LAPACK-Style Codes for Pivoted Cholesky and QR Updating

Hammarling, Sven and Higham, Nicholas J. and Lucas, Craig

2006

MIMS EPrint: **2006.385**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

LAPACK-Style Codes for Pivoted Cholesky and QR Updating

Sven Hammarling[1], Nicholas J. 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] School 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 EPSRC 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 a block of columns 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.

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* ([6, Thm. 10.9], for example). That is, there exists a permutation matrix $P \in \mathbb{R}^{n\times n}$ such that P^TAP has a unique Cholesky factorization

$$P^TAP = LL^T, \quad L = \begin{bmatrix} L_{11} & 0 \\ L_{12} & 0 \end{bmatrix},$$

where $L_{11} \in \mathbb{R}^{r\times r}$ is 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 routine `xCHDC` performs 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.

The LAPACK Level 3 algorithm cannot be pivoted, so we instead start with the Level 2 algorithm. The LAPACK 'Gaxpy' Level 2 BLAS algorithm is:

Algorithm 1 *This algorithm computes the Cholesky factorization $A = LL^T$ of a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$, overwriting A with L.*

```
    Set L = lower triangular part of A
    for j = 1:n
(*)     L(j,j) = L(j,j) - L(j,1:j-1)L(j,1:j-1)^T
(#)     if L(j,j) ≤ 0, return, end % Quit if A not positive definite.
        L(j,j) = sqrt(L(j,j))
        % Update jth column
        if 1 < j < n
            L(j+1:n,j) = L(j+1:n,j) - L(j+1:n,1:j-1)L(j,1:j-1)^T
        end
        if j < n
            L(j+1:n,j) = L(j+1:n,j)/L(j,j)
        end
    end
```

This algorithm requires $n^3/3$ flops.

We can introduce pivoting into Algorithm 1, for $L = (\ell_{ij})$, by finding the largest possible ℓ_{jj} at $(*)$ from the remaining $n-j+1$ diagonal elements and using it as the pivot. We find

$$q = \min\Big\{p : L(p,p) - d(p) = \max_{j\le i\le n}\{L(i,i) - d(i)\}\Big\}, \tag{1.1}$$

where d is a vector of dot products with

$$d(i) = L(i,1\colon j-1)L(i,1\colon j-1)^T, \quad i = j\colon n, \tag{1.2}$$

and swap rows and columns q and j, putting the pivot ℓ_{qq} into the lead position. This is *complete pivoting*.

For computational efficiency we can store the inner products in (1.2) and update them on each iteration. This approach gives a pivoted gaxpy algorithm. The pivoting overhead is $3(r+1)n - 3/2(r+1)^2$ flops and $(r+1)n - (r+1)^2/2$ comparisons, where $r = \mathrm{rank}(A)$.

The computed rank, $\hat{r}$, of A can be determined by a stopping criterion at $(\#)$ in Algorithm 1. At the jth iteration if the pivot, which we will denote by $\chi_{jj}^{(j)}$,

satisfies an appropriate condition then we set the trailing matrix $L(j\colon n, j\colon n)$ to zero and the computed rank is $j-1$. Three possible stopping criteria are discussed in [6, Sec. 10.3.2]. The first is used in LINPACK's code for the Cholesky factorization with complete pivoting, `xCHDC`. Here the algorithm is stopped on the kth step if

$$\chi_{ii}^{(k)} \le 0, \quad i = k\colon n. \tag{1.3}$$

In practice $\hat{r}$ may be greater than r due to rounding errors. In [6] the other two criteria are shown to work more effectively. The first is

$$\|\widetilde{S}_k\| \le \epsilon \|A\| \quad \text{or} \quad \chi_{ii}^{(k)} \le 0, \quad i = k\colon n, \tag{1.4}$$

where $\widetilde{S}_k = A_{22} - A_{12}^T A_{11}^{-1} A_{12}$, with $A_{11} \in \mathbb{R}^{k\times k}$ the leading submatrix of A, is the Schur complement of A_{11} in A, while the second related criterion is

$$\max_{k\le i\le n} \chi_{ii}^{(k)} \le \epsilon \chi_{11}^{(1)}, \tag{1.5}$$

where in both cases $\epsilon = nu$, and u is the unit roundoff. We have used the latter criterion, preferred for its lower computational cost.

