AIMS Mathematics

https://www.aimspress.com/journal/Math

AIMS Mathematics, 10(2): 2854–2868.
DOI: 10.3934/math.2025133
Received: 09 November 2024
Revised: 21 January 2025
Accepted: 05 February 2025
Published: 17 February 2025

Research article

The shift-based scaling and recursing algorithm for evaluating the action of the matrix φ-functions

Xianan Lu, Dongping Li[*] and Zhixin Zhao[*]

School of Mathematics, Changchun Normal University, 677 Changji North Road, Changchun, MO 130032, China

* **Correspondence:** Email: lidp@ccsfu.edu.cn; jczzx10@163.com; Tel: +8613844142920.

Abstract: In this paper, we present an efficient method for computing the action of matrix φ-functions. Our approach is based on a scaling and recursing procedure and incorporates a shifting technique as a preprocessing step to enhance efficiency. We conduct a forward error analysis to determine the optimal scaling parameter and polynomial degree for achieving the desired accuracy. Numerical comparisons with existing algorithms demonstrate that the proposed algorithm performs well in terms of both accuracy and efficiency.

Keywords: φ-functions; scaling and recursing method; shifting technique; Taylor series; forward error analysis; exponential integrators
Mathematics Subject Classification: 65L05, 65F10, 65F30

1. Introduction

In this paper, we develop efficient numerical methods for evaluating the product of exponential-like matrix functions with a vector

$$\varphi_l(A)b, \quad l \in \mathbb{N}, \tag{1.1}$$

where $A \in \mathbb{C}^{N\times N}$ is a large, sparse matrix; $b \in \mathbb{C}^N$ is a vector; and

$$\varphi_0(z) = e^z, \quad \varphi_l(z) = \int_0^1 e^{(1-t)z}\frac{t^{l-1}}{(l-1)!}dt, \quad l = 1,2,\dots. \tag{1.2}$$

These functions satisfy the recursive relation:

$$\varphi_{k-1}(z) = z\varphi_k(z) + \frac{1}{(k-1)!}, \quad k = 1,2,\dots. \tag{1.3}$$

This problem plays a key role in the implementation of exponential integrators, a competitive tool for solving the solution of large stiff ordinary differential equations (ODEs), see [1, 2]. Exponential integrators employ exponential-like matrix functions, commonly referred to as φ-functions, within their formulation. The fast and accurate computation of the action of φ-functions directly determines the efficiency and stability of exponential integrators.

In many practical applications, the matrix A is large and sparse making it infeasible to compute $\varphi_l(A)$ explicitly by direct methods as in [3–12] and then multiply by b. Instead, one seeks to approximate $\varphi_l(A)b$ using a numerical method based on matrix-vector products. In recent years, several numerical algorithms following this approach have emerged, with the action of matrix exponential on a vector being the most widely studied case, see [13–16]. Since the φ-functions can be reformulated in terms of an augmented matrix exponential (see [13, 14, 17]), the action of φ-functions can be computed by applying the matrix exponential to a vector. The `expmv` algorithm by Al-Mohy and Higham [13] is considered one of the most effective methods for this purpose. Some direct approaches have also been developed for general φ-functions. Typical methods include Krylov methods [14, 17–20], the scaling and recursing method based on a truncated Taylor series [21].

In this paper, we adopt a different strategy to implement the scaling and recursing method, fully integrating the shifting technique to enhance the algorithm's computational efficiency. We provide a forward error analysis that facilitates the selection of the scaling parameter s and the Taylor degree m to achieve a prescribed accuracy while minimizing the algorithm's computational cost.

The structure of this article is outlined as follows. Section 2 presents a brief overview of existing scaling and recursing methods. Section 3 introduces a shift-based scaling and recursing method, including forward error analysis and the selection of the scaling parameter s and the Taylor degree m. Section 4 demonstrates the algorithm's effectiveness through numerical experiments. Finally, Section 5 concludes with a summary of the main findings of the paper.

2. The existing scaling and recursing method for $\varphi_l(A)b$

In this section, we briefly review the scaling and recursing method proposed in [21], which is based on the recursive formula:

$$\varphi_l(kY)b = (1-\frac{1}{k})^l \varphi_0(Y)\varphi_l((k-1)Y)b + \sum_{j=1}^{l} f_{k,j}\varphi_j(Y)b, \;\; k = 2,3,\ldots,s, \tag{2.1}$$

where $f_{k,j} = (1-\frac{1}{k})^{l-j}(\frac{1}{k})^j \frac{1}{(l-j)!}$ and $Y = A/s$, with s being a nonnegative integer. If $\|Y\|$ sufficiently small, $\varphi_l(Y)$ can be well-approximated by its truncated Taylor series of degree m:

$$\varphi_l(Y) \approx T_{l,m}(Y) = \sum_{k=0}^{m} \frac{Y^k}{(l+k)!}.$$

By replacing $\varphi_l(Y)b$ by $b_l := T_{l,m}(Y)b$ and using the relation (1.3), the other terms $\varphi_k(Y)b$ for $k < l$ are sequentially approximated as follows:

$$b_k = Yb_{k+1} + \frac{1}{k!}b, \;\; k = l-1, l-2, \cdots, 1. \tag{2.2}$$

Essentially, b_k represents the product of the truncated Taylor series of degree $m+l-k$ for $\varphi_k(Y)$ and b. To explore the sparsity of Y, the vector $b_l = T_{l,m}(Y)b$ should be evaluated using matrix-vector products involving Y. Algorithm 1 presents a procedure for this task.

Algorithm 1 The algorithm computes $b_l = T_{l,m}(Y)b$.

```
Require: $Y \in \mathbb{C}^{N\times N}$, $b \in \mathbb{C}^N$, $l$ and $m$.
1: $b_l = b/l!$;
2: for $k = 1 : m$ do
3:     Compute $b = Yb/(l+k)$;
4:     Compute $b_l = b_l + b$;
5: end for
Ensure: $b_l$.
```

Once the vectors $b_1, b_2, \ldots, b_l$ are computed, the approximation ψ_s to $\varphi_l(A)b$ is obtain by applying the recursion:

$$\psi_k = (1-\frac{1}{k})^l T_{0,m+l}(Y)\psi_{k-1} + c_k, \ \ k = 2,3,\cdots,s. \tag{2.3}$$

Here, $\psi_1 := b_l$, $c_k := \sum_{j=1}^{l} f_{k,j}b_j$, and $T_{0,m+l}(Y)$ is the truncated Taylor approximation of degree $m+l$ for e^Y. The total computational cost of the method is $s(m+l)-1$ matrix-vector products for $s>1$.

