Codes for Rank-Revealing QR Factorizations of Dense Matrices

Christian H. Bischof
Mathematics and Computer Science Division, Bldg. 223, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439. Phone (708) 252-8875. bischof@mcs.anl.gov
and
Gregorio Quintana-Ortí
Departamento de Informática, Universidad Jaime I, Campus Penyeta Roja, 12071 Castellón, Spain. gquintan@inf.uji.es.

This paper describes a suite of codes as well as associated testing and timing drivers for computing rank-revealing QR factorizations (RRQR factorizations) of dense matrices. The main contribution is an efficient block algorithm for approximating an RRQR factorization, employing a windowed version of the commonly used Businger/Golub pivoting strategy, and improved versions of the RRQR algorithms for triangular matrices originally suggested by Chandrasekaran and Ipsen, and Pan and Tang, respectively. We highlight usage and features of these codes, and give an example of their use in the context of solving rank-deficient least-squares systems.

Additional Key Words and Phrases: rank-revealing QR factorization, numerical rank, block algorithm

---

Argonne Preprint ANL-MCS-P560-0196
submitted to ACMTransactions on Mathematical Software

1. OVERVIEW

Given a $m \times n$ matrix A with singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$ and a threshold τ, we define the numerical rank r of A through the smallest singular value σ_r which satisfies $\sigma_r \leq \sigma_1/\tau$. That is, if there is a reasonable gap between σ_r and σ_{r+1}, and σ_{r+1} is small, it makes sense to consider A to numerically have rank r.

This work was supported by the Applied and Computational Mathematics Program, Advanced Research Projects Agency, under contracts DM28E04120 and P-95006.
Bischof was also supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U. S. Department of Energy, under Contract W-31-109-Eng-38.
Quintana also received support through the European ESPRIT Project 9072–GEPPCOM and the Spanish Research Agency CICYT under grant TIC-91-1157-C03-02. During part of this work, Quintana was a research fellow of the Spanish Ministry of Education and Science of the Valencian Government at the Universidad Politécnica de Valencia and while visiting the Mathematics and Computer Science Division at Argonne National Laboratory.

Our goal is to compute a QR factorization

$$A\,P = QR = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{1}$$

where

$$\kappa(R_{11}) \approx \sigma_1/\sigma_r \leq \tau \text{ and } \|R_{22}\|_2 = \sigma_{max}(R_{22}) \approx \sigma_{r+1}. \tag{2}$$

Here P is a permutation matrix, Q has orthonormal columns, R is upper triangular, R_{11} is of order r, and $\kappa(A)$ denotes the two-norm condition number of matrix A. That is, R_{11} captures the well-conditioned part of the spectrum of A.

The implementations presented in this paper are based on the work described in [Bischof and Quintana-Ortí 1996]. The main computational routines are:

xGEQPB. An approximate RRQR factorization employing a restricted column pivoting scheme. For large enough matrices, this routine performs almost all of its work using BLAS-3 kernels.

xTRQPX. An efficient variant of the Chandrasekaran/Ipsen RRQR algorithm for triangular matrices by Chandrasekaran and Ipsen [1994].

xTRQPY. An efficient variant of the RRQR algorithm for triangular matrices by Pan and Tang [1992].

We have provided codes in single, double, complex, and double complex precisions, and we use to denote the "x" at the beginning of a routine name to generically refer to a particular routine. Executing either xTRQPX or xTRQPY after xGEQPB, we arrive at an algorithm combining the efficiency of block orthogonal computations with the guaranteed reliability provided by the postprocessing algorithms, and this functionality is provided in the xGEQPX and xGEQPY routines. We expect that in most instances, these routines will provide a more reliable and efficient substitute for the LAPACK [Anderson et al. 1994] routine xGEQPF.

The paper is structured as follows. In section 2 we describe the usage of the DGEQPX and DGEQPY routines for computing a RRQR factorization of a dense matrix. Section 3 describes the contents of the code package and where to find it. Lastly, we present our conclusions.

2. THE DGEQPX AND DGEQPY ROUTINES FOR COMPUTING A RRQR FACTORIZATION

The routines xGEQPX and xGEQPY implement dense RRQR factorizations combining xGEQPB and one of xTRQPX and xTRQPY. Their calling sequence and usage is identical, and we discuss here the double precision implementation

```
      SUBROUTINE DGEQPX( JOB, M, N, K, A, LDA, C, LDC, JPVT, RCOND,
     $                   RANK, SVLUES, WORK, LWORK, INFO )
```

of the routine incorporating DGEQPB and DGEQPX. We comment on the arguments in sequence, denoting the usage of the argument (input, output, both, or workspace) as well as its type. For brevity, we omitted obvious facts such as implied conformities of matrix sizes. They are described in detail in the documentation provided as part of the codes. In addition, all codes employ extensive checking of input arguments and will flag unsuitable use of a particular argument using the LAPACK XERBLA routine.

