A particular block Vandermonde matrix

Malika Yaici and Kamel Hariche

Abstract—The Vandermonde matrix is ubiquitous in mathematics and engineering. Both the Vandermonde matrix and its inverse are often encountered in control theory, in the derivation of numerical formulas, and in systems theory. In some cases block vandermonde matrices are used. Block Vandermonde matrices, considered in this paper, are constructed from a full set of solvents of a corresponding matrix polynomial. These solvents represent block poles and block zeros of a linear multivariable dynamical time-invariant system described in matrix fractions. Control techniques of such systems deal with the inverse or determinant of block vandermonde matrices. Methods to compute the inverse of a block vandermonde matrix have not been studied but the inversion of block matrices (or partitioned matrices) is very well studied. In this paper, properties of these matrices and iterative algorithms to compute the determinant and the inverse of a block Vandermonde matrix are given. Instructions give you guidelines for preparing papers. A parallelization of these algorithms is also presented. The proposed algorithms are validated by a comparison based on algorithmic complexity.

Keywords—Block Vandermonde matrix, Matrix determinant, Matrix polynomials, Matrix inverse, Parallelization, Solvents.

I. INTRODUCTION

The Vandermonde matrix is very important and its uses include polynomial interpolation, coding theory, signal processing, etc. Literature on Vandermonde matrix goes back to 1965 and even before, where many papers deal with the study of its properties, its inverse and its determinant [1]-[5]. The importance of the Vandermonde matrix in control theory particularly has been emphasized by Tou [6], Brule [7], and Reis [8]. Vandermonde matrix may be also encountered in other domains, as in computer science, for example for the design of cross layer protocols with recovering from errors and packet loss impairments [9], and in [10] the quasi-cyclic (QC) protograph low-density parity-check (LDPC) codes are based on Vandermonde matrices.

In [11], the author proved that every generic nxn matrix is a product of a Vandermonde matrix and its transpose, and in [12] a Vandermonde matrix is decomposed to obtain variants of the Lagrange interpolation polynomial of degree≤n that passes through the n+1 points. Generally, the inverse of the usual Vandermonde matrix as well as the inverse of the generalized Vandermonde matrix is based on using interpolation polynomials.

The inversion of the Vandermonde matrix has received much attention for its role in the solution of some problems of numerical analysis and control theory. The work presented in [13] deals with the problem of getting an explicit formula for the generic element of the inverse to result in two algorithms in O(n2) and O(n3).

One of the first papers where the term block vandermonde matrix (BVM) is used, to my knowledge, is [14] but it is in [15] and [16] that the concept is fully studied; the block Vandermonde matrix is defined and its properties are explored. A method, based on the Gaussian elimination, to compute the determinant is also proposed.

In [17], the author gives a method to determine the biggest integer n=ν(q,t) for which there exist t×t matrices \{A_1,⋯,A_t\} with the highest power q such that the BVM is invertible.

In [18], linear diffusion layers achieving maximal branch numbers called MDS (maximal distance separable) matrices are constructed from block Vandermonde matrices and their transposes. Under some conditions these MDS matrices are involutory (its inverse is itself) which is of great value in cryptography.

Methods to compute the inverse of a block vandermonde matrix have not been studied but the inversion of block matrices (or partitioned matrices) is very well studied! The method to compute the inverse of a 2×2 block matrix is known, under the conditions that at least one of the two diagonal matrix entries must be non-singular. In [19], this condition is overcome by using three new types of symbolic block matrix inversion.

In [20], the properties of block matrices with block banded inverses are investigated to derive efficient matrix inversion algorithms for such matrices. In particular, a recursive algorithm to invert a full matrix whose inverse is structured as a block tridiagonal matrix and a recursive algorithm to compute the inverse of a structured block tridiagonal matrix are proposed.

Block Vandermonde matrices constructed using matrix polynomials solvents are very useful in control engineering, for example in control of multi-variable dynamic systems described in matrix fractions (see [21]). It is in these particular BVM that we are interested.

Parallelization may be a solution to problems where large size matrices, as BVM, are used. Large scale matrix inversion has been used in many domains and block-based Gauss-Jordan (G-J) algorithm as a classical method of large matrix inversion.