We derive a blocked algorithm by using the fact that we can write, for 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

$$A^{(k)} = A_{22}^{(k-1)} - L_{21} L_{21}^T, \tag{1.6}$$

which we can explicitly form, taking advantage of symmetry.

From this representation we can derive a block algorithm. At the kth step we factor n_b columns, by applying a pivoted Algorithm 1 to the leading principal $n_b \times n_b$ submatrix of $A^{(k)}$ and then update the trailing matrix according to (1.6) and continue.

At each step the Level 2 part of the algorithm requires $(n-(k-1)n_b)n_b^2$ flops and the Level 3 update requires $(n-kn_b)^3/3$ flops. The Level 3 fraction is approximately $1 - 3n_b/2n$.

1.3 Numerical Experiments

We tested and compared four Fortran subroutines on a 1400MHz AMD Athlon: LINPACK's `DCHDC`, `DCHDC` altered to use our stopping criterion, and LAPACK-style implementations of a level 2 pivoted Gaxpy algorithm (`LEV2PCHOL`) and level 3 pivoted Gaxpy algorithm (`LEV3PCHOL`) .

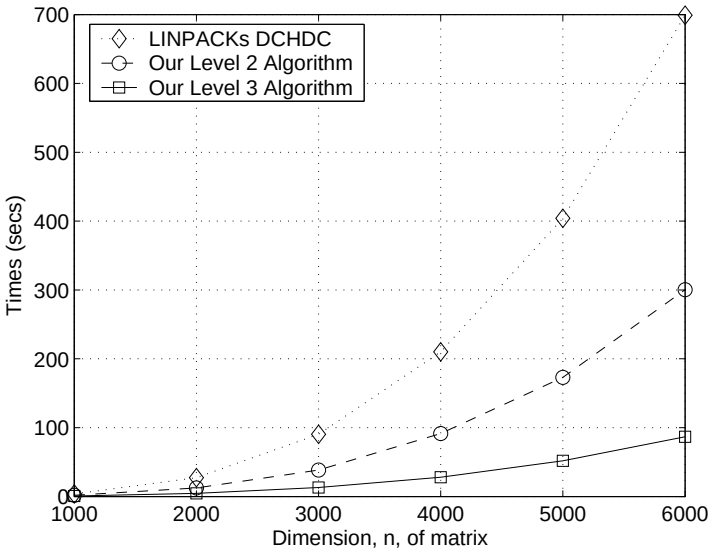

Fig. 1. Comparison of speed for different n.

We first compared 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 generated random symmetric positive semidefinite matrices of order n and rank $r = 0.7n$. For each value of n the codes were run four times and the mean times are shown in Figure 1.3. We achieve a good speedup, with the Level 3 code as much as 8 times faster than the LINPACK code.

We also compared the speed of the unpivoted LAPACK subroutines against our Level 3 pivoted code, using full rank matrices, to demonstrate the pivoting overhead. The ratio of speed of the pivoted codes to the unpivoted codes varies smoothly from 1.6 for $n = 1000$ to 1.01 for $n = 6000$, so the pivoting overhead is negligible in practice for large n (recall that the pivoting overhead is about $3rn - 3/2r^2$ flops within the $O(n^3)$ algorithm). The use of the pivoted codes instead of the unpivoted ones could be warranted if there is any doubt over whether a matrix is positive definite.

We tested all four subroutines on a further set of random positive semidefinite matrices, this time with pre-determined eigenvalues, similarly to the tests in [5]. For matrices of rank r we chose the nonzero eigenvalues in three ways:

- Case 1: $\lambda_1 = \lambda_2 = \cdots = \lambda_{r-1} = 1, \quad \lambda_r = \alpha \leq 1$
- Case 2: $\lambda_1 = 1, \quad \lambda_2 = \lambda_3 = \cdots = \lambda_r = \alpha \leq 1$
- Case 3: $\lambda_i = \alpha^{i-1}, \quad 1 \leq i \leq r, \quad \alpha \leq 1$

Here, α was chosen to vary $\kappa_2(A) = \lambda_1/\lambda_r$.

