ISSN 1360-1725

THE UNIVERSITY
of MANCHESTER

LAPACK-Style Codes for Level 2 and 3 Pivoted Cholesky Factorizations

C. Lucas

Numerical Analysis Report No. 442

February 2004

Manchester Centre for Computational Mathematics
Numerical Analysis Reports

DEPARTMENTS OF MATHEMATICS

Reports available from:
Department of Mathematics
University of Manchester
Manchester M13 9PL
England

And over the World-Wide Web from URLs
`http://www.ma.man.ac.uk/nareports`
`ftp://ftp.ma.man.ac.uk/pub/narep`

LAPACK-Style Codes for Level 2 and 3 Pivoted Cholesky Factorizations

Craig Lucas*

February 5, 2004

Abstract

Fortran 77 codes exist in LAPACK for computing the Cholesky factorization of a symmetric positive definite matrix using Level 2 and 3 BLAS. In LINPACK there is a Level 1 BLAS routine for computing the Cholesky factorization with complete pivoting of a symmetric positive *semi*-definite matrix. We present two new algorithms and Fortran 77 LAPACK-style codes for computing this pivoted factorization: one using Level 2 BLAS and one using Level 3 BLAS. We show that on modern machines the new codes can be many times faster than the LINPACK code. Also, with a new stopping criterion they provide more reliable rank detection and can have a smaller normwise backward error.

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

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

*Department of Mathematics, University of Manchester, M13 9PL, England. (clucas@ma.man.ac.uk, http://www.ma.man.ac.uk/~clucas). This work was supported by an Engineering and Physical Sciences Research Council Research Studentship.

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.

In LINPACK the routine `xCHDC` performs the Cholesky factorization with complete pivoting, but it uses only Level 1 BLAS. For computational efficiency we would like a routine that exploits the Level 2 or Level 3 BLAS.

In this paper we describe a Gaxpy Level 2 BLAS algorithm for the positive definite case, and show how complete pivoting can be incorporated for the semi-definite case. 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 be pivoted. The Level 3 code calls the Level 2 code for small n. Finally we give some numerical experiments.

2 A Level 2 Gaxpy Algorithm

Comparing the jth columns in $A = LL^T$ we have [3]

$$A(:,j) = \sum_{k=1}^{j} L(:,k)L^T(k,j) = \sum_{k=1}^{j} L(j,k)L(:,k),$$

and therefore

$$L(j,j)L(:,j) = A(:,j) - \sum_{k=1}^{j-1} L(j,k)L(:,k)$$

Defining

$$v = A(:,j) - \sum_{k=1}^{j-1} L(j,k)L(:,k),$$

then if we know the first $j-1$ columns of L then v is computable.

Now, $L(1\!:\!j-1,j) = 0$ which implies $v(1\!:\!j-1) = 0$ and comparing terms we have that

$$L(j,j)^2 = v(j),$$

so we have finally

$$L(j\!:\!n,j) = v(j\!:\!n)/\sqrt{v(j)},$$

which leads to the following gaxpy-based algorithm, which we express in a MATLAB-like pseudo-code.