M. Yaici is with the Laboratoire LTII, University of Bejaia, Bejaia, 06000, Algeria (corresponding author : 213 664608272; fax: 213 34813721; e-mail: yaici.m@hotmai.com).

K. Hariche is with IGEE institute, University of Boumerdes, Boumerdes, 35000, Algeria. (e-mail: kharicher@yahoo.com).
has become the focus of many researchers. But the large parallel granularity in existing algorithms restricts the performance of parallel block-based G-J algorithm, especially in the cluster environment consisting of PCs or workstations. The author of [22] presents a fine-grained parallel G-J algorithm to settle the problem presented above.

In this paper a new algorithm to compute the inverse and determinant of a block Vandermonde matrix constructed from solvents are given. An implementation using Matlab has been undergone, in order to obtain the speed-up of the proposed algorithms compared to Matlab built-in functions. A parallelization of the proposed algorithms is given in section 4. A conclusion finishes the paper.

After this introduction, some needed mathematical preliminaries are presented in section 2, and then the main results come in section 3. A parallelization of the proposed algorithms is given in section 4. A conclusion finishes the paper.

II. MATHEMATICAL PRELIMINARIES

In linear algebra, a Vandermonde matrix, named after Alexandre-Théophile Vandermonde, is an m×m matrix with the terms of a geometric progression in each row:

\[ V = \begin{bmatrix}
1 & x_1 & \cdots & x_1^{m-1} \\
1 & x_2 & \cdots & x_2^{m-1} \\
\vdots & \vdots & \ddots & \vdots \\
1 & x_m & \cdots & x_m^{m-1}
\end{bmatrix} \tag{1} \]

And a column Vandermonde matrix of order n is the transpose of V.

For a set of n m×m matrices \( \{A_1, A_2, \ldots, A_n\} \), the corresponding block Vandermonde matrix (BVM) of order t is defined as follows:

\[ V = \begin{bmatrix}
I & I & \cdots & I \\
A_1 & A_2 & \cdots & A_n \\
\vdots & \vdots & \ddots & \vdots \\
A_1^{-1} & A_2^{-1} & \cdots & A_n^{-1}
\end{bmatrix} \tag{2} \]

The block Vandermonde matrices, we will be dealing with, are constructed from solvents of matrix polynomials. In this section, a recall on matrix polynomials, solvents, and their properties, will be given.

A. Matrix polynomials

**Definition 1**: A polynomial matrix (also called a λ-matrix), of order m, is an m x m matrix given as follows:

\[ A(t) = \begin{pmatrix}
a_{11}(t) & a_{12}(t) & \cdots & a_{1m}(t) \\
a_{21}(t) & a_{22}(t) & \cdots & a_{2m}(t) \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1}(t) & a_{m2}(t) & \cdots & a_{mm}(t)
\end{pmatrix} \tag{3} \]

where \( a_{ij}(t) \) are scalar polynomials (of degree r) over the field of complex numbers.

From a polynomial matrix we can construct a matrix polynomial and vice-versa.

**Definition 2**: An m’th order, r’th degree matrix polynomial is given by:

\[ A(t) = A_1t^r + A_{r-1}t^{r-1} + \cdots + A_0 \]

where \( A_i \) are m x m real matrices and t a complex number.

**Definition 3**: Let \( X \) be an m x m complex matrix. A right matrix polynomial is defined by:

\[ A_r(t) = A_0X^r + A_{r-1}X^{r-1} + \cdots + A_1X + A_0 \tag{5} \]

And a left matrix polynomial is defined by:

\[ A_L(t) = X^rA_r + X^{r-1}A_{r-1} + \cdots + XA_1 + A_0 \tag{6} \]

**Definition 4**: The complex number \( \lambda_i \) is called a latent value of \( A(t) \) if it is a solution of the scalar polynomial equation \( \det(A(t))=0 \). The non-trivial vector \( v_i \), solution of the equation \( A(\lambda_i)v_i=0 \), is called a primary right latent vector associated to the latent value \( \lambda_i \). Similarly, the non-trivial row vector \( w_i \), solution of the equation \( w_iA(\lambda_i)=0 \) is called a primary left latent vector associated with \( \lambda_i \) [21].

