Computing matrix functions arising in engineering models with orthogonal matrix polynomials[☆]

Emilio Defez[a,d], Jorge Sastre[b,d], Javier Ibáñez[c,d], Pedro Ruiz[c,d]

[a]*Instituto de Matemática Multidisciplinar*
[b]*Instituto de Telecomunicaciones y Aplicaciones Multimedia*
[c]*Instituto de Instrumentación para Imagen Molecular*
[d]*Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain*

Abstract

Trigonometric matrix functions play a fundamental role in the solution of second order differential equations. Hermite series truncation together with Paterson-Stockmeyer method and the double angle formula technique allow efficient computation of the matrix cosine. A careful error bound analysis of the Hermite approximation is given and a theoretical estimate for the optimal value of its parameters is obtained. Based on the ideas above, an efficient and highly-accurate Hermite algorithm is presented. A MATLAB implementation of this algorithm has also been developed and made available online. This implementation has been compared to other efficient state-of-the-art implementations on a large class of matrices for different dimensions, obtaining higher accuracy and lower computational costs in the majority of cases.

Keywords: Hermite matrix approximation, matrix cosine, MATLAB, error bound.

1. Introduction

Matrix functions play a relevant role in different areas of science and technology. They arise most frequently in connection with the solution of differential systems and control theory. For example, it is well known that

[☆]This work has been supported by the Spanish *Ministerio de Educación* grant MTM2009-08587.

the wave equation

$$v^2 \frac{\partial^2 \psi}{\partial x^2} = \frac{\partial^2 \psi}{\partial t^2} \ , \tag{1}$$

plays an important role in many areas of engineering and applied sciences. When we use the spatially semi-discretization method of the wave equation (1), we obtain the matrix differential problem

$$Y''(t) + AY(t) = 0 \ , \ Y(0) = Y_0 \ , \ Y'(0) = Y_1 \ , \tag{2}$$

where A is a square matrix and Y_0 and Y_1 are vectors, see [1] for details. Matrix problem (2) has the solution

$$Y(t) = \cos\left(\sqrt{A}t\right)Y_0 + \left(\sqrt{A}\right)^{-1} \sin\left(\sqrt{A}t\right)Y_1, \tag{3}$$

where $\sqrt{A}$ denotes any square root of a non-singular matrix A (see *e.g.* expression (1.2) of [2]). More general problems of type (2), with a forcing term $F(t)$ on the right-hand side arise from mechanical systems without damping, and their solutions can be expressed in terms of integrals involving the matrix sine and cosine [3]. Thus, trigonometric matrix functions play an important role in second order differential systems, similar to matrix exponential e^{At} in first order differential systems [4].

Moreover, the matrix cosine is used in the method of Yau-Lu for reducing the symmetric eigenvalue problem (finding all the eigenvalues and eigenvectors of a dense symmetric matrix) to a number of matrix multiplications [5].

Computing the matrix sine reduces to computing the matrix cosine through $\sin(A) = \cos(A - \frac{\pi}{2}I)$. Thus we concentrate on the matrix cosine. Serbin and Blalock proposed a general algorithm for computing the matrix cosine in [1], which uses rational approximations and the double angle formula

$$\cos(2A) = 2\cos^2(A) - I. \tag{4}$$

In [6] new methods for computing matrix exponential, sine and cosine based on Hermite matrix polynomial series were presented. A new bound for Hermite matrix polynomials was provided and it was used to give expressions for obtaining the number of Hermite series terms depending on the desired approximation error in exact arithmetic. Later, Higham, Smith and Hargreaves developed two algorithms based on (4) and Padé approximation in [7, 2], including truncation and rounding error analysis. Recently, in [8] the authors introduced a new parameter in the matrix cosine and sine Hermite series from [6] and new error bounds, improving both accuracy and efficiency.

