routines, SPQR_NULL, would, at times, return an incorrect numerical rank and a warning flag of zero. For this matrix and method occasionally the estimated singular value bounds were inaccurate. 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 with a warning flag of zero, i.e. “false convergence”. This matrix has singular values that decay very gradually to zero (see http://www.math.sjsu.edu/singular/matrices/html/Sandia/oscil_dcop_33.html) and therefore the numerical rank is not well defined. Note that reducing the value of the input option `opts.ssi.convergence.factor` from the default value of 0.1 to 0.001 eliminated this problem in our testing.

We should mention again that our singular value bounds are estimated bounds and for this reason the difficulty illustrated with matrix Sandia/oscii_dcop_33 and SPQR_NULL could, potentially, 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. Note that SPQR_SSI converges faster (see (10)) 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, optionally, its transpose. The basis for the numerical null space is represented by an implicit, sparse representation which allowed construction of orthonormal null space bases for matrices with large nullity including examples whose numerical null space dimensions are larger than 100,000. We have estimated that for large matrices the new code can be faster than MATLAB’s SVD 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. A warning flag of zero indicates, but does not guarantee since the singular value bounds are estimates, that the calculated numerical rank is correct. Our experiments indicate that the warning flag 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 a few matrices in our tests the basic solutions calculated by SPQR_BASIC (see Figure 4), the null space bases calculated by SPQR_NULL and SPQR_PINV (see Figure 3) or the approximate pseudoinverse solutions calculated by SPQR_PINV (see Figure 5) can be inferior to the corresponding results calculated by a dense matrix routine, even when the warning flag is zero. These examples occur when the estimated upper and lower bounds on $\sigma_r(A)$ where r is the calculated numerical rank, are significantly (orders of magnitude) different. The user can check these estimated bounds and if they differ significantly use SPQR_COD which does not have these difficulties in our tests. SPQR_COD can be more accurate but usually requires more time and memory (see Figures 6 and 7) than the other routines.

A key tool in the new algorithms is the routine SPQR from the SuiteSparseQR package [Davis 2011]. The new routines add facilities to the SuiteSparseQR package including, for almost all matrices, reliable calculation of numerical rank, pseudoinverse solutions and orthonormal bases for numerical null spaces as well as, in most cases, calculation of good basic solutions.

Appendix A. STOPPING CRITERIA FOR ALGORITHM SPQR_SSI

As part of the stopping criteria for Algorithm SPQR_SSI we introduce two additional parameters which are not discussed in the algorithm outline in the paper body: a maximum block size, `max_block_size`, and a maximum number of iterations, `max_iters`. The error flag for SPQR_SSI is set to a value greater than zero if the algorithm does not have $s_1 > \tau$ and pass the additional convergence tests described below for the block size $b \leq \text{max_block_size}$ and the number of iterations less than or equal to `max_iters`. Without these limitations SPQR_SSI can take too much time in those cases where the numerical rank determined by SPQR differs by a substantial amount from the true numerical rank.

To describe the remainder of the stopping criteria note that the goal of the algorithm is to return a numerical rank, r , that is the same as the numerical rank calculated using singular values. Let $\sigma_i(R)$, $i = 1, 2, \dots$ represent singular value i of a matrix R . The numerical rank r determined by the algorithm is correct if

$$\sigma_r(R) > \tau \text{ and } \sigma_{r+1}(R) \leq \tau. \quad (19)$$

One condition in our stopping criteria is that the block size b is increased to ensure

$$s_1 > \tau \text{ and } s_2 \leq \tau. \quad (20)$$

We would like to continue the iterations in the algorithm until the estimates s_1 for $\sigma_r(R)$ and s_2 for $\sigma_{r+1}(R)$ are accurate enough so that (20) implies (19).