Algorithm 2.1 *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    % A is not positive definite
        end
        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
```

The (Level 2 BLAS) LAPACK [1] subroutine xPOTF2 uses this algorithm. It requires $n^3/3$ floating point operations.

3 A Level 2 Pivoted Gaxpy Algorithm

We can introduce pivoting into Algorithm 2.1, 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

$$q = \min\Big\{p : L(p,p) - d(p) = \max_{j\le i\le n}\{L(i,i) - d(i)\}\Big\}, \tag{3.1}$$

where d is a vector of dot products with

$$d(i) = L(i,1\colon j-1)L(i,1\colon j-1)^T, \quad i = j\colon n, \tag{3.2}$$

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 (3.2) and update them on each iteration. This approach gives the following pivoted gaxpy algorithm.

Algorithm 3.1 *This algorithm computes the pivoted Cholesky factorization with complete pivoting $P^TAP = LL^T$ of a symmetric positive semi-definite matrix $A \in \mathbb{R}^{n\times n}$, overwriting A with L. The nonzero elements of the permutation matrix P are given by $P(piv(k), k) = 1$, $k = 1\colon n$.*

Set L = lower triangular part of A
dots$(1\colon n) = 0$ % Store accumulated dot products
piv$= 1\colon n$
for $j = 1\colon n$
  if $j > 1$
    dots(i) =dots$(i) + L(i, j-1)^2, \quad i = j\colon n$
  end
  $q = \min\Big\{p : L(p,p)-\text{dots}(p) = \max_{j\le i\le n}\{L(i,i)-\text{ dots}(i)\}\Big\}$
(#) if stopping criterion is met
    return % computed rank of A is $j-1$
  end
  swap $L(j,:)$ and $L(q,:)$
  swap $L(:,j)$ and $L(:,q)$
  swap dots(j) and dots(q)
  swap piv(j) and piv(q)
  $L(j,j) = L(j,j)-$dots(j)
  $L(j,j) = \sqrt{L(j,j)}$
  % Update jth column
  if $1 < j < n$
    $L(j+1\colon n, j) = L(j+1\colon n, j) - L(j+1\colon n, 1\colon j-1)L(j, 1\colon j-1)^T$
  end
  if $j < n$
    $L(j+1\colon n, j) = L(j+1\colon n, j)/L(j,j)$
  end
end

The pivoting overhead is $3rn - 3/2r^2$ floating point operations and $r^2/2$ comparisons, where $r = \text{rank}(A)$.

The computed rank, $\hat{r}$, of A is determined by a stopping criterion at (#) in Algorithm 3.1. For a positive semi-definite matrix A, in exact arithmetic [3, Thm. 4.2.6]

$$a_{ii} = 0 \Rightarrow A(i,:) = 0,\ A(:,i) = 0.$$

Then at the jth iteration if the pivot, which we'll denote $\eta_{jj}^{(j)}$, is less than or equal to some tolerance, tol, then

$$tol \ge \eta_{jj}^{(j)} \ge \eta_{ii}^{(j)},\ i = j+1\colon n,$$

and we set the trailing matrix $L(j{:}n, j{:}n) = 0$ 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 xCHDC and the MATLAB function `cholp` in [4]. Here the algorithm is stopped on the kth step if

$$\eta_{ii}^{(k)} \le 0, \quad i = k{:}n. \tag{3.3}$$

In practice $\hat{r} > \mathrm{rank}(A)$ due to rounding errors.

In [6] the other two criteria are shown to work more effectively. The first is

$$\|\widetilde{S}_k\| \le \epsilon \|A\| \quad \text{or} \quad \eta_{ii}^{(k)} \le 0, \quad i = k{:}n, \tag{3.4}$$

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

$$\max_{k\le i\le n} \eta_{ii}^{(k)} \le \epsilon \eta_{11}^{(1)}, \tag{3.5}$$

where in both cases $\epsilon = nu$, and u is the unit roundoff. This is related to (3.4) in that if A and $\widetilde{S}_k$ are positive semi-definite then

$$\eta_{11}^{(1)} = \max_{i,j} |a_{ij}| \approx \|A\|_2, \quad \text{and,} \quad \max_{k\le i\le n} \eta_{ii}^{(k)} \approx \|\widetilde{S}_k\|_2.$$

We have used the latter criterion, preferred for its lower computational cost. See Appendix A for the double precision Fortran 77 code `lev2pchol.f`.

4 LAPACK's Level 3 Algorithm

The (Level 3 BLAS) LAPACK subroutine xPOTRF computes the Cholesky factorization of a block partitioned matrix. Starting with $L^{(0)} = A$, the algorithm computes the current block column of L using the previously computed blocks and does not update the trailing matrix. We have at the kth step

$$\begin{bmatrix} L_{11}^{(k-1)} & L_{12}^{(0)} & L_{13}^{(0)} \\ L_{21}^{(k-1)} & L_{22}^{(0)} & L_{23}^{(0)} \\ L_{31}^{(k-1)} & L_{32}^{(0)} & L_{33}^{(0)} \end{bmatrix},$$

where $L_{11}^{(k-1)} \in \mathbb{R}^{((k-1)n_b)\times((k-1)n_b)}$, for some block size n_b, is lower triangular and we wish to update the kth block column

$$\begin{bmatrix} L_{12}^{(0)} \\ L_{22}^{(0)} \\ L_{32}^{(0)} \end{bmatrix},$$

making $L_{22}^{(k)} \in \mathbb{R}^{n_b\times n_b}$ lower triangular and $L_{12}^{(k)}$ zero. The kth step is as follows:

set $L_{12}^{(k)} = 0$
$L_{22}^{(k)} = L_{22}^{(0)} - L_{21}^{(k-1)} L_{21}^{(k-1)^T}$
factorize $L_{22}^{(k)} = \widetilde{L}\widetilde{L}^T$
if this factorization fails
    A is not positive definite
else
    $L_{22}^{(k)} = \widetilde{L}$
    $L_{32}^{(k)} = L_{32}^{(0)} - L_{31}^{(k-1)} L_{21}^{(k-1)^T}$
    solve $XL_{22}^{(k)} = L_{32}^{(k)}$, for X
    $L_{32}^{(k)} = X$
end

In order to add pivoting to this algorithm we would need to decide all the pivots for the kth block column, carry out the required permutations, and continue with the step above.

The pivot for the first column can be found by computing all the possible diagonal elements as (3.1). To repeat this to find the second pivot we need first to update the vector of dot products, which can only be achieved by updating the first column of the kth block. So we have performed a complete step of Algorithm 3.1, before we find the second pivot. Thus in determining all the pivots for the current block column we will have formed $L_{22}^{(k)}$ and $L_{32}^{(k)}$ by Algorithm 3.1. We would like an algorithm with Level 3 operations that are independent of the pivoting.

5 A Level 3 Pivoted Algorithm

We can write for the semi-definite matrix $A^{(k)} \in \mathbb{R}^{(n-kn_b \times n-kn_b)}$ 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

$$A^{(k)} = A_{22}^{(k-1)} - L_{21} L_{21}^T, \tag{5.1}$$

which we can explicitly form, taking advantage of symmetry.

From this representation we can derive a block algorithm. We start with $A^{(0)} = A$ and a block size n_b. At the kth step we apply the equivalent of n_b steps of Algorithm 3.1 to $A^{(k-1)}$ to form n_b columns of L. We then update the trailing

matrix $A^{(k)}$, which is of dimension $n-k*n_b$, according to (5.1). This is a pivoted algorithm, as Algorithm 3.1 acts on the whole trailing matrix.

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

Algorithm 5.1 *This algorithm computes the pivoted Cholesky factorization with complete pivoting $P^TAP = LL^T$ of a symmetric positive semi-definite matrix $A \in \mathbb{R}^{n\times n}$, overwriting A with L, using a Level 3 update and block size n_b. The nonzero elements of the permutation matrix P are given by $P(piv(k),k) = 1$, $k = 1\colon n$.*

Set L = lower triangular part of A
$\epsilon = nu$
$piv = 1\colon n$
for $k = 1\colon n_b\colon n$
  $jb = \min(n_b, n-k+1)$ % Allow for last incomplete block
  $dots(k\colon n) = 0$ % Store accumulated dot products
  $tol = n*u*\max(\mathrm{diag}(A))$ % Tolerance in stopping criterion
  for $j = k\colon k+jb-1$
    if $j > k$
      $dots(i) = dots(i) + L(i,j-1)^2, \quad i = j\colon n$
    end
    $q = \min\left\{p : L(p,p) - dots(p) = \max_{j\le i\le n}\{L(i,i) - dots(i)\}\right\}$
    if $L(q,q) \le tol$
      return % computed rank of A is $j-1$
    end
    swap $L(j,:)$ and $L(q,:)$
    swap $L(:,j)$ and $L(:,q)$
    swap $dots(j)$ and $dots(q)$
    swap $piv(j)$ and $piv(q)$
    $L(j,j) = L(j,j) - dots(j)$
    $L(j,j) = \sqrt{L(j,j)}$
    % Update jth column
    if $1 < j < n$
      $L(j+1\colon n, j) = L(j+1\colon n, j) -$
        $L(j+1\colon n, 1\colon j-1)L(j, 1\colon j-1)^T$
    end
    if $j < n$

$L(j+1{:}n, j) = L(j+1{:}n, j)/L(j, j)$
end
end
if $k + jb < n$
% perform Level 3 update
$L(j+1{:}n, j+1, n) = L(j+1{:}n, j+1, n) -$
$L(j+1{:}n, 1{:}j)L(j+1{:}n, 1{:}j)^T$
end