For each case we constructed a set of 100 matrices by using every combination of:

$$n = \{70,\ 100,\ 200,\ 500,\ 1000\},$$
$$\kappa_2(A) = \{1,\ 1e{+}3,\ 1e{+}6,\ 1e{+}9,\ 1e{+}12\},$$
$$r = \{0.2n,\ 0.3n,\ 0.5n,\ 0.9n\},$$

where $r = \mathrm{rank}(A)$. We computed the relative normwise backward error

$$\frac{\|A - \widehat{P}\widehat{L}\widehat{L}^T\widehat{P}^T\|_2}{\|A\|_2},$$

for the computed Cholesky factor $\widehat{L}$ and permutation matrix $\widehat{P}$.

Table 1. Maximum normwise backward errors.

n	70	100	200	500	1000
`DCHDC`	3.172e-13	1.498e-13	1.031e-12	2.823e-12	4.737e-11
`DCHDC` with (1.5)	7.778e-15	9.014e-15	1.810e-14	7.746e-14	1.991e-13
`LEV2PCHOL`	4.633e-15	9.283e-15	1.458e-14	7.290e-14	1.983e-13
`LEV3PCHOL`	4.633e-15	9.283e-15	1.710e-14	8.247e-14	2.049e-13

There was little difference between the normwise backward errors in the three test cases; Table 1 shows the maximum values over all cases for different n. The codes with the new stopping criterion give smaller errors than the original LINPACK code. In fact, for all the codes with our stopping criterion $\hat{r} = r$, and so the rank was detected exactly. This was not the case for the unmodified `DCHDC`, and the error, $\hat{r} - r$, is shown in Table 2.

Table 2. Errors in computed rank for `DCHDC`.

n	70	100	200	500	1000
min	0	0	1	4	4
max	10	12	16	16	19

The larger backward error for the original `DCHDC` is due to the stopping criterion. As Table 2 shows, the routine is often terminated after more steps than our codes, adding more nonzero columns to $\widehat{L}$.

1.4 Conclusions

Our codes for the Cholesky factorization with complete pivoting are much faster than the existing LINPACK code. Furthermore, with a new stopping criterion

the rank is revealed much more reliably, and this leads to a smaller normwise backward error.

For more detailed information on the material in this section see [7].

2 Updating the QR Factorization

2.1 Introduction

We wish to update efficiently the QR factorization

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

where $Q \in \mathbb{R}^{m\times m}$ is orthogonal and $R \in \mathbb{R}^{m\times n}$ is upper trapezoidal. That is we wish to find $\widetilde{A} = \widetilde{Q}\widetilde{R}$, where $\widetilde{A}$ is A with 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.

We consider the cases of adding blocks of rows and columns and deleting blocks of columns. Where possible we derive blocked algorithms.

2.2 Adding a Block of Rows

If we add a block of p rows, $U \in \mathbb{R}^{p\times n}$, just before the kth row of A we can write

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

and we can define a permutation matrix, P, such that

$$P\widetilde{A} = \begin{bmatrix} A \\ U \end{bmatrix},$$

and

$$\begin{bmatrix} Q^T & 0 \\ 0 & I_p \end{bmatrix} P\widetilde{A} = \begin{bmatrix} R \\ U \end{bmatrix}. \tag{2.1}$$

Thus to find $\widetilde{A} = \widetilde{Q}\widetilde{R}$, we can define n Householder matrices to eliminate U to give

$$H_n \dots H_1 \begin{bmatrix} R \\ U \end{bmatrix} = \widetilde{R},$$

so we have

$$\widetilde{A} = \left(P^T \begin{bmatrix} Q & 0 \\ 0 & I_p \end{bmatrix} H_1 \dots H_n \right) \widetilde{R} = \widetilde{Q}\widetilde{R}.$$

