On the use of Euler polynomials to approximate the hyperbolic matrix cosine

José M. Alonso$^{\natural 1}$, Emilio Defez* and Javier Ibáñez*

($\natural$) Instituto de Instrumentación para Imagen Molecular,
(*) Instituto de Matemática Multidisciplinar,
Universitat Politècnica de València. Camino de Vera s/n, 46022, Valencia, Spain.

1 Introduction

Coupled partial differential systems are frequent in many different scientific and engineering fields. For instance, coupled hyperbolic systems appear in microwave heating processes [1] and optics [2]. The exact solution of these problems is given in terms of matrix hyperbolic sine and cosine functions [3]. In control theory and dynamical systems, the matrix hyperbolic cosine is used to analyze the stability of linear systems. For example, the response of a linear system to a perturbation can be described using matrix hyperbolic functions [4]. In theoretical physics, the matrix hyperbolic cosine appears in the context of the time evolution of quantum systems and in the solution of time-dependent Schrödinger equations [5]. In the field of artificial intelligence, particularly in the design of neural network architectures, matrix hyperbolic functions have found applications in normalization and stabilization of models training [6].

For a matrix A of dimension $r \times r$, the matrix hyperbolic cosine $\cosh(A)$ is defined using the Taylor series of the hyperbolic cosine function [8]:

$$\cosh(A) = \sum_{k=0}^{\infty} \frac{A^{2k}}{(2k)!}. \tag{1}$$

Alternatively, it can be defined in terms of matrix exponential functions as

$$\cosh(A) = \frac{e^{A} + e^{-A}}{2}. \tag{2}$$

Clearly, this second alternative would be computationally more expensive owing to the need to compute the exponential of two matrices.

Recent Hermite or Bernoulli-based approximations to matrix hyperbolic cosine function can be found in [9] and [10]. In this paper, two new numerical methods to compute this matrix hyperbolic function by means of Euler polynomials have been designed and implemented. Numerical experiments to evaluate their computational performance have been also performed.

[1]jmalonso@dsic.upv.es

2 Euler polynomials

According to [7], Euler polynomials $E_n(x)$ are defined as the coefficients of the generating function

$$g(x,t)=\frac{2e^{tx}}{e^t+1}=\sum_{n\geq 0}\frac{E_n(x)}{n!}t^n\ ,\ |t|<\pi. \tag{3}$$

Euler polynomials $E_n(x)$ has the explicit expression

$$E_n(x)=\sum_{k=0}^{n}\binom{n}{k}\frac{\mathcal{E}_k}{2^k}\left(x-\frac{1}{2}\right)^{n-k}, \tag{4}$$

where $\mathcal{E}_k$ represents the Euler number defined as $\mathcal{E}_k=2^kE_k(1/2)$. Euler numbers satisfy that

$$\mathcal{E}_{2n}=1-\sum_{k=1}^{n}\binom{2n}{2k-1}\frac{2^{2k}(2^{2k}-1)}{2k}\mathcal{B}_{2k}\ ,\ \mathcal{E}_{2n+1}=0, n\geq 0, \tag{5}$$

where $\mathcal{B}_{2k}$ stands for the $2k$-th Bernoulli number.

On the other hand, it is well-known that the exponential of a matrix $A\in\mathbb{C}^{r\times r}$ can be computed by means of the series expansion

$$e^{At}=\frac{e^t+1}{2}\sum_{n\geq 0}\frac{E_n(A)t^n}{n!}\ ,\ |t|<\pi, \tag{6}$$

where the nth Euler matrix polynomial is defined by the expression

$$E_n(A)=\sum_{k=0}^{n}\binom{n}{k}\frac{\mathcal{E}_k}{2^k}\left(A-\frac{1}{2}I\right)^{n-k}. \tag{7}$$

Using Euler polynomials, numerical methods to approximate the matrix exponential e^A and matrix cosine $cos(A)$ were respectively developed in [11] and [12].

From expression (6), it can be obtained that

$$\cosh(A)\ =\ \frac{1}{2}\left(\cosh(1)+1\right)\sum_{n\geq 0}\frac{E_{2n}(A)}{(2n)!}+\frac{1}{2}\sinh(1)\sum_{n\geq 0}\frac{E_{2n+1}(A)}{(2n+1)!}. \tag{8}$$

Note that in the development of $\cosh(A)$ by means of the expression (8), all Euler polynomials are needed (not just the even-numbered), unlike what occurs with Taylor or Hermite polynomials-based approaches.

Notwithstanding, it can be addressed as well that

$$\cosh(A)=\cosh(1/2)\sum_{n\geq 0}\frac{E_{2n}\left(A+\frac{1}{2}I\right)}{(2n)!}, \tag{9}$$

where only even terms appear.

3 The proposed algorithms

By truncating series (8), the m-th order Euler approximation to the matrix hyperbolic cosine is obtained. For the sake of simplicity in exposition, m will be assumed to be even:

$$\cosh(A)\approx P_m(A)=\frac{1}{2}\left(\cosh(1)+1\right)\sum_{n=0}^{m/2}\frac{E_{2n}(A)}{(2n)!}+\frac{1}{2}\sinh(1)\sum_{n=0}^{m/2-1}\frac{E_{2n+1}(A)}{(2n+1)!}, \tag{10}$$

where $P_m(A)$ is a polynomial of order m whose coefficients $p_k^{(m)}$ vary according to the polynomial degree:

$$P_m(A) = \sum_{k=0}^{m} p_k^{(m)} A^k.$$

In a similar way and in accordance with (9), we get that:

$$\cosh(A) \approx \bar{P}_m(A) = \cosh(1/2) \sum_{n=0}^{m/2} \frac{E_{2n}\left(A + \frac{1}{2}I\right)}{(2n)!}, \tag{11}$$

where $\bar{P}_m(A)$ is now a polynomial of order m whose all odd-order coefficients are equal to 0.

Directly from formulas (10) and (11) and together with the scaling and squaring technique, two novel numerical algorithms have been designed, with the specific objective of computing the matrix hyperbolic cosine. For the efficient evaluation of the above matrix polynomials in terms of response time, the Paterson-Stockmeyer algorithm will be used.

4 Numerical experiments

It is intended to compare the numerical and computational performance of the new algorithms with those implementations already present in the literature. In this sense, a set of numerical experiments have been carried out using the following MATLAB codes:

- `cosmh_euler_ataf`: this is an implementation of the algorithm that arises from the expression (10). It uses absolute forward errors and polynomial orders from 42 to 56.
- `cosmh_euler_etab`: it consists of a development starting from the formula (11). It employs absolute backward errors and polynomial orders from 25 to 42.
- `coshmber_ataf` and `coshmber_etrf`: they correspond to the coding of Algorithms 1 and 2 described in [10] to compute the matrix hyperbolic cosine by means of Bernoulli polynomials. They use absolute and relative forward errors, respectively.
- `coshmtayher`: this code is in charge of approximating the hyperbolic cosine by means of the Hermite polynomials, as described in [9].
- `funmcosh`: this is a small function that directly calls the MATLAB built-in function `funm` to approximate the matrix hyperbolic cosine. Function `funm` is based on a Schur decomposition with reordering and blocking, and a block recurrence of Parlett [13].

The test battery used to compare the performance of above codes consists of a series of 128×128 matrices grouped into the following three types of sets:

- **Set 1**: 100 diagonalizable complex matrices generated as $A = V \cdot D \cdot V^{-1}$, where V is an orthogonal matrix and D is a random diagonal matrix with complex eigenvalues. The 2-norm of the matrices took values from 0.1 to 350.
- **Set 2**: 100 non-diagonalizable complex matrices computed as $A = V \cdot J \cdot V^{-1}$. V is an orthogonal matrix and J is a Jordan matrix with complex eigenvalues. The 2-norm ranged from 3.76 to 339.11.
- **Set 3**: 36 matrices from the Matrix Computation Toolbox [14] and 8 from the Eigtool MATLAB Package [15]. Their 2-norm varied from 1 to 5428.98.

Table 1: Improvement percentages of `cosmh_euler_ataf` with respect to the rest of the codes, for the 3 sets of matrices.

	Set 1	Set 2	Set 3
Er(`cosmh_euler_ataf`)<Er(`cosmh_euler_etab`)	72%	72%	58.14%
Er(`cosmh_euler_ataf`)>Er(`cosmh_euler_etab`)	28%	28%	41.86%
Er(`cosmh_euler_ataf`)=Er(`cosmh_euler_etab`)	0%	0%	0%
Er(`cosmh_euler_ataf`)<Er(`coshmber_ataf`)	78%	80%	76.74%
Er(`cosmh_euler_ataf`)>Er(`coshmber_ataf`)	22%	20%	20.93%
Er(`cosmh_euler_ataf`)=Er(`coshmber_ataf`)	0%	0%	2.33%
Er(`cosmh_euler_ataf`)<Er(`coshmber_etrf`)	80%	86%	55.81%
Er(`cosmh_euler_ataf`)>Er(`coshmber_etrf`)	20%	14%	44.19%
Er(`cosmh_euler_ataf`)=Er(`coshmber_etrf`)	0%	0%	0%
Er(`cosmh_euler_ataf`)<Er(`coshmtayher`)	72%	74%	58.14%
Er(`cosmh_euler_ataf`)>Er(`coshmtayher`)	28%	26%	41.86%
Er(`cosmh_euler_ataf`)=Er(`coshmtayher`)	0%	0%	0%
Er(`cosmh_euler_ataf`)<Er(`funmcosh`)	100%	100%	95.35%
Er(`cosmh_euler_ataf`)>Er(`funmcosh`)	0%	0%	4.65%
Er(`cosmh_euler_ataf`)=Er(`funmcosh`)	0%	0%	0%

The normwise relative error $Er(A)$ committed by each of the codes will be measured as indicated below:

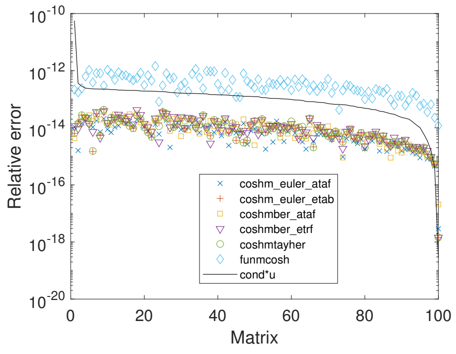
$$Er(A) = \frac{\| \cosh(A) - \widetilde{\cosh}(A) \|_2}{\| \cosh(A) \|_2},$$

where $\cosh(A)$ and $\widetilde{\cosh}(A)$ represent, on the one hand, the exact solution and, on the other hand, the approximate solution obtained by each of the codes.

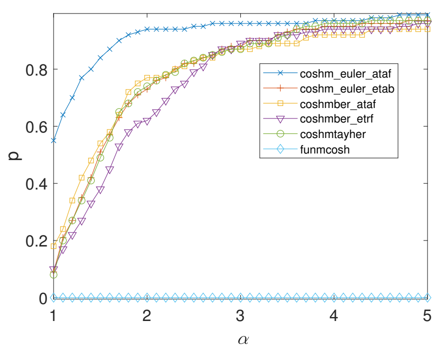
For each of the 3 sets of matrices, Table 1 shows the percentage of cases in which the normwise relative error incurred by code `cosmh_euler_ataf` is lower, higher or equal than that of the other codes used in the comparison. Broadly speaking, it can be appreciated that `cosmh_euler_ataf` is the function that offered the most accurate results, improving the rest of the codes from 72% to 100% of the cases for the matrices of Sets 1 and 2. For the matrices of Set 3, this improvement ranged from 55.81% to 95.35%.

Figures 1, 2 and 3 depict the results of the numerical experiments, respectively for the three types of matrices. In this way, Figures 1a, 2a and 3a show the relative error assumed by each of the codes. The solid line in these graphs represents the relative error expected a priori in each matrix computation. Thus, it is an clear estimate of the numerical method stability. Although results slightly above are accepted as valid, they will be better the further below the line. Obviously, for Sets 1 and 2, all the codes presented values below the mentioned line, except for `funmcosh`. For Set 3, it can be seen how a few matrices exceeded the expected value regardless of the code applied, being much more evident once again for `funmcosh`.

