AppliedMath

MDPI

Article

An Efficient Algorithm for Basic Elementary Matrix Functions with Specified Accuracy and Application

Huizeng Qin [1] and Youmin Lu [2,*]

[1] The School of Mathematics and Statistics, Shandong University of Technology, Zibo 255049, China; qinhz@sdut.edu.cn
[2] Department of Mathematical and Digital Sciences, Bloomsburg University, Bloomsburg, PA 17815, USA
* Correspondence: ylu@bloomu.edu

Abstract: If the matrix function $f(At)$ posses the properties of $f(At) = g(f((\frac{t}{k}A))$, then the recurrence formula $f_{i-1} = g(f_i), i = N, N-1, \cdots, 1, f(tA) = f_0$, can be established. Here, $f_N = f(A_N) = \sum_{j=0}^{m} a_j A_N^j, A_N = \frac{t}{k^N} A$. This provides an algorithm for computing the matrix function $f(At)$. By specifying the calculation accuracy p, a method is presented to determine m and N in a way that minimizes the time of the above algorithm, thus providing a fast algorithm for $f(At)$. It is important to note that m only depends on the calculation accuracy p and is independent of the matrix A and t. Therefore, $f(A_N)$ has a fixed calculation format that is easily computed. On the other hand, N depends not only on A, but also on t. This provides a means to select t such that N is equal to 0, a property of significance. In summary, the algorithm proposed in this article enables users to establish a desired level of accuracy and then utilize it to select the appropriate values for m and N to minimize computation time. This approach ensures that both accuracy and efficiency are addressed concurrently. We develop a general algorithm, then apply it to the exponential, trigonometric, and logarithmic matrix functions, and compare the performance with that of the internal system functions of Mathematica and Pade approximation. In the last section, an example is provided to illustrate the rapid computation of numerical solutions for linear differential equations.

Keywords: matrix function; efficient algorithms; high accuracy; linear differential equations

MSC: 68W25; 68U99

check for updates

Citation: Qin, H.; Lu, Y. An Efficient Algorithm for Basic Elementary Matrix Functions with Specified Accuracy and Application. *AppliedMath* **2024**, *4*, 690–708. https://doi.org/10.3390/appliedmath4020037

Received: 26 January 2024
Revised: 4 March 2024
Accepted: 19 April 2024
Published: 5 June 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Matrix exponent functions plays a crucial role in various fields, including ordinary differential equations and control theory. In particular, they are instrumental in analyzing general structural dynamic systems, represented by the initial value problem:

$$Mx'' + Gx' + Kx = r(t), x(0) = x_0, x'(0) = x_1, \tag{1}$$

where M, G, and K are n by n square matrices, and $r(t)$ denotes the external force applied. The introduction of dual variables of a Hamiltonian system,

$$p = Mx' + Gx/2, \tag{2}$$

results in the following system:

$$U' = HU + F(t), U(0) = U_0, \tag{3}$$

where

$$\begin{gathered} U=\begin{pmatrix} q \\ p \end{pmatrix}, H=\begin{pmatrix} -M^{-1}G/2 & M^{-1} \\ GM^{-1}G/4-K & -M^{-1}G/2 \end{pmatrix}, \\ F(t)=\begin{pmatrix} 0 \\ r(t) \end{pmatrix}, q=x, U_0=\begin{pmatrix} x_0 \\ Mx_1+Gx_0/2 \end{pmatrix}. \end{gathered} \tag{4}$$

The solution of (3) is provided as follows:

$$U(t)=e^{Ht}U_0+\int_0^t e^{H(t-s)}F(s)ds. \tag{5}$$

When a problem is oscillatory, the formal solution often involves both the sine and cosine of a matrix. Famous examples include the Schrődinger equation in quantum mechanics:

$$i\frac{d\psi}{dt}=H(t)\psi, \quad \psi(t_0)=\psi_0, \tag{6}$$

where $H(t)$ is a Hermitian operator and ψ is a complex wave function. If $H(t)=A$ is a real and constant matrix, the solution of (6) is as follows:

$$\psi(t)=e^{-iAt}\psi_0=(\cos(tA)-i\sin(tA))\psi_0. \tag{7}$$

There are other examples where the computation of the sine and cosine of a matrix can be of interest. Wave equations provided by the generic second order system are written as follows:

$$x''+Kx=F(t), x(0)=x_0, x'(0)=x_1, \tag{8}$$

where the solution is provided by the variation of the constant method:

$$x(t)=\cos(tA)x_0+\sin_1(tA)x_1+\int_0^t \sin_1((t-s)A)F(s)ds, \tag{9}$$

where $A=\sqrt{K}, \sin_1(tA)=\int_0^t \cos(sA)ds$. If A is invertible, then (9) can be written as follows:

$$x(t)=\cos(tA)x_0+\sin(tA)A^{-1}x_1+A^{-1}\int_0^t \sin((t-s)A)F(s)ds. \tag{10}$$

For (5), providing $U(k\tau)(k=1,2,\cdots)$ as accurately as possible is a concern for engineers and mathematicians, where τ is called the step. The specific method for this problem is as follows:

$$U_k=TU_{k-1}+\int_0^\tau e^{Hs}F(k\tau-s)ds, k=1,2,\cdots, T=e^{H\tau}. \tag{11}$$

Similarly, for (10), providing the algorithm $x(k\tau)(k=1,2,\cdots)$ as accurately as possible involves introducing vectors $x'=y$ and the following algorithm:

$$\begin{gathered} x_k=\cos(\tau A)x_{k-1}+\sin(\tau A)A^{-1}y_{k-1}+A^{-1}\int_0^\tau \sin(sA)F(k\tau-s)ds, \\ y_k=-A\sin(\tau A)x_{k-1}+\cos(\tau A)y_{k-1}+\int_0^\tau \cos(sA)F(k\tau-s)ds, \\ x_0=x(0), y_0=x'(0)=x_1. \end{gathered} \tag{12}$$

To improve the accuracy of $U(k\tau)$ and $x(k\tau)$, it is essential to develop an algorithm with high accuracy for computing $T=e^{H\tau}, \cos(\tau A)$ and $\sin(\tau A)$ and the integral containing these matrix functions. This paper improves the algorithms developed for the computation of e^A and $\cos A$ in the existing literature, see [1–15], and also discusses the computation of other trigonometric, hyperbolic, logarithmic, and related inverse triangular and inverse hyperbolic matrices.

2. The General Theorem of the Fast Algorithm for Matrix Functions

In [16], we addressed the challenge of rapidly calculating elementary functions with specified precision. The implementation method involves an analytic function $f(x)$ specifying property,

$$f(x) = g\left(f\left(\frac{x}{k}\right)\right), \text{ where } k > 1. \tag{13}$$

If C_0 is an approximation as close as possible to $f\left(\frac{x}{k^N}\right)$, with N as a positive integer, for example,

$$C_0 = \sum_{j=0}^{m} \frac{f^{(j)}(0)(\Delta x)^j}{j!}, \Delta x = \frac{x}{k^N}, m = 1, 2, \cdots \tag{14}$$

or C_0 being the Padé approximation of $f\left(\frac{x}{k^N}\right)$, then the following calculations are performed:

$$\text{for}(i = 0, i < N, i++)\ C_{i+1} = g(C_i). \tag{15}$$

Completing (14) and (15), $f(x) \approx C_N$ is obtained. This method is also applicable to the calculation of matrix functions, where the variable x is replaced by the n by n square matrix A. Now, we have the following:

$$C_0 = \sum_{j=0}^{m} \frac{f^{(j)}(0)(\Delta A)^j}{j!}, \Delta A = \frac{1}{k^N} A, m = 1, 2, \cdots \tag{16}$$

or C_0 is the Padé approximation of $f(\Delta A)$. Our objective is to determine m and N to minimize the computation time under the condition that the truncation error and computational precision are 10^{-p}. For the matrix function calculation, the coefficient calculation of (14) can be ignored, mainly focusing on the multiplication calculation of the matrix in (16) and (15). If the number of multiplications to complete $g(A)$ is K, then the total number of times to complete (16) and (15) is roughly

$$q(m, N) = (m + KN)M_n, \tag{17}$$

where M_n is the number of times to complete one multiplication of the n by n matrix. Based on the above discussion, we present a fast algorithm for the matrix function as follows.

Theorem 1. *Let m_p be an integer close to the minimum point of $h(m)$, where*

$$h(m) = m + \frac{pK \ln 10}{(m+1)\ln k} + \frac{K}{(m+1)\ln k} \ln\left|\frac{f^{(m+1)}(0)}{(m+1)!}\right| \tag{18}$$

and

$$N = \left[\frac{p \ln 10}{(m+1)\ln k} + \frac{1}{(m+1)\ln k} \ln\left|\frac{f^{(m+1)}(0)}{(m+1)!}\right| + \frac{\ln\|A\|}{\ln k}\right] + 1. \tag{19}$$

(1) $C_0 = \sum_{j=0}^{m_p} \frac{f^{(j)}(0)(\Delta A)^j}{j!}$, where $\Delta A = \frac{1}{k^N} A$, or C_0 is a Padé approximation of $f(\Delta A)$.
(2) for$(i = 0, i < N, i++)\ C_{i+1} = g(C_i)$ and $(A) \approx C_N$.

The above algorithm is a fast algorithm for the general matrix function.