We begin by considering the first condition in (19). To insure that the first condition in (20) implies the first condition in (19) the algorithm needs to carry out enough iterations so that

$$s_1 > \tau \text{ implies } \sigma_r(R) > \tau. \quad (21)$$

In algorithm SPQR_SSI from the first four steps in the repeat loop and from the redefinitions when the stopping criteria for success is met it follows that

$$\begin{pmatrix} 0 & R \\ R^T & 0 \end{pmatrix} \begin{pmatrix} U \\ V \end{pmatrix} - \begin{pmatrix} U \\ V \end{pmatrix} S \equiv B \begin{pmatrix} U \\ V \end{pmatrix} - \begin{pmatrix} U \\ V \end{pmatrix} S = \begin{pmatrix} RV - US \\ 0 \end{pmatrix} \quad (22)$$

The redefinition $V = VX_2$ is in the code to ensure that the second block row of (22) has a right hand side equal to zero (in exact arithmetic).

Let v_i be the i^{th} column of V and u_i be the i^{th} column of U , $i = 1, \dots, k$. From (22), since the norm of a column of $\begin{pmatrix} U \\ V \end{pmatrix}$ is $\sqrt{2}$ and due to the eigenvalue error bound, Theorem 4, it follows that for each $i = 1, \dots, k$ there exist an eigenvalue α of B such that

$$|s_i - \alpha| \leq \|Rv_i - u_i s_i\|/\sqrt{2} \equiv e_i. \quad (23)$$

Note that singular values of R are also eigenvalues of B [Golub and Van Loan 1996, p. 448]. We will use e_i in (23) as estimates of the errors in using s_i as an approximations for $\sigma_{r+i-1}(R)$.

Consider (23) for $i = 1$. The calculated e_1 is only an estimate of a bound in the error in using s_1 to approximate $\sigma_r(R)$ because the theory does not insure that α is the r^{th} singular value of A . However it is usually the case that $|s_1 - \sigma_r(R)| \leq e_1$. To be conservative, for $f \leq 1$ let us assume only that

$$|s_1 - \sigma_r(R)| \leq (1/f)e_1. \quad (24)$$

If we continue iterations in Algorithm SPQR_SSI until

$$e_1 \leq f|s_1 - \tau|. \quad (25)$$

then (24) and (25) imply $|s_1 - \sigma_r(R)| \leq |s_1 - \tau|$. From this result (21) follows. The default value of f is 0.1 and occasionally (24) will not be true for this f . However, when A has a well defined numerical rank the accuracy of the estimate of σ_r is not critical in SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD. In the experiments described in Section 3 these routines worked well.

Consider the second condition in (19). In exact arithmetic, by the singular value minimax property, Theorem 3, s_1, s_2, \dots estimated by Algorithm SPQR_SSI will be upper bounds for singular values $\sigma_r(R), \sigma_{r+1}(R), \dots$, respectively. Although in exact arithmetic, the second condition in (20) will imply the second condition in (19), in finite precision arithmetic it can be important to have a second test to confirm the second condition in (19). To do so SPQR_SSI calculates $\|RV(:, 2:k)\|$ and $\|R^T U(:, 2:k)\|$ (using MATLAB notation to refer to submatrices of U and V). By the singular value minimax property, Theorem 3, each of these are upper bounds on $\sigma_{r+1}(R)$. The algorithm determines that the second condition in (19) is satisfied if

$$\|RV(:, 2:k)\| \leq \tau \text{ and } \|R^T U(:, 2:k)\| \leq \tau. \quad (26)$$

The stopping criteria in Algorithm SPQR_SSI reports that the code succeeds when (19), (25) and (26) are true and, in addition, if $e_1 \leq f s_1$. This last requirement improves the relative accuracy in the calculated s_1 .

The error bound estimates returned by SPQR_SSI are calculated using (23). Finally we should note that to improve efficiency some of the error bound computations in the SPQR_SSI code are different but mathematically equivalent to the above computations.

Appendix B. ESTIMATING SINGULAR VALUES OF AN USING SPQR_SSP

When one of the routines SPQR_BASIC, SPQR_NULL, SPQR_PINV or SPQR_COD returns an orthonormal basis, N , for the null space of A or A^T the routines return estimates of $\|AN\|$ or $\|A^TN\|$. Also, as discussed in Section 2.4, singular values of AN can be used to improve, in some cases, the estimated singular value upper bounds for A that are returned by SPQR_BASIC, SPQR_NULL, SPQR_PINV or SPQR_COD. The routine SPQR_SSP, listed at the end of Appendix B, is used to carry out these calculation. SPQR_SSP uses subspace iteration to calculate estimates of the large singular values and corresponding singular vectors of a matrix A or, optionally, AN where N is a orthonormal basis for the null space of A as returned by SPQR_BASIC, SPQR_NULL, SPQR_PINV or SPQR_COD.

