Simple, Fast and Practicable Algorithms for Cholesky, LU and QR Decomposition Using Fast Rectangular Matrix Multiplication

Cristóbal Camarero

Department of Computer Science and Electronics

Universidad de Cantabria, UNICAN, Spain.

Abstract

This note presents fast Cholesky/LU/QR decomposition algorithms with $O(n^{2.529})$ time complexity when using the fastest known matrix multiplication. The algorithms have potential application, since a quickly made implementation using Strassen multiplication has lesser execution time than the employed by the GNU Scientific Library for the same task in at least a few examples.

The underlaying ideas are very simple. Despite this, I have been unable to find these methods in the literature.

1 Introduction

Cholesky and LU decompositions are used everywhere to solve linear systems; and they have multiple applications beyond that. The problem is given a positive-definite matrix A to find two triangular matrices L and U such that $A = LU$, L being lower triangular and U being upper triangular. When A is symmetric, it is hold that $U = L^T$ after normalization, and $A = LL^T$ is called a Cholesky decomposition. In the QR decomposition the problems is to find orthogonal Q and upper triangular R such that $A = QR$. The new presented algorithms reduce their time complexity from $O(n^3)$ to $O(n^{2.529})$. Although in [2] it was presented an algorithm for LU decomposition with lower complexity, it was not very practical because of the large constants involved. In contrast, the present algorithms are very practical, since they have no additional large constants respect to the typical Cholesky/LU/QR algorithms except the ones being present on the matrix multiplication algorithm of choice. The same

strategy could also be applied to tridiagonalization of symmetric matrices, but then we would just remade the algorithm in [1].

The speed of the algorithms depends on the capacity to perform rectangular matrix multiplications. Matrix multiplication algorithms are usually studied for square matrices and they have time complexity of $O(n^\omega)$, where n is the matrix side and ω depends on the algorithm: $\omega = 3$ for classical multiplication, $\omega = 2.8074$ for Strassen's algorithm and $\omega = 2.3728639$ for the asymptotically best known matrix multiplication algorithm [7]. For the present algorithms for Cholesky and LU decompositions, the interest lies on multiplying a matrix with dimensions $n \times s$ with a $s \times n$ matrix, where s is smaller than n. In [6] it is shown that such multiplication can be performed with $O(s^{\omega-2}n^2)$ arithmetic operations using square-based algorithms, but also that algorithms operating directly on rectangles may outperform the square based ones. In [3, 4] rectangular matrix multiplication is studied more thoroughly. Let $s = n^k$, the ω notation may be extended as $\omega(k)$ to indicate that the multiplication of a $n \times s$ by a $s \times n$ can be performed in $O(n^{\omega(k)})$. Furthermore, for small values of k, the optimal complexity of $O(n^2)$ is obtained, which is currently at $\alpha = 0.31389$, and thus $\omega(\alpha) = 2$. Nevertheless, algorithms with good $\omega(k)$ for every k are found in [4]. We highlight the value $\omega(0.5286) = 2.057085$, which we will use later.

2 The algorithms

The new Cholesky decomposition algorithm is shown as Algorithm 1, the new LU decomposition algorithm as Algorithm 2, and the new QR decompositon algorithm as Algorithm 3. All indices start at 1 and end at the size of the matrix, usually n, and all ranges are inclusive. The notation $A_{i,j}$ indicates access to the (i,j)-entry of the matrix A. The three algorithms are analogous, in the sense they have the same kind of variation respect to the classical algorithms, so let us comment from just the Cholesky viewpoint. It combines the ideas of the recursive Cholesky algorithm, where the lower right submatrix is updated by $-vv^T$ in each call; and of the Cholesky–Crout algorithm, which computes a column using the previous computed columns. In this new algorithm there is a parameter s indicating the maximum number of columns to process before updating the submatrix. Thus, when $s = 1$ it particularizes to the recursive algorithm and when $s = n$ it becomes Cholesky–Crout algorithm. The correctness of the algorithm is trivially reduced to the correctness of these two algorithms it generalizes.

