mathematics

MDPI

Article

Advances in the Approximation of the Matrix Hyperbolic Tangent

Javier Ibáñez [1], José M. Alonso [1], Jorge Sastre [2], Emilio Defez [3] and Pedro Alonso-Jordá [4,*]

[1] Instituto de Instrumentación para Imagen Molecular, Universitat Politècnica de València, Av. dels Tarongers, 14, 46011 Valencia, Spain; jjibanez@dsic.upv.es (J.I.); jmalonso@dsic.upv.es (J.M.A.)
[2] Instituto de Telecomunicación y Aplicaciones Multimedia, Universitat Politècnica de València, Ed. 8G, Camino de Vera s/n, 46022 Valencia, Spain; jsastrem@upv.es
[3] Instituto de Matemática Multidisciplinar, Universitat Politècnica de València, Ed. 8G, Camino de Vera s/n, 46022 Valencia, Spain; edefez@imm.upv.es
[4] Department of Computer Systems and Computation, Universitat Politècnica de València, Ed. 1F, Camino de Vera s/n, 46022 Valencia, Spain
* Correspondence: palonso@upv.es

Abstract: In this paper, we introduce two approaches to compute the matrix hyperbolic tangent. While one of them is based on its own definition and uses the matrix exponential, the other one is focused on the expansion of its Taylor series. For this second approximation, we analyse two different alternatives to evaluate the corresponding matrix polynomials. This resulted in three stable and accurate codes, which we implemented in MATLAB and numerically and computationally compared by means of a battery of tests composed of distinct state-of-the-art matrices. Our results show that the Taylor series-based methods were more accurate, although somewhat more computationally expensive, compared with the approach based on the exponential matrix. To avoid this drawback, we propose the use of a set of formulas that allows us to evaluate polynomials in a more efficient way compared with that of the traditional Paterson–Stockmeyer method, thus, substantially reducing the number of matrix products (practically equal in number to the approach based on the matrix exponential), without penalising the accuracy of the result.

Keywords: matrix functions; matrix hyperbolic tangent; matrix exponential; Taylor series; matrix polynomial evaluation

check for updates

Citation: Ibáñez, J.; Alonso, J.M.; Sastre, J.; Defez, E.; Alonso-Jordá, P. Advances in the Approximation of the Matrix Hyperbolic Tangent. *Mathematics* **2021**, *9*, 1219. https://doi.org/10.3390/math9111219

Academic Editor: Mariano Torrisi

Received: 23 March 2021
Accepted: 20 May 2021
Published: 27 May 2021

1. Introduction and Notation

Matrix functions have been an increasing focus of attention due to their applications to new and interesting problems related, e.g., to statistics [1], Lie theory [2], differential equations (the matrix exponential function e^{At} can be considered as a classical example for its application in the solution of first order differential systems $Y'(t) = AY(t)$ with $A \in \mathbb{C}^{n\times n}$ and for its use in the development of exponential integrators for nonlinear ODEs and PDEs, see [3] for example), approximation theory, and many other areas of science and engineering [4].

There are different ways to define the notion of the function $f(A)$ of a square matrix A. The most common are via the Jordan canonical form, via the Hermite interpolation, and via the Cauchy integral formula. The equivalence among the different definitions of a matrix function can be found in [5]. Several general methods have been proposed for evaluating matrix functions, among which, we can highlight the Taylor or Padé approximations and methods based on the Schur form of a matrix [4].

Among the most well-known matrix functions, we have the matrix hyperbolic cosine and the matrix hyperbolic sine functions, respectively defined in terms of the matrix exponential function e^A by means of the following expressions:

$$\cosh(A) = \frac{1}{2}\left(e^{A} + e^{-A}\right),\ \sinh(A) = \frac{1}{2}\left(e^{A} - e^{-A}\right). \tag{1}$$

These matrix functions are applied, e.g., in the study of the communicability analysis in complex networks [6], or to construct the exact series solution of coupled hyperbolic systems [7]. Precisely due to their applicability, the numerical computation of these functions has received remarkable and growing attention in recent years. A set of state-of-the-art algorithms to calculate these functions developed by the authors can be found in [8–11].

On the other hand, we have the matrix hyperbolic tangent function, defined as

$$\tanh(A) = \sinh(A)(\cosh(A))^{-1} = (\cosh(A))^{-1}\sinh(A), \tag{2}$$

and used, for instance, to give an analytical solution of the radiative transfer equation [12], in the heat transference field [13,14], in the study of symplectic systems [15,16], in graph theory [17], and in the development of special types of exponential integrators [18,19].

In this work, we propose and study two different implementations that compute the matrix hyperbolic tangent function: the first uses the matrix exponential function whereas the second is based on its Taylor series expansion. In addition, for the second approach, we use and compare two different alternatives to evaluate the matrix polynomials involved in the series expansion.

1.1. *The Matrix Exponential Function-Based Approach*

This first option is derived from the matrix hyperbolic tangent function definition as expressed in Equations (1) and (2), from which the following matrix rational expression is immediately deduced:

$$\tanh(A) = \left(e^{2A} - I\right)\left(e^{2A} + I\right)^{-1} = \left(e^{2A} + I\right)^{-1}\left(e^{2A} - I\right), \tag{3}$$

where I denotes the identity matrix with the same order as A. Equation (3) reduces the approximation of the matrix hyperbolic tangent function to the computation of the matrix exponential e^{2A}.

There exists profuse literature (see e.g., [4,20]) about the approximation of the matrix exponential function and the inconveniences that its calculation leads to [21]. The most competitive methods used in practice are either those based on polynomial approximations or those based on Padé rational approaches, with the former, in general, being more accurate and with lower computational costs [22].

In recent years, different polynomial approaches to the matrix exponential function have been proposed, depending on the type of matrix polynomial used. For example, some approximations use the Hermite matrix polynomials [23], while others derived from on Taylor polynomials [22,24]. More recently, a new method based on Bernoulli matrix polynomials was also proposed in [25].

All these methods use the scaling and squaring method based on the identity

$$e^{A} = \left(e^{2^{-s}A}\right)^{2^{s}},$$

which satisfies the matrix exponential. In the scaling phase, an integer scaling factor s is taken, and the approximation of $e^{2^{-s}A}$ is computed using any of the proposed methods so that the required precision is obtained with the lowest possible computational cost. In the squaring phase, we obtain e^{A} by s repeated squaring operations.

1.2. The Taylor Series-Based Approach

The other possibility for computing the matrix hyperbolic tangent function is to use its Taylor series expansion

$$\tanh(z) = \sum_{k\geq 1} \frac{2^{2k}(2^{2k}-1)\mathcal{B}_{2k}}{(2k)!} z^{2k-1}, |z| < \frac{\pi}{2},$$

where $\mathcal{B}_{2k}$ are the Bernoulli's numbers.

As in the case of the matrix exponential, it is highly recommended to use the scaling and squaring technique to reduce the norm of the matrix to be computed and, thus, to obtain a good approximation of the matrix hyperbolic tangent with an acceptable computational cost. Due to the double angle formula for the matrix hyperbolic tangent function

$$\tanh(2A) = 2\left(I + \tanh^2(A)\right)^{-1} \tanh(A), \tag{4}$$

which is derived from the scalar one

$$\tanh(2z) = \frac{2\tanh(z)}{1+\tanh^2(z)},$$

it is possible to compute $T_s = \tanh(A)$ by using the following recurrence:

$$\begin{aligned} T_0 &= \tanh(2^{-s}A), \\ T_i &= 2\left(I + T_{i-1}^2(A)\right)^{-1} T_{i-1}(A), i = 1, \ldots, s. \end{aligned} \tag{5}$$

Throughout this paper we will denote by $\sigma(A)$ the set of eigenvalues of matrix $A \in \mathbb{C}^{n\times n}$ and by I_n (or I) the matrix identity of order n. In addition, $\rho(A)$ refers to the spectral radius of A, defined as

$$\rho(A) = \max\{|\lambda|; \lambda \in \sigma(A)\}.$$

With $\lceil x \rceil$, we denote the value obtained by rounding x to the nearest integer greater than or equal to x, and $\lfloor x \rfloor$ is the value obtained rounding x to the nearest integer less than or equal to x. The matrix norm $||\cdot||$ will stand for any subordinate matrix norm and, in particular, $||\cdot||_1$ denotes the 1-norm.

This work is organised as follows. First, Section 2 incorporates the algorithms corresponding to the different approaches previously described for approximating the matrix hyperbolic tangent and for computing the scaling parameter and the order of the Taylor polynomials. Next, Section 3 details the experiments carried out to compare the numerical properties of the codes to be evaluated. Finally, in Section 4, we present our conclusions.