In this paper we use the new matrix cosine Hermite series from [8], providing sharper bounds for Hermite matrix polynomials and the approximation error, and computing the optimal values of the series parameter to develop a competitive Hermite algorithm for computing the matrix cosine in IEEE double precision arithmetic that uses: matrix scaling based on (4), Paterson-Stockmeyer's method for the evaluation of Hermite series [9, 10], and accuracy bound tests similar to those proposed in [10, p. 6456-6457]. A MATLAB implementation of this algorithm is made available online and it is compared with the MATLAB function `funm` [11] and a MATLAB implementation based on the Padé algorithm given in [2], i.e. function `cosm`, providing higher accuracy and efficiency than both methods in the majority of test matrices.

This paper is organized as follows. Section 2 summarizes previous results of Hermite matrix polynomial series expansion of $\cos(A)$ and the development of a new error bound. In Section 3, an algorithm based on that error bound is described. Numerical experiments are presented in Section 4. Finally, conclusions are given in Section 5.

Throughout this paper, $[x]$ denotes the integer part of x. To obtain the above mentioned error bound, we will use any subordinate matrix norm $\| A \|$, $A \in \mathbb{C}^{r\times r}$, and in the subsequent error analysis, we will use the 1-norm $\| A \|_1$. If $\mathcal{A}(k,n)$ is a matrix in $\mathbb{C}^{r\times r}$ for $n \geq 0, k \geq 0$, the following identity holds [6]:

$$\sum_{n\geq 0}\sum_{k\geq 0}\mathcal{A}\left(k,n\right) = \sum_{n\geq 0}\sum_{k=0}^{n}\mathcal{A}\left(k,n-k\right). \tag{5}$$

2. Hermite matrix polynomial series expansions of matrix cosine. Error bound

For the sake of clarity in the presentation of the following results we recall some properties of Hermite matrix polynomials which have been established in [12, 6, 8]. From (3.4) of [12, p. 25] the nth Hermite matrix polynomial satisfies

$$H_n\left(x,\frac{1}{2}A^2\right) = n!\sum_{k=0}^{[\frac{n}{2}]}\frac{(-1)^k\,(xA)^{n-2k}}{k!(n-2k)!}\ , \tag{6}$$

for an arbitrary matrix A in $\mathbb{C}^{r\times r}$. From [8], we have the following Hermite matrix polynomial series expansion of the matrix cosine $\cos(Ay)$:

$$\cos(Ay) \; = e^{-\frac{1}{\lambda^2}} \sum_{n\geq 0} \frac{(-1)^n}{\lambda^{2n}(2n)!} H_{2n}\left(y\lambda, \frac{1}{2}A^2\right). \tag{7}$$

Denoting by $C_N(\lambda, A^2)$ the Nth partial sum of series (7) for $y=1$, one gets the approximation

$$C_N(\lambda, A^2) = e^{-\frac{1}{\lambda^2}} \sum_{n=0}^{N} \frac{(-1)^n}{\lambda^{2n}(2n)!} H_{2n}\left(\lambda, \frac{1}{2}A^2\right) \approx \cos(A), \;\; \lambda \in \mathbb{C}. \tag{8}$$

Working similarly as in [13, pp. 1913], we can obtain a bound for Hermite matrix polynomials $\left\|H_{2n}\left(x, \frac{1}{2}A^2\right)\right\|$ based on $||A^2||$, see [2], using the Taylor series for the hyperbolic cosine $\cosh(y) = \sum_{n\geq 0} y^{2n}/(2n)!$. Taking norms in (6), one gets

$$\left\|H_{2n}\left(x, \frac{1}{2}A^2\right)\right\| \leq (2n)! \sum_{k=0}^{n} \frac{\left(\|A^2\|^{\frac{1}{2}}\right)^{2(n-k)}}{k!(2(n-k))!} |x|^{2(n-k)}. \tag{9}$$