In [21], the authors partially employ the shifting technique as a preprocessing step to enhance the method's performance. The shifting approach can often reduce the norm of matrix A, thereby yielding lower Taylor degree m or smaller scaling factor s. For the matrix exponential, such a shift can be incorporated using the property $e^A = e^{\mu}e^{A-\mu I}$, where μ is a given shift. However, this property is unavailable for more general φ-functions. Consequently, the shift-based approximation to $\varphi_l(A)b$ becomes

$$\psi_k = (1-\frac{1}{k})^l e^{\mu/s} T_{0,m+l}(Y-\mu I/s)\psi_{k-1} + c_k, \ \ k = 2,3,\cdots,s. \tag{2.4}$$

Due to the lack of a shifted property for $\varphi_l(Y)$, they had to use a truncated Taylor series with a larger degree $\widehat{m}$ than m to approximate $\varphi_l(Y)$ and thereby compensate for the loss of accuracy. If $\widehat{m}$ is significantly greater than m, then this approach can result in additional matrix-vector products even leading to algorithmic instability.

In this paper, we propose an alternative strategy for evaluating $\varphi_l(A)b$. Our approach emphasizes a novel method for assessing $\varphi_k(Y)$ for $k = 1,2,\ldots,l$. A significant advantage of this new method is its ability to fully incorporate the shifting technique into the evaluation process.

3. The shift-based scaling and recursing method for $\varphi_l(A)b$

We now derive a new algorithm based on the scaling and recursing formula (2.1). Let $Y = A/s$ and $T_m(Y)$ denote the truncated Taylor series of order m for e^Y, that is,

$$T_m(Y) = \sum_{k=0}^{m} \frac{Y^k}{k!}. \tag{3.1}$$

The error in $T_m(Y)$ is given by

$$h_m(Y) := e^Y - T_m(Y) = \sum_{k=m+1}^{\infty} \frac{Y^k}{k!}. \tag{3.2}$$

If $\mu \in \mathbb{R}$ is a chosen shift satisfying $\|A - \mu I\| \le \|A\|$, and let $\nu = \mu/s$, then the matrix exponential e^Y can also be approximated via the matrix $Y - \nu I$ as follows:

$$e^{\nu} T_m(Y - \nu I). \tag{3.3}$$

From definition (1.2), we can approximate $\varphi_j(Y)$ by

$$\tilde{\varphi}_j(Y) = \int_0^1 e^{(1-t)\nu} T_m\left((1-t)(Y - \nu I)\right) \frac{t^{j-1}}{(j-1)!} dt. \tag{3.4}$$

Let

$$\widetilde{h}_m(Y) = \sum_{k=m+1}^{\infty} \frac{1}{k!} \|Y^k\|. \tag{3.5}$$

Then, the associated error satisfies

$$\begin{aligned} \|\varphi_j(Y) - \tilde{\varphi}_j(Y)\| &= \int_0^1 e^{(1-t)\nu} \|e^{(1-t)(Y-\nu I)} - T_m((1-t)(Y-\nu I))\| \frac{t^{j-1}}{(j-1)!} dt \\ &= \int_0^1 e^{(1-t)\nu} \|h_m\left((1-t)(Y-\nu I)\right)\| \frac{t^{j-1}}{(j-1)!} dt \\ &\le \frac{1}{(j-1)!} \widetilde{h}_m(Y - \nu I) \int_0^1 e^{(1-t)\nu} dt \\ &\le \frac{1}{(j-1)!} \varphi_1(\nu) \widetilde{h}_m(Y - \nu I). \end{aligned} \tag{3.6}$$

By replacing the functions $\varphi_j(Y)$ in (2.1) with the approximations $\tilde{\varphi}_j(Y)$ for $j = 0, 1 \ldots, l$, respectively, we obtain the shift-based scheme for computing $\varphi_l(A)b$ by

$$\phi_k = (1 - \frac{1}{k})^l e^{\nu} T_m(Y - \nu I) \phi_{k-1} + \tilde{c}_k, \;\; k = 2, 3, \cdots, s, \tag{3.7}$$

where $\tilde{c}_k = \sum_{j=1}^{l} f_{k,j} \tilde{\varphi}_j(Y) b$ and $\phi_1 = \tilde{\varphi}_l(Y) b$. If $\nu = 0$, then scheme (3.7) becomes to a case without shifting.

The following result shows how the final error in the scheme (3.7) is affected by the initial errors in approximating $\varphi_j(Y)$ for $j = 0, 1, \ldots, l$.

Theorem 1. *Let $A \in \mathbb{C}^{N \times N}$, $\nu \in \mathbb{R}$, and $Y = A/s$ for a nonnegative integer s, and assume that $\widetilde{h}_m(Y - \nu I) \le \varepsilon$. Then, the error E_s of approximating $\varphi_l(A)b$ by ϕ_s, generated by the recursion (3.7) in exact arithmetic, satisfies*

$$\|E_s\| \le C_0 \|b\| (1 + e^{\nu} \varepsilon)^{s-1} \left(C_1 e^{\nu} + \frac{1}{(l-1)!} \varphi_1(\nu) \right) (\frac{s}{l+1} + 1) \varepsilon, \tag{3.8}$$

where $C_0 = \max_{1 \le k \le s} \{\|e^{kY}\|\}$, $C_1 = \max_{1 \le k \le s} \{\|\varphi_l(kY)\|\}$.

Proof. Let $y_k = \varphi_l(kY)b$ and $c_k = \sum_{j=1}^{l} f_{k,j}\varphi_j(Y)b$. We can rewrite (2.1) as

$$y_k = (1-\frac{1}{k})^l \varphi_0(Y) y_{k-1} + c_k, \;\; k = 2,3,\ldots,s. \tag{3.9}$$

Let $E_k = y_k - \phi_k$ and $e_k = c_k - \tilde{c}_k$ denote the differences between the numerical and exact solutions. By subtracting (3.7) from the exact scheme, we obtain the error recursion

$$\begin{aligned} E_k =&(1-\frac{1}{k})^l e^{\nu}\left(\varphi_0(Y-\nu I)y_{k-1} - T_m(Y-\nu I)\phi_{k-1}\right) + e_k \\ =&(1-\frac{1}{k})^l e^{\nu}\left(\varphi_0(Y-\nu I)y_{k-1} - T_m(Y-\nu I)y_{k-1} + T_m(Y-\nu I)y_{k-1} - T_m(Y-\nu I)\phi_{k-1}\right) + e_k \\ =&(1-\frac{1}{k})^l e^{\nu}\left(h_m(Y-\nu I)y_{k-1} + T_m(Y-\nu I)E_{k-1}\right) + e_k. \end{aligned} \tag{3.10}$$

This recursion yields

$$E_s = \left(\frac{1}{s}\right)^l e^{(s-1)\nu} T_m(Y-\nu I)^{s-1} E_1 + \sum_{i=2}^{s} e^{(s-i)\nu} T_m(Y-\nu I)^{s-i} \left(\left(\frac{i-1}{s}\right)^l e^{\nu} h_m(Y-\nu I) y_{i-1} + \left(\frac{i}{s}\right)^l e_i\right). \tag{3.11}$$

Taking norm and utilizing the following inequalities

$$\begin{aligned} \|e^{k\nu}T_m(Y-\nu I)^k\| =&\|e^{k\nu}\left(e^{Y-\nu I} - h_m(Y-\nu I)\right)^k\| \\ \le& \sum_{i=0}^{k}\binom{k}{i}\|e^{(k-i)Y}\| \cdot \|e^{\nu}h_m(Y-\nu I)\|^i \\ \le& C_0 \sum_{i=0}^{k}\binom{k}{i}\left(e^{\nu}\widetilde{h}_m(Y-\nu I)\right)^i \\ \le& C_0(1+e^{\nu}\widetilde{h}_m(Y-\nu I))^k, \end{aligned} \tag{3.12}$$

$$\begin{aligned} \|E_1\| =& \|\varphi_l(Y)b - \tilde{\varphi}_l(Y)b\| \\ \le& \frac{1}{(l-1)!}\varphi_1(\nu)\widetilde{h}_m(Y-\nu I)\|b\|, \end{aligned} \tag{3.13}$$

and

$$\begin{aligned} \|e_i\| \le& \sum_{j=1}^{l}\frac{f_{i,j}}{(j-1)!}\varphi_1(\nu)\widetilde{h}_m(Y-\nu I)\|b\| \\ \le& \frac{1}{(l-1)!}\varphi_1(\nu)\widetilde{h}_m(Y-\nu I)\|b\|, \end{aligned} \tag{3.14}$$

we have

$$\begin{aligned} \|E_s\| \le& C_0\|b\|(1+e^{\nu}\widetilde{h}_m(Y-\nu I))^{s-1}\left(C_1 e^{\nu}\sum_{i=1}^{s-1}\left(\frac{i}{s}\right)^l + \frac{1}{(l-1)!}\varphi_1(\nu)\sum_{i=1}^{s}\left(\frac{i}{s}\right)^l\right)\widetilde{h}_m(Y-\nu I) \\ \le& C_0\|b\|(1+e^{\nu}\widetilde{h}_m(Y-\nu I))^{s-1}\left(C_1 e^{\nu} + \frac{1}{(l-1)!}\varphi_1(\nu)\right)\sum_{i=1}^{s}\left(\frac{i}{s}\right)^l\widetilde{h}_m(Y-\nu I) \\ \le& C_0\|b\|(1+e^{\nu}\widetilde{h}_m(Y-\nu I))^{s-1}\left(C_1 e^{\nu} + \frac{1}{(l-1)!}\varphi_1(\nu)\right)(\frac{s}{l+1}+1)\widetilde{h}_m(Y-\nu I). \end{aligned} \tag{3.15}$$

The assumption on $\widetilde{h}_m(Y-\nu I)$ concludes the proof. $\square$

In particular, if A is normal with negative eigenvalues λ_i, $1 \le i \le N$, then under the 2-norm, we have

$$C_0 = \max_{1\le k\le s}\{\|e^{kY}\|_2\} = \max\{e^{k\lambda_i/s}, 1\le k\le s, 1\le i\le N\} < 1,$$

$$C_1 = \max_{1\le k\le s}\{\|\varphi_l(kY)\|_2\} = \max\{\varphi_l(k\lambda_i/s), 1\le k\le s, 1\le i\le N\} < \frac{1}{(l-1)!},$$

and (3.8) yields

$$\|E_s\|_2 \le \frac{1}{(l-1)!}\|b\|_2(1+e^{\nu}\varepsilon)^{s-1}\,(e^{\nu}+\varphi_1(\nu))\,(\frac{s}{l+1}+1)\varepsilon.$$

Theorem 1 shows that if the initial error bound $\widetilde{h}_m(Y-\nu I)$ is sufficiently small, then the final error will also remain small.

3.1. Implementation issues of the method

In this subsection, we present some implementation details of the scheme in (3.7). Let $f_k = (f_{k,1}, f_{k,2}, \dots, f_{k,l})^T$ denote the vector with elements $f_{k,j}$ introduced in (2.1). We define

$$R = [b, (Y-\nu I)b, \frac{1}{2!}(Y-\nu I)^2 b, \cdots, \frac{1}{m!}(Y-\nu I)^m b] \in \mathbb{C}^{N\times(m+1)}, \tag{3.16}$$

and $W = (w_{i,j}) \in \mathbb{C}^{(m+1)\times l}$ with elements

$$w_{i,j} = \frac{1}{j!}\int_0^1 (1-t)^i e^{(1-t)\nu} t^j dt,\ i = 0,1,\dots,m,\ j = 0,1,\dots,l-1. \tag{3.17}$$

Let

$$F = R\cdot W. \tag{3.18}$$

Then, the initial value ϕ_1 and the vector $\tilde{c}_k$ in (3.7) can be equivalently expressed as

$$\phi_1 = F(:,l),\ \ \tilde{c}_k = F\cdot f_k, \tag{3.19}$$

where $F(:,l)$ denotes the l-th column of F.

Another point to consider is the computation of the matrix W. For $\nu = 0$, the elements $w_{i,j}$ can be conveniently computed using the MATLAB function `beta`. However, for $\nu \neq 0$, we employ a recursive method in combination with a quadrature rule, which is typically more efficient than relying solely on the quadrature rule.

Lemma 1. *Let $w_{i,j}$ be defined as in (3.17), and let ν be a non-zero constant. Then, the following identities hold:*
(a) $w_{i,0} = \frac{1}{\nu}(e^{\nu} - i w_{i-1,0}),\ \ i = 1,2,\dots,m;$
(b) $w_{0,j} = \frac{1}{\nu}(w_{0,j-1} - \frac{1}{j!}),\ \ j = 1,2,\dots,l-1;$
(c) $w_{i,j} = \frac{1}{\nu}(w_{i,j-1} - i w_{i-1,j}),\ \ i = 1,2,\cdots,m,\ \ j = 1,2,\cdots,l-1;$
(d) $w_{i,j} = \frac{1}{i+1}(w_{i+1,j-1} - \nu w_{i+1,j}),\ \ i = 1,2,\cdots,m,\ \ j = 1,2,\cdots,l-1.$

Proof. The proof follows from the integration by parts formula. For simplicity, we provide the proofs for cases (c) and (d), omitting those for cases (a) and (b). For case (c), the integration by parts formula is expressed as follows:

$$\begin{aligned} w_{i,j} =& -\frac{1}{j!\nu}\int_0^1 t^j(1-t)^i de^{\nu(1-t)} \\ =& -\frac{1}{j!\nu}\left(t^j(1-t)^i e^{\nu(1-t)}|_0^1 - \int_0^1 e^{\nu t} dt^j(1-t)^i\right) \\ =& \frac{1}{j!\nu}\int_0^1 e^{\nu(1-t)}\left(jt^{j-1}(1-t)^i - it^j(1-t)^{i-1}\right)dt \\ =& \frac{1}{\nu}(w_{i,j-1} - iw_{i-1,j}). \end{aligned} \tag{3.20}$$

On the other hand, applying integration by parts yields

$$\begin{aligned} w_{i,j} =& \frac{1}{j!}\int_0^1 (1-t)^i e^{\nu(1-t)} t^j dt \\ =& \frac{1}{j!(i+1)}\int_0^1 e^{\nu(1-t)} t^j d(1-t)^{i+1} \\ =& \frac{1}{j!(i+1)}\left(e^{\nu(1-t)} t^j (1-t)^{i+1}|_0^1 - \int_0^1 (1-t)^{i+1} de^{\nu(1-t)} t^j\right) \\ =& -\frac{1}{j!(i+1)}\int_0^1 \left(-\nu e^{\nu(1-t)}(1-t)^{i+1}t^j + je^{\nu(1-t)}(1-t)^{i+1}t^{j-1}\right)dt \\ =& \frac{1}{i+1}(w_{i+1,j-1} - \nu w_{i+1,j}). \end{aligned} \tag{3.21}$$

□

Remark 1. *Applying the identities (a), (b), and (c) to calculate the elements of matrix W is theoretically straightforward; however, this approach often proves to be numerically unstable in practice. To address this, we employ identity (d) for these calculations. This requires first computing the elements of the last row of matrix W, which we achieve using a quadrature rule.*

In practice, selecting the shift parameter μ is a nontrivial task. We set the shift $\mu = \mathtt{trace}(A)/N$, as used in [13], which has been empirically shown to typically reduce the norm of the matrix $A - \mu I$. However, if $\|A - \mu I\| \geq \|A\|$, then the shift is unnecessary.

In Algorithm 2, we provide a brief outline of the process for computing $\varphi_l(A)b$. The computational effort of the algorithm primarily focuses on matrix-vector products involving matrix A, requiring a total of $C_m = sm$ matrix-vector products.

3.2. Choice of the Taylor approximation degree m and the scaling parameter s

Algorithm 2 mentions two key parameters: the Taylor degree m and the scaling parameter s, both of which should be selected appropriately. In [21], a quasi-backward error analysis is employed to determine these parameters; however, this strategy is not feasible for the new implementation. We now consider a forward error-based approach.

Algorithm 2 `shift_phimv`: the shift-based scaling and recursing method for approximating $\varphi_l(A)b$.

```
Require: $A \in \mathbb{C}^{N\times N}$, $b \in \mathbb{C}^N$, $l$ and $\mu$.
1: $\mu = \mathtt{trace}(A)/N$;
2: Compute $\tilde{A} = A - \mu I$;
3: if $\|\tilde{A}\| \le \|A\|$, then $A = \tilde{A}$ else $\mu = 0$; end if
4: Select the parameters $m$ and $s$;
5: Compute $Y = A/s$;
6: Compute $R = [b, Yb, \frac{1}{2!}Y^2 b, \cdots, \frac{1}{m!}Y^m b]$;
7: Compute $W = (w_{i,j})$ with $w_{i,j} = \frac{1}{j!}\int_0^1 (1-t)^i e^{(1-t)\mu} t^j dt$ for $i = 0,1,\ldots,m,\ j = 0,1,\ldots,l-1$;
8: Compute $F = R \cdot W$;
9: $\phi_1 := F(:,l)$;
10: if $s = 1$ then $\varphi_l = \phi_1$ return; end if
11: for $k = 2 : s$ do
12:     Compute the vector $f_k = (f_{k,1}, f_{k,2}, \ldots, f_{k,l})^T$ with $f_{k,j} = (1-\frac{1}{k})^{l-j}(\frac{1}{k})^j \frac{1}{(l-j)!}$, $j = 1,2,\ldots,l$;
13:     Compute $\phi_k$ by the recurrence $\phi_k = (1-\frac{1}{k})^l e^{\mu/s} T_m(Y)\phi_{k-1} + F \cdot f_k$ based on matrix-vector products;
14: end for
Ensure: $\varphi_l := \phi_s$.
```

The error bound (3.8) provides support for the use of an absolute error criterion. To achieve high computational accuracy, it is essential to minimize $\widetilde{h}_m(Y - \nu I)$ as much as possible. Let θ_m denote the largest value of θ such that $\widetilde{h}_m(\theta)$ does not exceed the tolerance `Tol`, that is,

$$\theta_m = \max\{\theta \mid \widetilde{h}_m(\theta) \le \mathtt{Tol}\}. \tag{3.22}$$

In practice, the value of θ_m can be evaluated by solving the smallest positive solution of algebraic equation

$$\sum_{k=m+1}^{m+\upsilon} \frac{1}{(k)!}\theta^k = \mathtt{Tol}, \tag{3.23}$$

where υ is a large integer. In Table 1 we list some values of θ_m for $\upsilon = 150$ and $\mathtt{Tol} = 2^{-53}$ using MATLAB symbolic computation.

For a nonnegative integer p, let $\alpha_p(Y - \nu I) = \max\{\|(Y - \nu I)^p\|^{1/p},\ \|(Y - \nu I)^{p+1}\|^{1/(p+1)}\}$ and $\eta_m(Y - \nu I) = \min\{\alpha_p(Y - \nu I) \mid p(p-1) \le m+1\}$. According to Theorem 4.2 of [3], we obtain

$$\widetilde{h}_m(Y - \nu I) = \sum_{k=m+1}^{\infty} \frac{1}{k!}\|(Y - \nu I)^k\| \le \widetilde{h}_m\left(\eta_m(Y - \nu I)\right). \tag{3.24}$$

Choose a scaling parameter s such that

$$\eta_m(Y - \nu I) = \eta_m((A - \mu I)/s) = \eta_m(A - \mu I)/s \le \theta_m. \tag{3.25}$$

Then, we have

$$\widetilde{h}_m(Y - \nu I) \le \mathtt{Tol}. \tag{3.26}$$

Table 1. The values of θ_m satisfying (3.23) for $\upsilon = 150$ and $\texttt{Tol} = 2^{-53}$.

m	θ_m	m	θ_m	m	θ_m	m	θ_m
1	1.49011611568402e-08	14	5.53490515693853e-01	27	3.03035372202815	40	6.56123464544190
2	8.73347022584872e-06	15	6.82758074718948e-01	28	3.27521355907414	41	6.85713423697126
3	2.27195870977283e-04	16	8.24603191638609e-01	29	3.52556376698837	42	7.15562009043849
4	1.67839429827810e-03	17	9.78344888569965e-01	30	3.78106962698314	43	7.45656075832843
5	6.56229738373172e-03	18	1.14329611222606	31	4.04142067325413	44	7.75983325403385
6	1.77645270836847e-02	19	1.31878122619485	32	4.30632888011671	45	8.06532241011747
7	3.81185198063615e-02	20	1.50414732239516	33	4.57552694565151	46	8.37292029066545
8	6.99327848078254e-02	21	1.69877113848913	34	4.84876668177908	47	8.68252565299361
9	1.14831747477397e-01	22	1.90206289621254	35	5.12581751448866	48	8.99404345434753
10	1.73788723247484e-01	23	2.11346801475487	36	5.40646509379029	49	9.30738439960222
11	2.47239754095914e-01	24	2.33246738440124	37	5.69051001024739	50	9.62246452631257
12	3.35213687828615e-01	25	2.55857668841814	38	5.97776661325278	51	9.93920482378810
13	4.37449366712157e-01	26	2.79134511801270	39	6.26806192522873	52	10.2575308831652

For a given m, it is natural to set the scaling parameter s to be

$$s = \max\left(\lceil \eta_m(A - \mu I)/\theta_m \rceil, 1\right). \tag{3.27}$$

Furthermore, one can choose m to minimize the computational cost C_m, that is,

$$m^* = \operatorname{argmin}\{m\lceil \eta_m(A - \mu I)/\theta_m \rceil,\ 0 \le m \le m_{\max}\}, \tag{3.28}$$

where $m_{\max}$ is the maximum permissible value for m. A brief sketch of this process is summarized in Algorithm 3. In this algorithm, the 1-norm is utilized primarily because existing algorithms can effectively estimate the norms of matrix powers using matrix-vector products, as noted in [22].

4. Numerical experiments

In this section, we present two numerical experiments to demonstrate the performance of the new algorithm, referred to as `shift_phimv`. All tests are conducted using MATLAB R2018b on a desktop equipped with an Intel Core i5 processor (1.8 GHz) and 8 GB of RAM. The tests involved the following three existing MATLAB implementations:

- `phipm`: A MATLAB implementation based on Krylov subspace method described in [20].
- `expmv`: A MATLAB implementation based on scaling and squaring method presented in [13].
- `phimv(s)`: A MATLAB implementation based on scaling and recursing method with partial shifting, as described in [21].

The accuracy of the algorithm is evaluated by computing the relative error, defined as

$$\texttt{Error} = \frac{\|y_l - \widehat{y}_l\|_2}{\|y_l\|_2}, \tag{4.1}$$

where y_l and $\widehat{y}_l$ represent the reference solution and the computed solution, respectively. The reference solutions are obtained using `expmv` with maximal precision.

Algorithm 3 select_m_s(A): This code determines m and s for a given A.

```
Require: $A \in \mathbb{C}^{N\times N}$ and $m_{\max}$.
1:  $p_{\max} = \text{argmax}\{p(p-1) \le m_{\max}\}$;
2:  $d_1 = \|A\|_1$;
3:  for $p = 2 : p_{\max}+1$ do
4:      Estimate $d_p = \|A^p\|^{1/p}$;
5:  end for
6:  $\alpha_1 = d_1$;
7:  for $p = 2 : p_{\max}$ do
8:      $\alpha_p = \max(d_p, d_{p+1})$;
9:  end for
10: for $m = 1 : m_{\max}$ do
11:     $\eta_m = \min(\alpha_p :\ p(p-1) \le m+1)$;
12: end for
13: $m = \arg\min\{m\lceil \eta_m/\theta_m \rceil : 1 \le m \le m_{\max}\}$;
14: $s = \max(\lceil \eta_m/\theta_m \rceil, 1)$;
Ensure: $m, s$.
```

Experiment 1. *In this experiment, we aim to demonstrate the advantages of the shifting strategy on four different sparse matrices described below, applied to a vector b with all elements equal to 1. We compute $\varphi_l(A)b$ using* `shift_phimv` *both with and without shifting. The case without shifting is denoted as* `shift_phimv(0)`*. The four matrices considered are described as follows:*

• *The first matrix,* `lesp`*, is a nonsymmetric tridiagonal matrix of order $N = 1{,}000$ with $2{,}998$ nonzero elements, its 1-norm is $3{,}003$, and after shifting, the 1-norm is $1{,}999$. This matrix is generated using the MATLAB command* `gallery`('`lesp`', 1000).

• *The second matrix,* `triw`*, is an upper triangular matrix of order $N = 1{,}000$ with $500{,}500$ nonzero elements. Its 1-norm is $10{,}090$, and after shifting, the 1-norm is $9{,}990$. This matrix is generated using the MATLAB command* $-100*$`gallery`('`triw`', 1000, 0.1).

• *The third matrix,* `wilkinson`*, is a symmetric, tridiagonal matrix of order $N = 3{,}000$ with Wilkinson's eigenvalues and $500{,}500$ nonzero elements. Its 1-norm is $1{,}500.5$, and after shifting, the 1-norm is 751.5. This matrix is generated using the MATLAB command* $-$`wilkinson`(3000).

• *The fourth matrix,* `poisson`*, is a symmetric block tridiagonal matrix of order $N = 10{,}000$ with $49{,}600$ nonzero elements. Its 1-norm is $20{,}000$ and after shifting, the 1-norm is $10{,}000$. This matrix is generated using the MATLAB command* $-2500*$`gallery`('`poisson`', 100).

Table 2 presents the relative errors and execution times (in seconds) for each test. Both methods achieve high computational accuracy; however, the case with shifting is even more accurate than the method without shifting. As indicated in Theorem 1, this is primarily because the shifting method reduces the scaling parameter s, and a smaller s results in lower error.

Furthermore, the shifting case is significantly more efficient than the non-shifting case as it leads to fewer scaling steps and less matrix-vector products, particularly when there is a substantial difference between the matrix norms before and after shifting. For the final two matrices, the 1-norm of the shifted matrix is nearly half that of the original matrix, leading to nearly a twofold decrease in computational time.

Table 2. The relative errors and CPU time when computing $\varphi_l(A)b$ for $l = 1,2,\ldots,8$ of Experiment 1.

l	method	lesp		triw		wilkinson		poisson	
		Error	Time	Error	Time	Error	Time	Error	Time
	`shift_phimv(0)`	1.28e-14	4.04	3.14e-12	3.58	8.56e-12	26.75	6.05e-14	8.52
1	`shift_phimv`	6.39e-15	2.65	9.93e-13	3.35	1.69e-15	13.39	6.87e-14	4.21
	`shift_phimv(0)`	2.33e-14	4.24	4.19e-12	3.73	8.88e-12	26.86	4.30e-13	10.10
2	`shift_phimv`	2.65e-14	2.71	2.04e-12	3.43	9.36e-15	13.35	4.36e-13	4.54
	`shift_phimv(0)`	1.54e-14	3.85	3.48e-12	3.95	8.85e-12	27.15	1.16e-12	10.33
3	`shift_phimv`	1.58e-14	2.64	2.35e-12	3.43	9.20e-15	13.74	1.16e-12	4.70
	`shift_phimv(0)`	5.11e-14	3.91	3.53e-12	3.78	8.47e-12	27.97	5.04e-15	9.65
4	`shift_phimv`	4.82e-14	2.61	1.02e-12	3.46	4.26e-14	13.86	2.67e-15	4.48
	`shift_phimv(0)`	2.47e-14	4.78	2.78e-12	3.74	8.41e-12	27.34	1.27e-13	10.02
5	`shift_phimv`	2.06e-14	3.59	8.25e-13	3.50	9.14e-15	13.95	1.21e-13	4.57
	`shift_phimv(0)`	5.39e-14	3.96	3.36e-12	3.65	8.33e-12	27.02	1.59e-13	10.00
6	`shift_phimv`	5.49e-14	2.67	9.76e-13	3.45	5.88e-14	13.66	1.55e-13	4.58
	`shift_phimv(0)`	1.01e-13	4.30	3.16e-12	4.01	8.29e-12	28.16	1.67e-13	10.02
7	`shift_phimv`	9.81e-14	2.84	1.18e-12	3.65	6.78e-15	14.08	1.71e-13	4.57
	`shift_phimv(0)`	6.53e-14	4.46	2.84e-12	3.76	8.25e-12	28.08	2.90e-14	9.92
8	`shift_phimv`	6.21e-14	2.89	1.02e-12	3.33	2.51e-14	14.05	3.38e-14	4.55

Experiment 2. *In this experiment, we compare our algorithm,* `shift_phimv`*, with three other algorithms:* `phimp`*,* `expmv`*, and* `phimv(s)`*, for the computation of $\varphi_l(tA)b$ for $l = 1,2,\ldots,8$, where $b = [1,1,\ldots,1]^T$. The following four matrices are considered in this experiment.*

• *The first matrix,* `poisson3Da`*, from the SuiteSparse Matrix Collection [23], is an unsymmetric sparse matrix of order $N = 13,514$ with $352,762$ nonzero elements. We set $t = 100$.*

• *The second matrix,* `Bruss2D`*, from the two-dimensional Brussels problem [24], is an unsymmetric sparse matrix of order $N = 20,000$ with $119,200$ nonzero elements. We set $t = 10$.*

• *The third matrix,* `fdm_2d`*, is generated by MATLAB routine* `fdm_2d_matrix`*(100,'10*x','100*y','0') from LYAPACK toolbox [25]. This is an unsymmetric sparse matrix of order $N = 10,000$ with $49,600$ nonzero elements. We set $t = 1$.*