Proof. With a truncation error of 10^{-p}, for (16) to be satisfied,

$$\left\|\frac{f^{(m+1)}(0)(\Delta A)^{m+1}}{(m+1)!}\right\| = \left|\frac{f^{(m+1)}(0)}{(m+1)!k^{(m+1)N}}\right| \|A^{m+1}\| \le \left|\frac{f^{(m+1)}(0)}{(m+1)!k^{(m+1)N}}\right| \|A\|^{m+1} < 10^{-p}, \tag{20}$$

where $\|A\|$ is a norm of A. So,

$$N > \frac{p \ln 10}{(m+1)\ln k} + \frac{1}{(m+1)\ln k} \ln\left|\frac{f^{(m+1)}(0)}{(m+1)!}\right| + \frac{\ln\|A\|}{\ln k}. \tag{21}$$

From (17), we obtain the following:

$$q(m,N) > g(m) = \left(m + \frac{pK\ln 10}{(m+1)\ln k} + \frac{K}{(m+1)\ln k} \ln \left| \frac{f^{(m+1)}(0)}{(m+1)!} \right| + \frac{K \ln \|A\|}{\ln k} \right) M_n. \tag{22}$$

Note that the minimum point of $g(m)$ has nothing to do with $\frac{K\ln\|A\|}{\ln k}$. By making $q(m,N)$ as small as possible, the algorithm obtained by (1) and (2) is a fast algorithm, thus completing the proof of the theorem. □

Remark 1. *In the specific calculation, $\|A\|$ can be replaced by the approximate eigenvalue with the maximum modulus in this paper.*

$$\left| \frac{f^{(m+1)}(0)}{(m+1)!k^{(m+1)N}} \right| \|A\|^{m+1} < 10^{-p}. \tag{23}$$

3. Fast Algorithm for e^A, $\cos A$, $\sin A$, $\sinh A$, and $\cosh A$

Suppose $\lambda = \|A\|$ is a norm for A. If we let $A = \lambda A_0$, then $\|A_0\| \leq 1$. From (19), we can obtain the following:

$$N_p = \left[\frac{p\ln 10}{(m+1)\ln k} + \frac{1}{(m+1)\ln k} \ln \left| \frac{f^{(m+1)}(0)}{(m+1)!} \right| + \frac{\ln \lambda}{K \ln k} \right] + 1. \tag{24}$$

In order to reduce the length of the article, the algorithms for several matrix functions are compiled into a table.

Theorem 2. *For $e^A, \cos A, \sin A, \cosh A$, and $\sinh A$, we devised the following fast algorithm using Theorem 1.*

Remark 2. *It is worth noting that m_p in Table 1 only depends on the calculation accuracy p and the matrix function, and is independent of matrix A.*

Table 1. The fast algorithm for $e^A, \cos A, \sin A, \cosh A$, and $\sinh A$.

	m_p, N_p	Algorithm
e^A	$m_p = \left[\sqrt{\frac{9p\ln 10}{8\ln 2}} \right], N_p = [N_e] + 1$	(1) $B = \frac{A}{2^{N_p}}, C = I + \sum_{j=1}^{m_p} \frac{B^j}{j!}$ (2) $for(i = 0, i < N, i++)\ C = C^2$ (3) $e^A = C$
$\left\{ \begin{array}{c} \sin A \\ \sinh A \end{array} \right\}$	$m_p = \left[\sqrt{\frac{p\ln 10}{2\ln 3}} \right], N_p = [N_{\sin}] + 1$	(1) $B = \frac{A^2}{3^{2N_p}}, C = \frac{A}{3^N} + \frac{A}{3^N} \sum_{j=1}^{m_p} \frac{(\mp B)^j}{(2j+1)!}$ (2) $for(i = 0, i < N, i++)\ C = 3C \mp 4C^3$ (3) $\left\{ \begin{array}{c} \sin A \\ \sinh A \end{array} \right\} = C$
$(I)\ \left\{ \begin{array}{c} \cos A \\ \cosh A \end{array} \right\}$	$m_p = \left[\sqrt{\frac{p\ln 10}{2\ln 3}} \right] + 1, N_p = [N_{\sin}]$	(1) $B = \frac{A^2}{3^{2N_p}}, C = I + \sum_{j=1}^{m_p} \frac{(\mp B)^j}{(2j)!}$ (2) $for(i = 0, i < N, i++)\ C = 4C^3 - 3C$ (3) $\left\{ \begin{array}{c} \cos A \\ \cosh A \end{array} \right\} = C$
$(II)\ \left\{ \begin{array}{c} \cos A \\ \cosh A \end{array} \right\}$	$m_p = \left[\sqrt{\frac{p\ln 10}{2\ln 3}} \right] + 1, N_p = [N_{\sin}]$	(1) $B = \frac{A^2}{3^{2N_p}}, C = I + \sum_{j=1}^{m_p} \frac{(\mp B)^j}{(2j)!}$ (2) $for(i = 0, i < N, i++)\ C = C^2 - I$ (3) $\left\{ \begin{array}{c} \cos A \\ \cosh A \end{array} \right\} = C$
$N_e = \frac{p\ln 10}{(m_p+1)\ln 2} + \frac{\ln\lambda}{\ln 2} - \frac{\ln(m_p+1)!}{(m_p+1)\ln 2}, N_{\cos} = \frac{p\ln 10}{(2m_p+1)\ln 2} + \frac{\ln\lambda}{\ln 2} - \frac{\ln(2m_p+1)!}{(2m_p+1)\ln 2},$ $N_{\sin} = \frac{p\ln 10}{(2m_p+2)\ln 3} + \frac{\ln\lambda}{\ln 3} - \frac{\ln(2m_p+2)!}{(2m_p+2)\ln 3}.$		

Now, we utilize the general fast algorithm to specifically demonstrate the correctness of the algorithms in Table 1. The algorithms in Table 1 are straightforward; they involve placing the specific functions f and g into the general algorithm, and the tricky part lies in deriving the formulas of m_p and N_p. We calculate the formulas for m_p and N_p for the function e^A for $12 \leq p \leq 36$ here. In fact, the graphing of the function of $h(m)$ supports our results for all values of p. The derivation of m_p and N_p for the other functions is very similar.

(1) For the function e^A, we have $g(u) = u^2, k = 2, K = 1$, and

$$h(m) = m + \frac{p \ln 10}{(m+1)\ln 2} - \frac{1}{(m+1)\ln 2} \ln(m+1)!. \tag{25}$$

Using Sterling's approximation formula, we obtain the following:

$$\ln((m+1)!) = \frac{1}{2}\ln(2\pi) + \frac{1}{2}\ln(m+1) + (m+1)\ln(m+1) - m - 1 + O(\frac{1}{m+1}).$$

Now, implementing it into (25), we obtain the following:

$$h(m) \approx m + \frac{p \ln 10}{(m+1)\ln 2} - \frac{\ln(2\pi)}{2(m+1)\ln 2} - \frac{\ln(m+1)}{2(m+1)\ln 2} + \frac{1}{\ln 2}.$$

Let

$$q(x) = x + \frac{p \ln 10}{(x+1)\ln 2} - \frac{\ln(2\pi)}{2(x+1)\ln 2} - \frac{\ln(x+1)}{2(x+1)\ln 2} + \frac{1}{\ln 2}.$$

Taking its derivative with respect to x, one obtains

$$q'(x) = 1 - \frac{p \ln 10}{(x+1)^2 \ln 2} + \frac{\ln(2\pi)}{2(x+1)^2 \ln 2} + \frac{\ln(x+1)}{2(x+1)^2 \ln 2} - \frac{1}{2(x+1)^2 \ln 2}.$$

To determine extreme values, we set it to 0 and multiply both sides by $(m+1)^2$:

$$(x+1)^2 - \frac{1}{\ln 2}(x+1) + \frac{\ln(x+1)}{2\ln 2} - \frac{2p\ln 10 - \ln(2\pi) + 1}{2\ln 2} = 0.$$

Using simple algebra, we obtain the following:

$$(x+1)^2 > \frac{(m+1) - \ln(m+1) + (m+1) - \ln(2\pi)}{2\ln 2} + \frac{2p\ln 10 + 1}{2\ln 2} > \frac{2p\ln 10 + 1}{2\ln 2}.$$

Using $p = 12$, we obtain a lower bound for x :

$$x > \sqrt{\frac{2p\ln 10 + 1}{2\ln 2}} - 1 = 6.37.$$

Thus, we have a lower bound for m: $m \geq 6$. We also can solve the quadratic equation to get

$$m + 1 = \frac{1}{\ln 2} + \sqrt{\frac{1}{4\ln^2 2} + \frac{8p\ln 10 - 4\ln(2\pi) + 4}{8\ln 2}}.$$

Now, using the smallest value for m obtained and $p = 36$, we can obtain our first upper bound:

$$m < -1 + \frac{1}{\ln 2} + \sqrt{\frac{1}{4\ln^2 2} - \frac{\ln 7}{2\ln 2} + \frac{276\ln 10 - 4\ln(2\pi) + 4}{8\ln 2}} \approx 11.31.$$

We can again use the values of m and $p = 12$ to update the lower bound of m as follows:

$$m > -1 + \frac{1}{\ln 2} + \sqrt{\frac{1}{4\ln^2 2} + -\frac{\ln 12}{2\ln 2} + \frac{96\ln 10 - 4\ln(2\pi) + 4}{8\ln 2}} \approx 6.61.$$

Now, our new lower bound for m is 7. With this small range of values for m, by inspecting the formula for the values of m, we can see that the value of m is very close to $\frac{p\ln 10}{\ln 2}$, and we can safely claim that m$= \left[\sqrt{\frac{9p\ln 10}{8\ln 2}}\right]$. Our following numerical calculation supports this claim completely.

Case (a): $p = 12$. Using the formula obtained by the quadratic formula and the upper bound $m = 11$, we obtain a new lower bound: $m \geq 6.63$. Using the formula obtained by the quadratic formula and lower bound $m = 7$, we can also obtain the upper bound: $m \leq 6.63$. Using our claim $\frac{9p\ln 10}{8\ln 2}$, we obtain $m = 6.70$. This clearly matches the lower and upper bound quite well.

Case (b): $p = 18$. Using the formula obtained by the quadratic formula and the upper bound $m = 11$, we obtain a new lower bound: $m \geq 8.05$. Using the formula obtained by the quadratic formula and lower bound $m = 7$, we can also obtain the upper bound: $m \leq 8.07$. Using our claim $\frac{9p\ln 10}{8\ln 2}$, we obtain $m = 8.20$. These values match very well for taking the closest integer.

Case (c): $p = 24$. Using the formula obtained by the quadratic formula and the upper bound $m = 11$, we obtain a new lower bound: $m \geq 9.27$. Using the formula obtained by the quadratic formula and lower bound $m = 7$, we can also obtain the upper bound: $m \leq 9.28$. Using our claim $\frac{9p\ln 10}{8\ln 2}$, we obtain $m = 9.47$.