On the other hand, using (5), it follows that

$$\begin{aligned} e\cosh\left(|x|\, \|A^2\|^{\frac{1}{2}}\right) &= \sum_{n\geq 0} \frac{\left(|x|\, \|A^2\|^{\frac{1}{2}}\right)^{2n}}{(2n)!} \sum_{k\geq 0} \frac{1}{k!} = \sum_{n\geq 0}\sum_{k\geq 0} \frac{\left(|x|\, \|A^2\|^{\frac{1}{2}}\right)^{2n}}{(2n)!k!} \\ &= \sum_{n\geq 0}\sum_{k=0}^{n} \frac{\left(\|A^2\|^{\frac{1}{2}}\right)^{2(n-k)}}{k!(2(n-k))!} |x|^{2(n-k)}, \end{aligned} \tag{10}$$

and as a consequence

$$\sum_{k=0}^{n} \frac{\left(\|A^2\|^{\frac{1}{2}}\right)^{2(n-k)}}{k!(2(n-k))!} |x|^{2(n-k)} \leq e\cosh\left(|x|\, \|A^2\|^{\frac{1}{2}}\right). \tag{11}$$

Multiplying by $(2n)!$ in (11) and using (9), we have the result:

$$\left\|H_{2n}\left(x, \frac{1}{2}A^2\right)\right\| \leq (2n)!\, e\cosh\left(x\, \|A^2\|^{\frac{1}{2}}\right), \; \forall x \in \mathbb{R}, \; n\geq 0, \; \forall A \in \mathbb{C}^{r\times r}. \tag{12}$$

Taking into account approximation (8) and bound (12), it follows that

$$\begin{aligned}
\left\|\cos(A) - C_N(\lambda, A^2)\right\| &\leq e^{-\frac{1}{\lambda^2}} \sum_{k\geq N+1} \frac{1}{\lambda^{2k}(2k)!} \left\| H_{2k}\left(\lambda, \frac{1}{2}A^2\right)\right\| \\
&\leq e^{1-\frac{1}{\lambda^2}} \cosh\left(\lambda \left\|A^2\right\|^{\frac{1}{2}}\right) \sum_{k\geq N+1} \frac{1}{\lambda^{2k}} \\
&= e^{1-\frac{1}{\lambda^2}} \cosh\left(\lambda \left\|A^2\right\|^{\frac{1}{2}}\right) \left[\sum_{k\geq 0} \frac{1}{\lambda^{2k}} - \sum_{k=0}^{N} \frac{1}{\lambda^{2k}}\right]. \qquad (13)
\end{aligned}$$

Simplifying the geometric series in (13), we have finally the bound:

$$\left\|\cos(A) - C_N(\lambda, A^2)\right\| \leq \frac{e^{1-\frac{1}{\lambda^2}} \cosh\left(\lambda \left\|A^2\right\|^{\frac{1}{2}}\right)}{(\lambda^2-1)\lambda^{2N}}. \qquad (14)$$

3. Algorithm

In this section we describe Algorithm 1 (`cosher`), based on Hermite series, which uses bound (14) for choosing the scaling factor for computing the matrix cosine.

Algorithm 1 (`cosher`) Given a matrix $A \in \mathbb{C}^{r\times r}$ and the order N of Hermite matrix approximation of the cosine function, this algorithm computes $B \cong \cos(A)$.

```
1: Preprocessing of matrix $A$: $\check{A}$.
2: Compute optimal values of $\lambda$ and $s$.
3: SCALING PHASE: $\tilde{A} = \check{A}/2^s$
4: Compute $\tilde{B} = C_N(\lambda, \tilde{A}^2)$, where $C_N$ is the Hermite approximation of the
   cosine function.
5: for $i = 1 : s$ do
6:     $\tilde{B} = 2\tilde{B}^2 - I$
7: end for
8: Postprocessing of matrix $\tilde{B}$: $B$.
```