The Householder matrix, $H_j \in \mathbb{R}^{(m+p)\times(m+p)}$, will zero the jth column of U. Its associated Householder vector, $v_j \in \mathbb{R}^{(m+p)}$, is such that

$$\begin{aligned} v_j(1\colon j-1) &= 0, \quad v_j(j) = 1, \\ v_j(j+1\colon m) &= 0, \\ v_j(m+1\colon m+p) &= x/(r_{jj} - \| \begin{bmatrix} r_{jj} & x^T \end{bmatrix} \|_2), \text{ where } x = U(1\colon p, j). \end{aligned}$$

We can derive a blocked algorithm by using the representation of the product of Householder matrices in [8].

2.3 Deleting a Block of Columns

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

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

and then

$$Q^T \widetilde{A} = [\, R(1\colon m, 1\colon k-1) \quad R(1\colon m, k+p\colon n) \,]. \tag{2.2}$$

Thus we can define $n-p-k+1$ Householder matrices, $H_j \in \mathbb{R}^{m\times m}$, with associated Householder vectors, $v_j \in \mathbb{R}^{(p+1)}$ such that

$$\begin{aligned} v_j(1\colon j-1) &= 0, \quad v_j(j) = 1, \\ v_j(j+1\colon j+p) &= x/((\widetilde{Q}^T\widetilde{A})_{jj} - \| \,[\, (\widetilde{Q}^T\widetilde{A})_{jj} \quad x^T \,]\, \|_2), \\ &\qquad \text{where } x = Q^T\widetilde{A}(j+1\colon j+p, j), \\ v_j(j+p+1\colon m) &= 0. \end{aligned}$$

The H_j can be used to eliminate the subdiagonal of $Q^T\widetilde{A}$ to give

$$(H_{n-p}\ldots H_k Q^T)\widetilde{A} = \widetilde{Q}^T\widetilde{A} = \widetilde{R},$$

where $\widetilde{R} \in \mathbb{R}^{m\times(n-p)}$ is upper trapezoidal and $\widetilde{Q} \in \mathbb{R}^{m\times m}$ is orthogonal.

2.4 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

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

and

$$Q^T\widetilde{A} = \begin{bmatrix} R_{11} & V_{12} & R_{12} \\ 0 & V_{22} & R_{23} \\ 0 & V_{32} & 0 \end{bmatrix},$$

where $R_{11} \in \mathbb{R}^{(k-1)\times(k-1)}$ and $R_{23} \in \mathbb{R}^{(n-k+1)\times(n-k+1)}$ are upper triangular. Then if V_{32} has the (blocked) QR factorization $V_{32} = Q_V R_V \in \mathbb{R}^{(m-n)\times p}$ we have

$$\begin{bmatrix} I_n & 0 \\ 0 & Q_V^T \end{bmatrix} Q^T\widetilde{A} = \begin{bmatrix} R_{11} & V_{12} & R_{12} \\ 0 & V_{22} & R_{23} \\ 0 & R_V & 0 \end{bmatrix}.$$

We then eliminate the upper triangular part of R_V and the lower triangular part of V_{22} with Givens matrices, which makes R_{23} full and the bottom right block upper trapezoidal. So we have finally

$$\begin{aligned} &G(k+2p-2, k+2p-1)^T \ldots G(k+p, k+p+1)^T G(k, k+1)^T \\ &\qquad\qquad \ldots G(k+p-1, k+p)^T \begin{bmatrix} I_n & 0 \\ 0 & Q_V^T \end{bmatrix} Q^T\widetilde{A} = \widetilde{R}, \end{aligned}$$

where $G(i, j)$ are Givens rotations acting on the ith and jth rows.

3 Numerical Experiments

We tested the speed of LAPACK-style implementations of our algorithms for updating after adding (`DELCOLS`) and deleting (`ADDCOLS`) columns, against LAPACK's `DGEQRF`, for computing the QR factorization of a matrix.