B. Solvents

**Definition 5**: A right solvent (or a block root) \( R \) of a polynomial matrix \( A(t) \) is defined by:

\[ A(R) = A_1R^r + A_{r-1}R^{r-1} + \cdots + A_1R + A_0 = 0_m \tag{7} \]

and the left solvent \( L \) of a polynomial matrix \( A(t) \) is defined by:

\[ A(L) = L^rA_r + L^{r-1}A_{r-1} + \cdots + LA_1 + A_0 = 0_m \tag{8} \]

A solvent is automatically non-singular. The determinant is non-null because its latent values (eigenvalues) are distinct; the latent vectors (eigenvectors) must be linearly independent.

C. Block Vandermonde matrix

As for an eigenvalue system, a block Vandermonde matrix can be defined for solvents with particular properties [21].

Let a set of r right solvents \( R_i \) (m×m matrices) of a corresponding matrix polynomial \( A(t) \). A row block Vandermonde matrix of order r is a \( r^m x r^m \) matrix defined as:
\[
V = \begin{pmatrix}
I_m & I_m & \cdots & I_m \\
R_1 & R_2 & \cdots & R_r \\
\vdots & \vdots & \ddots & \vdots \\
R_1^{r-1} & R_2^{r-1} & \cdots & R_r^{r-1}
\end{pmatrix}
\]  \quad (9)

And given a set of \( r \) left solvents \( L_i \) \((m \times m)\) matrices of a polynomial matrix \( A(t) \) a column block Vandermonde matrix of order \( r \) is defined as:

\[
V = \begin{pmatrix}
I_m & L_1 & \cdots & L_1^{r-1} \\
I_m & L_2 & \cdots & L_2^{r-1} \\
\vdots & \vdots & \ddots & \vdots \\
I_m & L_r & \cdots & L_r^{r-1}
\end{pmatrix}
\]  \quad (10)

**Remark 1:** In [23], the general right (left) block Vandermonde matrix constructed by solvents, where a right (left) solvent \( R_i \) \((L_i)\) with multiplicity \( m_i \) exists, is given.

**D. Non-singularity**

**Theorem 1:** If \( A(t) \) has distinct latent roots, then there exists a complete set of right solvents of \( A(X) \) \( \{R_i, \ldots, R_m\} \) and for any such set of solvents, The associated BVM \( V \) is nonsingular.

**Proof:** see [15]

**Theorem 2:** If we let \( \sigma[A(t)] \) denote the set of all latent roots of \( A(t) \) and \( \sigma[R_i] \) the set of eigenvalues of the right solvent \( R_i \), then a complete set of right solvents is obtained if we can find \( r \) right solvents such that:

\[
\bigcup_{i=1}^r \sigma[R_i] = \sigma[A(t)]
\]

\[
\sigma[R_i] \cap \sigma[R_j] = \emptyset
\]  \quad (11)

and the block Vandermonde matrix thus constructed is nonsingular.

**Proof:** see [16]

Just as for the right solvents, the existence of a left block root depends on the existence of a set of \( m \) linearly independent left latent vectors. A complete set of left block roots is obtained if we can find \( r \) left block roots where each block root involves a distinct set of \( m \) latent roots of \( A(t) \). This in turn requires that for each such a distinct set, we can find a corresponding set of linearly left latent vectors.

**Theorem 3:** A block Vandermonde matrix as defined in (9) is non-singular if and only if the set of \( k \) solvents \( \{R_1, \ldots, R_k\} \) with multiplicities \( \{m_1, \ldots, m_k\} \) is a complete set.

**Proof:** see [15, 23]

**Theorem 4:** The set of left solvents of \( A(t) \), which satisfies the following properties,
B. Inverse of a BVM

From [25], the inverse of a block partitioned matrix is given as follows:

If A is nonsingular:

\[
\begin{pmatrix}
A & B \\
C & D
\end{pmatrix}^{-1} = \begin{pmatrix}
A^{-1} + E*S_{i-1}^{-1}*F & -E*S_{i-1}^{-1} \\
-S_{i-1}^{-1}*F & S_{i-1}^{-1}
\end{pmatrix}
\]  \hspace{1cm} (17)

Where

\[ E = A^{-1} * B, F = C * A^{-1}, S_A = D - C * A^{-1} * B. \]

\[ S_i \] is the Shur complement of matrix A, is nonsingular.