The preprocessing and postprocessing are based on applying transformations to reduce the norm of matrix A and recover the matrix $B \cong \cos(A)$ from the matrix $\tilde{B}$ obtained in the Loop 5-7. The available techniques to reduce the norm of a matrix are argument translation and balancing [14, p. 299]. The argument translation is based on the formula

$$\cos(A - \pi jI) = (-1)^j \cos(A),\ k \in \mathbb{Z},$$

and on finding the integer q such that the norm of matrix $A - \pi qI$ is minimum. This value can be calculated by using Theorem 4.18 from [14]. Balancing is a heuristic that attempts to equalize the norms of the kth row and kth column, for each k, by a diagonal similarity transformation defined by a non singular matrix D. Balancing tends to reduce the norm, though this is not guaranteed, so we will use it only for matrices where the norm is really reduced. For those matrices, if $\breve{A} = D^{-1}(A - \pi qI)D$ is the obtained matrix in the preprocessing, the postprocessing consists of computing $B = (-1)^q D \cos(\tilde{B}) D^{-1}$.

For the evaluation of $C_N(\lambda, \tilde{A}^2)$ the Horner and Paterson-Stockmeyer's method can be applied [9, 10], obtaining previously with MATLAB the symbolic expression of $C_N(\lambda, \tilde{A}^2)$ as a polynomial of matrix $\tilde{A}^2$ with degree N and coefficients depending on λ, for each considered value of N. The double angle formula (4) is used to recover $\cos(\breve{A})$ from the matrix $\tilde{B}$ obtained in Step 4, see [1] for details.

The optimal values of N with respect to cost of the evaluation of matrix polynomial $C_N(\lambda, \tilde{A}^2)$ with Paterson-Stockmeyer method are included in the set $S_N = \{1, 2, 4, 6, 9, 12, 16, 20, \cdots\}$, see [10, p. 6454]. The scaling factor s and the parameter λ is chosen as follows. If N_k, where k is the position of N in S_N, is the chosen order of Hermite approximation, the number of matrix product evaluations in Algorithm 1 is equal to $k + s$. Hence, the number of matrix products depends on the scaling factor s. For the evaluation of $\cos(A)$ with IEEE double precision arithmetic, analogously to [2], we consider an absolute error-based algorithm. If we take the error in (14) to be lower than or equal to the unit roundoff in double precision floating-point $u = 2^{-53}$, i.e,

$$\frac{e^{1-\frac{1}{\lambda^2}} \cosh\left(\lambda \sqrt{\left\| \left(\breve{A}/2^s\right)^2 \right\|_1}\right)}{(\lambda^2 - 1)\lambda^{2N}} \le u, \tag{15}$$

then

$$s \ge \frac{1}{2} \log_2 \left(\left\| \breve{A}^2 \right\|_1 \right) + \log_2 \left(\frac{\lambda}{\operatorname{arc\,cosh}\left(\frac{u(\lambda^2-1)\lambda^{2N}}{e^{1-1/\lambda^2}} \right)} \right). \tag{16}$$

We choose the parameter $\lambda = \lambda_{\min}$, where $\lambda_{\min}$ is the value of λ such that the right-hand side of (16) reaches the minimum value. Hence, if we define

$$\Theta_N = \frac{1}{\lambda_{\min}} \operatorname{arc\,cosh}\left(\frac{u(\lambda_{\min}^2 - 1)\lambda_{\min}^{2N}}{e^{1-1/\lambda_{\min}^2}}\right) \tag{17}$$

then, using (16), it follows that

$$s = \left\lceil \log_2\left(\frac{\sqrt{\|\check{A}^2\|_1}}{\Theta_N}\right)\right\rceil . \tag{18}$$

We discard the values of N except for $\{1, 2, 4, 6, 9, 12, 16, 20\}$, because for $N = 25$, 30 the corresponding values of $\lambda_{\min}$ are negative. The values of $\lambda_{\min}$ are listed in Table 1a. These values are independent of the norm of matrix $\check{A}^2$, see (16), and they have been computed by using the symbolic functions `diff` and `solve` from the Symbolic Math Toolbox 5 of MATLAB. The values Θ_N of (17) are listed in Table 1b.