2. Algorithms for Computing the Matrix Hyperbolic Tangent Function

2.1. The Matrix Exponential Function-Based Algorithm

The first algorithm designed, called Algorithm 1, computes the matrix hyperbolic tangent by means of the matrix exponential according to Formula (3). In Steps 1 and 2, Algorithm 2 from [26] is responsible for computing $e^{2^{-s}B}$ by means of the Taylor approximation of order m_k, being $B = 2A$. In Step 3, $T \simeq \tanh(2^{-s}B)$ is worked out using Formula (3). In this phase, T is computed by solving a system of linear equations, with $\left(e^{2^{-s}B} + I\right)$ being the coefficient matrix and $\left(e^{2^{-s}B} - I\right)$ being the right hand side term. Finally, through Steps 4–8, $\tanh(A)$ is recovered from T by using the squaring technique and the double angle Formula (5).

Algorithm 1: Given a matrix $A \in \mathbb{C}^{n\times n}$, this algorithm computes $T = \tanh(A)$ by means of the matrix exponential function.

```
1 $B = 2A$
2 Calculate the scaling factor $s \in \mathbb{N} \cup \{0\}$, the order of Taylor polynomial
    $m_k \in \{2,4,6,9,12,16,20,25,30\}$ and compute $e^{2^{-s}B}$ by using the Taylor
    approximation /* Phase I (see Algorithm 2 from [26]) */
3 $T = \left(e^{2^{-s}B} + I\right)^{-1}\left(e^{2^{-s}B} - I\right)$ /* Phase II: Work out $\tanh(2^{-s}B)$ by (3) */
4 for $i = 1$ to $s$ do /* Phase III: Recover $\tanh(A)$ by (5) */
5     $B = I + T^2$
6     Solve for $X$ the system of linear equations $BX = 2T$
7     $T = X$
8 end
```

Algorithm 2: Given a matrix $A \in \mathbb{C}^{n\times n}$, this algorithm computes $T = \tanh(A)$ by means of the Taylor approximation Equation (8) and the Paterson–Stockmeyer method.

```
1 Calculate the scaling factor $s \in \mathbb{N} \cup \{0\}$, the order of Taylor approximation
    $m_k \in \{2,4,6,9,12,16,20,25,30\}$, $2^{-s}A$ and the required matrix powers of $4^{-s}B$
    /* Phase I (Algorithm 4) */
2 $T = 2^{-s}AP_{m_k}(4^{-s}B)$ /* Phase II: Compute Equation (8) */
3 for $i = 1$ to $s$ do /* Phase III: Recover $\tanh(A)$ by Equation (5) */
4     $B = I + T^2$
5     Solve for $X$ the system of linear equations $BX = 2T$
6     $T = X$
7 end
```

2.2. *Taylor Approximation-Based Algorithms*

Let

$$f(z) = \sum_{n\geq 1} \frac{2^{2n}(2^{2n}-1)\mathcal{B}_{2n}}{(2n)!} z^{2n-1} \tag{6}$$

be the Taylor series expansion of the hyperbolic tangent function, with the radius of convergence $r = \pi/2$, where $\mathcal{B}_{2n}$ are the Bernoulli's numbers, defined by the recursive expression

$$\mathcal{B}_0 = 1\,,\ \mathcal{B}_k = -\sum_{i=0}^{k-1} \binom{k}{i} \frac{\mathcal{B}_i}{k+1-i}, k \geq 1.$$

The following proposition is easily obtained:

Proposition 1. *Let $A \in \mathbb{C}^{n\times n}$ be a matrix satisfying $\rho(A) < \pi/2$. Then, the matrix hyperbolic tangent* $\tanh(A)$ *can be defined for A as the Taylor series*

$$\tanh(A) = f(A) = \sum_{n\geq 1} \frac{2^{2n}(2^{2n}-1)\mathcal{B}_{2n}}{(2n)!} A^{2n-1}. \tag{7}$$

From [4] Theorem 4.7, this series in Equation (7) converges if the distinct eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_t$ of A satisfy one of these conditions:

1. $|\lambda_i| < \pi/2$.
2. $|\lambda_i| = \pi/2$ and the series $f^{(n_i-1)}(\lambda)$, where $f(z)$ is given by Equation (6) and n_i is the index of λ_i, is convergent at the point $\lambda = \lambda_i$, $i = 1, \ldots, t$.

To simplify the notation, we denote with

$$\tanh(A) = \sum_{k\geq 0} q_{2k+1} A^{2k+1},$$

the Taylor series (7), and with

$$T_{2m+1}(A) = \sum_{k=0}^{m} q_{2k+1} A^{2k+1} = A \sum_{k=0}^{m} p_k B^k = A P_m(B), \tag{8}$$

the Taylor approximation of order $2m+1$ of $\tanh(A)$, where $B = A^2$.

There exist several alternatives that can be applied to obtain $P_m(B)$, such as the Paterson–Stockmeyer method [27] or the Sastre formulas [28], with the latter being more efficient, in terms of matrix products, compared with the former.

Algorithm 2 works out $\tanh(A)$ by means of the Taylor approximation of the scaled matrix $4^{-s}B$ Equation (8). In addition, it uses the Paterson–Stockmeyer method for the matrix polynomial evaluation, and finally it applies the recurrence Equation (5) for recovering $\tanh(A)$.

Phase I of Algorithm 2 is in charge of estimating the integers m and s so that the Taylor approximation of the scaled matrix B is computed accurately and efficiently. Then, in Phase II, once the integer m_k has been chosen from the set

$$\mathbb{M} = \{2, 4, 6, 9, 12, 16, 20, 25, 30, \ldots\},$$

and powers B^i, $2 \leq i \leq q$ are calculated, with $q = \lceil \sqrt{m_k} \rceil$ or $q = \lfloor \sqrt{m_k} \rfloor$ as an integer divisor of m_k, the Paterson–Stockmeyer method computes $P_{m_k}(B)$ with the necessary accuracy and with a minimal computational cost as

$$\begin{aligned} P_{m_k}(B) = (((&p_{m_k} B^q + p_{m_k-1} B^{q-1} + p_{m_k-2} B^{q-2} + \cdots + p_{m_k-q+1} B + p_{m_k-q} I) B^q \\ &+ p_{m_k-q-1} B^{q-1} + p_{m_k-q-2} B^{q-2} + \cdots + p_{m_k-2q+1} B + p_{m_k-2q} I) B^q \\ &+ p_{m_k-2q-1} B^{q-1} + p_{m_k-2q-2} B^{q-2} + \cdots + p_{m_k-3q+1} B + p_{m_k-3q} I) B^q \\ &\ldots \\ &+ p_{q-1} B^{q-1} + p_{q-2} B^{q-2} + \cdots + p_1 B + p_0 I. \end{aligned} \tag{9}$$

Taking into account Equation (9), the computational cost of Algorithm 2 is $O\big((2k + 4 + \frac{8s}{3}) n^3\big)$ flops.

Finally, in Phase III, the matrix hyperbolic tangent of matrix A is recovered by squaring and repeatedly solving a system of linear equations equivalent to Equation (4).

With the purpose of evaluating $P_m(B)$ in Equation (8) in a more efficient way compared with that offered by the Paterson–Stockmeyer method, as stated in the Phase II of Algorithm 2, the formulas provided in [28] were taken into consideration into the design of Algorithm 3. Concretely, we use the evaluation formulas for Taylor-based matrix polynomial approximations of orders $m = 8$, 14+, and 21+ in a similar way to the evaluation described in [22] (Sections 3.1–3.3) for the matrix exponential function. Nevertheless, the Paterson–Stockmeyer method is still being used for orders equal to $m = 2$ and $m = 4$.

Following the notation given in [22] (Section 4) for an order m, the suffix "+" in $m = 14+$ and $m = 21+$ means that these Taylor approximations are more accurate than those approximations of order $m = 14$ and $m = 21$, respectively, since the former will be composed of a few more polynomial terms. The coefficients of these additional terms will be similar but not identical to the corresponding traditional Taylor approximation ones. It is convenient to clarify that we have used the order $m = 14+$, instead of the order $m = 15+$, because we have not found a real solution for the coefficients of the corresponding evaluation formula with order $m = 15+$.

The evaluation formulas for the order $m = 8$ that comprise the system of non-linear equations to be solved for determining the unknown coefficients $c_i, i = 1, \ldots, 6$, are:

$$\begin{aligned} B &= -A^2, \\ B_2 &= B^2, \\ y_0(B) &= B_2(c_1 B_2 + c_2 B), \\ y_1(B) &= A((y_0(B) + c_3 B_2 + c_4 B)(y_0(B) + c_5 B_2) + c_6 y_0(B) \\ &\quad +2B_2/15 + B/3 + I), \end{aligned} \tag{10}$$

where

$$y_1(B) = T_{17}(A) = AP_8(B),$$

and $T_{17}(A)$, or $AP_8(B)$, refers to the Taylor polynomial of order 17 of function $\tanh(A)$ given by Equation (8).

Algorithm 3: Given a matrix $A \in \mathbb{C}^{n \times n}$, this algorithm computes $T = \tanh(A)$ by means of the Taylor approximation Equation (8) and the Sastre formulas.

```
1 Calculate the scaling factor $s \in \mathbb{N} \cup \{0\}$, the order of Taylor approximation
  $m_k \in \{2, 4, 8, 14, 21\}$, $2^{-s}A$ and the required matrix powers of $4^{-s}B$ /* Phase I
  (Algorithm 5) */
2 $T = 2^{-s}AP_{m_k}(4^{-s}B)$ /* Phase II: Compute Equation (8) */
3 for $i = 1$ to $s$ do /* Phase III: Recover tanh($A$) by Equation (5) */
4     $B = I + T^2$
5     Solve for $X$ the system of linear equations $BX = 2T$
6     $T = X$
7 end
```

Regarding the non-linear equations for order $m = 14+$ and its unknown coefficients $c_i, i = 1, \ldots, 13$, we have

$$\begin{aligned} y_0(B) &= B_2(c_1 B_2 + c_2 B), \\ y_1(B) &= (y_0(B) + c_3 B_2 + c_4 B)(y_0(B) + c_5 B_2) + c_6 y_0(B), \\ y_2(B) &= A((y_1(B) + c_7 y_0(B) + c_8 B_2 + c_9 B)(y_1(B) + c_{10} B_2 + c_{11} B) \\ &\quad +c_{12} y_1 + c_{13} B_2 + B/3 + I), \end{aligned} \tag{11}$$

where

$$y_2(B) = A(P_{14} + b_{15} B^{15} + b_{16} B^{16}),$$

and AP_{14} represents the Taylor polynomial of order 29 of function $\tanh(A)$ given by Equation (8). If we denote as p_{15} and p_{16} the Taylor polynomial coefficients corresponding to the powers B^{15} and B^{16}, respectively, the relative error of coefficients b_{15} and b_{16} with respect to them, with two decimal digits, are:

$$\begin{aligned} |(b_{15} - p_{15})/p_{15}| &= 0.38, \\ |(b_{16} - p_{16})/p_{16}| &= 0.85. \end{aligned}$$

Taking

$$B_3 = B_2 B,$$

the evaluation formulas related to the system of non-linear equations for order $m = 21+$ with the coefficients $c_i, i = 1, \ldots, 21$ to be determined are

$$\begin{aligned} y_0(B) &= B_3(c_1B_3 + c_2B_2 + c_3B), \\ y_1(B) &= (y_0(B) + c_4B_3 + c_5B_2 + c_6B)(y_0(B) + c_7B_3 + c_8B_2) \\ &\quad + c_9y_0(B) + c_{10}B_3, \\ y_2(B) &= A((y_1(B) + c_{11}B_3 + c_{12}B_2 + c_{13}B)(y_1(B) + c_{14}y_0(B) \\ &\quad + c_{15}B_3 + c_{16}B_2 + c_{17}B) + c_{18}y_1 + c_{19}y_0(B) + c_{20}B_3 \\ &\quad + c_{21}B_2 + B/3 + I), \end{aligned} \tag{12}$$

where

$$y_2(B) = A(P_{21} + b_{22}B^{22} + b_{23}B^{23} + b_{24}B^{24}),$$

and AP_{21} stands for the Taylor polynomial of order 43 of the function $\tanh(A)$ given by Equation (8). With two decimal digits of accuracy, the relative error made by the coefficients b_{22}, b_{23}, and b_{24} with respect to their corresponding Taylor polynomial coefficients p_{22}, p_{23}, and p_{24} that accompany their respective powers B^{22}, B^{23}, and B^{24}, are the following:

$$\begin{aligned} |(b_{22} - p_{22})/p_{22}| &= 0.69, \\ |(b_{23} - p_{23})/p_{23}| &= 0.69, \\ |(b_{24} - p_{24})/p_{24}| &= 0.70. \end{aligned}$$

Similarly to [22] (Sections 3.1–3.3), we obtained different sets of solutions for the coefficients in Equations (10)–(12) using the `vpasolve` function (https://es.mathworks.com/help/symbolic/vpasolve.html, accessed on 7 March 2020) from the MATLAB Symbolic Computation Toolbox with variable precision arithmetic. For the case of $m = 21+$, the random option of `vpasolve` has been used, which allowed us to obtain different solutions for the coefficients, after running it 1000 times. From all the sets of real solutions provided, we selected the most stable ones according to the stability check proposed in [28] (Ex. 3.2).

2.3. Polynomial Order m and Scaling Value s Calculation

The computation of m and s from Phase I in Algorithms 2 and 3 is based on the relative forward error of approximating $\tanh(A)$ by means of the Taylor approximation Equation (8). This error, defined as $E_f = \left\|\tanh(A)^{-1}(I - T_{2m+1})\right\|$, can be expressed as

$$E_f = \sum_{k \geq 2m+2} c_k A^k,$$

and it can be bounded as (see Theorem 1.1 from [29]):

$$E_f = \left\| \sum_{k\geq 2m+2} c_k A^k \right\| = \left\| \sum_{k \geq m+1} \bar{c}_k B^k \right\| \leq \sum_{k\geq m+1} |\bar{c}_k| \beta_m^k \equiv h_m(\beta_m),$$

where $\beta_m = \max\left\{ ||B^k||^{1/k} : \ k \geqslant m+1, \, \bar{c}_{m+1} \neq 0 \right\}$.

Let Θ_m be

$$\Theta_m = \max\left\{ \theta \geq 0 : \sum_{k\geq m+1} |\bar{c}_k| \theta^k \leq u \right\},$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double precision arithmetic. The values of Θ_m can be computed with any given precision by using symbolic computations as is shown in Tables 1 and 2, depending on the polynomial evaluation alternative selected.

Algorithm 4 provides the Taylor approximation order $m_k \in \mathbb{M}$, $lower \leq k \leq upper$, where m_{lower} and m_{upper} are, respectively, the minimum and maximum order used, the scaling factor s, together with $2^{-s}A$, and the necessary powers of $4^{-s}B$ for computing T in

Phase II of Algorithm 2. To simplify reading this algorithm, we use the following aliases: $\beta_k \equiv \beta_{m_k}$ and $\Theta_k \equiv \Theta_{m_k}$.

Algorithm 4: Given a matrix $A \in \mathbb{C}^{n\times n}$, the values Θ from Table 1, a minimum order $m_{lower} \in \mathbb{M}$, a maximum order $m_{upper} \in \mathbb{M}$, with $\mathbb{M} = \{2,4,6,9,12,16,20,25,30\}$, and a tolerance tol, this algorithm computes the order of Taylor approximation $m \in \mathbb{M}$, $m_{lower} \le m_k \le m_{upper}$, and the scaling factor s, together with $2^{-s}A$ and the necessary powers of $4^{-s}B$ for computing $P_{m_k}(4^{-s}B)$ from (9).

```
$B_1 = A^2; k = lower; q = \lceil\sqrt{m_k}\rceil; f = 0$
for $j = 2$ to $q$ do
    $B_j = B_{j-1}B_1$
end
Compute $\beta_k \approx \left\|B^{m_k+1}\right\|^{1/(m_k+1)}$ from $B_1$ and $B_q$ ; /* see [30] */
while $f = 0$ and $k < upper$ do
    $k = k+1$
    if  mod $(k,2) = 1$ then
        $q = \lceil\sqrt{m_k}\rceil$
        $B_q = B_{q-1}B_1$
    end
    Compute $\beta_k \approx \|B^{m_{k+1}}\|^{1/(m_{k+1})}$ from $B_1$ and $B_q$ ; /* see [30] */
    if $|\beta_k - \beta_{k-1}| < tol$ and $\beta_k < \Theta_k$ then
        $f = 1$
    end
end
$s = \max\left(0, \left\lceil \frac{1}{2}\log_2(\beta_k/\Theta_k)\right\rceil\right)$
if $s > 0$ then
    $s_0 = \max\left(0, \left\lceil \frac{1}{2}\log_2(\beta_{k-1}/\Theta_{k-1})\right\rceil\right)$
    if $s_0 = s$ then
        $k = k-1$
        $q = \lceil\sqrt{m_k}\rceil$
    end
    $A = 2^{-s}A$
    for $j = 1$ to $q$ do
        $B_j = 4^{-sj}B_j$
    end
end
$m = m_k$
```

In Steps 1–4 of Algorithm 4, the required powers of B for working out $P_{m_k}(B)$ are computed. Then, in Step 5, β_k is obtained by using Algorithm 1 from [30].

As $\lim_{t\to\infty} ||B^t||^{1/t} = \rho(B)$, where ρ is the spectral radius of matrix B, then $\lim_{m\to\infty} |\beta_m - \beta_{m-1}| = 0$. Hence, given a small tolerance value tol, Steps 6–16 test if there is a value β_k such that $|\beta_k - \beta_{k-1}| < tol$ and $\beta_k < \Theta_k$. In addition, the necessary powers of B for computing $P_{m_k}(B)$ are calculated. Next, the scaling factor $s \ge 0$ is provided in Step 17:

$$s = \max\left(0, \left\lceil \frac{1}{2}\log_2 \frac{\beta_k}{\Theta_k}\right\rceil\right).$$

With those values of m_k and s, we guarantee that:

$$E_f(2^{-s}A) \le h_{m_k}(4^{-s}\beta_k) < h_{m_k}(\Theta_k) < u, \tag{13}$$

i.e., the relative forward error of $T_{2m_k+1}(2^{-s}A)$ is lower than the unit roundoff u.