See Appendix B for the double precision Fortran 77 code `lev3pchol.f`.

6 Numerical Experiments

We tested and compared four Fortran subroutines:

- LINPACK's `DCHDC` [2], which uses Level 1 BLAS and stopping criterion (3.3).
- LINPACK's `DCHDC`, altered to use stopping criterion (3.5).
- An implementation of Algorithm 3.1, obtained by modifying LAPACK's `DPOTF2`, using stopping criterion (3.5): `lev2pchol.f` in Appendix A.
- An implementation of Algorithm 5.1, again using stopping criterion (3.5): `lev3pchol.f` in Appendix B.

The tests were performed on a 1400MHz AMD Athlon running Red Hat Linux version 6.2 with kernel 2.2.22 and a Sun 167MHz UltraSparc running Solaris version 2.7. All test matrices were generated in MATLAB 6.5. The unit roundoff $u \approx$ 1.1e-16.

We test the speed of computation, the normwise backward error and the rank revealing properties of the routines. In all cases the results for the Linux machine are given. The results on the Sun for backward error and rank detection were indistinguishable from those on the Linux machine. The timings on the Sun were much greater than, but in proportion to, those on the Linux machine.

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 semi-definite matrices of order n and rank $0.7n$ by computing

$$A = \sum_{i=1}^{0.7n} xx^T, \quad x = \mathrm{rand}(n, 1),$$

where the MATLAB command rand(n,1) generates a random vector, with elements uniformly distributed on $(0, 1)$, of length n. For each value of n the codes

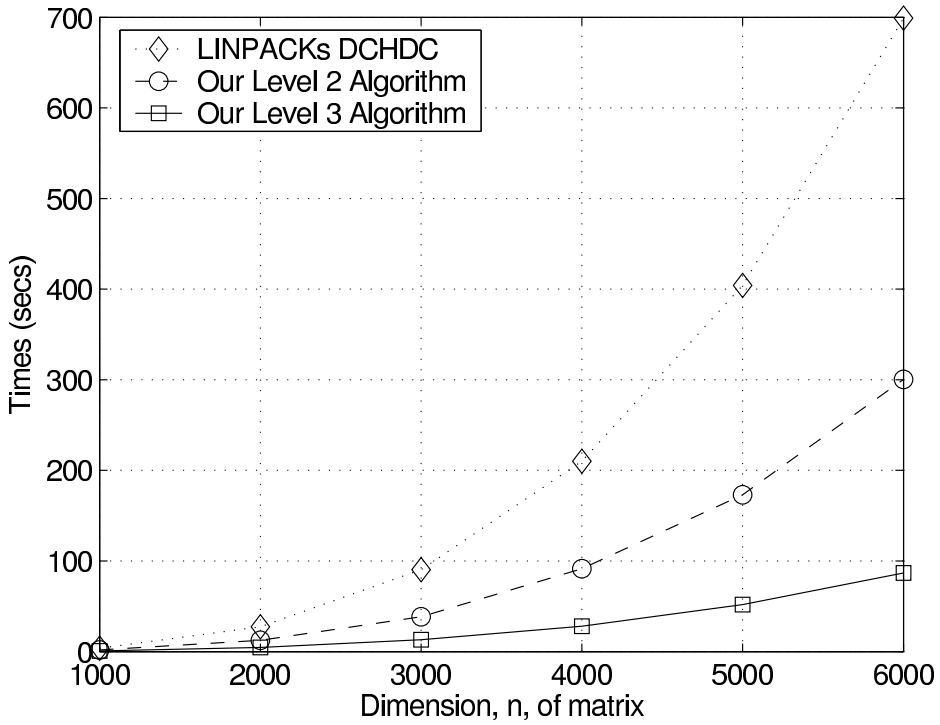

Figure 6.1: Comparison of speed for different n.

were run four times and the mean times are shown in Figure 6.1. The speed-ups of the new codes over the LINPACK code are given in Table 6.1.

We achieve a good speed-up, with the Level 3 code as much as 8 times faster than the LINPACK code.

Table 6.1: Speed-ups from LINPACK code.

n	1000	2000	3000	4000	5000	6000
LEV2PCHOL	2.05	2.21	2.35	2.29	2.33	2.32
LEV3PCHOL	5.30	6.03	6.90	7.52	7.78	8.05

We also compared 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. Figure 6.2 shows the ratio of speed of the pivoted codes to the unpivoted codes. These show that the pivoting overhead is negligible for large n, recalling the pivoting overhead is $3rn - 3/2r^2$ floating point operations,

and the unpivoted factorization is of order n^3. The use of the pivoted codes could be warranted if there is any doubt over whether a matrix is positive definite..

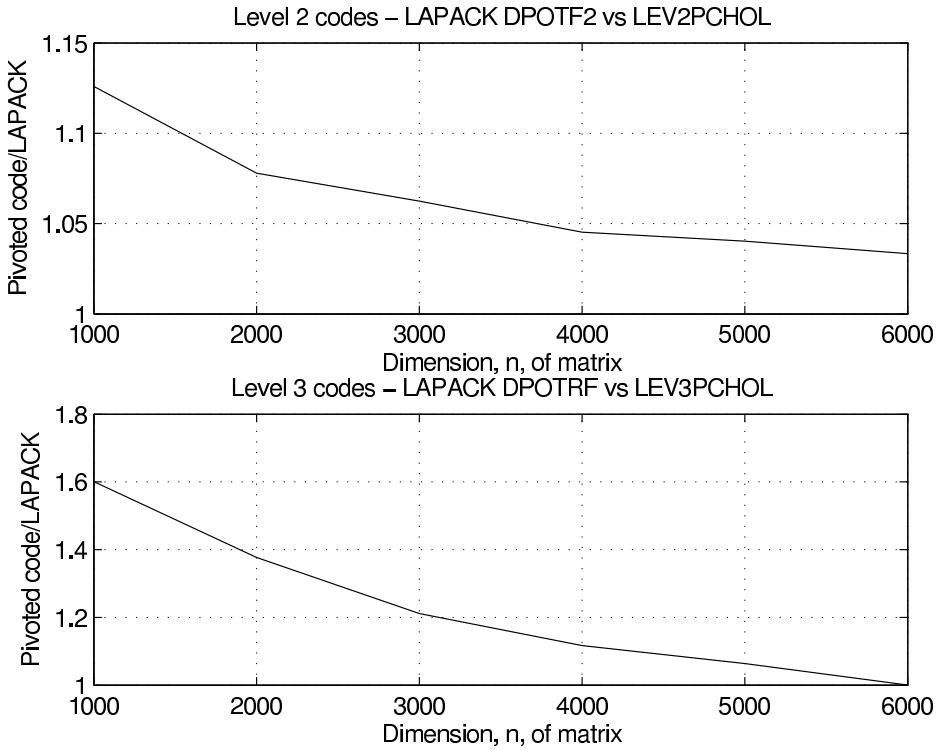

Figure 6.2: Comparison between LAPACK and pivoted codes.

We tested all four subroutines on a further set of random positive semi-definite matrices, this time with pre-determined eigenvalues, similar to the tests in [5]. The matrices were generated by setting $A = Q\Lambda Q^T$ where Q was a random orthogonal matrix computed by the method of [9] using [7]. 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:

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

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 $\widehat{P}$. We have, from [6, Thm. 10.22], the following upper bound

$$\frac{\|A - \widehat{P}\widehat{L}\widehat{L}^T\widehat{P}^T\|_2}{\|A\|_2} \leq 2r\gamma_{r+1}(\|W\|_2 + 1)^2 + O(u^2), \tag{6.1}$$

where

$$W = \widehat{L}_{11}^{-1}\widehat{L}_{12}, \quad \widehat{L} = \begin{bmatrix} \widehat{L}_{11} & 0 \\ \widehat{L}_{12} & 0 \end{bmatrix}, \quad \widehat{L}_{11} \in \mathbb{R}^{r\times r},$$

and from [6, Lemma 10.13],

$$\|W\|_2 \leq \sqrt{\frac{1}{3}(n-r)(4^r - 1)}, \tag{6.2}$$

and so there is no guarantee of stability of the algorithm for large n and r.

There was little difference for the normwise backward errors between the three cases and Table 6.2 shows minimum and maximum values for different n. The codes with the new stopping criterion give smaller errors than the original LINPACK code. The minimum and maximum values of $\|W\|$ are also given, and show that after r stages the algorithm will have produced a stable factorization. In fact, for all the codes with our stopping criterion $\hat{r} = r$, and the so rank was detected exactly. This was not the case for the unmodified `DCHDC`, and the error, $\hat{r} - r$, is shown in Table 6.3.

We assume that the larger backward error for the original `DCHDC` is due to the stopping criterion. As Table 6.3 shows, the routine is terminated after more steps than our codes, adding more nonzero columns to $\widehat{L}$.

7 Checking for Indefiniteness

The algorithm does not attempt to check if the matrix is positive semi-definite. For example, if we supplied

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

then the algorithm would stop after one step, and give the rank to be 1, whereas A is indefinite with eigenvalues of $\{1, 1, -1\}$. A check of the residual

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

Table 6.2: Comparison of normwise backward errors.

	n	70	100	200	500	1000
$\|W\|_2$	min	3.58	4.39	7.91	15.12	27.11
	max	10.67	12.26	20.62	32.52	66.03
`DCHDC`	min	1.654e-16	3.654e-16	6.651e- 16	5.504e-15	1.933e-14
	max	3.172e-13	1.498e-13	1.031e-12	2.823e-12	4.737e-11
`DCHDC` with (3.5)	min	1.707e-16	2.561e-16	4.737e-16	1.273e-15	2.687e-15
	max	7.778e-15	9.014e-15	1.810e-14	7.746e-14	1.991e-13
`LEV2PCHOL`	min	1.671e-16	2.526e-16	5.121e-16	1.240e-15	2.597e-15
	max	4.633e-15	9.283e-15	1.458e-14	7.290e-14	1.983e-13
`LEV3PCHOL`	min	1.671e-16	2.476e-16	5.121e-16	1.271e-15	2.600e-15
	max	4.633e-15	9.283e-15	1.710e-14	8.247e-14	2.049e-13

Table 6.3: Errors in computed rank for DCHDC.

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

bearing in mind (6.1) and (6.2), may allow confirmation that A is not close to being positive semi-definite.

The user could also update the trailing matrix by (5.1) to give

$$\widetilde{A}^{r+1} = A^{(r+1)}(r+1\colon n, r+1\colon n) = A^{(r)}(r+1\colon n, r+1\colon n) - \widehat{L}_{21}\widehat{L}_{21}^T.$$

We know the diagonal elements are 'small' since the algorithm stopped after $\hat{r}$ steps and consequently $\|\widetilde{A}^{r+1}\|$ should be negligible, but if $\|\widetilde{A}^{r+1}\|$ is large then

the original matrix could have been indefinite. However, if $\|W\|$ is large then it is more difficult to reach this conclusion, as there may have been significant error growth during the factorization.

Of course, if there is serious doubt over semi-definiteness then a symmetric indefinite factorization should be used. It is also worth noting that the stopping criterion is based on A being positive semi-definite.

8 Concluding Remarks

We have presented two Fortran 77 codes for the Cholesky factorization with complete pivoting of a positive semi-definite matrix: a Level 2 BLAS version and a Level 3 BLAS version. Our tests show that our codes are much faster than the existing LINPACK code on our test machines.

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

We propose that the double precision Fortran 77 codes `lev2pchol.f` and `lev3pchol.f`, and their single precision and complex equivalents, be included in LAPACK.

9 Acknowledgements

I thank Sven Hammarling of NAG Ltd., Oxford and Nick Higham of the University of Manchester for their invaluable comments and suggestions throughout this work.

Appendices

The Level 3 code calls the Level 2 code when the block size is greater than n. The block size is determined by the LAPACK function `ILAENV`. Note we pass the function name `DPOTRF`, the existing Level 3 LAPACK routine for the full rank Cholesky factorization, to return a suitable value.

The subroutine `BLAS_DMAX_VAL` is modified from the BLAS function `IDAMAX` to return the largest algebraic value of a vector and the smallest index that contains that value. The name and interface conform to the BLAS Technical Forum Standard [8].

A lev2pchol.f