	$\lambda_{\min}$
N=1	28614.3702451495925
N=2	1304.99637514915918
N=4	110.428178898694292
N=6	38.3201292093300207
N=9	17.3255806739152432
N=12	11.2995380153548675
N=16	8.08117035928883672
N=20	6.56678564572528643

(a) Optimal values of $\lambda_{\min}$.

	Θ_N
N=1	$1.3988322173046763 \cdot 10^{-4}$
N=2	$4.5977704110066707 \cdot 10^{-3}$
N=4	$9.0556596644120163 \cdot 10^{-2}$
N=6	$3.6534325997941364 \cdot 10^{-1}$
N=9	1.1543637495804793
N=12	2.3009899711770276
N=16	4.2073703112196084
N=20	6.3959908727565082

(b) Values of Θ_N from (17).

Figure 1: Tables of $\lambda_{\min}$ and Θ_N.

In the final MATLAB implementation of `cosher`, available online in http://personales.upv.es/∼jorsasma/cosher.m, we have applied similar accuracy bound tests to those in [10, p. 6456-6457] to the sum of the highest degree terms of the Hermite series $C_N(\lambda, \tilde{A}^2)$. The tests consist of verifying if the norm of the sum of several highest degree terms is lower than the unit roundoff. If that is the case, then the corresponding series terms can be neglected, permitting to save matrix products. This kind of tests were already performed with the matrix exponential Hermite series in [10], saving matrix products and therefore reducing the cost for some matrices. For a complete description of algorithm `cosher` see MATLAB implementation `cosher.m` mentioned above.

The analysis of rounding errors of Algorithm 1 can be made analogously to the analysis of rounding errors given in [14, p. 293] for polynomials $p_{2m}(A)$ or $q_{2m}(A)$ from the Padé approximation, see (12.25) from [14, p. 293], and the analysis of the rounding error in the evaluation of A^2 given in the same reference. A complete analysis is given in [15].

4. Numerical examples.

In this section we compare MATLAB implementation `cosher` with functions `funm` and `cosm`. `funm` is a MATLAB function that computes matrix cosine and other matrix functions at square matrices using the Schur-Parlett algorithm from [11]. `cosm` is a MATLAB implementation of Algorithm 5.1 proposed in [2] which uses Padé approximants of cosine function (http://www.maths.manchester.ac.uk/ higham/mftoolbox). MATLAB 7.9 (R2009b) implementations were tested on an Intel Core 2 Duo processor at 2.52 GHz with 4 GB main memory. Algorithm accuracy was tested by computing the relative error

$$\mathrm{E} = \frac{\| \cos(A) - \tilde{Y} \|_1}{\|\cos(A)\|_1},$$

where $\tilde{Y}$ is the computed solution and $\cos(A)$ the exact solution. In the tests we did not use any preprocessing/postprocessing in the implemented algorithms. Analogously to the experiments in [16], we found that turning on preprocessing in this algorithm provided similar results to those presented in this section without preprocessing. We used a set of 102 test matrices: forty-seven 10×10 matrices obtained from the function `matrix` of the Matrix Computation Toolbox [17], twenty four 9×9 or 10×10 matrices from the Eigtool MATLAB package (http:/web.comlab.ox.ac.uk/pseudospectra/eigtool/), twenty four matrices from the state-of-the-art of matrix functions [11, 18, 4, 19, 20, 21, 22, 10] and seven from built-in MATLAB functions, sized from 2×2 to 20×20. For a complete description of the set of text matrices see [10]. The "exact" matrix cosine was calculated analytically when possible, and otherwise using MATLAB's Symbolic Math Toolbox with high precision arithmetic.

