Computing matrix functions by matrix Bernoulli series

Emilio Defez, Javier Ibáñez, Jesús Peinado, Pedro Alonso-Jordá, and **José M. Alonso**

Universitat Politècnica de València, Spain

19th International Conference Computational and Mathematical Methods in Science and Engineering
2019 CMMSE

Introduction

Computation of matrix functions has received attention in the last years because of its several applications in different areas of science and technology. Of all matrix functions, the exponential matrix e^A, $A \in \mathbb{C}^{n\times n}$, stands out due to both its applications in the resolution of systems of differential equations and the difficulty of its computation, see [1].

Among all the proposed methods to approximate the exponential matrix, there are two fundamental ones:

- those based on *rational Padé approximations* [2], and
- those based on *polynomial approximations* using either Taylor series developments [3] or serial developments of Hermite matrix polynomials [4].

As demonstrated in the recent past years, polynomial approximations are generally more efficient than approximations based on the Padé algorithm since they are more accurate despite a slightly higher cost. Polynomial approximations use the basic property of *Scaling-Squaring*, based on the relationship

$$e^A = \left(e^{A/2^s}\right)^{2^s}.$$

For a given a matrix A, the algorithm determines a scaling factor s to compute an approximation $P_m(A/2^s)$ for $e^{A/2^s}$, so that

$$e^A \approx \left(P_m(A/2^s)\right)^{2^s}.$$

Bernoulli polynomials and ***Bernoulli numbers*** have been extensively used in several areas of mathematics, as number theory, and they appear in many mathematical formulas, such as the residual term of the Euler-Maclaurian quadrature rule [5, p. 63], the Taylor series expansion of the trigonometric functions $\tan(x), \csc(x)$ and $\cot(x)$ [5, p. 116-117] and in the Taylor series expansion of the hyperbolic function $\tanh(x)$ [5, p. 125]. They are also employed in the well known exact expression of the even values of the Riemann zeta function:

$$\xi(2k) = \sum_{i\geq 1} \frac{1}{i^{2k}} = \frac{(-1)^{k-1} B_{2k} (2\pi)^{2k}}{2(2k)!}, k \geq 1.$$

Moreover, they are even used for solving other problems, such as initial value problem [6], boundary value problem [7], etc. An excelent survey about Bernoulli polynomials and its applicacions can be found in [8].

Our proposal

This paper presents a new series development of the exponential matrix in terms of the Bernoulli matrix polynomials which shows that the polynomial approximations of the exponential matrix are more accurate and less computationally expensive in most cases than those based on Padé approximants.

On Bernoulli matrix polynomials

Bernoulli polynomials $B_m(x)$ are defined in [5, p.588] as the coefficients of the generating function

$$g(x,t) = \frac{te^{tx}}{e^t - 1} = \sum_{m\geq 0} \frac{B_m(x)}{m!} t^m \ , \ |t| < 2\pi, \tag{1}$$

where $g(x,t)$ is an holomorphic function in $\mathbb{C}$ for the variable t, with an avoidable singularity in $t = 0$.
A Bernoulli polynomial $B_m(x)$ has the explicit expression

$$B_m(x) = \sum_{k=0}^{m} \binom{m}{k} B_k x^{m-k}.$$

where the Bernoulli numbers are defined by $B_m = B_m(0)$. Therefore, it follows that the Bernoulli numbers satisfy

$$\frac{z}{e^z - 1} = \sum_{k\geq 0} \frac{B_k}{k!} z^k \ , \ |z| < 2\pi,$$

where

$$B_k = -\sum_{i=0}^{k-1} \binom{k}{i} \frac{B_i}{k+1-i} \ , \ k \geq 1 \ ,$$

with $B_0 = 1$. Note that $B_3 = B_5 = \cdots = B_{2k+1} = 0$, for $k \geq 1$. Thus, for a matrix $A \in \mathbb{C}^{n\times n}$, we define the m-th Bernoulli matrix polynomial by the expression

$$B_m(A) = \sum_{k=0}^{m} \binom{m}{k} B_k A^{m-k}.$$

In this way, the exponential of matrix A can be computed as

$$e^{At} = \left(\frac{e^t - 1}{t}\right) \sum_{k\geq 0} \frac{B_k(A) t^k}{k!} \ , \ |t| < 2\pi, \tag{2}$$

where $B_k(A)$ is the k-th Bernoulli matrix polynomial. If we take s as the scaling value of matrix A and $t = 1$ in (2), then we can compute the matrix exponential approximation as

$$e^{A2^{-s}} \approx (e-1) \sum_{k=0}^{m} \frac{B_k\left(A2^{-s}\right)}{k!}. \tag{3}$$

To use this formula operatively for the approximation of the exponential matrix, we must determine, for a given matrix A, the scaling factor s and the degree m of the approximation (3).