```
      SUBROUTINE LEV2PCHOL( UPLO, N, A, LDA, PIV, RANK, TOL, WORK,
     $                      INFO )
*
*     Modified to include pivoting for semidefinite matrices by
*     Craig Lucas, University of Manchester. January, 2004
*
*     Original LAPACK routine DPOTF2
*     Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*     Courant Institute, Argonne National Lab, and Rice University
*     February 29, 1992
*
*     .. Scalar Arguments ..
      DOUBLE PRECISION   TOL
      INTEGER            INFO, LDA, N, RANK
      CHARACTER          UPLO
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), WORK( 2*N )
      INTEGER            PIV( N )
*     ..
*
*  Purpose
*  =======
*
*  LEV2PCHOL computes the Cholesky factorization with complete
*  pivoting of a real symmetric positive semi-definite matrix A.
*
*  The factorization has the form
*     P' * A * P = U' * U ,  if UPLO = 'U',
*     P' * A * P = L  * L',  if UPLO = 'L',
*  where U is an upper triangular matrix and L is lower triangular, and
```

```
*  P is stored as vector PIV.
*
*  This algorithm does not attempt to check that A is positive
*  semi-definite. This version of the algorithm calls level 2 BLAS.
*
*  Arguments
*  =========
*
*  UPLO    (input) CHARACTER*1
*          Specifies whether the upper or lower triangular part of the
*          symmetric matrix A is stored.
*          = 'U':  Upper triangular
*          = 'L':  Lower triangular
*
*  N       (input) INTEGER
*          The order of the matrix A.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the symmetric matrix A.  If UPLO = 'U', the leading
*          n by n upper triangular part of A contains the upper
*          triangular part of the matrix A, and the strictly lower
*          triangular part of A is not referenced.  If UPLO = 'L', the
*          leading n by n lower triangular part of A contains the lower
*          triangular part of the matrix A, and the strictly upper
*          triangular part of A is not referenced.
*
*          On exit, if INFO = 0, the factor U or L from the Cholesky
*          factorization as above.
*
*  PIV     (output) INTEGER array, dimension (N)
*          PIV is such that the nonzero entries are P( PIV( K ), K ) = 1.
*
*  RANK    (output) INTEGER
*          The rank of A given by the number of steps the algorithm
*          completed.
*
*  TOL     (input) DOUBLE PRECISION
*          User defined tolerance. If TOL < 0, then N*U*MAX( A(k,k) )
*          will be used. The algorithm terminates at the (k-1)st step
*          if the pivot <= TOL.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,N).
*
*  WORK    DOUBLE PRECISION array, dimension (2*N)
```