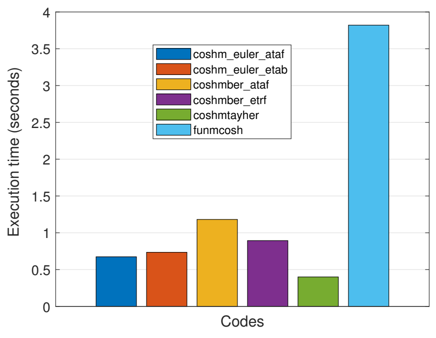
Figures 1b, 2b and 3b display the so-called performance profile. Each point of this graph indicates, expressed as a percentage of one, the percentage of matrices for which the relative error committed by each method is less than or equal to α times the smallest relative error achieved by any of the methods used in the comparison. It is therefore desirable to reach the highest values of the picture at any point of it to become the method that obtains the most accurate



(a) Normwise relative error.

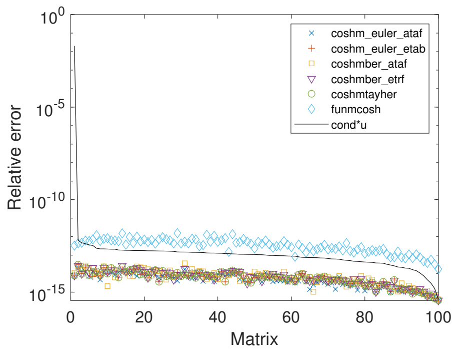


(b) Performance profile.

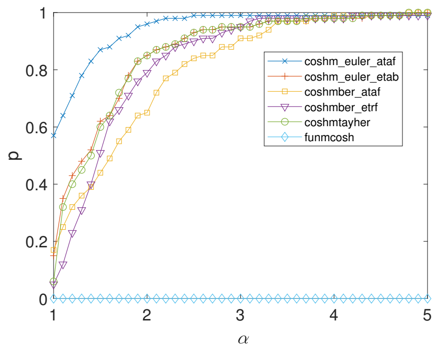


(c) Execution time (in seconds).

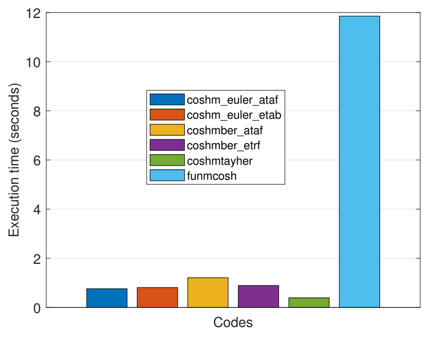
Figure 1: Results for Set 1.



(a) Normwise relative error.

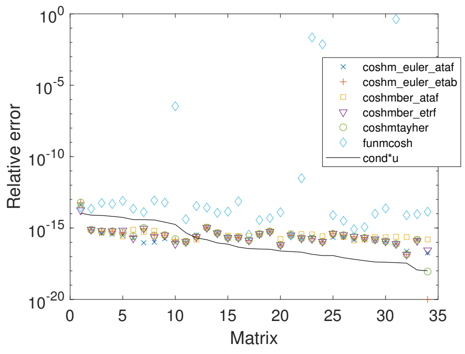


(b) Performance profile.

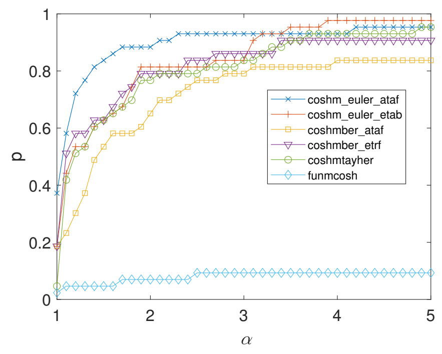


(c) Execution time (in seconds).

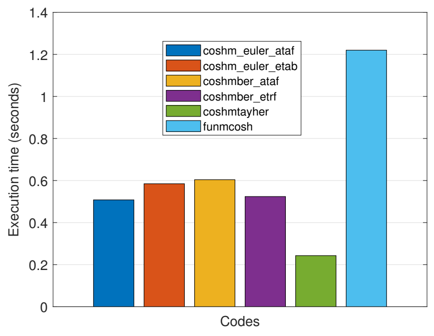
Figure 2: Results for Set 2.