And if D is non-singular:

\[
\begin{pmatrix}
A & B \\
C & D
\end{pmatrix}^{-1} = \begin{pmatrix}
S_D^{-1} & -S_D^{-1}*F \\
-E*S_D^{-1} & E*S_D^{-1}*F + D^{-1}
\end{pmatrix}
\]  \hspace{1cm} (18)

Where

\[ E = D^{-1} * C, F = B * D^{-1}, S_D = A - B * D^{-1} * C \]

\[ S_D \] is the Schur complement of matrix D and should be nonsingular.

Let us compute the inverse of a BVM of order r as given in (9) by using (17). In our case both diagonal entries are nonsingular.

\[ V_r^{-1} = \begin{pmatrix}
V_{r-1}^{-1} + E*S_{r-1}^{-1}*F & -E*S_{r-1}^{-1} \\
-S_{r-1}^{-1}*F & S_{r-1}^{-1}
\end{pmatrix} \]  \hspace{1cm} (19)

where \( S_{r-1} \) is the Shur complement of matrix \( V_{r-1} \) (computed at iteration \( r-1 \))

\[ S_{r-1} = D_{r-1}^{-1} - C_{r-1} * V_{r-1}^{-1} * B_{r-1} \]  \hspace{1cm} (20)

\[ E = V_{r-1}^{-1} * B_{r-1}, F = C_{r-1} * V_{r-1}^{-1} \]

and \( D_{r-1}, C_{r-1}, B_{r-1} \) are as given in (15).

The same procedure will be used to determine the inverse of the BVM \( V_{r-1} \). So the algorithm is an iterative procedure.

Algorithm: Let a complete set of solvents \( \{R_1, ..., R_r\} \) and the corresponding BVM \( V_r \) as given in (9). From the matrix \( V_r \), all sub-matrices \( (B_i, C_i, D_i) \) and \( S_i \) can be constructed and the determinant computed. The algorithm uses a function which computes the inverse of the Shur complement.

Step1: Let \( \text{INV} = \text{Im} \)

Step2: for i = 2*m to r*m with step=m

\[ B_{i-1} = V_r(1:i-2, 1:i-2); \]

\[ C_{i-1} = V_r(1:i-1:I, 1:i-2); \]

\[ D_{i-1} = V_r(1:i-1:I, 1:i-1); \]

\[ E_{i-1} = \text{INV} * B_{i-1}; \]

\[ F_{i-1} = C_{i-1} * \text{INV}; \]

\[ S_{i-1} = D_{i-1} - C_{i-1} * V_{i-1}^{-1} * B_{i-1}; \]

endfor

Algorithmic Complexity: The iterative algorithm requires r-2 iterations. The procedure consists of a set of affectations and the computation of the inverse of \( S_i \). The size of \( S_i \) is \( m \times m \) and Matlab uses Gaussian elimination method rather than an inverse algorithm. The Gaussian elimination has a complexity of \( O(m^3) \). The overall complexity of our algorithm is: \( O((r-2) * m^3) \).

C. Determinant of a BVM

From [25], the determinant of a block partitioned matrix is as follows:

If A is non-singular

\[ \det \begin{pmatrix}
A & B \\
C & D
\end{pmatrix} = \det A * \det (D - C * A^{-1} * B) \]  \hspace{1cm} (21)

And if D is non-singular:

\[ \det \begin{pmatrix}
A & B \\
C & D
\end{pmatrix} = \det D * \det (A - B * D^{-1} * C) \]  \hspace{1cm} (22)

Using this equation and the block decompositions given in the previous section we deduced the determinant of a BVM of order r as given in (9) by using (21) (both equations hold in our case).

\[ \det V_r = \det V_{r-1} * \det S_{r-1} \]  \hspace{1cm} (23)

where \( S_{r-1} \) is the Shur complement of matrix \( V_{r-1} \) defined in (20).

Remark 2: The determinant is, in general, needed with the inverse. So the inverse of \( V_{r-1} \) is computed using the previous algorithm.

Algorithm: Let a complete set of solvents \( \{R_1, ..., R_r\} \) and the corresponding BVM \( V_r \) as given in (9). From the matrix \( V_r \), all sub-matrices \( (B_i, C_i, D_i) \) and \( S_i \) can be constructed and the determinant computed. The algorithm uses a function which computes the determinant of the Shur complement.

Step1: Let \( \text{Det} = 1 \)

Step2: for i = 2*m to r*m with step=m

\[ V_{i-1} = V_r(1:i-2, 1:i-2); \]

\[ B_{i-1} = V_r(1:i-2, 1:i-1); \]

\[ C_{i-1} = V_r(i-1:I, 1:i-2); \]

\[ D_{i-1} = V_r(i-1:I, 1:i-1); \]

\[ E_{i-1} = \text{INV} * B_{i-1}; \]

\[ F_{i-1} = C_{i-1} * \text{INV}; \]

\[ S_{i-1} = D_{i-1} - C_{i-1} * V_{i-1}^{-1} * B_{i-1}; \]

\[ \text{Det} = \text{Det} * \text{determinant}(S_{i-1}); \]

endfor

Algorithmic Complexity: As for the inverse algorithm, the determinant iterative algorithm needs r-2 iterations. The procedure consists of a set of affectations, the computation of the inverse of a BVM and of the determinant of \( S \). The size of \( S \) is \( m \times m \) and Matlab uses the triangular factors of Gaussian
elimination method to compute the determinant and the inverse of a square matrix. The Gaussian elimination has a complexity of $O(m^3)$. So the complexity depends also on the size of $S$.

The overall complexity of our algorithm is: $O((r-2)*m^3)$.

D. Illustrative example

Let $P(\lambda)$ be a matrix polynomial of degree 4 and order 2:

$$P(\lambda) = I_2 x^4 + p_3 x^3 + p_2 x^2 + p_1 x + p_0$$

Where the matrix coefficients are:

$$p_3 = \begin{pmatrix} -1 & 2 \\ 3 & -5 \end{pmatrix}, \quad p_2 = \begin{pmatrix} 2 & 5 \\ 3 & -1 \end{pmatrix}, \quad p_1 = \begin{pmatrix} -1 & -1 \\ 3 & -2 \end{pmatrix}, \quad p_0 = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$$

This matrix polynomial presents a full set of 4 solvents:

$$R_1 = \begin{pmatrix} 87.40 & -181.70 \\ 42.21 & -87.70 \end{pmatrix}, \quad R_2 = \begin{pmatrix} 0.57 & -1.11 \\ 0.48 & -0.37 \end{pmatrix}, \quad R_3 = \begin{pmatrix} 0.96 & 4.42 \\ -0.60 & -1.86 \end{pmatrix}, \quad R_4 = \begin{pmatrix} -3.23 & -4.50 \\ 7.75 & 10.25 \end{pmatrix}$$

Then the associated BVM of order 4 is constructed:

$$V_4 = \begin{pmatrix} I_2 & I_2 & I_2 & I_2 \\ R_1 & R_2 & R_3 & R_4 \\ R_1^2 & R_2^2 & R_3^2 & R_4^2 \\ R_1^3 & R_2^3 & R_3^3 & R_4^3 \end{pmatrix}$$

The previous algorithms have been implemented on Matlab and the numerical results are as follows:

$$V_4^* = \begin{pmatrix} -0.05 & -0.10 & -0.02 & 0.12 & -0.14 & -0.22 & 0.02 & 0.03 \\ -0.02 & -0.05 & -0.01 & 0.06 & -0.07 & -0.11 & 0.01 & 0.01 \\ 0.55 & 0.72 & 0.44 & 0.68 & 0.34 & -0.15 & 0.12 & 0.09 \\ 0.39 & 0.64 & -0.14 & -0.57 & 0.23 & -0.21 & -0.11 & 0.01 \\ 0.03 & -0.30 & -0.59 & 0.10 & -0.59 & 1.10 & -0.02 & -0.21 \\ -0.01 & 0.17 & 0.28 & -0.19 & 0.15 & -0.24 & 0.01 & 0.06 \\ 0.47 & -0.32 & 0.16 & -0.90 & 0.40 & -0.73 & -0.12 & 0.10 \\ -0.37 & 0.24 & -0.13 & 0.70 & -0.31 & 0.57 & 0.09 & -0.08 \end{pmatrix}$$

$$\text{Det} = -1.0037e+007$$

We used the tic/toc functions of Matlab to determine the execution time $T_1$ of our procedure of the BVM inverse and determinant to be compared to the time $T_2$ of the “inv” and “det” functions of Matlab.

The results are for the inverse are:

$T_1 = 2.2283e-005$

$T_2 = 2.4673e-004$

Speedup = $T_2/T_1 = 11.073$

The results are for the determinant are:

$T_1 = 7.2925e-006$

$T_2 = 1.0615e-004$

Speedup = $T_2/T_1 = 14.556$

IV. PARALLELIZATION

For both Determinant and Inverse, a parallelization is possible because of the decomposition step in the proposed algorithm. Even though the iterative approach is difficult to optimally parallelize! The above decomposition is useful only if a parallel execution is possible, otherwise the benefits are negligible.

There exist two kinds of parallelization of matrix calculus: data or tasks (calculation) decomposition. Because data decomposition is already performed, so task decomposition is proposed.

A. Parallel inverse of BVM

From the data decomposition, master-slave task decomposition was performed on the sequential algorithm. The master task will execute the data scattering and gathering, and sequential instructions and at least three (3) slave tasks will execute the parallel blocks.

Algorithm:

Step1: Let $\text{INV} = I_n$

Step2: for $i = 2*m$ to $r*m$ with step=m

Parallel block

$$B_{i-1} = V_r(1:i-2; i-1:i);$$

$$C_{i-1} = V_r(i-1:i; 1:i-2);$$

$$D_{i-1} = V_r(i-1:i; i-1:i);$$

End

Parallel block

$$E_{i-1} = \text{INV} * B_{i-1};$$

$$F_{i-1} = C_{i-1} * \text{INV};$$

$$S_{i-1} = D_{i-1} - C_{i-1} * \text{INV} * B_{i-1};$$

end

$$iS = S_{i-1}^{-1};$$

Parallel block

$$\text{INV} = \begin{pmatrix} \text{INV} + E^*iS^*F & -E^*iS \\ -iS^*F & iS \end{pmatrix}$$

end

endfor

B. Parallel determinant of BVM

As for the precedent algorithm, master-slave task decomposition has been performed, and the master task will execute the data scattering and gathering and sequential parts, and at least four (4) slave tasks will execute the parallel blocks.

Algorithm:

Step1: Let $\text{Det}=1$

Step2: for $i = 2*m$ to $r*m$ with step=m
Parallel block

\[ V_{l-1} = V_l(1:1-l; 1:1-l); \]
\[ B_{l-1} = V_l(1:1-l; 1:1-l); \]
\[ C_{l-1} = V_l(1:1-l; 1:1-l); \]
\[ D_{l-1} = V_l(1:1-l; 1:1-l); \]
end

\[ S_{l-1} = D_{l-1} - C_{l-1} * V_{l-1}^{-1} * B_{l-1}; \]
\[ \text{Det} = \text{Det} * \text{determinant}(S_{l-1}); \]
endfor

C. Algorithmic complexity

The overall time complexity of the two algorithms is the same as before: \( O((r-2) * m^2) \). But the detailed complexity is slightly better.

An implementation using Matlab has been done. Matlab (classical) offers parallel execution of a set of instructions (parallel block) using \texttt{parfor}. Matlab uses the number of cores available on the used computer using \texttt{matlabpool}.

V. CONCLUSION

In this paper new results on the computation of the inverse and the determinant of a particular block Vandermonde matrix are given. Efficient algorithms are proposed with their algorithmic complexities.

We used the \texttt{tic/toc} functions of Matlab to determine the execution time of our algorithms to be compared to the execution time of Matlab functions, and the proposed inverse algorithm is found 11 times quicker, and the determinant algorithm is 14 times quicker. The parallel execution time was greater than the sequential execution time, because of the large amount of data flowing between the cores at each iteration.

These new computation techniques are very useful in control theory, where systems are described in matrix fractions description and their properties are deduced from solvents. In this case block Vandermonde matrices constructed from solvents, their inverse and determinants are needed. The future work is within this axis and it consists in proposing a parallel algorithm to solve the Compensator (Diophantine) equation where block Vandermonde matrices constructed using matrix polynomial solvents are involved.