```
*          Work space.
*
*  INFO    (output) INTEGER
*          < 0: if INFO = -k, the k-th argument had an illegal value
*          = 0  algorithm completed successfully.
*
*  =====================================================================
*
*     .. Parameters ..
      DOUBLE PRECISION   ONE, ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*     ..
*     .. Local Scalars ..
      DOUBLE PRECISION   AJJ, DSTOP, DTEMP, U
      INTEGER            ITEMP, J, P, PVT
      LOGICAL            UPPER
*     ..
*     .. External Functions ..
      DOUBLE PRECISION   DLAMCH
      LOGICAL            LSAME
      EXTERNAL           DLAMCH, LSAME
*     ..
*     .. External Subroutines ..
      EXTERNAL           BLAS_DMAX_VAL, DGEMV, DSCAL, DSWAP, XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          MAX, SQRT
*     ..
*
*     Test the input parameters
*
      INFO = 0
      UPPER = LSAME( UPLO, 'U' )
      IF( .NOT.UPPER .AND. .NOT.LSAME( UPLO, 'L' ) ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
         INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'LEV2PCHOL', -INFO )
         RETURN
      END IF
*
```

```
*     Quick return if possible
*
      IF( N.EQ.0 )
     $   RETURN
*
*     Initialize PIV
*
      DO 10 P = 1, N
         PIV( P ) = P
   10 CONTINUE
*
*     Get unit roundoff
*
      U = DLAMCH( 'E' )
*
*     Compute stopping value
*
      CALL BLAS_DMAX_VAL( N, A( 1, 1 ), LDA+1, PVT, DTEMP )
      AJJ = A( PVT, PVT )
      IF( AJJ.EQ.ZERO ) THEN
         RANK = 0
         GO TO 80
      END IF
*
*     Compute stopping value if not supplied
*
      IF( TOL.LT.ZERO ) THEN
         DSTOP = N*U*AJJ
      ELSE
         DSTOP = TOL
      END IF
*
*     Set first half of WORK to zero, holds dot products
*
      DO 20 P = 1, N
         WORK( P ) = 0
   20 CONTINUE
*
      IF( UPPER ) THEN
*
*        Compute the Cholesky factorization P' * A * P = U' * U
*
         DO 40 J = 1, N
*
*        Find pivot, test for exit, else swap rows and columns
```

