LAPACK-Style Codes for Pivoted Cholesky and QR Updating

Sven Hammarling[1], Nick Higham[2], and Craig Lucas[3]

[1] NAG Ltd.,Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR, England, sven@nag.co.uk, http://www.nag.co.uk/about/shammarling.asp
[2] Department of Mathematics, University of Manchester, M13 9PL, England, higham@ma.man.ac.uk, http://www.ma.man.ac.uk/~higham
[3] Manchester Computing, University of Manchester, M13 9PL, England. craig.lucas@manchester.ac.uk, http://www.ma.man.ac.uk/~clucas. This work was supported by an Engineering and Physical Sciences Research Council Research Studentship.

Abstract. Routines exist in LAPACK for computing the Cholesky factorization of a symmetric positive definite matrix and in LINPACK there is a pivoted routine for positive *semi*definite matrices. We present new higher level BLAS LAPACK-style codes for computing this pivoted factorization. We show that these can be many times faster than the LINPACK code. Also, with a new stopping criterion, there is more reliable rank detection and smaller normwise backward error.
We also present algorithms that update the QR factorization of a matrix after it has had a block of rows or columns added or deleted. This is achieved by updating the factors Q and R of the original matrix. We present some LAPACK-style codes and show these can be much faster than computing the factorization from scratch, and the backward error of our updated factors is comparable to that of a standard QR factorization.

1 Pivoted Cholesky Factorization

1.1 Introduction

The Cholesky factorization of a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$ has the form

$$A = LL^T,$$

where $L \in \mathbb{R}^{n\times n}$ is a lower triangular matrix with positive diagonal elements. If A is positive *semi*definite, of rank r, there exists a Cholesky factorization with *complete pivoting* ([4, Thm. 10.9], for example). That is, there exists a permutation matrix $P \in \mathbb{R}^{n\times n}$ such that

$$P^T AP = LL^T,$$

where L is unique in the form

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{12} & 0 \end{bmatrix},$$

with $L_{11} \in \mathbb{R}^{r\times r}$ lower triangular with positive diagonal elements.

1.2 Algorithms

In LAPACK [1] there are Level 2 BLAS and Level 3 BLAS routines for computing the Cholesky factorization in the full rank case and without pivoting. In LINPACK [2] the routines `xCHDC` perform the Cholesky factorization with complete

pivoting, but effectively uses only Level 1 BLAS. For computational efficiency we would like a pivoted routine that exploits the Level 2 or Level 3 BLAS.

In this section of the talk we describe a pivoted 'Gaxpy' Level 2 BLAS algorithm for positive semidefinite matrices. We describe the existing LAPACK Level 3 code and explain why this code cannot be altered to include pivoting. We give an alternative Level 3 algorithm and show that this can include pivoting. This is given by writing the semidefinite matrix $A^{(k-1)} \in \mathbb{R}^{n\times n}$ and $n_b \in \mathbb{R}$ [3]

$$A^{(k-1)} = \begin{bmatrix} A_{11}^{(k-1)} & A_{12}^{(k-1)} \\ A_{12}^{T(k-1)} & A_{22}^{(k-1)} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I_{n-n_b} \end{bmatrix} \begin{bmatrix} I_{n_b} & 0 \\ 0 & A^{(k)} \end{bmatrix} \begin{bmatrix} L_{11} & 0 \\ L_{21} & I_{n-n_b} \end{bmatrix}^T,$$

where $L_{11} \in \mathbb{R}^{n_b\times n_b}$ and $L_{21} \in \mathbb{R}^{(n-n_b)\times n_b}$ form the first n_b columns of the Cholesky factor L of $A^{(k-1)}$. Now to complete our factorization of $A^{(k-1)}$ we need to factor the reduced matrix

$$A^{(k)} = A_{22}^{(k-1)} - L_{21}L_{21}^T,$$

which we can explicitly form, taking advantage of symmetry.

We also discuss possible stopping criteria and their effect on the computed Cholesky factor.

Finally we report on the following numerical experiments.

1.3 Numerical Experiments

We tested and compared four Fortran subroutines: LINPACK's `DCHDC`, `DCHDC` altered to use a different stopping criteria and implementations of a level 2 pivoted Gaxpy algorithm and level 3 pivoted Gaxpy algorithm.

We report on the following tests:

- We first compare the speed of the factorization of the LINPACK code and our Level 2 and 3 routines for different sizes of $A \in \mathbb{R}^{n\times n}$.
- We also compare the speed of the unpivoted LAPACK subroutines against our Level 2 and Level 3 pivoted codes, using full rank matrices, to demonstrate the pivoting overhead.
- Finally we look at the four subroutines acting on a set of random positive semidefinite matrices with pre-determined eigenvalues to assess the rank detection properties and backward error of the computed factor.

1.4 Results

The tests show that our codes are much faster than the existing LINPACK codes. We also show that the pivoting overhead is negligible for large n.

Furthermore, with a new stopping criterion the rank is revealed much more reliably, and this leads to a smaller normwise backward error.