Step 18 checks whether the matrices A and B should be scaled or not. If so, the algorithm analyses the possibility of reducing the order of the Taylor polynomial, but at the same time ensuring that Equation (13) is verified (Steps 19–23). For this purpose, the scaling value corresponding to the order of the Taylor polynomial immediately below the one previously obtained is calculated as well. If both values are identical, the polynomial order reduction is performed. Once the optimal scaling parameter s has been determined, the matrices A and B are scaled (Steps 24–27).

Algorithm 5 is an adaptation of Algorithm 4, where the orders in the set $\mathbb{M} = \{2, 4, 8, 14, 21\}$ are used. Steps 1–15 of Algorithm 5 are equivalent to Steps 1–16 of Algorithm 4. Both values β_1 and β_2 are computed in the same way in both algorithms while values β_3, β_4, and β_5 are worked out in Algorithm 5 for the polynomial orders $m_3 = 8$, $m_4 = 14$, and $m_5 = 21$, respectively. Steps 16–31 of Algorithm 5 correspond to Steps 17–29 of Algorithm 4.

Algorithm 5: Given a matrix $A \in \mathbb{C}^{n\times n}$, the values Θ from Table 2, $\mathbb{M} = \{2, 4, 8, 14, 21\}$ and a tolerance *tol*, this algorithm computes the order of Taylor approximation $m_k \in \mathbb{M}$ and the scaling factor s, together with $2^{-s}A$ and the necessary powers of $4^{-s}B$ for computing $2^{-s}AP_{m_k}(4^{-s}B)$ from (10), (11) or (12).

```
1  $B_1 = -A^2$; $B_2 = B_1^2$
2  Compute $\beta_1 \approx \left\|B^3\right\|^{1/3}$ from $B_1$ and $B_2$
3  $f = 0$; $k = 1$
4  while $f = 0$ and $k < 5$ do
5      $k = k + 1$
6      if $k < 5$ then
7          Compute $\beta_k \approx \|B^{m_{k+1}}\|^{1/(m_{k+1})}$ from $B_1$ and $B_2$
8      else
9          $B_3 = B_1 B_2$
10         Compute $\beta_5 \approx \left\|B^{22}\right\|^{1/22}$ from $B_1$ and $B_3$
11     end
12     if $|\beta_k - \beta_{k-1}| < tol$ and $\beta_k < \Theta_k$ then
13         $f = 1$; $s = 0$
14     end
15 end
16 if $f = 0$ then
17     $s = \max\left(0, \left\lceil \frac{1}{2}\log_2(\beta_k/\Theta_k)\right\rceil\right)$
18 end
19 if $s > 0$ then
20     $s_0 = \max\left(0, \left\lceil \frac{1}{2}\log_2(\beta_{k-1}/\Theta_{k-1})\right\rceil\right)$
21     if $s_0 = s$ then
22         $k = k - 1$
23     end
24     $A = 2^{-s}A$
25     $B_1 = 4^{-s}B_1$
26     $B_2 = 16^{-s}B_2$
27     if $k = 5$ then
28         $B_3 = 64^{-s}B_3$
29     end
30 end
31 $m = m_k$
```

Table 1. Values of Θ_{m_k}, $1 \le k \le 9$, for polynomial evaluation by means of the Paterson–Stockmeyer method.

$m_1 = 2$	$1.1551925093100 \times 10^{-3}$
$m_2 = 4$	$2.8530558816082 \times 10^{-2}$
$m_3 = 6$	$9.7931623314428 \times 10^{-2}$
$m_4 = 9$	$2.3519926145338 \times 10^{-1}$
$m_5 = 12$	$3.7089935615781 \times 10^{-1}$
$m_6 = 16$	$5.2612365603423 \times 10^{-1}$
$m_7 = 20$	$6.5111831924355 \times 10^{-1}$
$m_8 = 25$	$7.73638541973549 \times 10^{-1}$
$m_9 = 30$	$8.68708923627294 \times 10^{-1}$

Table 2. Values of Θ_{m_k}, $1 \le k \le 5$, for polynomial evaluation using the Sastre formulas.

$m_1 = 2$	$1.1551925093100 \times 10^{-3}$
$m_2 = 4$	$2.8530558816082 \times 10^{-2}$
$m_3 = 8$	$1.88126704493647 \times 10^{-1}$
$m_4 = 14+$	$4.65700446893510 \times 10^{-1}$
$m_5 = 21+$	$6.84669656651721 \times 10^{-1}$

3. Numerical Experiments

The following codes have been implemented in MATLAB to test the accuracy and the efficiency of the different algorithms proposed:

- `tanh_expm`: this code corresponds to the implementation of Algorithm 1. For obtaining $m \in \{2, 4, 6, 9, 12, 16, 20, 25, 30\}$ and s and computing e^{2A}, it uses function `exptaynsv3` (see [26]).
- `tanh_tayps`: this development, based on Algorithm 2, incorporates Algorithm 4 for computing m and s, where m takes values in the same set than the `tanh_expm` code. The Paterson–Stockmeyer method is considered to evaluate the Taylor matrix polynomials.
- `tanh_pol`: this function, corresponding to Algorithm 3, employs Algorithm 5 in the m and s calculation, where $m \in \{2, 4, 8, 14, 21\}$. The Taylor matrix polynomials are evaluated by means of Sastre formulas.

Three types of matrices with distinct features were used to build a battery of tests that enabled us to compare the numerical performance of these codes. The MATLAB Symbolic Math Toolbox with 256 digits of precision was used to compute the *"exact"* matrix hyperbolic tangent function using the `vpa` (variable-precision floating-point arithmetic) function. The test battery featured the following three sets:

(a) Diagonalizable complex matrices: one hundred diagonalizable 128×128 complex matrices obtained as the result of $A = V \cdot D \cdot V^{-1}$, where D is a diagonal matrix (with real and complex eigenvalues) and matrix V is an orthogonal matrix, $V = H/\sqrt{n}$, being H a Hadamard matrix and n is the matrix order. As 1-norm, we have that $2.56 \le \|A\|_1 \le 256$. The matrix hyperbolic tangent was calculated *"exactly"* as $\tanh(A) = V \cdot \tanh(D) \cdot V^T$ using the `vpa` function.

(b) Non-diagonalizable complex matrices: one hundred non-diagonalizable 128×128 complex matrices computed as $A = V \cdot J \cdot V^{-1}$, where J is a Jordan matrix with complex eigenvalues whose modules are less than 5 and the algebraic multiplicity is randomly generated between 1 and 4. V is an orthogonal random matrix with elements in the interval $[-0.5, 0.5]$. As 1-norm, we obtained that $45.13 \le \|A\|_1 \le 51.18$. The *"exact"* matrix hyperbolic tangent was computed as $\tanh(A) = V \cdot \tanh(J) \cdot V^{-1}$ by means of the `vpa` function.

(c) Matrices from the Matrix Computation Toolbox (MCT) [31] and from the Eigtool MATLAB Package (EMP) [32]: fifty-three matrices with a dimension lower than or equal to 128 were chosen because of their highly different and significant characteristics from each other. We decided to scale these matrices so that they had 1-norm not exceeding 512. As a result, we obtained that $1 \leq \|A\|_1 \leq 489.3$. The *"exact"* matrix hyperbolic tangent was calculated by using the two following methods together and the `vpa` function:

- Find a matrix V and a diagonal matrix D so that $A = VDV^{-1}$ by using the MATLAB function `eig`. In this case, $T_1 = V \tanh(D) V^{-1}$.
- Compute the Taylor approximation of the hyperbolic tangent function (T_2), with different polynomial orders (m) and scaling parameters (s). This procedure is finished when the obtained result is the same for the distinct values of m and s in IEEE double precision.

The *"exact"* matrix hyperbolic tangent is considered only if

$$\frac{\|T_1 - T_2\|}{\|T_1\|} < u.$$

Although MCT and EMP are really comprised of 72 matrices, only 53 matrices of them, 42 from MCT and 11 from EMP, were considered for our purposes. On the one hand, matrix 6 from MCT and matrix 10 from EMP were rejected because the relative error made by some of the codes to be evaluated was greater or equal to unity. This was due to the ill-conditioning of these matrices for the hyperbolic tangent function. On the other hand, matrices 4, 12, 17, 18, 23, 35, 40, 46, and 51 from MCT and matrices 7, 9, 16, and 17 from EMP were not generated because they did not satisfy the described criterion to obtain the *"exact"* matrix hyperbolic tangent. Finally, matrices 8, 13, 15, and 18 from EMP were refused as they are also part of MCT.