• *The fourth matrix,* `helm2d03`*, is also from the SuiteSparse Matrix Collection. This is a symmetric sparse matrix of order $N = 392,257$ with $2,741,935$ nonzero elements. We set $t = 100$.*

The results are presented in Table 3, where the relative errors for `expmv` are computed using the solutions of `phimv(s)` as the reference, while the relative errors for the other three methods are computed using the solutions of `expmv` as the reference. All four methods demonstrate high

Table 3. The relative errors and the CPU time when computing $\varphi_l(tA)b$ for $l = 1, 2, \ldots, 8$ of Experiment 2.

l	method	poisson3Da		Bruss2D		fdm_2d		helm2d03	
		Error	Time	Error	Time	Error	Time	Error	Time
1	phipm	9.24e-15	0.69	5.53e-14	7.22	1.33e-14	29.69	5.35e-16	0.42
	expmv	2.86e-15	0.26	5.77e-14	8.82	1.23e-14	24.81	3.99e-16	1.79
	phimv(s)	5.51e-16	0.27	1.53e-13	6.99	1.21e-14	19.96	3.99e-16	0.84
	shift_phimv	**6.15e-16**	**0.24**	**5.63e-14**	**6.31**	**7.50e-15**	**17.45**	**3.87e-16**	**0.36**
2	phipm	2.87e-14	0.48	9.78e-14	5.9	1.28e-12	28.18	6.19e-16	0.47
	expmv	4.68e-15	0.26	9.95e-14	8.89	1.28e-12	23.27	4.33e-16	2.10
	phimv(s)	1.89e-15	0.29	3.15e-14	7.23	1.30e-12	20.16	4.33e-16	0.94
	shift_phimv	**2.10e-15**	**0.20**	**1.00e-13**	**6.23**	**1.30e-12**	**18.38**	**6.26e-16**	**0.39**
3	phipm	8.26e-15	0.52	5.30e-14	5.09	1.47e-12	22.02	6.15e-16	0.49
	expmv	3.26e-15	0.28	5.28e-14	9.11	1.47e-12	22.49	4.20e-16	2.00
	phimv(s)	9.79e-16	0.29	1.08e-13	7.12	1.45e-12	19.86	4.20e-16	1.00
	shift_phimv	**1.41e-15**	**0.21**	**5.22e-14**	**6.16**	**1.45e-12**	**17.18**	**2.11e-15**	**0.39**
4	phipm	1.20e-14	0.71	3.18e-13	4.44	2.99e-13	19.98	1.22e-15	0.43
	expmv	1.39e-14	0.42	3.19e-13	9.25	2.98e-13	22.44	1.12e-15	2.27
	phimv(s)	1.00e-14	0.42	3.64e-13	7.68	3.21e-13	19.69	1.12e-15	0.97
	shift_phimv	**1.21e-14**	**0.26**	**3.17e-13**	**6.32**	**3.16e-13**	**17.94**	**7.92e-15**	**0.41**
5	phipm	2.09e-14	0.40	9.74e-13	3.88	3.57e-12	17.66	5.79e-16	0.49
	expmv	3.36e-14	0.38	9.75e-13	8.85	3.57e-12	23.63	5.90e-16	2.07
	phimv(s)	2.97e-14	0.42	1.02e-12	8.01	3.55e-12	20.82	5.90e-16	1.07
	shift_phimv	**2.50e-14**	**0.39**	**9.75e-13**	**5.98**	**3.55e-12**	**17.54**	**2.87e-14**	**0.46**
6	phipm	2.05e-14	0.32	5.80e-13	3.44	1.47e-12	15.26	7.64e-16	0.49
	expmv	9.09e-15	0.32	5.80e-13	9.00	1.47e-12	22.99	5.09e-16	2.33
	phimv(s)	1.25e-14	0.30	6.15e-13	7.40	1.49e-12	19.58	5.09e-16	1.29
	shift_phimv	**1.62e-15**	**0.24**	**5.79e-13**	**6.05**	**1.49e-12**	**17.54**	**9.66e-14**	**0.47**
7	phipm	2.01e-14	0.30	9.83e-13	3.16	5.82e-12	16.40	8.67e-16	0.47
	expmv	9.62e-15	0.31	9.84e-13	8.62	6.62e-13	22.57	5.78e-16	2.27
	phimv(s)	1.15e-14	0.31	1.02e-12	7.25	6.40e-13	19.41	5.78e-16	1.12
	shift_phimv	**3.20e-14**	**0.25**	**9.85e-13**	**6.05**	**6.44e-13**	**17.89**	**3.33e-13**	**0.47**
8	phipm	3.05e-15	0.31	2.46e-00	12.81	1.80e-11	21.45	3.54e-15	0.40
	expmv	1.05e-14	0.34	6.36e-13	8.90	1.33e-12	23.13	3.14e-15	2.09
	phimv(s)	8.44e-15	0.33	6.69e-13	7.88	1.31e-12	19.68	3.14e-15	0.93
	shift_phimv	**2.78e-14**	**0.25**	**6.36e-13**	**6.07**	**1.31e-12**	**17.81**	**1.19e-12**	**0.39**

accuracy, except for `phipm` in the case of the second test matrix `Bruss2D`, where it fails to obtain a convergent solution when solving for $l = 8$. For the remaining three test matrices, the results show that `shift_phimv` is the most efficient method.

5. Conclusions

This paper introduces an algorithm for computing the product of the φ-function and a vector, termed `shift_phimv`. The algorithm approximates the solution of $\varphi_l(tA)b$ using a scaling and recursing method that incorporates the shifting technique to further reduce the norm of the matrix and enhance computational efficiency. We employ forward error analysis and computational cost to determine the optimal Taylor approximation degree m and scaling parameter s. Numerical experiments comparing our algorithm with other popular methods demonstrate its efficiency and stability for large, sparse matrices.

Author contributions

Xianan Lu: Methodology, Investigation, Software, Writing-original draft; Dongping Li: Funding acquisition, Methodology, Investigation, Validation, Software, Writing-review & editing; Zhixin Zhao: Funding acquisition, Validation, Writing-review & editing. All authors have read and approved the final version of the manuscript for publication.