We tested the codes with $m = \{1000,\ 2000,\ 3000,\ 4000,\ 5000\}$ and $n = 0.3m$, and the number of columns added or deleted was $p = 100$. 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 in our timings the computation of Q^TU, which we formed with the BLAS routine `DGEMM`. 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. Here we can construct $\widetilde{R}$ with this computation and the original R. Finally, we timed `DGEQRF` acting on $\widetilde{A}$. We aim to show our codes are faster than these alternatives. In all cases an average of three timings is given.

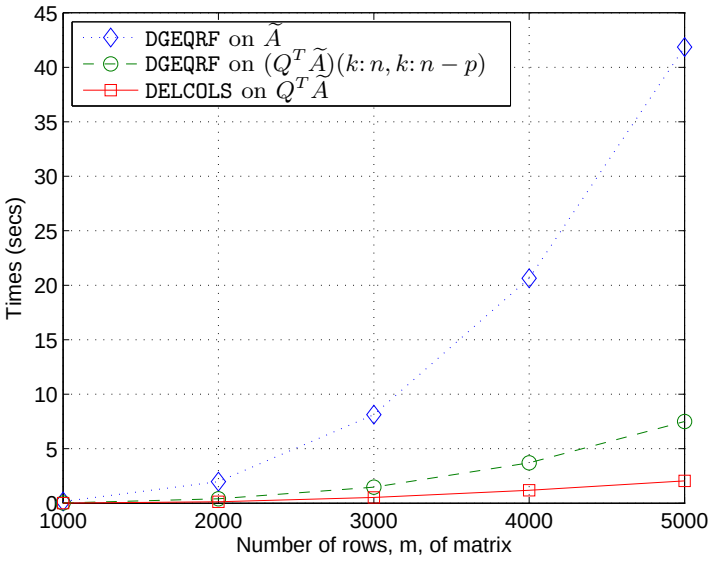

Fig. 2. Comparison of speed for `DELCOLS` with $k = 1$ for different m.

To test our code `DELCOLS` we chose $k = 1$, the position of the first column deleted, where the maximum amount of work is required to update the factorization. We timed `DGEQRF` on $\widetilde{A}$, `DGEQRF` on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$ which computes the nonzero entries of $\widetilde{R}(k\colon m, p+1\colon n)$ and `DELCOLS` on $Q^T\widetilde{A}$. The results are given in Figure 2. Our code is much faster than recomputing the factorization from scratch with `DGEQRF`, and for $n = 5000$ there is a speedup of 20. Our code is also faster than using `DGEQRF` on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$, where there is a maximum speedup of over 3.

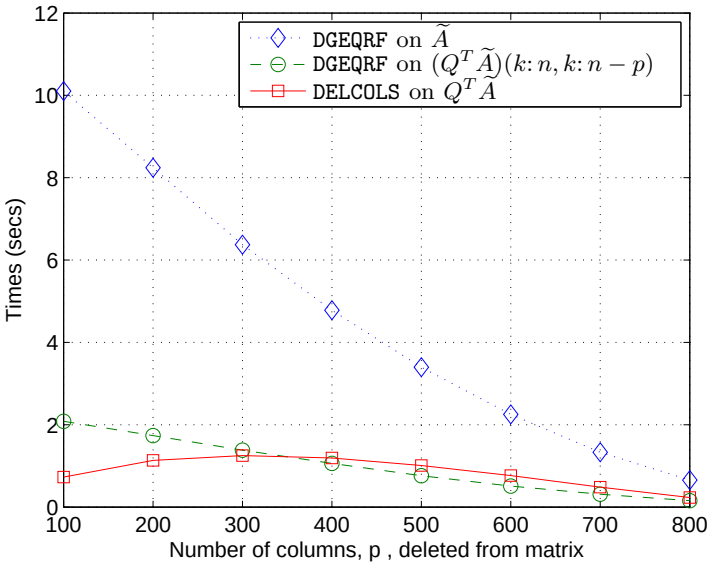

Fig. 3. Comparison of speed for `DELCOLS` for different p.