Case (d): $p = 36$. Using the formula obtained by the quadratic formula and the upper bound $m = 11$, we obtain a new lower bound: $m \geq 11.29$. Using the formula obtained by the quadratic formula and lower bound $m = 7$, we can also obtain the upper bound: $m \leq 11.37$. Using our claim $\frac{9p\ln 10}{8\ln 2}$, we obtain $m = 11.60$.

Using $m_p = \sqrt{\frac{9p\ln 10}{8\ln 2}}$, we can obtain

$$N_p = \left[\frac{p\ln 10}{(m_p+1)\ln 2} - \frac{\ln(m_p+1)!}{(m_p+1)\ln 2} + \frac{\ln\lambda}{\ln 2}\right] + 1. \tag{26}$$

(2) For $\left\{\begin{array}{c}\sin A\\ \sinh A\end{array}\right\}$, we have $g(u) = 3u \mp 4u^3, k = 3, K = 2$ and

$$h(m) = m + \frac{2p\ln 10}{(2m+2)\ln 3} - \frac{2\ln(2m+2)!}{(2m+2)\ln k}. \tag{27}$$

In this case, for $p = 12, 16, 24$, and 32, the corresponding integer points for $h(m)$ to achieve its minimum value are $3, 3, 5$, and 6. Again, it can be estimated that

$$m_p = \sqrt{\frac{3p\ln 10}{5\ln 3}}, N_p = \left[\frac{p\ln 10}{(2m_p+2)\ln 3} - \frac{\ln(2m_p+2)!}{(2m_p+2)\ln 3} + \frac{\ln\lambda}{\ln 3}\right] + 1. \tag{28}$$

(3) For $\left\{\begin{array}{c}\cos A\\ \cosh A\end{array}\right\}$, there are two cases:

Case (I): $g(u) = 4u^3 - 3u, k = 3, K = 2$, and (29) hold. Compared to I, the values in the second part of $C = I + \sum_{j=1}^{m_p} \frac{(-B)^j}{(2j)!}$ are very small, so we may replace N with $N - 1$, and m_p with $m_p + 1$.

Case (II): $g(u) = u^2 \mp 1, k = 2, K = 1$ and

$$h(m) = m + \frac{p\ln 10}{(2m+1)\ln 2} - \frac{1}{(2m+1)\ln 2}\ln(2m+1)!. \tag{29}$$

Similarly, for $p = 12, 16, 24$, and 32, the corresponding integer points for $h(m)$ to achieve its minimum value are $4, 5, 6$, and 7. Consequently, we can determine the following:

$$m_p = \sqrt{\frac{3p\ln 10}{5\ln 2}}, N_p = \left[\frac{p\ln 10}{(2m_p+1)\ln 2} - \frac{\ln(2m_p+1)!}{(2m_p+1)\ln 2} + \frac{\ln\lambda}{\ln 2}\right] + 1. \tag{30}$$

The effectiveness of the algorithms is displayed in Table 1 by implementing numerical computations in Mathematica. N_p and m_p are replaced with $N_{p,\mu} = \mu N_p, m_{p,\mu}$ in Table 1, defined correspondingly by

$$\begin{aligned}
m_{p,\mu} &= \min_{\|B^j/j!\|<10^{-p}} j, B = \left\|\frac{A}{2^{N_{p,\mu}}}\right\| \text{ for } e^A,\\
m_{p,\mu} &= \min_{\|B^{j+1/2}/(2j+1)!)\|<10^{-p}} j, B = \left\|\frac{A^2}{3^{2N_{p,\mu}}}\right\| \text{ for } \sin A,\\
\text{(I) } m_{p,\mu} &= \min_{\|B^j/(2j)!\|<10^{-p}} j, B = \left\|\frac{A^2}{3^{2N_{p,\mu}}}\right\| \text{ for } \cos A,\\
\text{(II) } m_{p,\mu} &= \min_{\|B^j/(2j)!\|<10^{-p}} j, B = \left\|\frac{A^2}{2^{2N_{p,\mu}}}\right\| \text{ for } \cos A,
\end{aligned} \tag{31}$$

where one has $N_{p,1} = N_p$ and $m_p = m_{p,1}$.

For comparison, let A take the following form:

$$A = UDU^{-1}, \tag{32}$$

where U is a randomly generated n by n matrix, $D_a = Dial(d_1, d_2, \cdots, d_n)$ is a randomly generated n by n diagonal matrix with $d_i \in (a, b), i = 1, 2, \cdots, n$, so $f(A) = Uf(D)U^{-1}$, $f(D) = Dial(f(d_1), f(d_2), \cdots, f(d_n))$. In all the examples in this paper, we agree on the relative error as follows:

$$err(g)(A) = \frac{\|f(A) - Uf(D)U^{-1}\|_\infty}{\|Uf(D)U^{-1}\|}. \tag{33}$$

Let $A_i (i = 1, 2, \cdots, M)$ be M matrices produced by (32). From Figure 1, it is evident that the calculation accuracy is essentially the same for $\mu = 1, \frac{2}{3}, \frac{4}{3}$, but $\mu = 1$ provides the highest efficiency. The double-angle formula is notably more efficient and accurate than the triple-angle formula for $\cos A$. Henceforth, we will exclude the case that utilizes the triple-angle formula for $\cos A$.

Now let us juxtapose the algorithms presented in Table 1 with the internal functions and commonly used algorithms found in mathematical software. For ease of reference, let $err(g)$ denote the calculation error of matrix function $f(A_i)(i = 1, 2, \cdots, M)$, where a blank g implies the utilization of the algorithms in Tables 1–3 of this article. If $g = Sym, Pade, \cdots$, it signifies calculation according to the internal function, Padé approximation, $\cdots$, respectively. When $p = 16$, $m_p = 7, 4, 3$ for $e^A, \cos A(\cosh A), \sin A(\sinh A)$, respectively. We denote the Padé approximation of $\sin x$ as $r_{66}(x)$ and the Padé approximation of e^x and $\cos x$ as $r_{88}(x)$. In Table 1, the Padé approximation of $e^A, \cos A(\cosh A), \sin A(\sinh A)$ is as follows:

$$\begin{aligned}
r_{88}(x) &= \frac{1+\frac{x}{2}+\frac{x^2}{60}+\frac{x^3}{60}+\frac{x^4}{624}+\frac{x^5}{9360}+\frac{x^6}{205920}+\frac{x^7}{7207200}+\frac{x^8}{518918400}}{1-\frac{x}{2}+\frac{x^2}{60}-\frac{x^3}{60}+\frac{x^4}{624}-\frac{x^5}{9360}+\frac{x^6}{205920}-\frac{x^7}{7207200}+\frac{x^8}{518918400}} \text{ for } e^x,\\
r_{88}(x) &= \frac{1\mp\frac{260735x^2}{545628}+\frac{4375409x^4}{141863280}\mp\frac{7696415x^6}{1310867072}+\frac{807373373x^8}{2359470072960}}{1\pm\frac{12079x^2}{545628}+\frac{34709x^4}{141863280}\pm\frac{109247x^6}{65540835360}+\frac{11321x^8}{1814976979200}} \text{ for } \left\{\begin{array}{c}\cos x\\ \cosh x\end{array}\right\},\\
r_{66}(x) &= \frac{x\mp\frac{7x^3}{60}+\frac{x^5}{600}}{1\pm\frac{x^2}{20}+\frac{x^4}{600}\pm\frac{x^6}{14400}} \text{ for } \left\{\begin{array}{c}\sin x\\ \sinh x\end{array}\right\}.
\end{aligned} \tag{34}$$

For the algorithms in Table 1, the $err(g)(A)$ of random matrix from order 1 to 800 is as follows.

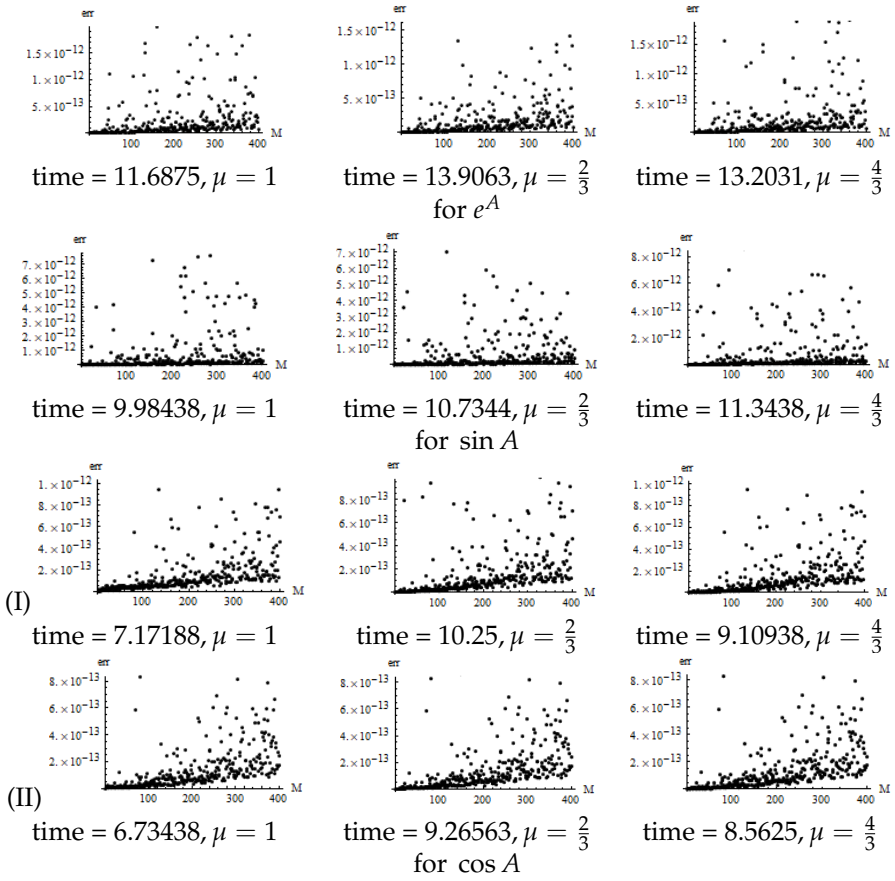


Figure 1. Error comparison of the algorithms in Table 1 for $\mu = 1, \frac{2}{3}, \frac{4}{3}$.

Table 2. The fast algorithms for $\tan A$, $\sec A$, $\tanh A$, and $sechA$.

	m_p, N	**Algorithm**
$\left\{ \begin{array}{c} \tan A \\ \tanh A \end{array} \right\}$	$m_p = \left[\sqrt{\frac{2p\ln 10}{\ln 3}}\right], N = N_1 + 1$	$B = \frac{A^2}{3^{2N}}$, $C = \frac{2A}{3^N}\sum_{j=0}^{m}(\pm 1)^j \sigma(j+1)\left(4^{j+1}-1\right)B^j$ $for(i=0, i<N, i++)$ $\quad C = C(3I \mp C^2)(I \mp 3C^2)^{-1}$ $\left\{ \begin{array}{c} \tan A \\ \tanh A \end{array} \right\} \approx C$
$\left\{ \begin{array}{c} \sec A \\ \sec hA \end{array} \right\}$	$m_p = \left[\sqrt{\frac{2p\ln 10}{\ln 3}}\right], N = N_2 + 1$	$B = \frac{A^2}{3^{2N}}, I + \sum_{j=1}^{m}(\mp 1)^j \widehat{E}_j B^j$ $for(i=0, i<N, i++)$ $\quad C = C^3(4I - 3C^2)^{-1}$ $\left\{ \begin{array}{c} \sec A \\ \sec hA \end{array} \right\} \approx C$
$N_1 = \left[\frac{p\ln 10 + \ln\left(2\sigma(m_p+2)\right)}{(2m_p+3)\ln 3} + \frac{\ln\lambda}{\ln 3}\right], N_2 = \left[\frac{p\ln 10 + \ln \widehat{E}_{m_p+1}}{(2m_p+3)\ln 3} + \frac{\ln\lambda}{\ln 3}\right]$.		

It is evident from Figure 2 that, for e^A, $\cos A$, and $\sin A$, the accuracy obtained by the three methods is essentially the same. However, when the matrix order is relatively high, the Taylor expansion method presented in Table 1 outperforms the others. In particular, for cos and sin A, the calculation time of the internal functions is approximately eight times longer than that of the algorithms in Table 1. Numerical calculations demonstrate that while the calculation speed of $\cos A$ and $\sin A$ using the internal functions $Re(e^{iA}) = \cos A$

and $Im(e^{iA}) = \sin A$ is faster than that of the internal functions $\cos A$ and $\sin A$, their calculation accuracy is relatively poor.

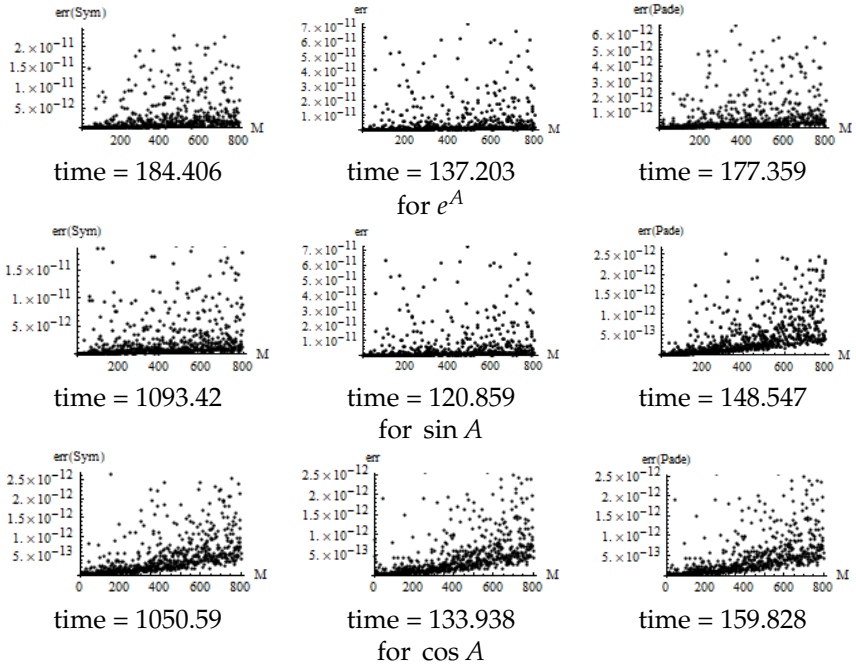


Figure 2. Error of e^A, $\cos A$, and $\sin A$ for three algorithms.

Table 3. The fast algorithms for $\cot A$, $\coth A$, $\csc A$, and $\mathrm{csch} A$.

	m_p, N	Algorithm
$\left\{ \begin{array}{c} \cot A \\ \coth A \end{array} \right\}$	$m_p = \left[\sqrt{\frac{7p\ln 10}{5\ln 3}}\right], N = N_1 + 1$	$B = \frac{A^2}{3^{2N}}, C = I - 2\sum_{j=1}^{m}(\pm 1)^j \sigma(j) B^j$ $for(i = 0, i < N, i++)$ $(C = 3(C(3B \mp C^2)(B \mp 3C^2)^{-1}; B = 9B)$ $\left\{ \begin{array}{c} \cot A \\ \coth A \end{array} \right\} \approx C$
$\left\{ \begin{array}{c} \csc A \\ \mathrm{csch} A \end{array} \right\}$	$m_p = \left[\sqrt{\frac{5p\ln 10}{2\ln 3}}\right], N = N_2 + 1$	$B = \frac{A^2}{3^{2N}}, I + 2\sum_{j=1}^{m+1}(\pm 1)^j \sigma(j)\left(1 - \frac{2}{4^j}\right) B^j$ $for(i = 0, i < N, i++)$ $(C = 3C^3(3C^2 \mp 4B)^{-1}; B = 9B)$ $\left\{ \begin{array}{c} \csc A \\ \mathrm{csch} A \end{array} \right\} \approx C$
$N_1 = \left[\frac{p\ln 10 + \ln(2\sigma(m_p+2))}{(2m_p+3)\ln 3} + \frac{\ln\lambda}{\ln 3}\right], N_2 = \left[\frac{p\ln 10 + \ln \widehat{E}_{m_p+1}}{(2m_p+3)\ln 3} + \frac{\ln\lambda}{\ln 3}\right]$		

For $\cosh A$ and $\sinh A$, the internal functions can be calculated by utilizing the following equation:

$$\cosh A = \frac{1}{2}(e^A + e^{-A}), \sinh A = \frac{1}{2}(e^A - e^{-A}). \tag{35}$$

Our experiment provides the following results for $err(g)(A)$ of random matrices with orders from 1 to 800.

It is evident from Figure 3 that using (35) not only doubles the calculation time but also makes the calculation error substantial. Furthermore, by observing Figures 2 and 3, it becomes apparent that the algorithms in Table 1 outperform the Padé approximation algorithm.

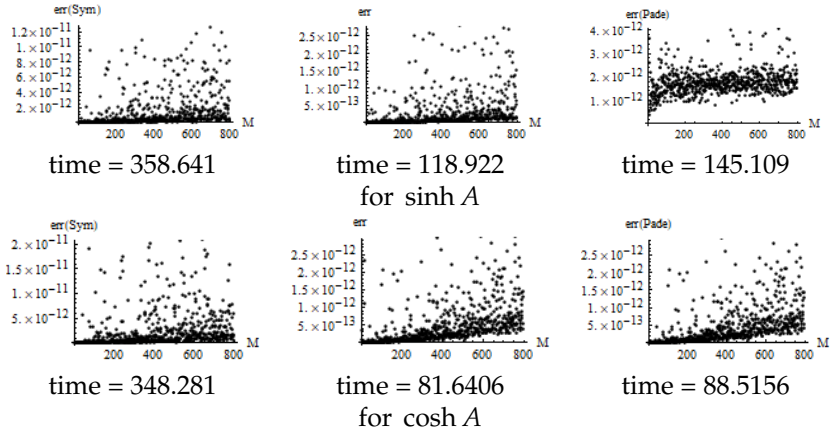


Figure 3. Error of sinh A and cosh A for three algorithms.

4. Fast Algorithm for tan A, tanh A, sec A, and $sechA$

Now let us contemplate the swift computation of tan A, tanh A, sec A, and $sechA$. Although these matrix functions can be computed using the following approach, we aim to directly compute them, as demonstrated in the previous section.

$$\begin{aligned} &\tan A = \sin A(\cos A)^{-1}, \sec A = (\cos A)^{-1}, \\ &\tanh A = \sinh A(\cosh A)^{-1} = \left(e^{A} - e^{-A}\right)\left(e^{A} + e^{-A}\right)^{-1}, \\ &sechA = (\cosh A)^{-1} = 2\left(e^{A} + e^{-A}\right)^{-1}. \end{aligned} \tag{36}$$

While the Padé approximation significantly contributes to matrix function computation, the numerical results in the previous section indicate that, based on the algorithm presented in this paper, neither the calculation accuracy nor the calculation speed is predominant. Therefore, this section does not adopt it.

Theorem 3. *For* tan A, sec A, tanh A, *and* $sechA$, *we have the following fast algorithm based on Theorem 1.*

In Table 2, for $\sigma(j)$ and $\widehat{E}_j$, we have the following recursive formula [1]:

$$\begin{aligned} &\sigma(2n) = \tfrac{4}{4n+1}\sum_{l=1}^{n-1}\sigma(l)\sigma(2n-l) + \tfrac{2}{4n+1}\sigma^{2}(n), n = 1, 2, \cdots, \\ &\sigma(2n-1) = \tfrac{4}{4n-1}\sum_{l=1}^{n-1}\sigma(l)\sigma(2n-1-l), n = 2, 3, \cdots \\ &\widehat{E}_n = -\sum_{k=0}^{n-1}\tfrac{\widehat{E}_k}{(2n-2k)!}, \widehat{E}_0 = 1, \sigma(1) = \tfrac{1}{6}. \end{aligned} \tag{37}$$

Using the algorithms in Table 2, our experiments generated the following data for the $err(g)(A)$ of random matrices from order 1 to 200.

Here, *Sym* indicates that matrix functions tan A, tanh A, sec A, and sech$A$, expressed by $\sin A(\cos A)^{-1}$, $e^{A} - e^{-A}\left(e^{A} + e^{-A}\right)^{-1}$, $(\cos A)^{-1}$ and $2\left(e^{A} + e^{-A}\right)^{-1}$, respectively, are computed using the internal functions. As observed in Figure 4, the algorithm's computation speed in Table 2 surpasses that of the internal function, particularly for tan A and sec A, where the speed is increased by 8–11 times compared to the internal function, while maintaining consistent accuracy is consistent. When the order n of the matrix is relatively large, for tanh A and sec hA, the calculation speed is twice that of the internal function, yet the precision remains the same as the internal function.

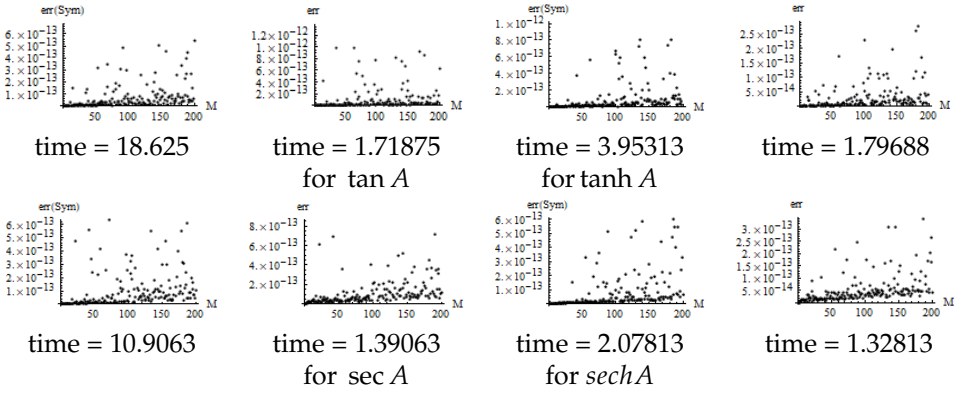


Figure 4. Error of the tan A, sec A, tanh A, and $sechA$ for two algorithms.

Let us consider implementing our algorithms for calculating the functions cot A, cot A, csc A, and csch$A$. By

$$\begin{aligned}
&\cot(At) \approx A^{-1} - 2\sum_{j=0}^{m} \sigma(j+1)A^{2j+1};\ (o < |A| < \pi, z \neq 0),\\
&\coth(At) \approx A^{-1} + 2\sum_{j=0}^{m} (-1)^j\sigma(j+1)A^{2j+1} \quad (0 < |A| < \pi),\\
&\operatorname{csc}(At) \approx A^{-1} + 2\sum_{j=0}^{m} \sigma(j+1)\left(1 - \tfrac{2}{4^{j+1}}\right)A^{2j+1} \quad (0 < |A| < \pi),\\
&\operatorname{csch}(At) \approx A^{-1} - 2\sum_{j=0}^{m} (-1)^j\sigma(j+1)\left(1 - \tfrac{2}{4^{j+1}}\right)A^{2j+1} \quad (0 < |A| < \pi),
\end{aligned} \tag{38}$$

we see that when $\|A\|$ is small, the norm of the inverse of A will be large, so the calculation using (38) will produce large errors. Therefore, we must utilize the following method:

$$\begin{aligned}
&\cot A = A^{-1}\cot_1 A, \coth A = A^{-1}\coth_1 A,\\
&\operatorname{csc} A = A^{-1}\operatorname{csc}_1 A, \operatorname{csch} A = A^{-1}\operatorname{csch}_1 A,
\end{aligned} \tag{39}$$

where

$$\begin{aligned}
&\cot_1 A \approx I - 2\sum_{j=1}^{m+1} \sigma(j)A^{2j};\ (|z| < \pi, z \neq 0),\\
&\coth_1 A \approx I - 2\sum_{j=1}^{m+1} (-1)^j\sigma(j)A^{2j} \quad (|z| < \pi, z \neq 0),\\
&\operatorname{csc}_1 A \approx I + 2\sum_{j=1}^{m+1} \sigma(j)\left(1 - \tfrac{2}{4^j}\right)A^{2j} \quad (|z| < \pi, z \neq 0),\\
&\operatorname{csch}_1 A \approx I + 2\sum_{j=1}^{m+1} (-1)^j\sigma(j)\left(1 - \tfrac{2}{4^j}\right)A^{2j} \quad (|z| < \pi, z \neq 0).
\end{aligned} \tag{40}$$

Theorem 4. *Using Theorem 1, we have the following algorithm for* cot A, coth A, csc A, *and* sec hA.

Implementation of the algorithms in Table 3 provides the following results for the matrix functions in (39).

Here, *Sym* signifies that matrix functions cot A, coth A, csc A, and sec hA are calculated using the internal functions with the regular formulas $\cos A(\sin A)^{-1}$, $\left(e^A + e^{-A}\right)\left(e^A - e^{-A}\right)^{-1}$, $(\sin A)^{-1}$, and $2\left(e^A - e^{-A}\right)^{-1}$, respectively. The results in Figure 5 are essentially consistent with the corresponding results in Figure 2.

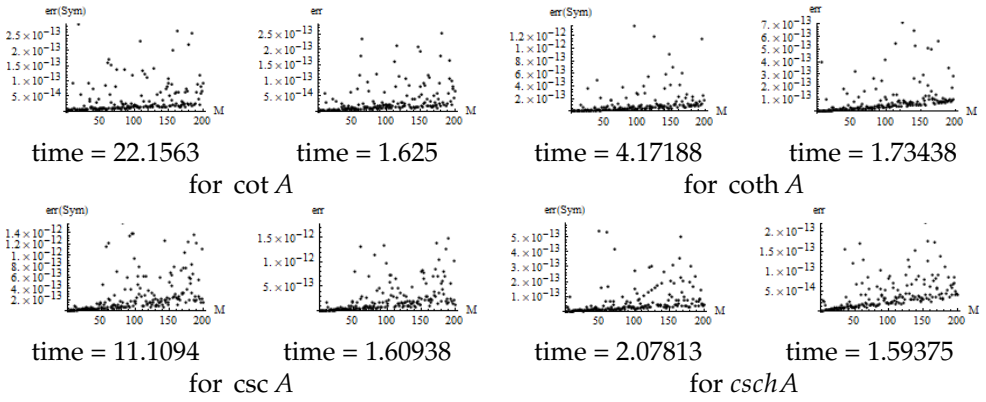


Figure 5. Error of cot A, cot A, csc A, and *cschA for two algorithms*.

5. Fast Algorithm for Logarithmic Matrix Function ln A and Some Related Functions

People consider various algorithms [17–22] for logarithmic matrix function ln A. One of the most effective methods we noticed was the following algorithm. The standard way of dealing with this problem is to use the square root operator repeatedly to bring A near the identity:

$$\ln A = 2^k \ln A_k, A_k = A^{1/2^k}. \tag{41}$$

As k increases, $A^{1/2^k} \to I$, so for sufficiently large k, we can apply a direct method to $A^{1/2^k}$. This procedure for the logarithm was introduced by Kenney and Laub [19]. The specific method is to choose k such that

$$\left\| A^{1/2^k} - I \right\| < \alpha < 1, \tag{42}$$

and usually, k is selected from 8. We slightly optimize the method as follows:

$$\ln A = \ln \lambda \cdot I + 2^k \ln A_k, A_k = A_0^{1/2^k}, A_0 = \frac{1}{\|A\|} A, \tag{43}$$

and

$$\left\| A_0^{1/2^k} - I \right\| < \alpha < 1, \tag{44}$$

where k is usually selected from 4.

For ln x, the algorithm that converges faster is

$$\ln x = \ln \frac{1-y}{1+y} \approx -2 \sum_{j=1}^{m} \frac{y^{2j+1}}{2j+1}, \tag{45}$$

where $y = \frac{1-x}{1+x}$. For ln A_k, we have

$$\ln A_k = \ln(I - B) - \ln(I + B) \approx -2 \sum_{j=1}^{m} \frac{B^{2j}}{2j+1}, \tag{46}$$

where $B = (I - A_k)(I + A_k)^{-1}$. Applying the Euler–Abel transformation [21], we obtain the following accelerated series:

$$\ln A_k \approx \tfrac{1}{3} BF^2 (I - 12 \sum_{j=1}^{m} \frac{B^{2j}}{(2j+3)(4j^2-1)}) - F, \tag{47}$$

where $F = 2B(I - B^2)^{-2}$. We can use the following Padé approximation of ln A_k [20].

$$\begin{aligned} \ln A_k &\approx (B + 2B^2 + \tfrac{47}{36} B^3 + \tfrac{11}{36} B^4 + \tfrac{137}{7560} B^5) F, \\ F &= (I + \tfrac{5}{2} B + \tfrac{20}{9} B^2 + \tfrac{5}{6} B^3 + \tfrac{5}{42} B^4 + \tfrac{1}{252} B^5)^{-1}, \end{aligned} \tag{48}$$

where $B = A_k - I$. We also provide some numerical results for the matrix function $\ln A$.

Based on (41)–(44) we also provide some numerical results of $\ln A$ using (46)–(48). It can be seen from Figure 6 that through optimization (43), the calculation accuracy of (46)–(48) is significantly improved, and the calculation speed of (46)–(48) is more than twice as fast as the inner function. The calculation speed advantage of the Padé approximation is obvious.

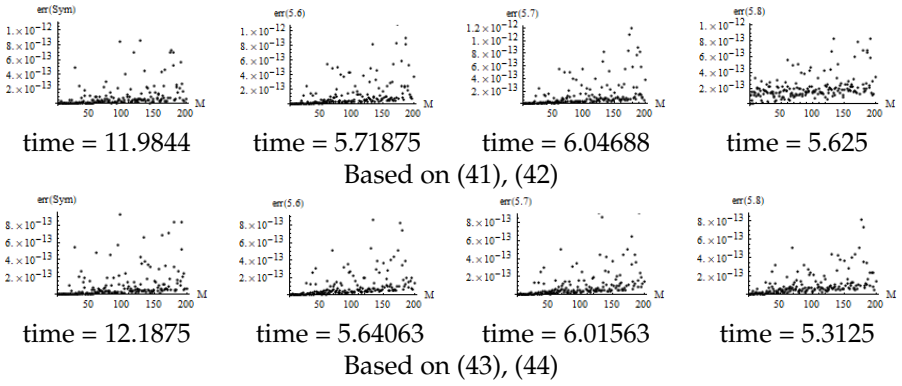


Figure 6. Error of $\ln A$ for the several algorithms.

We used the following identities to implement our general algorithm for the computations of the inverse trigonometric and hyperbolic functions. To control the length of the paper, we will not display the results here.

$$\begin{aligned} &\arcsin x = 2\arcsin \tfrac{x}{\sqrt{2(1+\sqrt{1-x^2})}}, \\ &\arctan x = 2\arctan \tfrac{x}{1+\sqrt{1+x^2}}, \\ &\arcsin x = x + \sum_{l=1}^{\infty} \tfrac{(2l-1)x^{2l+1}}{(2l)!(2l+1)}, \arctan x = \sum_{l=0}^{\infty} \tfrac{x^{2l+1}}{(2l+1)}, \end{aligned} \tag{49}$$

$$\begin{aligned} &\mathrm{arcsinh} x = 2\mathrm{arcsinh} \tfrac{x}{\sqrt{2(1+\sqrt{1+x^2})}}, \mathrm{arctanh} x = 2\mathrm{arctanh} \tfrac{x}{1+\sqrt{1-x^2}}, \\ &\mathrm{arcsinh} x = x + \sum_{l=1}^{\infty} \tfrac{(-1)^l(2l-1)x^{2l+1}}{(2l)!(2l+1)}, arctanhx = \sum_{l=0}^{\infty} \tfrac{(-1)^l x^{2l+1}}{(2l+1)}. \end{aligned} \tag{50}$$

For $\arcsin A, \arctan A, \mathrm{arcsinh} A,$ and $\mathrm{arctanh} A,$ under certain conditions, we can use the following identity.

$$\begin{aligned} &\arcsin A = -i\ln\left(\sqrt{I-A^2} + iA\right), \mathrm{arcsinh} A = \ln\left(\sqrt{I+A^2} + A\right), \\ &\arctan A = \tfrac{i}{2}\ln\left((iI+A)(iI-A)^{-1}\right), \\ &\mathrm{arctanh} A = \tfrac{1}{2}\ln\left((I+A)(I-A)^{-1}\right). \end{aligned} \tag{51}$$

6. Improved Calculation Formula for Initial Value Problems (3) and (8)

Now, let us consider the calculation of the integral in (11), i.e.,

$$\widehat{F}_k = \int_0^{\tau} e^{Hs} F_H(k\tau - s)ds. \tag{52}$$

If numerical integration is used to calculate this integral, it is quite time-consuming, and the calculation accuracy cannot be guaranteed. We adopt the following method here. For $nn = 1, 2, \cdots$, there is the following relationship:

$$\begin{aligned} \widehat{F}_k &= \sum_{j=0}^{nn-1} \int_{\frac{j\tau}{nn}}^{\frac{j\tau}{nn}} e^{Hs} F_H(k\tau - s)ds = \sum_{j=0}^{nn-1} T_{nn,j} \int_0^{\frac{\tau}{nn}} e^{Hs} F_H(k\tau - \tfrac{j\tau}{nn} - s)ds \\ &\approx \sum_{j=0}^{nn-1} T_{nn,j} \sum_{l=0}^{m} \tfrac{H^l}{l!} \int_0^{\frac{\tau}{nn}} s^l F_H(k\tau - \tfrac{j\tau}{nn} - s)ds, \end{aligned} \tag{53}$$

where

$$T_{nn,0} = I, T_{nn} = e^{H\frac{\tau}{nn}}, T_{nn,j} = e^{H\frac{j\tau}{nn}} = T_{nn}T_{nn,j-1}, j = 1, 2, \cdots, nn-1. \tag{54}$$

T_{nn} can be calculated by the following method:

$$\begin{array}{l} T_{nn} = I, zz = I; \\ \text{for}(i = 1, i \le m, i++)\ (zz = \frac{\frac{\tau}{nn}}{i} H \times zz; T_{nn} = T_{nn} + zz). \end{array} \tag{55}$$

So, (53) can be written as follows:

$$\widehat{F}_k \approx \sum_{j=0}^{nn-1} T_{nn,j} H_j, H_j = \sum_{l=0}^{m} \frac{H^l}{l!} \int_0^{\frac{\tau}{nn}} s^l F_H(k\tau - \frac{j\tau}{nn} - s)ds, \tag{56}$$

so we can convert the integral of the product of matrix exponential function and vector function into the integral of pure vector function.

The improved calculation formula for (3) is as follows:

$$\begin{array}{l} (55), (56), \\ U_k = TU_{k-1} + \sum_{j=0}^{nn-1} T_{nn,j} H_j, k = 1, 2, \cdots, \end{array} \tag{57}$$

where $T = e^{H\tau}$ is calculated by the algorithm in Table 1, and $T_{nn,j}$ *and* H_j are calculated by (55) and (56).

For

$$\int_0^{\frac{\tau}{nn}} s^l F_H(k\tau - \frac{j\tau}{nn} - s)ds, \tag{58}$$

the following two algorithms can be used.

(1) Integrable function method If $F(t) = (f_1(t), f_2(t), \cdots, f_v(t), \cdots, f_n(t))^T$, $f_v(t)$ is the algebraic sum of functions of the following types:

$$t^\alpha, t^l e^{-\omega t}, t^l \sin \omega t, t^l \cos \omega t, \tag{59}$$

then (58) can be expressed in analytical form, so H_j can also be expressed in analytical form. In this way, the calculation of (57) can be completed.

(2) Gauss three-point integral formula

Note that, under 16-bit calculation accuracy,

$$\frac{\tau}{nn} < \frac{1}{2^{n_{16}}\lambda_H} \approx \frac{0.0376434}{|\lambda_H|} < 0.0376434, \tag{60}$$

and the algebraic accuracy of the Gauss three-point integral formula is 5. Using the Gauss three-point integral formula,

$$\frac{1}{2}\left(\frac{\tau}{nn}\right)^{l+1} \sum_{i=1}^{3} w_i \left(\frac{1+b_i}{2}\right)^l F_H(k\tau - \frac{(j+1/2)\tau}{nn} - \frac{b_i\tau}{2nn}), \tag{61}$$

where $w_1 = w_3 = \frac{5}{9}, w_2 = \frac{8}{9}, b_1 = -\frac{\sqrt{15}}{5}, b_2 = 0, b_3 = \frac{\sqrt{15}}{5}$, to approximate (58) with high accuracy. So, (57) can be replaced by the following algorithm.

Algorithm 1. *Numerical solution algorithm of initial value problem (3): for* $nn = 1, 2, \cdots$

$$U_k = TU_{k-1} + \frac{\tau}{2nn} \sum_{j=0}^{nn-1} T_{nn,j} \sum_{i=1}^{3} w_i e^{H\frac{(1+b_i)\tau}{2nn}} F_H\left(k\tau - \frac{(2j+1+b_i)\tau}{2nn}\right), k = 1, 2, \cdots, \tag{62}$$

where

$$e^{H\frac{(1+b_i)\tau}{2nn}} \approx \frac{w_i\tau}{2nn}\left(I + \sum_{l=1}^{m} \frac{1}{l!}\left(\frac{(1+b_i)\tau}{2nn}\right)^l H^l\right). \tag{63}$$

In particular, if we let

$$nn = \begin{cases} [2^{n_p}\lambda_H\tau]+1 & 2^{n_p}\lambda_H\tau > 1 \\ 1 & 2^{n_p}\lambda_H\tau \le 1 \end{cases}, \tag{64}$$

then $N_p = 0, m = m_p = \left[\sqrt{\frac{9p\ln 10}{8\ln 2}}\right]$,

$$W_{i,H} \approx \frac{w_i\tau}{2nn}\left(I + \sum_{l=1}^{m_p}\frac{1}{l!}\left(\frac{(1+b_i)\tau}{2nn}\right)^l H^l\right). \tag{65}$$

At this time, the theoretical calculation accuracy of (62) is p.

Remark 3. *Note that,* $W_{i,H}(i = 1, 2, 3)$ *in (63) and (65) only needs to be calculated once, and is independent of* $F(t)$*, which is not only efficient but also easy to program.*

Now let us consider the numerical solution of the initial value problem (8). If it is converted to the initial value problem (3), then

$$\begin{pmatrix} x(t) \\ x'(t) \end{pmatrix} = U, H = \begin{pmatrix} 0 & I_{3\times 3} \\ -K & 0 \end{pmatrix}, F_H(t) = \begin{pmatrix} 0 \\ F(t) \end{pmatrix}, U_0 = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}. \tag{66}$$

By

$$W_{i,H} = e^{\tau H} \approx \begin{pmatrix} \cos_p(\tau A) & \tau\sin_{1,p}(\tau A) \\ -K\tau\sin_{1,p}(\tau A) & \cos_p(\tau A) \end{pmatrix}, A = \sqrt{K}, \tag{67}$$

where

$$\sin_1 A = \sum_{j=0}^{\infty}\frac{(-1)^j}{(2j+1)!}A^{2j} = \sum_{j=0}^{\infty}\frac{(-1)^j}{(2j+1)!}K^j, \tag{68}$$

and (68), we obtain the following algorithm.