Tables 2a and 2b show the comparatives `cosher-cosm` and `cosher-funm` for $N \in \{9, 12, 16, 20\}$. The first three rows show the percentages of times that the relative error of the first function is lower, equal or greater than the relative error of the second function. The fourth row shows the ratio of matrix products needed for computing the matrix cosine for over all the test matrices. In the same way as in [10, p. 6459] we have considered that the asymptotic cost in terms of matrix products for solving the multiple right-hand side linear system that appears in Padé algorithm is 4/3. The computational cost of `funm` depends greatly on the eigenvalue distribution of the considered matrix. If $A \in \mathbb{C}^{r \times r}$, this cost is roughly between $28r^3$ flops and $r^4/3$ flops [14, p. 228]. Since the cost of a square matrix product is $2r^3$ flops, we estimated by default that the cost of `funm` is 14 matrix products.

	N=9	N=12	N=16	$N=20$
L	33.33	67.65	84.31	72.55
E	0	0	0	0
G	66.67	32.35	15.69	27.45
R	0.94	0.92	0.92	0.91

(a) Comparative `cosher-cosm`.

	N=9	N=12	N=16	N=20
L	64.71	74.51	77.45	73.53
E	0	0	0	0
G	35.29	25.49	22.55	26.47
R	0.60	0.58	0.58	0.58

(b) Comparative `cosher-funm`.

Figure 2: Comparatives `cosher-cosm` and `cosher-cosm`. The first three rows show the percentage of times that relative error of `cosher` is lower (L), equal (E) or greater (G) than relative error of `cosm` or `funm`. The last row shows the ratio (R) of costs in terms of matrix products between `cosher` and `cosm` (2a), and `cosher` and `funm` (2b).

Figure 3 shows the performances [23] of the functions compared, where α coordinate varies between 1 and 5 in steps equal to 0.1, and p coordinate is the probability that the considered algorithm has a relative error lower than or equal to α-times the smallest error over all the methods, where probabilities are defined over all matrices.

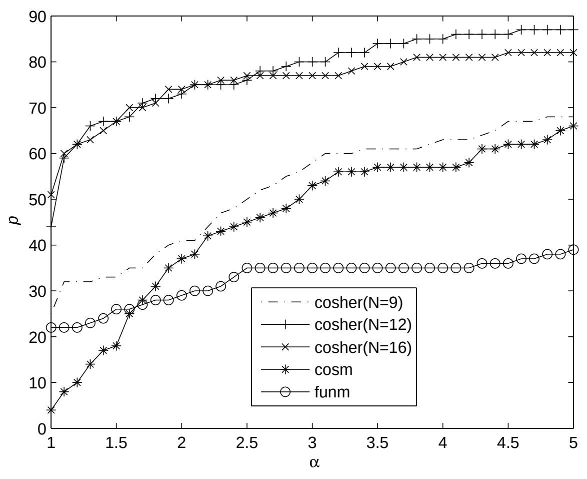

Figure 3: Performance profile of `cosher`, `cosm` and `funm` for the set of test matrices.

From tests, the following conclusions can be emphasized:

- Computational cost of `cosher` is lower than computational costs of `cosm` and `funm` for all considered orders.
- Function `cosher` is more accurate than `cosm` for $N = 12, 16, 20$, and it is more accurate than `funm` for all considered orders.
- In the performed numerical tests, the optimal order of Hermite algorithm is $N = 16$.

5. Conclusions.

In this work an efficient algorithm to compute the matrix cosine based on Hermite matrix polynomial expansions has been proposed, improving the algorithms proposed by the authors in [6, 8]. The new algorithm uses a scaling technique based on the double angle formula, the Horner and Paterson-Stockmeyer's method for computing the Hermite matrix polynomial approximation, a new bound of the absolute error in exact arithmetic, and accuracy bound tests for neglecting higher order series terms. A MATLAB implementation of this algorithm has been compared with the built-in MATLAB function `funm` and the MATLAB function `cosm` based on the Padé algorithm given in [2]. Numerical tests show that the new algorithm has lower computational cost and higher accuracy than both functions `funm` and `cosm` for several orders of the Hermite approximation, reaching its best performance when Hermite approximation of order $N = 16$ is used.