JOB – (input)– INTEGER
: Describes whether to update the matrix C with the orthogonal matrix Q computed in the RRQR factorization. If JOB == 1, C is not touched, if JOB == 2, C is overwritten by $Q^T \cdot C$, and if JOB == 3, C is overwritten by $C \cdot Q$.

M – (input) – INTEGER
: The number of rows of A.

N – (input) – INTEGER
: The number of columns of A.

K – (input) – INTEGER
: The number of columns (if JOB == 2) or rows (if JOB == 3) of C.

A – (input/output) – DOUBLE PRECISION array, dimension (LDA,N)
: On entry, the $m \times n$ matrix A. On exit, the upper triangle of the array contains the $\min(m, n) \times n$ upper trapezoidal matrix R; the lower triangle is zero.

LDA – (input) – INTEGER
: The leading dimension of array A.

C – (input/output) DOUBLE PRECISION array, dimension (LDC, NC)
: The number of columns of this matrix is K if JOB == 2, or M if JOB == 3. If JOB == 1, this matrix is not touched, otherwise it is updated with Q.

LDC – (input) INTEGER
: The leading dimension of array C.

JPVT – (output) INTEGER array, dimension (N)
: JPVT encodes the permutation matrix P: If JPVT(I) = J, then column j of A has been permuted into position i of AP.

RCOND –(input/output) DOUBLE PRECISION
: On entry, 1/RCOND specifies the upper bound τ on the condition number of R_{11}. If RCOND == 0 on entry, $\tau = 1/\epsilon$, where ϵ is the machine precision, is chosen as default. On exit, 1/RCOND is an estimate for the two norm condition number of R_{11}.

RANK – (output) – INTEGER
: RANK is an estimate for the numerical rank of A with respect to the threshold 1/RCOND in the sense that RANK = $\mathrm{arg\_max}(\kappa(R(1:r, 1:r)) < 1/\mathrm{RCOND})$

SVLUES (output) DOUBLE PRECISION array, dimension (4)
: SVLUES contains estimates of some of the singular values of the triangular factor R.
: SVLUES(1). largest singular value of R(1:RANK,1:RANK)
: SVLUES(2). smallest singular value of R(1:RANK,1:RANK)
: SVLUES(3). smallest singular value of R(1:RANK+1,1:RANK+1)
: SVLUES(4). smallest singular value of R

Due to the properties of a RRQR factorization, SVLUES(1) will also be an estimate for the largest singular value of A, SVLUES(2) and SVLUES(3) will be estimates for the RANK-th and (RANK+1)-st singular value of A, and SVLUES(4) will be an estimate for the smallest singular value of A. By examining these values, one can confirm that the rank is well defined with respect to the condition number threshold chosen.

WORK – (workspace) DOUBLE PRECISION array, dimension (LWORK)

On exit, work(1) is the size of the storage array needed for optimal performance.

LWORK – (input) – INTEGER

Dimension of the array WORK. If JOB=1, the unblocked strategy requires that LWORK $\geq$ 2*MN+3*N and the block algorithm requires that LWORK $\geq$ 2*MN+N*NB.

If JOB $\neq$ 1, the unblocked strategy requires that: LWORK $\geq$ 2*MN+2*N+MAX(K,N), and the block algorithm requires that: LWORK $\geq$ 2*MN+NB*NB+NB*MAX(K,N). Here MN = min(M,N) and NB is the block size. In both cases, the minimum required workspace is the one for the unblocked strategy. The code chooses the blocksize that is returned by the LAPACK ILAENV routine for the xGEQRF routine.

INFO – (output) – INTEGER

This argument returns the exit status of the subroutine:

INFO == 0. Successful exit.

INFO < 0. If INFO = -i, the i-th input argument had an illegal value

INFO == 1, 2 or 3. In some cases, we have come across cases where that our estimates for both $\kappa_2(R(1:r,1:r))$ and $\kappa_2(R(1:r+1,1:r+1))$ are smaller than the threshold τ or alternatively, both $\kappa_2(R(1:r+1,1:r+1))$ and $\kappa_2(R(1:r+2,1:r+2))$ are larger than the threshold τ. Even though we have not observed it, we cannot rule out the possibility of catastrophic failure of our condition estimation procedure, but in all likelyhood this occurrence is due to the cutoff for the rank being in the middle of a singular value cluster, and thus ill-defined. Thus, we return as rank the value r, but we warn the user that the rank is not well defined. The singular value estimates returned in SVLUES should provide further information.