The updates of the lower right submatrix are made by substractions of a RR^T block, where R has (or can be extended to) size $n \times s$. When $s \leq n^{0.313}$,

---

Algorithm 1: A fast Cholesky decomposition algorithm

Data: A square, symmetric, and positive-definite matrix A;
The size step s.
Result: The lower triangular matrix L such that $A = LL^T$.
n:= size of A ;
L:= zero matrix of size n (could be done inplace of A);
z:= 1 ;
for *c from 1 to n* **do**
    if $c = z + s$ **then**
        R:= submatrix of L by $[c$ to $n]$ and $[z$ to $c-1]$;
        S:= RR^T, using a fast rectangular matrix multiplication
        algorithm ;
        for *i from c to n* **do**
            for *j from c to n* **do**
                $A_{i,j}$:= $A_{i,j} - S_{i-c+1,j-c+1}$;
            end
        end
        z:= c ;
    end
    $L_{c,c}$:= $A_{c,c}$;
    for *k from z to c − 1* **do**
        $L_{c,c}$:= $L_{c,c} - L_{c,k}^2$;
    end
    $L_{c,c}$:= $\sqrt{L_{c,c}}$;
    for *i from c + 1 to n* **do**
        $L_{i,c}$:= $A_{i,c}$;
        for *k from z to c − 1* **do**
            $L_{i,c}$:= $L_{i,c} - L_{i,k}L_{c,k}$;
        end
        $L_{i,c}$:= $\frac{L_{i,c}}{L_{c,c}}$;
    end
end
return L

Algorithm 2: A fast LU decomposition algorithm

Data: A square matrix A with nonzero leading principal minors;
The size step s.

Result: The lower triangular matrix L and upper unitriangular
matrix U such that $A = LU$.

n:= size of A ;
L:= zero matrix of size n ;
U:= zero matrix of size n ;
z:= 1 ;
for c *from 1 to n* **do**

    if $c = z + s$ **then**

        R_L:= submatrix of L by $[c$ to $n]$ and $[z$ to $c-1]$;

        R_U:= submatrix of U by $[z$ to $c-1]$ and $[c$ to $n]$;

        S:= $R_L R_U$, using a fast rectangular matrix multiplication
        algorithm ;

        for i *from c to n* **do**

            for j *from c to n* **do**

                $A_{i,j}$:= $A_{i,j} - S_{i-c+1,j-c+1}$;

            end

        end

        z:= c ;

    end

    for i *from c to n* **do**

        $L_{i,c}$:= $A_{i,c}$;

        for k *from z to c − 1* **do**

            $L_{i,c}$:= $L_{i,c} - L_{i,k}U_{k,c}$;

        end

    end

    for i *from c to n* **do**

        $U_{c,i}$:= $A_{c,i}$;

        for k *from z to c − 1* **do**

            $U_{c,i}$:= $U_{c,i} - L_{c,k}U_{k,i}$;

        end

        $U_{c,i}$:= $\frac{U_{c,i}}{L_{c,c}}$;

    end

end
return L, U

Algorithm 3: A fast QR decomposition algorithm

Data: A matrix A;
The size step s.
Result: An orthogonal matrix Q and right upper triangular R such
         that $A = QR$.
n:= size of A ;
M:= a copy of A ;
Q:= zero matrix of size n ;
z:= 1 ;
for c *from 1 to n* **do**
    if $c = z + s$ **then**
        B_Q:= submatrix of Q by $[1$ to $n]$ and $[z$ to $c - 1]$;
        B_M:= submatrix of M by $[1$ to $n]$ and $[c$ to $n]$;
        C:= $B_Q^T B_M$, using a fast rectangular matrix multiplication
        algorithm ;
        S:= $B_Q C$, using a fast rectangular matrix multiplication
        algorithm ;
        for i *from* 1 *to* n **do**
            for j *from* c *to* n **do**
                $M_{i,j}$:= $M_{i,j} - S_{i,j-c+1}$;
            end
        end
        z:= c ;
    end
    v:= column c of M ;
    for j *from* z *to* $c - 1$ **do**
        u:= column j of Q ;
        v:= $v - u(u^T v)$;
    end
    v:=$\frac{v}{||v||}$;
    for i *from* 1 *to* n **do**
        $Q_{i,c}$:= v_i ;
    end
end
R := $Q^T A$, using a fast square matrix multiplication algorithm ;
return Q, R