Algorithm SPQR_SSP is similar to Algorithm SI in [Vogel and Wade 1994, p. 741] and to routine SISVD in [Berry 1994]. Our stopping criteria is different than that in [Vogel and Wade 1994, p. 741] and the codes in [Vogel and Wade 1994, p. 741] and [Berry 1994] require one rather than two singular value decompositions per step. The calculation of singular value decompositions is only a small part of our overall calculations in routines SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD. Other alternatives to SPQR_SSP would be Lanczos/Arnoldi based routines such as SVDS which uses ARPACK [Lehoucq et al. 1998] or LASVD [Berry 1994]. We have not explored these alternatives to SPQR_SSP in any detail since existing code for these routines would require recoding to include the null space basis N , which we need, and since SPQR_SSP is usually not the major part of our overall calculations.

The error bounds and the stopping criteria in SPQR_SSP are based on the eigenvalue error bound, Theorem 4. From the third and fourth steps of the repeat loop and from the redefinition $V = VX_2$ it follows that $BV - US = 0$ where $B = A$ or, if N is input, $B = AN$. Therefore

$$\begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} U \\ V \end{pmatrix} - \begin{pmatrix} U \\ V \end{pmatrix} S \equiv C \begin{pmatrix} U \\ V \end{pmatrix} - \begin{pmatrix} U \\ V \end{pmatrix} S = \begin{pmatrix} 0 \\ B^T U - VS \end{pmatrix} \quad (27)$$

Let v_i be the i^{th} column of V and u_i be the i^{th} column of U , $i = 1, \dots, k$. From (27), since the norm of a column of $\begin{pmatrix} U \\ V \end{pmatrix}$ is $\sqrt{2}$ and due to the eigenvalue error bound, Theorem 4, it follows that for each $i = 1, \dots, k$ there exist an eigenvalue α of C such that

$$|s_i - \alpha| \leq \|B^T u_i - v_i s_i\| / \sqrt{2} \equiv e_i, \quad i = 1, \dots, k. \quad (28)$$

Note that singular values of B are also eigenvalues of C [Golub and Van Loan 1996, p. 448]. The routine SPQR_SSP uses e_i in (28) as estimates of the errors in using s_i as an approximations for $\sigma_i(A)$. e_k in the stopping criteria is equivalent to e_k as calculated in (28). 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, the codes SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD, which use SPQR_SSP, work well in practice.

The routines SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD set the default values of max_iters and convergence_factor in SPQR_SSP to 10 and 0.1,

respectively. This appears to provide sufficient accuracy in practice. The estimates of the singular values of AN in SPQR_BASIC, SPQR_NULL, SPQR_PINV and SPQR_COD do not require high accuracy.

Data:

- A : a $m \times n$ matrix
- N (optional): an explicit $n \times p$ matrix or an orthonormal basis, either in implicit form or as an explicit matrix, for a p dimensional numerical null space of A returned by SPQR_BASIC, SPQR_NULL, SPQR_PINV or SPQR_COD
- k (optional): the number of large singular values to estimate; 1 is the default value
- $opts$ (optional): additional options including `convergence_factor` and `max_iters` which are used in the stopping test

Result:

- U : an $n \times k$ matrix containing estimates of the left singular vectors of A corresponding to estimated singular values in S .
- S : a $k \times k$ diagonal matrix whose diagonal entries $s_i, i = 1, \dots, k$, are estimates of the largest k singular values of A . Also for $i = 1, \dots, k$ s_i is a lower bound for singular value i of A .
- V : an $n \times k$ matrix containing estimates of the right singular vectors of A .
- $stats$: a structure containing additional information including $e_i, i = 1, \dots, k$ which are estimates for bounds on the errors in s_i .