[1] S. M. Serbin, S. A. Blalock, An algorithm for computing the matrix cosine, SIAM Journal on Scientific and Statistical Computing 1 (2) (1980) 198–204.

[2] G. I. Hargreaves, N. J. Higham, Efficient algorithms for the matrix cosine and sine, Numerical Algorithms 40 (2005) 383–400.

[3] S. Serbin, Rational approximations of trigonometric matrices with application to second-order systems of differential equations, Applied Mathematics and Computation 5 (1) (1979) 75–92.

[4] C. B. Moler, C. V. Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Review 45 (2003) 3–49.

[5] S.-T. Yau, Y. Y. Lu, Reducing the symmetric matrix eigenvalue problem to matrix multiplications, SIAM Journal on Scientific Computing 14 (1) (1993) 121–136. doi:10.1137/0914008.

[6] E. Defez, L. Jódar, Some applications of Hermite matrix polynomials series expansions, Journal of Computational and Applied Mathematics 99 (1998) 105–117.

[7] N. J. Higham, M. I. Smith, Computing the matrix cosine, Numerical Algorithms 34 (2003) 13–26.

[8] E. Defez, J. Sastre, J. J. Ibáñez, P. A. Ruiz, Computing matrix functions solving coupled differential models, Mathematical and Computer Modelling 50 (5-6) (2009) 831–839.

[9] M. S. Paterson, L. J. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials, SIAM Journal on Computing 2 (1) (1973) 60–66.

[10] J. Sastre, J. J. Ibáñez, E. Defez, P. A. Ruiz, Efficient orthogonal matrix polynomial based method for computing matrix exponential, Applied Mathematics and Computation 217 (14) (2011) 6451–6463.

[11] P. I. Davies, N. J. Higham, A Schur–Parlett algorithm for computing matrix functions, SIAM Journal Matrix Analysis Applications 25 (2) (2003) 464–485.

[12] L. Jódar, R. Company, Hermite matrix polynomials and second order matrix differential equations, Journal Approximation Theory Application 12 (2) (1996) 20–30.

[13] E. Defez, M. M. Tung, J. Sastre, Improvement on the bound of hermite matrix polynomials, Linear Algebra and its Applications 434 (8) (2011) 1910–1919. doi:10.1016/j.laa.2010.12.015.

[14] N. J. Higham, Functions of Matrices: Theory and Computation, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[15] E. Defez, J. Sastre, J. J. Ibáñez, P. A. Ruiz, Solving engineering models using matrix functions, in: Mathematical Modelling in Engineering & Human Behaviour 2011, UPV-Generalitat Valenciana, 2011, in Press.

[16] N. J. Higham, The scaling and squaring method for the matrix exponential revisited, SIAM Journal Matrix Analysis Applications 26 (4) (2005) 1179–1193.

[17] N. J. Higham, The Test Matrix Toolbox for MATLAB, Numerical Analysis Report No. 237, Manchester, England (Dec. 1993).

[18] R. C. Ward, Numerical computation of the matrix exponential with accuracy estimate, SIAM Journal on Numerical Analysis 14 (4) (1977) 600–610.

[19] I. Najfeld, T. F. Havel, Derivatives of the matrix exponential and their computation, Advances in Applied Mathematics 16 (1995) 321–375.

[20] D. Westreich, A practical method for computing the exponential of a matrix and its integral, Communications in Applied Numerical Methods 6 (1990) 375–380.

[21] Y. Y. Lu, Computing a matrix function for exponential integrators, Journal of Computational and Applied Mathematics 161 (2003) 203–216.

[22] L. Dieci, A. Papini, Padé approximation for the exponential of a block triangular matrix, Linear Algebra and its Applications 308 (2000) 183–202.

[23] E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, Mathematical Programming 91 (2002) 201–213.