Algorithm 2.

$$\begin{array}{l} \Delta t = \tau/3^N; T_\alpha = I; zz = I; A_2 = (A\Delta t)\times(A\Delta t); \\ for(i = 1, i \le m, i++)\,(zz = -\frac{A_2}{(2i)(2i+1)}\times zz; T_\alpha = T_\alpha + zz), \end{array} \tag{69}$$

and

$$\begin{array}{l} for(i = 0, i < N, i++) \\ (zz = T_\alpha \times T_\alpha, zz = zz \times A_2, A_2 = 9A_2, T_\alpha = T_\alpha - \frac{4}{3}T_\alpha \times zz)\,, \end{array} \tag{70}$$

where m_p *and* N_p *are determined using (31).*

Algorithm 3. *Numerical solution algorithm of initial value problem (8):*

$$\begin{array}{l} x_k = (\cos(\tau A)x_{k-1} + \tau\sin_1(\tau A)y_{k-1}) \\ \quad + \frac{\tau}{2nn}\sum\limits_{j=0}^{nn-1}\sum_{i=1}^{3}\frac{w_i(2j+1+b_i)\tau}{2nn}\sin_1\left(\frac{(2j+1+b_i)\tau}{2nn}A\right)F(k\tau - \frac{(2j+1+b_i)\tau}{2nn}) \\ y_k = -A\sin(\tau A)x_{k-1} + \cos(\tau A)y_{k-1} \\ \quad + \frac{\tau}{2nn}\sum\limits_{j=0}^{nn-1}\sum_{i=1}^{3}w_i\cos\left(\frac{(2j+1+b_i)\tau}{2nn}A\right)F(k\tau - \frac{(2j+1+b_i)\tau}{2nn}) \end{array} \tag{71}$$

for $nn = 1, 2, \cdots$, where $A = \sqrt{K}$. In particular, if nn is determined using (64), the theoretical accuracy of this algorithm is p and

$$\begin{aligned} x_k &= \cos(\tau A)x_{k-1} + \tau \sin_1(\tau A)y_{k-1} + \tfrac{\tau}{2nn}\sum_{j=0}^{nn-1}\sum_{i=1}^{3} w_i TW_{j,i}^{\sin} F(k\tau - \tfrac{(2j+1+b_i)\tau}{2nn}), \\ y_k &= \cos(\tau A)y_{k-1} - A\sin(\tau A)x_{k-1} + \tfrac{\tau}{2nn}\sum_{j=0}^{nn-1}\sum_{i=1}^{3} w_i TW_{j,i}^{\cos} F(k\tau - \tfrac{(2j+1+b_i)\tau}{2nn}), \\ x_0 &= x(0), y_0 = x'(0), \end{aligned} \tag{72}$$

where

$$\begin{aligned} TW_{j,i}^{\sin} &= \tfrac{(1+b_i)\tau}{2nn}\cos\Big(\tfrac{j\tau}{nn}A\Big)\sin_{1,m_p}\Big(\tfrac{(1+b_i)\tau}{2nn}A\Big) \\ &\quad + \tfrac{j\tau}{nn}\sin_1\Big(\tfrac{j\tau}{nn}A\Big)\cos_{m_p}\Big(\tfrac{(1+b_i)\tau}{2nn}A\Big), \\ TW_{j,i}^{\cos} &= \cos\Big(\tfrac{j\tau}{nn}A\Big)\cos_{m_p}\Big(\tfrac{(1+b_i)\tau}{2nn}A\Big) \\ &\quad - \sin\Big(\tfrac{j\tau}{nn}A\Big)\sin_{m_p}\Big(\tfrac{(1+b_i)\tau}{2nn}A\Big), \end{aligned} \tag{73}$$

and

$$\begin{aligned} \sin_{m_p}(sA) &= A\sum_{l=0}^{m_p}\tfrac{(-1)^l s^{2l+1}K^l}{(2l+1)!}, \\ \sin_{1,m_p}(sA) &= \sum_{l=0}^{m_p}\tfrac{(-1)^l s^{2l}K^l}{(2l+1)!}, \\ \cos_{m_p}(sA) &= I + \sum_{l=1}^{m_p+1}\tfrac{(-1)^l s^{2l}K^l}{(2l)!}. \end{aligned} \tag{74}$$

If A is reversible, then

$$\begin{aligned} x_k &= \cos(\tau A)x_{k-1} + A^{-1}\sin(\tau A)y_{k-1} \\ &\quad + \tfrac{\tau}{2nn}A^{-1}\sum_{j=0}^{nn-1}\textstyle\sum_{i=1}^{3} w_i \sin\Big(\tfrac{(2j+1+b_i)\tau}{2nn}A\Big)F(k\tau - \tfrac{(2j+1+b_i)\tau}{2nn}) \\ y_k &= -A\sin(\tau A)x_{k-1} + \cos(\tau A)y_{k-1} \\ &\quad + \tfrac{\tau}{2nn}\sum_{j=0}^{nn-1}\textstyle\sum_{i=1}^{3} w_i \cos\Big(\tfrac{(2j+1+b_i)\tau}{2nn}A\Big)F(k\tau - \tfrac{(2j+1+b_i)\tau}{2nn}). \end{aligned} \tag{75}$$

Therefore, (75) can be written in the form of complex function as follows:

$$z_k = e^{-i\tau A}z_{k-1} + \tfrac{i\tau}{2nn}A^{-1}\sum_{j=0}^{nn-1} e^{-\frac{j\tau i}{nn}A}\textstyle\sum_{l=1}^{3} w_l e^{-\frac{(1+b_l)\tau i}{2nn}A}F\Big(k\tau - \tfrac{(2j+1+b_l)\tau}{2nn}\Big), \tag{76}$$

or

$$Z_k = e^{-i\tau A}Z_{k-1} + \tfrac{\tau}{2nn}\sum_{j=0}^{nn-1} e^{-\frac{j\tau i}{nn}A}\textstyle\sum_{l=1}^{3} w_l e^{-\frac{(1+b_l)\tau i}{2nn}A}F\Big(k\tau - \tfrac{(2j+1+b_l)\tau}{2nn}\Big), \tag{77}$$

where $i = \sqrt{-1}$,

$$\begin{aligned} z_k &= x_k + iA^{-1}y_k, x_k = Rez_k, y_k = AImz_k, \\ Z_k &= y_k - iAx_k, y_k = ReZ_k, x_k = -A^{-1}ImZ_k. \end{aligned} \tag{78}$$

We first provide an examples to illustrate the effectiveness of Algorithms 1. Although we skip examples of Algorithm 2, we then include an example for the effectiveness of Algorithm 3.

Example 1. *In (8), let*

$$K = \begin{pmatrix} 1 & -1 & \\ -1 & \frac{5}{2} & \\ & 1 & 4 \end{pmatrix}, F(t) = \begin{pmatrix} \sin t \\ \frac{1}{2}\sin\ t \\ \cos\frac{\sqrt{2}t}{2} - \cos\Big(\sqrt{3}t\Big) + \sin t \end{pmatrix}, \tag{79}$$
$$x_0 = \begin{pmatrix} \frac{5}{2} & 0 & -2 \end{pmatrix}^T, x_1 = \begin{pmatrix} 1 & 1 & 2 \end{pmatrix}^T.$$

The analytic solution of the initial value problem (8) is

$$x(t) = \begin{pmatrix} 2\cos\frac{\sqrt{2}t}{2} + \frac{1}{2}\cos\left(\sqrt{3}t\right) + \sin t \\ \cos\frac{\sqrt{2}t}{2} - \cos\left(\sqrt{3}t\right) + \sin t \\ \sin(2t) - 2\cos(2t) \end{pmatrix}. \tag{80}$$

Let $err_{i,k} = \left\|x(k\tau) - x_{i,k}\right\|_\infty i = 1,2$, where $x_{1,k}, x_{2,k}(k = 0,1,2,\cdots,N)$ are the numerical solutions of the initial value problem (8) determined by Algorithm 1 and Algorithm 3, respectively. $err_{sym,k} = \left\|x(k\tau) - x_{sym}(k\tau)\right\|_\infty$, where $x_{sym}(t)$ is the numerical solution obtained by calling the function *NDSolve* in Mathematica. The numerical results of $err_{1,k}, err_{2,k}, err_{sym,k}[0,T]$ $(k = 0,1,2,\cdots,N\tau = T)$ are as follows.

It can be seen from Figure 7 that the calculation accuracy of Algorithm 1 is higher than that of Algorithm 3 but the calculation speed of Algorithm 3 is three times that of Algorithm 1. However, the error of numerical results of Mathematica internal functions is far greater than the error of Algorithms 1 and 3. In particular, the collapse is calculated at $t = 667.787$. After this point, the error increases exponentially with time t. However, Algorithms 1 and 3 are quite stable. For example, the calculation error in $[0, 8000]$ is still very small.

In Algorithms 1 and 3, let $nn = \mu nn_A, \mu = 0.5, 0.25, 0.125$, where nn_A is calculated according to (64). The numerical results are as follows.

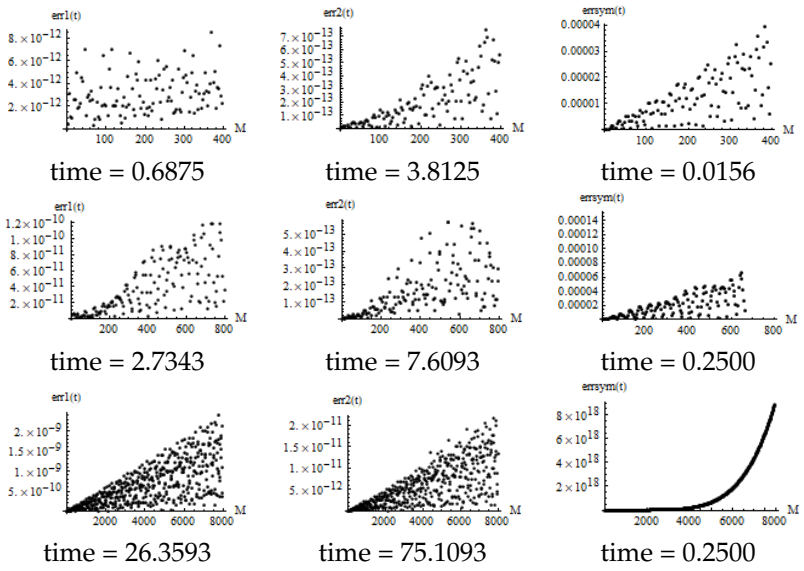


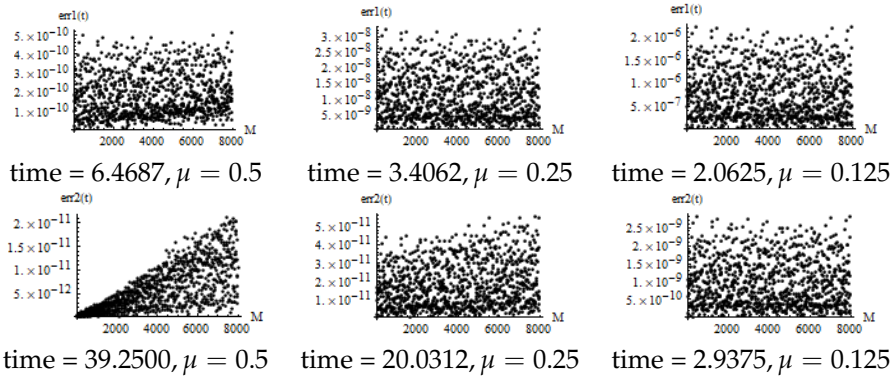
Figure 7. Error of numerical solution (8) for several algorithms (*nm* is calculated according to (64)).