Numerical Experiments

- We have compared the next three routines to compute the exponential of a matrix:
  - → **expade**: a MATLAB function based on the Padé rational approximation for the matrix exponential [9];
  - → **exptay**: a MATLAB code based on the Taylor series evaluated by means of Paterson-Stockmeyer [10].
  - → **expber**: the new MATLAB function based on Bernoulli series presented in this paper;
- The test has consisted on 256 matrices of size 128×128 taken from the next 4 types:
  - → 100 diagonalizable matrices with real and complex eigenvalues.
  - → 100 non-diagonalizable complex matrices.
  - → 40 real matrices from the function **matrix** of the Matrix Computation Toolbox [11].
  - → 16 matrices taken from the *Eigtool MATLAB package* [12].
- The ***"exact"*** matrix exponential has been computed using MATLAB symbolic versions of a scaled Padé rational approximation and a scaled Taylor Paterson-Stockmeyer approximation, both with 256 decimal digit arithmetic and several orders m and/or scaling parameters s, all of them higher than the ones used by **expade** and **exptay**, respectively. We checked that their relative difference was small enough. The algorithm accuracy was tested by computing the relative error

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

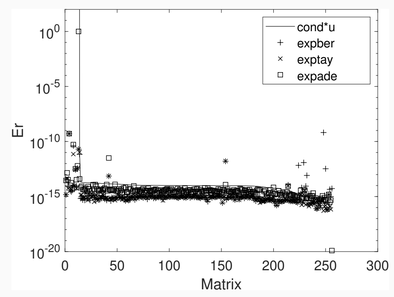
where $\tilde{Y}$ is the computed solution and $\exp(A)$ is the exact one. We also have used MATLAB function **funm_condest1** to estimate the condition number of the matrix 1-norm.

- Table 1 shows the percentage of cases in which the relative errors of **expber** are, respectively, lower than, greater than, or equal to the relative errors of the other algorithms under test.

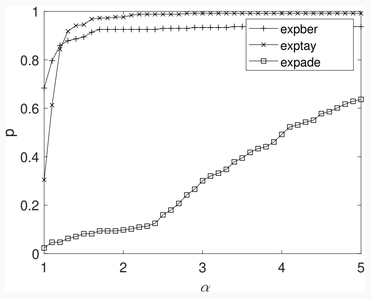
Table 1: Relative error comparison of expber with expade and exptay, respectively.

E(**expber**)<E(**expade**)	91.41%	E(**expber**)<E(**exptay**)	67.97%
E(**expber**)>E(**expade**)	8.59%	E(**expber**)>E(**exptay**)	30.47%
E(**expber**)=E(**expade**)	0.00%	E(**expber**)=E(**exptay**)	1.56%

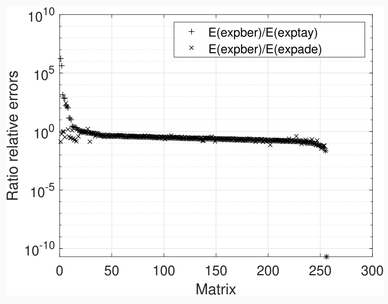
- Figure 2 shows some results of this test:
  - → Figure 2 a) shows the normwise relative errors. The solid line is the function $k_{\exp}u$, where $k_{\exp}$ is the condition number of matrix exponential function [13, Chapter 3] and $u = 2^{-53}$ is the unit roundoff in the double precision floating-point arithmetic.
  - → In the performance profile (Fig. 2 b)), the α coordinate varies between 1 and 5 in steps equal to 0.1, and the 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 methods.
  - → The ratios of relative errors (Fig. 2 c)) are presented in decreasing order with respect to $E($**expber**$)/E($**exptay**$)$ and $E($**expber**$)/E($**expade**$)$.
  - → Figure 2 d) shows the ratio of the computational cost measured in the number of matrix products.