For each of the three previously mentioned sets of matrices, one test was respectively and independently carried out, which indeed corresponds to an experiment to analyse the numerical properties and to account for the computational cost of the different implemented codes. All these experiments were run on an HP Pavilion dv8 Notebook PC with an Intel Core i7 CPU Q720 @1.60 Ghz processor and 6 GB of RAM, using MATLAB R2020b. First, Table 3 shows the percentage of cases in which the normwise relative errors of `tanh_expm` are lower than, greater than, or equal to those of `tanh_tayps` and `tanh_pol`. These normwise relative errors were obtained as

$$\mathrm{Er} = \frac{\|\tanh(A) - \tilde{\tanh}(A)\|_1}{\|\tanh(A)\|_1},$$

where $\tanh(A)$ represents the exact solution and $\tilde{\tanh}(A)$ stands for the approximate one.

Table 3. The relative error comparison, for the three tests, between `tanh_expm` vs. `tanh_tayps` and `tanh_expm` vs `tanh_pol`.

	Test 1	**Test 2**	**Test 3**
Er(`tanh_expm`)<Er(`tanh_tayps`)	32%	0%	22.64%
Er(`tanh_expm`)>Er(`tanh_tayps`)	68%	100%	77.36%
Er(`tanh_expm`)=Er(`tanh_tayps`)	0%	0%	0%
Er(`tanh_expm`)<Er(`tanh_pol`)	44%	0%	30.19%
Er(`tanh_expm`)>Er(`tanh_pol`)	56%	100%	69.81%
Er(`tanh_expm`)=Er(`tanh_pol`)	0%	0%	0%

As we can appreciate, from the point of view of the accuracy of the results, `tanh_tayps` outperformed `tanh_expm` in 68% of the matrices for Test 1 and 100% and 77.36% of them for Tests 2 and 3. On the other hand, `tanh_pol` obtained slightly more modest results with improvement percentages equal to 56%, 100%, and 69.81% for Tests 1, 2, and 3, respectively.

Table 4 reports the computational complexity of the algorithms. This complexity was expressed as the number of matrix products required to calculate the hyperbolic tangent of the different matrices that make up each of the test cases. This number of products includes the number of matrix multiplications and the cost of the systems of linear equations that were solved in the recovering phase, by all the codes, together with one more in Step 2 of Algorithm 1 by `tanh_expm`.

The cost of each system of linear equations with n right-hand side vectors, where n denotes the size of the square coefficient matrices, was taken as $4/3$ matrix products. The cost of other arithmetic operations, such as the sum of matrices or the product of a matrix by a vector, was not taken into consideration. As can be seen, the lowest computational cost corresponded to `tanh_expm`, followed by `tanh_pol` and `tanh_tayps`. For example, `tanh_expm` demanded 1810 matrix products to compute the matrices belonging to Test 1, compared to 1847 by `tanh_pol` and 2180 by `tanh_tayps`.

Table 4. Matrix products (P) corresponding to the `tanh_expm`, `tanh_tayps`, and `tanh_pol` functions for Tests 1–3.

	Test 1	**Test 2**	**Test 3**
`P(tanh_expm)`	1810	1500	848
`P(tanh_tayps)`	2180	1800	1030
`P(tanh_pol)`	1847	1500	855

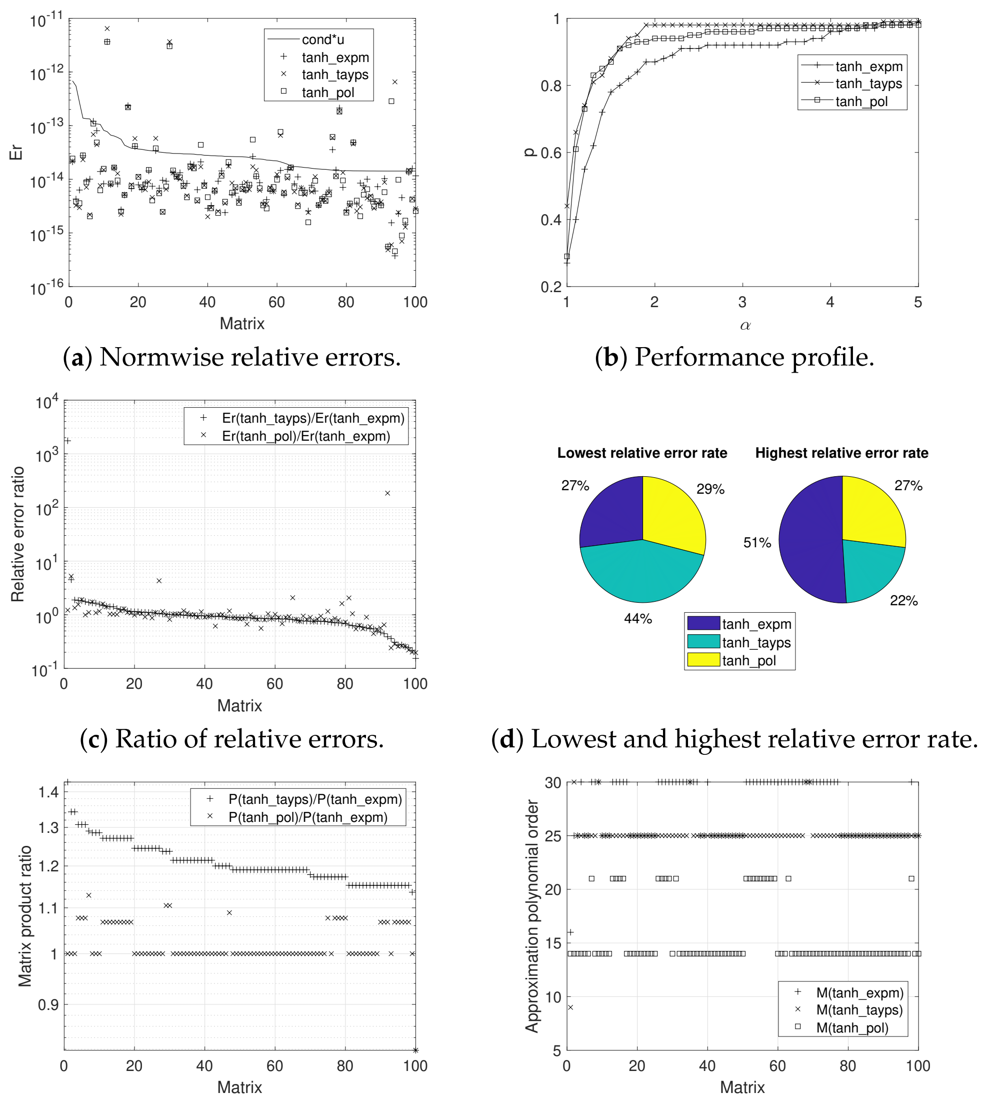
Respectively, for the three experiments, Figures 1–3 illustrate the normwise relative errors (a), the performance profiles (b), the ratio of the relative errors (c), the lowest and highest relative error rate (d), the ratio of the matrix products (e), and the polynomial orders (f) for our three codes to be evaluated.

Figures 1a, 2a and 3a correspond to the normwise relative error. As they reveal, the three methods under evaluation were numerically stable for all the matrices that were computed, and all of them provided very accurate results, where the relative errors incurred were always less than 10^{-11}. The solid line that appears in these figures traces the function $k_{\tanh}u$, where $k_{\tanh}$ (or *cond*) stands for the condition number of the matrix hyperbolic tangent function [4] (Chapter 3), and u represents the unit roundoff.

It is clear that the errors incurred by all the codes were usually quite close to this line for the three experiments and even below it, as was largely the case for Tests 1 and 3. For the sake of readability in the graphs, normwise relative errors lower than 10^{-20} were plotted with that value in the figures. Notwithstanding, their original quantities were maintained for the rest of the results.

Figures 1b, 2b and 3b depict the performance profile of the codes. In them, the α coordinate on the x-axis ranges from 1 to 5 in steps equal to 0.1. For a concrete α value, the p coordinate on the y-axis indicates the probability that the considered algorithm has a relative error lower than or equal to α-times the smallest relative error over all the methods on the given test.

The implementation of `tanh_tayps` always achieved the results with the highest accuracy, followed closely by `tanh_pol`. For Tests 1 and 2, Figures 1b and 2b indicate that the results provided by them are very similar, although the difference in favour of `tanh_tayps` is more remarkable in Test 3. As expected from the percentages given in Table 3, `tanh_expm` computed the hyperbolic tangent function with the worst accuracy, most notably for the matrices from Tests 2 and 3.



(**a**) Normwise relative errors. (**b**) Performance profile.

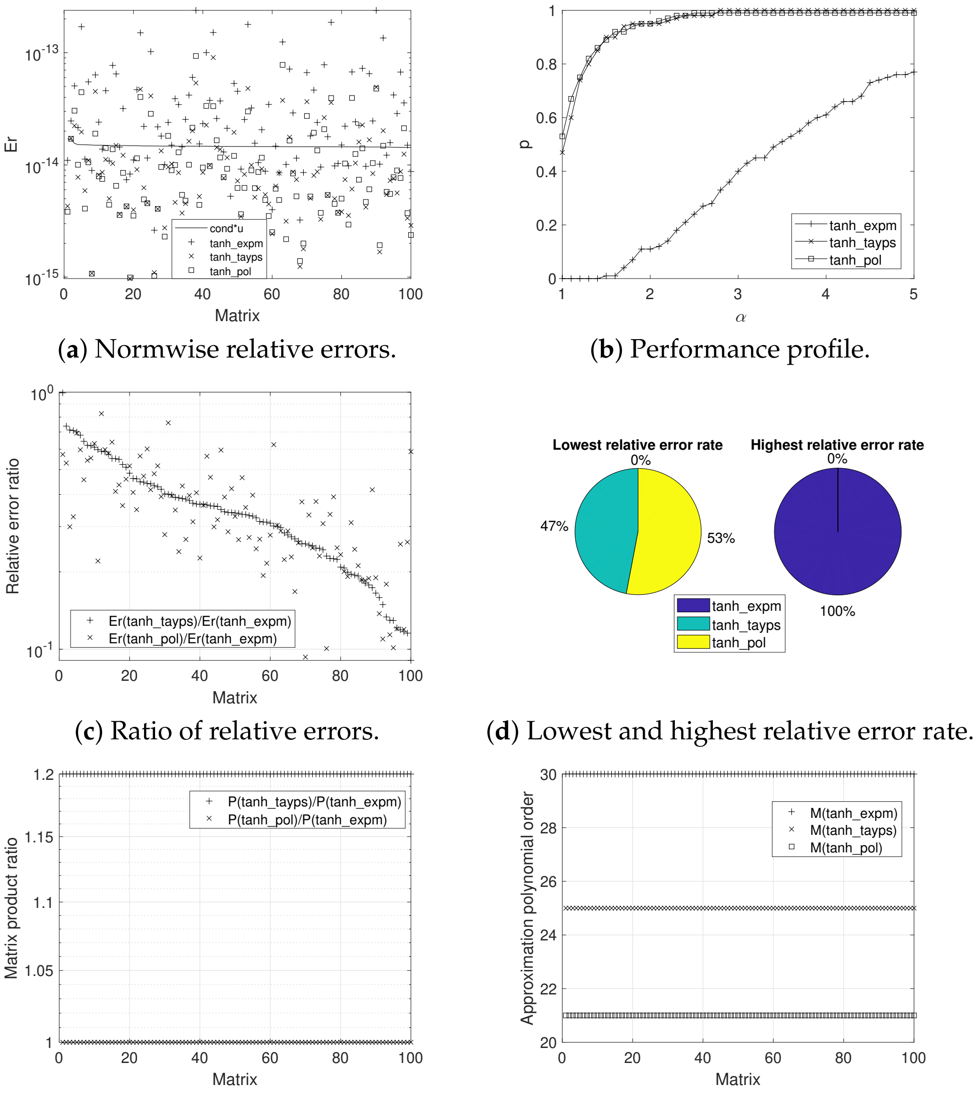
(**c**) Ratio of relative errors. (**d**) Lowest and highest relative error rate.

(**e**) Ratio of matrix products. (**f**) Polynomial orders.

Figure 1. Experimental results for Test 1.

Precisely, the relationship among the normwise relative errors incurred by the codes to be examined is displayed in Figures 1c, 2c and 3c. All these ratios are presented in decreasing order with respect to Er(`tanh_tayps`)/Er(`tanh_expm`). This factor is less than 1 for the great majority of the matrices, which indicates that `tanh_tayps` and `tanh_pol` are more accurate codes than `tanh_expm` for estimating the hyperbolic tangent function.

These data are further corroborated by the results shown in Figures 1d, 2d and 3d. These graphs report the percentages of the matrices, for each of the tests, in which each code resulted in the lowest or highest normwise relative error among the errors provided by all of them. Thus, `tanh_tayps` exhibited the smallest relative error in 44% of the matrices for Test 1 and in 47% of them for Test 3, followed by `tanh_pol` in 29% and 36% of the cases, respectively. For all the other cases, `tanh_expm` was the most reliable method. For Test 2, `tanh_pol` was the most accurate code in 53% of the matrices, and `tanh_tayps` was the most accurate in 47% of them. In line with these results, `tanh_expm` was found to be the approach that led to the largest relative errors in 51% of the cases in Test 1, in 100% of them in Test 2, and in 64% for Test 3.



(**a**) Normwise relative errors. (**b**) Performance profile.

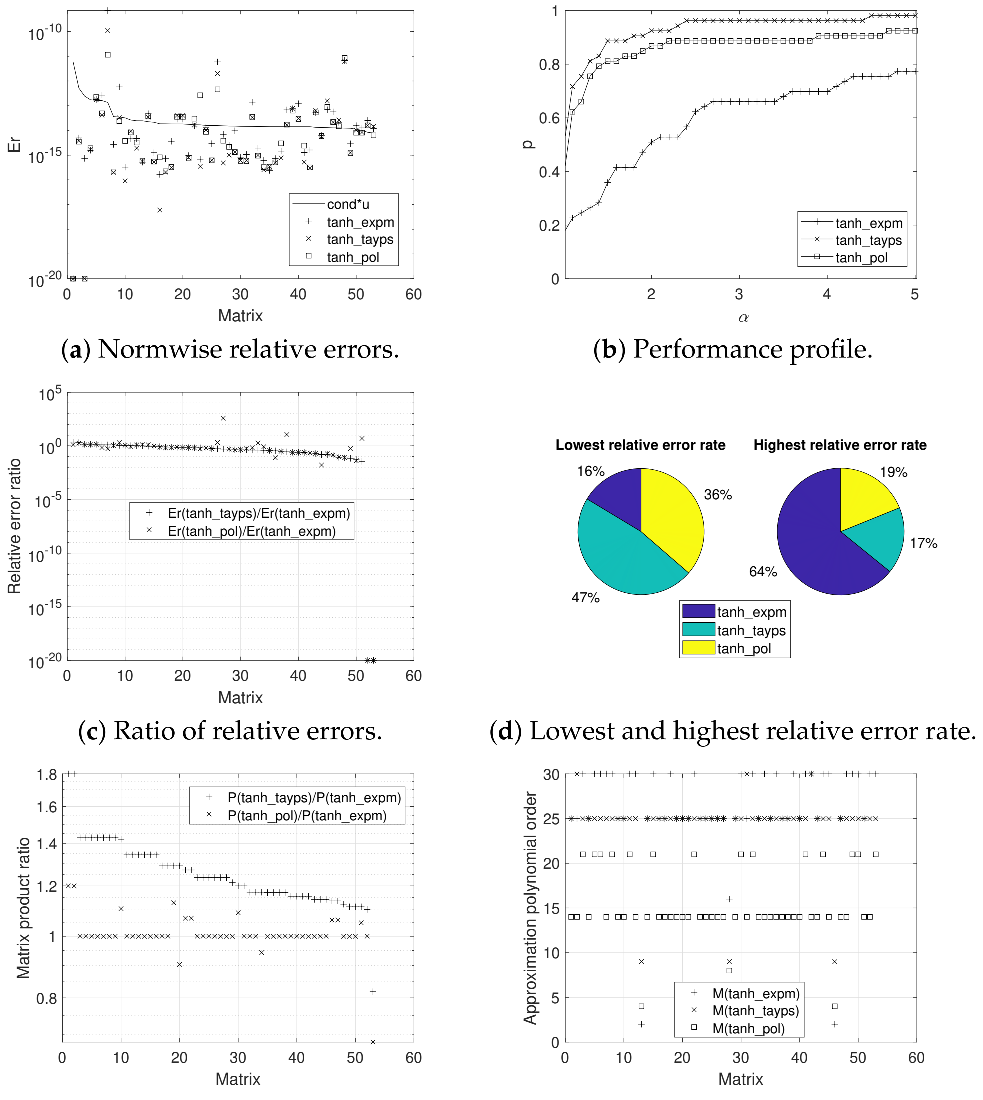
(**c**) Ratio of relative errors. (**d**) Lowest and highest relative error rate.

(**e**) Ratio of matrix products. (**f**) Polynomial orders.

Figure 2. Experimental results for Test 2.

In contrast, although `tanh_expm` proved to be the most inaccurate code, it is also evident that its computational cost was usually the lowest one, as Table 4 and Figures 1e, 2e and 3e reported. The ratio between the number of `tanh_tayps` and `tanh_expm` matrix products ranged from 0.82 to 1.43 for Test 1, was equal to 1.20 for Test 2, and ranged from 0.82 to 1.8 for Test 3. Regarding `tanh_pol` and `tanh_expm`, this quotient varied from 0.82 to 1.13 for Test 1, from 0.68 to 1.2 for Test 3, and was equal to 1 for Test 2.

Table 5 lists, in order for Tests 1, 2, and 3, the minimum, maximum, and average values of the degree of the Taylor polynomials m and the scaling parameter s employed by the three codes. Additionally, and in a more detailed way, Figures 1f, 2f and 3f illustrate the order of the polynomial used in the calculation of each of the matrices that compose the testbed. The value of m allowed to be chosen was between 2 and 30 for `tanh_expm` and `tanh_tayps` and between 2 and 21 for `tanh_pol`. As we can see, the average value of m that was typically used varied from 25 and 30 for `tanh_expm`. It was around 25 for `tanh_tayps`, and it ranged from 14 to 21 for `tanh_pol`.