We then considered the effect of varying p with `DELCOLS` for fixed $m = 3000$, $n = 1000$ and $k = 1$. We chose $p = \{100,\ 200,\ 300,\ 400,\ 500,\ 600\ 700,\ 800\}$. As we delete more columns from A there are fewer columns to update, but more work is required for each one. We timed `DGEQRF` on $\widetilde{A}$, `DGEQRF` on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$ which computes the nonzero entries of $\widetilde{R}(k\colon m, k\colon n-p)$ and `DELCOLS` on $Q^T\widetilde{A}$. The results are given in Figure 3. The timings for `DELCOLS` are relatively level and peak at $p = 300$, whereas the timings for the other codes obviously decrease with p. The speedup of our code decreases with p, and from $p = 300$ there is little difference between our code and `DGEQRF` on $(Q^T\widetilde{A})(k\colon n, k\colon n-p)$.

To test `ADDCOLS` we generated random matrices $A \in \mathbb{R}^{m\times n}$ and $U \in \mathbb{R}^{m\times p}$. We set $k = 1$ where maximum updating is required. We timed `DGEQRF` on $\widetilde{A}$ and `ADDCOLS` on $Q^T\widetilde{A}$, including the computation of Q^TU with `DGEMM`. The results are given in Figure 4. Here our code achieves a speedup of over 3 for $m = 5000$ over the complete factorization of $\widetilde{A}$.

We do not vary p as this increases the work for our code and `DGEQRF` on $(Q^T\widetilde{A})(k\colon m, k\colon n+p)$ roughly equally.

3.1 Conclusions

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, the only columns needing updating.

For more detailed information on the material in this section see [4].

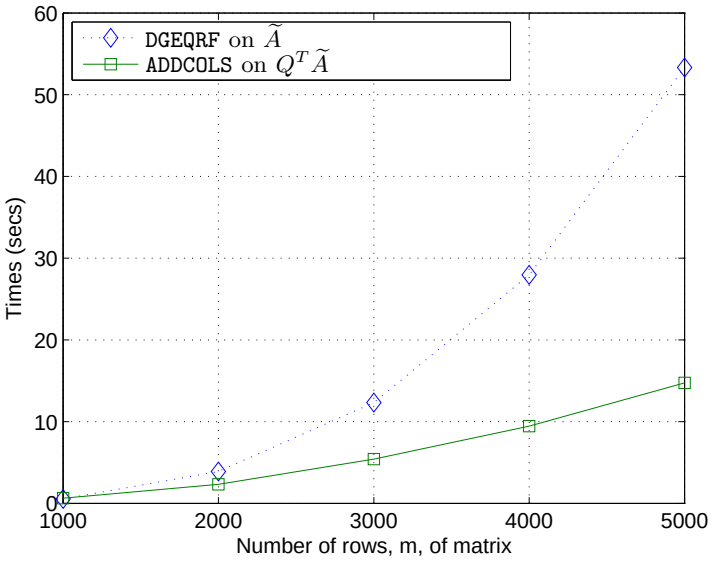

Fig. 4. Comparison of speed for `ADDCOLS` with $k = 1$ for different m.

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*. SIAM, 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. Sven Hammarling and Craig Lucas. Updating the QR factorization and the least squares problem, to appear. MIMS EPrint, Manchester Institute for Mathematical Sciences, University of Manchester, Manchester, UK, 2006.
5. N. J. Higham. Analysis of the Cholesky decomposition of a semidefinite matrix. In *Reliable Numerical Computation*, M. G. Cox and S. J. Hammarling, editors, Oxford University Press, 1990, pages 161–185.
6. Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second edition, SIAM, Philadelphia, PA, USA, 2002.
7. Craig Lucas. LAPACK-style codes for level 2 and 3 pivoted Cholesky factorizations. LAPACK Working Note 161, February 2004.
8. Robert Schreiber and Charles van Loan. A storage-efficient WY representation for products of householder transformations. *SIAM J. Sci. Stat. Comput.*, 10(1):53–57, 1989.