```
*        Update dot products, compute possible pivots which are
*        stored in the second half of WORK
*
            DO 30 P = J, N
*
               IF( J.GT.1 ) THEN
                  WORK( P ) = WORK( P ) + A( J-1, P )**2
               END IF
               WORK( N+P ) = A( P, P ) - WORK( P )
*
   30       CONTINUE
*
            IF( J.GT.1 ) THEN
               CALL BLAS_DMAX_VAL( N-J+1, WORK( N+J ), 1, ITEMP, DTEMP )
               PVT = ITEMP + J - 1
               AJJ = WORK( N+PVT )
               IF( AJJ.LE.DSTOP ) THEN
                  A( J, J ) = AJJ
                  GO TO 70
               END IF
            END IF
*
            IF( J.NE.PVT ) THEN
*
*              Pivot OK, so can now swap pivot rows and columns
*
               A( PVT, PVT ) = A( J, J )
               CALL DSWAP( J-1, A( 1, J ), 1, A( 1, PVT ), 1 )
               CALL DSWAP( N-PVT, A( J, PVT+1 ), LDA, A( PVT, PVT+1 ),
     $                     LDA )
               CALL DSWAP( PVT-J-1, A( J, J+1 ), LDA, A( J+1, PVT ), 1 )
*
*              Swap dot products and PIV
*
               DTEMP = WORK( J )
               WORK( J ) = WORK( PVT )
               WORK( PVT ) = DTEMP
               ITEMP = PIV( PVT )
               PIV( PVT ) = PIV( J )
               PIV( J ) = ITEMP
            END IF
*
            AJJ = SQRT( AJJ )
            A( J, J ) = AJJ
*
```

```
*           Compute elements J+1:N of row J
*
            IF( J.LT.N ) THEN
               CALL DGEMV( 'Trans', J-1, N-J, -ONE, A( 1, J+1 ), LDA,
     $                     A( 1, J ), 1, ONE, A( J, J+1 ), LDA )
               CALL DSCAL( N-J, ONE / AJJ, A( J, J+1 ), LDA )
            END IF
*
   40    CONTINUE
*
      ELSE
*
*        Compute the Cholesky factorization P' * A * P = L * L'
*
         DO 60 J = 1, N
*
*        Find pivot, test for exit, else swap rows and columns
*        Update dot products, compute possible pivots which are
*        stored in the second half of WORK
*
            DO 50 P = J, N
*
               IF( J.GT.1 ) THEN
                  WORK( P ) = WORK( P ) + A( P, J-1 )**2
               END IF
               WORK( N+P ) = A( P, P ) - WORK( P )
*
   50       CONTINUE
*
            IF( J.GT.1 ) THEN
               CALL BLAS_DMAX_VAL( N-J+1, WORK( N+J ), 1, ITEMP, DTEMP )
               PVT = ITEMP + J - 1
               AJJ = WORK( N+PVT )
               IF( AJJ.LE.DSTOP ) THEN
                  A( J, J ) = AJJ
                  GO TO 70
               END IF
            END IF
*
            IF( J.NE.PVT ) THEN
*
*              Pivot OK, so can now swap pivot rows and columns
*
               A( PVT, PVT ) = A( J, J )
               CALL DSWAP( J-1, A( J, 1 ), LDA, A( PVT, 1 ), LDA )
```

```
               CALL DSWAP( N-PVT, A( PVT+1, J ), 1, A( PVT+1, PVT ), 1 )
               CALL DSWAP( PVT-J-1, A( J+1, J ), 1, A( PVT, J+1 ), LDA )
*
*              Swap dot products and PIV
*
               DTEMP = WORK( J )
               WORK( J ) = WORK( PVT )
               WORK( PVT ) = DTEMP
               ITEMP = PIV( PVT )
               PIV( PVT ) = PIV( J )
               PIV( J ) = ITEMP
            END IF
*
            AJJ = SQRT( AJJ )
            A( J, J ) = AJJ
*
*           Compute elements J+1:N of column J
*
            IF( J.LT.N ) THEN
               CALL DGEMV( 'No tran', N-J, J-1, -ONE, A( J+1, 1 ), LDA,
     $                     A( J, 1 ), LDA, ONE, A( J+1, J ), 1 )
               CALL DSCAL( N-J, ONE / AJJ, A( J+1, J ), 1 )
            END IF
*
   60    CONTINUE
*
      END IF
*
*     Ran to completion, A has full rank
*
      RANK = N
*
      GO TO 80
   70 CONTINUE
*
*     Rank is number of steps completed
*
      RANK = J - 1
*
   80 CONTINUE
      RETURN
*
*     End of LEV2PCHOL
*
      END
```

B lev3pchol.f

```
      SUBROUTINE LEV3PCHOL( UPLO, N, A, LDA, PIV, RANK, TOL, WORK,
     $                      INFO )
*
*     Craig Lucas, University of Manchester. January, 2004
*     Some code taken from LAPACK routine DPOTF2
*
*     .. Scalar Arguments ..
      DOUBLE PRECISION   TOL
      INTEGER            INFO, LDA, N, RANK
      CHARACTER          UPLO
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), WORK( 2*N )
      INTEGER            PIV( N )
*     ..
*
*  Purpose
*  =======
*
*  LEV3PCHOL computes the Cholesky factorization with complete
*  pivoting of a real symmetric positive semi-definite matrix A.
*
*  The factorization has the form
*     P' * A * P = U' * U ,  if UPLO = 'U',
*     P' * A * P = L  * L',  if UPLO = 'L',
*  where U is an upper triangular matrix and L is lower triangular, and
*  P is stored as vector PIV.
*
*  This algorithm does not attempt to check that A is positive
*  semi-definite. This version of the algorithm calls level 3 BLAS.
*
*  Arguments
*  =========
*
*  UPLO    (input) CHARACTER*1
*          Specifies whether the upper or lower triangular part of the
*          symmetric matrix A is stored.
*          = 'U':  Upper triangular
*          = 'L':  Lower triangular
*
*  N       (input) INTEGER
*          The order of the matrix A.  N >= 0.
*
```