(a) Normwise relative error.



(b) Performance profile.



(c) Execution time (in seconds).

Figure 3: Results for Set 3.

results. We can see that `cosmh_euler_ataf` was the code that presented values in the top part of most of these graphs, as expected according to the results collected in Table 1. With respect to `cosmh_euler_etab`, it can be stated that it offered results that were almost as accurate as the best of the other codes included in the experiments. Obviously, `funmcosh` was the most imprecise.

Finally, Figures 1c, 2c and 3c are in charge of reporting the execution times in the computation of the hyperbolic cosine of the matrices that compose the testbed. As we can appreciate, `coshmtayher` was the code with the shortest response time, followed by `cosmh_euler_ataf`. Slightly higher computation times were achieved by `cosmh_euler_etab`. The codes based on Bernoulli polynomials occupied intermediate positions, with `funmcosh` certainly being the slowest one.

5 Conclusions

The matrix hyperbolic cosine plays an important role in several areas of applied mathematics, with applications ranging from control theory to quantum mechanics and machine learning. In this paper, two numerical methods related to the matrix hyperbolic cosine computation by means of Euler polynomials have been described. Both methods have been implemented and they have been numerically and computationally compared with distinct state-of-the-art codes. It has been noticed that one of the presented methods improved all other codes in terms of accuracy, with very competitive execution times. The other proposed method was somewhat slower, but its results was as accurate as those of the best of the other codes.

Last but no least, it should be emphasized that the fact of providing accurate, robust and efficient methods in the calculation of the matrix hyperbolic cosine not only enriches the mathematical theory, but also its practical applications in a multitude of scientific and engineering disciplines.

Acknowledgments

This work has been supported by the Vicerrectorado de Investigación de la Universitat Politècnica de València (PAID-11-22 and PAID-11-23).

References

[1] Pozar, D., Microwave Engineering. New York, Addison-Wesley, 1991.

[2] Das, P., Optical Signal Processing. New York, Springer, 1991.

[3] Jódar, J., Navarro, E., Posso, A., Casabán, M., Constructive solution of strongly coupled continuous hyperbolic mixed problems. *Applied Numerical Mathematics*, 47(3-4):477–492, 2003.

[4] Van Loan, C. F., The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(1-2):85–100, 1976.

[5] Dieci, L., Papini, A., Computation of the matrix exponential via parabolic and hyperbolic contours. *Numerische Mathematik*, 62(4):451–463, 1992.

[6] Goodfellow, I., Bengio, Y., Courville, A., Deep Learning. MIT Press, 2016.

[7] Olver, F.W., Lozier, D.W., Boisvert, R.F., Clark C.W. (Eds.), NIST handbook of mathematical functions hardback and CD-ROM. Cambridge University Press, 2010.

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

[9] Defez, E., Ibáñez, J., Peinado, J., Alonso-Jordá, P., Alonso, J.M. New Hermite series expansion for computing the matrix hyperbolic cosine. *Journal of Computational and Applied Mathematics*, 408, 114084, 2022.

[10] Alonso, J.M., Ibáñez, J., Defez, E., Alvarruiz, F. Accurate approximation of the matrix hyperbolic cosine using Bernoulli polynomials. *Mathematics*, 11(3), 520, 2023.

[11] Alonso, J.M., Ibáñez, J., Defez, E., Alonso-Jordá, P. Euler polynomials for the matrix exponential approximation. *Journal of Computational and Applied Mathematics*, 115074, 2023.

[12] Alonso, J.M., Defez, E., Ibáñez, J., Sastre, J. On the use of Euler polynomials to approximate the matrix cosine. *Modelling for Engineering and Human Behaviour 2023*, 427–433, 2023.

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

[14] Higham, N.J. The Matrix Computation Toolbox. 2002. Available online: http://www.ma.man.ac.uk/ higham/mctoolbox.

[15] Wright, T.G. Eigtool, Version 2.1. 2009. Available online: http://www.comlab.ox.ac.uk/pseudospectra/eigtool.