Compared with Figures 7 and 8, when $\mu = 0.5$, the calculation accuracy is not reduced by much, and the calculation speed is almost doubled. For $\mu = 0.125$ Algorithms 1 and 3, the calculation accuracy is still high, and the calculation speed is 10 times higher than that in Figure 8.

Example 2. *In (8), let* $K = (k_{i,j})_{n\times n}$ *be a tridiagonal matrix.* $F(t) = (f_1(t), f_2(t), \cdots, f_n(t))^T$, $f_i(t) = A_i cos(\omega_i t + \varphi_i)$ $(i = 1,2,\cdots,n)$, $x_0 = (x_{0,1}, x_{0,2}, \cdots, x_{0,n})^T$, $x_1 = (x_{1,1}, x_{1,2}, \cdots, x_{1,n})^T$, *and they are randomly generated and satisfy the following conditions:*

$$\begin{gathered} |k_{i,i} - 9| \le 1, \left|k_{i,i+1} - \tfrac{3}{2}\right| \le \tfrac{1}{2}, \left|k_{i+1,i} - \tfrac{3}{2}\right| \le \tfrac{1}{2}, \\ \left|A_i - \tfrac{3}{2}\right| \le \tfrac{1}{2}, \left|\omega_i - \tfrac{5}{2}\right| \le \tfrac{1}{2}, 0 \le \varphi_i \le \tfrac{\pi}{2}, \\ |x_{0,i}| \le 1, |x_{1,i}| \le 1 (i = 1,2,\cdots,n). \end{gathered}$$

For $n = 50, 100, 200,$ and 400, the following numerical results are provided by Algorithms 1 and 3.

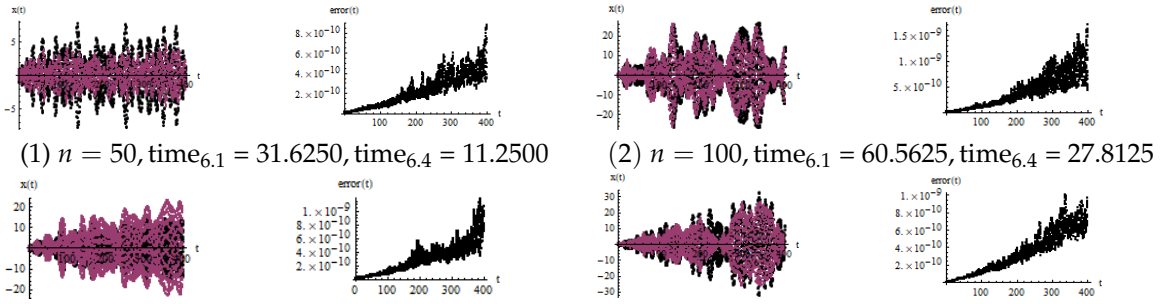


time = 6.4687, $\mu = 0.5$ time = 3.4062, $\mu = 0.25$ time = 2.0625, $\mu = 0.125$

time = 39.2500, $\mu = 0.5$ time = 20.0312, $\mu = 0.25$ time = 2.9375, $\mu = 0.125$

Figure 8. Error of numerical solution (8) for several algorithms.

In Figure 9, (1)–(4), the left is the first two components of the numerical solution, and the right is the difference between the numerical solutions obtained by the two algorithms. It is noted that sometimes ω_i $(i = 1, 2, \cdots, n)$ is close to the frequency of the solution, and flutter or resonance may occur. However, the values obtained by the two algorithms are still very close.



(1) $n = 50$, $\text{time}_{6.1} = 31.6250$, $\text{time}_{6.4} = 11.2500$ (2) $n = 100$, $\text{time}_{6.1} = 60.5625$, $\text{time}_{6.4} = 27.8125$

(3) $n = 200$, $\text{time}_{6.1} = 137.5625$, $\text{time}_{6.4} = 57.6875$ (4) $n = 100$, $\text{time}_{6.1} = 281.3125$, $\text{time}_{6.4} = 151.2187$

Figure 9. Numerical solutions and errors of two algorithms.

7. Conclusions

In this paper, we established algorithms for the computation of matrix exponential functions, matrix trigonometric functions, and matrix hyperbolic functions. Our numerical results show that these algorithms are faster with higher accuracy compared to existing algorithms and the system algorithms in Mathematica. In the series expansion of matrix function, the number of expanded items m_p is only related to the requirement of calculation accuracy, and has nothing to do with the matrix itself. However, the number of times in the iterative process N_p is not only related to the matrix itself but also closely related to t. When $|3^{n_p}\lambda_A t| \leq 1 (|2^{n_p}\lambda_A t| \leq 1)$, $N_p = 0$, the calculation of the matrix function becomes an algebraic operation, so the integral in (9)–(12) becomes an easy integral to compute. In brief, our algorithm allows users to set a required level of accuracy and then use it to choose appropriate values for parameters m and N to minimize computation time. In this manner, both accuracy and efficiency are addressed simultaneously.

Author Contributions: Conceptulation, H.Q. and Y.L.; mehtodology, H.Q. and Y.L.; software, H.Q.; validation, H.Q. and Y.L.; formal analysis, H.Q. and Y.L.; investigation, H.Q. and Y.L.; visualization, H.Q.; supervision, H.Q.; project administration, Y.L. All authors have read and agreed to the publication version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

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

References

1. Ruiz, P.; Sastre, J.; Ibáñeza, J.; Defez, E. High performance computing of the matrix exponential. *J. Comput. Applied Math.* **2015**, *291*, 370–379. [CrossRef]
2. Molert, C.; Loans, C.V. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Rev. Soc. Ind. Applied Math.* **1978**, *20*, 801–836.
3. Higham, N.J. *Functions of Matrices: Theory and Computation*; SIAM: Philadelphia, PA, USA, 2008.
4. Al-Mohy, A.H.; Higham, N.J. Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators. *SIAM J. Sci. Comput.* **2011**, *33*, 488–511. [CrossRef]
5. Higham, N.J. The scaling and squaring method for the matrix exponential resisted. *SIAM J. Matrix Anal. Appl.* **2005**, *26*, 1179–1193. [CrossRef]
6. Bader, P.; Blanes, S.; Casas, F. Computing the Matrix Exponential with an Optimized Taylor Polynomial approximation. *Mathematics* **2019**, *7*, 1174. [CrossRef]
7. Seydaoglu, K.M.; Bader, P.; Blanes, S.; Casas, F. Computing the matrix sine and cosine simultaneously with a reduced number of products. *arXiv* **2020**, arXiv:2010.00465v1.
8. Hargreaves, G.I.; Higham, N.J. Efficient algorithms for the matrix cosine and sine. *Numer. Algorithms* **2005**, *40*, 383–400. [CrossRef]
9. Alonso, P.; Ibáñez, J.; Sastre, J.; Peinado, J.; Defez, E. Efficient and accurate algorithms for computing matrix trigonometric functions. *J. Comput. Appl. Math.* **2017**, *309*, 325–332. [CrossRef]
10. Alonso, P.; Peinado, J.; Ibáñez, J.; Sastre, J.; Defez, E. Computing Matrix Trigonometric Functions with GPUs through Matlab. *J. Supercomput.* **2019**, *75*, 1227–1240. [CrossRef]
11. Al-Mohy, A.H.; Higham, N.J.; Relton, S.D. New Algorithms for Computing the Matrix Sine and Cosine Separately or Simultaneously. *SIAM J. Sci. Comput.* **2015**, *37*, A456–A487. [CrossRef]
12. Serbin, S.M.; Blalock, S.A. An algorithm for computing the matrix cosine. *SIAM J. Sci. Statist. Comput.* **1980**, *1*, 198–204. [CrossRef]
13. Cooper, B.L.; Bingulac, S. Computational improvement in the calculation of the natural log of a square matrix. *Electron. Lett.* **1990**, *26*, 861–862. [CrossRef]
14. Wu, F.; Zhu, L.; Shi, Q. Efficient Computational method for matrix function in dynamic problems. *Acta Mech. Sin.* **2023**, *39*, 522451. [CrossRef]
15. Cardoso, J.R.; Sadeghi, A. A technique for improving the computation of functions of triangular matrices. *Int. J. Comput. Math.* **2022**, *99*, 2449–2465. [CrossRef]
16. Jiang, Y.; He, C.; Qin, H. An accelerated algorithm for high precision and fast calculation of basic elementary functions. *Math. Pract. Theory* **2017**, *47*, 238–246.
17. Fasi, M.; Higham, N.J. Multiprecision algorithm for Computing matrix logarithm. *SIAM J. Matrixanal. Appl.* **2018**, *39*, 472–491. [CrossRef]
18. Ibáñez, J.; Sastre, J.; Ruiz, P.; Alonso, J.M.; Defez, E. An Improved Taylor Algorithm for Computing the Matrix Logarithm. *Mathematics* **2021**, *9*, 2018. [CrossRef]
19. Cheng, S.H.; Higham, N.J.; Kenney, C.S.; Laub, A.J. Approximating the Logarithm of a Matrix to Specified Accuracy. *SIAM J. Matrix Anal. Appl.* **2001**, *22*, 1112–1125. [CrossRef]
20. Kenney, C.; Laub, A.J. Condition estimates for matrix functions. *SIAM J. Matrix Anal. Appl.* **1989**, *10*, 191–209. [CrossRef]
21. Kenney, C.; Laub, A.J. Padé error estimates for the logarithm of a matrix. *Internat. J. Control.* **1989**, *50*, 707–730. [CrossRef]
22. Milovanović, G.V.; Rančić, S.L. Some improvements in calculation of the natural log of a square matrix. In Proceedings of the Third International Conference on Functional Analysis and Approximation Theory, Aquafredda di Maratea, Potenza, Italy, 23–28 September 1996; Issue 52, Volume II.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.