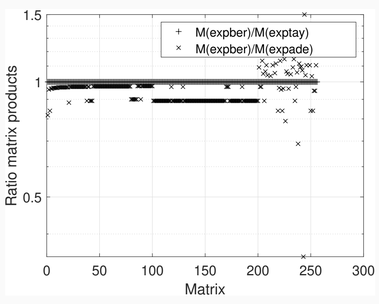
a) Normwise relative errors.



b) Performance Profile.



c) Relative error ratio.



d) Matrix product ratio.

Fig. 2: Results of the test.

- We have taken advantage of the previous developments regarding the MEX files for the **exptay** code in the new one, thus, both **exptay** and **expber** are able to exploit an existing NVIDIA GPU in the node.
- Table 2 shows the performance of **exptay** and **expber** for large matrices in both the CPU and GPU.
  - → The CPU is formed by two processors with 20 cores each (Intel Xeon CPU E5-2698 v4 @2.20GHz).
  - → The GPU is an NVIDIA Tesla P100-SXM2.

Table 2: Comparison in time (sec.) for large matrices in CPU and GPU.

	$n=$	1000	1500	2000	2500	3000	3500	4000	4500	5000	5500	6000
exptay	CPU	0.23	0.51	0.90	1.88	2.54	4.11	5.17	7.62	10.29	13.22	17.30
	GPU	0.11	0.18	0.27	0.53	0.83	1.22	1.73	2.29	2.89	3.47	4.71
expber	CPU	0.23	0.85	1.06	1.92	2.78	4.02	5.48	7.50	10.11	14.12	17.00
	GPU	0.10	0.27	0.29	0.57	0.89	1.35	1.81	2.37	3.11	3.51	4.47

Conclusions

- All the three implementations, i.e. **expade**, **exptay**, and **expber**, have similar numerical stability. Functions based on polynomial approximations are more accurate than the one based on Padé approximants, being the new function **expber** slightly more accurate than our former code **exptay**.
- In terms of matrix products, **expber** and **exptay** perform exactly the same and both have a 7% lower computational cost than **expade**.
- The executions carried out in our test showed that the execution time of the **exptay** and **expber** is very similar as expected.
- Using our implementation for GPU, both routines **exptay** and **expber** perform better than the CPU version in the target machine used in the experiments. In the whole range ($n = 1000, \ldots, 6000$) the speed up ranges from 2.1 to 3.8.

Acknowledgements

This work has been partially supported by Spanish Ministerio de Economía y Competitividad and European Regional Development Fund (ERDF) grants TIN2017-89314-P and by the Programa de Apoyo a la Investigación y Desarrollo 2018 of the Universitat Politècnica de València (PAID-06-18) grants SP20180016.

References

[1] C. Van Loan. *A study of the matrix exponential*, Numerical Analysis Report, Tech. rep., Manchester Institute for Mathematical Sciences, Manchester University (2006).

[2] G. A. Baker, P. R. Graves-Morris. *Padé Approximants*, Encyclopedia of Mathematics and its Applications Edition, Cambridge University Press, 1996.

[3] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz. *New scaling-squaring taylor algorithms for computing the matrix exponential*, SIAM Journal on Scientific Computing 37 (1) (2015) A439–A455.

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

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

[6] E. Tohidi, K. Erfani, M. Gachpazan, S. Shateyi. *A new Tau method for solving nonlinear Lane-Emden type equations via Bernoulli operational matrix of differentiation*, Journal of Applied Mathematics 2013 (2013).

[7] A. W. Islam, M. A. Sharif, E. S. Carlson. *Numerical investigation of double diffusive natural convection of co_2 in a brine saturated geothermal reservoir*, Geothermics 48 (2013) 101–111.

[8] O. Kouba. Lecture Notes, *Bernoulli Polynomials and Applications*, arXiv preprint arXiv:1309.7560 (2013).

[9] A. H. Al-Mohy, N. J. Higham, S. D. Relton. *New algorithms for computing the matrix sine and cosine separately or simultaneously*. SIAM J. Sci. Comput. 37 (1) A456-A487, 2015.

[10] J. Sastre, J. Ibáñez, P. Alonso, J. Peinado, E. Defez. *Two algorithms for computing the matrix cosine function*. Applied Mathematics and Computation, v. 312, pp. 66–77, 2017.

[11] N. J. Higham. *The Test Matrix Toolbox for MATLAB*. Numerical Analysis Report No. 237, Manchester, Eng., 1993.

[12] T. G. Wright. *Eigtool, version 2.1*. `web.comlab.ox.ac.uk/pseudospectra/eigtool`, 2009.

[13] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. SIAM, Philadelphia, PA, USA, 2008.