Acknowledgements

This work was partially supported by the Jilin Scientific and Technological Development Program (Grant Nos. YDZJ202401391ZYTS and YDZJ202501ZYTS635), the Natural Science Foundation of Jilin Province (Grant Nos. JJKH20240999KJ and JJKH20241001KJ), the Natural Science Foundation of Changchun Normal University (Grant No. 2021001), and the Graduate Innovation Project of Changchun Normal University (Grant No. YJSCX2024114).

Conflict of interest

The authors declare no conflict of interest.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

References

1 M. Hochbruck, A. Ostermann, Exponential Integrators, *Acta Numer.,* **19** (2010), 209–286. https://doi.org/10.1017/S0962492910000048

2 B. V. Minchev, W. M. Wright, A review of exponential integrators for first order semi-linear problems, Tech. report 2/05, Department of Mathematics, NTNU, 2005. `https://cds.cern.ch/record/848122`

3 A. H. Al-Mohy, N. J. Higham, A new scaling and squaring algorithm for the matrix exponential, *SIAM J. Matrix Anal. Appl.,* **31** (2009), 970–989. https://doi.org/10.1137/09074721X

4 E. Defez, J. Ibáñez, J. Sastre, J. Peinado, P. Alonso, A new efficient and accurate spline algorithm for the matrix exponential computation, *J. Comput. Appl. Math.,* **337** (2018), 354–365. https://doi.org/10.1016/j.cam.2017.11.029

5 N. J. Higham, The scaling and squaring method for the matrix exponential revisited, *SIAM J. Matrix Anal. Appl.,* **26** (2005), 1179–1193. https://doi.org/10.1137/090768539

6 Y. Y. Lu, Computing a matrix function for exponential integrators, *J. Comput. Appl. Math.,* **161** (2003), 203–216. https://doi.org/10.1016/j.cam.2003.08.006

7 C. Moler, C. V. Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, *SIAM Review,* **45** (2003), 3–49. https://doi.org/10.1137/S00361445024180

8 I. Najfeld, T. F. Havel, Derivatives of the matrix exponential and their computation, *Adv. Appl. Math.,* **16** (1995), 321–375. https://doi.org/10.1006/aama.1995.1017

9 J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, Efficient orthogonal matrix polynomial based method for computing matrix exponential, *Appl. Math. Comput.,* **217** (2011), 6451–6463. https://doi.org/10.1016/j.amc.2011.01.004

10 J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, New scaling-squaring Taylor algorithms for computing the matrix exponential, *SIAM J. Sci. Comput.,* **37** (2015), 439–455. https://doi.org/10.1137/090763202

11 B. Skaflestad, W. M. Wright, The scaling and modified squaring method for matrix functions related to the exponential, *Appl. Numer. Math.,* **59** (2009), 783–799. https://doi.org/10.1016/j.apnum.2008.03.035

12 R. C. Ward, Numerical computation of the matrix exponential with accuracy estimate, *SIAM J. Numer. Anal.,* **14** (1977), 600–610. https://doi.org/10.1137/0714039

13 A. H. Al-Mohy, N. J. Higham, Computing the action of the matrix exponential, with an application to exponential integrators, *SIAM J. Sci. Comput.,* **33** (2011), 488–511.https://doi.org/10.1137/100788860

14 Y. Saad, Analysis of some Krylov subspace approximations to the matrix exponential operator, *SIAM J. Numer. Anal.,* **29** (1992), 209–228. https://doi.org/10.1137/0729014

15 M. Caliari, P. Kandolf, A. Ostermann and S. Rainer, Comparison of software for computing the action of the matrix exponential, *BIT Numer. Math.,* **54** (2014), 113–128. http://doi.org/10.1007/s10543-013-0446-0

16 M. Caliari, P. Kandolf, A. Ostermann, S. Rainer, The leja method revisited: backward error analysis for the matrix exponential, *SIAM J. Sci. Comput.,* **38** (2016), A1639–A1661. https://doi.org/10.1137/15M1027620

17 R. B. Sidje, Expokit: A software package for computing matrix exponentials, *ACM Trans. Math. Softw.,* **24** (1998), 130–156. https://doi.org/10.1145/285861.285868

18 D. P. Li, Y. H. Cong, Approximation of the linear combination of φ-functions using the block shift-and-invert Krylov subspace method, *J. Appl. Anal. Comput.,* **4** (2017), 1402–1416. https://doi.org/10.11948/2017085

19 S. Gaudrealt, G. Rainwater, M. Tokman, KIOPS: A fast adaptive Krylov subspace solver for exponential integrators, *J. Comput. Phys.,* **372** (2018), 236–255. https://doi.org/10.1016/j.jcp.2018.06.026

20 J. Niesen, W. Wright, Algorithm 919: A Krylov subspace algorithm for evaluating the phi-functions appearing in exponential integrators, *ACM Trans. Math. Software,* **38** (2012), Article 22. https://doi.org/10.1145/2168773.2168781

21 D. P. Li, S. Y. Yang, J. M. Lan, Efficient and accurate computation for the φ-functions arising from exponential integrators, *Calcolo,* **59** (2022), 1–24. http://doi.org/10.1007/s10092-021-00453-2

22 N. J. Higham, F. Tisseur, A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra, *SIAM J. Matrix Anal. Appl.,* **21** (2000), 1185–1201. https://doi.org/10.1137/S0895479899356080

23 T. A. Davis, Y. F. Hu, The university of florida sparse matrix collection, *ACM Trans. Math. Software,* **38** (2011), Article 1. https://doi.org/10.1145/2049662.2049663

24 M. Hochbruck, C. Lubich, H. Selhofer, Exponential integrators for large systems of differential equations, *SIAM J. Sci. Comput.,* **19** (1998), 1552–1574. https://doi.org/10.1137/S1064827595295337

25 T. Penzl, LYAPACK: A MATLAB toolbox for large Lyapunov and Riccati equations, model reduction problems, and linear-quadratic optimal control problems, users' guide (Version 1.0), 1999. `https://netlib.org/lyapack/guide.pdf`