```
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the symmetric matrix A.  If UPLO = 'U', the leading
*          n by n upper triangular part of A contains the upper
*          triangular part of the matrix A, and the strictly lower
*          triangular part of A is not referenced.  If UPLO = 'L', the
*          leading n by n lower triangular part of A contains the lower
*          triangular part of the matrix A, and the strictly upper
*          triangular part of A is not referenced.
*
*          On exit, if INFO = 0, the factor U or L from the Cholesky
*          factorization as above.
*
*  PIV     (output) INTEGER array, dimension (N)
*          PIV is such that the nonzero entries are P( PIV( K ), K ) = 1.
*
*  RANK    (output) INTEGER
*          The rank of A given by the number of steps the algorithm
*          completed.
*
*  TOL     (input) DOUBLE PRECISION
*          User defined tolerance. If TOL < 0, then N*U*MAX( A(k,k) )
*          will be used. The algorithm terminates at the (k-1)st step
*          if the pivot <= TOL.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,N).
*
*  WORK    DOUBLE PRECISION array, dimension (2*N)
*          Work space.
*
*  INFO    (output) INTEGER
*          < 0: if INFO = -k, the k-th argument had an illegal value
*          = 0  algorithm completed successfully.
*
*  =====================================================================
*
*     .. Parameters ..
      DOUBLE PRECISION   ONE, ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*     ..
*     .. Local Scalars ..
      DOUBLE PRECISION   AJJ, DSTOP, DTEMP, U
      INTEGER            ITEMP, J, JB, K, NB, P, PVT
      LOGICAL            UPPER
*     ..
```

```
*     .. External Functions ..
      REAL               DLAMCH
      INTEGER            ILAENV
      LOGICAL            LSAME
      EXTERNAL           DLAMCH, ILAENV, LSAME
*     ..
*     .. External Subroutines ..
      EXTERNAL           BLAS_DMAX_VAL, DGEMV, DSCAL, DSWAP, DSYRK,
     $                   LEV2PCHOL, XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN, SQRT
*     ..
*
*     Test the input parameters.
*
      INFO = 0
      UPPER = LSAME( UPLO, 'U' )
      IF( .NOT.UPPER .AND. .NOT.LSAME( UPLO, 'L' ) ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
         INFO = -4
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'LEV3PCHOL', -INFO )
         RETURN
      END IF
*
*     Quick return if possible
*
      IF( N.EQ.0 )
     $   RETURN
*
*     Get block size
*
      NB = ILAENV( 1, 'DPOTRF', UPLO, N, -1, -1, -1 )
      IF( NB.LE.1 .OR. NB.GE.N ) THEN
*
*        Use unblocked code
*
         CALL LEV2PCHOL( UPLO, N, A( 1, 1 ), LDA, PIV, RANK, TOL, WORK,
     $                   INFO )
         GO TO 110
```

```
*
      ELSE
*
*     Initialize PIV
*
         DO 10 P = 1, N
            PIV( P ) = P
   10    CONTINUE
*
*     Get unit roundoff
*
         U = DLAMCH( 'E' )
*
*     Compute stopping value
*
         CALL BLAS_DMAX_VAL( N, A( 1, 1 ), LDA+1, PVT, DTEMP )
         AJJ = A( PVT, PVT )
         IF( AJJ.EQ.ZERO ) THEN
            RANK = 0
            GO TO 110
         END IF
*
*     Compute stopping value if not supplied
*
         IF( TOL.LT.ZERO ) THEN
            DSTOP = N*U*AJJ
         ELSE
            DSTOP = TOL
         END IF
*
*
         IF( UPPER ) THEN
*
*           Compute the Cholesky factorization P' * A * P = U' * U
*
            DO 50 K = 1, N, NB
*
*              Account for last block not being NB wide
*
               JB = MIN( NB, N-K+1 )
*
*              Set relevant part of first half of WORK to zero,
*              holds dot products
*
               DO 20 P = K, N
```

```
                  WORK( P ) = 0
   20          CONTINUE
*
               DO 40 J = K, K + JB - 1
*
*              Find pivot, test for exit, else swap rows and columns
*              Update dot products, compute possible pivots which are
*              stored in the second half of WORK
*
                  DO 30 P = J, N
*
                     IF( J.GT.K ) THEN
                        WORK( P ) = WORK( P ) + A( J-1, P )**2
                     END IF
                     WORK( N+P ) = A( P, P ) - WORK( P )
*
   30             CONTINUE
*
                  IF( J.GT.1 ) THEN
                     CALL BLAS_DMAX_VAL( N-J+1, WORK( N+J ), 1, ITEMP,
     $                                   DTEMP )
                     PVT = ITEMP + J - 1
                     AJJ = WORK( N+PVT )
                     IF( AJJ.LE.DSTOP ) THEN
                        A( J, J ) = AJJ
                        GO TO 100
                     END IF
                  END IF
*
                  IF( J.NE.PVT ) THEN
*
*                    Pivot OK, so can now swap pivot rows and columns
*
                     A( PVT, PVT ) = A( J, J )
                     CALL DSWAP( J-1, A( 1, J ), 1, A( 1, PVT ), 1 )
                     CALL DSWAP( N-PVT, A( J, PVT+1 ), LDA,
     $                           A( PVT, PVT+1 ), LDA )
                     CALL DSWAP( PVT-J-1, A( J, J+1 ), LDA,
     $                           A( J+1, PVT ), 1 )
*
*                    Swap dot products and PIV
*
                     DTEMP = WORK( J )
                     WORK( J ) = WORK( PVT )
                     WORK( PVT ) = DTEMP
```