this RR^T product can be computed in $O(n^2)$ [4] as said in the introduction, but other values of s also have interest. There is a total of n iterations. In each iteration there are $O(ns)$ arithmetic operations that always execute. Once every s iterations the first conditional executes, which contains the update of the submatrix in $O(n^2)$ arithmetic operations preceded by the rectangular matrix multiplication.

Therefore, if $s = n^{0.313}$ then the total volumen of computation is of $O(n^2 s + \frac{n}{s} n^2) = O(n^{2.687})$ arithmetic operations. Using the fastest matrix multiplication algorithm for square matrices, $\omega = 2.3728639$ [7], we get complexity $O(n^2 s + \frac{n}{s} s^{\omega-2} n^2)$. Its optimum is at $s = n^{0.61462815}$, where the algorithm becomes $O(n^{2.61462815})$. But it is better to use the value $\omega(0.5286) = 2.057085$, which leads to $s = n^{0.5285425}$ and time complexity of $O(n^2 s + \frac{n}{s} n^{\omega(k)}) = O(n^{2.5285425})$. Further improvements to rectangular matrix multiplication could in theory reduce the final complexity of these decomposition algorithms to $O(n^{2.5})$.

In practice those multiplication algorithms with small complexity incur in huge constants. A good alternative is Strassen algorithm, which can be used in rectangular matrices [6]. Taking $s = n^{0.8385}$, the whole algorithm becomes with Strassen multiplication $O(n^2 s + \frac{n}{s} s^{\omega-2} n^2) = O(n^{2.8385})$, where $\omega = \log_2(7) = 2.8074$ is the exponent in the cost of Strassen multiplication for square matrices.

An implementation of the algorithms using Strassen multiplication has proved to spend about 48% less time than GNU Scientific Library (GSL) [5] Cholesky implementation, about 57% less time than GSL LU implementation, and about 87% less time for the QR. For $n = 2000$, taking $s = 200$ and employing two Strassen recursions (and the remaining multiplications made with GSL) it took 1.14867 seconds, against the 2.28978 seconds of pure GSL. The test matrix was a symmetric one with random entries and large values in the diagonal. For $n = 4000$, we take $s = 400$ and employ three Strassen recursions. This consumes 8.99871 seconds, while GSL consumes 17.0262 seconds. The analogous for LU is, in $n = 2000$, 1.51041 seconds vs 3.53489 seconds; and in $n = 4000$, 12.203 seconds vs 28.7355 seconds; both with $s = 200$. Although one must note that GSL LU also computes a permutation matrix. The analogous for QR is, in $n = 2000$, 7.12522 seconds vs 51.8309 seconds; and in $n = 4000$, 64.3856 seconds vs 554.165 seconds; the first with $s = 100$ and the second with $s = 200$.

It is also remarkable that the s parameter could be varied during the execution. Modifications of this kind could lead to some improvements in practice.

3   Acknowledgments

I would like to give thanks to Luis M. Pardo for reading a draft of this note and giving some advice.

References

[1] Christian H. Bischof, Bruno Lang, and Xiaobai Sun. A framework for symmetric band reduction. *ACM Trans. Math. Softw.*, 26(4):581–601, December 2000.

[2] James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[3] François Le Gall. Faster algorithms for rectangular matrix multiplication. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 514–523, October 2012.

[4] François Le Gall and Florent Urrutia. *Improved Rectangular Matrix Multiplication using Powers of the Coppersmith-Winograd Tensor*, pages 1029–1046.

[5] Brian Gough. *GNU Scientific Library Reference Manual - Third Edition*. Network Theory Ltd., 3rd edition, 2009.

[6] Philip A. Knight. Fast rectangular matrix multiplication and QR decomposition. *Linear Algebra and its Applications*, 221:69 – 81, 1995.

[7] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM.