Another Dubious Way to Compute the Exponential of a Matrix

Jerzy Respondek(✉)

Faculty of Automatic Control, Electronics and Computer Science, Department of Applied Informatics, Silesian University of Technology, ul Akademicka 16, 44-100 Gliwice, Poland
jerzy.respondek@polsl.pl

Abstract. In this paper we analyzed recent works on inverting Vandermonde matrix, both classical and generalized, which were unknown during the publication of Moler's and Van Loan's paper 'Nineteen Dubious Ways to Compute the Exponential of a Matrix'. Upon that analysis we proposed the Vandermonde method as the fourth candidate for calculating exponent of generic matrices. On this basis we also proposed the Vandermonde based method to compute the exponential of certain class of special matrices, i.e. the companion matrices.

Keywords: Matrix exponential · Confluent Vandermonde matrix · Vandermonde matrix · Structured matrix · Special matrix · Companion matrix

1 Introduction

In the paper [1] Moler and Van Loan have concluded that the exponential of a matrix could be computed in many ways, but none are completely satisfactory; though they noticed that some of the methods are preferable to others. In this comments in general we agree with this opinion, but give arguments that the set of the preferable methods should be extended.

Particularly, we proposed the algorithms for inverting the generalized Vandermonde matrix as a main tool to find the exponent for the companion matrices, a kind of special (also called structured) form matrices.

The paper is organized as follows: in Sect. 2 we presented the range of applications of the Vandermonde matrices, in Sect. 3 we gave its definition followed by the efficient algorithm to find the inversion, in Sect. 4 we proposed a method for computing the exponential of a companion matrix, in Sect. 5 we illustrated the proposed method by the example of a decent, 10×10 dimensionality, in Sect. 6 we discussed the known Vandermonde method for generic matrices, with special focus to a degree, in which the new algorithms for the Vandermonde matrices directly improved the quality of that

This work was supported by Statutory Research funds of Department of Applied Informatics, Silesian University of Technology, Gliwice, Poland (02/100/BK_21/0008).

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12949, pp. 465–478, 2021.
https://doi.org/10.1007/978-3-030-86653-2_34

exponent algorithm. In Sect. 7 we discussed some other methods for generic matrices. In Sect. 8 by the example we showed how to apply the algorithms for inverting the Vandermonde matrices to obtain speed up the matrix multiplications, and in Sect. 9 we gave some final conclusions and proposed a potential ways of further progress in the area.

2 Importance of the Vandermonde Matrices

The confluent Vandermonde matrix is a generalization of classical Vandermonde matrix, allowing multiple nodes, thus the formulas and algorithms designed for them also works in the classical case. We formally defined it in Sect. 3. They arise in a broad range of both theoretical and practical problems. Below we surveyed the problems which make it necessary to inverse the confluent Vandermonde matrices.

- Control problems: investigating of the so-called controllability [2] of the higher order systems leads to the problem of inverting the classic Vandermonde matrix [3] (in case of distinct zeros of the system characteristic polynomial) and the confluent Vandermonde matrix [4] (for systems with multiply characteristic polynomial zeros). As the examples of the higher order models of the physical objects may be mentioned the Timoshenko's elastic beam equation [5] (4th order) and the Korteweg-de Vries equation of waves on shallow water surfaces [6] (3rd, 5th and 7th order).
- Interpolation: apart from the ordinary polynomial interpolation with single nodes (which leads to classical, well-known Newton interpolation) we consider the Hermite interpolation, allowing the multiple interpolation nodes. That problem leads to the system of linear equations, with the confluent Vandermonde matrix ([7] pp. 363–373, [8]).
- Information coding: the confluent Vandermonde matrix is used in coding and decoding the information in the Hermitian's code [9].
- Cryptography: decoding the Reed-Solomon codes [10].
- Optimization of the non-homogeneous differential equations [11].
- So-called fast matrix multiplication: transformation of so-called arbitrary precision approximating-algorithms to exactly-computing algorithms is presented in [12]. An example of approximating algorithms is given in [13] and [14]. More on this we gave in Sect. 8.
- The classic Vandermonde matrix still remains a subject of research e.g. [15, 16].
- Other branches of Vandermonde generalization also are investigated e.g. [17].

3 Theoretical Background

First let us explain what the confluent Vandermonde matrix V is. Let $\lambda_1, \lambda_2, \ldots, \lambda_r$ be given pair wise distinct zeros of the characteristic polynomial $p(s) = (s-\lambda_1)^{n_1} \ldots (s-\lambda_r)^{n_r}$ with $n_1 + \ldots + n_r = n$. The confluent Vandermonde matrix V

related to the zeros of $p(s)$ is defined to be the $n \times n$ matrix $V = [V_1\ V_2\ \ldots\ V_r\ ]$, where the block matrix $V_k = V(\lambda_k, n_k)$ is of order $n \times n_k$ having elements [18]:

$$[V(\lambda_k, n_k)]_{ij} = \begin{cases} \binom{i-1}{j-1}\lambda_k^{i-j}, & \text{for } i \geq j \\ 0, & \text{otherwise} \end{cases}$$

for $k = 1, 2, \ldots, r$; $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n_k$. For $n_1 = \ldots = n_r = 1$ we get the classical Vandermonde matrix.

The articles [4, 18] present the following theorem for inverting the confluent Vandermonde matrices:

3.1 Theorem – Inverting Confluent Vandermonde Matrices

The inverse of the confluent Vandermonde matrix V has the form $V^{-1} = \left[W_1^T\ W_2^T \cdots W_r^T \right]^T$. The column vectors h_{ki} of the block matrix $W_k = [h_{kn}, \ldots, h_{k1}]$ in the inverse confluent Vandermonde matrix V^{-1} may be recursively computed by the following scheme:

$$\begin{cases} h_{k1} = \left[K_{k,1} \cdots K_{k,n_k}\right]^T \\ h_{k2} = J_k(\lambda_k, n_k)h_{k1} + a_1 h_{k1} \\ \qquad\qquad \vdots \\ h_{kn} = J_k(\lambda_k, n_k)h_{k(n-1)} + a_{n-1}h_{k1} \end{cases}, \quad k = 1, 2, ..., r \tag{1}$$

where λ_k is the eigenvalue, a_k are the coefficients of the characteristic polynomial $p(s)$, $J_k(\lambda_k, n_k)$ is the elementary Jordan block (2):

$$J_k(\lambda_k, n_k) = \begin{bmatrix} \lambda_k & 1 & & \cdots & 0 \\ & \lambda_k & 1 & & \vdots \\ & & \ddots & \ddots & \\ & & & \lambda_k & 1 \\ 0 & & & & \lambda_k \end{bmatrix}_{n_k \times n_k}, \quad k = 1, 2, ..., r \tag{2}$$

and $K_{k,j}$ are the auxiliary coefficients that can be calculated recursively in a numerical way by the following algorithm [19]:

3.2 Algorithm – Fast Inversion of Confluent Vandermonde Matrices

The $K_{k,j}$, $j = n_k - 1, \ldots, 1$ (4) coefficients may be recursively computed by the following recursive scheme (3):

$$\begin{cases} L_{ki}^{(q+1)}(\lambda_k) = q!(\lambda_k - \lambda_i)^q \cdot K_{k,n_k-q} - q \cdot L_{ki}^{(q)}(\lambda_k), \ i = 1, .., k-1, k+1, .., r \\ K_{k,n_k-q-1} = -\frac{1}{(q+1)!} \sum\limits_{i=1,\ i\neq k}^{r} n_i \frac{L_{ki}^{(q+1)}(\lambda_k)}{(\lambda_k - \lambda_i)^{q+1}} \end{cases} \tag{3}$$

for $q = 0, 1, .., n_k - 2$ *and* $k = 1, 2, \ldots, r$. *The* K_{k,n_k} *coefficients may be computed directly from the formula (4)*:

$$K_{k,n_k}(s)\big|_{s=\lambda_k} = \left[\frac{1}{p(s)}\big(s - \lambda_k\big)^{n_k}\right]_{s=\lambda_k}, \quad k = 1, 2, \ldots, r \tag{4}$$

4 Computing the Exponential of a Companion Matrix

The proposed method bases on the rule, that the Jordan canonical form similarity matrix of the companion[1] matrix is a confluent Vandermonde matrix ([20] pp. 86–95). Let the companion matrix is defined by the characteristic polynomial $p(s) = (s - \lambda_1)^{n_1} \ldots (s - \lambda_r)^{n_r}$. Then its matrix form is as follows (5):

$$C = \begin{bmatrix} 1 & & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 \\ -a_0 & \cdots & -a_{n-3} & -a_{n-2} & -a_{n-1} \end{bmatrix} \tag{5}$$

To calculate the coefficients $a_0, \ldots, a_{n-1}$ at first we need to rewrite the characteristic polynomial $p(s)$ in a "flattened" form. That step can be formalized in the way proposed in [21] p. 278. Namely, we define the series of n numbers $s_1, .., s_n$ as follows:

$$\begin{cases} s_1 = \ldots = s_{n_1} & := \lambda_1 \\ s_{n_1+1} = \ldots = s_{n_1+n_2} & := \lambda_2 \\ \vdots & \\ s_{n_1+\ldots+n_{r-1}+1} = \ldots = s_n & := \lambda_r \end{cases} \tag{6}$$

Thus for the characteristic polynomial the following equality (7) holds the true:

$$p(s) = (s - \lambda_1)^{n_1}(s - \lambda_2)^{n_2} \ldots (s - \lambda_r)^{n_r} = (s - s_1)(s - s_2) \cdot \ldots \cdot (s - s_n), \ n_1 + \ldots + n_r = n \tag{7}$$

[1] In mathematical literature in some languages instead is used the term 'Frobenius'. But in the English Literature the class of Frobenius matrices is more general, encompassing the companion matrices as its special case.

After the expansion of the polynomial $p(s)$ and sorting it with respect to decreasing powers of the s variable, we can receive (8):

$$p(s) = 1 \cdot s^n - w_1^{(n)}(s_1, \ldots, s_n)s^{n-1} + w_2^{(n)}(s_1, \ldots, s_n)s^{n-2} + \ldots + (-1)^n w_n^{(n)}(s_1, \ldots, s_n) \tag{8}$$

where $w_j^{(n)}(s_1, \ldots, s_n)$ are the so-called elementary symmetric[2] functions, defined by the formula (9) ([22] pp. 25):

$$\begin{cases} w_1^{(n)}(s_1, \ldots, s_n) = s_1 + s_2 + \ldots + s_n \\ w_2^{(n)}(s_1, \ldots, s_n) = s_1s_2 + s_1s_3 + \ldots + s_1s_n + \ldots + s_{n-1}s_n \\ w_3^{(n)}(s_1, \ldots, s_n) = s_1s_2s_3 + s_1s_2s_4 + \ldots + s_{n-2}s_{n-1}s_n \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ w_n^{(n)}(s_1, \ldots, s_n) = s_1s_2 \cdot \ldots \cdot s_n \end{cases} \tag{9}$$

And the searched coefficients $a_0, \ldots, a_{n-1}$ in the companion matrix (5) we can express by choosing the sign of the proper elementary symmetric function. Observe (8) and (9):

$$a_i = (-1)^{n-i} w_{n-i}^{(n)}(s_1, ..., s_n), \qquad i = 0, 1, ..., n-1 \tag{10}$$

Algorithm for calculating the elementary symmetric functions (9) can be found e.g. in [17]. Now the companion matrix can be rewritten in the – convenient - Jordan canonical form (11):

$$C = T \cdot J \cdot T^{-1} \tag{11}$$

The core of the proposed method is fact, that the similarity matrix T after the Jordan decomposition of the companion matrix is just the Vandermonde matrix ($T = V$, classical Vandermonde for single nodes and confluent, for multiple nodes, respectively), combined with the latest algorithms for their inverting. We perform the inversion of the Vandermonde matrix V by the algorithm 3.2.

Exponent of a matrix in a Jordan canonical form is easy to compute. A general function f of the matrix in a Jordan canonical form is expressed by the Eq. (12):

$$f(C) = V \cdot f(J) \cdot V^{-1} \tag{12}$$

Thus the searched exponent expresses the formula (13):

$$e^C = V \cdot e^J \cdot V^{-1} \tag{13}$$

Considering that the differential of the exponential function is equal to itself, matrix equality for the function $f(\psi) = e^{\psi}$ (here ψ stands for both a scalar or a matrix) is:

$$f(J_k(\lambda_k, n_k)) = \begin{bmatrix} \frac{f(\lambda_k)}{0!} & \frac{f'(\lambda_k)}{1!} & \ldots & \frac{f^{(n_k)}(\lambda_k)}{n_k!} \\ 0 & \frac{f(\lambda_k)}{0!} & \ldots & \frac{f^{(n_k-1)}(\lambda_k)}{(n_k-1)!} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \frac{f(\lambda_k)}{0!} \end{bmatrix} = \begin{bmatrix} e^{\lambda_k} & e^{\lambda_k} & \ldots & \frac{e^{\lambda_k}}{n_k!} \\ 0 & e^{\lambda_k} & \ldots & \frac{e^{\lambda_k}}{(n_k-1)!} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & e^{\lambda_k} \end{bmatrix} \tag{14}$$

[2] Any function we call symmetric, if and only if after any arbitrary permutation of its independent variables we receive the same polynomial ([23] pp. 77–84).

where $J_k(\lambda_k, n_k)$ is the elementary Jordan block of the form (2).

To sum up, finding the exponent of a companion matrix by the proposed in this article method requires the following steps, with their corresponding time complexities:

- Building the confluent Vandermonde matrix V and the Jordan block matrix J - time complexity of those both operations are $O(n^2)$.
- Inverting the confluent Vandermonde matrix V - algorithm 3.2 assures, that this step is of an $O(n^2)$ time class, regardless of the matrix parameters and particularly their multiplicities.
- Calculating the exponent of the Jordan block – also quadratic time (formula (14)).
- Applying the formula $e^C = V \cdot e^J \cdot V^{-1}$ (13) requires two matrix multiplications, which by standard algorithm can be performed in $O(n^3)$ time. We also have faster algorithms, and in Sect. 8 we discussed in details that one of them, which just involves Vandermonde matrices.
- Additionally, if we want to have the companion matrix in the explicit form, we need to calculate the elementary symmetric functions (9). The time complexity of this step is – at worse - $O(n^2)$ (17).

Thus the overall time complexity of the proposed method to calculate the exponent of a companion matrix is determined by the applied matrix multiplication algorithm, being - at worst – of the cubic time.

5 Example – Calculating the Exponent of a Companion Matrix

Let us consider companion matrix defined by the following characteristic polynomial:

$$p(s) = (s + 0.5)^1(s + 3.0)^2(s + 2.0)^3(s + 1.0)^4 \tag{15}$$

Its eigenvalues are summarized in the Table 1:

Table 1. Eigenvalues of the companion matrix with their respective multiplicity.

i	1	2	3	4
λ_i	-0.5	-3.0	-2.0	-1.0
n_i	1	2	3	4

To obtain the explicit form of the companion matrix, defined by the characteristic polynomial (15), we have to obtain the $s_1, .., s_n$ roots, in compliance with the scheme (6). Observe:

$$\begin{cases} s_1 := & \lambda_1 = -0.5 \\ s_2 = s_3 := & \lambda_2 = -3.0 \\ s_4 = s_4 = s_6 := & \lambda_3 = -2.0 \\ s_7 = s_8 = s_9 = s_{10} := & \lambda_4 = -1.0 \end{cases} \tag{16}$$

Execution of the algorithm for calculating the elementary symmetric functions (17) for the s_i nodes with values (16), together with formula (10), leads to the following form of the companion matrix, the exponent to be calculated of (17):

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 36.0 & 294.0 & 1039.0 & 2098.5 & 26807.0 & 22801.5 & 13002.0 & 493.5 & 119.0 & 16.5 \end{bmatrix} \tag{17}$$

The generalized, confluent Vandermonde matrix, corresponding to the eigenvalues given in Table 1, have the form (18):

$$V = \left[\begin{array}{ccc:ccc:ccc:c} 1.00000 & 1.00 & 0.00 & 1.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 \\ -0.5000 & -3.00 & 1.00 & -2.00 & 1.00 & 0.00 & -1.00 & 1.00 & 0.00 & 0.00 \\ 0.25000 & 9.00 & -6.00 & 4.00 & -4.00 & 1.00 & 1.00 & -2.00 & 1.00 & 0.00 \\ -0.1250 & -27.00 & 27.00 & -8.00 & 12.00 & -6.00 & -1.00 & 3.00 & -3.000 & 1.00 \\ 0.06250 & 81.00 & -108.00 & 16.00 & -32.00 & 24.00 & 1.00 & -4.00 & 6.00 & -4.00 \\ -0.0312 & -243.00 & 405.00 & -32.000 & 80.00 & -80.00 & -1.00 & 5.00 & -10.00 & 10.00 \\ 0.01562 & 729.00 & -1458.00 & 64.00 & -192.00 & 240.00 & 1.00 & -6.00 & 15.00 & -20.00 \\ -0.0078 & -2187.00 & 5103.00 & -128.00 & 448.00 & -672.00 & -100 & 7.00 & -21.00 & 35.00 \\ 0.00390 & 6561.00 & -17496.0 & 256.00 & -1024.0 & 1792.00 & 1.00 & -8.00 & 28.00 & -56.00 \\ -0.0019 & -19683.00 & 59049.0 & -512.00 & 2304.00 & -4608.00 & -100 & 9.00 & -36.00 & 84.00 \end{array}\right] \tag{18}$$

and the Jordan matrix built by the eigenvalues given in Table 1, have the form of a block diagonal matrix (19):

$$J = diag[J_1(\lambda_1, n_1),\ J_2(\lambda_2, n_2),\ J_3(\lambda_3, n_3),\ J_4(\lambda_4, n_4)] \tag{19}$$

To invert the Vandermonde matrix (18) we apply the algorithm 3.2, and directly we obtain (20):

$$V^{-1} = \begin{bmatrix} 54.61 & 336.78 & 902.63 & 1378.23 & 1319.82 & 821.47 & 332.23 & 84.195 & 12.136 & 0.758 \\ 1.72 & 13.440 & 44.90 & 84.42 & 98.58 & 74.20 & 36.005 & 10.860 & 1.8475 & 0.135 \\ 0.300 & 2.3500 & 7.87 & 14.86 & 17.43 & 13.20 & 6.4500 & 1.9625 & 0.3375 & 0.025 \\ -100.3 & -758.22 & -2433.04 & -4362.15 & -4820.22 & -3405.55 & -1539.11 & -429.55 & -67.29 & -4.518 \\ -38.00 & -288.33 & -929.55 & -1675.22 & -1861.33 & -1322.33 & -600.66 & -168.33 & -26.44 & -1.77 \\ -12.00 & -92.00 & -300.33 & -549.33 & -621.00 & -450.0 & -209.00 & -60.00 & -9.66 & -0.66 \\ 45.00 & 408.00 & 1485.50 & 2899.50 & 3401.81 & 2509.87 & 1170.86 & 334.50 & 53.312 & 3.625 \\ -85.50 & -612.75 & -1872.88 & -3222.06 & -3439.06 & -2358.75 & -1039.00 & -283.56 & -43.56 & -2.875 \\ 18.00 & 147.00 & 501.5 & 938.25 & 1064.00 & 761.50 & 345.50 & 96.25 & 15.00 & 1.00 \\ -18.00 & -129.00 & -390.5 & -658.75 & -684.75 & -456.00 & -195.00 & -51.75 & -7.75 & -0.50 \end{bmatrix} \tag{20}$$

Finally, the searched exponent for the companion matrix (17) have the following form, by the matrix formulas (13) and (14):

$$e^C = \begin{bmatrix} 116.33 & 712.2 & 1906.64 & 2910.64 & 2787.18 & 1734.76 & 701.59 & 177.8 & 25.62 & 1.60 \\ -57.66 & -354.6 & -952.07 & -1454.74 & -1393.4 & -867.33 & -350.78 & -88.89 & -12.8143 & -0.80 \\ 28.83 & 177.79 & 477.53 & 728.61 & 697.27 & 433.84 & 175.43 & 44.45 & 6.40 & 0.40 \\ -14.41 & -88.90 & -238.29 & -362.86 & -347.46 & -216.41 & -87.57 & -22.20 & -3.20 & -0.20 \\ 7.20 & 44.41 & 118.99 & 181.61 & 174.80 & 109.06 & 44.11 & 11.17 & 1.60 & 0.10 \\ -3.61 & -22.35 & -60.05 & -91.99 & -88.54 & -54.58 & -21.84 & -5.4996 & -0.79 & -0.04 \\ 1.77 & 10.88 & 28.88 & 43.44 & 40.53 & 23.98 & 9.63 & 2.49 & 0.36 & 0.02 \\ -0.85 & -5.17 & -13.68 & -20.72 & -20.07 & -13.40 & -6.79 & -2.035 & -0.31 & -0.02 \\ 0.73 & 5.17 & 16.10 & 29.30 & 34.31 & 26.65 & 13.26 & 3.30 & 0.40 & 0.02 \\ -0.69 & -4.96 & -14.99 & -24.61 & -22.83 & -9.95 & 1.39 & 3.68 & 1.00 & 0.08 \end{bmatrix}$$

6 Discussion of the Vandermonde Method for Generic Matrices

As the main drawback of the Vandermonde method 11 Moler and Van Loan [1] points out weak time complexity, claimed to be $O(n^4)$, resulting from the necessity of calculating the matrix series A^{k-1}.

With available now algorithms enabling fast matrix multiplication that statement is not precise enough to compare the matrix exponential calculation ways. Nowadays the Strassen's algorithm is well known since decades with $n \cdot O(n^{2.807}) = O(n^{3.807})$. Thus we can decrease the time complexity to $O(n^{3.807})$ effectiveness. In general, with the help of $O(n^k)$ matrix multiplication we can improve the method 11 from [1] to the $O(n^{k+1})$ time efficiency algorithm. Since 1978 that exponential has been decreased a number of times, from 2.78041 in 1978 (Pan [24]), 2.7799 in 1979 (Bini et al. [13]), 2.376 in 1990 (Coppersmith, Winograd [25]) to 2.373 in 2014 (Vassilevska [26]).

Thus, for this time, involving the fastest matrix multiplication algorithm to the method 11 gives the $O(n^{3.373})$ algorithm, being much closer to $O(n^3)$ than to – the considered in [1] - $O(n^4)$.

Here one can complain that those algorithms are useful only in case of large matrices, but that is just the case indicated by 1, i.e. the efficiency gains importance for large n.

Another issue related to the time complexity of the method 11 which is analyzed by the Moler's and Van Loan's work [1], is the necessity to invert the Vandermonde matrix. The efficiency of the used algorithm to invert the Vandermonde matrices is no less important than the efficiency of involved matrix multiplication algorithms, because in case of applying the generic matrix inversion algorithms we have, besides calculating the A^{k-1} series, the $O(n^3)$ operation to be performed by the method 11, which is a not negligible time.

Moler and Van Loan as the example of implementing method 11 proposes expressions developed by Vidyasagar [27]. The selected work of a matrix exponential copes with distinct eigenvalues. The topic in the article [28] allowed multiple eigenvalues. At the time the Moler and Van Loan work [1] appeared, there were available, now a bit obsolete, references [29] and [30] for inverting the Vandermonde matrix. They contain a generic symbolic methodology, which cannot be assessed in the view of numerical computing.

Another fast but dubious way to invert a Vandermonde matrix is to apply the theorem given by Cormen et al. [31], showing how to invert a matrix in a time no worse than the applied matrix multiplication algorithm. Thus applying e.g. the matrix multiplication way proposed in [26] we directly gain the method to obtain the v_{jk} entries in $O(n^{2.373})$ time.

In practice, such a way of proceeding is worthless, because Vandermonde matrices are known for their ill-conditioning, caused by their very structure. That is one of the reasons why there is a necessity to develop specialized algorithms for the structured matrices, besides usually also with better efficiency[3].

In the opposite to - though indirectly - referenced works [29] and [30], nowadays we dispose by fast and reliable algorithms designed specially to invert Vandermonde matrices, both in the single eigenvalues case as well as the multiple ones. In the next paragraphs we selected, to our best knowledge, the fastest together with a decent stability - importantly - applicable to a general form of Vandermonde matrix, with allowed single or multiple each of the eigenvalues, with independent, different multiplicity for each of the eigenvalues[4].

For the single eigenvalues case the $O(n^2)$ algorithms can be found in [15] and [16]. Like one could expect, a more delicate is the problem for the multiple eigenvalues case. Articles [4, 18, 19] together give the universal $O(n^2)$ algorithm, covering every input parameters allowed by the mathematical definition of the confluent Vandermonde matrix.

Another approach is used in the work Zhong, Zhaoyong [32] which achieves $O(n^2)$ time only in a special case of small eigenvalues multiplicities, but in return copes with more general form of the type of structured Vandermonde matrix in question.

When it goes to the space requirements, storage of the method 11 is indeed n^3 even if the spanning matrices $A_0, \ldots, A_{n-1}$ are not saved, because to avoid it still it's necessary to save the consecutive A^{k-1} powers. But a short lookout on operating memory market shows that n^3 space complexity is not an issue now.

7 Discussion Other Methods for Generic Matrices

7.1 The Langrange Interpolation by the Eigenvectors Method

Moler and Van Loan [1] in section 14 propose another approach to utilize the Langrange interpolation by the eigenvectors. They conclude that the $O(n^4)$ work involved in the computations of the consecutive A^{k-1} powers is unnecessary.

[3] The problem is neatly defined in the very title of the classical in the structured matrices field monograph [35].

[4] The trait of generality is - surprisingly - not a standard for the algorithms available in the literature. For example the classical in the associated problem of solving the Vandermonde linear systems article [33] only sketches algorithm for a very peculiar version of the confluence, i.e. with allowed multiplicity equal to only of the first eigenvalue, with all the rest single. The same work [33] suggests that the general case cannot be easily treated, stating in the second paragraph of the page 900: "(can be treated easily)… *with only the two endpoints of confluency greater than one, or that with all points of the same order of confluency.*". Significantly, all of the four Pascal-like codes in the appendix of [33] copes only with a classical Vandermonde linear systems, with single eigenvalues (pp. 901–902).Obviously algorithms with such an artificial restrictions are worthless in the view of computing the exponential of a matrix.

In general that observation is proper, but it says nothing about the numerical price of that gain. The numerical stability of inverting a matrix consisting of eigenvectors, which has no special structure, in fact is a lottery, since we cannot apply any specialized algorithm. Also due to lack of structure of the matrix to the inverted, its $O(n^3)$ cost is inevitable, unless we use even more dubious theorem on inverting the matrix by multiplication from Cormen et al. [31].

Thus the eigenvectors method in our opinion is more dubious than the Vandermonde approach, also in the view of the new works on inverting that kind of matrix, that appeared after first publishing of [1].

7.2 The Schur Decomposition Method

Beside scaling and squaring and ODE-based methods, Moler and Van Loan [1] as the one of the best method sees the Schur decomposition with eigenvalue clustering.

Unfortunately also the Schur decomposition suffers from numerical stability. It can be only partially mitigated by the block clustering, merging into a separate blocks nearly confluent eigenvalues. Higham in [34] pp. 228–229 proposes to set the blocking parameter $\delta = 0.1$ and notices, that the optimal choice of δ is problem-dependent. What's worse instability can be present for all choices of δ and all orders of blocking.

Moreover, the blocking operation itself is costly. When computing the separation on $m \times m$ blocks by exact methods it costs $O(m^4)$ (!), whereas by approximate methods it is still $O(m^3)$ [34] p. 226.

In the Vandermonde method the numerical stability lets know only during inverting this kind of matrix. But this matrix is known for its very ill-conditioning and thanks to it each algorithm must be followed by the numerical stability analysis, or at least by thorough experimental tests in this matter. That is necessary since for any choice of points its grow rate is at least exponential, and for the harmonic points it grows faster than factorial. That is worse than Pascal and Hilbert matrices, also known for their ill-conditioning ([35] p. 428).

Moreover, for the Vandermonde matrix with single nodes the well-known algorithm by Bjorck And Pereyra [33] achieves (very) high accuracy, despite 'embedded' into this matrix ill-conditioning ([35] pp. 434–436, [36]). The fair price we pay for it is to calculate the A^{k-1} series.

In the view of the above analysis in our opinion the Vandermonde method is worthy to be consider method to calculate the exponent of a generic matrix.

8 Vandermonde Matrices in Fast Matrix Multiplication

Both the groups of methods to compute the exponential of a matrix, i.e. for the generic and structured matrices, requires to multiply matrices. An the effectiveness of that operation decides on the overall efficiency of finding the exponential. The topic of fast matrix multiplication have already a broad literature; survey of most known algorithms we gave in the beginning of Sect. 6.

In certain class of the fast matrix multiplication (FMM) algorithms a Vandermonde matrix plays important role, namely to transform so-called arbitrary precision FMM

algorithm (abbreviated as APA) to its exact-precision equivalent (abbreviated as EC) we need to invert the Vandermonde matrix. The first APA algorithm in the literature was given in [13], for the base size matrices 12×12. In this section as the example of APA algorithm we present an algorithm for 3×3 base dimension, discovered in work [14] p. 438.

To calculate the matrix product $D = AB$ of two matrices, at first we calculate the following 21 auxiliary – so-called – aggregated products, for $i, j \in \{1, 2, 3\}$:

$$\begin{cases} u_{i,i} = (a_{i,1} + \varepsilon^2 a_{i,2})(\varepsilon^2 b_{1,i} + b_{2,i}) \\ v_{i,i} = (a_{i,1} + \varepsilon^2 a_{i,3}) b_{3,i} \\ w_i = a_{i,1}(b_{2,i} + b_{3,i}) \\ u_{i,j} = (a_{i,1} + \varepsilon^2 a_{j,2})(b_{2,i} - \varepsilon b_{1,j}), \quad (i \neq j) \\ v_{i,j} = (a_{i,1} + \varepsilon^2 a_{j,3})(b_{3,i} + \varepsilon b_{1,j}), \quad (i \neq j) \end{cases} \tag{21}$$

The product is called aggregated, because in a single multiplications the APA algorithm groups a series of the A, B matrix factors entries. The crucial issue in the APA algorithm is to select small ε, but simultaneously the ε must not be equal to zero.

We the use of the products (21), and only with that products, we can express the desired product $D = AB$ as follows:

$$\begin{cases} d'_{i,i} = \frac{1}{\varepsilon^2}(u_{i,i} + v_{i,i} - w_i) \\ d'_{j,i} = \frac{1}{\varepsilon^2}(u_{i,j} + v_{i,j} - w_i) + \frac{1}{\varepsilon}(v_{j,i} - v_{j,j}), \quad (i \neq j) \end{cases} \tag{22}$$

Poking (21) into (22), in matrix notation, gives:

$$\begin{aligned} D = AB + & \begin{bmatrix} 0 & a_{12}b_{11} - a_{13}b_{11} + a_{13}b_{31} - a_{23}b_{31} & a_{12}b_{11} - a_{13}b_{11} + a_{13}b_{31} - a_{33}b_{31} \\ a_{22}b_{12} - a_{23}b_{12} - a_{13}b_{32} + a_{23}b_{32} & 0 & a_{22}b_{12} - a_{23}b_{12} + a_{23}b_{32} - a_{33}b_{32} \\ a_{32}b_{13} - a_{33}b_{13} - a_{13}b_{33} + a_{33}b_{33} & a_{32}b_{13} - a_{33}b_{13} - a_{23}b_{33} + a_{33}b_{33} & 0 \end{bmatrix} \varepsilon \\ & + \begin{bmatrix} -a_{12}b_{11} & -a_{23}b_{12} & -a_{33}b_{13} \\ -a_{13}b_{11} & -a_{22}b_{12} & -a_{33}b_{13} \\ -a_{13}b_{11} & -a_{23}b_{12} & -a_{32}b_{13} \end{bmatrix} \varepsilon^2 = AB + R_1\varepsilon + R_2\varepsilon^2 \end{aligned} \tag{23}$$

Thus if $\varepsilon \to 0$ and $\varepsilon \neq 0$, then D converges to exact value AB but never is exactly equal. There is a question whether it is possible to obtain the exact value. The general, method how to transform any APA matrix multiplication algorithm to its exact version is presented in [12] p. 92.

The highest degree of the ε parameter in (23) is equal to $d = 2$. By the method presented in [12] to construct an exact algorithm we select $d + 1 = 3$ pairwise distinct parameters ε, named ε_1, ε_2, ε_3, and make a linear combination of (23) with coefficients α_1, α_2, α_3:

$$\begin{aligned} D &= \alpha_1(AB + R_1\varepsilon_1 + R_2\varepsilon_1^2) + \alpha_2(AB + R_1\varepsilon_2 + R_2\varepsilon_2^2) + \alpha_3(AB + R_1\varepsilon_3 + R_2\varepsilon_3^2) \\ &= (\alpha_1 + \alpha_2 + \alpha_3)AB + (\alpha_1\varepsilon_1 + \alpha_2\varepsilon_2 + \alpha_3\varepsilon_3)R_1 + (\alpha_1\varepsilon_1^2 + \alpha_2\varepsilon_2^2 + \alpha_3\varepsilon_3^2)R_2 \end{aligned} \tag{24}$$

We can now observe, that if $\alpha_1 + \alpha_2 + \alpha_3 = 1$ and $\alpha_1\varepsilon_1^k + \alpha_2\varepsilon_2^k + \alpha_3\varepsilon_3^k = 0$ for $1 \leq k \leq d+1$, we obtain an exact result for the desired product.

That condition can be rewritten in the following matrix form:

$$\begin{bmatrix} 1 & 1 & 1 \\ \varepsilon_1 & \varepsilon_2 & \varepsilon_3 \\ \varepsilon_1^2 & \varepsilon_2^2 & \varepsilon_3^2 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{25}$$

We can notice, that the system matrix in (25) is just the classical Vandermonde matrix. Article [37] proved, that if an APA algorithm is efficient on the level $O(n^k)$, the corresponding EC algorithm, constructed by the method proposed by [12], have efficiency $O(n^k \log n)$. That means that the cost of that is an additional logarithmic factor, what is not a big price.

Particularly, for the proposed in current section APA algorithm, requiring 21 multiplications to find the product of 3×3 dimension[5] matrices, we have the efficiency:

$$O(n^{\log_3 21} \cdot \log n) \approx O(n^{2.77} \cdot \log n) \tag{26}$$

instead of "standard" $O(n^3)$. And in the same level the algorithms to compute the exponential of a matrix are accelerated, both for the generic as well as structured matrices.

9 Summary

In this article we showed how to apply the Vandermonde matrices to the aim of finding an exponent of a matrix, both generic as well as certain class of special matrices, i.e. the companion matrices. We use the algorithms for inverting two classes of Vandermonde matrices:

- The classical Vandermonde, in the case that all of the eigenvalues are distinct
- The so-called confluent Vandermonde, in the case of multiple eigenvalues.

The new algorithms to invert both types of matrices assures to return the result after quadratic time in every case. Thus the meaning of the methods to find the matrix exponents, which bases on Vandermonde special matrices, raised. To this fact is devoted this article.

Interestingly, we can use the Vandermonde matrices in two different stages of the algorithm:

- To calculate the sole exponent.
- To speed up matrix multiplication, necessary in the main exponent algorithm.

As the further directions of research we can see:

- Involving the vector hardware units, which are embedded into contemporary CPUs.
- Designing algorithms for the GPU units to find the exponent.

[5] Higher dimensions 'can be multiplied by the 3×3 algorithm by recursion.

References

1. Moler, C., Van Loan, C.: Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. SIAM Rev. **45**(1), 3–49 (2003)
2. Klamka, J.: Controllability of Dynamical Systems. Kluwer Academic Publishers, Dordrecht (1991)
3. Respondek, J.: Approximate controllability of infinite dimensional systems of the n-th order. Int. J. Appl. Math. Comput. Sci. **18**(2), 199–212 (2008)
4. Respondek, J.: On the confluent Vandermonde matrix calculation algorithm. Appl. Math. Lett. **24**, 103–106 (2011)
5. Timoshenko, S.: Vibration Problems in Engineering, 3rd edn. D. Van Nostrand Company, London (1955)
6. El-Sayed, S.M., Kaya, D.: An application of the ADM to seven order Sawada-Kotara equations. Appl. Math. Comput. **157**, 93–101 (2004)
7. Kincaid, D.R., Cheney, E.W.: Numerical Analysis: Mathematics of Scientific Computing, 3rd edn. Brooks Cole, California (2001)
8. Spitzbart, A.: A generalization of Hermite's interpolation formula. Am. Math. Mon. **67**(1), 42–46 (1960)
9. Lee, K., O'Sullivan, M.E.: Algebraic soft-decision decoding of Hermitian codes. IEEE Trans. Inf. Theory **56**(6), 2587–2600 (2010)
10. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. SIAM J. Soc. Ind. Appl. Math. **8**(2), 300–304 (1960)
11. Gorecki, H: On switching instants in minimum time control problem. One dimensional case n-tuple eigenvalue. Bull. de L'Acad. Pol. Des. Sci. **16**, 23–30 (1968)
12. Bini, D.: Relations between exact and approximate bilinear algorithms. Applications. Calcolo **17**, 87–97 (1980)
13. Bini, D., Capovani, M., Lotti, G., Romani, F.: $O(n^2.7799)$ complexity for approximate matrix multiplication. Inform. Process. Lett. **8**(5), 234–235 (1979)
14. Schönhage, A.: Partial and total matrix multiplication. SIAM J. Comput. **10**(3), 434–455 (1981)
15. Eisinberg, A., Fedele, G.: On the inversion of the Vandermonde matrix. Appl. Math. Comput. **174**, 1384–1397 (2006)
16. Yan, S., Yang, A.: Explicit algorithm to the inverse of Vandermonde matrix. In: International Conference on Test and Measurements, pp. 176–179 (2009)
17. El-Mikkawy, M.E.A.: Inversion of a generalized Vandermonde matrix. Int. J. Comput. Math. **80**, 759–765 (2003)
18. Hou, S., Hou, E.: Recursive computation of inverses of confluent Vandermonde matrices. Electron. J. Math. Technol. **1**, 12–26 (2007)
19. Respondek, J.: Numerical recipes for the high efficient inverse of the confluent Vandermonde matrices. Appl. Math. Comput. **218**, 2044–2054 (2011)
20. Gorecki, H.: Optimization of the Dynamical Systems. PWN, Warsaw (1993)
21. Hou, S., Hou, E.S.: A recursive algorithm for triangular factorization of inverse of confluent Vandermonde matrices. In: AIP Conference Proceedings, vol. 1089, no. 277, pp. 277–288 (2009)
22. Barbeau, E.J.: Polynomials, Problem Books in Mathematics. Springer, Berlin (1989). https://www.springer.com/gp/book/9780387406275. ISBN 978-0-387-40627-5. Series ISSN 0941-3502
23. Prasolov, V.: Polynomials, Algorithms and Computation in Mathematics. Springer, Berlin (2004). https://doi.org/10.1007/978-3-642-03980-5

24. Pan, V.Ya: Strassen's algorithm is not optimal - trilinear technique of aggregating uniting and cancelling for constructing fast algorithms for matrix operations. In: Proceedings of 19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Mich (1978)
25. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. **9**(3), 251–280 (1990)
26. Williams, V.V.: Multiplying matrices in $O(n^{2.373})$ time. Tech. rep. Stanford University (2014)
27. Vidyasagar, M.: A novel method of evaluating e^{At} in closed form. IEEE Trans. Automatic Contr. AC **15**, 600–601 (1970)
28. Luther, U., Rost, K.: Matrix exponentials and inversion of confluent Vandermonde matrices. Electron. Trans. Numer. Anal. **18**, 91–100 (2004)
29. Tou, J.T.: Determination of the inverse Vandermonde matrix. IEEE Trans. Autom. Contr. AC **9**, 314 (1964)
30. Wu, S.H.: On the inverse of Vandermonde matrix. IEEE Trans. Autom. Contr. AC **11**, 769 (1966)
31. Cormen, T.H., Leiserson, Ch.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)
32. Zhong, X., Zhaoyong, Y.: A fast algorithm for inversion of confluent Vandermonde-like matrices involving polynomials that satisfy a three-term recurrence relation. SIAM J. Matrix Anal. Appl. **19**(3), 797–806 (1998)
33. Bjorck, A., Pereyra, V.: Solution of Vandermonde systems of equations. Math. Comp. **24**(112), 893–903 (1970)
34. Higham, N.J.: Functions of Matrices. Theory and Computation, SIAM, Philadelphia (2008)
35. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. SIAM, Philadelphia (1996)
36. Higham, N.J.: Error analysis of the Bjorck-Pereyra algorithms for solving Vandermonde systems. Numer. Math. **50**(5), 613–632 (1987)
37. Romani, F.: Complexity measures for matrix multiplication algorithms. Calcolo **17**, 77–86 (1980)