(**a**) Normwise relative errors. (**b**) Performance profile.

(**c**) Ratio of relative errors. (**d**) Lowest and highest relative error rate.

(**e**) Ratio of matrix products. (**f**) Polynomial orders.

Figure 3. Experimental results for Test 3.

Table 5. The minimum, maximum, and average parameters m and s employed for Tests 1–3, respectively.

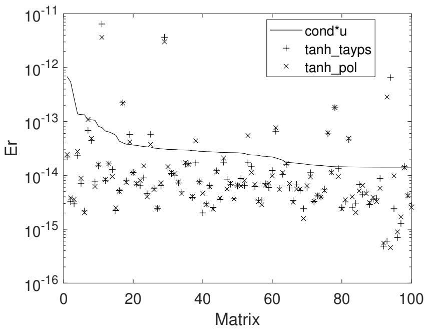
	m			s		
	Min.	**Max.**	**Average**	**Min.**	**Max.**	**Average**
`tanh_expm`	16	30	27.41	0	5	3.55
`tanh_tayps`	9	30	25.09	0	6	4.65
`tanh_pol`	14	21	15.47	0	6	4.83
`tanh_expm`	30	30	30.00	2	2	2.00
`tanh_tayps`	25	25	25.00	3	3	3.00
`tanh_pol`	21	21	21.00	3	3	3.00
`tanh_expm`	2	30	26.23	0	8	2.77
`tanh_tayps`	9	30	24.38	0	9	3.74
`tanh_pol`	4	21	15.36	0	9	3.87

Having concluded the first part of the experimental results, we continue by comparing `tanh_tayps` and `tanh_pol`, the Taylor series-based codes that returned the best accuracy in the results. Table 6 presents the percentage of cases in which `tanh_tayps` gave place to relative errors that were lower than, greater than, or equal to those of `tanh_pol`. According to the exposed values, both methods provided very similar results, and the percentage of cases in which each method was more accurate than the other was approximately equal to 50% for the different tests.

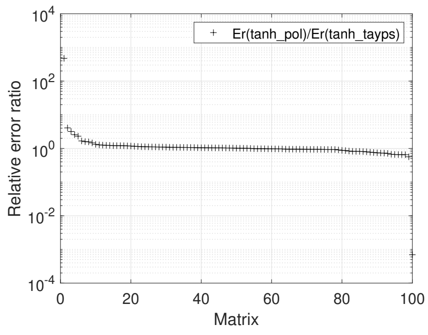
Table 6. The relative error comparison for the three tests between `tanh_tayps` vs. `tanh_pol`.

	Test 1	**Test 2**	**Test 3**
Er(`tanh_tayps`)<Er(`tanh_pol`)	56%	47%	50.94%
Er(`tanh_tayps`)>Er(`tanh_pol`)	44%	53%	45.28%
Er(`tanh_tayps`)=Er(`tanh_pol`)	0%	0%	3.77%

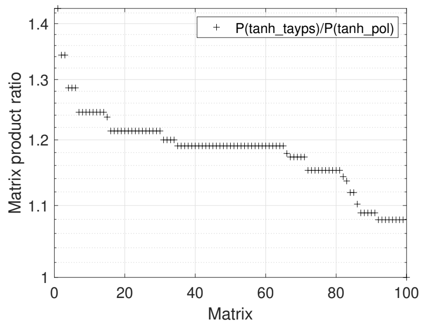
Figures 4–6 incorporate the normwise relative errors (a), the ratio of relative errors (b), and the ratio of matrix products (c) between `tanh_tayps` and `tanh_pol`. The graphs corresponding to the performance profiles and the polynomial orders are not included now since the results match with those already shown in the previous figures. All this information is also complemented by Table 7, which collects, respectively for each test, the minimum, maximum, and average relative errors incurred by both methods to be analysed, together with the standard deviation.



(**a**) Normwise relative errors.



(**b**) Ratio of relative errors.

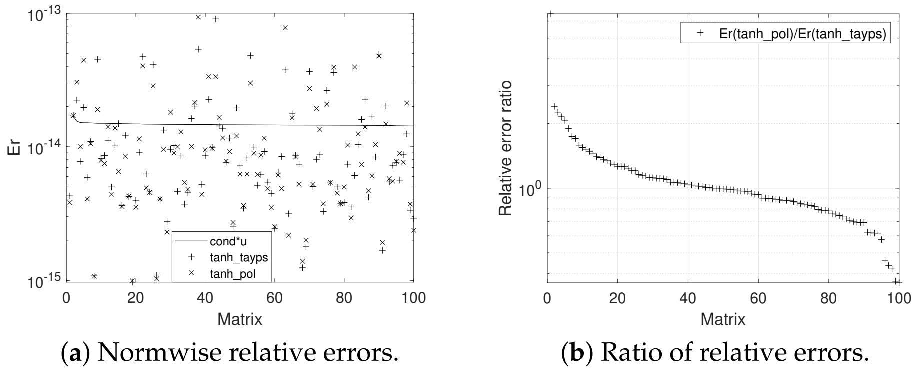


(**c**) Ratio of matrix products.

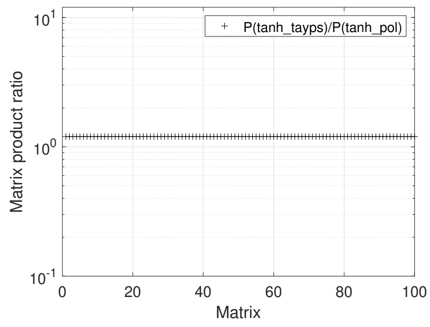
Figure 4. Experimental results for Test 1.

As Table 6 details, for Tests 1 and 3, `tanh_tayps` slightly improved on `tanh_pol` in the percentage of matrices in which the relative error committed was lower, although it is true that the difference between the results provided by the two methods was small in most cases. However, when such a difference occurred, it was more often in favour of `tanh_tayps` than the other way around, in quantitative terms.

With all this, we can also appreciate that the mean relative error and the standard deviation incurred by `tanh_pol` were lower than those of `tanh_tayps`. For matrices that were part of Test 2, the numerical results achieved by both methods were almost identical, and the differences between them were not significant.

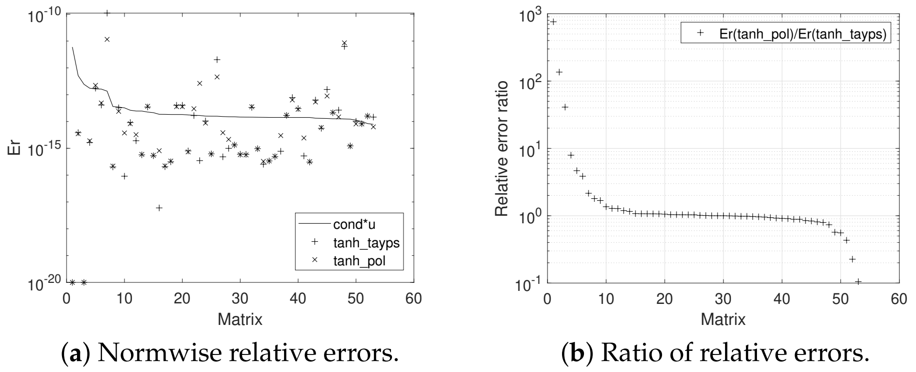


(**a**) Normwise relative errors. (**b**) Ratio of relative errors.

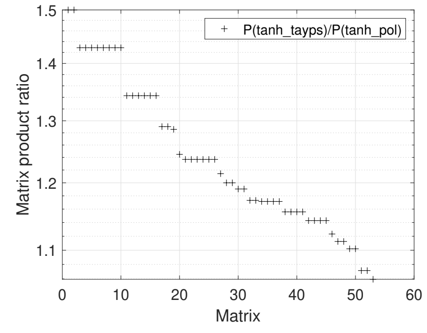


(**c**) Ratio of matrix products.

Figure 5. Experimental results for Test 2.



(**a**) Normwise relative errors. (**b**) Ratio of relative errors.



(**c**) Ratio of matrix products.

Figure 6. Experimental results for Test 3.

To conclude the analysis and with regard to the computational cost of both codes, such as depicted in Figures 4c, 5c and 6c, it simply remains to note that `tanh_tayps` performed between 1 and 1.43 more matrix products compared with `tanh_pol` for Test 1, 1.2 more for Test 2, and between 1.06 and 1.5 more for Test 3. Therefore, `tanh_pol` computed the matrix tangent function with a very similar accuracy in the results compared to `tanh_tayps` but with a considerably lower computational cost.

Table 7. The minimum, maximum, and average values and the standard deviation of the relative errors committed by `tanh_tayps` and `tanh_pol` for Tests 1–3.