Initialize:

- Let $k = \min(k, m, n)$ or, if N is input, $k = \min(k, m, p)$
- $U =$ a random $n \times k$ matrix with orthonormal columns

repeat

- $V_1 = A^T U$ or, if N is input, $V_1 = N^T A^T U$
- determine the compact SVD of V_1 : $V D_1 X_1^T = V_1$, where V is $n \times k$, D_1 is a $k \times k$ diagonal matrix, and X_1 is $k \times k$
- $U_1 = AV$ or, if N is input, $U_1 = ANV$
- determine the compact SVD of U_1 : $U S X_2^T = U_1$, where U is $n \times k$, S is a $k \times k$ diagonal matrix, and X_2 is $k \times k$
- Let $s_i, i = 1, \dots, k$ be the diagonal entries of S
- Let u_k be column k of U and x_k be column k of X_2 . Calculate $e_k = \|B^T u_k - V x_k s_k\|/\sqrt{2}$ where $B = A$ or, if N is input, $B = AN$

until $e_k \leq (\text{convergence_factor}) s_k$ or *max_iter iterations have been reached* ;

- $V = V X_2$
- For $i = 1, \dots, k - 1$, let $e_i =$ the Euclidean norm of the i^{th} column of $(B^T U - V S)/\sqrt{2}$ where $B = A$ or, if N is input, $B = AN$
- if $e_k \leq (\text{convergence_factor}) s_k$ let $flag = 0$; otherwise let $flag = 1$.

Algorithm SPQR_SSP: subspace iteration to estimate large singular values of A or AN

Appendix C. DEFLATION

Step 2 in SPQR_BASIC and step 3 in SPQR_COD require the solution to $\ell \times \ell$ triangular systems whose numerical rank r may be less than ℓ . Consider

$$R\hat{z} = \hat{c} \tag{29}$$

where R is an $\ell \times \ell$ triangular matrix (R_{11} in SPQR_BASIC or T^T in SPQR_COD) whose numerical rank is less than ℓ . We can solve (29) approximately using the method of deflation [Chan 1984],[Chan and Hansen 1990],[Stewart 1981]. SPQR_BASIC and SPQR_COD apply algorithm SPQR_SSI to R to calculate the numerical rank of R . The routine returns an $\ell \times k$ matrix U of approximate left singular vectors of R and an $\ell \times k$ matrix V of approximate right singular vectors of R . When SPQR_SSI succeeds and returns an error flag of 0 then $k = \ell - r + 1$. If so $V_2 =$ the last $\ell - r$ columns of V and $U_2 =$ the last $\ell - r$ columns of U are singular vectors corresponding to approximate singular values that are less than the tolerance τ defining the numerical rank. The approximate solution to (29) produced by deflation is

$$\hat{z} = (I - V_2 V_2^T) R^{-1} (I - U_2 U_2^T) \hat{c} \quad (30)$$

Four observations relating to (30) are worth noting. First, note that due to the application of SPQR in the construction of R , R will not be exactly singular (assuming the tolerance is > 0) so that (30) makes sense. Second, note that (30) can be written so that only matrix vector multiplications or solutions to triangular systems are used. Therefore the calculations in (30) can be computed quickly. Third, we should address the concern that, since R will be ill-conditioned when $r < \ell$, there appears to be the potential for an unacceptable growth in computer arithmetic errors in the calculation (30). To address this concern note that U_2 will contain approximations for the left singular vectors of R that correspond to the $\ell - r$ smallest singular values of R and V_2 will contain approximations for the right singular vectors of R that correspond to the $\ell - r$ smallest singular values of R . The calculation $(I - U_2 U_2^T) \hat{c}$ in (30) will filter out components in \hat{c} that will produce error growth in $R^{-1} (I - U_2 U_2^T) \hat{c}$ and the application of $(I - V_2 V_2^T)$ will also filter errors in the calculation. Therefore the calculations in (30) are usually reliable. See [Chan 1984],[Chan and Hansen 1990],[Stewart 1981] for further discussion. Finally, note that if we assume that U_2 and V_2 contain exact left and right singular vectors corresponding to the $\ell - r$ smallest singular values we can motivate (30). To do so let U_1 and V_1 contain the exact left and right singular vectors corresponding to the r largest singular values of R and let D_1 be the $r \times r$ diagonal matrix containing the r largest singular values of R . It then follows that (30) is equivalent to the truncated singular value solution [Chan and Hansen 1990] to (29):

$$\hat{z} = V_1 D_1^{-1} U_1^T \hat{c}.$$

The truncated singular value solution is widely used [Björck 1996], [Chan and Hansen 1990], [Enting 2002], [Hansen 1998].

