Applied Mathematics and Computation 328 (2018) 125–136

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Memory-usage advantageous block recursive matrix inverse

Iria C.S. Cosme[a,b,*], Isaac F. Fernandes[c], João L. de Carvalho[a], Samuel Xavier-de-Souza[a]

[a] Departamento de Engenharia de Computação e Automação, Universidade Federal do Rio Grande do Norte, Natal-RN 59078-970, Brazil
[b] Instituto Federal do Rio Grande do Norte, São Gonçalo do Amarante-RN 59261-727, Brazil
[c] Instituto Metrópole Digital, Universidade Federal do Rio Grande do Norte, Natal-RN, 59078-970 Brazil

ARTICLE INFO

Keywords:
Block matrices
Large matrices
Schur complement
Recursive matrix inversion
Low memory usage

ABSTRACT

The inversion of extremely high order matrices has been a challenging task because of the limited processing and memory capacity of conventional computers. In a scenario in which the data does not fit in memory, it is worth to consider exchanging less memory usage for more processing time in order to enable the computation of the inverse which otherwise would be prohibitive. We propose a new algorithm to compute the inverse of block partitioned matrices with a reduced memory footprint. The algorithm works recursively to invert one block of a $k \times k$ block matrix M, with $k \geq 2$, based on the successive splitting of M. It computes one block of the inverse at a time, in order to limit memory usage during the entire processing. Experimental results show that, despite increasing computational complexity, matrices that otherwise would exceed the memory-usage limit can be inverted using this technique.

1. Introduction

Matrix inversion is a computation task necessary in many scientific applications, such as signal processing, complex network analysis, statistics [1] and some eigenvalue-related problems [2], to name just a few. There are some commonly available matrix inversion algorithms for nonsingular matrices, like Gaussian elimination, Gauss–Jordan, LU Decomposition and Cholesky decomposition. The majority of these algorithms are computationally intensive in use of memory and processor. For example, computing the inverse of a $n \times n$ matrix with the Gauss–Jordan method has computational complexity of $O(n^3)$ and memory storage complexity of $O(n^2)$. This can forbid the applicability of such methods for large-scale matrices, mainly, because data may simply not fit in memory.

Working with big data is a situation becoming increasingly common in today's world due to advances in sensor and communication technologies and to the evolution of digital data. This has become a challenging task because the handled data exceed conventional sizes and are often collected at speed greater than the processing and memory capacity of conventional computers can handle. In classification (or regression) problems [3], for example, inverting matrices of extremely high order can lead to exceeding the computer memory capacity just to store its inverse. Many algorithms were developed with the objective to speed up the processing of large matrix computations. In the sequel, we list a few of those efforts regarding block and recursive algorithms.

* Corresponding author at: Instituto Federal do Rio Grande do Norte, São Gonçalo do Amarante-RN 59261-727, Brazil.
E-mail addresses: iria.cosme@ifrn.edu.br (I.C.S. Cosme), isaacfranco@imd.ufrn.br (I.F. Fernandes), samuel@dca.ufrn.br (S. Xavier-de-Souza).

https://doi.org/10.1016/j.amc.2018.01.051
0096-3003/

The use of block partitioned matrices is commonly used to cut down processing time of matrix computations. Block matrices may occur naturally due to the ordering of the equations and the variables in a wide variety of scientific and engineering applications, such as in the incompressible Navier-Stokes equations [4], mixed finite elements approximation of elliptic partial differential equations [5], optimal control [6], electrical networks [7] and the Strassen's Algorithm [8] for fast matrix multiplication. Algorithms that manipulate matrices at the block level are often more efficient because they are more abundant in level-3 operations [9], and thus, can be implemented recursively.

There are many related papers on the inverse of block matrices. In [10], the authors provide inverse formulae for 2×2 block matrices applied in block triangular matrices and various structured matrices such as Hamiltonian, per-Hermitian and centro-Hermitian matrices. The inversion of block circulant matrices has been extensively investigated in [11] and [12]. Also, in [13], the authors suggest the use block preconditioner for the block partitioned matrices.