	Minimum	Maximum	Average	Standard Deviation
`tanh_tayps`	4.84×10^{-16}	6.45×10^{-12}	1.22×10^{-13}	7.40×10^{-13}
`tanh_pol`	4.55×10^{-16}	3.64×10^{-12}	8.48×10^{-14}	4.69×10^{-13}
`tanh_tayps`	9.71×10^{-16}	9.06×10^{-14}	1.25×10^{-14}	1.41×10^{-14}
`tanh_pol`	9.92×10^{-16}	9.35×10^{-14}	1.26×10^{-14}	1.47×10^{-14}
`tanh_tayps`	1.09×10^{-254}	1.10×10^{-10}	2.26×10^{-12}	1.52×10^{-11}
`tanh_pol`	1.09×10^{-254}	1.16×10^{-11}	4.10×10^{-13}	1.96×10^{-12}

4. Conclusions

Two alternative methods to approximate the matrix hyperbolic tangent were addressed in this work. The first was derived from its own definition and reduced to the computation of a matrix exponential. The second method deals with its Taylor series expansion. In this latter approach, two alternatives were developed and differed on how the evaluation of the matrix polynomials was performed. In addition, we provided algorithms to determine the scaling factor and the order of the polynomial. As a result, we dealt with a total of three MATLAB codes (`tanh_expm`, `tanh_tayps`, and `tanh_pol`), which were evaluated on a complete testbed populated with matrices of three different types.

The Taylor series-based codes reached the most accurate results in the tests, a fact that is in line with the recommendation suggested in [33] of using a Taylor development against other alternatives whenever possible. However, codes based on Taylor series can be computationally expensive if the Paterson–Stockmeyer method is employed to evaluate the polynomial, as we confirmed with the `tanh_tayps` implementation. One idea to reduce this problem is to use Sastre formulas, as we did in our `tanh_pol` code, resulting in an efficient alternative that significantly reduced the number of matrix operations without affecting the accuracy.

The results found in this paper demonstrated that the three codes were stable and provided acceptable accuracy. However, and without underestimating the other two codes, the `tanh_pol` implementation proposed here offered the best ratio of accuracy/computational cost and proved to be an excellent method for the computation of the matrix hyperbolic tangent.

Author Contributions: Conceptualization, E.D.; methodology, E.D., J.I., J.M.A. and J.S.; software, J.I., J.M.A. and J.S.; validation, J.M.A.; formal analysis, J.I. and J.S.; investigation, E.D., J.I., J.M.A. and J.S.; writing—original draft preparation, E.D., J.I., J.M.A. and J.S.; writing—review and editing, P.A.-J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Spanish Ministerio de Ciencia e Innovación under grant number TIN2017-89314-P.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Constantine, A.; Muirhead, R. Partial differential equations for hypergeometric functions of two argument matrices. *J. Multivar. Anal.* **1972**, *2*, 332–338. [CrossRef]
2. James, A.T. Special functions of matrix and single argument in statistics. In *Theory and Application of Special Functions*; Academic Press: Cambridge, MA, USA, 1975; pp. 497–520.
3. Hochbruck, M.; Ostermann, A. Exponential integrators. *Acta Numer.* **2010**, *19*, 209–286. [CrossRef]
4. Higham, N.J. *Functions of Matrices: Theory and Computation*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2008; pp. 425.
5. Rinehart, R. The Equivalence of Definitions of a Matrix Function. *Am. Math. Mon.* **1955**, *62*, 395–414. [CrossRef]
6. Estrada, E.; Higham, D.J.; Hatano, N. Communicability and multipartite structures in complex networks at negative absolute temperatures. *Phys. Rev. E* **2008**, *78*, 026102. [CrossRef] [PubMed]
7. Jódar, L.; Navarro, E.; Posso, A.; Casabán, M. Constructive solution of strongly coupled continuous hyperbolic mixed problems. *Appl. Numer. Math.* **2003**, *47*, 477–492. [CrossRef]
8. Defez, E.; Sastre, J.; Ibáñez, J.; Peinado, J.; Tung, M.M. A method to approximate the hyperbolic sine of a matrix. *Int. J. Complex Syst. Sci.* **2014**, *4*, 41–45.
9. Defez, E.; Sastre, J.; Ibáñez, J.; Peinado, J. Solving engineering models using hyperbolic matrix functions. *Appl. Math. Model.* **2016**, *40*, 2837–2844. [CrossRef]
10. Defez, E.; Sastre, J.; Ibáñez, J.; Ruiz, P. Computing hyperbolic matrix functions using orthogonal matrix polynomials. In *Progress in Industrial Mathematics at ECMI 2012*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 403–407.
11. Defez, E.; Sastre, J.; Ibánez, J.; Peinado, J.; Tung, M.M. On the computation of the hyperbolic sine and cosine matrix functions. *Model. Eng. Hum. Behav.* **2013**, 1, 46-59.
12. Efimov, G.V.; Von Waldenfels, W.; Wehrse, R. Analytical solution of the non-discretized radiative transfer equation for a slab of finite optical depth. *J. Quant. Spectrosc. Radiat. Transf.* **1995**, *53*, 59–74. [CrossRef]
13. Lehtinen, A. Analytical Treatment of Heat Sinks Cooled by Forced Convection. Ph.D. Thesis, Tampere University of Technology, Tampere, Finland, 2005.
14. Lampio, K. Optimization of Fin Arrays Cooled by Forced or Natural Convection. Ph.D. Thesis, Tampere University of Technology, Tampere, Finland, 2018.
15. Hilscher, R.; Zemánek, P. Trigonometric and hyperbolic systems on time scales. *Dyn. Syst. Appl.* **2009**, *18*, 483.
16. Zemánek, P. New Results in Theory of Symplectic Systems on Time Scales. Ph.D. Thesis, Masarykova Univerzita, Brno, Czech Republic, 2011.
17. Estrada, E.; Silver, G. Accounting for the role of long walks on networks via a new matrix function. *J. Math. Anal. Appl.* **2017**, *449*, 1581–1600. [CrossRef]
18. Cieśliński, J.L. Locally exact modifications of numerical schemes. *Comput. Math. Appl.* **2013**, *65*, 1920–1938. [CrossRef]
19. Cieśliński, J.L.; Kobus, A. Locally Exact Integrators for the Duffing Equation. *Mathematics* **2020**, *8*, 231. [CrossRef]
20. Golub, G.H.; Loan, C.V. *Matrix Computations*, 3rd ed.; Johns Hopkins Studies in Mathematical Sciences; The Johns Hopkins University Press: Baltimore, MD, USA, 1996.
21. Moler, C.; Van Loan, C. Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. *SIAM Rev.* **2003**, *45*, 3–49. [CrossRef]
22. Sastre, J.; Ibáñez, J.; Defez, E. Boosting the computation of the matrix exponential. *Appl. Math. Comput.* **2019**, *340*, 206–220. [CrossRef]
23. Sastre, J.; Ibáñez, J.; Defez, E.; Ruiz, P. Efficient orthogonal matrix polynomial based method for computing matrix exponential. *Appl. Math. Comput.* **2011**, *217*, 6451–6463. [CrossRef]
24. Sastre, J.; Ibáñez, J.; Defez, E.; Ruiz, P. New scaling-squaring Taylor algorithms for computing the matrix exponential. *SIAM J. Sci. Comput.* **2015**, *37*, A439–A455. [CrossRef]
25. Defez, E.; Ibánez, J.; Alonso-Jordá, P.; Alonso, J.; Peinado, J. On Bernoulli matrix polynomials and matrix exponential approximation. *J. Comput. Appl. Math.* **2020**, 113207. [CrossRef]
26. Ruiz, P.; Sastre, J.; Ibáñez, J.; Defez, E. High perfomance computing of the matrix exponential. *J. Comput. Appl. Math.* **2016**, *291*, 370–379. [CrossRef]
27. Paterson, M.S.; Stockmeyer, L.J. On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials. *SIAM J. Comput.* **1973**, *2*, 60–66. [CrossRef]
28. Sastre, J. Efficient evaluation of matrix polynomials. *Linear Algebra Appl.* **2018**, *539*, 229–250. [CrossRef]
29. Al-Mohy, A.H.; Higham, N.J. A New Scaling and Squaring Algorithm for the Matrix Exponential. *SIAM J. Matrix Anal. Appl.* **2009**, *31*, 970–989. [CrossRef]
30. Higham, N.J. FORTRAN Codes for Estimating the One-norm of a Real or Complex Matrix, with Applications to Condition Estimation. *ACM Trans. Math. Softw.* **1988**, *14*, 381–396. [CrossRef]
31. Higham, N.J. The Matrix Computation Toolbox. 2002. Available online: http://www.ma.man.ac.uk/~higham/mctoolbox (accessed on 7 March 2020).

32. Wright, T.G. Eigtool, Version 2.1. 2009. Available online: http://www.comlab.ox.ac.uk/pseudospectra/eigtool (accessed on 7 March 2020).
33. Corwell, J.; Blair, W.D. Industry Tip: Quick and Easy Matrix Exponentials. *IEEE Aerosp. Electron. Syst. Mag.* **2020**, *35*, 49–52. [CrossRef]