INFO == 4. The postprocessing steps are based on iterative processes to reveal the rank. If the number of iterations exceeds a maximum threshold, we terminate with INFO = 4. We have not observed this event in our experiments. For xTRQPX, the limit is $n+25$ steps, where 4 steps make roughly one iteration (a step is one call to Golub-I or Stewart-II). For xTRQPY, the limit is $n+25$ iterations.

We point out the following differences to LAPACK's DGEQPF routine:

—The matrix Q is not returned in factored form, rather it is applied "on the fly" to a matrix C if desired. This approach avoids the regeneration of the orthogonal factors computed in DGEQPB and avoids problems arising from the unpredictable number of permutations that could potentially be required in the postprocessing stage.

—There are no provisions for "fixed" columns. DGEQPF allows one to force certain columns to be pivoted up front, by initializing the corresponding JPVT entries to nonzero values on entry. DGEQPX and DGEQPY cannot provide this functionality since we have no guarantee that the user would move the "right" columns for a RRQR factorization up front.

—DGEQPF does not provide any estimates of the singular values of A. In contrast, our routines provide good estimates of the singular values of R (which are guaranteed to be close to those of A due to the RRQR property) around the rank defined by the threshold of A, and of the largest and smallest singular value of A. Thus, a user can verify that his threshold is numerically meaningful with respect to rank determination. This is important, as determining a sensible threshold and a resulting rank is very much application-dependent and usually requires considerable user insight into the problem at hand.

3. CONTENTS OF THE RRQR SUBROUTINE PACKAGE

The package containing our RRQR program suite is available via anonymous ftp from `ftp.super.org` as file `rrqr.tar.gz` in the `pub/prism` directory. The package contains the following pieces:

Main Routines and Drivers:.

—The xGEQPB, xTRQPX, and xTRQPY routines for approximating and validating a RRQR factorization.

—The xGEQPX and xGEQPY drivers combining xGEQPB with xTRQPX and xTRQPY, respectively.

—The xGELSA and XGELSB drivers respectively employing xGEQPX and xGEQPY in solving a rank-deficient linear least squares problem.

Testing Environment:. We supply code for testing the numerical reliability of xGEQPB, xGEQPX, and xGEQPY, as well as the LAPACK routine xGEQPF. This code computes residuals, compares singular value estimates with actual values, and compares the "best possible" result for the condition number of the leading factor of an RRQR factorization with our actual results. Contained herein is also the matrix generator suite that we employed for the results reported in [Bischof and Quintana-Ortí 1996].

Timing Environment:. We supply code for timing xGEQPB, xGEQPX, xGEQPY, as well as the LAPACK routine xGEQPF.

Support Routines:. We supply code for all routines called by the main routines or testing and timing codes, including

Needed LAPACK Routines:. We include those routines from the LAPACK Version 1 release [Anderson et al. 1994] that we call upon. The complete LAPACK library can be obtained from `netlib`, see `http://www.netlib.org`.

All codes are provided for in "templatized" real and complex versions. That is, we employ a simple textual replacement utility to generate codes in `REAL` and `DOUBLE PRECISION` from the "meta-real" and in `COMPLEX` and `DOUBLE COMPLEX` from the "meta-complex" version. The C preprocessor "cpp" is employed to generate timing and testing versions of the code from the usual two common sources. Unix makefiles are provided that facilitate installation and compilation of the codes.

4. CONCLUSIONS

This paper described the usage of the main routines in a suite of codes for computing reliable rank-revealing orthogonal factorizations in an efficient fashion. The codes build to a large extent on LAPACK infrastructure and can easily be substituted for most uses of the LAPACK xGEQPF routine which computes a QR factorization employing the traditional column pivoting strategy.

References

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. 1994 . *LAPACK User's Guide Release 2.0*. SIAM, Philadelphia.

Bischof, C. H. and Quintana-Ortí, G. 1996 . Computing rank-revealing qr factorizations of dense matrices. Preprint MCS-P559-0196, Mathematics and Computer Science Division, Argonne National Laboratory.

Chandrasekaran, S. and Ipsen, I. 1994 . On rank-revealing QR factorizations. *SIAM Journal on Matrix Analysis and Applications 15*, 2.

Pan, C.-T. and Tang, P. T. P. 1992 . Bounds on singular values revealed by QR factorizaton. Technical Report MCS-P332-1092, Mathematics and Computer Science Division, Argonne National Laboratory.