Likewise, recursive algorithms have been applied in [11,14] and [15] for the inversion of particular cases of matrices, such as circulant-structured matrices. In [14] a recursive method is proposed for the LU decomposition of a real symmetric circulant matrix. In [11], a recursive algorithm is applied to calculate of the first block row of the inverse of a block circulant matrix with circulant blocks. And, finally, in [15], the authors propose a recursive algorithm based on the inversion of $k \times k$ block matrices for cases of matrices with circulant blocks based on the previous diagonalization of each circulant block.

In addition, in [16], the authors propose an efficient method for the inversion of matrices with U-diagonalizable blocks (being U a fixed unitary matrix) by utilizing the U-diagonalization of each block and subsequently a similarity transformation procedure. This approach allows getting the inverse of matrices with U-diagonalizable blocks without having to assume the invertibility of the blocks involved in the procedure, provided certain conditions met.

Despite these numerous efforts, the inversion of large dense matrices is still challenging for large datasets. In a scenario in which the data does not fit in the memory, exchanging less memory usage for more processing time could be an alternative to enable the computation of the inverse which otherwise would be prohibitive.

In this paper, motivated by the preceding considerations, we introduce a recursive method for the inversion of a $k \times k$ block matrix $M \in \mathbb{R}^{m \times m}$ with square blocks of order b. The basic idea of this algorithm, called Block Recursive Inversion (BRI), lies in the determination of one block of the inverse of the matrix at a time. The method is based on the successive splitting of the original matrix into 4 square matrices of an inferior order, called frames. For this, it is considered two stages, namely the forward recursive procedure and backward recursive procedure. The forward recursive procedure terminates after $k-2$ steps when the resulting frames have 2×2 blocks. Thereafter, in the backward recursive procedure, for each 4 frames generated, operations are carried out reducing them to a single block. Differently from those proposed in [11,14,15], our recursive algorithm may be applied for inverting any $M \in \mathbb{R}^{m \times m}$, provided that M is nonsingular and that all submatrices that need to be inverted in the recurrent procedure are also nonsingular.

The recursive algorithm proposed in [15] presents lower computational complexity than the BRI in inversion processes for the cases of matrices with circulant blocks. However, for the cases that the data do not fit in memory, using the BRI might be more adequate since it has lower memory storage complexity.

Besides requiring a much smaller memory footprint to work, having an algorithm that computes only parts of the inverse at a time may be useful for some applications. In [3], for example, only the diagonal blocks of the kernel matrix of the Least Squares Support Vector Machine (LS-SVM) are used to compute the predicted labels in the cross-validation algorithm. Thus, it is not strictly necessary to compute and, in fact, store in memory the entire inverse of the kernel matrix.

This paper is organized as follows. In Section 2, we introduce some notations and summarize the main definitions about the inversion of 2×2 block matrices using Schur Complement. In Section 3, we present the proposed recursive algorithm. In Section 4, we demonstrate a representative example of the recursive inversion of 4×4 block matrices. So, we introduce details of the implementation in Section 5. In Section 6, we present the complexity analysis of the proposed algorithm, including an investigation of the cost of memory storage computational. In Section 7, we describe the experiments as well as the results obtained. Finally, in Section 8, we present the conclusions and future works this research.

2. Preliminaries

In this section, we begin with some basic notations which are frequently used in the sequel.

Let $M \in \mathbb{R}^{m \times m}$ be a nonsingular matrix composed of square blocks $M_{\alpha\beta}$ of the same order

$$M = \begin{bmatrix} M_{11} & \cdots & M_{1k} \\ \vdots & \ddots & \vdots \\ M_{k1} & \cdots & M_{kk} \end{bmatrix}, \tag{1}$$

where $M_{\alpha\beta}$ designates the (α, β) block. With this notation, block $M_{\alpha\beta}$ has dimension $b \times b$, with $b = \frac{m}{k}$, and $M = (M_{\alpha\beta})$ is a $k \times k$ block matrix.