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*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.
- BARLOW, J. L. AND VEMULAPATI, U. B. 1992. Rank detection methods for sparse matrices. *SIAM J. Matrix Anal. Appl.* 13, 4, 1279–1297.
- 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. 1998a. Algorithm 782: codes for rank-revealing QR factorizations of dense matrices. *ACM Trans. Math. Software* 24, 2, 254–257.
- BISCHOF, C. H. AND QUINTANA-ORTÍ, G. 1998b. Computing rank-revealing QR factorizations of dense matrices. *ACM Trans. Math. Software* 24, 2, 226–253.
- BJÖRCK, A. 1996. *Numerical methods for least squares problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.
- CHAN, T. F. 1984. Deflated decomposition of solutions of nearly singular systems. *SIAM J. Numer. Anal.* 21, 4, 738–754.
- CHAN, T. F. 1987. Rank revealing QR factorizations. *Linear Algebra Appl.* 88/89, 67–82.
- 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.
- CHANDRASEKARAN, S. AND IPSEN, I. C. F. 1994. On rank-revealing factorisations. *SIAM J. Matrix Anal. Appl.* 15, 2, 592–622.
- COLEMAN, T. F. AND POTHEM, A. 1986. The null space problem i. complexity. *SIAM Journal on Algebraic and Discrete Methods* 7, 4, 527–537.
- COLEMAN, T. F. AND POTHEM, 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 9xx, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization. to appear *ACM TOMS*.
- DAVIS, T. A. AND HU, Y. F. 2011. The University of Florida sparse matrix collection. Tech. Rep. to appear ACM TOMS, see <http://www.cise.ufl.edu/research/sparse/matrices/>.
- DEMME, 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.
- FIERRO, R. D., HANSEN, P. C., AND HANSEN, P. S. K. 1999. UTV tools: Matlab templates for rank-revealing UTV decompositions. *Numer. Algorithms* 20, 2-3, 165–194.
- 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. 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. AND KOMMU, R. 2006. Algorithm 853: an efficient algorithm for solving rank-deficient least squares problems. *ACM Transactions on Mathematical Software* 32, 1 (Mar.).
- 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. 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 University Singular Matrix Data Base. See <http://www.math.sjsu.edu/singular/matrices/>.
- 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. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA. Numerical aspects of linear inversion.
- HEATH, M. T. 1982. Some extensions of an algorithm for sparse linear least squares problems. *SIAM J. Sci. Statist. Comput.* 3, 2, 223–237.
- 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.
- LARSEN, R. M. 1998. PROPACK software for SVD of sparse matrices. Tech. Rep. see <http://soi.stanford.edu/rmunk/PROPACK/>.
- LEHOUCQ, R. B., SORENSEN, D. C., AND YANG, C. 1998. *ARPACK users' guide*. Software, Environments, and Tools, vol. 6. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA. Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.
- 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.
- POLIZZI, E. 2009. Density-matrix-based algorithms for solving eigenvalue problems. *Phys. Rev. B* 79, 115112, 1–6.
- QUINTANA-ORTÍ, G., SUN, X., AND BISCHOF, C. H. 1998. A BLAS-3 version of the QR factorization with column pivoting. *SIAM J. Sci. Comput.* 19, 5, 1486–1494 (electronic).
- SAUNDERS, M. 2006. LUSOL: A basis package for constrained optimization. Linear Algebra and Optimization Seminar, SCCM, Stanford University, , February, 2006. See <http://www.stanford.edu/group/SOL/talks/saunders-LUSOL-linopt2006.pdf>. Software for the package is at <http://www.stanford.edu/group/SOL/software/lusol.html>.
- 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. Iterative methods in numerical linear algebra (Copper Mountain Resort, CO, 1992).
- ZHANG, H., SMITH, B., STERNBERG, M., AND ZAPOL, P. 2007. SIPs: shift-and-invert parallel spectral transformations. *ACM Trans. Math. Software* 33, 2, Art. 9, 19.

Received xxx; xxx; accepted xxx