```
                  ITEMP = PIV( PVT )
                  PIV( PVT ) = PIV( J )
                  PIV( J ) = ITEMP
               END IF
*
               AJJ = SQRT( AJJ )
               A( J, J ) = AJJ
*
*              Compute elements J+1:N of row J.
*
               IF( J.LT.N ) THEN
                  CALL DGEMV( 'Trans', J-K, N-J, -ONE, A( K, J+1 ),
     $                        LDA, A( K, J ), 1, ONE, A( J, J+1 ),
     $                        LDA )
                  CALL DSCAL( N-J, ONE / AJJ, A( J, J+1 ), LDA )
               END IF
*
   40          CONTINUE
*
*              Update trailing matrix,  J already incremented
*
               IF( K+JB.LE.N ) THEN
                  CALL DSYRK( 'Upper', 'Trans', N-J+1, JB, -ONE,
     $                        A( J, K ), LDA, ONE, A( J, J ), LDA )
               END IF
*
   50       CONTINUE
*
         ELSE
*
*        Compute the Cholesky factorization P' * A * P = L * L'
*
            DO 90 K = 1, N, NB
*
*              Account for last block not being NB wide
*
               JB = MIN( NB, N-K+1 )
*
*              Set relevant part of first half of WORK to zero,
*              holds dot products
*
               DO 60 P = K, N
                  WORK( P ) = 0
   60          CONTINUE
*
```

```
            DO 80 J = K, K + JB - 1
*
*           Find pivot, test for exit, else swap rows and columns
*           Update dot products, compute possible pivots which are
*           stored in the second half of WORK
*
               DO 70 P = J, N
*
                  IF( J.GT.K ) THEN
                     WORK( P ) = WORK( P ) + A( P, J-1 )**2
                  END IF
                  WORK( N+P ) = A( P, P ) - WORK( P )
*
   70          CONTINUE
*
               IF( J.GT.1 ) THEN
                  CALL BLAS_DMAX_VAL( N-J+1, WORK( N+J ), 1, ITEMP,
     $                                DTEMP )
                  PVT = ITEMP + J - 1
                  AJJ = WORK( N+PVT )
                  IF( AJJ.LE.DSTOP ) THEN
                     A( J, J ) = AJJ
                     GO TO 100
                  END IF
               END IF
*
               IF( J.NE.PVT ) THEN
*
*                 Pivot OK, so can now swap pivot rows and columns
*
                  A( PVT, PVT ) = A( J, J )
                  CALL DSWAP( J-1, A( J, 1 ), LDA, A( PVT, 1 ), LDA )
                  CALL DSWAP( N-PVT, A( PVT+1, J ), 1,
     $                        A( PVT+1, PVT ), 1 )
                  CALL DSWAP( PVT-J-1, A( J+1, J ), 1, A( PVT, J+1 ),
     $                        LDA )
*
*                 Swap dot products and PIV
*
                  DTEMP = WORK( J )
                  WORK( J ) = WORK( PVT )
                  WORK( PVT ) = DTEMP
                  ITEMP = PIV( PVT )
                  PIV( PVT ) = PIV( J )
                  PIV( J ) = ITEMP
```

```
                  END IF
*
                  AJJ = SQRT( AJJ )
                  A( J, J ) = AJJ
*
*                 Compute elements J+1:N of column J.
*
                  IF( J.LT.N ) THEN
                     CALL DGEMV( 'No tran', N-J, J-K, -ONE, A( J+1, K ),
     $                           LDA, A( J, K ), LDA, ONE, A( J+1, J ),
     $                           1 )
                     CALL DSCAL( N-J, ONE / AJJ, A( J+1, J ), 1 )
                  END IF
*
   80          CONTINUE
*
*             Update trailing matrix, J already incremented
*
               IF( K+JB.LE.N ) THEN
                  CALL DSYRK( 'Lower', 'No Trans', N-J+1, JB, -ONE,
     $                        A( J, K ), LDA, ONE, A( J, J ), LDA )
               END IF
*
   90       CONTINUE
*
         END IF
      END IF
*
*     Ran to completion, A has full rank
*
      RANK = N
*
      GO TO 110
  100 CONTINUE
*
*     Rank is the number of steps completed
*
      RANK = J - 1
*
  110 CONTINUE
      RETURN
*
*     End of LEV3PCHOL
*
      END
```

C blas_dmax_val.f

```
      SUBROUTINE BLAS_DMAX_VAL( N, X, INCX, K, R )
*
*     BLAS_DMAX_VAL finds the largest component of x, r, and
*     determines the smallest index, k, such that x(k) = r.
*     Craig Lucas, University of Manchester. June, 2003
*
*     Modified from the BLAS function IDAMAX:
*     Jack dongarra, LINPACK, 3/11/78.
*
*     .. Scalar Arguments ..
      DOUBLE PRECISION   R
      INTEGER            INCX, K, N
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   X( * )
*     ..
*     .. Local Scalars ..
      INTEGER            I, IX
*     ..
      K = 0
      IF( N.LT.1 .OR. INCX.LE.0 )
     $   RETURN
      K = 1
      IF( N.EQ.1 )
     $   RETURN
      IF( INCX.EQ.1 )
     $   GO TO 30
*
*     Code for increment not equal to 1
*
      IX = 1
      R = X( 1 )
      IX = IX + INCX
      DO 20 I = 2, N
         IF( X( IX ).LE.R )
     $      GO TO 10
         K = I
         R = X( IX )
   10    CONTINUE
         IX = IX + INCX
   20 CONTINUE
      RETURN
*
```

```
*     Code for increment equal to 1
*
   30 CONTINUE
      R = X( 1 )
      DO 40 I = 2, N
         IF( X( I ).LE.R )
     $      GO TO 40
         K = I
         R = X( I )
   40 CONTINUE
      RETURN
      END
```

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. 320 pp. ISBN 0-89871-172-X.

[3] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Third edition, The Johns Hopkins University Press, Baltimore, MD, USA, 1996. xxx + 698 pp. ISBN 0-8018-5413-X, 0-8018-5414-8.

[4] Nicholas J. Higham. The Matrix Computation Toolbox. `http://www.ma.man.ac.uk/~higham/mctoolbox`.

[5] Nicholas J. Higham. Analysis of the Cholesky decomposition of a semi-definite 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, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. xxx+680 pp. ISBN 0-89871-521-0.

[7] `/matlab6.5/toolbox/matlab/elmat/private/qmult.m` from the MATLAB distibution.

[8] BLAS Technical Forum Standard. `http://www.netlib.org/blas/blast-forum/`.

[9] G. W. Stewart. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal on Numerical Analysis*, 17(3):403–409, 1980.