2 Updating the QR Factorization

2.1 Introduction

We wish to update the QR factorization

$$A = QR \in \mathbb{R}^{m\times n},$$

where $Q \in \mathbb{R}^{n\times n}$ is orthogonal and $R \in \mathbb{R}^{m\times n}$ is upper trapezoidal, efficiently. That is we wish to find $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is the updated A, it has had rows or columns

added or deleted. We seek to do this without recomputing the factorization from scratch. We will assume that A and $\widetilde{A}$ have full rank.

Where possible we derive blocked algorithms to exploit the Level 3 BLAS and existing Level 3 LAPACK routines. We present LAPACK style code for updating the QR factorization in the cases of adding and deleting blocks of columns.

In the following section we will talk about all the cases where rows and columns are added to or deleted from the QR factorization. We derive algorithms for updating and finding a suitable orthogonal matrix H. This matrix is a product of Householder and/or Givens matrices.

2.2 Deleting a Block of Rows

If we delete a block of p rows, $A(k\colon k+p-1, 1\colon n)$, from A we have for a permutation matrix, P,

$$PA = \begin{bmatrix} A(k\colon k+p-1, 1\colon n) \\ \widetilde{A} \end{bmatrix}.$$

Thus we require an orthogonal matrix H such that

$$\begin{bmatrix} A(k\colon k+p-1, 1\colon n) \\ \widetilde{A} \end{bmatrix} = (PQH)H^T R = \begin{bmatrix} I & 0 \\ 0 & \widetilde{Q} \end{bmatrix} \begin{bmatrix} V \\ \widetilde{R} \end{bmatrix}.$$

2.3 Adding a Block of Rows

If we add a block of p rows, $U \in \mathbb{R}^{(p\times n)}$, in the kth to $(k+p-1)$st positions of A we can write

$$\widetilde{A} = \begin{bmatrix} A(1\colon k-1, 1\colon n) \\ U \\ A(k\colon m, 1\colon n) \end{bmatrix}, \quad P\widetilde{A} = \begin{bmatrix} A \\ U \end{bmatrix},$$

for a permutation matrix, P, and

$$\begin{bmatrix} Q^T & 0 \\ 0 & I_p \end{bmatrix} P\widetilde{A} = \begin{bmatrix} R \\ U \end{bmatrix}.$$

Thus we seek an orthogonal matrix H such that

$$\widetilde{A} = \left(P^T \begin{bmatrix} Q & 0 \\ 0 & I_p \end{bmatrix} H \right) H^T \begin{bmatrix} R \\ U \end{bmatrix} = \widetilde{Q}\widetilde{R}.$$

2.4 Deleting a Block of Columns

If we delete a block of p columns, from the kth column onwards, from A, we can write

$$\widetilde{A} = [\, A(1\colon m, 1\colon k-1) \quad A(1\colon m, k+p\colon n) \,]$$

then

$$Q^T\widetilde{A} = [\, R(1\colon m, 1\colon k-1) \quad R(1\colon m, k+p\colon n) \,].$$

Thus we require an orthogonal matrix H such that

$$\widetilde{A} = (QH)H^T\, [\, R(1\colon m, 1\colon k-1) \quad R(1\colon m, k+p\colon n) \,] = \widetilde{Q}^T\widetilde{R}.$$

2.5 Adding a Block of Columns

If we add a block of p columns, $U \in \mathbb{R}^{m\times p}$, in the kth to $(k+p-1)$st positions of A, we can write

$$\widetilde{A} = [\, A(1\colon m, 1\colon k-1) \quad U \quad A(1\colon m, k\colon n)\,]$$

then

$$Q^T\widetilde{A} = [\, R(1\colon m, 1\colon k-1) \quad V \quad R(1\colon m, k\colon n)\,],$$

where $V = Q^TU$, and we require an orthogonal matrix H such that

$$\widetilde{A} = (QH)H^T\,[\, R(1\colon m, 1\colon k-1) \quad V \quad R(1\colon m, k\colon n)\,] = \widetilde{Q}\widetilde{R}.$$

2.6 Numerical Experiments

Here we look at implementations of algorithms for the latter two cases above, and report on the following tests:

- We compare the speed of our double precision Fortran 77 codes against LAPACK's `DGEQRF`, a Level 3 BLAS routine for computing the QR factorization of a matrix. We timed our codes acting on $Q^T\widetilde{A}$, the starting point for computing $\widetilde{R}$, and in the case of adding columns we included the computation of Q^TU in our timings. We also timed `DGEQRF` acting on only the part of $Q^T\widetilde{A}$ that needs to be updated, the nonzero part from row and column k onwards.
- We also look at the computed backward error of an updated factorization after repeatedly adding and deleting columns.

2.7 Results

The speed tests show that our updating algorithms are faster than computing the QR factorization from scratch or using the factorization to update columns k onward.

Furthermore, the normwise backward error tests show that the errors are within the bound for computing the Householder QR factorization of $\widetilde{A}$. Thus, within the parameters of our experiments, the increase of speed is not at the detriment of accuracy.

References

1. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Third edition, SIAM, Philadelphia, 1999.
2. J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1979.
3. G. H. Golub and C. F. Van Loan. *Matrix Computations*. Third edition, The Johns Hopkins University Press, Baltimore, MD, USA, 1996.
4. Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.